Département
Éducation
et Technologie

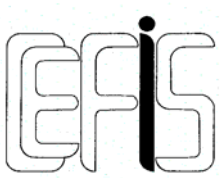
- Les formulaires en HTML
- JavaScript et CSS
- PHP
- MySQL
- Démarche de conception

Développer une application avec PHP et MySQL

Étienne Vandeput

5.87

Juin 2005



Centre pour la Formation à l'Informatique dans le Secondaire

Toutes vos remarques pour améliorer ces notes sont les bienvenues. Vous pouvez les envoyer à : etienne.vandeput@fundp.ac.be

Table des matières

1. Les formulaires en HTML

1.1 Introduction

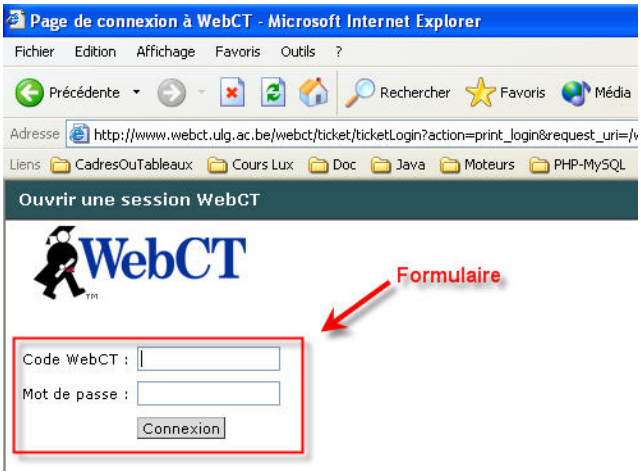
Depuis ses premiers développements, HTML inclut des éléments[1] qui permettent la communication entre un client Web et un serveur capable de recevoir des informations de ce client, de les stocker et/ou d'effectuer des traitements en dépendant. La réalisation de sites Web interactifs et la création de pages Web dynamiques demandent que cette communication soit possible. Dès lors, la connaissance de ces éléments est déterminante car ils constituent, en quelque sorte, l'interface de communication entre l'internaute et les applications qui sont développées sur des serveurs.

Dans ce chapitre, nous nous intéresserons à chacun de ces éléments, à leurs attributs possibles, mais aussi aux informations qui vont être véhiculées vers un serveur ainsi qu'à leur forme.

Les applications côté serveur sont développées dans un langage qualifié de langage de script. *PHP* est le langage de script que nous avons choisi[2] pour illustrer le mécanisme de construction dynamique des pages Web. Le langage est capable de communiquer et d'exploiter des données gérées par de nombreux SGBD[3]. Nous nous intéresserons à des applications exploitant des bases de données gérées par le SGBD *MySQL* parce que la combinaison de ces deux outils (langage de script et SGBD) semble aujourd'hui la plus fréquente parmi les nombreuses combinaisons possibles. C'est aussi celle qui apparaît comme la mieux adaptée aux applications exploitant Internet et ses technologies. Les bases de données gérées via des sites Web sont généralement particulières et comprennent peu d'enregistrements par rapports à certains autres types d'application. *MySQL* est adapté à ce type de bases de données en ce sens qu'il offre juste les services qu'il faut, avec l'efficacité souhaitée.

Une observation importante : le fait de confier l'exécution de scripts à des serveurs n'exclut pas la possibilité, pour le client, d'exécuter des scripts localement. C'est le cas, par exemple, lorsqu'on souhaite faire valider les données avant de les envoyer. Les traitements seront donc parfois répartis entre le client et le serveur même si l'objet de cet ouvrage est de s'intéresser aux scripts côté serveur.

1.2 Le formulaire (élément «form»)



Le formulaire est un élément qui rend possible la fourniture d'un certain nombre de données et l'exécution d'une action par un serveur. Un simple exemple, pour avoir accès à certains services sur le Web (commande en ligne, cours en ligne, homebanking,…) vous devez généralement fournir un identificateur (login) et un mot de passe (password). La fourniture de ces informations se fait au travers d'un formulaire (zones à compléter).

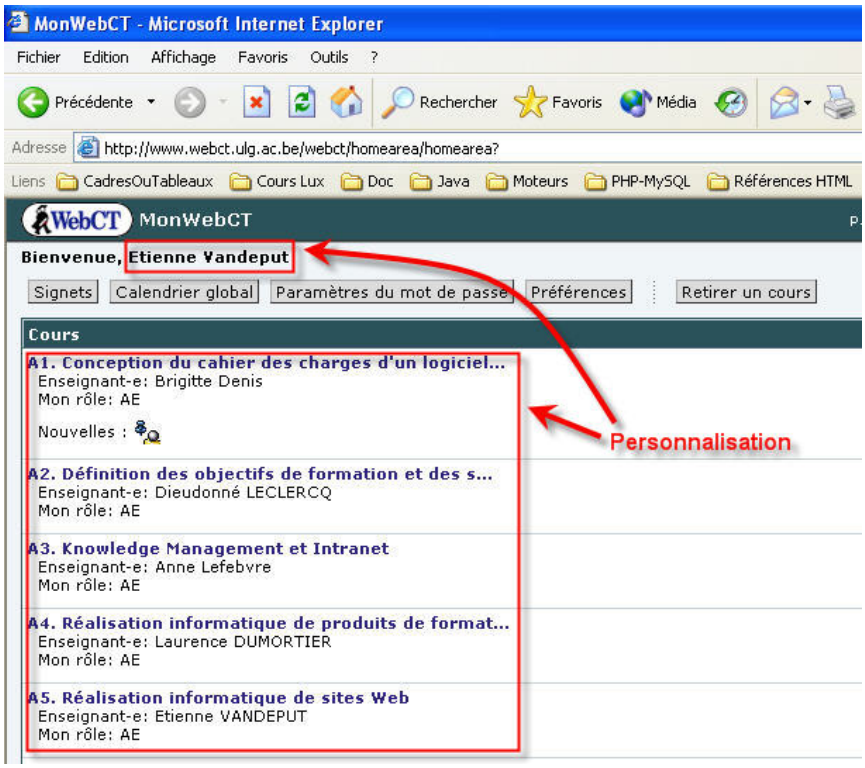
En fonction des données reçues, l'application construit dynamiquement une page Web dépendant des données reçues. Dans ce cas, il s'agit d'une page personnalisée (voir ci-après).

---

1 *p*, *h1*, *a*, *form* sont des éléments au sens de HTML.

2 En ce qui concerne les langages de script, le développeur a le choix. Ce sont cependant les langages utilisables gratuitement et pouvant être distribués qui connaissent le plus grand succès. Parmi eux, PHP, Python et dans une moindre mesure Perl semblent les mieux placés dans le hit parade des développeurs.

3 Système de Gestion de Bases de Données



En HTML, le formulaire correspond à l'élément **form**. Cet élément contient généralement plusieurs éléments interactifs que nous allons décrire par la suite (champs de texte, boutons, cases à cocher,…). Ces éléments interactifs permettent d'encoder ou de sélectionner les données à transmettre au serveur. Comme nous le verrons, il peut également contenir des éléments cachés qui sont aussi des données à transmettre au serveur.

Un document HTML peut contenir **plusieurs formulaires**[4] mais ceux-ci **ne peuvent** pas **être imbriqués**.

1.2.1 Attributs importants

1.2.1.1 action

Cet attribut est obligatoire. Sa valeur est une URL[5] (adresse absolue ou relative) qui précise le script qui doit recevoir les données et être exécuté sur le serveur. En général, le script a pour but de créer une page HTML qui sera renvoyée au client Web.

1.2.1.2 method

Il n'y a que deux valeurs possibles: *get* ou *post*. La méthode *get* concatène les données à la fin de l'URL du script alors que la méthode *post* envoie les données dans le corps de requête.

La méthode *post* est préférable pour tout une série de raisons et notamment, la confidentialité.

1.2.1.3 name

Cet attribut permet de nommer le formulaire. Il n'est d'aucune utilité pour le serveur mais peut servir à un script local.

---

4 Encore faut-il trouver une illustration intéressante de cette possibilité…

5 Beaucoup de serveurs sont sensibles à la casse des noms de fichiers et de dossiers. Il convient donc d'être prudent dans l'écriture de l'URL. Nous verrons également qu'une URL vide a du sens dans certaines circonstances.

Dans l'illustration qui précède, on voit le code engendré par *Macromedia Dreamweaver MX* lors de l'insertion d'un formulaire dans un document HTML. Un nom par défaut est donné et l'action par défaut est vide. La méthode choisie est *post*.

1.2.2 Contenu du formulaire

Tous ces attributs n'ont aucun sens si le formulaire n'a pas de contenu. L'élément **form** est donc un élément qui va en contenir d'autres porteurs d'informations ou d'instructions.

On trouve trois types d'éléments dans un formulaire:

- **textarea**: pour définir une zone de texte
- **select**: pour sélectionner une information dans une liste
- **input**: pour tous les autres types d'entrées

Ce sont ces éléments que nous allons décrire maintenant.

1.3 L'élément «textarea»

On emploie cet élément lorsqu'il y a plusieurs lignes de texte à entrer. Ainsi, lorsqu'on souhaite réaliser une application de Webmail, il est clair que la zone de rédaction du message doit s'étendre sur plusieurs lignes. Les valeurs par défaut sont 4 lignes de 40 caractères. Les attributs qui suivent permettent néanmoins de modifier ces valeurs.

Cet élément nécessite une balise initiale et une balise finale. Un texte par défaut peut se trouver entre ces deux balises. Il est à noter que le texte n'est pas limité dans sa taille, ce qui peut se révéler assez gênant dans une perspective de stockage dans une base de données, par exemple. Un contrôle peut toutefois être effectué, soit par un script local, soit par un script PHP au niveau du serveur.

1.3.1 Les attributs les plus importants

1.3.1.1 name

La zone de texte doit porter un nom si on veut l'exploiter en PHP. Ce nom correspondra au nom d'une variable en PHP. C'est le moment de se souvenir que le langage est sensible à la casse.

Si certains objets interactifs portent le même nom, PHP ne prend en compte que le dernier. Les noms doivent commencer par une lettre ou le caractère _ suivi de lettres, chiffres ou caractères _.

1.3.1.2 rows

Cet attribut précise la hauteur de la zone de texte, ce qui n'en limite pas la taille.

1.3.1.3 cols

Cet attribut spécifie le nombre de colonnes visibles mais ne correspond que fort approximativement au nombre de caractères.

Voici un exemple de code et de son interprétation dans un navigateur.

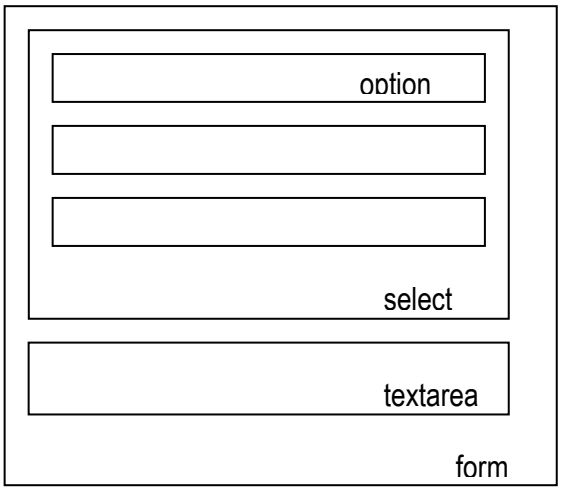
```
<form name="form1" method="post" action="">
  <p><font size="2" face="Verdana, Arial, Helvetica, sans-serif"><strong>Objectifs
  du cours</strong></font></p>
  <textarea name="textarea" cols="60" rows="10"></textarea>
</form>
```

Objectifs du cours

Le cours a pour objectif principal de...

1.4 L'élément «select»

Cet élément défini un menu déroulant dans lequel l'utilisateur va pouvoir sélectionner une ou plusieurs informations. Ces informations seront définies grâce à des éléments **option** qui seront inclus entre les deux balises de l'élément **select**.



Notez que chacun des éléments décrits jusqu'ici peut être vu comme un bloc contenant éventuellement d'autres blocs. L'élément **form** contient des éléments **textarea**, **select**,… L'élément (bloc) **select** contient des éléments **option**.

Un formulaire peut évidemment contenir plusieurs éléments **select**. PHP crée une variable ou un tableau (en cas de sélection multiple) pour chaque élément **select**. La variable ou le tableau porte le même nom que l'élément.

1.4.1 Les attributs les plus importants

1.4.1.1 name

Pour les raisons qui précèdent, cet attribut est évidemment indispensable. Attention, en cas de sélection multiple possible, ce nom doit se terminer par des crochets ([]).

1.4.1.2 multiple

Cet attribut autorise les sélections multiples dans la liste.

1.4.1.3 size

Cet attribut spécifie le nombre d'items visibles de la liste. En son absence, un seul item est visible.

1.4.2 Les attributs les plus importants de l'élément «option»

Cet élément n'est pas directement un objet de formulaire mais une composante de l'élément **select**. Il nécessite une balise initiale et une balise finale.

1.4.2.1 value

L'attribut précise la valeur qui est envoyée au script et donc pas nécessairement celle qui est affichée dans la liste. Il est en effet plus simple, parfois, d'envoyer des valeurs techniques plus simples à manipuler tout en affichant des choix compréhensibles.

1.4.2.2 selected

Les options qui ont cet attribut correspondent à des valeurs présélectionnées. Il n'est pas nécessaire qu'il y en ait. Dans le cas d'une sélection simple, s'il y en a plusieurs, seule la dernière est prise en compte. Lorsque la liste est déroulante, c'est la première option ou, le cas échéant, l'option présélectionnée qui apparaît. Voici deux exemples incluant les types d'éléments concernés.

```
<p><font size="2" face="Verdana, Arial, Helvetica, sans-serif"><strong>Type
  de cours</strong></font></p>
  <select name="typeCours" size="4">
    <option value="0" selected>inconnu</option>
    <option value="1">pr&eacute;senciel</option>
    <option value="2">&agrave; distance</option>
    <option value="3">semi-pr&eacute;senciel</option>
  </select>
```

Type de cours


```
<p><font size="2" face="Verdana, Arial, Helvetica, sans-serif"><strong>Type
  de cours</strong></font></p>
<select name="typeCours">
  <option value="0" selected>inconnu</option>
    <option value="1">pr&eacute;senciel</option>
    <option value="2">&agrave; distance</option>
    <option value="3">semi-pr&eacute;senciel</option>
  </select>
```

Type de cours

1.5 L'élément «input»

Cet élément correspond à une panoplie d'objets de formulaire. C'est l'attribut ***type*** de cet élément **input** qui les distingue. Nous allons donc examiner ces différents objets, les uns après les autres. Un formulaire peut évidement contenir plusieurs éléments **input**. L'élément **input** n'a qu'une seule balise.

1.5.1 Le champ texte <input type="text">

Cet attribut définit un champ texte d'une seule ligne. PHP crée une variable portant le nom du champ dont la valeur est l'information que ce dernier contient. Les attributs courants sont:

1.5.1.1 name

voir ci-dessus

1.5.1.2 value

la valeur par défaut du champ texte

1.5.1.3 size

la taille (approximative) d'affichage du champ texte

1.5.1.4 maxlength

le nombre de caractères admis

1.5.1.5 accesskey

crée un raccourci-clavier pour accéder directement au champ

Voici un petit exemple.

```
  <p><font size="2" face="Verdana, Arial, Helvetica, sans-serif">
<strong><u>D</u>&eacute;nomination</strong></font></p>
  <p>
    <input name="denomination" type="text" size="50" maxlength="80" accesskey="D">
  </p>
```

Le raccourci-clavier alt-D premettra d'accéder directement au champ texte. L'attribut ***accesskey*** existe aussi pour les éléments **textarea** et **select**.

1.5.2 Le champ texte spécial mot de passe <input type="password">

L'attribut définit un champ texte particulier car l'information introduite dans ce champ n'est pas lisible à l'écran. Il faut toutefois savoir que le mot de passe est transmis en clair au serveur. Il est possible de demander à PHP un encryptage par la suite.

Comme pour un champ texte ordinaire, PHP crée une variable portant le nom du champ. Les attributs courants sont les mêmes que ceux du champ texte normal.

1.5.3 Le champ texte spécial référence de fichier <input type="file">

Dans ce champ texte particulier, l'information est la référence absolue et locale d'un fichier que l'on veut transférer au serveur. L'usage de cette fonctionnalité nécessite que la méthode d'envoi du formulaire soit *post* et que l'attribut ***enctype***, dont nous n'avons pas parlé, soit présent avec comme valeur *multipart/form-data*. Le champ est accompagné d'un bouton «Parcourir» qui ouvre une fenêtre de l'explorateur.

```
  <p><font size="2" face="Verdana, Arial, Helvetica, sans-serif">
<strong>T&eacute;l&eacute;verser</strong></font></p>
  <p>
    <input type="file" name="telechargement">
  </p>
```

Les attributs courants sont sensiblement les mêmes, si ce n'est que l'attribut ***value*** n'est pas utilisé.

1.5.4 Le bouton simple <input type="button">

L'attribut définit un bouton simple. L'intérêt de ce type de bouton réside dans l'exécution de scripts locaux. Aucune donnée n'est envoyée au serveur lorsqu'un tel bouton est activé. On mentionnera les attributs:

1.5.4.1 name

le nom du bouton

1.5.4.2 value

son étiquette

1.5.4.3 disabled

son état (désactivé)

1.5.4.4 accesskey

crée un raccourci-clavier pour accéder directement au bouton

```
    <input name="validation" type="button" id="validation" value="Validation des
donn&eacute;es" disabled>
```

1.5.5 Le bouton radio <input type="radio">

Les boutons radio sont utilisés en groupe. Un seul d'entre eux peut être actif à la fois. Le groupe de boutons porte un nom qui sera celui de la variable PHP contenant la valeur du bouton sélectionné. Citons comme attributs:

1.5.5.1 name

le nom du **groupe** de boutons

1.5.5.2 value

la valeur du bouton (information transmise)

1.5.5.3 checked

la sélection du bouton

1.5.5.4 accesskey

le raccourci-clavier

Un exemple:

```
  <p><font size="2" face="Verdana, Arial, Helvetica, sans-serif"><strong>Type
    d'enseignement</strong></font></p>
  <p>
    <input type="radio" name="type" value="1">
    <font size="2" face="Verdana, Arial, Helvetica, sans-serif">Général</font>
    <br>
    <input type="radio" name="type" value="2">
    <font size="2" face="Verdana, Arial, Helvetica, sans-serif">Technique</font>
    <br>
    <input type="radio" name="type" value="3">
    <font size="2" face="Verdana, Arial, Helvetica, sans-serif">Professionnel</font>
</label>
  </p>
```

Type d'enseignement

- ○ Général
- ◉ Technique
- ○ Professionnel

1.5.6 Le bouton d'envoi <input type="submit">

Grâce à cet attribut, le bouton est le déclencheur de l'envoi des données au script décrit dans l'attribut ***action*** de l'élément **form**. Un formulaire peut contenir plusieurs boutons d'envoi. Dans ce cas, c'est au script de choisir le traitement à effectuer en fonction de la valeur du bouton. Comme d'habitude, PHP créera une variable portant le nom de ce bouton. Les attributs importants sont:

1.5.6.1 name

le nom du bouton nécessaire pour la création de la variable

1.5.6.2 value

la valeur du bouton nécessaire pour un éventuel choix de script

1.5.6.3 accesskey

un caractère pour un raccourci-clavier d'accès rapide au bouton

```
    <input name="envoi" type="submit" id="envoi" value="Envoyer">
```

Envoyer

1.5.7 Le bouton d'envoi image <input type="image">

Le bouton d'envoi image joue le même rôle que le bouton d'envoi classique. La différence réside dans le fait que la source de l'image doit être fournie. De plus, les coordonnées précises de l'endroit où l'on a cliqué sont également envoyées. Les noms des deux variables sont constitués du nom du bouton suivi respectivement des symboles *_x* et *_y*. Les attributs à retenir sont:

1.5.7.1 name

le nom du bouton image

1.5.7.2 src

la localisation de l'image

1.5.7.3 alt

le texte alternatif (si l'image ne peut être affichée)

1.5.7.4 accesskey

le caractère pour un raccourci-clavier éventuel d'accès au bouton

```
    <input name="cathedrale" type="image" id="cathedrale"
src="/images/cathedrale.jpg" alt="Saint-Aubain" width="100" height="80" border="0">
```

1.5.8 Le bouton de réinitialisation <input type="reset">

Le bouton de réinitialisation permet à chaque objet du formulaire de reprendre sa valeur initiale. Il n'y a donc aucune information envoyée au serveur lorsque celui-ci est activé. Les attributs essentiels sont:

1.5.8.1 name

le nom du bouton

1.5.8.2 value

son étiquette

1.5.8.3 accesskey

le caractère pour un raccourci-clavier

```
    <input name="initialisation" type="reset" id="initialisation"
value="R&eacute;tablir">
```

Rétablir

1.5.9 La case à cocher <input type="checkbox">

L'attribut fait de cet objet une case à cocher. Il est possible mais pas nécessaire de grouper les cases à cocher. PHP crée une variable pour chaque case à cocher ou un tableau pour un groupe de cases à cocher. La case ou au moins une des cases dans le cas d'un groupe doit être cochée pour que la variable ou le tableau soit créé. La programmation devra veiller à s'assurer qu'une telle variable ou un tel tableau existe. Les attributs importants sont:

1.5.9.1 name

le nom de la case sera celui de la variable; dans le cas d'un groupement, il faut donner le même nom à chaque case et faire suivre ce nom de crochets ([]).

1.5.9.2 value

la valeur transmise au script qui ne correspond pas à l'affichage

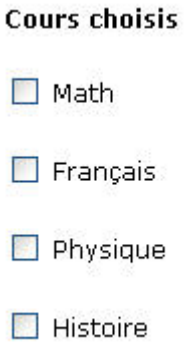
1.5.9.3 checked

pour signaler si la case et cochée

1.5.9.4 accesskey

un caractère pour le raccourci-clavier

```
<p><font size="2" face="Verdana, Arial, Helvetica, sans-serif"><strong>Cours
  choisis</strong></font></p>
<p> <strong><font size="2" face="Verdana, Arial, Helvetica, sans-serif">
  <input type="checkbox" name="cours" value="1">
  </font></strong><font size="2" face="Verdana, Arial, Helvetica, sans-serif">
  Math </font></p>
<p> <font size="2" face="Verdana, Arial, Helvetica, sans-serif">
  <input type="checkbox" name="cours" value="2">
  Fran&ccedil;ais</font></p>
<p> <font size="2" face="Verdana, Arial, Helvetica, sans-serif">
  <input type="checkbox" name="cours" value="3">
  Physique</font></p>
<p> <font size="2" face="Verdana, Arial, Helvetica, sans-serif">
  <input type="checkbox" name="cours" value="4">
  Histoire</font></p>
```



1.5.10 Le champ caché <input type="hidden">

L'attribut confère ici au champ le statut «caché». L'intérêt réside dans la possibilité de transmettre au serveur des valeurs qui seraient disponibles au niveau du navigateur. C'est aussi de pallier le caractère « sans état » de http. Une variable portant le nom du champ est créée. Les attributs sont:

1.5.10.1 name

le nom du champ caché

1.5.10.2 value

la valeur à transmettre

```
    <input name="login" type="hidden" id="login" value="eva">
```

1.6 Attributs vs méthodes

Sans entrer dans le grand détail, nous rappellerons que *JavaScript* est un langage de script interprété par le client (le navigateur Web) et que ce langage est un langage « orienté objet »[6]. Il s'appuie sur le *Document Object Model* (DOM) développé par le consortium *w3*[7]. Parmi ces « objets », on trouve bien sûr les formulaires et tous les objets qu'ils peuvent eux-mêmes contenir. *JavaScript* définit des **méthodes** qu'il est possible d'invoquer sur ces objets. Nous nous intéresserons peu à ces méthodes étant donné que la programmation en *JavaScript* n'est pas le sujet principal de cet ouvrage. En revanche, nous avons besoin des **propriétés** et des **événements** détectables au niveau des objets car ils se traduisent en attributs des éléments correspondants. À titre d'exemple, voici pour un formulaire quelques attributs et méthodes disponibles.

Quelques attributs possibles de l'**élément** formulaire (du point de vue *HTML*)

- Propriétés :

  name : le nom du formulaire

  action : l'adresse du script de serveur à exécuter

  method : la méthode d'appel du script (*get* ou *post*)

- Événements :

  onSubmit : pour détecter la soumission du formulaire

  onReset : pour détecter la réinitialisation

Il en existe bien d'autres qui sont moins spécifiques (***id***, ***class***,… pour les propriétés ; ***onclick***, ***ondblclick***,… pour les événements).

Quelques méthodes possibles de l'**objet** formulaire[8] (du point de vue *JavaScript*) :

submit : pour déclencher l'action du formulaire

reset : pour réinitialiser les données du formulaire avec les valeurs par défaut

---

6 La signification de ce qualificatif, en particulier en *JavaScript*, mériterait un large débat que nous ne mèneront pas.

7 http://w3.org

8 On pourrait trouver dans un script une instruction du genre *if (valide) form.submit() ;* pour autant que *valide* désigne une variable booléenne et *form* un formulaire.

Après avoir illustré les propriétés des éléments correspondant aux objets de formulaire, nous consacrons la section suivante à examiner les **événements** qui peuvent également y être associés sous forme d'attributs.

1.7 La gestion des événements

Les attributs correspondant à des événements peuvent se révéler utiles dans le contexte d'utilisation de scripts locaux. En voici une description en fonction des éléments concernés.

1.7.1 Élément « form » et transmission des données

1.7.1.1 onsubmit

Cet attribut indique un script à exécuter localement avant l'envoi des données (par exemple, un certain nombre de vérifications). Ce script doit renvoyer une valeur booléenne qui détermine l'envoi ou non des données. Il est déclenché lorsque l'utilisateur clique sur le bouton d'envoi (voir plus loin).

La valeur doit être constituée du mot *return* suivi du nom du script à exécuter. Exemple: ***onsubmit*** = *"return verification()"*

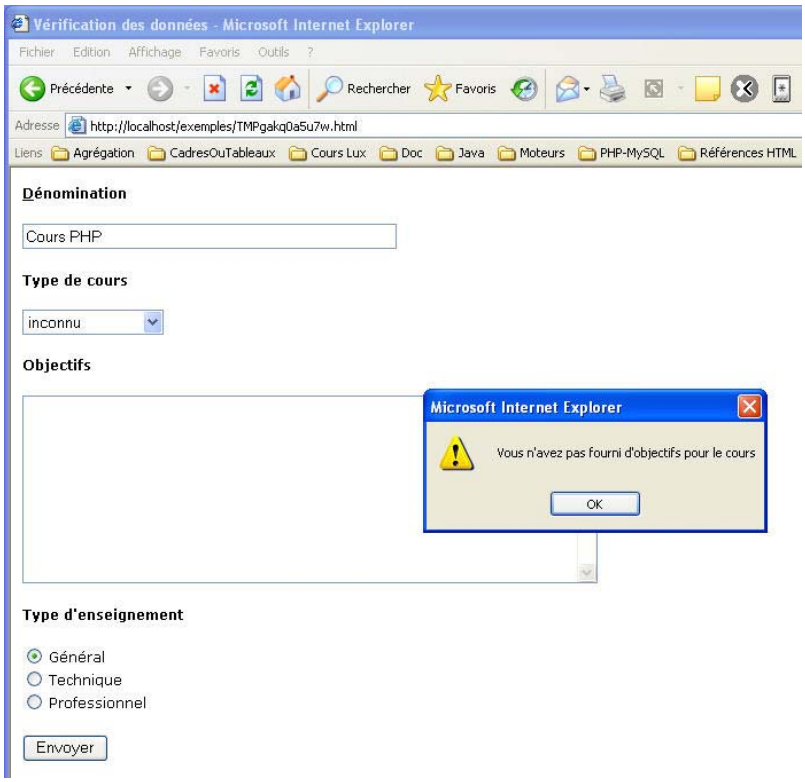
```
<form action="test.php" method="post" name="formulaire" onSubmit="return
verification()">
```

Le script doit figurer dans le document, soit dans la partie entête, soit dans la partie corps, en tant que contenu de l'élément **script**. Cet élément possède un attribut ***language*** qui précise dans quel langage le script est rédigé (autrement dit, comment il doit être interprété) et un attribut ***type*** qui précise quel encodage est utilisé.

```
<script language="JavaScript" type="text/JavaScript">
function verification(){
 if(document.forms["formulaire"].elements["denomination"].value==""){
  alert("Vous n'avez pas fourni de dénomination pour le cours");
  return false;
 }
 if(document.forms["formulaire"].elements["objectifs"].value==""){
  alert("Vous n'avez pas fourni d'objectifs pour le cours");
  return false;
 } else{
   return true;
 }
}
</script>
```

Comme nous ne donnons pas trop de détails à propos de *JavaScript*, nous apportons à ce script quelques commentaires.

Il se compose d'un double test de vérification de remplissage pour les rubriques *Dénomination* et *Objectifs*. La fonction n'a pas d'argument et renvoie une valeur booléenne (`return=…`). *alert()* est une fonction prédéfinie qui affiche une boîte de dialogue d'avertissement. Elle prend comme argument une chaîne de caractère. Notez que *JavaScript* est un langage faiblement typé. Le typage des valeurs et des variables est implicite. Observez aussi la syntaxe d'écriture pour une structure alternative.



Le point le plus délicat concerne l'accès aux informations de la page. *JavaScript* utilise le modèle objet du document (DOM) pour accéder à ces informations. L'objet principal est *document*. Il contient un tableau indexé *forms* de ses (éventuels) formulaires. Chaque formulaire contient un tableau indexé de ses éléments : *elements*. Pour accéder à la valeur d'un de ces éléments, on utilise la propriété *value*. Notez encore le symbole de comparaison (== et non =). Il est possible d'accéder à l'item d'un tableau en fournissant son numéro d'ordre en commençant par 0 (exemple : si *formulaire* est le seul formulaire, *forms[0]*).

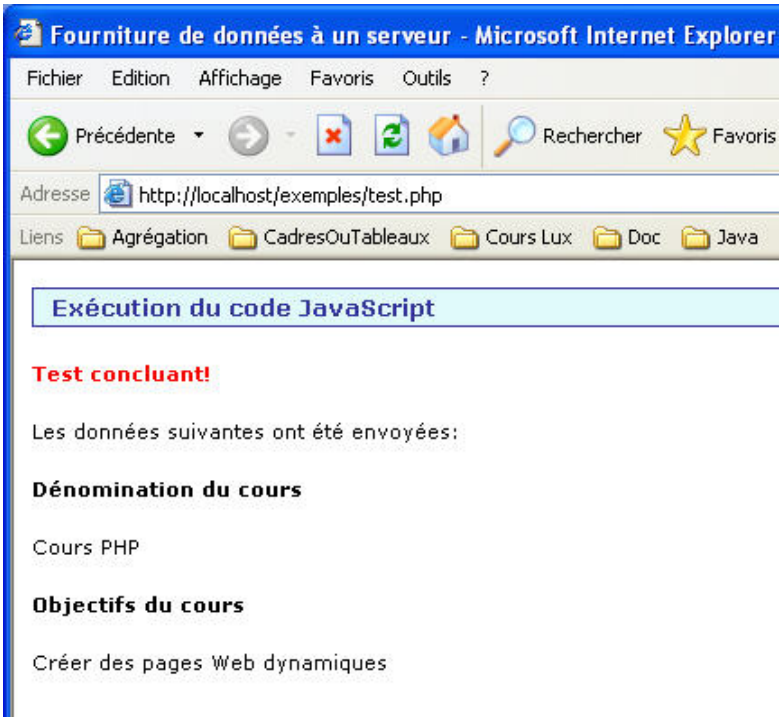
Voici maintenant le code du fichier *test.php*. Ce dernier contient l'action à effectuer si la valeur renvoyée par le script est *true* à savoir, l'affichage d'un message de réussite et des données transmises. Pour ce faire, des commandes élémentaires en *PHP* sont nécessaires. Une feuille de style, *cefis.css*, est attachée à la page *test.php*.

```
<html>
<head>
<title>Fourniture de données à un serveur</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="../cefis.css" rel="stylesheet" type="text/css">
</head>
<body>
<h4>Ex&eacute;cution du code JavaScript</h4>
<p class="important1">Test concluant!</p>
<p>Les donn&eacute;es suivantes ont &eacute;t&eacute; envoy&eacute;es:</p>
<p class="important2">D&eacute;nomination du cours</p>
<p><?php echo $_POST["denomination"]; ?></p>
<p class="important2">Objectifs du cours</p>
```

```
<p><?php echo $_POST["objectifs"]; ?></p>
</body>
</html>
```

Cet exemple nous permet de découvrir un tout petit peu *PHP*. Lorsque le navigateur rencontre la suite des symboles **<?php**, il sait qu'il doit interpréter la suite comme du *PHP* et cela, tant qu'il n'a pas rencontré les deux symboles **?>**. Comme vous vous en doutez[9], la commande **echo** est une commande d'affichage.

Une autre chose importante à observer, c'est que (à partir de sa version 4) *PHP* crée une variable prédéfinie[10] sous forme d'un tableau s'appelant **$_POST**[11] et contenant les valeurs transmises au serveur par la méthode *post*. On accède à une composante du tableau par son numéro d'ordre ou le nom (attribut ***name***) de l'élément contenant l'information concernée.



1.7.1.2 onreset

Cet attribut indique un script à exécuter localement avant la réinitialisation du formulaire. Pour la valeur de cet attribut, la syntaxe est identique à celle de l'attribut ***onsubmit***.

1.7.2 Élément « textarea »

Voici quelques-uns des événements (ou attributs de script) de l'élément **textarea**.

1.7.2.1 onfocus

Le script associé sera exécuté lorsque la zone de texte recevra le focus[12].

---

9 Rappelez-vous la création de fichiers *batch* en *DOS*.

10 Il en crée plein d'autres.

11 En *PHP*, toutes les variables ont un nom commençant par le symbole **$**.

12 Un objet reçoit le focus quand l'utilisateur clique dessus, ou bien qu'il y accède par la touche de tabulation ou un raccourci-clavier.

1.7.2.2 onblur

Il s'agit ici de la perte du focus par l'élément.

1.7.2.3 onselect

Cet événement se produit si tout ou partie du texte de la zone a été sélectionné.

1.7.2.4 onchange

Celui-ci se produit en cas de perte de focus et pour autant que la valeur de la zone de texte ait été modifiée.

1.7.3 Élément « select »

Parmi les événements associables à l'élément **select** on trouve ***onfocus***, ***onblur*** et ***onchange***.

1.7.4 Élément « input »

1.7.4.1 Champs texte, texte spécial mot de passe et texte spécial référence de fichier

Les événements intéressants à associer sont ***onfocus***, ***onblur***, ***onselect*** et ***onchange***.

1.7.4.2 Bouton simple, d'envoi,de réinitialisation, radio, case à cocher

Les événements à associer sont ***onfocus***, ***onblur*** et ***onchange***.

1.7.4.3 Bouton d'envoi image

Les événements à associer sont ***onfocus*** et ***onblur***.

1.8 Autres événements, autres éléments

Si vous utilisez un éditeur spécialisé, vous constaterez peut-être qu'il propose beaucoup d'autres événements et quelques objets de formulaire supplémentaires.

Parmi les événements, citons, par exemple : ***onclick***, ***ondblclick***, ***onmousedown***, ***onmouseup***, ***onmouseover***, ***onmouseout***, ***onmousemove***, ***onkeypress***, ***onkeydown***, ***onkeyup***…

Les éléments **button**, **fieldset**, **label** et **optgroup** existent également. Ils offrent de nouvelles possibilités mais ils ne sont pas correctement interprétés par tous les navigateurs.

Par exemple, l'élément **button** permet de combiner texte et image. L'élément **fieldset** permet de regrouper des champs sous un même titre, grâce à un élément **legend** par lequel l'élément **fieldset** doit commencer. L'élément **label** permet d'attribuer une étiquette à un objet, ce qui n'a de véritable intérêt que pour les personnes malvoyantes utilisant des navigateurs couplés avec un système sonore. L'élément **optgroup** sert à regrouper des éléments options à l'intérieur d'un élément **select** ce qui permet la création de menus en cascade.

1.9 Exercices

Écrire des scripts JavaScript pour :

- rendre un champ texte inaccessible
- donner le focus à un élément précis
- permettre l'exécution d'une action en fonction du choix d'un bouton radio

1.9.1 Rendre un champ texte inaccessible

Il suffit de faire perdre le focus à l'objet au moment où il la reçoit.

```
<input type="text" name="couleur" onFocus="this.blur()">
```

Comme dans la plupart des langages objets, la référence à l'objet courant est *this*.

Une autre solution consiste à donner à l'objet l'attribut ***readonly***.

```
<input type="text" name="couleur" readonly>
```

Les effets graphiques sont légèrement différents dans la mesure où, dans le premier cas, le point d'insertion apparaît avant de disparaître.

1.9.2 Donner le focus à un élément précis

Cette action est réalisée au chargement de la page. C'est donc l'attribut ***onload*** de l'élément **body** qui est utilisé.

```
<body onLoad="document.form1.nom.focus()">
<form name="form1" method="post" action="">
  <p>Nom
    <input type="text" name="nom">
  </p>
</form>
</body>
```

1.9.3 Permettre l'exécution d'une action en fonction du choix d'un bouton radio

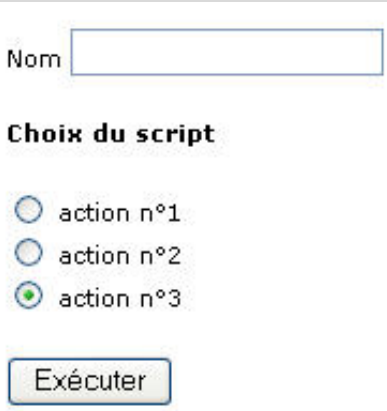
Le problème est plus délicat. Il s'agit d'identifier le bouton radio qui est sélectionné et de compléter l'action en conséquence.

La première fonction a pour but de renvoyer l'index correspondant à ce bouton dans le tableau des boutons radios concernés (index qui commence à 0). La seconde fonction fixe l'action qui est la valeur du bouton et provoque la soumission du formulaire.

```
<script language="JavaScript" type="text/JavaScript">
function valeurChoix(groupeBouton){
  for (var i=0;i<groupeBouton.length;i++){
    if (groupeBouton[i].checked) {return i}
  }
}
function choisirAction(formulaire,script){
  var i=valeurChoix(script)
  formulaire.action=script[i].value
  formulaire.submit()
}
</script>
```

```
<form name="form1" method="post" action="">
  <p>Nom
```

```
    <input type="text" name="nom">
  </p>
  <p><strong>Choix du script</strong></p>
  <p>
    <label>
    <input name="choix" type="radio" value="action1.php" checked>
    action n&deg;1</label>
    <br>
    <label>
    <input type="radio" name="choix" value="action2.php">
    action n&deg;2</label>
    <br>
    <label>
    <input type="radio" name="choix" value="action3.php">
    action n&deg;3</label>
  </p>
  <p>
    <input type="button" name="run" value="Ex&eacute;cuter" onClick=
"choisirAction(this.form,this.form.choix)">
    <br>
  </p>
</form>
```



Voici à quoi ressemble l'interface. Le choix d'un des boutons avant un clic sur le bouton simple permet de sélectionner le script dont l'exécution sera demandée au serveur.

1.10 Les styles CSS

Cette section, un peu hors contexte, trouve toutefois son utilité dans la mesure où elle présente un langage (un de plus) interprété par les navigateurs.

CSS signifie *Cascading Style Sheets* (feuilles de style en cascade). C'est un langage dont l'objectif est de paramétrer finement la mise en page et la mise en forme des éléments d'un document *HTML*. Il convient d'en comprendre les principes, et notamment, le fonctionnement de la cascade. Il est également utile de se familiariser avec la syntaxe du langage.

L'utilisation de *CSS* permet de réfléchir à des stratégies efficaces de conception en prenant en compte, dès le départ, les possibles mises à jour.

1.10.1 Feuille de styles

Le concept de **feuille de styles** est un concept abstrait qui trouve sa concrétisation sous de multiples formes. On peut définir une feuille de styles comme un **ensemble de règles** utilisées par le navigateur pour la mise en page et en forme des différents éléments (au sens de *HTML*). Ces règles sont définies:

- par défaut

- par l'internaute
- par l'auteur

Nous verrons plus loin quelles priorités le navigateur accorde à ces différents niveaux.

1.10.2 Règle

Une règle se compose d'un **sélecteur** (d'éléments) suivi d'une **déclaration** (des paramètres à appliquer). La syntaxe en est la suivante :

<*sélecteur*> **{**<*déclaration*>**}**

Examinons d'abord la syntaxe d'une déclaration. Elle se compose d'un ou plusieurs ensembles constitués d'un mot-clé et d'une valeur.

<*mot-clé*> **:** <*valeur*> **;** [<*mot-clé*> **:** <*valeur*> **;**]*

Il existe 52 mots-clés dans la version *CSS1* que sont censés respecter tous les navigateurs. Chaque mot-clé est séparé de sa valeur par un double point et l'ensemble se termine par un point-virgule. Toutes les valeurs ne sont pas acceptables.

1.10.2.1 Exemples

```
color : red ;
color : #FF0000 ;
font-family : Arial ;
color : red; font-size : 18px ;
```

1.10.3 Sélecteur

Les sélecteurs sont les paramètres qui autorisent le navigateur à sélectionner les règles de mise en forme applicables aux éléments *HTML*. On distingue les **sélecteurs simples** des **sélecteurs contextuels**.

1.10.3.1 Sélecteur simple

Un sélecteur simple permet de sélectionner les éléments de trois manières. Soit :

- qu'ils portent le même nom ;
- qu'ils ont les mêmes valeurs pour certains attributs précis (***class*** et ***id***) ;
- qu'ils portent le même nom et ont les mêmes valeurs pour certains attributs précis (***class*** ou ***id***)

1.10.3.2 Exemples

- Même nom : **p**

Tous les éléments **p** du document HTML tels `<p>`, `<p class = « rouge »>`, `<p id = « chapeau »>`,...

- Même valeur pour l'attribut ***class*** : **.important** (attention au point initial)

Tous éléments (quel que soit leur nom) qui ont un attribut ***class*** valant "*important*" tels : `<??? class="important" …>`, ...

- Même valeur pour l'attribut ***id*** : **#titre** (attention au dièse initial)

Tous éléments (quel que soit leur nom) qui ont un attribut ***id*** valant "*titre*" tels : `<??? id="titre" …>`,...

- Même nom et même valeur pour l'attribut ***class*** ou l'attribut ***id*** : **p.important** ou **td#titre**

Tous les éléments **p** (**td**) qui ont un attribut ***class*** (***id***) valant "*important*" ("*titre*")

`<p class="important" …>`, `<td id="titre" …>`,...

1.10.3.3 Sélecteur contextuel

Les sélecteurs contextuels permettent de sélectionner tous les éléments qui sont dans un certain contexte. La syntaxe est relativement simple puisqu'un sélecteur contextuel se compose de plusieurs sélecteurs simples séparés par des espaces.

Il est bon, pour en comprendre l'intérêt, de prendre en compte le principe d'emboîtement des éléments *HTML*. Les éléments *HTML* sont emboîtés. Cela signifie qu'ils peuvent contenir d'autres éléments (**p**, **h1**, **strong**, **ul**,…). Un sélecteur contextuel fait référence à un élément contenu dans un ou plusieurs autres.

La possibilité d'utiliser des sélecteurs contextuels donne une grande souplesse à la mise en forme. Ainsi, par exemple, les paragraphes peuvent-ils être formatés différemment s'ils sont dans un tableau, dans un calques ou hors de tout contexte.

1.10.3.4 Exemples

h4 strong désigne les éléments **strong** qui sont contenus dans un élément **h4**.

```
<h4>Titre en <strong>gras</strong></h4>
```

Autres exemples : **h4 strong em** ou encore **td .important**

Attention, **p.important** (sélecteur simple) est différent de **p .important** (sélecteur contextuel).

1.10.4 Exemples

Voici quelques exemples de règles. Notez que ces règles ne sont que du texte et qu'elles peuvent donc être éditées avec un simple éditeur de texte. Il reste à savoir à quel endroit on peut les définir.

p {color:red;}

les contenus des paragraphes en rouge

h1 {font-size:20px;font-family:Verdana;}

les titres de niveau 1 en taille 20 pixels et police Verdana

p.important {font-weight:bold;}

les paragraphes qualifiés "important" en gras

div p,h1,.important {color:red;}

les paragraphes dans les calques, les titres de niveau 1 et tous les éléments qualifiés "important" en rouge

td p.mineur {color=#333333;font-size:15px;}

les paragraphes qualifiés "mineur" dans les cellules de tableau en gris et taille 15 pixels

Il faut signaler que la plupart des éditeurs *HTML* donnent au concepteur la possibilité de créer les règles de manière dynamique sans avoir à en écrire le code.

1.10.5 La cascade

Rappelons que la feuille de style se compose tant des règles définies au niveau du navigateur (feuille de style par défaut) que de celles qui pourraient résulter du choix de l'internaute (préférences au niveau du navigateur) et, évidemment, de celles du concepteur de la page.

D'autre part, une règle est souvent partielle. Elle ne définit pas tous les paramètres des éléments considérés. Pour établir l'ensemble des paramètres, le navigateur doit tenir compte d'une certaine hiérarchie.

Enfin, comme les règles peuvent être définies à plusieurs endroits, il convient de déterminer, en cas de règles concernant les mêmes éléments, quelle est celle qui doit l'emporter. Nous verrons que dans ce cas, c'est la proximité de la définition par rapport à l'élément qui compte.

1.10.5.1 Algorithme

Pour comprendre l'algorithme, il faut se rappeler que les éléments *HTML* sont emboîtés (l'élément qui contient tous les autres étant **html**).

- Rechercher toutes les déclarations qui concernent l'élément ou la propriété concernée
  S'il n'y en a pas, prendre en considération les déclarations héritées (emboîtement)
  S'il n'y en a toujours pas, prendre en considération les valeurs par défaut
- Trier les déclarations par poids explicites (certaines règles peuvent être déclarées prioritaires par le concepteur ou l'internaute)
- Trier par origine (auteur, internaute, agent)
- Trier par spécificité du sélecteur

Ce dernier point mérite une explication. Il s'agit de calculer le poids des règles qu'il n'est pas encore possible de trier, de la manière qui suit.

Il s'agit de concaténer les trois nombres suivants :

- le nombre d'attributs ***id*** dans le sélecteur (ex: 1)
- le nombre d'attributs ***class*** dans le sélecteur (ex: 1)
- le nombre d'éléments (ex: 3)

Dans notre exemple, le poids est 113. C'est la règle qui a le poids le plus élevé qui l'emporte.

1.10.6 Exercice

1.10.6.1 Énoncé

Si toutes les déclarations qui suivent sont valables pour un même élément **li**, quel sera le sélecteur dont la déclaration sera retenue?

ol ul li.rouge {…}

ul li {…}

li {…}

ol ul li {…}

#titre {…}

li.rouge {…}

Imaginez une partie de la syntaxe de cet élément et ses emboîtements pour que cette situation soit plausible.

1.10.6.2 Correction

ol ul li.rouge {…} 013

ul li {…}	002
li {…}	001
ol ul li {…}	003
#titre {…}	**100**
li.rouge {…}	011

C'est donc la définition de **#titre** qui sera prise en compte. C'est assez logique si on se dit qu'il s'agit d'un élément très particulier puisqu'il est appelé *titre*.

Voici une structure qui rend toutes ces définitions plausibles.

```
<ol>
…
<ul>
…
<li class = « rouge » id = « titre »>…</li>
…
</ul>
…
</ol>
```

1.10.7 Localisation des feuilles de styles

Les règles d'une feuille de styles peuvent se retrouver à plusieurs endroits :

- dans un fichier externe
- dans l'entête du document *HTML*
- dans la balise d'ouverture d'un élément
- dans les paramètres du navigateur

1.10.7.1 Fichier externe

Il est possible, et c'est même la meilleure des solutions, de rassembler les règles dans un fichier texte qui sera téléchargé. La liaison entre le document *HTML* et le fichier « feuille de style » se fait par l'intermédiaire de l'élément **link**.

```
<link rel="stylesheet" href="essai.css" type="text/css">
```

L'attribut **rel** décrit le type de relation entre les fichiers. L'attribut **href** précise la localisation du fichier. L'attribut **type** décrit le type de fichier au navigateur.

1.10.7.2 Entête du document HTML

Des règles peuvent également être incluses à l'intérieur de l'élément **style** de l'entête. La syntaxe des règles est rigoureusement la même.

```
<style type="text/css">
<!--
.rouge {font-family: Arial, Helvetica, sans-serif; font-size: 12px; color: #FF0000}
-->
</style>
```

Les balises de commentaires sont ignorées par les navigateurs s'ils sont capables d'interpréter le langage *CSS*.

1.10.7.3 Élément

La seule possibilité à ce niveau est d'utiliser un attribut. Il s'agit de l'attribut ***style*** dont la valeur est la règle à prendre en compte. La syntaxe est légèrement différente (pas de sélecteur évidemment et pas d'accolades).

```
<h1 style="color : #00ff00">
```

Rappelons que la priorité est accordée aux paramètres qui sont les plus proches de l'élément. Cette dernière technique est donc une façon de revoir, en dernière minute, le style d'un élément. Il n'est pas dit que c'est une manière efficace de procéder.

1.11 Les formulaires et JavaScript

L'intérêt d'un langage de script côté client est évident :

- il permet une certaine interactivité ;
- il autorise un certain contrôle des interactions qui serait trop lourd à demander à un serveur.

Dans cette section, nous présentons et commentons certains scripts. La plupart sont liés à la gestion des formulaires. Le but est de montrer comment fonctionnent le modèle objet et le modèle des événements en *JavaScript* et de faire percevoir qu'à peu près tout est permis en matière d'interaction. Beaucoup de ces scripts sont des classiques que l'on retrouvera sous d'autres formes sur des sites Web.

1.11.1 Scripts généraux

Nous avons déjà proposés des scripts à titre d'exercices. En voici d'autres. Certains illustrent la communication possible entre les langages et en particulier, entre *JavaScript* et *PHP*.

1.11.1.1 Faire afficher la date et l'heure

Cet exemple illustre la possibilité de créer un objet de type *Date* et d'en extraire les informations. Un objet est créé en utilisant le mot-clé *new*. Diverses méthodes peuvent être invoquées sur des objets de type *Date* : *getDate()*, *getMonth()*,…

Notez que *getDate()* renvoie un nombre entre 1 et 31 alors que *getDay()* renverrait un nombre entre 0 et 6 correspondant au jour de la semaine.

```
<script language="javascript">
<!--
maintenant = new Date()
document.write("<br>Aujourd'hui, nous sommes le ", maintenant.getDate(), "/",
maintenant.getMonth() + 1, "/" , maintenant.getYear())
document.write("<br>Il est ",maintenant.getHours(),":",maintenant.getMinutes(),'.');
-->
</script>
```

1.11.1.2 Faire afficher la date du serveur

Cet exemple illustre le passage d'un langage à l'autre. Nous utilisons ici la méthode *write()* de l'objet **document**. Le mini script *PHP* fournira la date du serveur sous forme d'une chaîne de caractères.

```
<script language="Javascript">
```

```
<!--
document.write('Nous sommes le ' + '<?php echo date("d/m/Y"); ?>'+'.');
-->
</script>
```

Cette dernière sera concaténée à un bout de phrase par *JavaScript* (opérateur + pour la concaténation). Notez que *date()* est une fonction *PHP*[13].

Le fichier qui contiendra ce script devra avoir l'extension *php*.

1.11.1.3 Faire afficher la date de dernière modification du document

On s'intéresse ici à la propriété ***lastModified*** de l'objet **document**. Le résultat (une chaîne de caractères) est ensuite découpé en morceaux pour reconstituer la date sous un format européen.

```
<script language="javascript">
<!--
date=document.lastModified
jour=date.charAt(3)+date.charAt(4)
mois=date.charAt(0)+date.charAt(1)
annee=date.charAt(6)+date.charAt(7)+date.charAt(8)+date.charAt(9)
document.write("<br>Ce document a été modifié pour la dernière fois le  " + jour +
"/" + mois + "/" + annee)
-->
</script>
```

La méthode *charAt()* s'applique à tous les objets chaînes de caractères et renvoie le caractère à la position indiquée par l'argument (l'index commence à 0 comme d'habitude). Une autre possibilité est d'utiliser la méthode *substring()*.

On observe aussi que *JavaScript* peut produire des balises (exemple ici : `<br>`) qui seront interprétées par le navigateur.

1.11.1.4 Recommander un site à un(e) ami(e)

Il s'agit d'un script simple mais qui montre la souplesse d'un langage comme *JavaScript*. Les variables *email*, *sujet* et *message* correspondent à des chaînes de caractères. La propriété ***location*** de l'objet **window** sert à rediriger la fenêtre vers une nouvelle adresse (URL). Autrement dit, le navigateur fera appel au client mail en fournissant le destinataire (*adresse*), le sujet (*sujet*) et le corps du message (*message*). L'adresse est fournie par l'internaute à travers une boîte de dialogue (fonction *prompt()*).

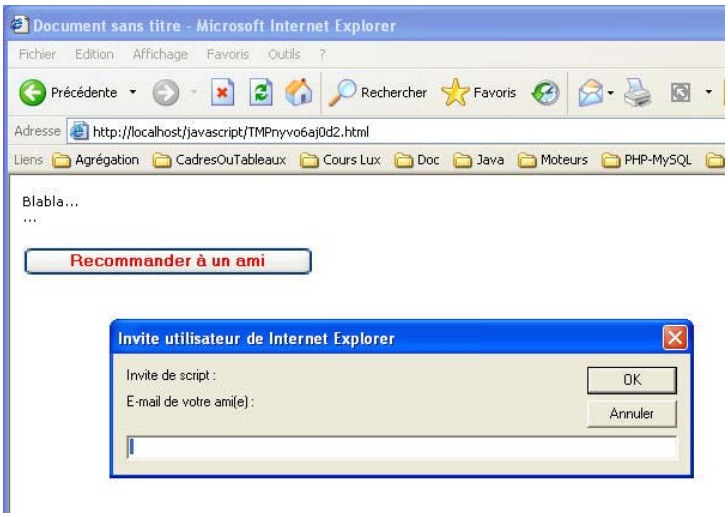
```
<script language="javascript">
<!--
function recommander(){
var adresse= prompt('E-mail de votre ami(e) :',' ');
var sujet= "Le CeFIS - un centre de formation aux technologies";
var message= "Le CeFIS est le centre de formation que vous cherchez !! Cliquez-ici :
http://www.det.fundp.ac.be/cefis/";
window.location="mailto:"+adresse+"?subject="+sujet+"&body="+message;
```

13 Pour information : http://be.php.net/manual/fr/function.date.php

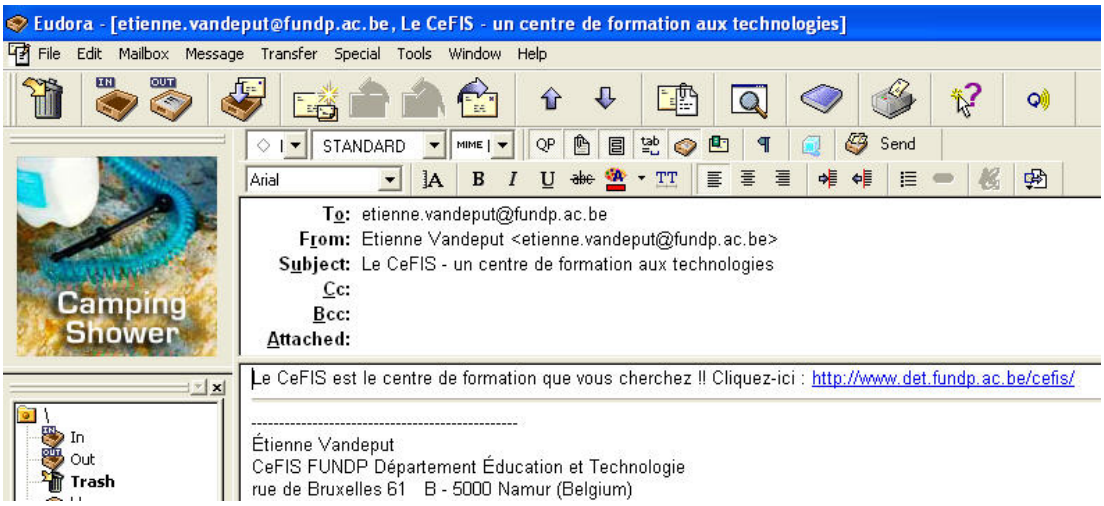
```
}
-->
</script>
```

Un seul bouton est nécessaire pour le déclenchement de la fonction *recommander()*.

```
<form name="form1" method="post" action="">
  <p>
    <input name="envoi" type="button" class="important1" value="Recommander &agrave;
un ami" onclick="recommander()">
  </p>
  </form>
```



En voici une illustration :



1.11.2 Scripts liés aux formulaires

1.11.2.1 Transformer en majuscules le contenu saisi dans un champ

Dans cet exemple, on découvre que la valeur d'un attribut événement peut être une instruction et non une fonction. La valeur de l'attribut commence par le mot-clé *javascript:* suivi de l'instruction.

```
<form name="form1" method="post" action="">
  <p>Nom:
    <input type="text" name="titre"
onChange="javascript:this.value=this.value.toUpperCase();">
  </p>
  <p>Pr&eacute;nom:
    <input type="text" name="auteur">
  </p>
</form>
```

Le script se résume à cette seule instruction. Lorsque l'objet *titre* perdra le focus, son contenu sera transformé en majuscules.

1.11.2.2 Vider le contenu d'un champ en cliquant sur un bouton

Ce script peut être intéressant lorsque le champ contient une valeur par défaut et que cette valeur ne convient pas. Cela évite à l'utilisateur du formulaire de devoir effacer manuellement la valeur.

```
<script language="javascript">
<!--
function vide(n)
{
document.form1.elements[n].value="";
}
-->
</script>
```

En cas de clic sur le bouton, le contenu de l'élément numéro 0 est effacé.

```
<form name="form1" method="post" action="">
  <p>Valeur:
    <input type="text" value="Valeur par défaut">
    <input type="button" value="Effacer" onclick="vide(0)">
  </p>
</form>
```

1.11.2.3 Vider le contenu d'un champ (valeur par défaut) lorsqu'on lui donne le focus

L'utilité de ce script est semblable à celle du script précédent. La manipulation est encore plus rapide. Il est parfois intéressant d'employer cette technique pour garnir le champ d'une valeur par défaut qui donne une indication sur l'information à fournir (*« votre nom »*).

```
<input name="nom" type="text" id="nom" value="Votre nom" size=18 maxlength=75
onfocus="this.value=''">
```

1.11.2.4 Contrôler que le contenu d'un champ est d'une longueur minimale

La fonction *alert()* affiche une boîte de dialogue comportant un message d'alerte. C'est la propriété *length* du contenu du champ qui est testée. La fonction renvoie un booléen car elle est utilisée comme valeur de l'attribut ***onsubmit*** du formulaire.

```
<script language="javascript">
<!--
function minimum(champ,nbre){
if (champ.length < nbre){
  alert("Le mot de passe doit compter "+nbre+" caractères au minimum.");
  return false;
}
return true;
}
-->
</script>
```

Au niveau du formulaire, on aura :

```
<form name="form1" method="post" action="../exemples/ok.php" onsubmit="return
minimum(this.mp.value,6)">
  <p>Identificateur
    <input name="id" type="text" id="id">
  </p>
  <p>Mot de passe
    <input name="mp" type="password" id="mp">
  </p>
  <p>
    <input name="envoi" type="submit" id="envoi" value="Envoyer">
  </p>
</form>
```

Un des paramètres est la référence faite à la propriété ***value*** du champ *mp* du formulaire courant (**this**).

1.11.2.5 Augmenter la taille d'un champ en fonction de son contenu

Ce script permet à un champ de s'élargir si son contenu est trop « large » pour la taille du champ. Cet agrandissement n'a toutefois lieu que lorsque le champ perd le focus.

```
<script language="JavaScript">
function augmente(){
var taille = document.form1.champ.size;
if(document.form1.champ.value.length>=taille){
document.form1.champ.size = document.form1.champ.value.length;}
else {document.form1.champ.size = taille;}
}
</script>
```

C'est la valeur de l'attribut ***size*** du champ texte et la propriété de longueur du contenu de celui-ci qui sont concernés par ce script. Lorsqu'une modification du contenu du champ a lieu, la fonction *augmente()* est activée.

```
<form name="form1" method="post" action="">
  <p>Champ qui s'agrandit:
    <input type="text" name="champ" maxlength="50" size="2" onChange="augmente()">
  </p>
  <p>Autre champ
    <input type="text" name="autrechamp">
  </p>
</form>
```

1.11.2.6 Vérification (simple) d'une adresse email

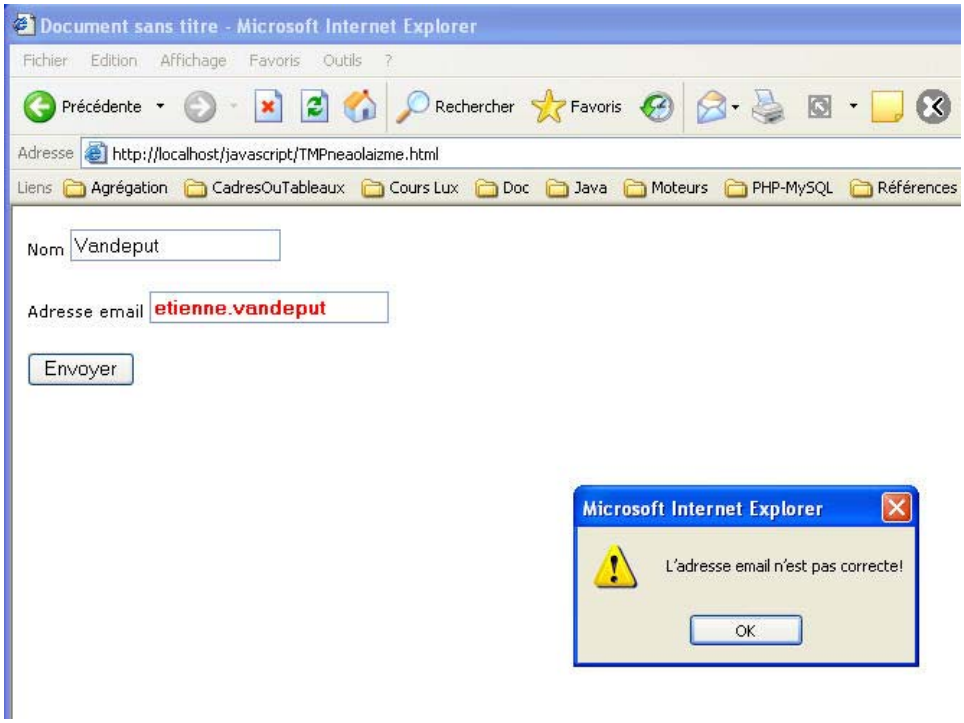
On se contente ici de vérifier que l'arrobase fait bien partie de la chaîne encodée.

```
<script language="javascript">
<!--
function arrobase_ok(email){
var a=document.forms[0].email.value;
var test="" + a;
for(var k = 0; k < test.length;k++){
  var c = test.substring(k,k+1);
  if(c == "@"){
    return true;
  }
}
alert("L'adresse email n'est pas correcte!");
return false;
}
-->
</script>
```

Le corps du document contiendra la définition suivante de formulaire:

```
<form name="form1" method="post" action="" onsubmit="return arrobase_ok(email)">
  <p>Nom
    <input type="text" name="nom">
  </p>
  <p>Adresse email
    <input name="email" type="text" class="important1">
  </p>
  <p>
    <input type="submit" name="envoi" value="Envoyer">
  </p>
</form>
```

En cas de fourniture d'adresse dépourvue d'arrobase, un message sera affiché comme le montre l'illustration ci-après.



1.11.2.7 Remplir une liste en fonction d'un choix par bouton radio

Cet exemple montre que les choix proposés aux internautes peuvent être fonction d'autres choix que font ces derniers. Il illustre aussi la création d'objet et inclus des éléments de mise en page au moyen de styles *CSS* (voir dernière ligne de code et illustration).

```
<script language="javascript">
<!--
function animaux(form,list){
list.options.length=0;
var o=new Option("Lapin","1");
form.liste.options[form.liste.options.length]=o;
```

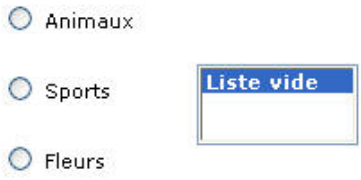
```
var o=new Option("Chat","2");
form.liste.options[form.liste.options.length]=o;
var o=new Option("Chien","3");
form.liste.options[form.liste.options.length]=o;
var o=new Option("Renard","4");
form.liste.options[form.liste.options.length]=o;
var o=new Option("Canard","5");
form.liste.options[form.liste.options.length]=o;
}

function sports(form,list){
list.options.length=0;
var o=new Option("Tennis","1");
form.liste.options[form.liste.options.length]=o;
var o=new Option("Basket","2");
form.liste.options[form.liste.options.length]=o;
var o=new Option("Natation","3");
form.liste.options[form.liste.options.length]=o;
}

function fleurs(form,list){
list.options.length=0;
var o=new Option("Rose","1");
form.liste.options[form.liste.options.length]=o;
var o=new Option("Tulipe","2");
form.liste.options[form.liste.options.length]=o;
var o=new Option("Narcisse","3");
form.liste.options[form.liste.options.length]=o;
var o=new Option("Jonquille","4");
form.liste.options[form.liste.options.length]=o;
}
-->
</script>
<link href="../cefis.css" rel="stylesheet" type="text/css">
```

Les fonctions *animaux()*, *sports()* et *fleurs()* ont pour but de créer des listes. Une de ces listes sera créée en fonction du choix de l'internaute. Un objet **Option** est créé en donnant son étiquette et sa valeur.

Dans la description du formulaire qui suit, on voit qu'en fonction du bouton radio cliqué, c'est l'une ou l'autres des trois fonctions qui est activée, créant une liste d'animaux, de sports ou de fleurs.

```
<form name="form1" action="" method="get">
  <table width="260" cellspacing="2" cellpadding="5">
```

```
    <tr>
      <td width="96"><p>
          <input type="radio" name="choix" value="Animaux" onClick =
"animaux(this.form,this.form.liste)">
          Animaux</p>
          <p>
          <input type="radio" name="choix" value="Sports" onClick =
"sports(this.form,this.form.liste)">
          Sports </p>
              <p>
              <input type="radio" name="choix" value="Fleurs" onClick =
"fleurs(this.form,this.form.liste)">
          Fleurs </p></td>
      <td width="132"><select name="liste" size="3" class="gris" align="top">
          <option value="Liste vide" selected>Liste vide</option>
        </select></td>
    </tr>
  </table>
  </form>
```



Au départ, la liste est vide ou plutôt, elle ne contient que l'item *« Liste vide »*. Lorsqu'un choix est fait, la liste se remplit des items correspondants.

1.12 Exercice

Créez un formulaire à votre goût et selon vos intérêts. Il devra contenir des champs texte, une liste déroulante, un groupe de boutons radios et l'une ou l'autre case à cocher, un bouton simple qui permet de faire exécuter un script local (vérification du remplissage de certains champs) et un bouton de soumission. N'oubliez pas de donner des étiquettes aux objets du formulaire.

La page contiendra la date du jour et la date de dernière modification.

Lorsque le formulaire sera soumis, une page contenant des bribes de code *PHP* reconstituera un document *HTML* reprenant quelques-unes des données fournies avec, si possible, un petit traitement sur l'une ou l'autre donnée (par exemple en réécrivant nom et prénom dans l'ordre inverse avec l'initiale du prénom seulement)[14].

[14] Vous pouvez déjà consulter la liste des fonctions *PHP* pour faire quelques manipulations simples.

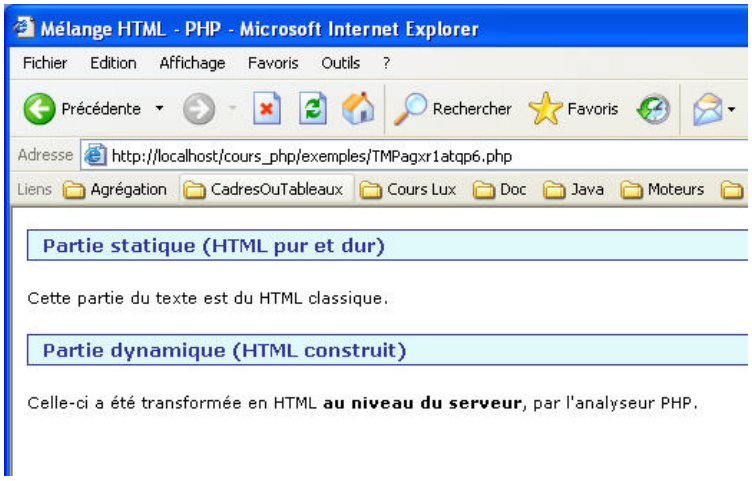
2. Le langage PHP

2.1 En quoi consiste PHP ?

PHP est un langage de scripts *open ource*[15]. Comme beaucoup d'autres langages, il a été spécialement conçu pour le développement d'applications web. Il peut être intégré au *HTML*. Pour ce faire, le code *PHP* est inclus entre une balise de début (ensemble de symboles) et une balise de fin qui permettent au serveur web de passer en mode *PHP*. La partie *PHP* correspond donc à la partie créative et dynamique du document *HTML* finalement envoyé par le serveur et que le navigateur transformera en page Web.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>M&eacute;lange HTML - PHP</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="../cefis.css" rel="stylesheet" type="text/css">
</head>
<body>
<h4>Partie statique (HTML pur et dur)</h4>
<p>Cette partie du texte est du HTML classique.</p>
<h4>Partie dynamique (HTML construit)</h4>
<p><?php echo "Celle-ci a été transformée en HTML <strong>au niveau du
serveur</strong>, par l'analyseur PHP."; ?></p>
</body>
</html>
```

À l'affichage, cela donne :



[15] Un logiciel est dit *open source* si le code source est inclus avec sa version compilée, encourageant ainsi réellement sa modification ou son adaptation. Pour être considéré comme tel, le logiciel doit être distribué gratuitement, le code source doit y être inclus de sorte que chacun puisse le modifier et le redistribuer.

Contrairement au code *JavaScript*, le code *PHP* est exécuté par le serveur et non par le client. Le client ne reçoit que **le résultat du script**, sans aucun moyen d'avoir accès au code qui a produit ce résultat. Si vous donnez aux fichiers *HTML* correspondant à des pages statiques l'extension *PHP*, l'internaute n'a aucun moyen de distinguer les pages qui sont produites dynamiquement des pages statiques. Voici la partie intéressante de la source qu'a reçue le navigateur dans l'exemple précédent :

```
<body>
<h4>Partie statique (HTML pur et dur)</h4>
<p>Cette partie du texte est du HTML classique.</p>
<h4>Partie dynamique (HTML construit)</h4>
<p>Celle-ci a été transformée en HTML <strong>au niveau du serveur</strong>, par
l'analyseur PHP.</p>
</body>
```

Tous les fichiers contenant des instructions en *PHP* doivent posséder l'extension **.php**[16].

2.2 Qu'offre PHP?

PHP est principalement conçu pour servir de **langage de script coté serveur**. Il est capable de réaliser tout ce qu'un script *CGI* peut faire[17]. Il est principalement utilisé pour écrire des scripts de collecte de données issues de formulaires, de stockage éventuel de ces données dans une base de données, de génération dynamique de contenu (voir petit exemple ci-dessus),… C'est l'utilisation la plus traditionnelle et le principal objet de *PHP*. Trois composants sont nécessaires:

- un analyseur *PHP*,
- un serveur Web (*Apache*, par exemple),
- un navigateur web.

Le serveur Web doit s'exécuter en corrélation avec *PHP*, ce qui nécessite une configuration particulière et pas forcément évidente à réaliser. C'est pourquoi il existe des environnements logiciels grâce auxquels ces problèmes passent au second plan[18].

Notez qu'il est aussi possible d'écrire des scripts *PHP* et de les exécuter en ligne de commande, sans l'aide du serveur Web et d'un navigateur. Il faut, pour cela, disposer de l'exécutable *PHP*. Cette utilisation est intéressante pour des scripts qui doivent être exécutés régulièrement.

PHP est utilisable sur la majorité des systèmes d'exploitation et supporte la plupart des serveurs Web actuels et notamment *Apache* et *Microsoft Internet Information Server* (*IIS*).

Vous avez également le choix d'utiliser la programmation procédurale, la programmation objet, ou un mélange des deux.

---

16 D'autres extensions sont reconnues telles *php3*, *phtml*,… pour autant qu'elles soient fournies au serveur Web via le fichier *.httpd.conf*. Par exemple : *AddType application/x-httpd-php .phtml .pwml .php3 .php4 .php .php2 .inc*.

17 *CGI* (*Common Gateway Interface*) est un moyen de faire communiquer le client Web avec des applications écrites dans différents langages et disponibles au niveau du serveur. L'utilisation de scripts *CGI* demande, bien entendu, quelques efforts de configuration.

18 http://www.easyphp.org/telechargements.php3

PHP ne se limite pas à la production de code *HTML*. Il peut aussi générer des images, des fichiers *PDF*, et même des animations *Flash*. Il génère facilement du texte et donc aussi, du code *XML* ou *XHTML*. *PHP* génère ces fichiers et les sauve dans le système de fichiers, ou bien les envoie au navigateur web.

Mais, ce qui nous intéresse davantage, *PHP* possède de nombreuses fonctions permettant d'exploiter les bases de données parmi lesquelles: *InterBase*, *PostgreSQL*, *dBase*, *MySQL*, *IBM DB2*, *ODBC*, *Informix*, *Oracle* et *Ingres*, pour ne citer que les plus connues.

2.3 Les bases du langage

2.3.1 Éléments de syntaxe

2.3.1.1 Les balises PHP

Il y a quatre jeux de balises pouvant servir à délimiter des blocs de code *PHP*. Seuls deux d'entre eux sont actifs à coup sûr[19]: `<?php. . .?>` et `<script language="php">. . .</script>`.

```
<html>
  <head>
    <title>M&eacute;lange HTML - PHP</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  </head>
  <body>
    <p>Cette partie du texte est interpr&eacute;t&eacute;e par le navigateur.</p>
    <p>
    <script language="php"> echo "Celle-ci le sera par l'analyseur PHP."; </script>
    </p>
  </body>
</html>
```

Cette deuxième syntaxe est évidemment plus lourde. Dans la suite de ces notes, nous utiliserons toujours la première.

2.3.1.2 Les séparateurs d'instructions

Le séparateur d'instructions est le symbole "`;`". Il est **nécessaire partout** sauf devant la balise de fin d'exécution du PHP soit "`?>`".

2.3.1.3 Les commentaires

Comme dans de nombreux autres langages, les caractères se trouvant entre les couples de symboles "`/*`" et "`*/`" ne sont pas pris en compte par l'analyseur.

Les commentaires jusqu'en fin de ligne commencent par les symboles "`//`" ou le symbole "`#`".

```
<html>
  <head>
    <title>M&eacute;lange HTML - PHP</title>
```

[19] Les balises courtes (sans le mot-clé *php*) et les balises *ASP* peuvent être activées et déasctivées au niveau du fichier *php.ini*.

```
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  </head>
  <body>
    <p>Cette partie du texte est interpr&eacute;t&eacute;e par le navigateur.</p>
    <p>
    <?php
      /*
      echo est une fonction d'affichage.
      Elle est suivie d'une chaîne de caractères.
      */
      echo "Celle-ci le sera par l'analyseur PHP.<br>"; // La fonction print est ...
      print "Et celle-ci aussi!"; # La preuve!
    ?>
    </p>
  </body>
</html>
```

2.3.2 Variables

2.3.2.1 Variables internes

Les variables sont représentées par le signe dollar "`$`" suivi du nom de la variable. Le nom est sensible à la casse (`$x` ≠ `$X`).

Un nom de variable doit commencer par une lettre ou le caractère souligné (_), suivi de lettres, chiffres ou soulignés.

```
<html>
<head>
<title>Nommage des variables et affectations</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<?php
  $prenom='Étienne';
  $Prenom="Charles";
  echo $prenom,' est l\'assistant de ',$Prenom,'.';
?>
</body>
</html>
```

Dans cet exemple, on constate que

- la casse des caractères est importante dans les noms de variables,
- les chaînes de caractères constantes s'écrivent entre guillemets ou apostrophes,
- la fonction `echo` admet plusieurs arguments séparés par des virgules,

- une vraie apostrophe dans une chaîne de caractères, elle-même délimitée par des apostrophes, doit être précédée de la barre oblique inverse (\).

Les variables sont affectées par **valeur** mais peuvent aussi l'être par **référence** (à partir de *PHP4*). Dans ce deuxième cas, la nouvelle variable ne fait que référencer la variable originale. Les modifications de la nouvelle variable affecteront l'ancienne, et vice versa. Pour assigner par référence, on ajoute le symbole "&" au nom de la variable source.

```
<html>
<head>
<title>Nommage des variables et affectations</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<?php
  $a=12;
  $b=23;
  print $a+$b.' ';

// Affectation par valeur
  $c=$a;
  $c=20;
  print $a+$b.' ';

// Affectation par référence
  $c=&$a;
  $c=15;
  print $a+$b;
?>
</body>
</html>
```

Observez que l'affectation se fait au moyen du symbole "=" alors que l'opérateur de comparaison est (voir plus loin) "==".

L'instruction `print $a+$b.' ';` est particulière. La première expression est le résultat d'une opération entre deux entiers (voir les types, ci-après), alors que la seconde est une chaîne constituée du seul caractère "espace". L'opérateur entre les deux, symbolisé par le point (.), est un opérateur de concaténation. L'expression entière est donc automatiquement transformée en chaîne de caractères avant d'être affichée.

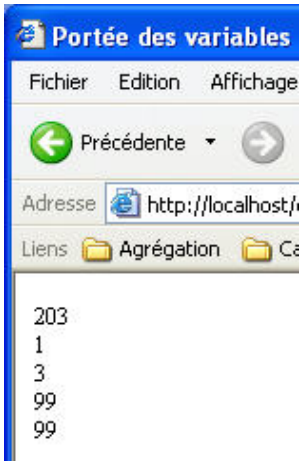
2.3.2.2 Portée des variables

Les variables ont une portée globale si elles ne sont pas définies à l'intérieur d'une fonction. Elles sont accessibles partout dans le script. Les variables définies à l'intérieur d'une fonction sont locales. De même, les variables définies en dehors d'une fonction ne sont pas directement accessibles à l'intérieur de celle-ci.

Il est possible d'accéder aux variables globales à l'intérieur des fonctions. De même, il est possible de rendre visibles partout les variables définies dans une fonction.

Le script qui suit illustre la portée des variables. À l'intérieur d'une fonction, il est possible d'accéder à une variable globale par l'intermédiaire du tableau prédéfini **`$GLOBALS[]`**. Ce tableau est indexé par les noms des variables (sans le symbole **`$`**). Pour rendre une variable locale visible, il faut utiliser la déclaration **`global`**.

```
<?php
  $a=1;
  $b=2;
  function portee(){
      $a=101;
      $b=102;
      echo $a+$b.'<br>'; // Affichage de la somme à l'intérieur de la fonction
      echo $GLOBALS['a'].'<br>'; // Accès à la variable globale $a (extérieure)
      global $c; // Rendre la variable $c locale visible partout
      $c=99;
  }
  portee();
  echo $a+$b; echo '<br>'; // Affichage de la somme à l'extérieur de la fonction
  echo $c.'<br>'; // Affichage de la valeur de $c à l'extérieur de la fonction
  echo $GLOBALS['c'].'<br>';
?>
```

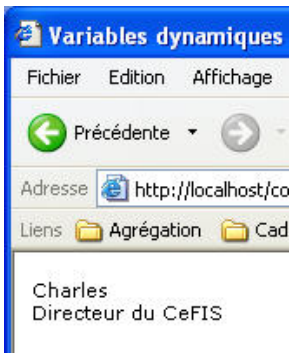


Voici le résultat : la fonction `portee()` est exécutée, fournissant la somme des valeurs des variables locales et la valeur de la variable globale `$a`. Puis c'est la somme des valeurs des variables globales qui est affichée, suivie de la valeur de la variable `$c` qui a été rendue globale et qui est obtenue de deux manières différentes, vu que la valeur de la variable `$c` garnit aussi le tableau `$GLOBALS[]`.

2.3.2.3 *Variables dynamiques*

Il est possible de s'arranger pour que le nom d'une variable soit lui-même variable. C'est particulièrement pratique pour éviter les problèmes de nomination. Cette opportunité existe en utilisant le symbole dollar (`$`) une seconde fois. Voyez l'exemple qui suit.

```
<?php
  $a="Charles";
  $$a="Directeur du CeFIS";
  echo $a.' '.$Charles;
?>
```



L'intérêt de pouvoir créer des variables dynamiquement est d'éviter de devoir utiliser des tableaux, voire de se casser la tête pour trouver des noms de variables.

En voici un exemple :

```
<?php
foreach($_POST as $key => $value) {
       $varname = "_".$key;
       $$varname = $value;
}
?>
```

Même si vous ne connaissez pas encore cette structure répétitive, il est facile de comprendre ce que fait cette partie de script. Les valeurs d'un formulaire décorent le tableau `$_POST` qui contient les valeurs envoyées au serveur par le client. Pour éclater ce tableau en autant de variables que nécessaire, on crée des variables contenant à chaque fois une chaîne de caractères commençant par le caractère de soulignement, suivi du nom de la variable (clé d'index du tableau). On utilise ces chaînes comme nom pour des variables qui contiendront les valeurs transmises.

Exemple : si les couples clés-valeurs transmis sont (*nom*, *Vandeput*), (*prenom*, *Étienne*), (*departement*, *DET*), les variables créées seront respectivement `$_nom`, `$_prenom` et `$_departement` et contiendront les valeurs *Vandeput*, *Étienne* et *DET*.

2.3.2.4 Variables externes

PHP manipule aussi des variables provenant de l'extérieur comme, nous venons juste de le rappeler, les variables provenant de la soumission d'un formulaire. C'est d'ailleurs là une de ses qualités. Ces variables sont aisément accessibles à travers des tableaux **prédéfinis**. Parmi ceux-ci, notons `$GLOBALS` qui rassemble les variables globales, `$_POST` et `$_GET` qui contiennent respectivement les valeurs transmises par la méthode *post* et celles qui ont été transmises par la méthode *get*, ou encore `$_SERVER`.

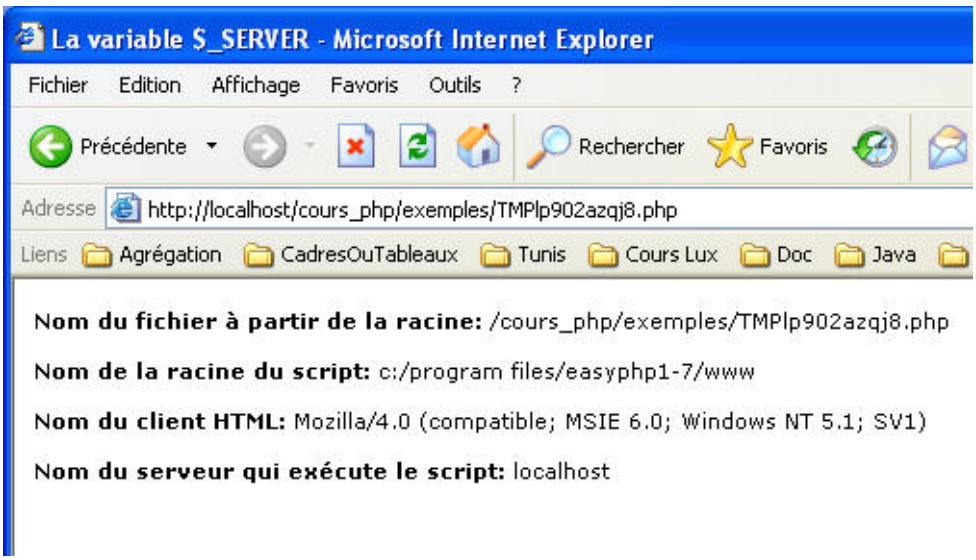
Nous vous renvoyons à la documentation pour plus de détails. Nous donnons ici quelques exemples d'informations normalement[20] disponibles au niveau du serveur…

```
<html>
<head>
<title>La variable $_SERVER</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="../cefis.css" rel="stylesheet" type="text/css">
</head>

<body>
<p>
<?php
  echo "<span class='important2'>Nom du fichier à partir de la racine: </span>";
  echo $_SERVER['PHP_SELF'].'<br><br>';
  echo "<span class='important2'>Nom de la racine du script: </span>";
  echo $_SERVER['DOCUMENT_ROOT'].'<br><br>';
  echo "<span class='important2'>Nom du client HTML: </span>";
```

[20] Il n'y a aucune garantie sur la fourniture de toutes les informations prévues par un serveur donné.

```
  echo $_SERVER['HTTP_USER_AGENT'].'<br><br>';
  echo "<span class='important2'>Nom du serveur qui exécute le script: </span>";
  echo $_SERVER['SERVER_NAME'].'<br><br>';
?>
</p>
</body>
</html>
```

…dont voici le résultat dans un contexte donné :



Un autre moyen d'accéder aux valeurs du tableau `$_server` est d'activer la fonction prédéfinie *phpinfo()* dans un script.

PHP Variables

Variable	Value
_SERVER["COMSPEC"]	C:\WINDOWS\system32\cmd.exe
_SERVER ["DOCUMENT_ROOT"]	c:/program files/easyphp1-7/www
_SERVER["HTTP_ACCEPT"]	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*
_SERVER ["HTTP_ACCEPT_ENCODING"]	gzip, deflate
_SERVER ["HTTP_ACCEPT_LANGUAGE"]	fr-be
_SERVER ["HTTP_CONNECTION"]	Keep-Alive
_SERVER["HTTP_HOST"]	localhost
_SERVER ["HTTP_USER_AGENT"]	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
_SERVER["PATH"]	C:\Program Files\Common Files\Fujitsu\COBOL;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
_SERVER["REMOTE_ADDR"]	127.0.0.1
_SERVER["REMOTE_PORT"]	1228
_SERVER ["SCRIPT_FILENAME"]	e:/data/site_php/tmpavgf9b14pf.php
_SERVER["SERVER_ADDR"]	127.0.0.1
_SERVER["SERVER_ADMIN"]	admin@localhost
_SERVER["SERVER_NAME"]	localhost
_SERVER["SERVER_PORT"]	80

L'instruction à placer dans le fichier *php* est :

```
<?php phpinfo() ?>
```

Voici un petit exemple qui illustre le profit que l'on peut éventuellement tirer de telles informations. La composante `$_SERVER[REMOTE_ADDR]` représente l'adresse IP du client. Supposons qu'une page soit réservée à une catégorie de clients identifiables par leur adresse IP (par exemple, les Facultés : 138.48….). Il suffit de vérifier que le client a bien une adresse commençant par ces deux valeurs. La fonction *explode*(<*séparateur*>,<*chaîne*>) renvoie un tableau dont les composantes sont les éléments de la chaîne <*chaîne*> séparées par le séparateur <*séparateur*>.

```
$adresseIP=explode(".",$_SERVER['REMOTE_ADDR']);
if(adresseIP[0]!="138"||adresseIP[1]!="48"){
      echo htmlentities("Accès limité aux FUNDP");
      exit;
}
```

La fonction *htmlentities*(<*chaîne*>) convertit les caractères spéciaux en entités *HTML* (exemple : `é` devient `&eacute;`. L'utilisation de cette fonction évite un certain nombre de surprises.

2.3.3 Les constantes

Une constante est un identifiant qui représente une valeur simple. Le nom de constante valide commence par une lettre ou un souligné (_), suivi d'un nombre quelconque de lettre, chiffres ou soulignés. Par convention, les constantes sont toujours en majuscules.

Les constantes sont définies grâce à la fonction *define()* et ne peuvent être redéfinies.

```
$a=3;
define("VALMAX","100");
$a+=VALMAX;
echo $a;
```

affichera la valeur 103.

Elles sont accessibles de manière globale.

```
$a=3;
define("VALMAX","100");
$a+=VALMAX;
echo $a.'<br>';
function test(){
echo VALMAX;
}
test();
```

```
103
100
```

2.3.4 Types

Habituellement en *PHP*, le type d'une variable n'est pas déclaré par le programmeur. Il est décidé au moment de l'exécution, en fonction du contexte dans lequel la variable est utilisée (voir exemple ci-dessus pour la variable *adresseIP*).

Il y a huit types de base dont:

quatre types scalaires (valeurs indécomposables)

- booléen
- entier
- nombre à virgule flottante
- chaîne de caractères

deux types composés

- tableau
- objet

et deux types spéciaux

- ressource
- null

Comme d'autres langages, *PHP* offre la possibilité de "transtyper" les variables (casting). Nous examinons rapidement les types scalaires. Les autres types seront analysés plus loin.

2.3.4.1 Booléen

Classiquement, les valeurs sont `true` et `false` (casse sans importance). Les variables des autres types peuvent être transtypées en variable booléenne. Ainsi, par exemple, la valeur entière 0, la valeur en nombre à virgule 0.0, la chaîne vide ou la chaîne "0" prennent la valeur `false` lorsqu'elles sont transtypées en booléen.

2.3.4.2 Entier

Les entiers peuvent être spécifiés en base décimale, en hexadécimal ou en octal et optionnellement être précédés d'un signe.

Selon les systèmes, environ 4 milliard de valeurs signées sont possibles (codage sur 32 bits).

```
<?php
  $i=123456;
  $j=123.456E3;
  $k=01234;
  $l=0xAA45;
  print $i."<br>".$j."<br>".$k."<br>".$l
?>
```

```
123456
123456
668
43589
```

2.3.4.3 Nombre à virgule flottante

Voici quelques exemples illustrant la syntaxe admise.

```
<?php
  $i=123.456;
  $j=1.23456E2;
  $k=.123456e3;
  print $i."<br>".$j."<br>".$k
?>
```

```
123.456
123.456
123.456
```

La taille des nombres décimaux dépend de la plateforme. La configuration la plus répandue permet de coder des nombres dont l'ordre de grandeur est de 1.8E308 avec une précision de 14 décimales. C'est le format 64 bits *IEEE*.

Il ne faut jamais faire confiance aux nombres à virgule jusqu'à leur dernière décimale et s'interdire de les comparer avec l'opérateur d'égalité.

2.3.4.4 Chaîne de caractères

Il y a trois manières différentes de construire des chaînes de caractères:

- en utilisant les apostrophes;
- en utilisant les guillemets;
- en utilisant la syntaxe *HEREDOC* (dont nous ne parlerons pas).

2.3.4.5 Les apostrophes

```
$prenom='Étienne';
```

Si la chaîne doit contenir des apostrophes, vous devez les échapper avec la barre oblique inverse (\).

```
$titre='L\'école et les TIC';
```

C'est pareil pour la barre oblique inverse elle-même.

```
$remarque='Utilisez la barre oblique inverse (\\) pour échapper les caractères.';
```

Les noms de variables présents dans la chaîne apparaîtront tels quels.

```
$c='La variable $nom';
```

```
<?php
$prenom='Étienne<br>';
$titre='L\'école et les TIC<br>';
$remarque='Utilisez la barre oblique inverse (\\) pour échapper les caractères.<br>';
$c='La variable $nom';
echo $prenom.'<br>'.$titre.'<br>'.$remarque.'<br>'.$c;
?>
```

Étienne

L'école et les TIC

Utilisez la barre oblique inverse (\) pour échapper les caractères.

La variable $nom

Observez que les balises HTML qui ne sont que du texte peuvent être incluses dans les chaînes de caractères. Elles seront interprétées par le navigateur.

2.3.4.6 Les guillemets

L'apostrophe ne doit plus être échappé mais bien le guillemet. **Les noms de variables sont remplacés par leurs valeurs respectives.**

```
<?php
$prenom="Étienne";
$titre="Le cours \"PHP\"<br>";
```

```
$remarque="Utilisez la barre oblique inverse (\\) pour échapper les caractères.<br>";
$c="Le cours PHP est donné par $prenom.<br>";
$d='Le cours PHP est donné par $prenom.<br>';
$e='Le cours PHP est donné par '.$prenom.'.<br>';
echo $prenom."<br><br>".$titre."<br>".$remarque."<br>".$c."<br>".$d."<br>".$e;
?>
```

Étienne

Le cours "PHP"

Utilisez la barre oblique inverse (\) pour échapper les caractères.

Le cours PHP est donné par Étienne.

Le cours PHP est donné par $prenom.

Le cours PHP est donné par Étienne.

2.3.5 Opérateurs

Comme tout langage de programmation, *PHP* propose des opérateurs et des fonctions prédéfinies, mais offre aussi au programmeur la possibilité de construire ses propres fonctions. Les opérateurs (comme les fonctions prédéfinies) sont trop nombreux pour être décrits ici. Vous pouvez vous référer à la documentation en ligne de *PHP*[21]. Dans ce qui suit, nous nous contentons d'évoquer ceux, parmi les moins classiques, qui présentent un intérêt immédiat. Ils sont utilisés et documentés dans les exemples qui suivent.

2.3.5.1 Opérateurs d'assignation

L'assignation se fait classiquement en utilisant le symbole **=**. Des facilités sont offertes comme dans de nombreux autres langages :

opérateurs combinés **+=** ou **.=**

```
$a += $b ; // $a = $a + $b ;
$a .= $b ; // $a = $a . $b ;
```

assignations combinées

```
$a = ($b = 5) + 1; // $b = 5 ; $a = $b + 1 ;
```

2.3.5.2 Opérateurs de comparaison

La comparaison peut se faire sur les valeurs mais également sur les types. C'est la raison pour laquelle, à l'opérateur classique de comparaison **==**, s'ajoute un opérateur qui vérifie en plus l'égalité des types **===**.

$a === $b possède la valeur **true** uniquement si les deux variables contiennent les mêmes valeurs et sont de même type.

On trouve de même des opérateurs **!=** et **!==** pour tester la différence des valeurs et la différence des valeurs ou des types.

$a !== $b possède la valeur **true** si les deux variables contiennent des valeurs différentes ou sont de types différents.

---

21 http://www.php.net/manual/fr/

2.3.5.3 Opérateurs d'incrémentation décrémentation

Ces opérateurs peuvent être utilisés de concert avec des tests. C'est la raison pour laquelle on parle de pré ou de post incrémentation/décrémentation.

++$a incrémente **$a** avant de renvoyer sa valeur (pour un test éventuel).

$a++** renvoie la valeur de **$a avant de l'incrémenter.

Les expressions suivantes **--$a** et **$a--** sont les équivalentes en décrémentation.

2.3.5.4 Opérateurs logiques

Les opérateurs logiques habituels, *ET*, *OU* et *NON* sont respectivement symbolisés par **&&**, **||** et **!**. Il existe aussi des opérateurs and et or qui sont équivalents à **&&** et à **||** à ceci près qu'ils ont une priorité différente. Profitons-en pour rappeler qu'il faut être attentif à la priorité des opérateurs et, en cas de doute, utiliser les parenthèses.

2.3.5.5 Opérateurs de chaînes

L'opérateur de concaténation des chaînes de caractères est le point.

Si **$nom** contient la valeur « *Vandeput* » et **$prenom**, la valeur « *Étienne* », le texte affiché par…

```
echo "Bonjour".$prenom." ".$nom.".";
```

… sera…

Bonjour Étienne Vandeput.

Notez la différence d'interprétation du point dans et en dehors des guillemets.

2.3.6 Fonctions

Une fonction est définie de la manière suivante :

```
function f(p1,p2,…,pn){
…
return v;
…
}
```

À partir de *PHP4*, les fonctions peuvent être définies n'importe où. Toutefois, une fonction définie à l'intérieur d'une autre fonction n'existe que si cette dernière est appelée. De même, une fonction définie de manière conditionnelle n'existe que si la condition est testée et vérifiée.

Il est possible de passer les arguments par valeur (méthode par défaut)…

```
function multiplier($a,$b){
$p=$a*$b ;
return $p ;
}
echo multiplier(4,8);
```

…ou par référence…

```
function tripler(&$n){
$n*=3 ;
}
```

```
$a=5 ;
tripler($a);
echo $a;
```

Il existe évidemment de très nombreuses fonctions prédéfinies en *PHP*.

2.3.7 Expressions

L'association des opérateurs, des fonctions, des variables et des constantes donne naissance à des expressions. On peut dire d'une expression, qu'elle a une valeur d'un certain type.

Dans l'exemple qui suit, l'opérateur `%` désigne le reste de la division entière de … par …

```
<?php
$x=12;
$y=$x%5;
echo $x." modulo 5 égale ".$y.".<br>";
$x=($y=4)+5;
echo "x vaut $x et y vaut $y.";
?>
```

…donnera…

```
12 modulo 5 égale 2.
x vaut 9 et y vaut 4.
```

Comme vous pouvez le constater, *PHP* offre les facilités d'écriture des langages récents.

2.3.8 Instructions de contrôle

Comme la plupart des langages de script, *PHP* offre une grande variété d'instructions de contrôle et une grande souplesse d'écriture.

2.3.8.1 L'instruction if

Pour commencer, il est important de se rendre compte que des variables de tous types peuvent être considérées comme des expressions booléennes. Ainsi, une variable **`$x`** utilisée dans un contexte booléen fournira la valeur **`true`** si elle ne contient ni la valeur zéro numérique, ni la chaîne vide, ni la valeur logique **`false`** et qu'elle est définie.

Par exemple,…

```
$x=1;
if($x){
  echo "vrai";
}
else{
  echo "true";
}
```

…affichera **`vrai`**.

Si l'utilisation des accolades n'est pas nécessaire, elle est souhaitable. D'autres instructions pourraient venir s'ajouter par la suite.

2.3.8.2 L'instruction if-elseif-else

L'instruction **if-elseif-else** est une instruction très souple qui permet d'effectuer des tests de natures très différentes. Elle peut comporter autant de **elseif** que nécessaire. À chaque nouveau test, comme dans le bloc **else**, on ne s'intéresse qu'aux cas restants.

```
if($nombre%2){
  echo "Nombre impair";
}
elseif($nombre%3){
  echo "Nombre non divisible par 3";
}
else{
  echo "Nombre pair, multiple de 3";
}
```

2.3.8.3 L'instruction switch

L'instruction **switch** est une alternative à la construction **if-elseif-else**. Elle est plus souple que dans d'autres langages car elle accepte d'autres valeurs que des valeurs entières. On la préfèrera dans les cas où c'est toujours la même expression qui doit être testée.

```
switch($a){
  case ($a>99): $b="excessif"; break;
  case ($a>75): $b="élevé"; break;
  case ($a>50): $b="normal"; break;
  default: $b="faible";
}
```

L'instruction **break** évite le passage aux autres instructions. En l'absence de ces instructions, une valeur de **$a** de **88** ferait afficher **faible** et non **élevé**.

2.3.8.4 L'instruction while

L'instruction **while** est la plus courante des instructions de boucles. Le bloc des instructions qu'elle encadre est exécuté tant que la condition est vérifiée, ce qui implique que celle-ci soit évaluée avant le parcours de la boucle.

```
while($a>50){
  $a-=5;
}
echo $a;
```

Si **$a** possède toujours la valeur **88**, la valeur **48** sera affichée.

Les instructions **break** et **continue** permettent respectivement de quitter la boucle ou d'interrompre le cycle en cours pour passer au cycle suivant.

2.3.8.5 L'instruction do…while

L'instruction **do**…**while** permet à la boucle d'être effectuée au moins une fois. La condition n'est en effet testée qu'en fin de boucle.

Les instructions qui suivent font afficher les dates des jours qui se succèdent d'aujourd'hui jusqu'au premier dimanche de mars.

```
$jourCourant=time();
echo "<ol>\n";
do{
echo "<li>".date("l j F",$jourCourant)." ".date("j",$jourCourant)."
".date("F",$jourCourant)."</li>\n";
$jourCourant+=86400;
}
while(date("l",$jourCourant)!="Sunday"||date("F",$jourCourant)!="March");
echo "</ol>\n";
```

La fonction **`time()`** génère une estampille (le moment présent exprimé en secondes depuis le 1er janvier 1970). La fonction **`date()`** a comme arguments un format et une estampille. Pour la signification des formats, référez-vous à la documentation *PHP*.

1. Tuesday 15 February
2. Wednesday 16 February
3. Thursday 17 February
4. Friday 18 February
5. Saturday 19 February
6. Sunday 20 February
7. Monday 21 February
8. Tuesday 22 February
9. Wednesday 23 February
10. Thursday 24 February
11. Friday 25 February
12. Saturday 26 February
13. Sunday 27 February
14. Monday 28 February
15. Tuesday 1 March
16. Wednesday 2 March
17. Thursday 3 March
18. Friday 4 March
19. Saturday 5 March

Le script augmente d'un jour le moment courant avant d'en afficher les paramètres dans une liste numérotée. Notez que si nous étions le premier dimanche de mars, le script afficherait les paramètres de 365 jours.

2.3.8.6 L'instruction for

L'instruction **`for`** permet de faire des économies d'écriture dans le cas où le nombre de passages dans la boucle est déterminable.

Voici, dans ces conditions, ce que devient le script précédent:

```
echo "<ol>\n";
for(
$jourCourant=time();
date("l",$jourCourant)!="Sunday"
||date("F",$jourCourant)!="March";
$jourCourant+=86400
){
 echo "<li>".date("l j F",$jourCourant)."</li>\n";
}
echo "</ol>\n";
```

2.3.8.7 L'instruction foreach

L'instruction **`foreach`** est relative aux tableaux. Nous la décrirons au moment où nous détaillerons davantage ces structures intéressantes.

2.4 Exercice

Un formulaire (incomplet) se présente de la sorte :

Il s'agit de faire en sorte que le choix du bouton radio « Madame » introduise les modifications graphiques suivantes :

On ajoutera que les différents champs doivent se vider de leur contenu dès qu'ils reçoivent le focus.

L'idée est d'ajouter un attribut événement à chacun des boutons radios en y associant la soumission du formulaire avec, comme action, un script de reconstitution du formulaire. Un premier fichier *HTML* contiendra le formulaire initial. Un second fichier *PHP* contiendra le script qui reconstituera une nouvelle page avec plus ou moins de champs textes selon le choix.

```
<form name="form1" method="post" action="transmis.php">
<p>
<label>
<input name="sexe" type="radio" onClick="this.form.action='sexe.php';
this.form.submit(); " value="0" checked>
Monsieur</label>
<br>
<label>
<input type="radio" name="sexe" value="1" onClick="this.form.action='sexe.php';
this.form.submit(); ">
Madame</label>
Coordonn&eacute;es:
<input name="textfield" type="text" class="gris" value="Nom"
onFocus="this.value='';">
<input name="Submit" type="submit" class="important1" value="Envoyer">
<br>
</p>
</form>
```

Les attributs `class` de certains éléments traduisent l'utilisation de styles personnalisés. Les attributs `onFocus` servent au vidage des champs. Lorsqu'un des boutons est sélectionné, l'action du formulaire est changée et le formulaire est soumis en invoquant le script `sexe.php` dont le code suit et qui va reconstituer un nouveau formulaire.

```
<form name="form1" method="post" action="transmis.php">
<p>
<label>
<input name="sexe" type="radio" onClick="this.form.action='sexe.php';
this.form.submit(); " value="0" checked>
Monsieur</label>
<br>
<label>
<input type="radio" name="sexe" value="1" <?php $sexe=$_POST["sexe"]; if($sexe=="1"){
echo "checked";} ?> onClick="this.form.action='sexe.php'; this.form.submit(); ">
Madame</label>
</p>
<p>Coordonn&eacute;es:
<?php
if($sexe=="1"){
echo "<input name='njf' type='text' class='gris' value='Nom de jeune fille'
onFocus='this.value=\"\"'>\n";
echo "<input name='nfm' type='text' size='25' class='gris' value='Nom de femme
mariée' onFocus='this.value=\"\"'>\n";
}
else{
echo "<input name='nom' type='text' class='gris' value='nom'
onFocus='this.value=\"\"'>\n";
}
?>
<input name="Submit" type="submit" class="important1" value="Envoyer">
<br>
</p>
</form>
```

Le code *PHP* de ce document se trouve à deux endroits. Le premier morceau de code sert à cocher le bouton radio correspondant au choix « Madame » dans le formulaire reconstitué. Il n'est pas nécessaire de le préciser pour l'autre choix. C'est le premier choix qui est sélectionné par défaut. On observera que le code est inséré à l'intérieur même de la définition d'un élément, ce qui montre que les deux langages peuvent être intimement mélangés. Le deuxième morceau permet la création des deux champs ou le retour à un seul champ avec les valeurs par défaut nécessaires.

Au niveau syntaxique, vous noterez que les apostrophes, placés à l'intérieur des guillemets, permettent d'éviter l'échappement de ces caractères. Toutefois, un troisième niveau d'inclusion, pour la chaîne de caractères vide, oblige l'échappement de certains guillemets.

2.5 Les tableaux en PHP

Un tableau est une association ordonnée. Une association est un type qui fait correspondre des **valeurs** à des **clés**. Ce type est optimisé de diverses façons, qui font qu'on peut le manipuler de diverses manières. Comme une valeur peut elle-même être un tableau, vous pouvez simuler facilement un arbre.

2.5.1 Créer un tableau

Les éléments d'un tableau peuvent être créés à la volée.

```
$monTableau[]="Pomme";
$monTableau[]="Poire";
$monTableau[]="Abricot";
$monTableau[]="Cerise";
$monTableau[]="Pêche";
```

Dans ce cas, un index de 0 à … est créé.

Un tableau peut également être créé avec la fonction **`array()`**. Cette fonction prend en argument des structures **`key => value`**, séparées par des virgules.

```
array( [key =>] value , ... )
// key est soit une chaîne soit un entier positif
// value peut être de n'importe quel type (y compris un autre tableau)
```

En voici une illustration :

```
<?php
$table = array("nom" => "Vandeput", 10 => 125);
echo $table["nom"]; // affiche Vandeput
echo $table[10];    // affiche 125
?>
```

Un tableau peut être multidimensionnel:

```
$voitures=array(
      "allemandes" => array("opel", "bmw", "vw", "mercedes"),
      "francaises" => array("peugeot", "citroen", "renault"),
      "italiennes" => array("alpha romeo", "fiat")
);
echo $voitures["francaises"][2]; // affichera renault
```

On constate que le tableau est une structure de données relativement souple.

2.5.2 Fonctions liées aux tableaux

Voici un exemple qui illustre quelques-unes des nombreuses fonctions prédéfinies prenant un tableau en argument. Le tableau est parcouru du début à la fin et les clés sont affichées en regard des valeurs correspondantes.

```
<?php
$coordonnees['nom']="Vandeput";
```

```
$coordonnees['prenom']="Étienne";
$coordonnees['organisme']="CeFIS";
for(reset($coordonnees);$k=key($coordonnees);next($coordonnees)){
  $v=current($coordonnees);
  echo "La valeur de la clé <span class='important2'>".$k."</span> est
  <span class='important2'>".$v."</span><br>";
}
?>
```

Le tableau est rempli à la volée. La boucle **`for`** fait apparaître les trois éléments habituels: initialisation, condition d'arrêt, évolution des variables. Examinons-les un à un.

`reset(coordonnees)` : la fonction **`reset`** place le pointeur interne sur le premier élément du tableau.

`key(coordonnees)` : la fonction **`key`** fournit la clé de l'élément courant. En bout de tableau, la valeur renvoyée est `false`, ce qui convient comme test d'arrêt.

`next(coordonnees)` : la fonction **`next`** place le pointeur interne sur l'élément suivant s'il existe.

`current(coordonnees)` : la fonction **`current`** renvoie la valeur de l'élément courant.

2.6 Les sessions

Dans un certain nombre de cas, il est bon que l'accès aux pages Web d'un site soit réglementé. La technique consiste généralement à demander au client de fournir un identificateur et un mot de passe. Pour éviter le stockage en clair de ce mot de passe dans une base de données, celui-ci peut être encrypté. L'intérêt d'une session réside dans la production de variables dont l'existence servira à vérifier que l'utilisateur s'est identifié correctement. Ces variables, globales en *PHP4*, évitent notamment de devoir véhiculer certaines valeurs d'une page à une autre du site.

Il faut distinguer la création d'un nouvel utilisateur de son authentification. Un script *JavaScript* peut aisément traiter ces deux options au moyen de boutons radios.

2.6.1 Création d'un nouvel utilisateur

Lors de la création d'un nouvel utilisateur, on veillera particulièrement à vérifier l'unicité de l'identificateur et aura soin d'encrypter le mot de passe avant de le stocker dans la base de données. On utilisera deux fichiers. Le premier permettra de saisir les informations provenant de l'utilisateur, le second, de préparer et de stocker ces informations dans la base de données (voir la section consacrée à *MySQL*). On en profitera aussi pour définir des variables de session qu'il ne sera plus nécessaire de véhiculer d'une page à l'autre.

Théoriquement, il faut également veiller à connecter l'utilisateur au site.

2.6.2 Authentification d'un utilisateur

Le processus d'authentification consiste à vérifier que l'identificateur et le mot de passe encrypté concordent. Cela passe évidemment par un accès à la base de données. Ici aussi, on en profitera pour définir les variables de session nécessaires et on connectera l'utilisateur au site.

2.6.3 Identificateur de session

Le principe est que chaque utilisateur reçoit un numéro unique identifiant sa session et appelé « *session ID* ». Sa création est liée au démarrage de la session provoquée par la fonction *session_start()*. Le but de cette fonction est soit de créer une session, si l'utilisateur ne s'est pas encore connecté, soit de restaurer

une session trouvée sur le serveur grâce à ce « *session ID* ». Dans ce cas, ce dernier est fourni, soit par un cookie ou via une requête *GET* ou *POST*. Le fait que certains clients refusent les cookies ne parle toutefois pas en faveur de cette solution toutefois très souple. Le « *session ID* » peut être aisément propagé, à travers les *URL*, grâce à la constante *SID* qui le contient.

```
<a href="autrePage.php?<?php echo SID; ?>">Autre page</a>
```

Grâce à l'identifiant de session, *PHP* peut retrouver toutes les variables associées à cette session.

2.7 Les inclusions de fichiers

Afin de modéliser les pages d'un site, il est utile de récupérer des parties entières de code. *PHP* fournit, avec la fonction *include()*, l'opportunité d'inclure dans un fichier le code de certains autres fichiers. Il existe une autre fonction qui est la fonction *require()*. Dans ce dernier cas, l'absence du fichier provoque l'interruption de l'exécution pour cause d'erreur fatale.

À titre d'exemple, on peut faire en sorte que l'exécution d'un fichier produise l'entête des pages et l'exécution d'un autre, le pied-de-page.

Voici le code du fichier *entete.php*.

```
<html>
<head>
<title>Document sans titre</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="../cefis.css" rel="stylesheet" type="text/css">
<style type="text/css">
<!--
.grand {
      font-size: 24px;
      color: #999999;
}
-->
</style>
<style type="text/css">
<!--
.blanc {
      color: #FFFFFF;
}
td {
      padding: 5px;
}
-->
</style>
</head>
```

```
<body>
<table width="555" height="113" border="1" cellpadding="10" cellspacing="0"
bordercolor="#FF8888" bgcolor="#FFDFD5" class="grand">
  <tr>
    <td height="109">
<div align="center"><img src="../images/cathedrale.jpg" width="112" height="89"
align="left"><span class="grand">D&eacute;partement &Eacute;ducation et
Technologie</span></div>
    </td>
  </tr>
</table>
```

Le fichier contient les instructions d'entête habituelles d'un fichier *HTML*. On y trouve également quelques définitions de styles propres à ce fichier. Notez que les styles seront disponibles au niveau de tous les fichiers qui incluront celui-ci.

L'élément `<body>` est ouvert dans ce fichier et sera fermé dans le fichier pied-de-page. Les autres instructions sont celles de la définition d'un tableau.

Voilà celui du fichier de *pieddepage.php*.

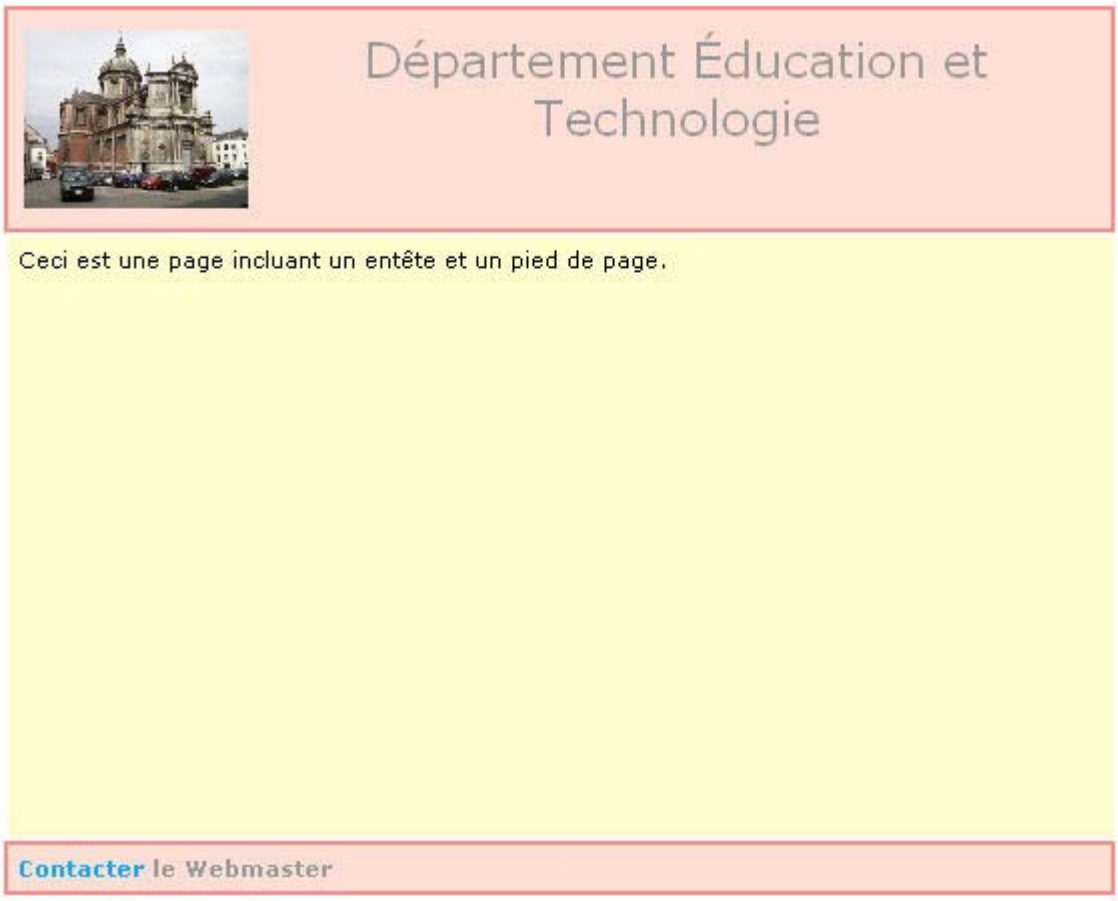
```
<table width="555" border="1" cellpadding="10" cellspacing="0" bordercolor="#FF8888"
bgcolor="#FFDFD5">
  <tr>
    <td><a href="mailto:webmaster@det.fundp.ac.be">Contacter</a>
      <span class="gris"> le Webmaster</span></td>
  </tr>
</table>
</body>
</html>
```

On y trouve la définition d'un autre tableau. Notez qu'il n'était pas nécessaire de clôturer le tableau dans le fichier d'entête. On pouvait en continuer la définition dans les autres fichiers. Le style *gris* est utilisé. Il est défini grâce au fichier précédent. On bénéficie aussi d'autres définitions de styles pour la mise en forme des liens.

Celui qui inclut les deux autres ne contient rien des balises de début et de fin d'un fichier *HTML* classique. En voici le code:

```
<?php include('entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">Ceci est une page incluant
      un ent&ecirc;te et un pied de page. </td>
  </tr>
</table>
<?php include('pieddepage.php'); ?>
```

Il s'agit à nouveau de la définition d'un tableau d'une seule cellule. On remarque la première et la dernière instructions qui sont des instructions *PHP* d'inclusion de fichiers. Et voici le résultat à l'affichage:



2.8 Exercice

Il s'agit de réaliser un entête (*entete.php*) et un pied de page (*pieddepage.php*) qui serviront de modèles pour les pages d'un site. Une première page de ce site (*login.php*) permettra de choisir, soit de s'y connecter, soit de se constituer en tant que nouvel utilisateur. Dans ce cas, on demandera quelques informations (*formulaire.php*): nom, prénom, adresse… Les informations reçues seront stockées dans une base de données et l'utilisateur sera connecté au site (*enregistrer.php*). Pour ce faire, un message indiquant que les données ont été enregistrées suffira. Auparavant, les données seront préparées: le mot de passe encrypté, le nom transformé en majuscules, les blancs inutiles détruits… Dans le cas d'une connexion immédiate (*connexion.php*), un message annoncera que l'utilisateur est bien connecté au site.

Toutes les pages auront le même aspect général grâce aux fichiers inclus. Les opérations de connexion à la BD, de vérification des mots de passe, d'enregistrement ne seront évidemment pas programmées à ce stade des connaissances développées.

3. MySQL

3.1 L'association entre MySQL et PHP

MySQL est un système de gestion de bases de données relationnelles basé sur le langage d'interrogation *SQL* (*Structured Query Language*). C'est un des derniers logiciels open source de cette catégorie apparu sur le marché (si ce terme a du sens). Développé à partir d'un autre *SGBD* portant le nom de *mSQL*, il possède de nombreuses qualités et notamment celle d'être portable, en ce sens qu'il s'exécute sur à peu près tous les systèmes d'exploitation et tous les types de matériel.

Comme tous les *SGBD*, *MySQL* permet de créer et de gérer des bases de données. Dans le contexte qui nous intéresse, un langage comme *PHP* est capable de prendre en compte les commandes de *MySQL* et donc d'interroger des bases de données créées avec ce *SGBD*. La communication de l'un à l'autre se fait de manière souple. Nous en donnons ci-dessous une illustration.

Voici une requête telle qu'on peut l'effectuer sous *MySQL*:

```
SELECT upper(nom),prenom,login FROM inscrits WHERE choix='C' ORDER BY nom;
```

Cette requête sélectionne les valeurs des colonnes[22] *nom* et *prenom* dans une table s'appelant *inscrits*, pour autant que la colonne *choix* contienne la valeur *C*, et en ordonne les résultats par ordre alphabétique des valeurs trouvées dans la colonne *nom*. *MySQL* produira un résultat sous la forme d'un affichage en mettant la valeur de la colonne *nom* en majuscules.

```
mysql> select upper(nom),prenom from inscrits where choix='C' order by nom;
+------------+-------------+
| upper(nom) | prenom      |
+------------+-------------+
| BOURGEOIS  | Pierre      |
| CHEFFERT   | Jean-Luc    |
| CORNILLY   | Michel      |
| DAVE       | Christine   |
| DELAITTE   | Jean        |
| DI STEFANO | corrado     |
| GOFFIN     | Céline      |
| GUILLAUME  | Damien      |
| HAINE      | Yvan        |
| JANNE      | Bernard     |
| KEMPENER   | Yves        |
| LAMBOTTE   | Marie-Agnès |
| LEDENT     | Maryse      |
| LÉNELLE    | Philippe    |
| LOZET      | Jean-Luc    |
| PIEDBOEUF  | Sandrine    |
| TASSIN     | Monique     |
| VERHAEGHE  | Michel      |
| VINCENT    | Guy         |
+------------+-------------+
19 rows in set (0.00 sec)

mysql>
```

La même requête, effectuée via un script *PHP* s'écrira:

[22] Les notions de *tables* et de *colonnes* sont les termes propres à utiliser. Ils sont définis dans la suite du chapitre.

```
$sqlquery="SELECT upper(nom),prenom FROM inscrits WHERE choix='C' ORDER BY nom";
$queryresult=mysql_query($sqlquery) or die ("<p>La requête a échoué.</p>");
```

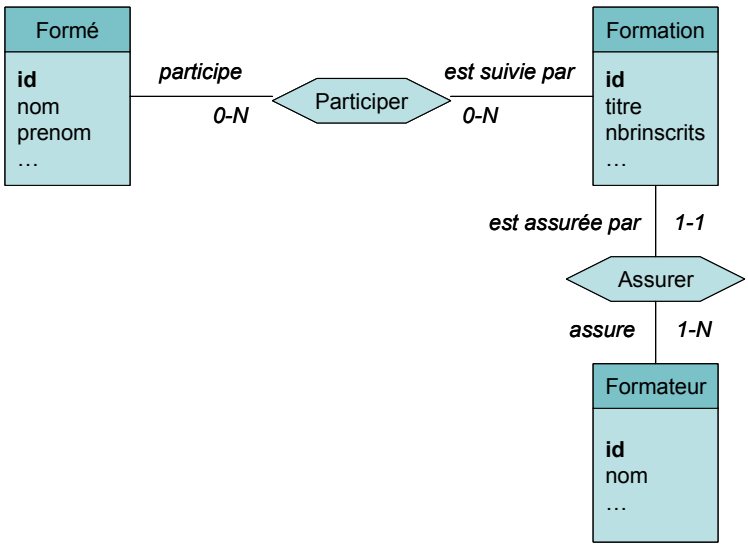
Comme on peut le constater, la requête est fournie sous forme d'une chaîne de caractères (stockée préalablement, ici, dans une variable `$sqlquery`). Syntaxiquement parlant, le point-virgule, nécessaire en *MySQL*, est omis dans la chaîne de caractères. Un point-virgule termine toutefois l'instruction *PHP*. La fonction *mysql_query* prend cette chaîne de caractère en argument et renvoie le résultat de la requête dans une variable tableau (`$queryresult` dans cet exemple), dont le script pourra exploiter le contenu à sa convenance.

3.2 La conception d'une base de données

La conception d'une base de données s'appuie généralement sur la réalisation d'un schéma décrivant de manière statique la réalité que l'on veut prendre en compte. Pour un de ces schémas que nous décrivons, la démarche consiste à isoler des entités et à les associer. C'est la raison pour laquelle on parle généralement de schéma entités-relations-associations ou schéma *ERA*.

La réalisation de ces schémas ne va pas de soi. Nous nous contentons ici de décrire les notions principales et leur représentation. Nous ne nous attardons pas sur les stratégies (notamment les différentes formes de normalisation des relations) et les méthodes de conception des bases de données.

3.2.1 Schémas ERA



Un schéma *ERA* identifie donc des entités. L'identification de ces entités est réalisée, soit au travers d'interviews de personnes occupées dans le domaine, soit de documents décrivant les activités. Par exemple, l'interview du directeur du *CeFIS* pourrait fournir des renseignements à propos des formations organisées, des personnes qui les organisent ou en sont responsables, des participants, etc. De cette interview, pourraient émerger des entités telles: **formation**, **formateur**, **formé**, **ouvrage** (de référence), **local**,…

En principe, la description s'arrête là ou la gestion s'arrête. Si, dans le problème de la gestion des formations du *CeFIS*, l'attribution des locaux n'est pas un problème, la notion de local ne sera pas prise en compte, ou en tous cas pas comme entité à gérer.

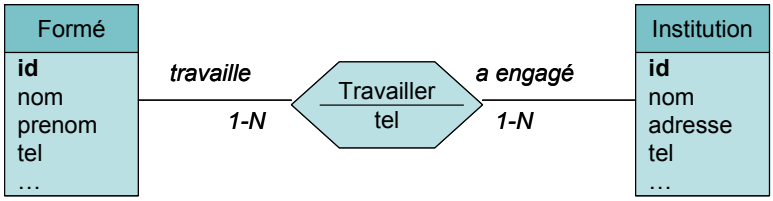
Les entités sont qualifiées par des **attributs** qui les caractérisent. Pour être identifiée, une entité doit avoir un **identifiant**. La nature d'un identifiant peut être variée. L'identifiant est souvent un simple attribut. Il est qualifié de naturel si l'information permet d'identifier naturellement l'enregistrement (le nom d'un pays, par exemple). Il est souvent construit automatiquement (numéro d'enregistrement) ou non (matricule, *ISBN*,…). Il peut être multiple (le code postal et la localité pour identifier une ville ou un village).

3.2.2 Entités, associations, attributs, rôles et cardinalités

Dans un schéma *ERA*, les **entités** sont reliées entre elles par des **associations**. Les associations peuvent être décrites par des verbes. Dans le schéma précédent, *participer*, *assurer* sont des associations.

Chaque entité possède des **attributs**. Une association peut en posséder aussi. Un attribut est une propriété qui ne doit pas être considérée comme une entité dans le problème traité. Ainsi, par exemple, le formé pourrait faire partie d'une institution. Si la base de données est conçue de manière à prendre en compte des traitements spécifiques sur les institutions, celles-ci doivent être considérées comme des entités. Sinon, l'institution peut être considérée comme un attribut du formé.

Chaque entité joue un **rôle** dans une association. Le formateur *assure* la formation, alors que la formation *est assurée par* le formateur. Le nombre d'entités pouvant jouer le même rôle est appelé **cardinalité**. Dans l'exemple qui précède, le formé est inscrit à un nombre de formations variant entre 0 et N. On pourrait s'étonner de trouver le nombre 0 et pas 1. Dans un contexte où le formé est connu du système, pour avoir participé certaines années à des formations, cela a du sens. Une formation est assurée par un et un seul formateur qui, dans l'exemple, donne au moins une formation.



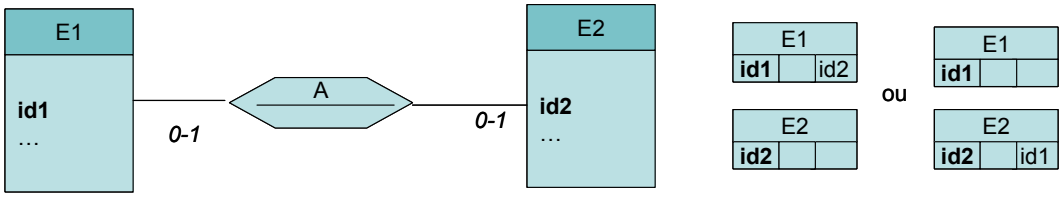
La présence d'un attribut *tel* dans chacun des éléments (entités et association) de ce schéma n'est pas contradictoire. Le téléphone du formé est son privé. Il y a celui de l'institution et celui de son lieu de travail (bureau).

3.2.3 Transformation en tables

Très concrètement, les entités et les associations qui vont subsister dans le schéma final vont être transformées en **tables**. Une table est tout simplement un ensemble de **lignes** et/ou de **colonnes**. Chaque ligne correspond à un enregistrement, chaque colonne à une information à propos de cet enregistrement. Tous les enregistrements d'une même table sont évidemment caractérisés de la même manière. À l'intersection d'une ligne et d'une colonne, on trouve une information qui, comme nous le verrons plus loin, est d'un certain type.

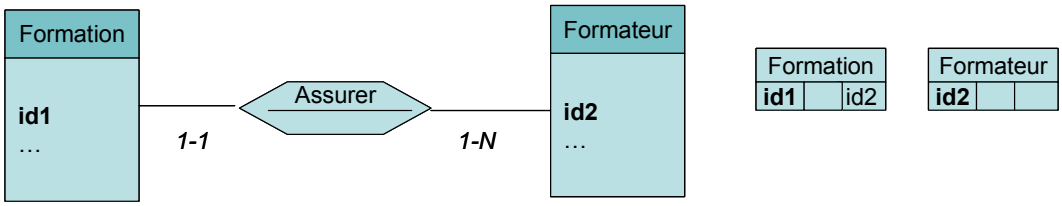
La transformation s'effectue de la manière suivante.

Pour des relations 1 à 1:



Les deux entités donnent naissance à deux tables et l'association disparaît. L'identifiant d'une des deux entités sert de lien entre les deux tables. On parle de **clé étrangère** car elle identifie les éléments d'une autre table.

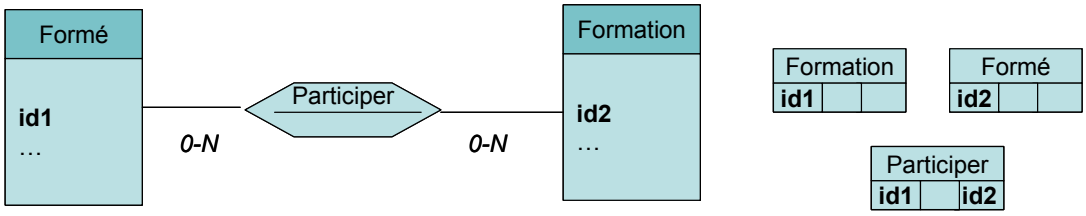
Pour des relations 1 à N:



Les deux entités donnent naissance à deux tables et l'association disparaît. L'identifiant de l'entité côté 1 sert de clé étrangère pour l'entité côté N.

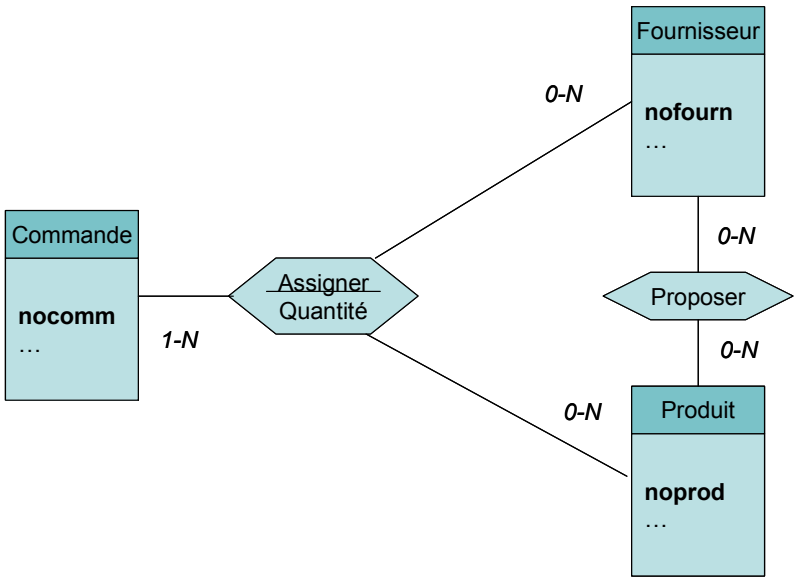
Pour des relations N à N:

Les deux entités donnent naissance à deux tables et l'association à une troisième. Les identifiants des deux entités se retrouvent dans la table correspondant à l'association.



3.2.4 Associations ternaires

Il existe des relations plus complexes que les relations binaires. En voici un exemple.



Le fournisseur propose de 0 à N produits. Un produit est proposé par 0 à N fournisseurs. Une commande est assignée à un fournisseur pour une certaine quantité d'un produit.

3.2.5 Clés étrangères

Lorsqu'un schéma *ERA* est établi, il peut donc être converti en une série de tables. Les relations entre ces tables dépendent des identifiants de chacune d'entre elles. Un identifiant d'une table présent dans une autre table est appelé clé étrangère. Ce sont les clés étrangères qui permettent de relier les informations entre elles. Les *SGBD* sont censés gérer correctement les clés étrangères et donc, notamment, les mises à jour, les insertions, les suppressions. Par exemple, si dans une table *notes* contenant les notes des étudiants, chaque enregistrement contient l'identifiant de l'étudiant *etud_id*, par exemple, la suppression d'un étudiant dans la table *etudiants* (dont l'identifiant est *etud_id*) devrait entraîner automatiquement la suppression de toutes les notes de cet étudiant. Ce n'est malheureusement pas le cas avec *MySQL*, le choix d'implémentation s'étant focalisé sur la vitesse de traitement, déterminante sur Internet. Cette gestion doit donc être prise en compte par le programmeur *PHP*.

3.3 Les tables et leurs contenus

3.3.1 Structure d'une table

Avant de préciser le contenu d'une table, il convient d'en définir la structure, à savoir le nom des colonnes, le type d'information qu'elles vont contenir et leur statut éventuel (identifiant, jamais vide, unique,…).

```
mysql> describe users;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| nom    | varchar(30) |      |     |         |                |
| prenom | varchar(20) |      |     |         |                |
| id     | tinyint(4)  |      | PRI | NULL    | auto_increment |
+--------+-------------+------+-----+---------+----------------+
3 rows in set (0.12 sec)

mysql> _
```

Ce tableau reprend le nom des colonnes, le type d'information contenue (voir plus loin pour une description plus complète), une indication qui précise si le champ peut être vide (dans l'exemple, le prénom peut l'être), une autre qui précise si le champ fait partie de l'identifiant (voir paragraphe suivant), la valeur par défaut s'il y en a une et une information supplémentaire (qui précise, dans ce cas, que la valeur du champ est incrémentée de manière automatique).

Plus de détails seront donnés par la suite à propos des différentes options possibles.

3.3.2 Types d'informations

Les types d'informations disponibles sont assez nombreux. Les types généraux sont évidemment: numérique, chaîne de caractères et date/heure mais ils se déclinent en de multiples sous-types.

3.3.2.1 Les types numériques

Les types possibles sont repris dans le tableau ci-dessous:

Nom du type	Espace requis	Signé	Non signé
TINYINT[(M)]	1 octet	-2^7 à 2^7-1	0 à 2^8-1
SMALLINT[(M)]	2 octets	-2^{15} à $2^{15}-1$	0 à $2^{16}-1$
MEDIUMINT[(M)]	3 octets	-2^{23} à $2^{23}-1$	0 à $2^{24}-1$
INT[(M)]	4 octets	-2^{31} à $2^{31}-1$	0 à $2^{32}-1$
BIGINT[(M)]	8 octets	-2^{63} à $2^{63}-1$	0 à $2^{64}-1$
Nom du type	**Espace requis**	**Minimum**	**Maximum**
FLOAT[(M,D)]	4 octets	±1,17E-38	±3,40E+38
DOUBLE[(M,D)]	8 octets	±2,23E-308	±1,80E-308
DECIMAL(M,D)	M+2 octets	texte	texte

Les valeurs M et D sont optionnelles, sauf pour le type *DECIMAL* et servent à gérer l'affichage. Par exemple, `DECIMAL(4,2)` permet de coder les nombre de -99.99 à 999,99.

C'est un attribut particulier *UNSIGNED* qui permet de déclarer les entiers signés ou non. De même, l'attribut *ZEROFILL* permet d'ajouter des zéros à l'affichage pour atteindre la valeur maximum autorisée.

3.3.2.2 Les types chaînes de caractères

Les types possibles sont repris dans le tableau ci-dessous:

Nom du type	Espace requis	Taille maximale
CHAR(M)	M octets	M octets
VARCHAR(M)	L+1 octets	M octets
TINYBLOB, TINYTEXT	L+1 octets	2^8-1 octets
BLOB, TEXT	L+2 octets	2^{16}-1 octets
MEDIUMBLOB, MEDIUMTEXT	L+3 octets	2^{24}-1 octets
LONGBLOB, LONGTEXT	L+4 octets	2^{32}-1 octets
ENUM("val1","val2",…)	1 ou 2 octets	65.535 valeurs
SET("val1","val2",…)	1, 2, 3, 4 ou 8 octets	64 valeurs

L est la longueur de la chaîne. Sa valeur est stockée dans les octets supplémentaires (1 à 4 selon cette longueur). Les valeurs de type *ENUM* sont stockées comme des nombres (leur numéro d'ordre), ce qui a pour effet de diminuer considérablement leur taille (2 octets suffisent pour encoder 65.535 valeurs). Chaque valeur d'un type *SET* est codée avec un seul bit à 1. Ainsi, s'il y a au plus 4 valeurs, elles seront codées 0001, 0010, 0100 et 1000, de sorte que si un des enregistrements prend la première et la troisième de ces valeurs, l'information codée soit 0101. Pour 64 valeurs, il faut donc au plus 8 octets.

Il faut encore noter que si une colonne est de type *VARCHAR*, toutes les autres colonnes de type *CHAR* le deviendront aussi.

3.3.2.3 Les types dates et heures

Les types possibles sont repris dans le tableau ci-dessous:

Nom du type	Espace requis	Intervalle
DATE	3 octets	"1000-01-01" à "9999-12-31"
TIME	3 octets	"-838:59:59" à "838:59:59"
DATETIME	8 octets	"1000-01-01 00:00:00" à "9999-12-31 00:00:00"
TIMESTAMP[(M)]	4 octets	19700101000000 à 2037…
YEAR[(M)]	1 octet	1901 à 2155

Comme vous pouvez le constater, les dates commencent toujours par l'année, suivie du mois et du jour. La valeur par défaut de *TIMESTAMP* correspond à la date et l'heure du serveur dans son propre fuseau horaire.

3.4 Les primitives de gestion d'une base de données

3.4.1 Accès à une BD et à ses tables

3.4.1.1 Connexion à MySQL

En ligne de commande, vous devez préciser le nom de l'hôte, le nom d'utilisateur et le mot de passe.

```
C:\Program Files\EasyPHP1-7\mysql\bin > mysql -h localhost -u root -p
```

Si le serveur se trouve sur la machine locale, le premier paramètre n'est pas obligatoire. Par défaut, le seul utilisateur reconnu par *mysql* est *root* et il ne faut pas fournir de mot de passe. L'instruction suivante convient donc:

```
C:\Program Files\EasyPHP1-7\mysql\bin> mysql -u root
```

ou encore

```
C:\Program Files\EasyPHP1-7\mysql\bin> mysql -uroot
```

Une des premières démarches à effectuer, si *MySQL* n'est pas installé localement, c'est de le sécuriser. L'application *PHPMyAdmin* dont il est question dans le paragraphe suivant, permet d'effectuer cette opération en remplaçant, par exemple, l'utilisateur *root* par un utilisateur administrateur qui devra fournir un mot de passe. L'administrateur pourra alors définir d'autres utilisateurs et décider finement de leurs privilèges.

Les opérations dont il vient d'être question sont toutefois à réaliser avec prudence pour éviter de perdre l'accès aux serveurs. Il est conseillé de rechercher un peu de documentation sur Internet à ce propos[23], avant de se lancer dans l'aventure. Toutefois, la désinstallation et la réinstallation de *EasyPHP* sont rapides, ce qui relativise le danger.

3.4.1.2 Déconnexion de MySQL

Pour se déconnecter de *MySQL*, on utilise la commande *quit*.

```
mysql> quit
```

Il faut noter que les commandes sont insensibles à la casse, sauf en ce qui concerne le nom des bases de données et les tables.

3.4.1.3 Informations sur les BD existantes

On peut obtenir la liste des bases de données par la commande

```
mysql> show databases;
```

```
mysql> show databases;
+------------+
| Database   |
+------------+
| basedetest |
| exemple    |
| mysql      |
+------------+
3 rows in set (0.00 sec)

mysql> _
```

Toutes les commandes fournies en ligne de commande doivent se terminer par le point-virgule, une instruction pouvant s'étendre sur plusieurs lignes.

Il convient alors de sélectionner une base de données pour y travailler. Cette opération est réalisée grâce à la commande *use*.

```
mysql> use exemple;
Database changed
mysql>
```

```
mysql> use exemple;
```

La base de données est alors accessible et vous pouvez vous inquiéter des

[23] …comme aux adresses http://www.wampserver.com/faq4.php ou http://linuxeduquebec.org/article.php3?id_article=4

tables qui la composent en vous servant de la commande *show tables*.

```
mysql> show tables;
```

```
mysql> show tables;
+-------------------+
| Tables_in_exemple |
+-------------------+
| users             |
+-------------------+
1 row in set (0.00 sec)

mysql>
```

La liste des tables constituant la base de données est affichée. Dans l'exemple ci-contre, on constate qu'il n'y a qu'une seule table appelée *users* dans la base de données.

La structure d'une table (nom des colonnes, types et autres attributs) peut être obtenue grâce à la commande *describe*. Ce résultat peut également être obtenu avec la commande *show columns from*.

```
mysql> describe users;
```

```
mysql> describe users;
+----------+---------------------------------+------+-----+---------+-------+
| Field    | Type                            | Null | Key | Default | Extra |
+----------+---------------------------------+------+-----+---------+-------+
| nom      | varchar(30)                     |      |     |         |       |
| prenom   | varchar(20)                     | YES  |     | NULL    |       |
| login    | varchar(8)                      |      | PRI |         |       |
| password | varchar(20)                     |      | PRI |         |       |
| statut   | set('chercheur','professeur') |      |     |         |       |
+----------+---------------------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql>
```

La commande *select database()* permet de connaître la base de données active.

```
mysql> select database();
```

```
mysql> select database();
+------------+
| database() |
+------------+
| exemple    |
+------------+
1 row in set (0.00 sec)

mysql> _
```

Voilà donc un aperçu des commandes qui permettent d'obtenir les renseignements souhaités à propos des bases de données existantes. Il n'est évidemment pas encore question, à ce stade, d'interroger les bases de données, d'effectuer des requêtes sur le contenu.

3.4.2 Création et gestion du contenu d'une BD

3.4.2.1 Création d'une base de données

La création d'une base de données implique évidemment que celle-ci ait été complètement décrite au niveau conceptuel (schéma *ERA*) et au niveau logique (transformation en tables). Elle commence par l'instruction *create database*.

```
mysql> create database formations;
```

Cette commande devra être suivie de la création des différentes tables: définition de leur structure (définition des colonnes) et encodage de leurs enregistrements (lignes).

La règle d'attribution des noms permet d'utiliser les lettres, les chiffres et les caractères `$` (déconseillé, on s'en doute, à cause de PHP) et `_`. Un nom ne peut se composer uniquement de chiffres.

3.4.2.2 Suppression d'une base de données

La suppression d'une base de données se fera par l'instruction *drop*.

```
mysql> drop database formations;
```

Il est clair que la suppression d'une base de données entraîne la suppression de toutes les tables et de tous les enregistrements qu'elle contient.

Voici pour suivre, une séquence qui illustre ces deux commandes ainsi que leurs effets.

```
mysql> show databases;
+------------+
| Database   |
+------------+
| basedetest |
| exemple    |
| mysql      |
+------------+
3 rows in set (0.00 sec)

mysql> create database formations;
Query OK, 1 row affected (0.01 sec)

mysql> show databases;
+------------+
| Database   |
+------------+
| basedetest |
| exemple    |
| formations |
| mysql      |
+------------+
4 rows in set (0.00 sec)

mysql> drop database formations;
Query OK, 0 rows affected (0.00 sec)

mysql> show databases;
+------------+
| Database   |
+------------+
| basedetest |
| exemple    |
| mysql      |
+------------+
3 rows in set (0.00 sec)

mysql> _
```

3.4.2.3 Création d'une table

Une table est créée au moyen de l'instruction *create table*. Cette instruction qui permet de nommer la table s'accompagne de la description des colonnes, de leur type et de leurs attributs éventuels. Elle s'écrit généralement sur plusieurs lignes pour plus de lisibilité[24].

```
mysql> create table users
    -> (
    ->   login varchar(8) not null primary key,
    ->   password varchar(20) not null,
    ->   nom varchar(30) not null,
    ->   prenom varchar(20),
    ->   statut set('chercheur','professeur')
    -> );
Query OK, 0 rows affected (0.65 sec)

mysql> describe users;
+----------+---------------------------------+------+-----+---------+-------+
| Field    | Type                            | Null | Key | Default | Extra |
+----------+---------------------------------+------+-----+---------+-------+
| login    | varchar(8)                      |      | PRI |         |       |
| password | varchar(20)                     |      |     |         |       |
| nom      | varchar(30)                     |      |     |         |       |
| prenom   | varchar(20)                     | YES  |     | NULL    |       |
| statut   | set('chercheur','professeur')   | YES  |     | NULL    |       |
+----------+---------------------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> _
```

[24] La plupart des instructions nécessitent une syntaxe (trop) rigoureuse. C'est pourquoi il existe des applications permettant d'en générer le code via des interfaces graphiques. Nous en décrivons une plus loin dans ce chapitre.

La colonne *login* est une colonne identifiante (*primary key*). Les colonnes *prenom* et *statut* peuvent ne pas contenir de valeurs contrairement aux colonnes *login*, *password* et *nom*.

Il est possible de garnir rapidement une table pour autant que ses données se trouvent dans un fichier, grâce à l'instruction *load data*. Par exemple, il est possible de créer un fichier texte à partir d'*Excel* et de le récupérer.

	A	B	C	D	E
1	cdu	aaaaaa	Duchateau	Charles	professeur
2	mco	bbbbbb	Colinet		
3	eva	cccccc	Vandeput	Etienne	chercheur
4					
5					

Ce fichier étant enregistré au format texte sous le nom *membres.txt* et sauvegardé dans le même répertoire que la base de données *exemples*, on aura:

```
mysql> load data infile 'membres.txt' into table users(login,password,nom,prenom
,statut);
Query OK, 3 rows affected (0.00 sec)
Enregistrements: 3  Effacús: 0  Non traitús: 0  Avertissements: 0

mysql> select * from users;
+-------+----------+-----------+---------+-------------+
| login | password | nom       | prenom  | statut      |
+-------+----------+-----------+---------+-------------+
| cdu   | aaaaaa   | Duchateau | Charles | professeur  |
| mco   | bbbbbb   | Colinet   |         |             |
| eva   | cccccc   | Vandeput  | Etienne | chercheur   |
+-------+----------+-----------+---------+-------------+
3 rows in set (0.00 sec)

mysql> describe users;
+----------+---------------------------------+------+-----+---------+-------+
| Field    | Type                            | Null | Key | Default | Extra |
+----------+---------------------------------+------+-----+---------+-------+
| login    | varchar(8)                      |      | PRI |         |       |
| password | varchar(20)                     |      |     |         |       |
| nom      | varchar(30)                     |      |     |         |       |
| prenom   | varchar(20)                     | YES  |     | NULL    |       |
| statut   | set('chercheur','professeur')   | YES  |     | NULL    |       |
+----------+---------------------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql>
```

3.4.2.4 Suppression d'une table

Une table est supprimée au moyen de l'instruction *drop table*.

```
mysql> drop table users;

mysql> drop table users;
Query OK, 0 rows affected (0.01 sec)

mysql> describe users;
ERROR 1146: La table 'exemple.users' n'existe pas
```

La suppression de la table *users* s'accompagne, comme le montre le résultat de la commande *describe*, de la suppression de sa structure.

3.4.2.5 Modification d'une table

Par modification d'une table, on entend évidemment une modification de sa structure. De nombreux changements peuvent être opérés. Nous en évoquons quelques-uns. C'est la commande *alter table* qui permet ces modifications. Elle est suivie du nom de(s) l'action(s) et de sa (leur) description(s). Parmi les actions, on trouve: *add*, *alter*, *drop*, *change*, *modify*, *rename as* (ou *to*),…

Si on souhaite ajouter une colonne *depuis* qui contiendra l'année d'entrée en service, on procèdera comme suit:

```
mysql> alter table users add (depuis varchar(4));
```

```
mysql> alter table users add (depuis varchar(4));
Query OK, 3 rows affected (0.71 sec)
Enregistrements: 3  Doublons: 0  Avertissements: 0

mysql> describe users;
+----------+---------------------------------+------+-----+---------+-------+
| Field    | Type                            | Null | Key | Default | Extra |
+----------+---------------------------------+------+-----+---------+-------+
| login    | varchar(8)                      |      | PRI |         |       |
| password | varchar(20)                     |      |     |         |       |
| nom      | varchar(30)                     |      |     |         |       |
| prenom   | varchar(20)                     | YES  |     | NULL    |       |
| statut   | set('chercheur','professeur')   | YES  |     | NULL    |       |
| depuis   | varchar(4)                      | YES  |     | NULL    |       |
+----------+---------------------------------+------+-----+---------+-------+
6 rows in set (0.09 sec)

mysql>
```

Pour supprimer la colonne depuis, on utilisera l'action *drop*:

```
mysql> alter table users drop depuis;
```

Avec l'action *add*, les options *first* ou *after* suivi du nom d'une colonne, permettent de positionner la nouvelle à l'endroit voulu. Par défaut, c'est naturellement à la fin.

Voici d'autres exemples:

```
mysql> alter table users add (depuis varchar(4)) after prenom;

mysql> alter table users add (depuis varchar(4), commentaires blob);
```

À titre d'illustration:

```
mysql> alter table users drop depuis;
Query OK, 3 rows affected (0.01 sec)
Enregistrements: 3  Doublons: 0  Avertissements: 0

mysql> describe users;
+----------+---------------------------------+------+-----+---------+-------+
| Field    | Type                            | Null | Key | Default | Extra |
+----------+---------------------------------+------+-----+---------+-------+
| login    | varchar(8)                      |      | PRI |         |       |
| password | varchar(20)                     |      |     |         |       |
| nom      | varchar(30)                     |      |     |         |       |
| prenom   | varchar(20)                     | YES  |     | NULL    |       |
| statut   | set('chercheur','professeur')   | YES  |     | NULL    |       |
+----------+---------------------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> alter table users add (depuis varchar(4),commentaires blob);
Query OK, 3 rows affected (0.01 sec)
Enregistrements: 3  Doublons: 0  Avertissements: 0

mysql> describe users;
+-------------+---------------------------------+------+-----+---------+-------+
| Field       | Type                            | Null | Key | Default | Extra |
+-------------+---------------------------------+------+-----+---------+-------+
| login       | varchar(8)                      |      | PRI |         |       |
| password    | varchar(20)                     |      |     |         |       |
| nom         | varchar(30)                     |      |     |         |       |
| prenom      | varchar(20)                     | YES  |     | NULL    |       |
| statut      | set('chercheur','professeur')   | YES  |     | NULL    |       |
| depuis      | varchar(4)                      | YES  |     | NULL    |       |
| commentaires | blob                           | YES  |     | NULL    |       |
+-------------+---------------------------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql>
```

Pour ajouter une clé primaire à une colonne, on utilisera l'action *add* avec l'option *primary key*. Ainsi, par exemple, si on estime que *login* seul n'est pas identifiant, mais la paire *login-password*, on commandera:

```
mysql> alter table users add (login,password) primary key;
```

Attention, *MySQL* considère qu'il n'y a qu'un seul identifiant possible. Donc, s'il en existe déjà un, il faut le supprimer avant d'en créer un autre. Pour cela, on utilise l'action *drop* et l'option *primary key*, comme dans l'exemple ci-dessous:

```
mysql> alter table users drop primary key;
Query OK, 3 rows affected (0.01 sec)
Enregistrements: 3  Doublons: 0  Avertissements: 0

mysql> describe users;
+--------------+---------------------------------+------+-----+---------+-------+
| Field        | Type                            | Null | Key | Default | Extra |
+--------------+---------------------------------+------+-----+---------+-------+
| login        | varchar(8)                      |      |     |         |       |
| password     | varchar(20)                     |      |     |         |       |
| nom          | varchar(30)                     |      |     |         |       |
| prenom       | varchar(20)                     | YES  |     | NULL    |       |
| statut       | set('chercheur','professeur')   | YES  |     | NULL    |       |
| depuis       | varchar(4)                      | YES  |     | NULL    |       |
| commentaires | blob                            | YES  |     | NULL    |       |
+--------------+---------------------------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql> alter table users add primary key (login,password);
Query OK, 3 rows affected (0.01 sec)
Enregistrements: 3  Doublons: 0  Avertissements: 0

mysql> describe users;
+--------------+---------------------------------+------+-----+---------+-------+
| Field        | Type                            | Null | Key | Default | Extra |
+--------------+---------------------------------+------+-----+---------+-------+
| login        | varchar(8)                      |      | PRI |         |       |
| password     | varchar(20)                     |      | PRI |         |       |
| nom          | varchar(30)                     |      |     |         |       |
| prenom       | varchar(20)                     | YES  |     | NULL    |       |
| statut       | set('chercheur','professeur')   | YES  |     | NULL    |       |
| depuis       | varchar(4)                      | YES  |     | NULL    |       |
| commentaires | blob                            | YES  |     | NULL    |       |
+--------------+---------------------------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql>
```

On peut aussi ajouter des index sur les colonnes. Les index sont intéressants en ce sens qu'ils accélèrent les recherches mais ils ralentissent certains autres traitements comme l'insertion d'une ligne dans une table, par exemple. Cet ajout se fait par l'option *index* suivi du nom de la ou des colonnes.

```
mysql> alter table users add index (nom, login);
```

Il existe plusieurs sortes de clés, chaque type ayant son intérêt. Vous connaissez les clés primaires et les clés d'index. Les clés *unique* sont aussi des index mais qui n'autorisent pas les valeurs multiples des colonnes sur lesquelles elles portent.

```
mysql> alter table users add unique (login);
```

ne permettra pas, lors d'une insertion, que deux enregistrements aient la même valeur pour *login*. Pour supprimer une clé *unique*, il faut utiliser l'action *drop* avec l'option *index* suivie du nom de la (des) colonne(s).

```
mysql> alter table users drop index login;
```

Les actions *change* et *modify* permettent de changer la définition d'une colonne et, éventuellement, de la repositionner. La première permet aussi de changer le nom de la colonne.

Ainsi, la commande:

```
mysql> alter table users modify depuis varchar(4) not null after prenom;
```

fournira le résultat suivant:

```
mysql> describe users;
+--------------+---------------------------------+------+-----+---------+-------+
| Field        | Type                            | Null | Key | Default | Extra |
+--------------+---------------------------------+------+-----+---------+-------+
| login        | varchar(8)                      |      | PRI |         |       |
| password     | varchar(20)                     |      | PRI |         |       |
| nom          | varchar(30)                     |      |     |         |       |
| prenom       | varchar(20)                     | YES  |     | NULL    |       |
| statut       | set('chercheur','professeur')   | YES  |     | NULL    |       |
| depuis       | varchar(4)                      | YES  |     | NULL    |       |
| commentaires | blob                            | YES  |     | NULL    |       |
+--------------+---------------------------------+------+-----+---------+-------+
7 rows in set (0.01 sec)

mysql> alter table users modify depuis varchar(4) NOT NULL after prenom;
Query OK, 4 rows affected (0.01 sec)
Enregistrements: 4  Doublons: 0  Avertissements: 0

mysql> describe users;
+--------------+---------------------------------+------+-----+---------+-------+
| Field        | Type                            | Null | Key | Default | Extra |
+--------------+---------------------------------+------+-----+---------+-------+
| login        | varchar(8)                      |      | PRI |         |       |
| password     | varchar(20)                     |      | PRI |         |       |
| nom          | varchar(30)                     |      |     |         |       |
| prenom       | varchar(20)                     | YES  |     | NULL    |       |
| depuis       | varchar(4)                      |      |     |         |       |
| statut       | set('chercheur','professeur')   | YES  |     | NULL    |       |
| commentaires | blob                            | YES  |     | NULL    |       |
+--------------+---------------------------------+------+-----+---------+-------+
7 rows in set (0.00 sec)

mysql>
```

Pour renommer une table, on se servira de l'action *rename as*.

```
mysql> show tables;
+-----------------+
| Tables_in_exemple |
+-----------------+
| users           |
+-----------------+
1 row in set (0.00 sec)

mysql> alter table users rename as utilisateurs;
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+-----------------+
| Tables_in_exemple |
+-----------------+
| utilisateurs    |
+-----------------+
1 row in set (0.00 sec)

mysql>
```

3.4.2.6 Copier des tables

On peut envisager la copie de tables de deux manières:

- une copie intégrale ou partielle de la structure et des données
- une copie de la structure

Les deux se réalisent au moyen de la commande *create table* combinée avec une simple commande de sélection. Les commandes de sélection font d'ailleurs l'objet de la section suivante.

Pour une copie intégrale:

```
mysql> create table util2 select * from utilisateurs;
```

Pour une copie de la structure:

```
mysql> create table util2 select * from utilisateurs where 1=0;
```

Cette commande donne déjà quelques indications sur les attributs de la commande *select*. L'attribut *where* est suivi d'une condition à vérifier par les enregistrements à prendre en compte. Comme 1 n'est jamais égal à 0, aucun enregistrement n'est repris dans la nouvelle table qui a toutefois la même structure que la table originale.

3.4.2.7 Ajouter, supprimer, mettre à jour des lignes

Pour ajouter une ligne à une table, on utilise la commande *insert*. Il y a différentes manières d'insérer une ligne et notamment, en précisant:

- une liste de colonnes et une liste de valeurs

```
mysql> insert users (login,password,nom) values(ldo,dddddd,Doumont);
```

Observez que les colonnes ne sont pas citées dans l'ordre qu'elles occupent dans la structure.

```
mysql> select * from utilisateurs;
+-------+----------+-----------+---------+--------+------------+--------------+
| login | password | nom       | prenom  | depuis | statut     | commentaires |
+-------+----------+-----------+---------+--------+------------+--------------+
| cdu   | aaaaaa   | Duchateau | Charles |        | professeur | NULL         |
| mco   | bbbbbb   | Colinet   |         |        |            | NULL         |
| eva   | cccccc   | Vandeput  | Etienne | 1993   | chercheur  | NULL         |
+-------+----------+-----------+---------+--------+------------+--------------+
3 rows in set (0.00 sec)

mysql> insert utilisateurs (nom,login,password) values ('Doumont','ldo','dddddd
);
Query OK, 1 row affected (0.00 sec)

mysql> select * from utilisateurs;
+-------+----------+-----------+---------+--------+------------+--------------+
| login | password | nom       | prenom  | depuis | statut     | commentaires |
+-------+----------+-----------+---------+--------+------------+--------------+
| cdu   | aaaaaa   | Duchateau | Charles |        | professeur | NULL         |
| mco   | bbbbbb   | Colinet   |         |        |            | NULL         |
| eva   | cccccc   | Vandeput  | Etienne | 1993   | chercheur  | NULL         |
| ldo   | dddddd   | Doumont   | NULL    |        | NULL       | NULL         |
+-------+----------+-----------+---------+--------+------------+--------------+
4 rows in set (0.01 sec)

mysql>
```

- une liste de paires colonne-valeur

```
mysql> insert utilisateurs set (nom='Donnay',login='jdo',password='eeeeee');
```

Cette commande ne permet d'insérer qu'une seule ligne à la fois.

- une sélection, comme dans l'exemple ci-dessus

```
mysql> insert utilisateurs select…
```

Cette instruction permet d'insérer plusieurs lignes à la fois mais demande l'existence d'une table de laquelle sont extraites les lignes à insérer avec les contraintes que cela suppose (cohérence des noms, des types et autres attributs de colonnes).

Pour supprimer des lignes, on utilise la commande *delete from*. Cette commande doit être agrémentée d'une condition semblable à celles que l'on trouve souvent dans les commandes *select* et qui précise quels sont les lignes qui doivent disparaître de la table.

```
mysql> delete from utilisateurs where isnull(statut);
```

La manière d'écrire les conditions s'apparente à celles que l'on trouve dans la plupart des langages de programmation. Les expressions de ces conditions font appel à des variables (dont les noms sont ceux des colonnes), des opérateurs, des fonctions, des constantes.

En voici une illustration:

```
mysql> select * from utilisateurs;
+-------+----------+-----------+---------+--------+------------+--------------+
| login | password | nom       | prenom  | depuis | statut     | commentaires |
+-------+----------+-----------+---------+--------+------------+--------------+
| cdu   | aaaaaa   | Duchateau | Charles |        | professeur | NULL         |
| mco   | bbbbbb   | Colinet   |         |        |            | NULL         |
| eva   | cccccc   | Vandeput  | Etienne | 1993   | chercheur  | NULL         |
| jdo   | eeeeee   | Donnay    | NULL    |        | NULL       | NULL         |
| ldo   | dddddd   | Doumont   | NULL    |        | NULL       | NULL         |
+-------+----------+-----------+---------+--------+------------+--------------+
5 rows in set (0.01 sec)

mysql> delete from utilisateurs where ISNULL(statut);
Query OK, 2 rows affected (0.00 sec)

mysql> select * from utilisateurs;
+-------+----------+-----------+---------+--------+------------+--------------+
| login | password | nom       | prenom  | depuis | statut     | commentaires |
+-------+----------+-----------+---------+--------+------------+--------------+
| cdu   | aaaaaa   | Duchateau | Charles |        | professeur | NULL         |
| mco   | bbbbbb   | Colinet   |         |        |            | NULL         |
| eva   | cccccc   | Vandeput  | Etienne | 1993   | chercheur  | NULL         |
+-------+----------+-----------+---------+--------+------------+--------------+
3 rows in set (0.00 sec)

mysql>
```

Dans l'expression de la condition utilisée dans cet exemple, *statut* joue le rôle de variable et *isnull* est une fonction qui renvoie une valeur booléenne.

Un attribut *limit* peut être utilisé pour limiter le nombre de lignes supprimées.

```
mysql> delete from utilisateurs where isnull(statut) limit 1;
```

n'aurait supprimé qu'une seule des deux lignes.

La mise à jour des informations contenues dans les lignes s'effectue au moyen de la commande *update*. Cette commande possède aussi des attributs *where* et *limit*. L'attribut *set* permet de préciser quelles colonnes doivent être mises à jour et de quelle manière.

La commande

```
mysql> update utilisateurs set (login=concat('c-',login);
```

aura pour effet d'ajouter un c et un tiret devant les *login* de tous les utilisateurs.

```
mysql> select * from utilisateurs;
+-------+----------+-----------+---------+--------+------------+--------------+
| login | password | nom       | prenom  | depuis | statut     | commentaires |
+-------+----------+-----------+---------+--------+------------+--------------+
| cdu   | aaaaaa   | Duchateau | Charles |        | professeur | NULL         |
| mco   | bbbbbb   | Colinet   |         |        |            | NULL         |
| eva   | cccccc   | Vandeput  | Etienne | 1993   | chercheur  | NULL         |
+-------+----------+-----------+---------+--------+------------+--------------+
3 rows in set (0.00 sec)

mysql> update utilisateurs set login=concat('c-',login);
Query OK, 3 rows affected (0.00 sec)
Enregistrements correspondants: 3  Modifiús: 3  Warnings: 0

mysql> select * from utilisateurs;
+-------+----------+-----------+---------+--------+------------+--------------+
| login | password | nom       | prenom  | depuis | statut     | commentaires |
+-------+----------+-----------+---------+--------+------------+--------------+
| c-cdu | aaaaaa   | Duchateau | Charles |        | professeur | NULL         |
| c-mco | bbbbbb   | Colinet   |         |        |            | NULL         |
| c-eva | cccccc   | Vandeput  | Etienne | 1993   | chercheur  | NULL         |
+-------+----------+-----------+---------+--------+------------+--------------+
3 rows in set (0.01 sec)

mysql>
```

Dans l'expression qui décrit ce que doit devenir la colonne *login*, on retrouve une variable (*login*) une fonction (*concat*) et une constante (la chaîne *'c-'*).

3.4.3 Gestion d'une BD avec PHPMyAdmin

Comme il a été dit par ailleurs, l'écriture des instructions en *MySQL* est souvent fastidieuse. Pour cette raison, il existe des générateurs de code qui permettent de gérer les bases de données via des interfaces graphiques. Toutefois, la connaissance de la syntaxe d'un tel langage reste un atout pour la résolution des problèmes qui peuvent surgir lors de la gestion d'une base de données.

EasyPHP comprend une application, *PHPMyAdmin*, qui offre la possibilité de gérer ses bases de données. Elle est accessible à partir du menu contextuel du programme *EasyPHP*. Si son interface mérite d'être améliorée, elle permet néanmoins d'éviter les problèmes de syntaxe. Voici une illustration d'un des écrans de cette interface:

Sans entrer dans le détail de cette application, précisons que:

- l'onglet *Structure* permet, comme on s'en doute, de modifier la structure des tables d'une base données;
- l'onglet *Afficher* permet non seulement de visualiser les lignes d'une table, mais aussi de les éditer facilement;
- l'onglet *SQL* permet de formuler une requête qu'il n'est pas possible de formuler par l'intermédiaire de l'interface;
- l'onglet *Sélectionner* permet de construire des requêtes (comme il en sera question dans la section suivante);

- l'onglet *Insérer* offre des facilités de génération des instructions permettant d'insérer des lignes dans les tables.

Les autres options sont moins importantes à ce stade.

Comme dans toutes les interfaces de ce type, la connaissance de la syntaxe est évidemment très éclairante quand à la signification des options qu'elles proposent.

3.4.4 Sélection d'informations dans une BD

3.4.4.1 Les requêtes

Toutes les primitives dont nous venons de parler ne servent qu'à créer, ou à utiliser pour mettre à jour, des bases de données et les tables qu'elles sont censées contenir. Une tout autre activité consiste à interroger les bases de données pour en tirer un certain nombre d'informations. On parle généralement de requête à formuler. Le résultat d'une requête se présente aussi sous forme d'une table. Dès lors, rien n'empêche que celle-ci soit stockée comme les autres dans une base de données.

Nous analysons, dans cette section, la commande *select* qui est à la base de la formulation des requêtes de ce type. Pour pouvoir effectuer des requêtes fines, la commande *select* est flanquée de nombreuses options. Nous présentons d'abord les plus importantes d'entre elles et nous reviendrons, à la fin de cette section, sur une syntaxe plus complète de cette commande.

3.4.4.2 Sélectionner

Quand on pense à sélectionner, on imagine extraire de l'information d'une ou plusieurs tables constituant une base de données. La commande *select* qui permet de réaliser ce souhait est toutefois plus générale. Il n'est donc pas absolument nécessaire de préciser un (des) nom(s) de table(s). L'écriture d'une simple expression suffit. En voici quelques illustrations:

```
mysql> select 45/12;
+-------+
| 45/12 |
+-------+
|  3.75 |
+-------+
1 row in set (0.00 sec)

mysql> _
```

```
mysql> select version();
+------------------+
| version()        |
+------------------+
| 4.0.15-max-debug |
+------------------+
1 row in set (0.00 sec)

mysql>
```

```
mysql> select 11/2=5.5;
+-----------+
| 11/2=5.5 |
+-----------+
|         1 |
+-----------+
1 row in set (0.00 sec)

mysql>
```

3.4.4.3 Sélectionner des colonnes

Quand on parle de sélection dans une seule table, il ne peut être question que de colonnes, de lignes ou d'un mélange des deux. On peut donc préciser quelles sont les colonnes dont on veut sélectionner les informations. La sélection la plus simple est celle qui consiste à sélectionner toutes les colonnes.

```
mysql> select * from utilisateurs;
```

fera afficher toutes les colonnes de la table *utilisateurs*.

```
mysql> select * from utilisateurs;
+-------+----------+-----------+---------+--------+----------------------+
| login | password | nom       | prenom  | depuis | statut               |
+-------+----------+-----------+---------+--------+----------------------+
| cdu   | aaaaaa   | Duchateau | Charles | NULL   | professeur,formateur |
| mco   | bbbbbb   | Colinet   | Monique | NULL   | formateur            |
| eva   | cccccc   | Vandeput  | Etienne | NULL   | chercheur,formateur  |
+-------+----------+-----------+---------+--------+----------------------+
3 rows in set (0.00 sec)

mysql>
```

Toutes les commandes de sélection demandent de préciser la ou les tables concernées par celle-ci. Le mot clé qui l(es)'introduit est *from*.

Dans le cas où les informations de toutes les colonnes ne sont pas souhaitées, il convient de préciser celles qui sont concernées.

```
mysql> select nom, prenom, login from utilisateurs;
```

ne fera afficher que les informations des colonnes *nom*, *prenom* et *login*.

```
mysql> select nom,prenom,login from utilisateurs;
+-----------+---------+-------+
| nom       | prenom  | login |
+-----------+---------+-------+
| Duchateau | Charles | cdu   |
| Colinet   | Monique | mco   |
| Vandeput  | Etienne | eva   |
+-----------+---------+-------+
3 rows in set (0.62 sec)

mysql>
```

La sélection peut être accompagnée d'un traitement des données. Ainsi, on peut souhaiter voir les noms apparaître en majuscules.

```
mysql> select upper(nom),prenom from utilisateurs;
```

C'est le rôle de la fonction *upper* de transformer les caractères en majuscules.

```
mysql> select upper(nom),prenom from utilisateurs;
+------------+---------+
| upper(nom) | prenom  |
+------------+---------+
| DUCHATEAU  | Charles |
| COLINET    | Monique |
| VANDEPUT   | Etienne |
+------------+---------+
3 rows in set (0.65 sec)

mysql>
```

Les résultats ne sont donc pas nécessairement composés que d'informations issues de la base de données, mais aussi de traitements effectués sur ces informations comme dans...

```
mysql> select concat(left(prenom,1),'. '),nom,login from utilisateurs;
+-----------------------------+-----------+-------+
| concat(left(prenom,1),'. ') | nom       | login |
+-----------------------------+-----------+-------+
| C.                          | Duchateau | cdu   |
| M.                          | Colinet   | mco   |
| E.                          | Vandeput  | eva   |
+-----------------------------+-----------+-------+
3 rows in set (0.00 sec)

mysql> _
```

…ou mieux,

```
mysql> select concat(nom,' ',prenom,' login: ',login) from utilisateurs;
+---------------------------------------+
| concat(nom,' ',prenom,' login: ',login) |
+---------------------------------------+
| Duchateau Charles login: cdu          |
| Colinet Monique login: mco            |
| Vandeput Etienne login: eva           |
+---------------------------------------+
3 rows in set (0.00 sec)

mysql>
```

3.4.4.4 Sélectionner des lignes

La sélection des lignes à partir d'une (de plusieurs) table(s) s'effectue en définissant une clause à l'aide du mot-clé *where*. La clause étant une expression booléenne, son écriture est, à nouveau, un assemblage de variables (colonnes), constantes, opérateurs et fonctions. Parmi les opérateurs et les fonctions, on retrouve les grands classiques présents dans la plupart des langages. Nous n'allons pas les énumérer ici[25]. Nous en mentionnons quelques-uns qui sont plus spécifiques à un *SGBD* à travers quelques exemples.

```
mysql> select * from utilisateurs where prenom in ('André','Charles','Marc','Mon
ique');
+-------+----------+-----------+---------+--------+----------------------+
| login | password | nom       | prenom  | depuis | statut               |
+-------+----------+-----------+---------+--------+----------------------+
| cdu   | aaaaaa   | Duchateau | Charles | 1981   | professeur,formateur |
| mco   | bbbbbb   | Colinet   | Monique | 1998   | formateur            |
+-------+----------+-----------+---------+--------+----------------------+
2 rows in set (0.00 sec)

mysql>
```

L'opérateur *in* permet de rechercher une valeur dans une liste. Il est évidemment possible de combiner une sélection de lignes avec une sélection de colonnes.

```
mysql> select * from utilisateurs;
+-------+----------+-----------+---------+--------+----------------------+
| login | password | nom       | prenom  | depuis | statut               |
+-------+----------+-----------+---------+--------+----------------------+
| cdu   | aaaaaa   | Duchateau | Charles | 1981   | professeur,formateur |
| mco   | bbbbbb   | Colinet   | Monique | 1998   | formateur            |
| eva   | cccccc   | Vandeput  | Etienne | 1993   | chercheur,formateur  |
+-------+----------+-----------+---------+--------+----------------------+
3 rows in set (0.00 sec)

mysql> select upper(nom) from utilisateurs where depuis between 1990 and 2000;
+-------------+
| upper(nom)  |
+-------------+
| COLINET     |
| VANDEPUT    |
+-------------+
2 rows in set (0.61 sec)

mysql>
```

Dans l'exemple qui précède, l'opérateur *between* permet de rechercher une valeur dans un intervalle.

```
mysql> select * from utilisateurs;
+-------+----------+-----------+---------+--------+----------------------+
| login | password | nom       | prenom  | depuis | statut               |
+-------+----------+-----------+---------+--------+----------------------+
| cdu   | aaaaaa   | Duchateau | Charles | 1981   | professeur,formateur |
| mco   | bbbbbb   | Colinet   | Monique | 1998   | formateur            |
| eva   | cccccc   | Vandeput  | Etienne | 1993   | chercheur,formateur  |
| ldo   | dddddd   | Doumont   | Ludovic | 2004   | NULL                 |
+-------+----------+-----------+---------+--------+----------------------+
4 rows in set (0.00 sec)

mysql> select upper(nom) from utilisateurs where nom like 'D%';
+-------------+
| upper(nom)  |
+-------------+
| DUCHATEAU   |
| DOUMONT     |
+-------------+
2 rows in set (0.00 sec)

mysql> _
```

[25] Vous pourrez trouver une documentation assez complète des opérateurs et des fonctions disponibles à l'adresse http://dev.mysql.com/doc/mysql/fr/.

Comme on peut le voir, l'opérateur *like* autorise des sélections utilisant des caractères génériques.

De nombreux autres opérateurs existent comme *not like*, *not between…and*, *not in*, *is null* ou encore *is not null* en dehors des opérateurs classiques arithmétiques, logiques et de comparaison.

3.4.4.5 Opérer des classements

L'option *order by* et son attribut *asc* ou *desc*, permettent de présenter les informations dans un ordre choisi pour les éléments d'une colonne.

```
mysql> select prenom,upper(nom),login order by prenom asc;
```

Les lignes sont ici présentées dans l'ordre (alphabétique) ascendant des prénoms.

```
mysql> select prenom,upper(nom),login from utilisateurs order by prenom asc;
+---------+-------------+-------+
| prenom  | upper(nom)  | login |
+---------+-------------+-------+
| Charles | DUCHATEAU   | cdu   |
| Etienne | VANDEPUT    | eva   |
| Ludovic | DOUMONT     | ldo   |
| Monique | COLINET     | mco   |
+---------+-------------+-------+
4 rows in set (0.68 sec)

mysql> select prenom,upper(nom) from utilisateurs order by depuis desc;
+---------+-------------+
| prenom  | upper(nom)  |
+---------+-------------+
| Ludovic | DOUMONT     |
| Monique | COLINET     |
| Etienne | VANDEPUT    |
| Charles | DUCHATEAU   |
+---------+-------------+
4 rows in set (0.00 sec)

mysql>
```

3.4.4.6 Opérer des regroupements

Outre le fait de trier les lignes, on peut souhaiter que ces dernières subissent un certain regroupement. Ce regroupement doit évidemment précéder le tri s'il y en a un. La clause utilisée dans ce cas est la clause *group by*. Cette clause permet de regrouper les lignes selon les valeurs d'une ou de plusieurs colonnes. Attention, même s'il n'y a qu'une seule colonne concernée par la clause, il ne s'agit pas d'un tri, mais bien d'un regroupement.

Pour illustrer ces regroupements, il convient de travailler avec une base de données un peu plus élaborée. C'est ce que nous faisons, après avoir précisé deux nouveaux opérateurs utilisables dans la commande *select*.

La clause *group by* peut être accompagnée d'une autre clause, *having*, permettant d'effectuer un filtrage supplémentaire lorsque ce regroupement est effectué. Des exemples suivront.

3.4.4.7 Limiter le nombre des résultats

On peut ne pas souhaiter obtenir tous les résultats, mais seulement un certain nombre ou même, quelquefois, un seul. L'attribut *limit* permet de fixer ce nombre.

Ainsi, si on ne s'intéresse qu'à la première ligne vérifiant un critère donné, on peut procéder comme dans l'exemple suivant.

```
mysql> select prenom,upper(nom) where depuis>'1990' order by nom limit 1;
```

La ligne retournée sera celle de la première personne, par ordre alphabétique (des noms), qui a été engagée après 1990.

```
mysql> select prenom,upper(nom) from utilisateurs where depuis>'1990' order by n
om limit 1;
+---------+------------+
| prenom  | upper(nom) |
+---------+------------+
| Monique | COLINET    |
+---------+------------+
1 row in set (0.63 sec)

mysql>
```

3.4.4.8 Éliminer les doublons

Il ne s'agit pas de considérer des lignes identiques dans la table de sélection, mais des lignes identiques dans la table des résultats. On utilise pour cela l'opérateur *distinct*.

```
mysql> select distinct left(nom,1) from utilisateurs;
+--------------+
| left(nom,1) |
+--------------+
| D           |
| C           |
| U           |
+--------------+
3 rows in set (0.00 sec)

mysql> _
```

Dans cet exemple, on recherche par quelles lettres commencent les noms des utilisateurs.

3.4.4.9 Syntaxe plus complète de la commande select

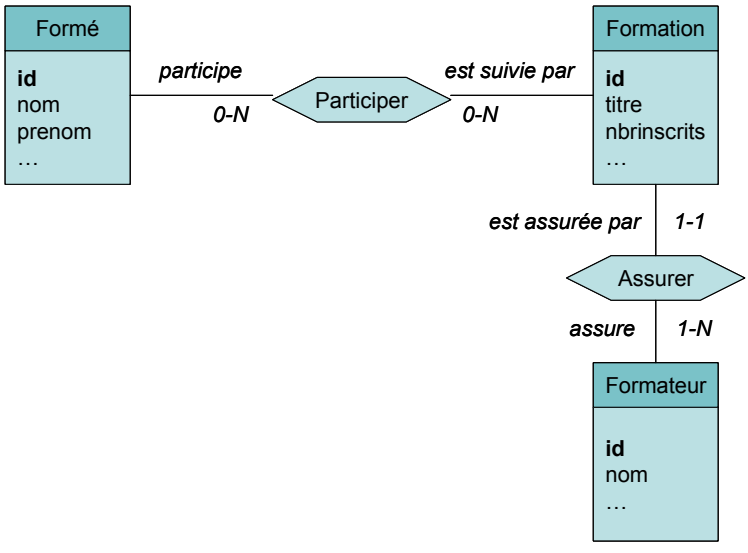
Une commande *select* peut donc s'écrire de manière relativement longue si plusieurs clauses et plusieurs opérateurs sont utilisés. Les principales clauses et les principaux opérateurs sont les suivants:

- *distinct* (opérateur)
- *from* (mot-clé)
- *where* (clause)
- *group by* (clause)
- *having* (clause)
- *order by… asc* ou *desc* (clause)
- *limit* (opérateur)

3.4.5 Sélections multi-tables

3.4.5.1 Création des tables

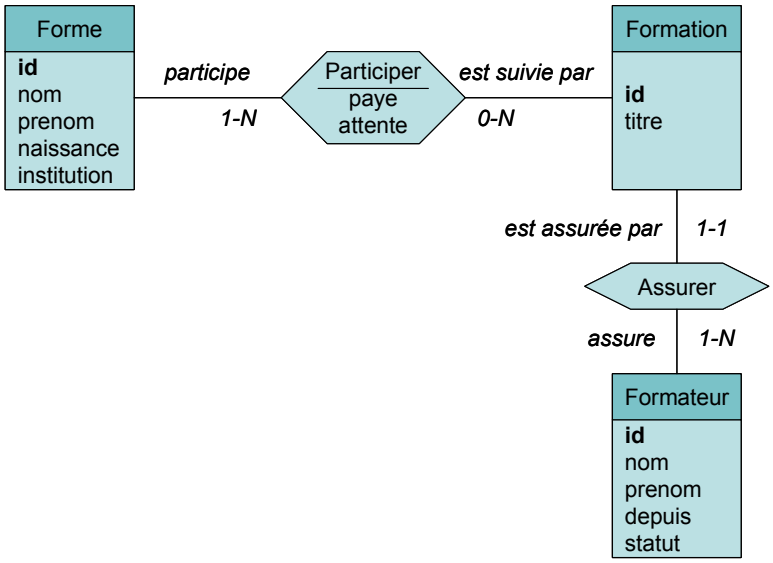
Rappelons-nous qu'une base de données est rarement composée d'une seule table, et que la transformation du schéma conceptuel aboutit nécessairement à un nombre plus ou moins élevé de tables. Pour continuer à illustrer notre propos, nous allons quelque peu modifier notre base de données et repartir d'un des schémas simples que nous avons proposés.



Ce schéma décrit que dans un centre de formation, il y a des formés qui participent à des formations assurées par des formateurs.

Tâchons de réfléchir à la transformation de ce schéma en tables.

Nous allons modifier un peu ce schéma en étoffant le nombre d'attributs de ses entités et en ajoutant un, voire deux attributs à l'association: un attribut *paye* qui précise si le formé a payé son inscription et un attribut *attente* qui précise si l'inscription est retenue ou si le formé fait partie d'une liste d'attente. Notez aussi que la table *utilisateurs* qui nous a servi d'exemple jusqu'ici sera reconvertie en table *formateurs* grâce au vertu de la commande *alter table … rename as …*



L'identificateur *id* fait office d'identifiant dans l'entité *formateur*. Il faudra donc s'assurer, par la programmation, que deux formateurs n'ont pas le même identifiant. Chaque formé participe au moins à une formation, sauf qu'il est peut-être en attente pour cette formation.

Ce schéma est simple et toutefois suffisant pour illustrer notre propos. Il est clair que nous pourrions rajouter des attributs tels l'adresse des formés, des formateurs, les prérequis d'une formation, etc. Mais ces informations supplémentaires ne vont pas nous aider à comprendre mieux ce qu'il y a à comprendre.

La première démarche consiste à transformer le schéma en table. La relation entre formé et formation est une relation de plusieurs à plusieurs. L'association *Participer* va donc donner naissance à une table reprenant les attributs de cette association, soit *paye* et *attente*, mais aussi, les identifiants de chacune des entités qu'elle associe.

La relation entre formateur et formation est une relation de un à plusieurs. En conséquence, et comme nous l'avons vu précédemment, l'association *Assurer* ne donnera pas naissance à une table.

Nous aurons donc en tout quatre tables:

- *formes* qui correspond à l'entité *Formé*
- *formations* qui correspond à l'entité *Formation*
- *inscriptions* qui correspond à l'association *Participer*
- *formateurs* qui correspond à l'entité *Formateur*

Dans la table *formes* nous avons décidé de choisir un identifiant automatiquement généré par *MySQL*. Le formé aura un nom, un prénom, une année de naissance et une institution. Si on souhaitait développer la notion d'institution, par exemple pour des raisons de contact par courrier ou autre, on en ferait certainement une entité.

```
mysql> describe formes;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| id          | tinyint(4)  |      | PRI | NULL    | auto_increment |
| nom         | varchar(30) |      |     |         |                |
| prenom      | varchar(20) |      |     |         |                |
| naissance   | year(4)     | YES  |     | NULL    |                |
| institution | varchar(50) | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
5 rows in set (0.84 sec)

mysql> _
```

La table *formations* comprend trois attributs: un identifiant construit (deux derniers chiffres de l'année et lettre de la formation), un titre et, provenant de l'association *Assurer*, l'identifiant du formateur.

```
mysql> describe formations;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| id           | char(3)     |      | PRI |         |       |
| titre        | varchar(20) |      |     |         |       |
| id_formateur | varchar(8)  |      |     |         |       |
+--------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

La table *inscriptions* se compose des identifiants de chacune des entités de l'association *Participer* et de ses attributs *paye* et *attente* qui ont été choisis de type *enum*.

```
mysql> describe inscriptions;
+--------------+--------------------+------+-----+---------+-------+
| Field        | Type               | Null | Key | Default | Extra |
+--------------+--------------------+------+-----+---------+-------+
| id_formation | char(3)            |      | PRI |         |       |
| id_forme     | varchar(4)         |      | PRI |         |       |
| paye         | enum('oui','non')  |      |     | non     |       |
| attente      | enum('oui','non')  |      |     | non     |       |
+--------------+--------------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql>
```

Enfin, la table *formateurs* est l'ancienne table *utilisateurs*. Nous avons décidé que l'identificateur était un identifiant à lui seul et non avec le mot de passe qui a d'ailleurs disparu de la structure.

```
mysql> describe formateurs;
+--------+-----------------------------------+------+-----+---------+-------+
| Field  | Type                              | Null | Key | Default | Extra |
+--------+-----------------------------------+------+-----+---------+-------+
| id     | varchar(8)                        |      | PRI |         |       |
| nom    | varchar(30)                       |      |     |         |       |
| prenom | varchar(20)                       | YES  |     | NULL    |       |
| depuis | varchar(5)                        | YES  |     | NULL    |       |
| statut | enum('chercheur','professeur')    | YES  |     | NULL    |       |
+--------+-----------------------------------+------+-----+---------+-------+
5 rows in set (0.00 sec)

mysql> _
```

Afin de bien comprendre les opérations de jointures qui font l'objet du paragraphe suivant, il importe de visualiser le contenu de ces différentes tables au moment où ces jointures sont effectuées. Les voici, avec pour commencer, la table *formes*…

```
mysql> select * from formes;
+----+-----------+----------+-----------+-----------------------+
| id | nom       | prenom   | naissance | institution           |
+----+-----------+----------+-----------+-----------------------+
|  1 | Dethy     | Pierre   |      1967 | Institut Sainte-Marie |
|  2 | Pichon    | Andrú    |      1970 | Institut Saint-Joseph |
|  3 | Delire    | Jacques  |      1976 | NULL                  |
|  4 | Courtin   | Pierre   |      1982 | NULL                  |
|  5 | Vandeuren | Isabelle |      1979 | Institut Saint-Joseph |
|  6 | Thomas    | Martine  |      1987 | Institut Sainte-Marie |
|  7 | Thiry     | Delphine |      1976 | CollÞge Saint-Pierre  |
|  8 | Maniette  | Vincent  |      NULL | CollÞge Saint-Pierre  |
|  9 | Lemoine   | Audrey   |      1983 | NULL                  |
| 10 | Cordonnier | Yvan    |      1965 | CollÞge Notre-Dame    |
+----+-----------+----------+-----------+-----------------------+
10 rows in set (0.07 sec)

mysql> _
```

…ensuite, la table *formations*…

```
mysql> select * from formations;
+-----+-----------+---------------+
| id  | titre     | id_formateur  |
+-----+-----------+---------------+
| 04A | CCM       |               |
| 04B | Flash     |               |
| 04C | PHP-MySQL |               |
| 04D | Word      |               |
+-----+-----------+---------------+
4 rows in set (0.11 sec)

mysql>
```

…la table *inscriptions*…

```
mysql> select * from inscriptions;
+--------------+----------+------+---------+
| id_formation | id_forme | paye | attente |
+--------------+----------+------+---------+
| 04C          | 2        | non  | non     |
| 04B          | 1        | non  | non     |
| 04D          | 2        | non  | non     |
| 04A          | 3        | oui  | non     |
| 04B          | 3        | oui  | non     |
| 04C          | 4        | non  | non     |
| 04A          | 5        | non  | non     |
| 04A          | 6        | non  | non     |
| 04B          | 6        | non  | non     |
| 04C          | 7        | non  | oui     |
| 04B          | 8        | non  | non     |
| 04D          | 8        | non  | non     |
| 04A          | 9        | non  | non     |
| 04C          | 9        | non  | non     |
| 04A          | 10       | oui  | non     |
| 04C          | 10       | oui  | non     |
| 04D          | 10       | oui  | non     |
+--------------+----------+------+---------+
17 rows in set (0.16 sec)

mysql>
```

…et finalement, la table *formateurs*.

```
mysql> select * from formateurs;
+-----+-----------+---------+--------+------------+
| id  | nom       | prenom  | depuis | statut     |
+-----+-----------+---------+--------+------------+
| cdu | Duchateau | Charles | 1981   | professeur |
| mco | Colinet   | Monique | 1998   |            |
| eva | Vandeput  | Etienne | 1993   | chercheur  |
| ldo | Doumont   | Ludovic | 2004   |            |
+-----+-----------+---------+--------+------------+
4 rows in set (0.00 sec)

mysql>
```

3.4.5.2 Jointures

L'opération qui consiste à fusionner, d'une manière ou d'une autre, les informations de deux ou plusieurs tables porte le nom de **jointure**. Il existe différents types de jointures et notre propos n'est pas de les décrire tous. Nous nous contenterons d'en illustrer deux.

Une des jointures les plus intéressantes est **l'équijointure**. C'est une jointure basée sur une condition d'égalité. Les noms des colonnes ne doivent pas nécessairement être identiques, comme dans les jointures dites naturelles[26], et l'égalité ne porte pas nécessairement sur des clés primaires ou étrangères.

---

[26] Les jointures naturelles se font sur base de colonnes portant les mêmes noms et ayant les mêmes caractéristiques.

```
mysql> select nom,prenom,titre from formes,formations,inscriptions where
id_formation=formations.id && id_forme=formes.id;
```

La commande qui précède va permettre de sélectionner deux colonnes dans la table *formes* et une dans la table *formations*. En réalité, elle va produire l'affichage de toutes les inscriptions prises sous la forme nom et prénom de l'inscrit et titre de la formation à laquelle il est inscrit. La jointure se fait par l'intermédiaire de l'identifiant de la table *inscriptions* qui est composé de deux clés étrangères.

```
mysql> select nom,prenom,titre from formes,formations,inscriptions where id_form
ation=formations.id && id_forme=formes.id;
+------------+----------+-----------+
| nom        | prenom   | titre     |
+------------+----------+-----------+
| Cordonnier | Yvan     | CCM       |
| Delire     | Jacques  | CCM       |
| Vandeuren  | Isabelle | CCM       |
| Thomas     | Martine  | CCM       |
| Lemoine    | Audrey   | CCM       |
| Dethy      | Pierre   | Flash     |
| Delire     | Jacques  | Flash     |
| Thomas     | Martine  | Flash     |
| Maniette   | Vincent  | Flash     |
| Cordonnier | Yvan     | PHP-MySQL |
| Pichon     | AndrÚ    | PHP-MySQL |
| Courtin    | Pierre   | PHP-MySQL |
| Thiry      | Delphine | PHP-MySQL |
| Lemoine    | Audrey   | PHP-MySQL |
| Cordonnier | Yvan     | Word      |
| Pichon     | AndrÚ    | Word      |
| Maniette   | Vincent  | Word      |
+------------+----------+-----------+
17 rows in set (0.63 sec)

mysql> _
```

Vous remarquerez que certaines colonnes de la clause *where* sont écrites sous leur nom long (*<nom de la table>.<nom de la colonne>*). C'est dû à la nécessité de ne pas confondre des noms de colonnes identiques dans deux tables (ici, *id*). Pour les autres colonnes, il n'y a pas d'ambiguïtés. On peut combiner une jointure et un tri, comme dans la commande suivante, en demandant d'afficher les résultats par ordre alphabétique des noms, puis des titres.

```
mysql> select nom,prenom,titre from formes,formations inscriptions where
id_formation=formations.id && id_forme=formes.id order by nom,titre
```

```
mysql> select nom,prenom,titre from formes,formations,inscriptions where id_form
ation=formations.id && id_forme=formes.id order by nom,titre;
+------------+----------+-----------+
| nom        | prenom   | titre     |
+------------+----------+-----------+
| Cordonnier | Yvan     | CCM       |
| Cordonnier | Yvan     | PHP-MySQL |
| Cordonnier | Yvan     | Word      |
| Courtin    | Pierre   | PHP-MySQL |
| Delire     | Jacques  | CCM       |
| Delire     | Jacques  | Flash     |
| Dethy      | Pierre   | Flash     |
| Lemoine    | Audrey   | CCM       |
| Lemoine    | Audrey   | PHP-MySQL |
| Maniette   | Vincent  | Flash     |
| Maniette   | Vincent  | Word      |
| Pichon     | AndrÚ    | PHP-MySQL |
| Pichon     | AndrÚ    | Word      |
| Thiry      | Delphine | PHP-MySQL |
| Thomas     | Martine  | CCM       |
| Thomas     | Martine  | Flash     |
| Vandeuren  | Isabelle | CCM       |
+------------+----------+-----------+
17 rows in set (0.00 sec)

mysql>
```

Ce qui est différent de la commande qui suit où le tri est d'abord réalisé sur le titre puis sur le nom.

```
mysql> select nom,prenom,titre from formes,formations inscriptions where
id_formation=formations.id && id_forme=formes.id order by titre,nom
```

```
mysql> select nom,prenom,titre from formes,formations,inscriptions where id_form
ation=formations.id && id_forme=formes.id order by titre,nom;
+------------+----------+-----------+
| nom        | prenom   | titre     |
+------------+----------+-----------+
| Cordonnier | Yvan     | CCM       |
| Delire     | Jacques  | CCM       |
| Lemoine    | Audrey   | CCM       |
| Thomas     | Martine  | CCM       |
| Vandeuren  | Isabelle | CCM       |
| Delire     | Jacques  | Flash     |
| Dethy      | Pierre   | Flash     |
| Maniette   | Vincent  | Flash     |
| Thomas     | Martine  | Flash     |
| Cordonnier | Yvan     | PHP-MySQL |
| Courtin    | Pierre   | PHP-MySQL |
| Lemoine    | Audrey   | PHP-MySQL |
| Pichon     | AndrÚ    | PHP-MySQL |
| Thiry      | Delphine | PHP-MySQL |
| Cordonnier | Yvan     | Word      |
| Maniette   | Vincent  | Word      |
| Pichon     | AndrÚ    | Word      |
+------------+----------+-----------+
17 rows in set (0.01 sec)

mysql> _
```

Dans certains cas, pour raccourcir l'écriture des requêtes, on utilise des alias pour les tables.

```
mysql> select nom,prenom,titre from formes as f1,formations as f2, inscriptions where
id_formation=f2.id && id_forme=f1.id order by titre,nom
```

La table *formes* a comme alias *f1* et la table *formations*, *f2*.

```
mysql> select nom,prenom,titre from formes as f1,formations as f2,inscriptions w
here id_formation=f2.id && id_forme=f1.id;
+------------+----------+-----------+
| nom        | prenom   | titre     |
+------------+----------+-----------+
| Cordonnier | Yvan     | CCM       |
| Delire     | Jacques  | CCM       |
| Vandeuren  | Isabelle | CCM       |
| Thomas     | Martine  | CCM       |
| Lemoine    | Audrey   | CCM       |
| Dethy      | Pierre   | Flash     |
| Delire     | Jacques  | Flash     |
| Thomas     | Martine  | Flash     |
| Maniette   | Vincent  | Flash     |
| Cordonnier | Yvan     | PHP-MySQL |
| Pichon     | AndrÚ    | PHP-MySQL |
| Courtin    | Pierre   | PHP-MySQL |
| Thiry      | Delphine | PHP-MySQL |
| Lemoine    | Audrey   | PHP-MySQL |
| Cordonnier | Yvan     | Word      |
| Pichon     | AndrÚ    | Word      |
| Maniette   | Vincent  | Word      |
+------------+----------+-----------+
17 rows in set (0.01 sec)
```

Dans d'autres cas, cette utilisation est inévitable comme dans le cas des **autojointures**.

Si nous recherchons la liste des inscrits qui sont nés la même année, nous donnerons la commande

```
mysql> select f1.nom,f1.naissance from formes as f1,formes as f2 where
f1.naissance=f2.naissance && f1.id!=f2.id;
```

Cette commande crée deux alias pour la même table, ce qui permet de comparer une ligne avec toutes les autres, par exemple. Seront retenus, les nom et date de naissance des lignes de la première instance de la table lorsque cette même année de naissance sera retrouvée dans une autre ligne.

```
mysql> select upper(nom),naissance from formes;
+-------------+-----------+
| upper(nom)  | naissance |
+-------------+-----------+
| DETHY       |      1967 |
| PICHON      |      1970 |
| DELIRE      |      1976 |
| COURTIN     |      1982 |
| VANDEUREN   |      1979 |
| THOMAS      |      1987 |
| THIRY       |      1976 |
| MANIETTE    |      NULL |
| LEMOINE     |      1983 |
| CORDONNIER  |      1965 |
+-------------+-----------+
10 rows in set (0.01 sec)

mysql> select f1.nom,f1.naissance from formes as f1,formes as f2 where f1.naissa
nce=f2.naissance && f1.id!=f2.id;
+--------+-----------+
| nom    | naissance |
+--------+-----------+
| Thiry  |      1976 |
| Delire |      1976 |
+--------+-----------+
2 rows in set (0.00 sec)

mysql>
```

Il existe d'autres types de jointures. On trouvera en bibliographie des ouvrages intéressants à ce propos.

3.5 Gérer une BD avec PHP

3.5.1 Le principe de communication

La gestion d'une base de données avec *MySQL* étant bien amorcée, il reste à voir comment *PHP* peut s'en accommoder. Nous avons vu que les requêtes étaient fournies à *PHP* sous forme de chaînes de caractères éventuellement stockées dans des variables avant d'être prises en charge par des fonctions. Les fonctions dont il est question sont des fonctions de communication avec *MySQL*. Il importe d'en connaître les principales, de savoir ce qu'elles prennent en argument et ce qu'elles retournent comme types de valeurs.

3.5.2 Les fonctions PHP

Il existe une bonne trentaine de fonctions dédiées à la communication entre *PHP* et *MySQL*. Nous illustrons ici celles qui paraissent les plus intéressantes aux premiers stades du développement d'une application de gestion de base de données avec *PHP*.

3.5.2.1 mySql_connect

Cette fonction permet d'établir la connexion avec un serveur *MySQL*. Elle prend trois arguments de type chaîne: le nom de l'hôte, le nom de l'utilisateur et le mot de passe.

```
entier mySql_connect(chaine hôte, chaîne utilisateur, chaîne mot de passe)
```

Tous les arguments sont optionnels. Ils existent par défaut (*localhost*, le propriétaire du processus et le mot de passe vide). Voici un exemple concret d'ouverture de connexion:

```
<?php
$link = mysql_connect("localhost", "root", "")
   or die("<p>Impossible de se connecter : ".mysql_error()."</p>");
```

```
print ("Connexion réussie.");
mysql_close($link);
?>
```

Cet exemple illustre diverses choses. La fonction renvoie un entier qui est un identifiant de la connexion. Cette valeur est utilisée pour clore la connexion au moyen d'une autre fonction *mySql_close* qui la prend en argument et qui renvoie une valeur booléenne. Notez que la précision de cette valeur, dans cet exemple, est inutile. Ce n'est que lorsque la confusion est possible qu'elle devient nécessaire (plusieurs connexions ouvertes, par exemple). De même, la fermeture de la connexion n'est pas nécessaire. Elle accompagne la fin de l'exécution du script.

Les paramètres fournis à la fonction sont ceux qui seraient fournis lors d'une connexion normale à *MySQL* via la ligne de commande.

L'exécution de la fonction s'accompagne d'une option qui commande au script de se terminer si la connexion ne peut avoir lieu. Le mot-clé *die* est synonyme de *exit*, mais *die* se comporte comme une fonction en ce sens qu'elle peut prendre comme argument une chaîne de caractères qui sera affichée en cas d'arrêt du script.

mySql_error est aussi une fonction qui renvoie une chaîne de caractères contenant le texte du message d'erreur. Il existe une fonction *mySql_pconnect* qui rend les connexions permanentes, ce qui évite des pertes de temps, dans certains cas.

3.5.2.2 mySql_create_db

Une fois connecté au serveur *MySQL*, il convient d'activer une base de données ou d'en créer une. La création d'une base de données s'effectue par le biais de la fonction *mySql_create_db*.

```
booléen mySql_create_db(chaîne base de données, entier lien)
```

Voici une illustration de l'utilisation de cette fonction:

```
<?php
$link = mysql_pconnect("localhost", "root", "") or die ("Connexion impossible");
if (mysql_create_db("ma_db")) {
   echo "<p>La base de données a été correctement créée.\n</p>";
}
else {
   echo "<p>Erreur lors de la création de la base : ".mysql_error()."</p>";
}
?>
```

Il est à noter que la fonction *mySql_query* que nous avons déjà illustrée tout au début de ce chapitre permet d'obtenir le même résultat avec la commande *create database*. Elle sera préférée à la précédente. Dans cet exemple, le résultat renvoyé par la fonction est utilisé dans un test.

3.5.2.3 mySql_select_db

Cette fonction qui renvoie un booléen permet de sélectionner une base de données. Elle prend comme arguments, le nom de la base de données et l'entier lien résultant de l'ouverture de la connexion.

```
booléen mySql_select_db(chaîne base de données, entier lien)
```

Une séquence classique dans un script *PHP* sera donc:

```
<?php
$link = mysql_connect('localhost', 'root', '');
```

```
if (!$link) {
   die('Non connecté : ' . mysql_error());
}
$db = mysql_select_db('ma_db', $link);
if (!$db) {
   die ('<p>Impossible d\'utiliser la base : '.mysql_error().'</p>');
}
?>
```

3.5.2.4 mySql_query

Si la connexion a bien été établie et la base de données bien sélectionnée, il est possible d'émettre toutes les requêtes possibles et imaginables: créations de tables, insertions, suppressions, mises à jour de lignes, sélections diverses,…

On ne confiera pas nécessairement à un script *PHP* le soin de créer des tables, surtout si celles-ci sont permanentes dans l'application, mais pourquoi pas?

La fonction *mySql_query* est importante car elle permet de véhiculer toutes les requêtes et d'en récupérer les résultats.

```
entier mySql_query(chaîne requête, entier lien)
```

Pour les requêtes d'insertion, de suppression et de mise à jour, elle retourne une valeur booléenne. Pour une sélection, elle renvoie un tableau de résultats.

Voici à nouveau une illustration de séquence possible:

```
<?php
$link = mysql_connect('localhost', 'root', '');
if (!$link) {
   die('<p>Impossible de se connecter : '. mysql_error().'</p>');
}
$db = mysql_select_db('exemple', $link);
if (!$db) {
   die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
}
$sqlquery="select prenom,nom,institution from formes order by nom";
$queryresult=mysql_query($sqlquery);
…
?>
```

La requête *MySQL* est stockée dans la variable `$sqlquery` qui est passée en argument à la fonction *mysql_query*.

Il existe également une fonction *mySql_db_query* semblable à la précédente, si ce n'est qu'elle prend un argument de plus, le nom de la base de données sur laquelle porte la requête.

La fonction renvoie un entier qui identifie la requête et qui est stocké, dans l'exemple, dans la variable `$queryresult`. Cet entier est utilisé, par la suite, par des fonctions d'exploration des résultats telles celles qui suivent.

3.5.2.5 mySql_fetch_row

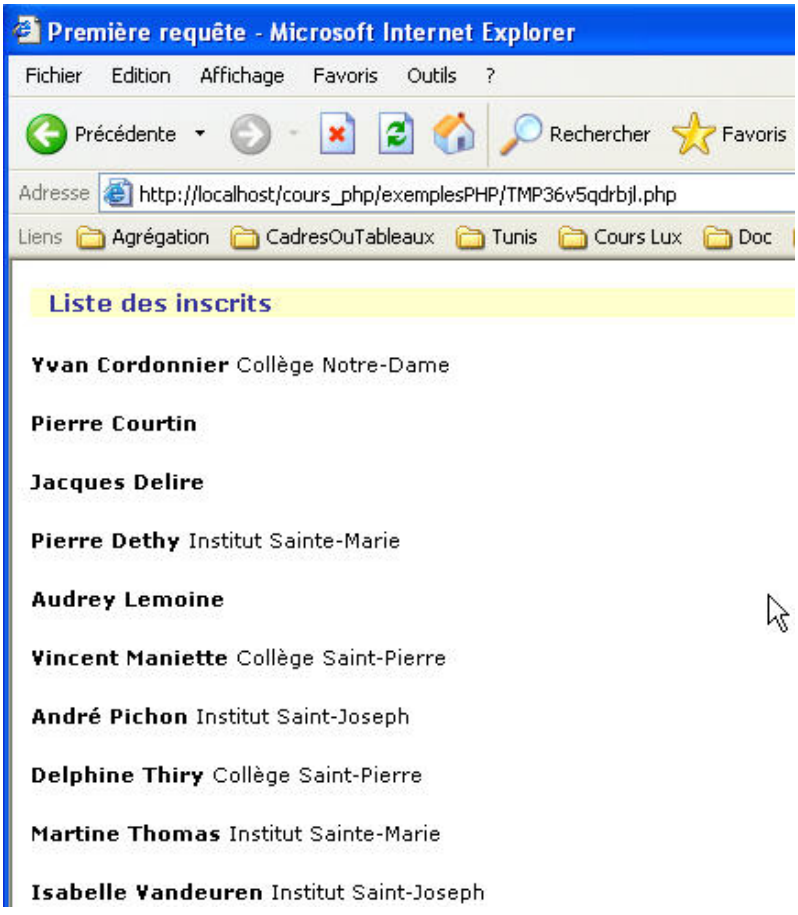
Cette fonction renvoie, sous forme d'un tableau, la ligne courante. À chaque appel, c'est la ligne suivante qui est prise en compte jusqu'à ce qu'il n'y en ait plus (auquel cas la fonction renvoie la valeur *false*).

```
tableau mySql_fetch_row(entier résultat)
```

L'entier résultat est celui qui identifie la requête effectuée via *mySql_query*. Le tableau est indexé à partir de 0 et peut être parcouru pour exploitation des valeurs comme dans l'exemple suivant:

```
<?php
echo "<h1>Liste des inscrits</h1>";
$link = mysql_connect('localhost', 'root', '');
if (!$link) {
   die('<p>Impossible de se connecter : '. mysql_error().'</p>');
}
$db = mysql_select_db('exemple', $link);
if (!$db) {
   die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
}
$sqlquery="select nom,prenom,institution from formes order by nom";
$queryresult=mysql_query($sqlquery);
while ($row=mysql_fetch_row($queryresult)){
   echo "<p><strong>".$row[1]." ".$row[0]."</strong> ".$row[2]."</p>";
}
?>
```

Ce petit script ouvre la base de données *exemple* et lance une requête portant sur trois colonnes de la table *formes*. Les informations sont récupérées, ligne par ligne, dans un tableau `$row` et mises en forme pour un affichage correct. Voici ce que cela donne.



3.5.2.6 mySql_fetch_array

Cette function est quasi identique à la précédente. La différence réside dans le fait qu'elle possède un argument de plus. Cet argument précise par l'intermédiaire d'une constante (*MYSQL_NUM* ou *MYSQL_ASSOC*) si le tableau est indexé classiquement ou si c'est un tableau associatif auquel cas, les noms de colonnes peuvent être utilisés.

```
tableau mySql_fetch_array(entier résultat, entier type)
```

La boucle de l'exemple précédent peut être remplacée par

```
while ($row=mysql_fetch_array($queryresult,MYSQL_ASSOC)){
   echo "<p><strong>".$row['prenom']." ".$row['nom']
        ."</strong> ".$row['institution']."</p>";
}
```

3.5.2.7 mySql_num_rows

Cette fonction renvoie, comme on s'en doute, le nombre de lignes du résultat dans le cas d'une requête de sélection. Elle est particulièrement utile lorsqu'on veut s'assurer qu'il y en a.

```
<p>
<?php
$link = mysql_connect("localhost", "root", "");
mysql_select_db("exemple", $link);
$queryresult = mysql_query("SELECT * FROM formateurs", $link);
$num_rows = mysql_num_rows($queryresult);
if (!$num_rows){
   echo "Pas de formateurs !";
}
else{
   echo "Il y a $num_rows formateurs.\n";
}
?>
</p>
```

Dans notre exemple:

Il y a 4 formateurs.

3.5.2.8 mySql_affected_rows

C'est tout simplement le correspondant de la fonction précédente pour les requêtes de mises à jour. Elle se révèle utile dans la mesure où elle permet de tenir l'utilisateur au courant de ce qui est en train de se passer.

3.5.2.9 Autres fonctions

Nous n'avons illustré que les fonctions les plus directement utiles. Les autres fonctions sont utilisées plus occasionnellement. Leur rôle est cependant facile à comprendre si vous avez compris le rôle de celles qui ont précédé. Le plus souvent, leur nom fournit déjà une bonne description de ce à quoi elles sont destinées. Citons tout de même les fonctions qui permettent de connaître les caractéristiques de

l'information issue d'une colonne dans un résultat et notamment, la longueur (*mySql_field_len*), le nom (*mySql_field_name*), le nom de la table (*mySql_field_table*), le type (*mySql_field_type*)…

Pour le reste, nous vous renvoyons à l'excellente documentation en ligne et aux ouvrages cités en bibliographie.

3.6 Bilan

Dans ce chapitre, nous avons survolé un grand nombre de connaissances et évoqué un certain nombre de savoir-faire:

- réaliser un schéma *ERA*,
- transformer un schéma en tables,
- maîtriser le langage *MySQL*,
- maîtriser les fonctions de base permettant à *PHP* de communiquer avec *MySQL*,
- …

Ces éléments nous permettent de traiter complètement, des applications qui allient la création de pages dynamiques avec la gestion des bases de données. Certaines requêtes n'ont pas été envisagées sous l'angle de *PHP*, telles l'insertion ou la mise à jour des lignes d'une table, mais vous en connaissez le principe.

L'utilisation des formulaires va s'avérer particulièrement intéressante pour la gestion des données par l'utilisateur. En les remplissant, il va permettre l'enregistrement de lignes dans des tables. Il sera également possible, via des scripts correctement écrits, de lui renvoyer des formulaires garnis d'information, à compléter ou à modifier.

Tout est donc en place pour la réalisation d'une application en ligne, même si les listes des primitives, des fonctions et des opérateurs des différents langages sont loin d'avoir été explorées à fond.

Voici un petit exercice pour commencer en douceur.

3.7 Exercice

- Il va s'agir de concevoir un mini schéma *ERA* décrivant une situation que vous connaissez. Ce schéma ne comprendra pas plus de deux ou trois entités (au minimum deux) et au moins une association de plusieurs à plusieurs.
- Lorsque ce schéma sera complet, transformez-le en tables selon les règles que nous avons établies. Déterminez clairement les attributs de chacune des entités et identifiez correctement les clés primaires et les clés étrangères.
- Créez la base de données et les tables en vous aidant de l'interface de *PHPMyAdmin*. Ajoutez-y déjà quelques lignes.
- Écrivez un script *PHP* qui affiche quelques colonnes d'une des tables avec une mise en forme particulière.
- Écrivez un script *PHP* qui effectue une requête portant sur plusieurs tables.
- Écrivez un script *PHP* qui permette de modifier les données d'une table. Pour ce faire, utilisez des formulaires que vous allez garnir avec les informations disponibles. L'utilisateur modifiera les données de ces formulaires et donnera l'ordre d'enregistrer. Une page dynamique sera construite pour lui fournir tous les renseignements nécessaires sur le déroulement du traitement effectué sur la base de données.

4. Application

4.1 Introduction

Afin de mettre en musique les différents éléments qui ont été présentés dans les trois premiers chapitres, nous nous proposons d'insister sur les démarches à mettre en œuvre lors du développement d'une application. Par application, nous entendons essentiellement deux choses:

- la mise en place d'un programme sous forme d'un ensemble de scripts ayant pour objectif principal, la fourniture d'informations extraites d'une base de données et leur mise à jour,
- la création de l'interface de communication permettant à un être humain, selon les privilèges qui lui sont accordés, de gérer cette base de données et/ou de formuler des demandes à propos d'informations qu'elle est censée contenir.

La conception d'une telle application, si élémentaire soit-elle, demande une bonne analyse de l'environnement dans lequel elle est supposée s'exécuter. Sans aller jusqu'au développement complet de l'exemple que nous allons traiter, nous insisterons sur les points importants à ne pas négliger dans la démarche de conception d'une telle application.

4.2 Un énoncé à raffiner

Lorsqu'il s'agit d'imaginer une application sur le Web dont le but est de couvrir un domaine d'activité relativement bien connu, il est nécessaire de disposer d'une bonne description de ce domaine. Diverses techniques sont possibles, afin de récolter un maximum d'informations pertinentes: interview des acteurs, des responsables, analyse de documentation (si elle existe), exercice de description de l'application par une analyse d'applications similaires,…

Les acteurs et les activités sont également importantes à cerner, de manière à modéliser une réalité qui ne soit pas trop étendue. D'où l'importance de prendre en compte le public auquel cette application est destinée.

Nous allons tâcher d'illustrer cela très concrètement sur base d'un énoncé issu de la description écrite d'une situation, par une personne réputée bien la connaître. Cette description est forcément incomplète et nous a amenés à faire des suppositions qui pourraient se révéler incorrectes si elles ne sont pas rapidement confrontées à l'avis des principaux acteurs. Ceci donne déjà quelques indications sur la manière de procéder avant de réfléchir à un schéma conceptuel.

Voici donc l'énoncé fourni.

Il s'agirait de mettre sur l'intranet d'une école, voire sur Internet, les informations concernant les stages des élèves. On peut décrire la situation en disant qu'il y a :

- *des maîtres de stages,*
- *des élèves de puériculture (12 classes réparties en 5e, 6e et 7e puériculture), de nursing (6 classes réparties en 5e et 6e nursing), de gériatrie (2 classes réparties en 5e et 6e gériatrie),*
- *des lieux de stages (une vingtaine de lieux différents),*
- *des périodes de stages.*

Pendant une même période, dans un même lieu, des élèves aussi bien de nursing que de puériculture, tant de 5e, 6e ou 7e se retrouvent parfois ensemble.

À titre d'exemples, voici quelques opérations que différents acteurs pourraient avoir envie de réaliser:

- *un élève de telle classe pourrait rechercher à quel moment il va en stage, en quel lieu, avec quel professeur et quels élèves (de quelles classes);*
- *un professeur chercherait quels élèves il supervise à quelle période;*
- *s'il y a un accès via Internet, telle crèche pourrait savoir quels élèves de telle classe, section, elle va devoir accueillir à tel moment.*

Cette simple description, bien sûr fort incomplète, va cependant nous permettre de mettre en évidence un certain nombre de démarches et de pratiques qui paraissent fondamentales.

4.3 Un bon schéma conceptuel

4.3.1 Première ébauche

La première démarche consiste à représenter le domaine de l'activité, sous forme d'un schéma conceptuel. Un schéma entités-associations peut très bien convenir pour ce genre d'opération. Les entités, comme les associations et les rôles, peuvent généralement être mis en évidence au travers du discours. Les cardinalités peuvent également être déduites du discours ou faire l'objet, sinon, d'une demande de précision.

De la description qui précède, nous retirons d'abord qu'il y a des élèves, des professeurs ou maîtres de stage, des lieux de stage.

Il est également question de stages, mais la notion de stage est plus délicate à cerner, notamment en ce qui concerne l'établissement des cardinalités. On peut considérer qu'un stage est la prestation notée d'un élève dans un lieu et un domaine fixés.

Les élèves font partie d'une classe. Nous considérons que les stages se déroulent pendant une période compacte, ce qui signifie d'une date à une autre[27].

Nous retenons donc dans un premier temps, les entités *Élève*, *Maître*, *Stage*, *Lieu*, *Date* et *Classe*. Un élève fait partie d'une classe et il effectue des stages. Les stages se déroulent dans des lieux et commencent à une certaine date pour finir à une autre. Les maîtres de stage supervisent des stages.

Un *élève* **appartient à** une (et une seule) *classe* et une *classe* **se compose d'**au moins un *élève*. L'*élève* effectue ou **participe** au minimum un *stage* et lorsque nous parlons d'un *stage*, il s'agit de celui d'un seul *élève*. Le *stage* **se déroule en** un seul *lieu*, même si des implantations différentes d'un même *lieu* peuvent exister. Nous considérons que celles-ci ne sont pas déterminantes dans l'activité de consultation et de mise à jour que nous envisageons. Un *lieu* **accueille** généralement plusieurs *stages*. Un *stage* **est supervisé par** un seul *maître* de stage, mais un *maître* de stage **supervise** habituellement plusieurs *stages*. Un *stage* **commence à** une certaine *date* et **se termine à** une autre. Une *date* **est** potentiellement **la date de début** de plusieurs stages et **la date de fin** de plusieurs autres.

Tout cela nous donne le schéma ERA suivant que nous allons remanier.

4.3.1.1 Réflexion sur les cardinalités

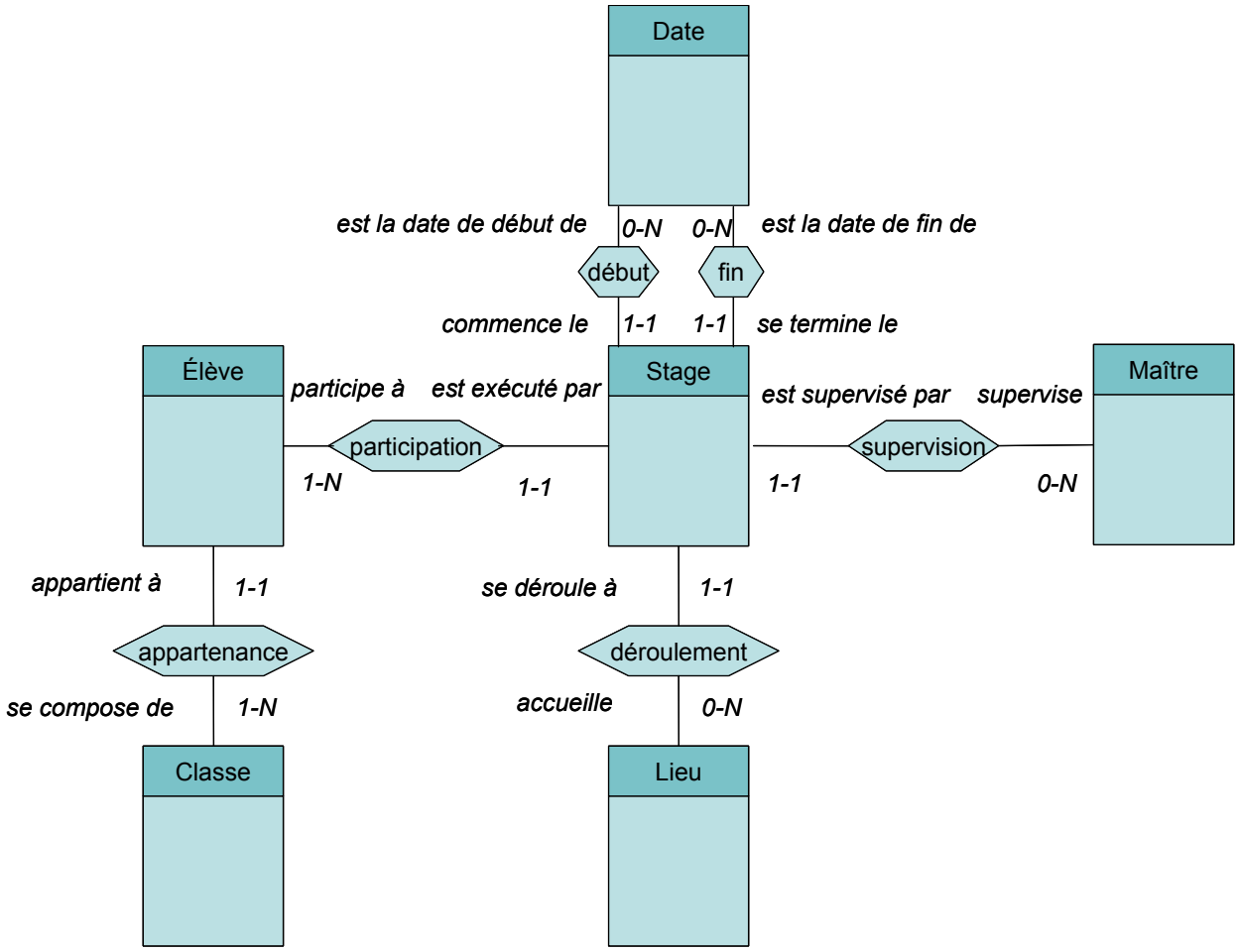
Plusieurs éléments sont à prendre en compte à propos des associations. S'agit-il de relations de un à un, de un à plusieurs ou de plusieurs à plusieurs? Ceci concerne les deux côtés de l'association. Si on s'intéresse à un seul de deux côtés, il faut également déterminer si la cardinalité démarre à zéro ou à un.

La réponse à la première question est évidente si la description du domaine est claire. La réponse à la deuxième dépend de l'importance accordée à l'entité. Existe-t-elle en l'absence d'instanciation de

---

27 Il est clair que de tels choix orientent la modélisation et que celle-ci représente plus ou moins bien la réalité.

l'association? Si oui, zéro est le bon choix. Par exemple, un lieu de stage peut exister dans la base de données alors qu'aucun élève ne s'y rend au cours de cette année. De même, un maître de stage peut se retrouver occasionnellement sans élèves à superviser. Cela n'empêche pas le système de les prendre en compte pour une série de traitements, comme par exemple, leur permettre de consulter les informations sur les stages.



Côté dates, on admettra facilement qu'une date donnée ne corresponde au début ou à la fin d'aucun stage. Mais en ce qui les concerne, nous allons quelque peu simplifier le schéma.

4.3.2 Améliorations du schéma

4.3.2.1 Retouches du schéma conceptuel

À ce stade, il importe de se demander si toutes les entités sont vraiment utiles dans le sens où, lors du passage au schéma logique, elles vont donner lieu à des tables. Dans le cas présent, il est aussi simple de considérer que l'entité *stage* a comme attributs une **date de début** et une **date de fin**. Cela réduira le nombre de tables à gérer.

De même, l'activité envisagée fait peu de place à la notion de classe. Cette entité peut donc être transformée en attribut de l'entité *élève*.

Pour le lieu, c'est plus délicat car nous souhaitons que les responsables d'un lieu de stage puissent également bénéficier des services de l'application. Nous devons donc maintenir *lieu* comme entité.

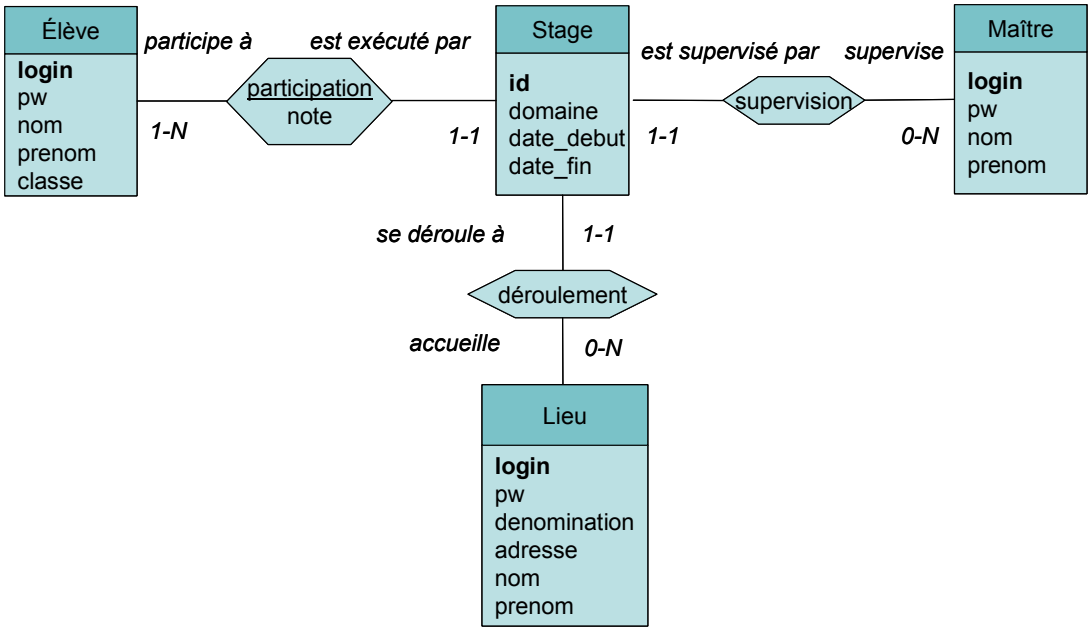
4.3.2.2 Attributs des entités et des associations

Il est bon que chacune des entités possède un identifiant. De même, notre application étant utilisée par différents types de clients, il est bon que chacun de ceux-ci possède en outre un mot de passe. On pourrait également faire évoluer le schéma en ce sens qu'un lieu de stage pourrait compter plusieurs utilisateurs qui auraient chacun un identifiant et un mot de passe. Nous allons simplifier ce dernier problème en considérant, ce qui est raisonnable, que ces utilisateurs potentiels se servent du même identifiant et du même mot de passe. Ce faisant, nous pouvons créer deux attributs de l'entité *lieu* qui les représentent.

Les autres attributs dépendent essentiellement des informations dont on souhaite disposer à propos des entités et des associations mises en évidence. Citons au minimum comme exemples: un **nom** et un **prénom** pour les *élèves* et les *maîtres*, une **dénomination** et une **adresse** principale pour les *lieux* et une **note** pour la *participation* à un stage.

Nous avons décidé que la **note** était un attribut de la participation, bien que dans ce cas, et vu la cardinalité 1-1 entre cette association et l'entité *stage*, nous aurions pu en faire directement un attribut de l'entité.

Voici donc comment évolue notre schéma.

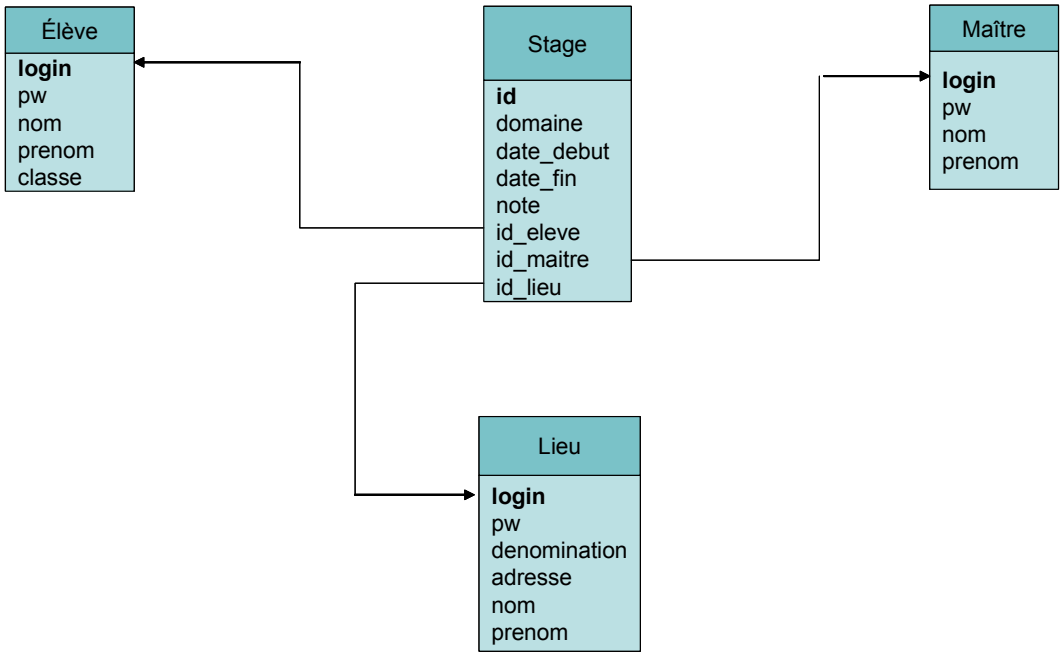


Il est un peu anormal de voir apparaître des attributs **nom** et **prénom** dans l'entité *lieu*. Il s'agit des coordonnées d'un responsable dont nous n'avons pas voulu faire une entité. Cet aménagement, non obligatoire, est dû à l'intérêt d'avoir une structure semblable pour l'ensemble des utilisateurs de l'application (élèves, maîtres de stage et responsables de site) qui autorise des simplifications au niveau du code (voir plus loin).

4.4 Le schéma logique

La transformation du schéma conceptuel en schéma logique est quasi automatique. Seules les associations de plusieurs à plusieurs donnent naissance à des tables supplémentaires. Il n'y en a pas dans l'exemple. Nous aurons donc seulement quatre tables correspondant aux entités du schéma.

L'attribut note se retrouve dans la table *stage* qui, occupant une place centrale dans le schéma, contiendra des clés étrangères permettant les liens aux entités *élève*, *maître* et *lieu*. Ces attributs porteront les noms de **id_eleve**, **id_lieu** et **id_maitre**.



Nous constatons que chacune des tables possède un identifiant (clé primaire) et que la table *stage* possède des clés étrangères permettant la liaison avec d'autres tables. D'autres attributs intéressants pourraient également figurer en fonction des besoins, mais nous avons voulu nous concentrer sur la démarche en évitant de noyer les explications de celle-ci dans une foule de détails.

4.5 La construction d'une interface

Une interface de communication entre des êtres humains et une application se doit d'être utile et utilisable. En gros, l'utilité se mesure à la coïncidence entre les attentes des utilisateurs (ses buts) lorsqu'il utilise l'application et les fonctionnalités proposées par celle-ci au travers de cette interface. Alors que l'utilisabilité mesure plutôt le degré de facilité d'utilisation de l'interface.

4.5.1 Utilité

Une bonne manière de développer une interface utile est d'imaginer les traitements que l'utilisateur aura envie de faire effectuer par l'application. Ceci peut se faire de différentes manières: interview des utilisateurs potentiels, examen de produits équivalents,…

Dans une application comme celle qui nous occupe, il s'agit surtout de déterminer quelles sont les requêtes de sélection d'informations qui risquent d'avoir le plus de succès aux yeux des utilisateurs et quelles sont les requêtes de mise à jour qui sont souhaitables, par qui.

Nous avons repris ici quelques requêtes possibles. Nous n'avons pas effectué un travail exhaustif qui nous aurait mené trop loin. L'interface que nous avons créée, dans l'état de son développement, permet de répondre aux besoins suivants.

- Pour un élève:
  - obtenir la liste des superviseurs de stage,
  - obtenir la liste des lieux et dates de ses stages,

    - prendre connaissance de son lieu de stage à une date donnée.
- Pour un maître de stage:
    - obtenir la liste des élèves qu'il supervise,
    - ajouter un élève dans la base de données,
    - programmer un stage pour un élève.

Nous n'avons pas programmé de requête pour les responsables de site, mais les exemples pris suffisent à faire comprendre quel est le principe de cette programmation.

Beaucoup de contraintes devraient être prises en compte. Elles relèvent de ce que l'on appelle la prévention des erreurs dans la conception des interfaces homme-machine (IHM). Parmi celles-ci notons, par exemple, la vérification de l'unicité de l'identifiant pour chacun des utilisateurs dans sa catégorie, le non chevauchement des périodes de stage pour un même élève (quoique), la validité des formats de date utilisés, la validité des dates elles-mêmes,… Chacune de ces contraintes, considérée séparément, ne demande pas un gros effort de programmation. La détermination et la prise en compte de toutes les contraintes est un problème qui requiert une analyse fouillée et surtout, l'écriture de nombreuses lignes de code.

4.5.2 Utilisabilité

L'utilisabilité se mesure davantage par la manière dont le concepteur de l'interface facilite le travail de l'utilisateur en le guidant dans ses actions, en réduisant autant que faire se peut sa charge de travail, en lui évitant de commettre des erreurs, notamment.

Sans entrer dans l'écriture d'un traité sur l'ergonomie des interfaces, nous donnons quelques conseils permettant de les rendre relativement ergonomiques.

Par la création d'une feuille de style, il est possible de promouvoir :

- l'utilisation d'une police de caractères lisibles (larges et sans empattements),
- la non prolifération des polices (deux au plus),
- des choix appropriés pour l'appariement des couleurs,
- …

La création de modèles, et notamment, l'utilisation de calques ou de tableaux, permet de gérer une certaine uniformité au niveau de la mise en page et favorise, par là, un bon guidage de l'utilisateur.

Ce sont les options essentielles que nous avons prises dans l'écriture des scripts *PHP* qui suivent.

4.5.3 Création de modèles

Avant d'aborder la création de modèles, il convient d'avoir une idée claire de la manière dont les fichiers seront organisés en dossiers, sans quoi, toute référence correcte est impossible.

Nous avons imaginé de procéder comme suit (voir illustration au point 4.7) :

- un dossier *scripts* pour contenir les différentes pages de l'application, la plupart donnant lieu à l'écriture de scripts PHP
- un dossier *images* pour y placer toutes les images du site
- un dossier modeles qui contiendra les fichiers permettant des économies de code

Le fichier *cefis.css* contenant la feuille de style, quant à lui, se trouvera à la racine du site.

De la sorte, il n'y a pas d'équivoque sur la manière de référencer les fichiers.

4.5.3.1 Un modèle basique

Si l'on considère ce que doit être une page d'un site tel celui que nous voulons mettre en place, nous pouvons en dégager un certain nombre de caractéristiques communes, notamment en ce qui concerne la répartition de diverses zones, la présence permanente de certaines informations (un bandeau, par exemple), la présence de certains liens,…

Voici un modèle de base. Il est constitué d'une zone d'entête (*entete.php*), d'une zone de pied de page (*pieddepage.php*). Le modèle (*modele.php*) mentionnera l'inclusion du code de ces deux fichiers. Voici son propre code.

```
<?php include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
    <em>&lt;Corps de texte ici&gt;</em>
    </td>
  </tr>
</table>
<?php include('../modeles/pieddepage.php'); ?>
```

On pourrait s'étonner de constater que la référence aux fichiers à inclure fasse appel au dossier parent pour ensuite se replonger dans le dossier *modeles* (*../modeles/*) alors que les fichiers se trouvent dans ce même dossier *modeles*. Il faut bien s'imaginer que le fichier construit va servir de modèle à des fichiers se trouvant dans le dossier *scripts*. C'est la raison pour laquelle il faut utiliser une référence plus complète.

Le corps du modèle est un tableau d'une seule cellule contenant un titre générique et un texte par défaut qui sera à supprimer lors de la création d'une page. L'intérêt du texte par défaut est de signaler à l'usager qu'il est en présence du modèle.

Ce modèle est très générique et peut donner naissance à d'autres modèles plus complets, résultant de l'inclusion d'autres fichiers.

Voici le code du fichier *entete.php*…

```
<html>
<head>
<title>Institut ABC</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<link href="../cefis.css" rel="stylesheet" type="text/css">
</head>
<body>
<table width="555" height="113" border="1" cellpadding="10" cellspacing="0"
bordercolor="#FF8888" bgcolor="#FFDFD5" class="grand">
  <tr>
    <td height="109">
    <div align="center"><img src="../images/cathedrale.jpg" width="112" height="89"
    align="left">
    <span class="grand">Institut ABC</span>
    </div>
```

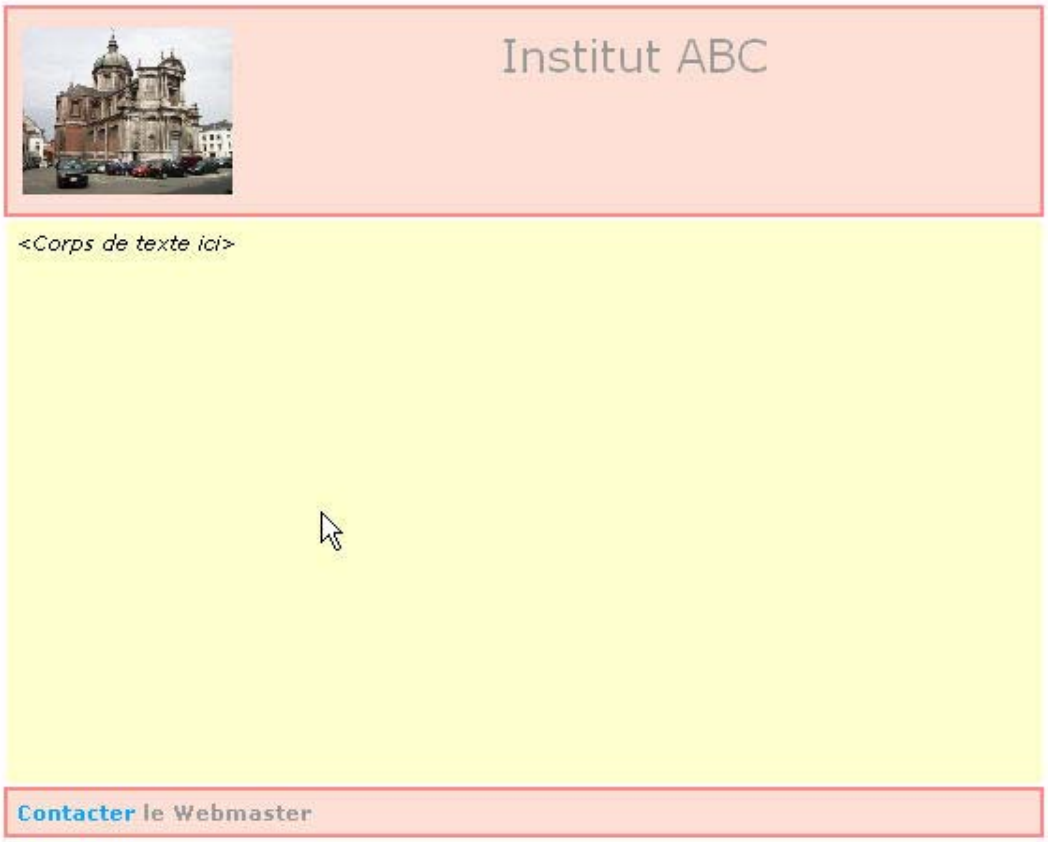
```
    </td>
  </tr>
</table>
```

Ce fichier contient le code initial et, en particulier, la liaison à la feuille de style *cefis.css* dont le code suit. Un tableau d'une seule cellule contient un calque dans lequel on trouve la photo et le titre du bandeau.

Voici le code du fichier *pieddepage.php*.

```
<table width="555" border="1" cellpadding="10" cellspacing="0" bordercolor="#FF8888"
bgcolor="#FFDFD5">
  <tr>
    <td>
      <a href="mailto:webmaster@det.fundp.ac.be">Contacter</a><span class="gris"> le
      Webmaster</span>
    </td>
  </tr>
</table>
</body>
</html>
```

Ce dernier contient la terminaison du fichier, mais aussi un tableau d'une cellule contenant un lien vers un message électronique à l'adresse du Webmaster.



Le code du modèle, interprété ci-dessus, n'utilise qu'une petite partie des styles définis dans la feuille de style contenue dans le fichier *cefis.css* et qui est décrite ci-après.

```
body {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-style: normal; background-color: #FFFFFF;
      font-size: 11px
}
td {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-style: normal;
      font-size: 11px;
      margin: 0px;
      padding: 5px;
}
a {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-style: normal; color: #00AAFF;
      text-decoration: none;
      font-weight: bold; font-size: 11px
}
a:hover {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-style: normal; color: #003399;
      font-weight: bold; background-color: #DFDFFF;
      font-size: 11px
}
a:link {
      font-family: Verdana, Arial, Helvetica, sans-serif; font-style: normal;
      font-size: 11px
}
div {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 11px;
      font-style: normal
}
br {
      font-family: Verdana, Arial, Helvetica, sans-serif; font-size: 11px;
      font-style: normal;
      margin-bottom: auto
}
p {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-style: normal ;
      margin-bottom: auto;
      font-size: 11px
}
ol {
      font-family: sans-serif;
      list-style-image: none
}
```

```
ul {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-style: normal ;
      list-style-image: url(images/puce.gif)
}
li {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-style: normal;
      padding-bottom: 3px;
      font-size: 11px
}
hr {
      border-color: #333399 black black;
      border-top-width: thick
}
h1 {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-style: normal;
      font-weight: bold;
      color: #333399;
      border-color: black #DFDFFF;
      width: auto;
      margin-right: 0px;
      margin-left: 0px;
      border: #DFDFFF;
      border-top-width: 0px;
      border-right-width: 10px;
      border-bottom-width: 0px;
      border-left-width: 10px ;
      font-size: 13px;
      background-color: #FFFFCC;
      padding-right: 10px;
      padding-left: 10px
}
h2 {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 13px;
      font-weight: bold;
      color: #CC0033;
      background-color: #FFE4CA;
      font-style: normal;
      padding-right: 10px;
      padding-left: 10px;
      border-color: black #FFE4CA;
      width: auto;
      margin-right: 0px;
      margin-left: 0px;
      border: #FFE4CA;
      border-top-width: 0px;
      border-right-width: 10px;
      border-bottom-width: 0px;
      border-left-width: 10px ;
      text-align: center
}
```

```
h3 {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 13px;
      font-style: normal;
      font-weight: bold;
      color: #CC0033;
      width: auto;
      border-width: 10px
}
h4 {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 13px;
      font-style: normal;
      font-weight: bold;
      color: #333399;
      background-color: #DFFAFB;
      padding-right: 10px;
      padding-left: 10px;
      width: auto;
      border: 1px solid #333399;
      padding-top: 2px;
      padding-bottom: 2px;
}
h5 {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 13px;
      font-style: normal;
      font-weight: bold;
      color: #FFFFCC;
      width: auto;
      border-width: 10px;
      background-color: #333399;
      padding-right: 10px;
      padding-left: 10px
}
h6 {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 13px;
      font-weight: bold;
      color: #CC0033;
      background-color: #FFE4CA;
      font-style: normal;
      width: auto;
      margin-right: 0px;
      margin-left: 0px;
      border: 1px solid #CC0033;
      padding: 10px;
}
b {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 11px;
      font-style: normal;
      font-weight: bold
}
```

```
.important1 {
      font-weight: bold;
      color: #FF0000
}
.important2 {
font-weight: bold
}
.different {
      font-style: italic
}
.inactif {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 11px;
      font-style: normal;
      font-weight: bold;
      color: #999999
}
.encart {
      background-color: #FFFFCC;
      color: #333399;
      padding-right: 20px;
      padding-left: 20px;
      padding-top: 20px;
      padding-bottom: 20px;
      font-weight: normal
}
.enum1 {
      list-style-image: url(images/puce.gif)
}
.retrait {
      margin-left: 20px
}
.justifie {
      text-align: justify
}
.grand {
      font-size: 24px;
      color: #999999
}
.blanc {
      color: #FFFFFF
}
.nouveau {
      color: #FFFFFF;
      font-weight: bolder;
```

```
      border: #FF0000;
      background-color: #FF0000;
      padding-top: 3px;
      padding-bottom: 3px;
      text-align: center;
      font-size: 13px
}
.titre {
      font-size: 15px;
      color: #333399;
      font-weight: bold;
      border-color: black black #333399;
      border-style: solid;
      border-top-width: 0px;
      border-right-width: 0px;
      border-bottom-width: 3px;
      border-left-width: 0px
}
.intertitre {
      font-size: 13px;
      color: #990000;
      font-weight: bold;
      border-color: black black #990033;
      border-style: solid;
      border-top-width: 0px;
      border-right-width: 0px;
      border-bottom-width: 1px;
      border-left-width: 0px
}
.evidence {
      color: #FFFFCC;
      background-color: #333399;
      padding-right: 20px;
      padding-top: 20px;
      padding-bottom: 20px;
      padding-left: 20px
}
.gris {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-style: normal;
      font-weight: bold;
      color: #999999;
      font-size: 11px
}
```

4.5.3.2 Des sous-modèles

Le principe des inclusions de fichiers que nous avons utilisé pour *entete.php* et *pieddepage.php* peut être étendu à d'autres fins comme, par exemple, l'ajout de boutons spécifiques, d'une barre de navigation,… de manière à créer d'autres modèles.

Il est clair que la notion de modèle telle que définie ici est beaucoup moins riche que la notion de modèle pouvant être implantée par des logiciels d'édition *HTML* sophistiqués. Le modèle est chargé pour créer la base du document et dès l'instant où ce document est enregistré sous un nom propre, différent de celui du

modèle, il n'y a plus aucune liaison avec ce dernier. Il y a donc un grand intérêt à réfléchir sérieusement, non seulement à l'élaboration de ces modèles, mais aussi à l'arborescence des fichiers, déterminante en ce qui concerne les liens relatifs décrivant les fichiers à inclure.

Dans notre exemple, nous avons créé deux autres modèles, pas très différents du modèle général, mais incluant des liens vers certaines pages importantes de l'application. Ces modèles sont liés aux interactions minimales que nous souhaitons dans le site, à savoir :

- l'utilisateur devra se connecter au site (à partir d'une page de *login*)
- l'utilisateur devra se déconnecter du site (ce qui ne sera possible qu'à partir de certaines pages seulement)
- l'utilisateur souhaitera régulièrement revenir à l'ensemble des fonctionnalités qui lui sont autorisées (également possible à partir de certaines pages)

Dès lors, nous envisageons un modèle qui inclut un lien de retour vers le menu (*modele_retour.php*) et un modèle qui inclut un lien pour la déconnexion (*modele_deconnexion.php*). Nous construisons, dans des fichiers séparés, la partie de code concernant l'ajout de ces liens. Les nouveaux modèles sont construits à partir du modèle basique.

Voici d'abord le code des fichiers modèles en commençant par *modele_retour.php* :

```
<?php include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
    <p><em>&lt;Corps de texte ici&gt;</em></p></td>
  </tr>
</table>
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

Ce modèle est construit à partir du code du modèle de base. On y a simplement ajouté l'inclusion d'un fichier *retour_menu.php* dont voici aussi le code :

```
<table width="555" border="0" cellspacing="2" cellpadding="5">
  <tr>
    <td> </td>
    <td><div align="right">Retour au <a href='menu.php?SID'>menu</a></div></td>
  </tr>
</table>
```

La structure de tableau à une cellule est à nouveau privilégiée pour y placer ce lien. Notez que celui-ci renvoie vers un script *menu.php* qui est encore à écrire. De plus, une valeur est fournie à ce script sous la forme de la constante *SID* qui n'est rien d'autre que l'identifiant de la session.

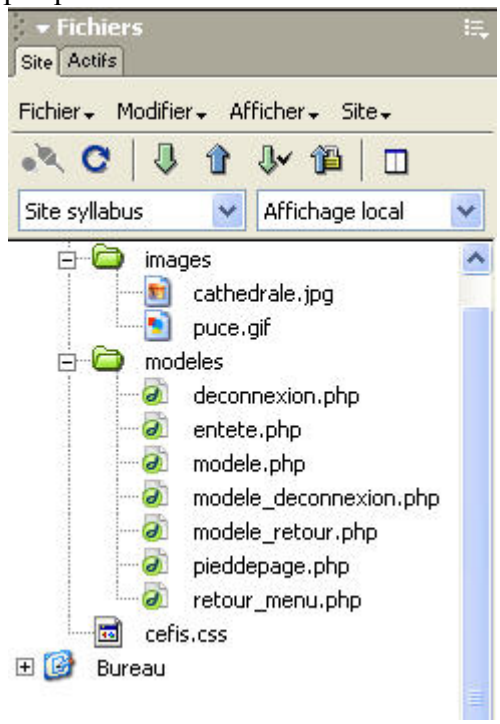
Voici l'autre sous-modèle *modele_deconnexion.php*. Sa construction est identique.

```
<?php include('../modeles/entete.php'); ?>
```

```
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h1>Système de gestion des stages</h1>
    <p><em>&lt;Corps de texte ici&gt;</em></p></td>
  </tr>
</table>
<?php
include('../modeles/deconnexion.php');
include('../modeles/pieddepage.php');
?>
```

Le fichier inclus, *deconnexion.php*, n'est composé que d'une seule ligne dans le cas présent. Il est donc peu utile d'en faire un fichier à inclure. Toutefois, c'est le principe que nous souhaitons illustrer ici. Il contient un lien vers le fichier *login.php* dont le code est évidemment à écrire.

```
<p><a href="login.php">Déconnecter</a></p>
```



À ce stade du développement de l'application, l'arborescence du site se présente comme suit.

Un dossier *images* contient les images du site, en ce comprises, les images correspondant à des puces qui auraient pu être définies dans les styles.

Un dossier *modeles* contient le modèle de base, les deux sous-modèles et les quatre fichiers pour les inclusions de code.

Le fichier contenant la définition de la feuille de style est resté au niveau de la racine du site.

Il reste maintenant à construire les différents scripts sans trop avoir à se préoccuper de la mise en page. Chaque fichier sera construit à partir d'un des deux sous-modèles. Dans une conception plus sophistiquée, on pourrait avoir une réflexion plus approfondie permettant de raffiner davantage les modèles et donc la mise en page et en forme.

4.5.4 La connexion

Le point de départ de l'application est une page de connexion. Celle-ci est généralement sobre et se limite à la demande de fourniture d'un identifiant et d'un mot de passe. Elle peut proposer d'autres services tels :

- la possibilité de créer soi-même son identifiant et son mot de passe (utilité à estimer en fonction de la visibilité du site - Internet ou intranet - et la capacité à gérer les inscriptions bidons)
- la possibilité de renvoyer le mot de passe oublié à un utilisateur sur base de renseignements qu'il aurait préalablement fournis
- …

Nous avons considéré, dans notre exemple, que chaque utilisateur recevait un identifiant et un mot de passe de la part de l'administrateur. On peut fournir un principe simple de construction des identifiants

(par exemple, la première lettre du prénom suivie des lettres du nom à concurrence de huit caractères) et attribuer la même information comme mot de passe en demandant à l'utilisateur d'en changer dès la première connexion. Cette méthode présente toutefois l'inconvénient que certains utilisateurs ne changeront jamais leur mot de passe, permettant à d'autres qui connaissent la manière de les construire, de s'en servir de manière peu loyale.

4.5.4.1 Login

La page de *login* est un formulaire au sens *HTML* du terme.

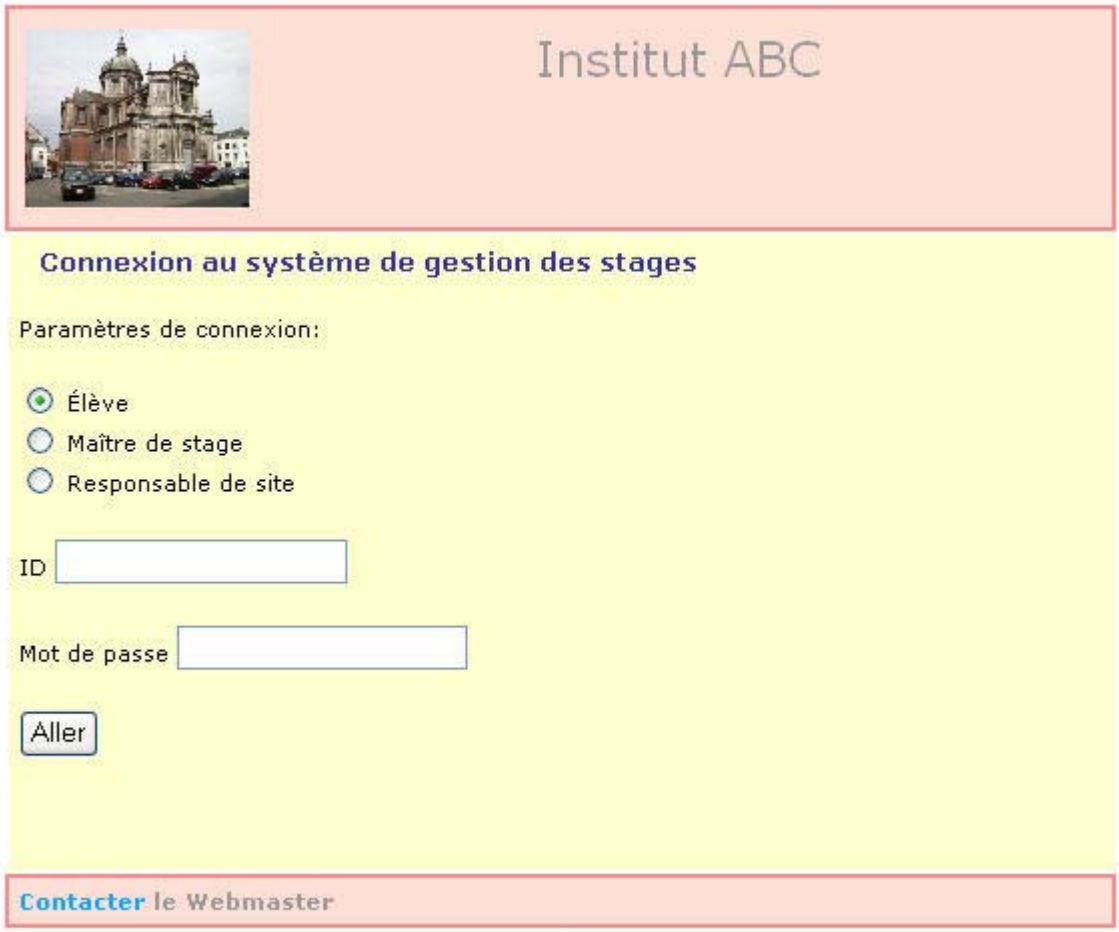
```
<?php include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC"><h6>Connexion au syst&egrave;me
        de gestion des stages</h6>
      <form name="form1" method="post" action="connexion.php">
        <p>
          <label> Param&egrave;tres de connexion:</label>
        </p>
        <p>
          <label>
          <input name="categorie" type="radio" value="1" checked>
          Él&egrave;ve</label>
          <br>
          <label>
          <input type="radio" name="categorie" value="2">
          Maître de stage</label>
          <br>
          <label>
          <input type="radio" name="categorie" value="3">
          Responsable de site</label>
        </p>
        <p>ID
          <input type="text" name="login">
        </p>
        <p>Mot de passe
          <input type="password" name="pw">
          <label></label>
        </p>
        <p>
          <input type="submit" name="Submit" value="Aller">
        </p>
      </form>
```

```
      <p> </p></td>
  </tr>
</table>
<?php include('../modeles/pieddepage.php'); ?>
```

Cette page est construite à partir du modèle basique (sans lien de retour ou de déconnexion). Le formulaire se compose d'un groupe de boutons radios, de deux zones de texte pour la saisie de l'identifiant et du mot de passe et d'un bouton de soumission.

Il n'y a pas de contrôle de format et de validation prévus ici pour éviter la prolifération de code, mais il est certain que des scripts *JavaScript* sont en mesure de gérer ces problèmes potentiels.

Le groupe de boutons sert à identifier la catégorie d'utilisateur à laquelle on a affaire. De la sorte, le script associé à la soumission (*connexion.php*) pourra transmettre cette information de manière à fournir des menus en rapport avec cette catégorie.



4.5.4.2 Connexion

Le script[28] précédent provoque (lorsque le bouton de soumission du formulaire est activé) l'exécution du script *connexion.php*. Celui se contente de sélectionner la bonne table pour la vérification de la correspondance identifiant - mot de passe et de proposer des alternatives en cas d'erreur. Si aucune erreur n'est détectée, un lien permet de continuer. Ce lien fait exécuter le script *menu.php* tout en fournissant l'identificateur de session.

---

[28] Le mot *script* est utilisé dans le sens large de partie de code contenant des instructions en *PHP*.

```
<?php
session_start();
include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
<?php
$_SESSION['categorie']=$_POST['categorie'];
$_SESSION['login']=$_POST['login'];
$_SESSION['pw']=$_POST['pw'];
echo "<h6>Système de gestion des stages</h6>";
$c=$_POST['categorie'];
if($c==1){$t='eleve';}
elseif($c==2){$t='maitre';}
else{$t='lieu';}
$link = mysql_connect('localhost', 'encodeur', 'aaaaaa');
if (!$link) {
   die('<p>Impossible de se connecter : '. mysql_error().'</p>');
}
$db = mysql_select_db('gestionstages', $link);
if (!$db) {
   die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
}
$login=$_POST['login'];
$sqlquery="select nom,prenom,login,pw from $t where login='$login'";
$queryresult=mysql_query($sqlquery);
if(mysql_num_rows($queryresult)==0){
      echo "<p>Identifiant inconnu!</p>";
      echo "<p><a href='login.php'>Retour</a></p>";}
else{
      $row=mysql_fetch_array($queryresult,MYSQL_ASSOC);
      if($row['pw']!=$_POST['pw']){
            echo "Mot de passe incorrect!";
            echo "<p><a href='login.php'>Retour</a></p>";}
      else{echo "<p>Bienvenue <strong>".$row['prenom']." ".$row['nom']."</strong>
"."</p>";
            echo "<p><a href='menu.php?".SID."'>Continuer</a></p>";
        include('../modeles/deconnexion.php');
      }
}
?>
```

```
  </tr>
</table>
<?php
include('../modeles/pieddepage.php'); ?>
```

La création d'une session et la génération d'un identifiant de session se fait grâce à la fonction *session_start*. Cette instruction doit être la première si on utilise la technique des cookies pour retrouver les variables de session.

Les valeurs fournies par le formulaire (variable `$_POST`) sont placées dans le tableau global prédéfini `$_SESSION`, de façon à pouvoir être retrouvées grâce à l'identifiant de session `SID`.

Comme ce script doit également identifier l'utilisateur, la page à produire pourrait ne pas comprendre de retour au menu, ni de possibilité de déconnexion. C'est la raison pour laquelle cette page est créée à partir du modèle basique. On a ajouté une instruction générant un lien de retour à la page précédente et l'instruction permettant la déconnexion, uniquement dans les cas concernés par ces options. Les autres pages seront toutes créées sur base d'un des deux sous-modèles.

En cas d'identification, le script appelé est *menu.php*. Ce script va générer un menu qui va dépendre du type d'utilisateur connecté. L'identificateur de session est fourni dans l'adresse (ce qui est nécessaire si le client web de l'utilisateur est configuré pour ne pas accepter les cookies).

4.5.5 Les scripts

4.5.5.1 Menu

Le script contient la même instruction de démarrage de session qui permet de récupérer, si nécessaire, les variables de session. La page est construite sur le modèle *modele_deconnexion.php*. La variable `$t` contiendra le nom de la table à sélectionner. Ce nom est évidemment fonction de la catégorie. Le menu est également constitué sur base de la catégorie de l'utilisateur. Les adresses de chacun des liens sont suivies de l'identifiant de session qui permet de propager, notamment, la catégorie de l'utilisateur, mais aussi d'autres informations.

```
<?php
session_start();
include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
<?php
$c=$_SESSION['categorie'];
if($c==1){$t='eleve';}
```

```
elseif($c==2){$t='maitre';}
else{$t='lieu';}
$link = mysql_connect('localhost', 'encodeur', 'aaaaaa');
if (!$link) {
   die('<p>Impossible de se connecter : '. mysql_error().'</p>');
}
$db = mysql_select_db('stages', $link);
if (!$db) {
   die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
}
$login=$_SESSION['login'];
$sqlquery="select nom,prenom,login,pw from $t where login='$login'";
$queryresult=mysql_query($sqlquery);
$row=mysql_fetch_array($queryresult,MYSQL_ASSOC);
echo "<p>Session de <strong>".$row['prenom']." ".$row['nom']."</strong> "."</p>";
echo "<h1>Faites votre choix</h1>";
if($c==1){
      echo "<p>Obtenir la liste des lieux et dates de vos stages <a
href='stages_eleve.php?SID'>Ici</a></p>";
      echo "<p>Savoir où vous êtes en stage à une date donnée <a
href='stages_par_date.php?SID'>Ici</a></p>";
      echo "<p>Obtenir la liste des superviseurs <a
href='superviseurs.php?SID'>Ici</a>";
      echo "<p>Obtenir la liste des étudiants présents au même endroit un même jour
<a href='date_stage.php?SID'>Ici</a>";
      echo "</p>";
}
if($c==2){
      echo "<p>Obtenir la liste des élèves que vous supervisez <a
href='supervision.php?SID'>Ici</a></p>";
      echo "<p>Ajouter un élève <a href='ajouter_eleve.php?SID'>Ici</a><p>";
      echo "<p>Ajouter un stage à un élève <a
href='ajouter_stage.php?SID'>Ici</a><p>";
}
if($c==3){ // à développer
}
?>
  </tr>
</table>
<?php
include('../modeles/deconnexion.php');
include('../modeles/pieddepage.php'); ?>
```

Voici l'interface de menu des élèves.

Le menu des maîtres de stage ressemblera à ce qui suit.

Chacun des sept scripts mentionnés dans cette page vont être décrits. Nous le ferons en respectant une certaine progression dans la difficulté. C'est la raison pour laquelle nous décrivons d'abord le script pour la construction d'une page destinée aux élèves avant de traiter les pages pour les maîtres de stage et de revenir aux pages destinées aux élèves.

4.5.5.2 Liste des superviseurs (élève)

Il s'agit d'une requête sélection relativement élémentaire et ne s'adressant qu'à une seule table, celle des maîtres de stage. La requête est :

```
select nom,prenom from maitre order by nom ;
```

Le reste du code est classique si l'on s'en réfère aux exemples qui ont déjà été donnés.

```
<?php
session_start();
include('../modeles/entete.php');
?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
```

```
      <h6>Système de gestion des stages</h6>
      <?php
      echo "<h1>Liste des superviseurs</h1>";
      $link = mysql_connect('localhost', 'encodeur', 'aaaaaa');
      if (!$link) {
      die('<p>Impossible de se connecter : '. mysql_error().'</p>');
      }
      $db = mysql_select_db('stages', $link);
      if (!$db) {
       die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
      }
      $sqlquery="select nom,prenom from maitre order by nom";
      $queryresult=mysql_query($sqlquery);
      echo "<table><tr><td><strong>PRÉNOM</strong></td><td><strong>NOM</strong>
</td></tr>";
      while($row=mysql_fetch_array($queryresult,MYSQL_ASSOC)){
      echo "<tr><td>".$row['prenom']."</td><td>".$row['nom']."</td></tr>";
      }
      echo "</table>";
      ?>
      </td>
  </tr>
</table>
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

Ce qui donne, par exemple :

4.5.5.3 Liste des supervisés (maître de stage)

La requête est un peu plus compliquée dès lors qu'elle travaille sur les tables *eleve*, *maitre* et *stage*. Il faut que dans la liste des stages, l'identifiant du maître soit celui de l'utilisateur et que les identifiants du maître et des élèves correspondent dans les tables *eleve* et *maitre*.

```
$sqlquery="select eleve.nom,eleve.prenom,stage.domaine from eleve,stage,maitre where
id_eleve=eleve.id && id_maitre=maitre.id && maitre.login='".$_SESSION['login']."'";
```

Ce qui donne quelque chose comme ceci :

4.5.5.4 Ajout d'un élève (maître de stage)

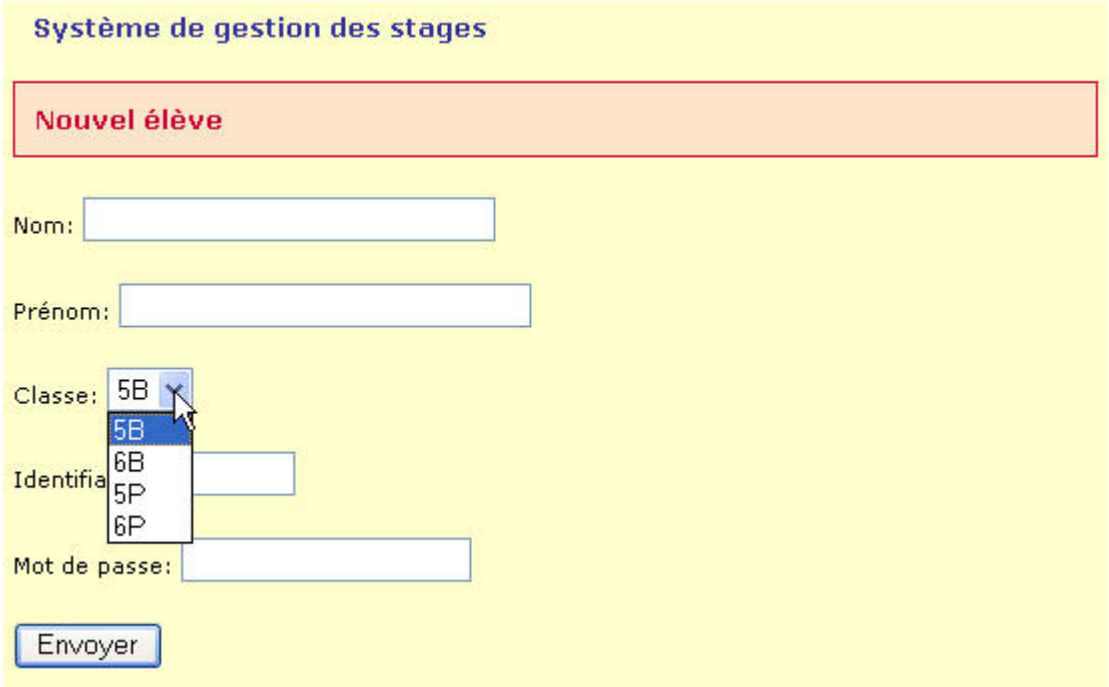
Voici un script dont l'effet ne sera pas visible. Il s'agit, en effet, de modifier le contenu de la base de données. D'où l'importance de fournir un feed-back à l'utilisateur lui indiquant comment se sont passées les opérations. C'est le rôle de la page qui sera générée.

Mais avant l'exécution de ce script, l'utilisateur doit fournir les données nécessaires à la requête de mise à jour de la base de données. Il y a donc deux fichiers à construire dans ce cas.

```
<?php include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC"> <h6>Syst&egrave;me de gestion
        des stages </h6>

      <h1>Nouvel &eacute;l&egrave;ve </h1>
      <form name="form1" method="post" action="fb_enregistrer_eleve.php">
        <p>Nom:
          <input name="nom" type="text" size="30" maxlength="50" >
        </p>
        <p>Pr&eacute;nom:
          <input name="prenom" type="text" id="prenom" size="30" maxlength="50">
        </p>
        <p>Classe:
          <select name="classe" size="1" id="classe">
            <option value="5G">5B</option>
            <option value="6G">6B</option>
```

```
            <option value="5P">5P</option>
            <option value="6P">6P</option>
          </select>
        </p>
        <p>Identifiant:
          <input name="login" type="text" id="login" size="8" maxlength="8">
        </p>
        <p>Mot de passe:
          <input name="pw" type="password" id="pw" size="20" maxlength="50">
        </p>
        <p>
          <input name="envoyer" type="submit" id="envoyer" value="Envoyer">
        </p>
        <p>  </p>
      </form></td>
  </tr>
</table>
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

Voici le formulaire :



La soumission du formulaire entraînera l'exécution du script *fb_enregister_eleve.php*. Observez encore que le choix de la classe se fait dans une liste fixe. On pourrait imaginer que le nombre de classes soit

variable dans le temps. Il faudrait alors en faire une entité dans le modèle *ERA*. Il serait commode, dans ce cas, de générer la liste sur base d'une requête dans la base de données.

Voici le code de ce script incluant les instructions d'enregistrement dans la base et les instructions d'affichage du feed-back.

```
<?php include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
      <h1>R&eacute;sultat de l'enregistrement</h1>
        <p>
        <?php
      $link = mysql_connect('localhost', 'encodeur', 'aaaaaa');
        if (!$link) {
      die('<p>Impossible de se connecter : '. mysql_error().'</p>');
        }
        $db = mysql_select_db('stages', $link);
        if (!$db) {
        die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
        }
        foreach($_POST as $key => $value) {
       $varname = "_".$key;
       $$varname = $value;
        }
        $sqlquery="insert eleve (nom,prenom,classe,login,pw) values
('$_nom','$_prenom','$_classe','$_login','$_pw')";
        $queryresult=mysql_query($sqlquery) or die("Échec de l'enregistrement");
        echo "L'élève $_prenom $_nom a été enregistré correctement."
        ?>
        </p>
</td>
  </tr>
</table>
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

La requête est une requête de mise à jour :

```
$sqlquery="insert eleve (nom,prenom,classe,login,pw) values ('$_nom','$_prenom',
'$_classe','$_login','$_pw')";
```

Les noms des variables ont été générés dynamiquement à l'aide d'une instruction `foreach` comme nous en avions déjà donné un exemple.

Le feed-back est donné par la seule instruction :

```
echo "L'élève $_prenom $_nom a été enregistré correctement." ;
```

Et voici l'écran donnant ce feed-back :

Chacun des scripts que nous venons de décrire affiche évidemment un lien permettant un retour au menu, ce que les copies d'écran ne montrent pas.

4.5.5.5 Ajout d'un stage (maître de stage)

Un maître de stage peut encoder un nouveau stage pour un élève. Nous ferons les mêmes observations que pour le cas précédent, à savoir que deux fichiers sont nécessaires : un pour générer un formulaire, l'autre pour enregistrer les données et fournir le feed-back.

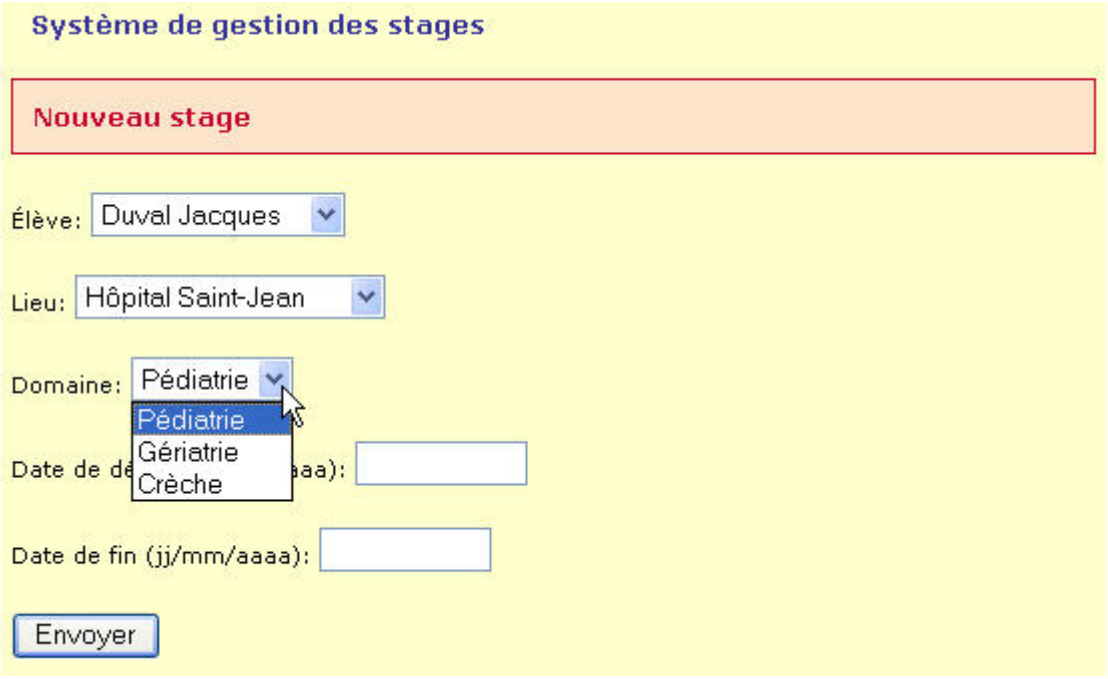
La particularité de ce script est que certaines données vont être extraites de la base de données pour faciliter les choix de l'utilisateur. C'est le cas de la liste des élèves, des lieux et des domaines. On aurait également pu associer un petit calendrier pour le choix des dates. Tous ces choix sont évidemment discutables.

```
<?php include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
<?php
$link = mysql_connect('localhost', 'encodeur', 'aaaaaa');
if (!$link) {
   die('<p>Impossible de se connecter : '. mysql_error().'</p>');
}
$db = mysql_select_db('stages', $link);
if (!$db) {
   die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
}
$sqlquery1="select nom,prenom,id from eleve";
$queryresult1=mysql_query($sqlquery1);
$sqlquery2="select denomination,id from lieu";
$queryresult2=mysql_query($sqlquery2);
?>
```

```
<h1>Nouveau stage</h1>
<form name="form1" method="post" action="fb_ajouter_stage.php">
<p>&Eacute;l&egrave;ve:
<select name="id_eleve" size="1">
<?php
while($row=mysql_fetch_array($queryresult1,MYSQL_ASSOC)){
      echo"<option value='".$row['id']."'>".$row['nom']." ".$row['prenom'].
"</option>";
}
?>
</select>
</p>
<p>Lieu:
<select name="lieu" size="1" id="lieu">
<?php
while($row=mysql_fetch_array($queryresult2,MYSQL_ASSOC)){
      echo"<option value='".$row['id']."'>".$row['denomination']."</option>";
}
?>
</select>
</p>
<p>Domaine:
<select name="domaine" size="1" id="domaine">
<option value="P&eacute;diatrie">P&eacute;diatrie</option>
<option value="G&eacute;riatrie">G&eacute;riatrie</option>
<option value="Cr&egrave;che">Cr&egrave;che</option>
</select>
</p>
<p>Date de d&eacute;but (jj/mm/aaaa):
<input name="date_debut" type="text" id="date_debut" size="10" maxlength="10">
</p>
<p>Date de fin (jj/mm/aaaa):
<input name="date_fin" type="text" id="date_fin" size="10" maxlength="10">
</p>
<p>
<input name="envoyer" type="submit" id="envoyer" value="Envoyer">
</p>
<p>  </p>
</form></p></td>
</tr>
</table>
```

```
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

Voici un aperçu de ce formulaire :



L'envoi de ce formulaire active le script *fb_ajouter_stage.php* dont voici le code :

```
<?php
session_start();
include('../modeles/entete.php');
?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
      <h1>R&eacute;sultat de l'enregistrement</h1>
      <p>
        <?php
      $link = mysql_connect('localhost', 'encodeur', 'aaaaaa');
        if (!$link) {
      die('<p>Impossible de se connecter : '. mysql_error().'</p>');
        }
        $db = mysql_select_db('gestionstages', $link);
        if (!$db) {
        die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
        }
```

```
        foreach($_POST as $key => $value) {
       $varname = "_".$key;
       $$varname = $value;
        }

$_date_debut=substr($_date_debut,6,4).'/'.substr($_date_debut,3,2).'/'.substr($_date_
debut,0,2);

$_date_fin=substr($_date_fin,6,4).'/'.substr($_date_fin,3,2).'/'.substr($_date_fin,0,
2);
        $sqlquery1="select id from maitre where login='".$_SESSION['login']."'";
        $queryresult1=mysql_query($sqlquery1) or die('La requête a échoué');
        $row=mysql_fetch_array($queryresult1,MYSQL_ASSOC);
        $sqlquery2="insert stage (domaine,date_debut,date_fin,id_eleve,id_lieu,
id_maitre) values ('$_domaine','$_date_debut','$_date_fin','$_id_eleve','$_lieu',
'".$row['id']."')";
        $queryresult2=mysql_query($sqlquery2) or die("L'enregistrement a échoué");
        echo "Le stage a été enregistré correctement.";
        ?>
        </p></td>
  </tr>
</table>
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

Deux requêtes sont effectuées :

la première est une requête sélection pour connaître l'identifiant du maître de stage (nécessaire pour créer le nouveau stage) ;

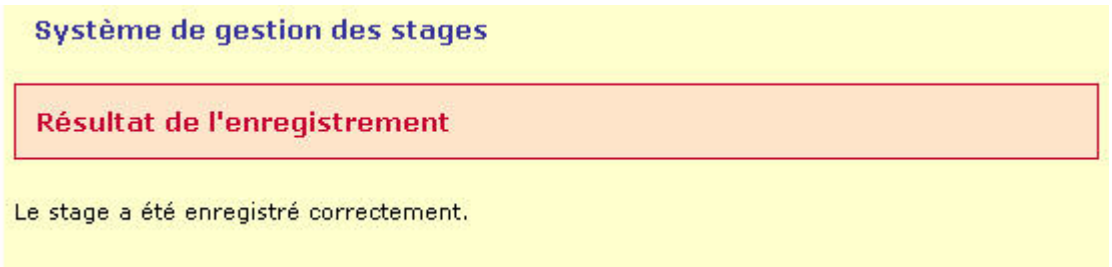
```
$sqlquery1="select id from maitre where login='".$_SESSION['login']."'";
```

la seconde est une requête de mise à jour pour l'insertion du nouveau stage.

```
$sqlquery2="insert   stage   (domaine,date_debut,date_fin,id_eleve,id_lieu,id_maitre)
values('$_domaine','$_date_debut','$_date_fin','$_id_eleve','$_lieu','".$row['id']
."')";
```

Comme dans le cas précédent, le feed-back est relativement sobre.

```
echo "Le stage a été enregistré correctement.";
```



4.5.5.6 Lieux et dates de stage (élève)

Les requêtes sélections ne sont pas très compliquées, si ce n'est que l'une d'elles porte sur les tables *eleve* et *stage*.

```
<?php
session_start();
include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
      <h1>Liste des stages &agrave; prester</h1>
      <p>
        <?php
      $link = mysql_connect('localhost', 'encodeur', 'aaaaaa');
        if (!$link) {
      die('<p>Impossible de se connecter : '. mysql_error().'</p>');
        }
        $db = mysql_select_db('gestionstages', $link);
        if (!$db) {
        die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
        }
        $sqlquery1="select stage.domaine,stage.date_debut,stage.date_fin,
stage.id_lieu,stage.id_eleve from eleve,stage where stage.id_eleve='".
$_SESSION['login']."' group by stage.domaine";
        $queryresult1=mysql_query($sqlquery1) or die('La requête 1 a échoué');
        echo "<table><tr><td><strong>Lieu</strong></td><td><strong>Domaine</strong>
</td><td><strong>Date de début</strong></td><td><strong>Date de fin</strong>
</td></tr>";
        while($row=mysql_fetch_array($queryresult1)){
        $sqlquery2="select denomination from lieu where login='$row[3]'";
        $queryresult2=mysql_query($sqlquery2) or die('La requête 2 a échoué');
        $row2=mysql_fetch_array($queryresult2);
        $row[1]=substr($row[1],8,2).'/'.substr($row[1],5,2).'/'.substr($row[1],0,4);
        $row[2]=substr($row[2],8,2).'/'.substr($row[2],5,2).'/'.substr($row[2],0,4);
        echo "<tr><td>$row2[0]</td><td>$row[0]</td><td>$row[1]</td><td>$row[2]</td>
</tr>";
        }
        echo "</table>";
        ?>
        </p></td>
  </tr>
</table>
```

```
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

La première requête :

```
$sqlquery1="select stage.domaine,stage.date_debut,stage.date_fin,
stage.id_lieu,stage.id_eleve from eleve,stage where stage.id_eleve='".
$_SESSION['login']."' group by stage.domaine";
```

permet de retrouver tous les stages de l'élève connecté.

La seconde requête sert à retrouver les dénominations des lieux en fonction de leur identifiant dans la table des stages.

```
$sqlquery2="select denomination from lieu where login='$row[3]'";
```

Voici un résultat possible :

4.5.5.7 Stages à date donnée (élève)

La recherche des stages d'un élève à une date donnée demande à nouveau deux fichiers :

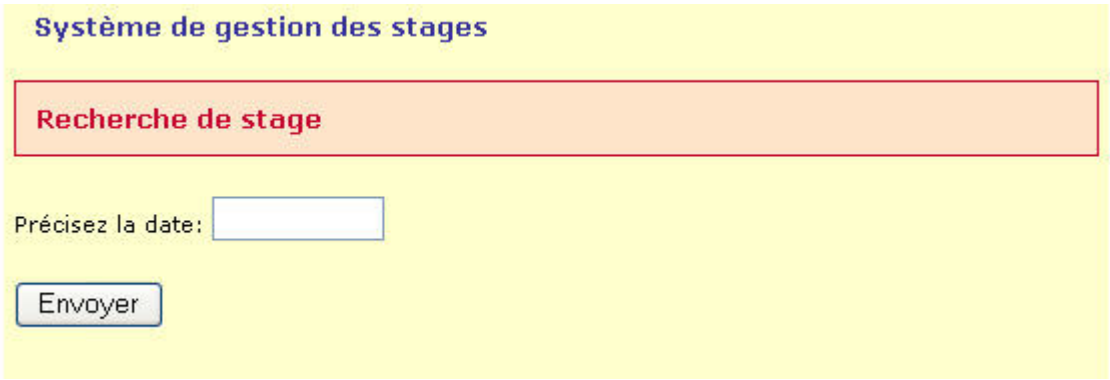
- un formulaire pour préciser cette date
- un script pour la recherche dans la base de données et l'affichage des résultats

La soumission du formulaire provoquera l'exécution de ce script appelé ici *verifier_stage.php*.

```
<?php
session_start();
include('../modeles/entete.php');
?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
      <h1>Recherche de stage</h1>
      <form name="form1" method="post" action="verifier_stage.php">
        <p>Précisez la date:
          <input name="jour_de_stage" type="text" size="10" maxlength="10">
```

```
        </p>
        <p>
          <input type="submit" name="Submit" value="Envoyer">
        </p>
      </form>
      </td>
  </tr>
</table>
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

Voici ce que donne à l'affichage ce formulaire :



Le code du script *verifier_stage.php* est le suivant :

```
<?php
session_start();
include('../modeles/entete.php');
?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
      <h1>En stage &agrave;...</h1>
      <p>
<?php
      $link = mysql_connect('localhost', 'encodeur', 'aaaaaa');
        if (!$link) {
      die('<p>Impossible de se connecter : '. mysql_error().'</p>');
        }
        $db = mysql_select_db('stages', $link);
        if (!$db) {
        die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
```

```
        }
        $jds=$_POST['jour_de_stage'];
        $jds=substr($jds,6,4).'/'.substr($jds,3,2).'/'.substr($jds,0,2);
        $sqlquery1="select stage.domaine,stage.date_debut,stage.date_fin from
eleve,stage where stage.id_eleve='".$_SESSION['login']."' && '".$jds."'>
stage.date_debut && '".$jds."'<stage.date_fin";
        $queryresult1=mysql_query($sqlquery1) or die('La requête a échoué');
        echo "<table><tr><td><strong>Lieu</strong></td><td><strong>Domaine</strong>
</td><td><strong>Date de début</strong></td><td><strong>Date de fin</strong></td>
</tr>";
        while($row=mysql_fetch_array($queryresult1)){
        $row[1]=substr($row[1],8,2).'/'.substr($row[1],5,2).'/'.substr($row[1],0,4);
        $row[2]=substr($row[2],8,2).'/'.substr($row[2],5,2).'/'.substr($row[2],0,4);
        $sqlquery2="select denomination from lieu where id='$row[3]'";
        $queryresult2=mysql_query($sqlquery2) or die('La requête a échoué');
        $row2=mysql_fetch_array($queryresult2);
        echo "<tr><td>$row2[0]</td><td>$row[0]</td><td>$row[1]</td><td>$row[2]
</td></tr>";
        }
        echo "</table>";
?>
        </p></td></tr>
</table>
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

La première requête recherche tous les stages correspondant à l'utilisateur dont l'intervalle de déroulement inclut la date fournie.

```
$sqlquery1="select stage.domaine,stage.date_debut,stage.date_fin from eleve,stage
where stage.id_eleve='".$_SESSION['login']."' && '".$jds."'> stage.date_debut &&
'".$jds."'<stage.date_fin";
```

La seconde requête sert à déterminer la dénomination du lieu.

```
$sqlquery2="select denomination from lieu where id='$row[3]'";
```

Voici un exemple de résultat :

4.5.5.8 Liste des accompagnants (élève)

Pour cette dernière fonctionnalité décrite, deux fichiers sont également nécessaires. Le premier est semblable à celui de l'exemple précédent et génère un formulaire pour la saisie d'une date. Rappelons qu'aucun script ne valide ici le format et la cohérence des dates.

```
<?php
include('../modeles/entete.php');
?>

<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
      <h1>Date de stage</h1>
      <form name="form1" method="post" action="accompagnants.php">
        <p>Date (jj/mm/aaaa):
          <input name="date_de_stage" type="text" id="date_de_stage">
        </p>
        <p>
          <input name="envoyer" type="submit" id="envoyer" value="Envoyer">
        </p>
      </form>
      <p></p></td>
  </tr>
</table>

<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

Le look du formulaire est semblable au précédent. Sa soumission provoque l'exécution du script *accompagnants.php* dont voici le code :

```
<?php
session_start();
include('../modeles/entete.php'); ?>
<table width="556" border="0" cellspacing="2" cellpadding="10">
  <tr>
    <td height="300" valign="top" bgcolor="#FFFFCC">
      <h6>Système de gestion des stages</h6>
      <h1>Liste des &eacute;tudiants en stage au m&ecirc;me endroit ce jour</h1>
      <p>
<?php
```

```
        $jds=$_POST['date_de_stage'];
        $jds=substr($jds,6,4).'/'.substr($jds,3,2).'/'.substr($jds,0,2);
        $link = mysql_connect('localhost', 'encodeur', 'aaaaaa');
        if (!$link) {
      die('<p>Impossible de se connecter : '. mysql_error().'</p>');
        }
        $db = mysql_select_db('gestionstages', $link);
        if (!$db) {
        die ('<p>Impossible d\'utiliser la base : '. mysql_error().'</p>');
        }
        $sqlquery1="select lieu.denomination,lieu.login from stage,eleve,lieu where
stage.id_eleve='".$_SESSION['login']."' && stage.date_debut<'".$jds."' &&
stage.date_fin>'".$jds."' && stage.id_lieu=lieu.login group by lieu.denomination";
        $queryresult1=mysql_query($sqlquery1) or die('La requête 1 a échoué');
        while($row=mysql_fetch_row($queryresult1)){
        echo "<table><tr><td><strong>Lieu</strong></td></tr>";
        echo "<tr><td>$row[0]</td></tr>";
        echo "<tr><td><strong>Nom des présents ce jour</strong></td></tr>";
        $sqlquery2="select eleve.nom from eleve,stage where stage.id_lieu=
'".$row[1]."' && eleve.login=stage.id_eleve && stage.date_debut<'".$jds."' &&
stage.date_fin>'".$jds."' group by eleve.nom order by eleve.nom";
        $queryresult2=mysql_query($sqlquery2) or die('La requête 2 a échoué');
        while($row2=mysql_fetch_row($queryresult2)){
        echo "<tr><td>$row2[0]</td></tr>";
        }
        }
        echo "</table>";
?>
        </p>
      </td>
  </tr>
</table>
<?php
include('../modeles/retour_menu.php');
include('../modeles/pieddepage.php');
?>
```

La première requête permet de trouver les lieux de stage de l'utilisateur à la date fournie :

```
$sqlquery1="select lieu.denomination,lieu.login from stage,eleve,lieu where
stage.id_eleve='".$_SESSION['login']."' && stage.date_debut<'".$jds."' &&
stage.date_fin>'".$jds."' && stage.id_lieu=lieu.login group by lieu.denomination";
```

La seconde requête permet de trouver la liste des élèves en stage au même endroit à cette date :

```
$sqlquery2="select eleve.nom from eleve,stage where stage.id_lieu= '".$row[1]."' &&
eleve.login=stage.id_eleve && stage.date_debut<'".$jds."' &&
stage.date_fin>'".$jds."' group by eleve.nom order by eleve.nom";
```

Voici un exemple de résultat:

Liste des étudiants en stage au même endroit ce jour

Lieu

Crèche Les Poussins

Nom des présents ce jour

Devreux

Duval

4.6 La base de données

Voici, à titre documentaire, la structure des quatre tables de la base de données et leur contenu.

4.6.1 La table eleve

4.6.1.1 Structure

Base de données *gestionstages* - Table *eleve* sur le serveur *localhost*

Structure | Afficher | SQL | Sélectionner | Insérer | Exporter

	Champ	Type	Attributs	Null	Défaut	Extra	Action
☐	nom	varchar(50)		Non			
☐	prenom	varchar(50)		Non			
☐	login	varchar(8)		Non			
☐	pw	varchar(30)		Non			
☐	classe	enum('5G', '6G', '5P', '6P')		Non	5G		

4.6.1.2 Contenu

←T→		nom	prenom	login	pw	classe
		Duval	Jacques	jdu	aaaaaa	5G
		Devreux	Ludovic	lde	cccccc	6G
		Martin	Luc	lma	eeeeee	6G
		Dethy	Marc	mde	dddddd	5P
		Lemoine	Pierre	ple	bbbbbb	5G

4.6.2 La table maitre

4.6.2.1 Structure

Base de données *gestionstages* - Table *maitre* sur le serveur *localhost*

Structure | Afficher | SQL | Sélectionner | Insérer | Exporter

	Champ	Type	Attributs	Null	Défaut	Extra	Action
☐	nom	varchar(50)		Non			
☐	prenom	varchar(50)		Non			
☐	login	varchar(8)		Non			
☐	pw	varchar(30)		Non			

4.6.2.2 Contenu

←T→		nom	prenom	login	pw
		Miller	François	fmi	333333
		Verbois	Jacques	jve	111111
		Desmedt	Paul	pde	222222
		Morel	Vincent	vmo	444444

4.6.3 La table lieu

4.6.3.1 Structure

Base de données *gestionstages* - Table *lieu* sur le serveur *localhost*

Structure | Afficher | SQL | Sélectionner | Insérer | Exporter

	Champ	Type	Attributs	Null	Défaut	Extra	Action
☐	denomination	varchar(50)		Non			
☐	adresse	varchar(50)		Non			
☐	nom	varchar(50)		Non			
☐	prenom	varchar(50)		Non			
☐	login	varchar(8)		Non			
☐	pw	varchar(30)		Non			

4.6.3.2 Contenu

←T→		denomination	adresse	nom	prenom	login	pw
		Crèche Les Poussins	avenue des Arts Namur	Herman	Josiane	jhe	c3c3c3
		Hôpital Saint-Jean	rue des Haies Namur	Renders	Michel	mre	a1a1a1
		Home Les Magnolias	rue du Centre Jambes	Falise	Roger	rfa	b2b2b2

4.6.4 La table stage

4.6.4.1 Structure

Base de données *gestionstages* - Table *stage* sur le serveur *localhost*

Structure | Afficher | SQL | Sélectionner | Insérer | Exporter | Opérations

	Champ	Type	Attributs	Null	Défaut	Extra	Action
☐	id	tinyint(5)		Non		auto_increment	
☐	domaine	varchar(50)		Non			
☐	date_debut	date		Non	0000-00-00		
☐	date_fin	date		Non	0000-00-00		
☐	note	tinyint(2)	UNSIGNED	Oui	*NULL*		
☐	id_eleve	varchar(8)		Non	0		
☐	id_lieu	varchar(8)		Non	0		
☐	id_maitre	varchar(8)		Non	0		

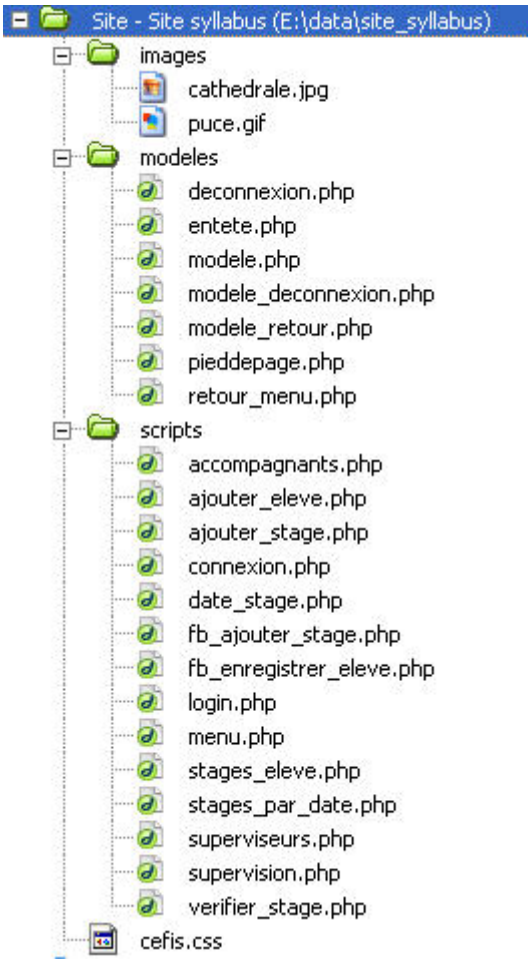
4.6.4.2 Contenu

←T→		id	domaine	date_debut	date_fin	note	id_eleve	id_lieu	id_maitre
		1	Crèche	2005-05-15	2005-05-30	*NULL*	mde	jhe	jve
		2	Pédiatrie	2005-05-10	2005-05-28	*NULL*	ple	mre	fmi
		3	Gériatrie	2005-05-12	2005-06-02	*NULL*	lde	rfa	fmi
		4	Gériatrie	2005-05-15	2005-06-04	*NULL*	ple	rfa	fmi
		6	Pédiatrie	2005-05-11	2005-06-04	*NULL*	jdu	mre	jve
		9	Crèche	2005-06-06	2005-06-13	*NULL*	lde	jhe	fmi
		8	Crèche	2005-06-06	2005-06-13	*NULL*	jdu	jhe	jve
		10	Gériatrie	2005-05-23	2005-05-29	*NULL*	jdu	rfa	jve

4.7 Structure du site

Pour terminer, voici ce que donne la structure du mini-site de notre application en termes de dossiers et de fichiers :

Constatez encore une fois que la structure d'un tel site doit être connue avant la rédaction du moindre script. L'usage des modèles, les références qu'ils font à d'autres fichiers dont le code doit être inclus, tous ces éléments font que chaque fichier doit être affecté à un dossier, avant même d'avoir été construit.



D'autres répartitions des fichiers sont possibles. Il faut également estimer la quantité de fichiers que l'application va nécessiter. Si le nombre de fonctionnalités proposées est importante, il faut, non seulement envisager un autre type de menu, voire une arborescence, mais aussi une autre répartition des fichiers en dossiers. On pourrait imaginer, par exemple, un dossier pour les scripts concernant les élèves, un dossier pour les scripts concernant les maîtres de stage et un dossier pour les scripts concernant les hôtes des stages.

La réalisation d'une feuille de style correcte est aussi déterminante quant à la qualité relative de l'interface que vous pourrez proposer. Il n'est pas vain de consacrer un certain temps à son élaboration. Il est également possible de vous inspirer de feuilles de styles utilisées dans des sites que vous trouvez très agréables et très lisibles. Il n'y a pas, que nous sachions, de copyright sur les feuilles de styles et celles-ci se retrouvent évidemment dans le cache Internet de la machine cliente. Dès lors,…

Il vous reste à appliquer la démarche que nous venons d'initier au développement d'une application qui vous concerne et dont vous connaissez les caractéristiques du domaine.

Bon travail.