REPUBLIQUE DU CAMEROUN
PAIX – TRAVAIL - PATRIE
-=-=-=-=-=-=-=-=-=-=- -=-=-=-= -=-=-=-=-=-=-
UNIVERSITE DE DSCHANG
- -
INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
FOTSO VICTOR DE BANDJOUN
B.P. 134 Bandjoun (Cameroun)

REPUBLIQUE OF CAMEROUN
PEACE – WORK – FATHERLAND
THE UNIVERSITY OF DSCHANG

SUPPORT DE COURS, TD ET TP

DEVELOPPEMENT DES APPLICATIONS & TECHNOLOGIE WEB

© M. NOULAMO THIERRY
Enseignant à l'IUT FOTSO Victor de Bandjoun
tnoulamo@yahoo.com

Année académique

SOMMAIRE

Chapitre 1 Programmation PHP : Rappel

1. Introduction

Les services Web sont une nouvelle technologie permettant aux pages Web d'accéder à des applications distribuées. En offrant à la fois l'accès à des informations et à des fonctionnalités applicatives sous la forme d'un service, les services Web sont fournis et vendus en tant que flux de services auxquels il est possible d'accéder à partir de n'importe quelle plate-forme. La page Web qui se connecte au service Web s'appelle généralement un consommateur tandis que le service même s'appelle un fournisseur. La création de consommateurs de services Web se fait à l'aide des types de documents ColdFusion, ASP.NET et JSP (Java Server Pages), PHP, etc. Nous Etudierons dans le cadre de ce cours PHP.

Un peu d'histoire

1994 : création de PHP/FI (Personal Home Page Tools/Form Interpreter) par Rasmus Lerdof (Canada) pour évaluer la consultation de son CV en ligne.

1995 : mise à disposition du code pour tout le monde.

1997 : redéveloppement du coeur par 2 étudiants (Andi Gutmans et Zeev Zuraski, Israëm) pour donner PHP (PHP: Hypertext Processor) 3.0.

2002 : 8 millions de sites basés sur PHP.

2004 : 15 millions de sites basés sur PHP.

Evolution du langage

Au départ basé sur le langage Perl, PHP a été réécrit ensuite en langage C.

Le moteur a été réécrit plusieurs fois.

D'abord procédural, le langage a évolué a partir de la version 4.0 pour intégrer les technologies objet.

Principales caractéristiques

Le moteur PHP s'intègre au serveur web Apache : il prend alors en charge toutes les pages avec une extension .php.

PHP est un langage interprété : il n'y a pas de phase de compilation et les erreurs sont découvertes au moment de l'exécution.

La syntaxe est très inspirée du langage C, tout en apportant beaucoup de simplification, notamment dans la gestion des variables.

Remarque : dans le présent support, on n'aborde pas les caractéristiques objet de PHP ; le code décrit est compatible PHP 3.0 et versions ultérieures.

2. L'architecture client-serveur

Depuis le début des années 90, les réseaux LAN (Local Area Network) et WAN (Wide Area Network) sont devenus l'épine dorsale du système informatique des entreprises en s'appuyant sur la technologie Internet.
Depuis le milieu des années 90, cette technologie est mise à la disposition du grand public WEB).
L'accès client-serveur aux données est donc devenu une activité essentielle pour un grand nombre de personnes partout dans le monde.
Les types d'accès sont :

- accès informationnel (requête d'interrogation et réponse),
- accès transactionnel (mises à jour rapides en temps réel),
- en accès décisionnel (exécution de requêtes complexes sur de gros volumes de données).

Les architectures client-serveur sont un modèle d'architecture où les programmes sont répartis entre processus clients et serveurs communiquant par des requêtes avec réponse.

2.1 Définitions de bases

Client

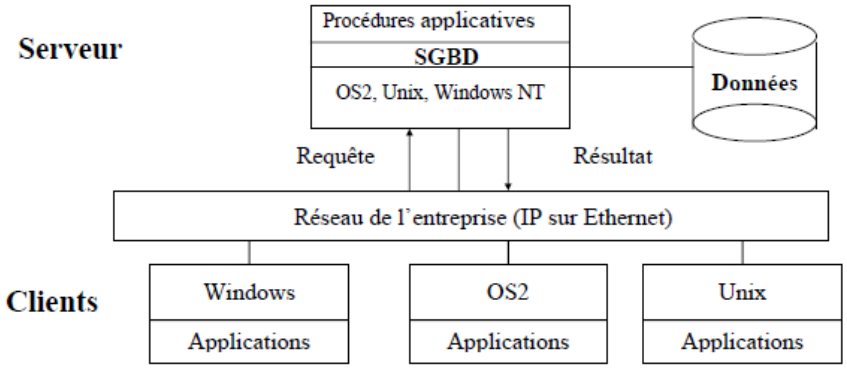
Processus demandant l'exécution d'une opération à un autre processus par envoi d'un message (requête) contenant le descriptif de l'opération à exécuter et attendant la réponse à cette opération par un message en retour.
Il supporte une partie du code de l'application. Le code implémente au minimum les dialogues avec les utilisateurs et l'affichage des résultats (GUI : Graphical User Interface).

Serveur

Processus accomplissant une opération sur demande d'un client et transmettant la réponse à ce client.
Il assure le stockage, la distribution, la gestion de la disponibilité et de la sécurité des données. Ces fonctionnalités sont offertes par le SGBD résidant sur le serveur ou accessibles sur un autre serveur connecté au réseau. Il supporte une partie du code de l'application :
un certain nombre de traitements sur les données peuvent être regroupés et exécutés sur le serveur.

2.2 Deux générations de clients/serveurs

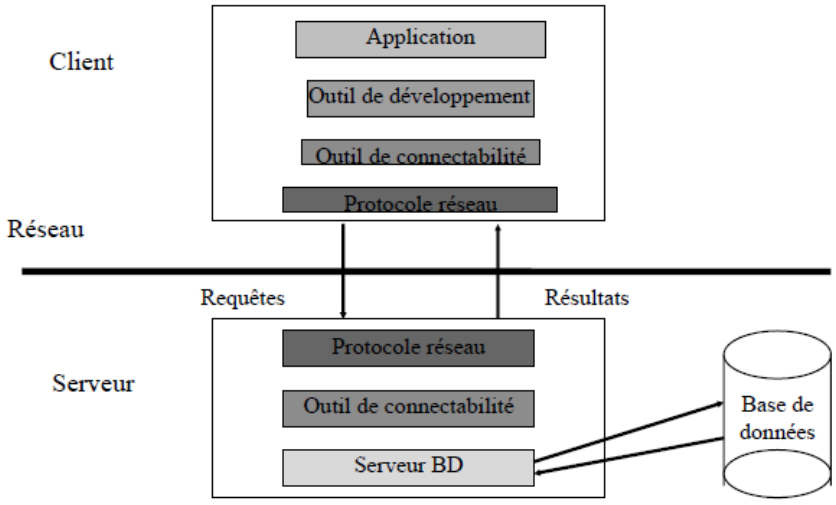
Première génération : Client-Serveur sur le réseau d'entreprise



Avantages du C/S de première génération :

- Cette architecture supporte des machines hétérogènes (Windows 3.11, Windows 95 ou 98, Windows NT, Mac OS, Unix, ...)
- Répartition des traitements entre clients et serveurs
- Relative indépendance vis à vis du choix des protocoles réseau et des systèmes d'exploitation sur les serveurs et les clients. Les vendeurs de logiciel doivent aujourd'hui avoir cette flexibilité.

Les composants de l'architecture C/S



Outil de connectabilité, généralement appelé le middleware, est le logiciel qui implémente le protocole qui permet au processus client de dialoguer avec le processus serveur (ODBC, SQL Net, ..).

Outils de développement :

- langages de 3ème génération : programmes C, Cobol appelant des bibliothèques du SGBD.

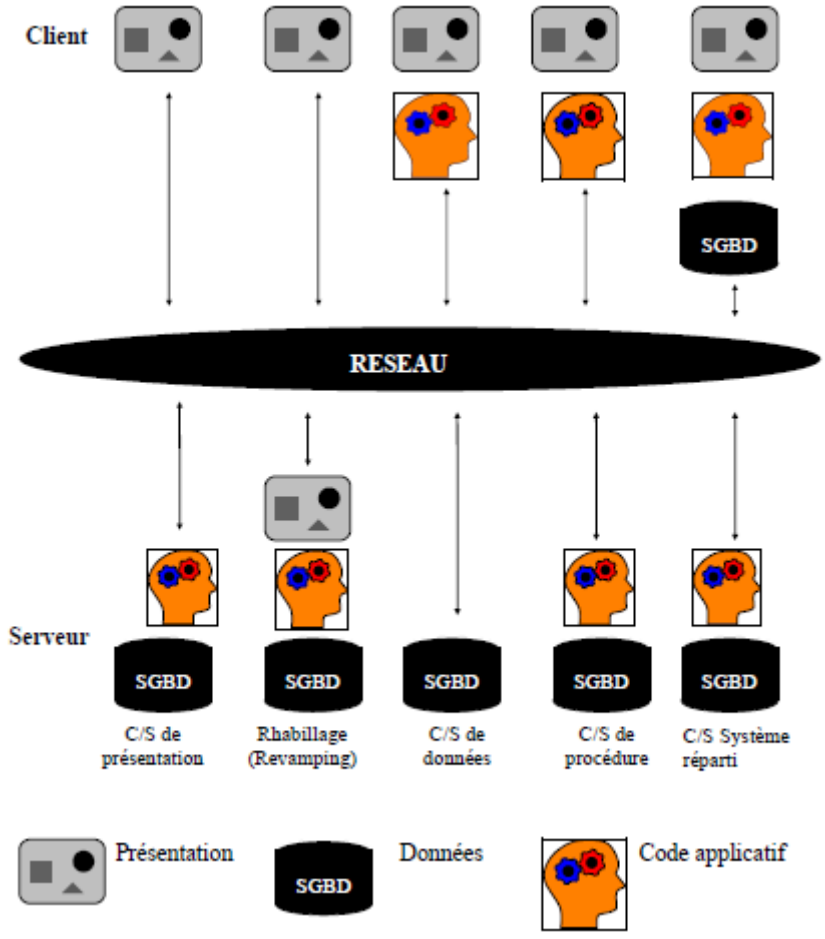
- langages de 4ème génération : ils offrent des composants logiciels qui permettent de développer très rapidement une application (Access, PowerBuilder, Uniface, ...) et intègrent un langage de programmation en général interprété.

Inconvénients du C/S de première génération :

La technologie C/S est diffusée depuis plus de 10 ans dans les entreprises.

- Il est parfois nécessaire de fabriquer une version d'application par type de plateforme
- En cas de mise à jour, il faut déployer la mise à jour sur tous les clients
- Sur chaque client, lorsque l'on installe un logiciel, il faut installer le ou les middlewares qui lui permet(tent) de communiquer avec des serveurs.
- Peu de développeurs maitrisent réellement l'ensemble des technologies mise en oeuvre dans une application C/S.

Les différents types de C/S de première génération

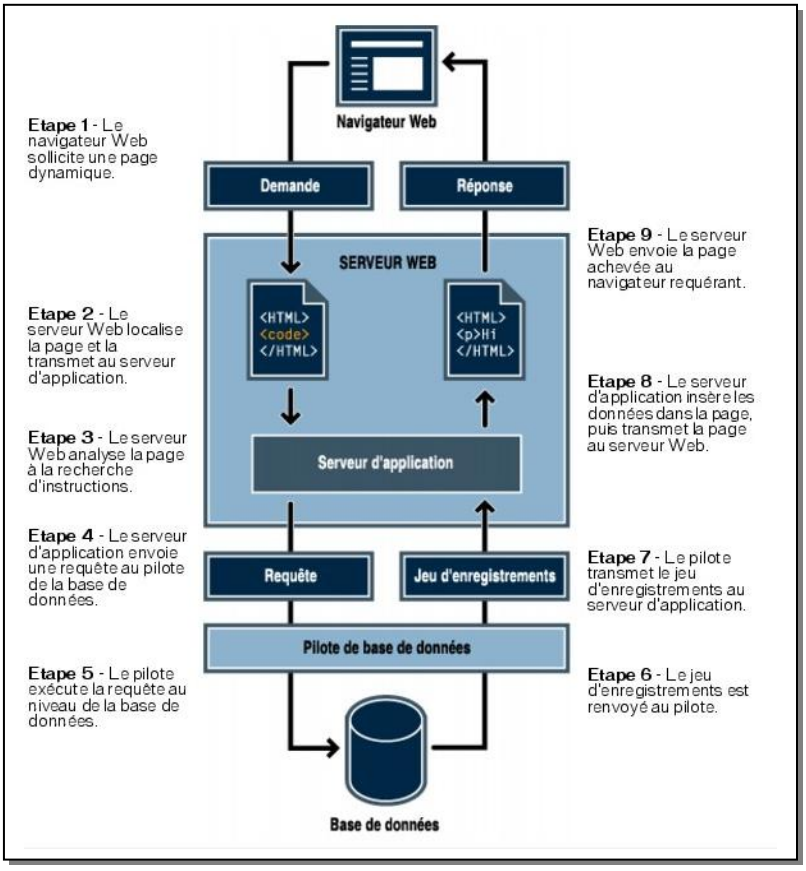


2.3 Deuxième génération : Client-serveur sur le WEB

Internet et Java sont les solutions largement utilisées pour réduire ces coûts.
Le client Legé est un PC équipé d'un navigateur Internet. Les applications sont des pages HTML téléchargées sur le PC.
On parle dans ce cas d'architecture three-tiers car le serveur WEB et le serveur d'application jouent le rôle d'intermédiaires entre le client et le serveur de la base de données.
On peut distinguer deux types de client-serveur sur Internet :

- le C/S HTML/WEB (CGI : Common Gateway Interface, servlets Java),
- le C/S à composants distribués (applets Java, Active X).

3. Principe des applications web



Les applications web fonctionnent grâce à des échanges réseau entre un logiciel client (le navigateur web) et un logiciel serveur (le serveur web).

Ces échanges sont basés sur le protocole HTTP : le navigateur envoie une requête au serveur web et reçoit du contenu à afficher.

La navigation sur un site est constituée d'un ensemble de requêtes/réponses qui s'enchaînent.

Il y a 2 types de requêtes : les requête de type GET (barre d'URL et clic sur un lien) et les requêtes de type POST (validation de formulaire).

Apport de PHP

Les pages HTML sont des pages de contenu statique stockées dans le système de fichier du serveur.

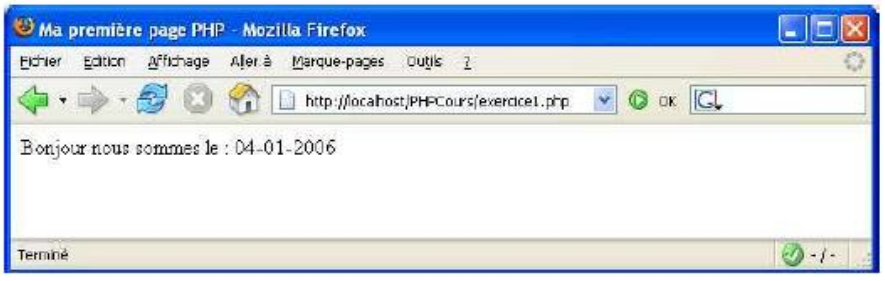
Les pages PHP contiennent du code HTML, mais aussi du code PHP qui est exécuté sur le serveur au moment de la requête et produit du code HTML ; la page est dite dynamique, car le contenu final HTML renvoyé au navigateur peut changer selon le contexte.

PHP nécessite l'ajout et la configuration d'un module spécifique dans le serveur web Apache.

Exemple simple

```
<?php echo "<?xml version=\"1.0\" encoding=\"ISO-8859- 15\"?>" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
        <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
                <head>
                        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
                        15" />
                        <meta name="keywords" content="agronomie,INA-PG">
                        <meta name="revisit-after" content="30 days" />
                        <title>Ma première page PHP</title>
                </head>
                <body>
                        <div>
                                Bonjour nous sommes le :
                                <?php
                                        $date = date("d-m-Y");
                                        echo "$date"; // Affichage de la date
                                ?>
                        </div>
                </body>
        </html>
```

Affichage



Code HTML généré
Si on demande l'affichage du code source dans le navigateur :

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
        <head>
            <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15" />
            <meta name="keywords" content="agronomie,INA-PG">
            <meta name="revisit-after" content="30 days" />
            <title>Ma première page PHP</title>
        </head>
        <body>
            <div>
                Bonjour nous sommes le :
                O4-01-2006
            </div>
        </body>
    </html>
```

Le navigateur reçoit et interprète toujours du code HTML car le code PHP est évalué au niveau du serveur.

Contrairement à une page HTML, on ne peut donc pas tester une page PHP directement en l'ouvrant depuis le système de fichier sans passer par une connexion HTTP sur un serveur de type Apache.

4. Notions de base

Code PHP au sein d'une page HTML

- Le code PHP est inséré directement au sein d'une page contenant du HTML.
- Le code PHP doit cependant être contenu dans des balises spéciales : <?php ... ?> ou <? ... ?>.
- Du code HTML est produit grâce à l'instruction echo.
- On peut mettre plusieurs instructions au sein d'une même balise <? ?>.
- Une instruction est toujours terminée par un ;.
- <? echo $variable; ?> peut être condensé en <?= $variable ?>.
- Pour être reconnu par l'interpréteur PHP, le fichier doit porter l'extension .php. Syntaxe de base
- L'instruction echo permet de générer de l'affichage HTML au sein d'une page.

- Les chaînes de caractères sont écrites entre "" : on doit faire précéder un " d'un \ pour l'inclure au sein d'une chaîne.
- Les noms de variables commencent toujours par le caractère $.
- Les variables n'ont pas besoin d'être déclarées, ni d'être typées ; elles sont créées implicitement lors de la première utilisation.

- Le langage fournit un certain nombre de fonction prédéfinies ; lors d'un appel de fonction, les paramètres sont passés entre ( ) et séparés par des ,.
- Les commentaires sont écrits en commençant par // sur une ligne ou
- entre /* et */ sur plusieurs lignes.
- Les accolades {} permettent de délimiter des blocs d'instructions.

Remarque : lorsqu'on respecte la norme XHTML, les caractères <? de la première ligne sont interprétés comme une balise PHP. Il faut donc obligatoirement générer cette première ligne avec une instruction PHP echo.

5. Structures de contrôle

PHP propose toutes les structures de contrôle classiques.

Tests conditionnels

```
<?
	if (is_numeric($texte)) {
		echo "La variable contient un nombre";
	}
	else {
		echo "La variable contient du texte";
	}
?>
```

On peut enchaîner des tests avec l'instruction elseif. On peut utiliser l'instruction switch lorsqu'on a beaucoup de conditions.

Schémas itératifs

```
<?
	for ( $i = 1; $i <= 10 ; $i++)
	{
		echo $i . ' ';
	}
	i = 1;

	while ($i<=10) {
		echo $i . ' ';
		$i++;
	}
```

6. Interactions avec l'utilisateur

Principe général

Dans une application web, l'interaction avec l'utilisateur est essentiellement fondée sur la validation de formulaire HTML, qui provoque l'envoie de données vers le serveur et l'affichage de nouvelles informations en provenance de celui-ci.

Des technologies telles que Javascript permettent la gestion d'événements, mais celle-ci reste locale au navigateur et donc limitée.

Formulaire : côté client

Soit un fichier formulaire.php :

```
...
<body>
	<div>
	<form action="validation_formulaire.php" method="post">
		<p><label for="nom">Nom :</label>
		<input type="text" size="30" name="nom" /></p>
		<p><label for="prenom">Prénom :</label>
		<input type="text" size="20" name="prenom" /></p>
		<p><input type="submit" name="btAction" value="OK" />
		<input type="reset" name="btAnnuler" value="Annuler" /></p>
	</form>
	</div>
</body>
...
```

Il faut préciser dans la balise form le nom du fichier PHP qui traitera les données, ainsi que la méthode de transfert de ces données (POST conseillé).

Il est important de donner un nom à chaque élément du formulaire : ce nom permet ensuite de retrouver les valeurs associées saisies par l'utilisateur.

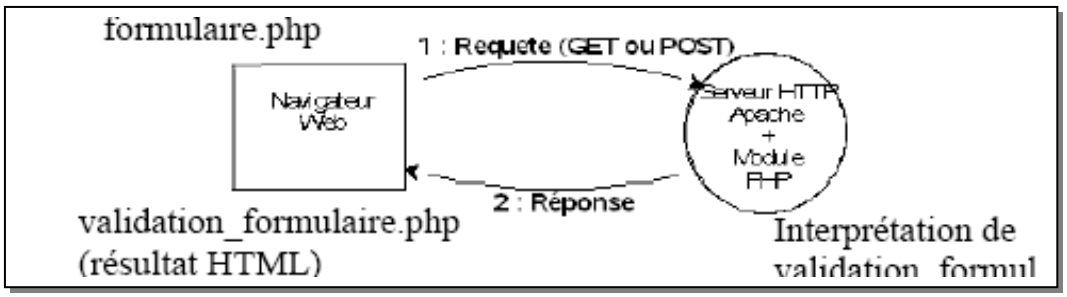
Formulaire : côté serveur

Le formulaire précédent est traité par validation_formulaire.php :

```
...
<body>
	<div>
	<?
		if (empty($_POST["nom"])
			or empty($_POST["prenom"])) {
			echo "Vous devez saisir un nom et un prénom !";
			}
		else {
			$nom = $_POST["nom"];
			$prenom = $_POST["prenom"];
			echo "Bonjour $prenom $nom";
		}
	?>
	</div>
```

```
</body>
```

Enchaînement

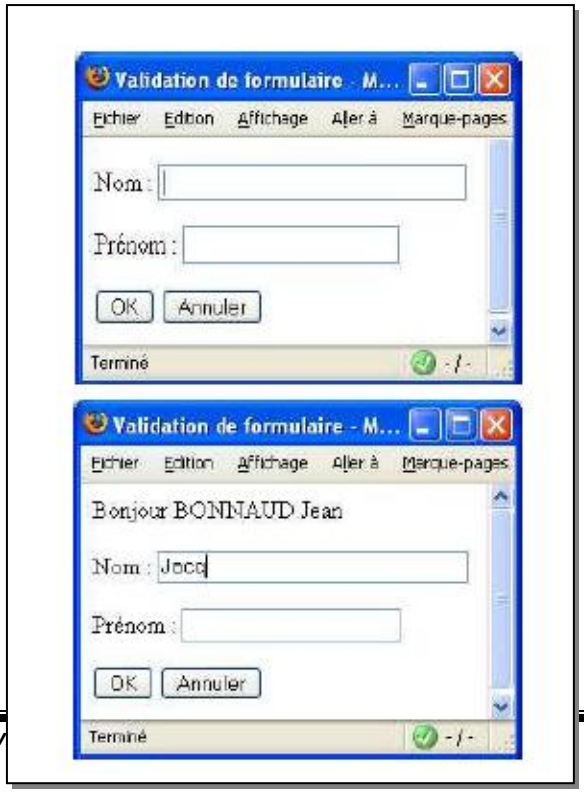


1. Le navigateur rassemble les informations saisies dans le formulaire (nom et prénom) et les envoie par une méthode POST au serveur.
2. Le serveur interprèle validation_formulaire.php avec les données reçues et génère un flux HTML résultat renvoyé au navigateur.

7. Formulaire validé par lui-même

Principe
Dans l'exemple précédent, lorsque l'utilisateur s'est trompé ou s'il veut faire une nouvelle saisie, il soit revenir en arrière en utilisant la touche back du navigateur.
Il est souvent plus judicieux d'afficher le formulaire et le résultat sur une même page :
lors du premier accès, la méthode GET est employée ;
lors de la validation, la méthode POST est employée.

Exemple



```
...
        <body>
            <div>
                <?
                    if ($_SERVER["REQUEST_METHOD"] == "POST") {
                    // on est appelé par la méthode POST, donc
                    // en validation du formulaire
                        if (empty($_POST["nom"])
                            or empty($_POST["prenom"])) {
                            echo "Vous devez saisir un nom et un prénom !";
                           }
                        else {
                            $nom = $_POST["nom"];
                            $prenom = $_POST["prenom"];
                            echo "Bonjour $prenom $nom";
                        }
                    }
                ?>
        <form action="toutenun.php" method="post">
            <p><label for="nom">Nom :</label>
            <input type="text" size="30" name="nom" />
            </p>
            <p><label for="prenom">Prénom :</label>
            <input type="text" size="20" name="prenom" />
            </p>
            <p><input type="submit" name="btAction" value="OK" />
            <input type="reset" name="btAnnuler" value="Annuler" />
            </p>
        </form>
        </div>
        </body>
...
```

8. Tableaux

Exemples précédents
Dans les précédents exemples de code, on a :

```
$_SERVER["REQUEST_METHOD"]
$_POST["nom"]
$_POST["prenom"]
```

$_SERVER et $_POST sont en fait des tableaux associatifs pré-remplis.

Tableaux associatifs
Un tableau associatif permet de faire correspondre des valeurs à des clés :

```
<?php
        // Création d'un tableau de 3 éléments
        $tableau = array(1=>"rouge", 2=>"bleu", 3=>"jaune");
        // Ajout d'un élément à la fin du tableau
        $tableau[] = "vert";
        // Modification du 2ème élément
        $tableau[2] = "vert";
        // Affichage du 3ème élement
        echo $tableau[3];
        // Ajout d'un élément associé à "rien"
        $tableau["rien"] = "transparent";
        // affichage de tout tableau :
        // Array ( [1] => rouge [2] => vert ... )
        print_r($tableau);
        // Parcours du tableau
        foreach ($tableau as $index => $couleur) {
        echo "<br /> $index => $couleur";
        }
        // Effacement du 2ème élément
        unset($tableau[2]);
        // Effacement de tout le tableau
        unset($tableau);
?>
```

Quelques fonctions prédéfinies

Chaînes de caractères	**strlen($chaine)** : longueur d'une chaîne **strpos($chaine, $car)** : position d'un caractère **strtolower($chaine)** : conversion minuscules **strtoupper($chaine)** : conversion majuscules
Dates	**getdate()** : date et heure **gettimeofday()** : heure actuelle **date($format, $date)** : formatage
Tableaux	**count($tableau)** : nombre d'éléments **sort($tableau)** : tri

Fonctions définies par l'utilisateur
PHP permet de créer des fonctions :

```
<?php
        function factorielle($nombre) {
                $resultat = 1;
                for ( $i = 2; $i <= $nombre ; $i++) {
                        $resultat = $resultat * $i;
                }
        return $resultat;
        }
?>
```

On peut définir des fonctions dans des fichiers .php et les utiliser dans d'autres fichiers grâce à l'instruction include :

```
<?php
        include "fonctions.php";
        echo factorielle(5);
?>
```

9 Passage et transmission de variables

9.1-Passage et transmission de variables par formulaire

Quand dans un site web un formulaire est rempli et envoyé, le contenu des champs saisis est transféré à la page destination sous forme de variables. Ce passage de variables ou de paramètres peut se faire de deux manières : en GET ou en POST.

Syntaxe

```
 <html>
         <body>
                <form method="post" action="destination.php">
                        <input type="text" name="nom" size="12">
                        <br> <input type="submit" value="OK">
                </form>

                <form method="get" action=" destination.php"> … </form>
        </body>
</html>
```

En GET les paramètres apparaissent associés à l'url sous formes de variables séparées par des & (http://localhost/destination.php?nom=Tarak&prenom=Joulak).

En POST le passage de paramètre se fait de manière invisible. Selon que la méthode d'envoi a été du GET ou du POST la récupération du contenu des variables est faite selon une syntaxe différente :

Syntaxe <?

//Dans le cas d'un envoi des paramètres en POST $variable1=$_POST'nom_du_champ'] ; [

//Dans le cas d'un envoi des paramètres en GET $variable1=$_GET['nom_du_champ'] ; ?>

Exemple 1 : Dans l'exemple suivant le fichier formulaire.html contient le script html permettant d'afficher un formulaire et d'envoyer les résultats de la saisie à la page resultat.php qui elle les affichera.

Fichier formulaire.html

```
<html>
<body>
<form method="post" action="resultat.php">
        Nom : <input type="text" name="nom" size="12"><br>
        Prénom : <input type="text" name="prenom" size="12">
        <input type="submit" value="OK">
</form>
</body>
</html>
```

Fichier resultat.php

```
<?
//Récupération des paramètres passés
$prenom = $_POST['prenom'];
$nom = $_POST['nom'] ;
 //affichage des paramètres
echo "<center>Bonjour $prenom $nom</center>";
?>
```

En exécutant à travers le serveur web le fichier formulaire.html, en remplissant le formulaire et en cliquant sur OK, nous sommes emmenés vers la page resultat.php qui nous affiche une phrase composée des champs saisis dans le formulaire. Les champs saisis sont donc passé de formulaire.html vers resultat.php.

Exemple 2 :
L'exemple précédent peut tenir dans un seul fichier qui s'enverrait à lui même les paramètres
Fichier formulaire.php

```
<?
        if ($_POST['submit'] == "OK") {
        <input type="submit" name="submit" value="OK">
        //Le formulaire a été transmis
        //Récupération des paramètres passés
        $prenom = $_POST['prenom'];
        $nom = $_POST['nom'];
        //affichage des paramètres
        echo "<center>Bonjour $prenom $nom</center>";
        } else
        {
        ?> <!-- Affichage du formulaire de saisie -->
        <form method="post" action="formulaire.php">
        Nom : <input type="text" name="nom" size="12"><br>
        Prénom : <input type="text" name="prenom" size="12">
```

```
        </form>
        <?
        }
?>
```

Remarque : La définition de la destination du formulaire (action="formulaire.php") aurait pu ne pas être nominative mais exploiter une variable serveur PHP_SELF puisque les paramètres sont envoyées à la page elle même. Ainsi $_SERVER['PHP_SELF'] retournera naturellement formulaire.php. Cela donnera donc :
<form method="post" action="<? echo $_SERVER['PHP_SELF']; ?>" > à la place de <form method="post" action="formulaire.php">

9.2-Passage et transmission de variables par hyperlien

Des paramètres ou variables peuvent passer d'une page source vers une page destination sans transiter par un formulaire pour leur envoi. Les hyperliens peuvent être des vecteurs de passage de paramètre.
Syntaxe
 Lien
La récupération des paramètres dans la page destination se fait par le tableau $_GET
<? $variable1=$_GET['variable1'] ; $variable2=$_GET['variable2'] ; … ?>
Exemple : Dans l'exemple suivant nous allons créer un fichier menu.php contenant un menu fait d'hyperliens. Chacun de ces hyperliens enverra des paramètres différents. Ce menu sera appelé dans une page qui réagira différemment selon le paramètre envoyé.

Fichier menu.php

```
<table width="200" border="0" cellspacing="0" cellpadding="1">
<tr>
        <td> <a href="page.php?menu=1">Menu1</a> </td>
</tr>
<tr>
        <td> <a href="page.php?menu=2">Menu2</a> </td>
</tr>
<tr>
         <td> <a href="page.php?menu=3">Menu3</a> </td>
</tr>
</table>
```

Fichier page.php

```
<? Include("menu.php") ;
echo "<br><br>" ;
$menu= $_GET['menu'] ;
switch ($menu) {
        case 1: echo "Ceci est la page obtenue du Menu1" ; break;
        case 2: echo "Ceci est la page obtenue du Menu2" ; break;
        case 3: echo "Ceci est la page obtenue du Menu3" ; break; default: echo "Ceci est la page
d'accueil" ; } ?>
```

En exécutant à travers le serveur web page.php, la sélection de chacun des menus chargera à nouveau la page en envoyant des paramètres différents qui seront traités par la page.

10. Les fichiers

10.1- Ouverture / fermeture de fichier

Avant toute opération de lecture ou écriture sur un fichier il y a nécessité de l'ouvrir. Et à la fin de tout traitement d'un fichier il y a nécessité de le fermer.

Syntaxe

```
<?
// Ouverture de fichier
"
$monfichier = fopen(nom_du_fichier", "r+");
// Traitements sur le fichier
// …..
// Fermeture du fichier fclose($monfichier);
?>
```

Fopen() prend en entrée :
-le nom du fichier (ou meme une url)
-le mode d'ouverture du fichier Il retourne un handle.
Fclose() prend en entrée le handle envoyé par le fopen.
Les modes d'ouverture d'un fichier sont les suivants :

- 'r' Ouvre en lecture seule, et place le pointeur de fichier au début du fichier.
- 'r+' Ouvre en lecture et écriture, et place le pointeur de fichier au début du fichier.
- 'w' Ouvre en écriture seule ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
- 'w+' Ouvre en lecture et écriture ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
- 'a' Ouvre en écriture seule ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.
- 'a+' Ouvre en lecture et écriture ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.

Exemple

```
<?
$handle = fopen("http://www.example.com/", "r");
$handle = fopen("test/monfichier.xt", "r+") t ;
?>
```

10.2- Lecture de fichier

a- fgets()

string fgets (resource handle ,int length)
fgets retourne la chaîne lue jusqu'à la longueur length - 1 octet depuis le pointeur de fichier handle , ou bien la fin du fichier, ou une nouvelle ligne (qui inclue la valeur retournée), ou encore

un EOF (celui qui arrive en premier). Si aucune longueur n'est fournie, la longueur par défaut est de 1 ko ou 1024 octets.

Syntaxe

```
<?
$monfichier = fopen("nom_du_fichier", "r+");
fgets ($monfichier, $taille) ; fclose($monfichier);
?>
```

La fonction prend en paramètre :

- le handle retourné par la fonction fopen()
- la taille en octets de lecture (optionnel)

Elle retourne en sortie une chaîne de caractères.

Exemple :

L'exemple suivant va permettre de lire dans un fichier texte (fichier.txt) ligne par ligne puis d'afficher le résultat à l'écran en mettant un numéro à chaque ligne.

Fichier fichier.txt

```
Tarak Joulak
Guest0
abc123
```

Fichier page.php

```
<?
$i=0;
$fichier = 'fichier.txt';
$fp = fopen($fichier,'r') //ouverture du fichier en lecture seulement
while (feo$fp)) // tant qu'on n'est pas a la fin du fichier
{
$ligne = fgets($fp); //Lecture ligne par ligne
echo 'Ligne '$i++.'--->'$ligne.'<br>';
}
fclose($fp);
?>
```

Le résultat obtenu sera :

```
Ligne 1 ---> Tarak Joulak
Ligne 2 ---> Guest0
Ligne 3 ---> abc123
```

b -fread()

string fread (resource handle , int length)

Une autre manière de faire de la lecture dans un fichier est d'utiliser fread()

fread lit jusqu'à length octets dans le fichier référencé par handle . La lecture s'arrête lorsque length octets ont été lus, ou que l'on a atteint la fin du fichier, ou qu'une erreur survient (le premier des trois).

Syntaxe

```
<?
$monfichier = fopen("nom_du_fichier", "r+"); fread ($monfichier, $taille) ;
```

```
fclose($monfichier);
?>
```

c- file()

array file (string filename)

Encore une autre manière de lire dans un fichier est d'utiliser la fonction file() Identique à readfile, hormis le fait que file retourne le fichier dans un tableau. Chaque élément du tableau correspondant à une ligne du fichier, et les retour−chariots sont placés en fin de ligne.

Syntaxe

```
<?
file ($nom_de_fichier)
?>
<?
// Lit un fichier, et le place dans une chaîne
$filename = "fichier.txt";
$handle = fopen ($filename, "r");
$contents = fread ($handle,filesize ($filename));
echo "-->".$contents; fclose ($handle);
?>
```

Le résultat obtenu sera :

```
--> Tarak Joulak
Guest0
abc123
```

Rq : la fonction file() ne nécessite pas l'usage de open() et close().

Exemple Fichier page.php

```
<?php
$filename = "test/fichier.txt";
$fcontents = file($filename);
echo $fcontents[0]."<br>";
echo $fcontents[1]."<br>";
?>
```

Le résultat obtenu sera :

```
Tarak Joulak
Guest0 3
```

- Ecriture dans fichier La fonction pour l'écriture dans un fichier est fputs()

int fputs (int handle , string string , int length)

fputs écrit le contenu de la chaîne string dans le fichier pointé par handle . Si la longueur length est fournie, l'écriture s'arrêtera après length octets, ou à la fin de la chaîne (le premier des deux). fwrite retourne le nombre d'octets écrits ou FALSE en cas d'erreur.

Syntaxe

```
<? $monfichier = fopen("nom_du_fichier", "r+");
fputs ($monfichier, "Texte à écrire");
fclose ($monfichier);
?>
```

Exemple

```
<? $filename = "fichier.txt"; $monfichier = fopen ($filename, "a+");
$contenu="ceci est le texte a ecrire \r\n";
fputs($monfichier, $contenu);
fclose ($monfichier);
?>
```

Le résultat obtenu sera :
Le fichier fichier.txt aura une ligne supplémentaire contenant « ceci est le texte à écrire »

10.3- Fonctions diverses de traitement de fichier :

-feof ($handle) fonction qui retourne un booleen pour indiquer la fin du fichier parcouru. Elle prend en entrée le handle retourné par la fonction fopen().
-filesize ($nom_du_fichier) fonction qui retourne la taille du fichier en octets.
-file_exists ($nom_du_fichier) fonction qui retourne un booleen indiquant si le fichier existe ou non.

11. Variables persistantes: Cookies et Session

Les variables ont une durée de vie limitée : celle du script qui les appelle. Ainsi que nous l'avons vu dans le chapitre précédent l'unique moyen de transmettre ces variables de pages en pages consiste à effectuer un passage de paramètres (méthode GET ou POST) ce qui d'une manière générale est contraignant à plus d'un titre :
- Contraignant pour le développeur puisque il doit gérer par code ces passages de paramètres,
- Contraignant pour la sécurité si l'on ne désire pas que le client accède à certaines informations.
PHP offre un mécanisme de stockage d'informations de manière persistante. Autrement dit tout au long de la navigation à l'intérieur d'un site des variables seront accessibles sans avoir pour autant à les passer en paramètres. Deux types de variables persistantes existent :
- Les variables persistantes côté client : les cookies
- Les variables persistantes côté serveur : la session

11.1- les Cookies

Les cookies sont un mécanisme d'enregistrement d'informations sur le client, et de lecture de ces informations. Ce système permet d'authentifier et de suivre les visiteurs d'un site. PHP supporte les cookies de manière transparente.

a- setcookie() :

Cette fonction permet de définir un cookie qui sera envoyé avec le reste des en−têtes. int setcookie (string nom_variable ,[string valeur_variable],[int expiration],[string chemin],[string domaine],[int securité])

- nom_variable : nom de la variable a stocker
- valeur_variable : Valeur de la variable a stocker
- expiration : durée pour l'expiration du cookie
- chemin : le chemin du répertoire ou doit être lu le cookie
- domaine : le nom domaine
- securité : le type d'entête (http, https)

Tous les arguments sauf nom_variable sont optionnels. A l'instar de la fonction header(), setcookie() doit impérativement être appelée avant tout affichage de texte.
Syntaxe

<? setcookie("variable1",$valeur1, $durée,$chemin,$domaine,$securite); ?>
Exemples
<? $valeur= "Ceci est la valeur de la variable" ; setcookie ("TestCookie",$value,time()+3600); /* expire dans une heure */ ?> 31

b- lire un cookie
Syntaxe <? $HTTP_COOKIE_VARS["$nom_de_la_variable"] ; ?>
Exemple <? echo $HTTP_COOKIE_VARS["TestCookie"]; ?>
Le résultat obtenu sera : Ceci est la valeur de la variable

c- Tableau de variables dans un cookie
Il est possible d'envoyer un tableau de variables à stocker dans un cookie
Exemple
<? setcookie("tcookie[trois]", "troisième cookie");
setcookie("tcookie[deux]", "second cookie");
setcookie("tcookie[un]", "premier cookie"); }?>

Exemple de lecture d'un tableau stocké dans un cookie
<? //récupération du tableau cookie dans la variable $cookie
$cookie=$HTTP_COOKIE_VARS["tcookie"];
//Vérification si la variable est vide ou non
if (isset($cookie)) {
while(list($name, $value) = each($cookie)) {
echo "$name == $value
\n";
}
}
?>
Le résultat obtenu sera :
trois == troisième cookie
deux == deuxième cookie
un == premier cookie

d-Suppression d'un cookie
Pour supprimer un cookie envoyez un cookie avec une variable sans valeur et un délai dépassé.
Exemple
<? Setcookie ("TestCookie","",time()-100); ?>

e- Limitation des cookies
Le problème majeur du cookie c'est que le client a le pouvoir de le refuser (en configurant spécifiquement son browser). Votre application risque donc de ne pas pouvoir fonctionner. Il y a aussi des risques plus graves quant à la sécurité. L'usurpation d'identité, car ce fichier peut être recopié facilement sur un autre ordinateur et modifié puisque ce n'est qu'un fichier texte.

11.2- les sessions

La gestion des sessions avec PHP est un moyen de sauver des informations entre deux accès. Cela permet notamment de construire des applications personnalisées, et d'accroître l'attrait de votre site.
Chaque visiteur qui accède à votre site se voit assigner un numéro d'identifiant, appelé plus loin "identifiant de session". Celui−ci est enregistré soit dans un cookie, chez le client, soit dans l'URL.
Les sessions vous permettront d'enregistrer des variables pour les préserver et les réutiliser tout au long de la visites de votre site. Lorsqu'un visiteur accède à votre site, PHP vérifiera si une session a déjà été ouverte. Si une telle session existe déjà, l'environnement précédent sera recréé.
L'inconvénient précédemment évoqué concernant les cookies est dépassé dans la mesure ou tout est stocké sur le serveur même.

a- session_start()

session_start() permet de démarrer une session pour le client .
Cette commande doit figurer dans toutes les pages elle perpétue le transfert des variables de session au cours de la navigation dans le site.
Le compilateur PHP va alors créer dans le répertoire de sauvegarde des sessions, un fichier dont le nom commence par sess_ et se termine par un identifiant généré de manière aléatoire.
L'identifiant de session peut être affiché par la commande session_id(). Vous pouvez également gérer vous-même ce nom de session en utilisant session_name() avant le démarrage de la session.
La durée de vie d'une session est paramétrée par session.cache_expire
La session est perdue définitivement pour l'utilisateur lorsque :
1- Il n'a exécuté aucune action (POST ou GET) au delà de la durée de vie définie .
2- Il ferme son navigateur.
3- La commande session_destroy est appelée.

b- Ajouter des variables dans la session

L'ajout de variable dans l'environnement de session se fait au travers d'une affectation classique. Toutefois la variable session définie doit être appelée vie la tableau $_SESSION[]
Dans cet exemple une variable du non de ville contenant la valeur Tunis a été enregistrée dans la session courante.

```
<?
session_start() ;
?>

<?
session_start() ;
$_SESSION['nom_variable'] = $valeur;
?>

<?
session_start() ;
$_SESSION['ville'] = "Tunis";
?>
```

c- Lire une variable dans la session

```
<?
session_start() ;
$variable = $_SESSION['nom_variable'] ;
?>

<?
session_start() ;
echo ' Le contenu de la variable VILLE de la session est : '. $_SESSION['ville'] ;
?>
```

Le résultat obtenu sera :
Le contenu de la variable VILLE de la session est : Tunis

d- Supprimer une variable de la session

```
<?
session_start() ;
unset ($_SESSION['nom_de_le_variable']);
?>
<?
session_start() ;
unset ($_SESSON['ville']); I /
/Verificatin de la suppression
if ( isset ($_SESSON['ville'])) I
{
$resultat = "La suppression a échoué .";
}
else
{
$resultat = "La ville a été effacée.";
}
echo $resultat;
?>
```

Le résultat obtenu sera :
La ville a été effacée.

e- Supprimer l'environnement de session

Dans l'exemple précédent nous avons vu comment supprimer une variable de l'environnement de session.
Il reste toutefois possible de supprimer tout un environnement de session donné. Pour cela il suffit de vider le tableau global des sessions en le réinitialisant.

```
<?
session_start() ;
$_SESSION = array();
?>
```

12. PHP et les bases de données

Vous pouvez utiliser pratiquement toutes les bases de données avec votre application Web, si vous possédez les pilotes de base de données requis.
Si vous prévoyez de créer de petites applications peu onéreuses, vous pouvez utiliser une base de données fichier, créée par exemple sous Microsoft Access. Si vous prévoyez de créer des applications stratégiques robustes, vous pouvez utiliser une base de données serveur, créée par exemple avec Microsoft SQL Server, Oracle 9i ou MySQL.
Si votre base de données réside sur un système autre que votre serveur Web, assurez-vous qu'il existe une connexion rapide entre les deux systèmes pour un fonctionnement efficace et rapide de votre application Web. Nous utiliserons dans le cadre de ce cours le SGBD MySQL.

12.1- Utilisation d'une base de données MySql

PHP fonctionne nativement avec une base de données MYSQL.
MYSQL est un système de gestion de base de données (SGBD) qui permet d'entreposer des données de manière structurée (Base, Tables, Champs, Enregistrements). Le noyau de ce système permet d'accéder à l'information entreposée via un langage spécifique le SQL.
Ainsi , dans les chapitres précédents nous avions vu que l'information au mieux pouvait être stockée dans des fichier accessibles en lecture et écriture par des scripts PHP.
MYSQL vient ajouter une couche supplémentaire de stockage des données qui est plus commode, rapide et puissante d'utilisation.
Voici ce qu'il peut se passer lorsque le serveur reçoit une demande d'un client de consultation d'une page en PHP qui fait appel à des données stockées sous MYSQL:
1. Le serveur WEB envoie le nom de la page PHP demandée à l'interpréteur PHP.
2. PHP exécute le script existant dans la page. Sitôt que des instructions relatives à la connexion à une base de données trouvées, PHP se charge d'envoyer les requêtes d'exécution à MYSQL.
3. MySQL exécute la requête et renvoie à PHP le jeu de données résultat.
4. PHP termine son traitement et renvoie la page HTML générée au serveur web qui la transmet à l'internaute.

12.2- Concepts fondamentaux : Bases, tables, champs, enregistrements.

Une base de donnée correspond donc à un ensemble de données sauvegardées de manière structurée. La structuration de ses données est hiérarchisée.
Une base de données peut regrouper plusieurs tables.
Ainsi une base de données BIBLIOTHEQUE par exemple contiendra une table PERIODIQUE pour stocker les périodiques, une table LIVRE pour stocker les livres.
La table LIVRE regroupera plusieurs champs destinés à décrire et qualifier les livres.
Ainsi cette table intègrera un champs NISBN, un champs TITRE et un champs AUTEUR par exemple.
La consultation de ces données se fait au travers du langage SQL.

12.3- Administration de MYSQL

EasyPHP installe un produit pour l'administration d'une base de données MYSQL appelé PHPMYADMIN. Ce produit est intégralement développé en PHP. Toutefois nous lui préférons MySQL-Front qui reste plus simple d'utilisation.

MYSQL-Front est une application qui permet d'administrer une base de données MYSQL à distance (Création de base, de tables, d'enregistrements, exécution de requêtes, gestion des comptes, …)

12.4- SQL : petit récapitulatif du langage

Structured Query Langage est langage standard pour communiquer avec une base de données. Il permet la création de bases, la définition de tables, la consultation des enregistrement ainsi que leur mise à jour.

a-Création d'une base

CREATE DATABASE exercice
Cette requête SQL va permettre de créer une base vide du nom de exercice.

b-Création d'une table

```
CREATE TABLE `t_personne` (
`id` INT (6) UNSIGNED AUTO_INCREMENT, `Nom` VARCHAR (50),
`Prenom` VARCHAR (50), `Age` TINYINT (3) UNSIGNED,
UNIQUE(`id`), INDEX(`id`) )
```

Cette requête va permettre de créer une table du nom de 't_personne' contenant 4 champs :
id : un identifiant entier qui s'auto incrémentera à chaque nouvel ajout d'enregistrement
Nom : une chaîne de caractères d'au maximum 50 caractères
Prenom : une chaîne de caractères d'au maximum 50 caractères
Age : un entier

c-Ajout d'un enregistrement

```
INSERT INTO t_personne (id, Nom, Prenom, Age) VALUES (NULL, 'Joulak', 'Tarak', 32)
```

Cette requête SQL va permettre d'ajouter une ligne d'enregistrement à la table t_personne encore vide.
Le champs Nom prendra pour valeur Joulak
Le champs Prenom prendra pour valeur Tarak
Le champs Age prendra pour valeur 32
Il est à noter que le champs id étant en mode auto incrémental il ne faut pas lui assigner de valeur. Le compteur s'incrémentant tout seul à chaque nouvel enregistrement.

d-Consultation d'un enregistrement

```
SELECT * FROM t_personne WHERE id=1
```

Cette requête va extraire de la base de données la ligne d'enregistrement ayant pour identifiant 1 (id=1)

e-Mise à jour d'un enregistrement

```
UPDATE t_personne SET Age= 35 WHERE id=1
```

Cette requête va mettre à jour dans la table t_personne l'enregistrement dont le champs id est égal à 1 (id=1) en modifiant le champs Age à 35.

f-Suppression d'un enregistrement

```
DELETE FROM t_personne WHERE id=1
```

Cette requête SQL va supprimer de la table t_personne l'enregistrement ayant pour identifiant 1 (id=1).

12.5- Accéder à MYSQL via PHP

Dans ce paragraphe nous allons voir comment combiner le langage SQL aux fonctions spécifiques de PHP pour exploiter le contenu d'une base de données MYSQL.

a- Connection à un serveur MYSQL

La première opération a réaliser pour accéder à MYSQL via un script PHP correspond à la connection à un serveur de base de données MYSQL.
La fonction mysql_connect() retourne un booléen ; True si la connection est possible False si elle ne l'est pas.

b- Connection à une base de données MYSQL

Suite à la connection au serveur MYSQL, il faut choisir quelle est la base du serveur MYSQL sur laquelle nous désirons nous connecter. Un serveur MYSQL pouvant contenir plusieurs bases de données.

```
<?
mysql_connect("$nom_serveur_MYSQL", "$utilisateur", "$mot_de_passe");
?>

<?
$nom_serveur_MYSQL="localhost" ;
$utilisateur="root" ;
$mot_de_passe="" ;
mysql_connect("$nom_serveur_MYSQL", "$utilisateur", "$mot_de_passe");
?>

<?
mysql_select_db("$nom_de_la_base");
?>

<?
$adresse_serveur_MYSQL="localhost" ;
$utilisateur="root" ;
$mot_de_passe="" ;
$nom_de_la_base="exercice" ;
mysql_connect("$adresse_serveur_MYSQL", "$utilisateur", "$mot_de_passe");
mysql_select_db("$nom_de_la_base");
?>

while ($data= mysql_fetch_array($res))
{
echo $data['Nom'].' '.$data['Prenom'] ' ' .$data['Age'] '<br>'; . .
}
<?
```

```
mysql_connect("localhost" , "root" , "");
mysql_select_db("exercice");
$requete_sql="SELECT * FROM t_personne" ;
$res = mysql_query("$requete_sql");
?>
```

De même la fonction mysql_select_db() retourne un booléen ; True si la connection est réussie, False si elle a échoué.

c- Exécution d'une requête SQL via PHP

Le lancement d'une requête SQl au travers d'un script PHP se fait par le biais de la fonction mysql_query() qui prend en paramètre la requête SQL et retourne true ou false selon que la requête ait réussi ou échoué. Pour le cas particulier d'une requête avec SELECT et si la requête a réussi alors un identifiant est retourné afin d'être exploité par d'autres fonctions.

Dans l'exemple précédent une ligne d'enregistrement a été ajoutée à la table t_personne contenant Tarak Joulak 32.

En remplaçant la requête de l'exemple précédent par les requêtes SQL de modification et suppression vues dans le paragraphe précédent on obtient les résultats déjà observés en exécutant directement la requête SQL.

d- Parcourir le résultat d'un SELECT

Une requête SQL de consultation peut retourner plusieurs enregistrements. De ce fait il y a besoin de pouvoir les consulter enregistrement par enregistrement et champs par champs. mysql_fetch_row() est la fonction PHP destinée pour ce type de traitements.

Cette fonction prend en entrée l'identifiant retourné par mysql_query(). Elle retourne un tableau contenant la ligne demandée ou false s'il n'y a plus d'enregistrement. Elle est généralement exploitée dans une boucle WHILE.

```
<?
…
$res = mysql_query("$requete_sql");
?>
<?
$adresse_serveur_MYSQL="localhost" ;
$utilisateur="root" ;
$mot_de_passe="" ;
$nom_de_la_base="exercice" ;
$requete_sql="INSERT INTO t_personne (id, Nom, Prenom, Age) VALUES (NULL,'Joulak', 'Tarak', 32)" ;
mysql_connect("$adresse_serveur_MYSQL", "$utilisateur", "$mot_de_passe");
mysql_select_db("$nom_de_la_base");
$res = mysql_query("$requete_sql");
?>

<?
…
$data = mysql_fetch_row($resultat_de_mysql_query) ;
?>
```

L'exécution de ce script affichera l'ensemble du contenu de la table t_personne avec un enregistrement par ligne.

12.6 Extraction des données

mysql_fetch_row($result) : retourne une ligne de résultat sous la forme d'un tableau. Les éléments du tableau étant les valeurs des attributs de la ligne. Retourne FALSE s'il n'y a plus aucune ligne.

Exemple 1 :

```
$requet = ''SELECT * FROM users'';
if($result = mysql_query($requet)) {
        while($ligne = mysql_fetch_row($result)) {
                $id = $ligne[0];
                $name = $ligne[1];
                $address = $ligne[2];
                echo ''$id - $name, $address <br />'';
        }
} else {
        echo ''Erreur de requête de base de données.'';
}
```

Ici, on accède aux valeurs de la ligne par leur indice dans le tableau.

Chapitre 2 PHP et la programmation objet

1- Introduction

Avant la version 5, PHP était loin d'être un langage de programmation orientée objet (POO), en comparaison de Java ou de C++, dont, il est vrai, la destination n'est pas la même. Dans ASP.Net, destiné typiquement au Web, toute action de programmation entraîne la création et la manipulation d'objets préexistants. Du fait de cette lacune de PHP 4, les projets de grande envergure le délaissaient au profit d'ASP.Net ou de JSP.

Ce chapitre aborde non pas l'utilisation d'objets prédéfinis mais l'ensemble des outils qui permettent au programmeur de créer ses propres objets.

De même que vous écrivez une fonction pour effectuer une tâche répétitive, vous avez un intérêt à créer des objets pour gérer des projets complexes. Un objet correspond à la modélisation d'une entité réelle ou abstraite, par exemple, un client, l'article qu'il commande ou la commande elle même. La POO permet de modulariser le code des scripts en décomposant les données à traiter en différentes entités, chacune étant représentée par un type d'objet. Elle offre de surcroît la

possibilité de réutiliser des classes déjà créées grâce au mécanisme de l'héritage, qui permet de créer de nouvelles classes à partir des classes existantes en leur ajoutant de nouvelles fonctionnalités.

2- Terminologie des objets

Si vous avez déjà pratiqué d'autres langages de programmation réellement orientés objet, les notions de classe et d'objet vous sont familières. Vous pouvez donc passer directement à la section suivante, qui vous permettra de voir la manière d'implémenter les classes et les objets dans PHP.

La terminologie propre aux classes et aux objets est très variable selon les sources. Pour cette raison, il apparaît utile de préciser le vocabulaire qui sera employé dans ce chapitre.

Un objet informatique est la représentation d'un objet réel au sens large. Il peut aussi bien s'agir d'un produit commercial, d'une personne ou d'un bon de commande. Le travail d'analyse du programmeur consiste dans un premier temps à dégager les différents types d'objets qui interviennent dans son application et leurs interactions. Il lui faut ensuite décrire les caractéristiques communes à chaque type d'objet.

Chaque client d'un site de commerce en ligne, par exemple, a un nom, un prénom, une carte bancaire, une adresse, etc., mais chaque personne est différente. Quand vous modélisez un objet, ses caractéristiques sont nommées champs, attributs, membres ou propriétés, selon les auteurs. De même, un objet modélisé peut réaliser des actions. Une personne vue sous l'angle client peut effectuer les actions « commander », « déménager » ou « payer ».

Ces actions, représentées par des fonctions, sont généralement nommées méthodes ou fonctions propres. Elles permettent d'agir sur les propriétés de l'objet. Vous utiliserez ici, une fois n'est pas coutume, le vocabulaire rencontré dans tous les outils de programmation Microsoft, qui consiste à définir un objet par ses propriétés et méthodes.

Ce vocabulaire est également employé dans JavaScript.

Une fois les différents types d'objets d'une application dégagés, leur représentation informatique abstraite est décrite dans une classe, aux moyens de variables, qui représentent les propriétés des objets, et de fonctions, qui représentent les méthodes. Ces variables et fonctions sont propres à la classe et ne devraient généralement être accessibles qu'aux objets.

C'est ce que l'on nomme l'encapsulation des données. Par abus de langage, vous rencontrerez

les termes « propriétés » et « méthodes » pour une classe, alors qu'elle contiendra des variables et des fonctions. La classe est donc le niveau d'abstraction le plus élevé pour la représentation d'une famille d'objets. En langage courant, on pourrait dire que la classe est le moule général, relativement flexible, permettant la fabrication d'autant d'objets que désiré, objets du même type, mais pas nécessairement identiques.

3- Le concept de classe

Un objet est un représentant de la classe à partir de laquelle il est créé. On dit qu'il est une instance de cette classe. L'objet créé a des propriétés correspondant aux variables de la classe et des méthodes qui correspondent aux fonctions de la classe. Il se distingue des autres objets grâce aux valeurs de ses propriétés. Si la classe représente un client humain, elle peut avoir quelque six milliards d'instances, toutes différentes.

Le mode opératoire d'utilisation des classes et des objets dans PHP 5 doit respecter les étapes suivantes :

1. Créez une classe de base pour l'objet. Elle doit avoir autant de variables que vous désirez de propriétés pour l'objet. Elle contient les fonctions qui permettent d'agir sur les propriétés.

2. Créez autant d'objets que nécessaire à partir du modèle défini par la classe.

3. Définissez une valeur particulière pour une ou plusieurs des propriétés de chaque objet que vous venez de créer. Vous verrez que cette définition peut aussi se faire lors de la création de l'objet à l'aide d'un constructeur.

4. Utilisez les objets et manipulez-les, généralement à l'aide des méthodes définies dans la classe.

4- Classe et instance

Les opérations de base pour l'utilisation des objets sont la création d'une classe et la définition de ses propriétés et des méthodes qui vont permettre aux objets créés à partir de la classe d'agir sur leurs propriétés ou de communiquer avec leur environnement.

Vient ensuite la création des objets proprement dits en tant qu'instances de la classe de base.

5- Création d'une classe

Pour créer une classe avec PHP 5, procédez de la façon suivante :

1. Déclarez la classe à l'aide du mot-clé class suivi du nom que vous souhaitez lui attribuer.

2. Ouvrez un bloc de code à l'aide d'une accolade ouvrante contenant l'intégralité de la définition de la classe.

3. Déclarez les variables propres de la classe comme des variables ordinaires, avec les mêmes règles de nommage et le caractère $ obligatoire. Chaque variable doit être précédée d'un modificateur d'accès précisant les possibilités d'accès à sa valeur.

Nous reviendrons sur les choix possibles. Dans un premier temps, faites précéder chaque variable du mot-clé public. Les variables peuvent être initialisées avec des valeurs de n'importe quel type reconnu par PHP. En particulier, une variable peut être utilisée comme un tableau créé à l'aide de la fonction array(). L'utilisation d'autres fonctions PHP ou d'expressions variables est en revanche interdite pour affecter une valeur à une variable. Le nombre de variables déclarées dans une classe n'est pas limité.

4. Déclarez les fonctions propres de la classe en suivant la même procédure que pour les fonctions personnalisées à l'aide du mot-clé function, précédé, comme les variables, d'un spécificateur d'accès (par défaut le mot-clé public). Le nom des fonctions propres ne doit pas être le même que celui de la classe qui les contient, faute de quoi la fonction concernée aura un rôle particulier, comme vous le verrez ultérieurement dans ce chapitre. Il ne doit pas non plus commencer par deux caractères de soulignement (__), cette notation étant réservée à certaines fonctions particulières de PHP 5.

5. Terminez le bloc de code de la classe par une accolade fermante.

L'ensemble de ces étapes est résumé dans la syntaxe générale de création d'une classe de l'exemple ci-dessous.

PHP 5 permet de définir des constantes propres à la classe à l'aide du mot-clé const suivi du nom de la constante puis de sa valeur. Ces constantes peuvent avoir le même nom que d'autres qui auraient été définies en dehors d'une classe avec la fonction define(). En contrepartie, elles ne sont pas accessibles à l'extérieur de la classe avec leur seul nom.

```php
<?php
class ma_classe
{
  //Définition d'une constante
  const lang="PHP 5"; ←❶
  //Définition des variables de la classe
  public $prop1; ←❷
  public $prop2 ="valeur"; ←❸
  public $prop3 = array("valeur0","valeur1"); ←❹
  //Initialisation interdite avec une fonction PHP
  //public $prop4= date("  d : m : Y"); Provoque une erreur fatale
  //***********************************
  //Définition d'une fonction de la classe
  public function ma_fonction($param1,$paramN) ←❺
  {
    //Corps de la fonction
  }
}
//fin de la classe
?>
```

Dans ce code, la variable $prop1 est déclarée mais pas initialisée, la variable $prop2 est déclarée et initialisée avec une valeur de type string, et la variable $prop3 est un tableau initialisé en utilisant la fonction array().

La classe contient de plus une fonction nommée ma_fonction() déclarée public, qui a la structure habituelle d'une fonction personnalisée et dont le corps peut utiliser des appels de fonctions natives de PHP.

L'exécution de ce code ne provoque aucun résultat visible, comme il se doit.

L'exemple ci-dessous présente la création d'une classe représentative d'une action boursière contenant des informations générales sur chaque action. La classe étant un modèle général, elle peut s'appliquer à toutes les actions cotées, quelle que soit la Bourse concernée.

Vous l'enrichirez par la suite avec d'autres propriétés et d'autres méthodes que celles de la classe d'origine. La classe nommée action contient une constante nommée PARIS définissant l'adresse de la Bourse de Paris, deux variables non initialisées, $nom, qui contiendront l'intitulé de l'action boursière, et $cours, qui en contiendra le prix, ainsi qu'une variable $bourse initialisée à la valeur "Bourse de Paris", qui représentera la place boursière par défaut.

Chaque variable peut donc avoir une valeur par défaut définie dans la classe de base, mais la valeur de chacune des propriétés reste entièrement modifiable pour chaque objet créé à partir de la classe, comme vous le verrez par la suite.

La classe action contient également une fonction propre définie par info(), qui, selon l'heure d'exécution du script et le jour de la semaine, indique si les bourses de Paris et de New York sont ouvertes ou non. Cette fonction n'utilise pour l'instant aucune des variables de la classe.

```php
<?php
class action
{
  //Constante
  const PARIS="Palais Brognard"; ←❶
  //variables propres de la classe
  public $nom; ←❷
  public $cours; ←❸
  public  $bourse="bourse de Paris "; ←❹

  //fonction propre de la classe
  public function info() ←❺
  {
    echo "Informations en date du ",date("d/m/Y H:i:s"),"<br>";
    $now=getdate();
    $heure= $now["hours"];
    $jour= $now["wday"];
    echo "<h3>Horaires des cotations</h3>";
    if(($heure>=9 && $heure <=17)&& ($jour!=0 && $jour!=6))
    {  echo "La Bourse de Paris est ouverte <br>";   }
    else
    {  echo "La Bourse de Paris est fermée <br>";   }
    if(($heure>=16 && $heure <=23)&& ($jour!=0 && $jour!=6) )
    {  echo "La Bourse de New York est ouverte <hr>";   }
    else
    {  echo "La Bourse de New York est fermée <hr>";   }
  }
}
?>
```

Le rôle de la POO étant de créer des bibliothèques de classes réutilisables par n'importe quel script, enregistrez le code de la classe action dans un fichier séparé. Pour l'utiliser, créez un fichier séparé destiné à utiliser cette classe en incorporant son code à l'aide de la fonction require(). Cette pratique est fortement recommandée.

Appel des méthodes et variable dans les instances de classe

Pour pouvoir utiliser en dehors de la création d'un objet instance de la classe, il vous faut faire appel à la syntaxe particulière suivante :

nom_classe::nom_fonction();

en précisant éventuellement ses paramètres, s'ils existent. Cela vous permet d'utiliser les fonctions propres d'une classe en dehors de tout contexte objet si elles sont déclarées public.

De même, vous pouvez accéder à la constante définie dans la classe en utilisant la même

syntaxe en dehors de la classe :

***echo "Constante PARIS =",nom_classe::nom_constante,"
";***

qui affichera bien « Palais Brognard ».

Dans la pratique professionnelle, vous avez toujours intérêt à séparer en deux fichiers distincts le code de création des classes et celui qui les utilise. Cela vous permet de réutiliser la même classe dans plusieurs scripts sans avoir à recopier le code de chacun d'eux.

5- Créer un objet

Vous venez de voir comment créer une classe, mais votre script ne dispose d'aucune fonctionnalité ni variable supplémentaire, ce qui a toutes les chances de produire un résultat

décevant. En reprenant l'analogie avec un moule, il vous faut maintenant utiliser le moule constitué par la classe pour créer des objets.

Chaque objet créé à partir de votre classe est appelé une instance de la classe. À chaque instance correspond un objet différent. Un objet particulier est identifié par un nom de variable tel que vous avez désormais l'habitude d'en écrire et est créé à l'aide du mot-clé

new selon le modèle suivant :

$var_objet = new nom_classe()

Prenez soin que le code de définition de la classe soit dans le même script ou soit inclus au début du script à l'aide des fonctions require() ou include().

À partir de maintenant, le script possède une variable $var_objet qui a les propriétés et les méthodes définies dans la classe de base.

Vous pouvez le vérifier en écrivant :

echo gettype($var_objet)

La valeur retournée par cette fonction est bien object, vous confirmant s'il en était besoin que la variable est d'un type nouveau par rapport à celles couramment utilisées jusqu'ici.

Le nom de la classe n'a pas besoin d'être défini par une expression explicite, comme vous venez de le faire, mais peut être contenu dans une variable chaîne de caractères.

La syntaxe suivante :

$maclasse ="nom_classe";

$var_objet = new $maclasse();

crée une instance de la même classe que le code précédent ce qui offre la possibilité de créer des objets dynamiquement, en fonction des saisies d'un visiteur du site par exemple.

Il est possible de vérifier si un objet particulier est une instance d'une classe donnée en utilisant l'opérateur *instanceof* pour créer une expression conditionnelle, selon la syntaxe suivante :

if($objet instanceof nom_classe) echo "OK";

Il vous faut maintenant personnaliser vos objets. Vous agissez pour cela sur les propriétés d'un objet que vous venez de créer afin de le distinguer d'un autre objet issu de la même classe.

Pour accéder, aussi bien en lecture qu'en écriture, aux propriétés d'un objet, PHP offre la syntaxe particulière suivante, propre aux variables objets. Pour utiliser la propriété déclarée dans la classe par $prop, appliquez la notation –> (caractère moins suivi du signe supérieur) sous la forme :

$nom_objet–>prop;

ou encore :

$nom_objet–>prop[n];

si la propriété prop de l'objet est un tableau.

Pour appeler une méthode de l'objet, appliquez la même notation :

$nom_objet–>nom_fonction();

Vous pouvez afficher la valeur d'une propriété ou lui en affecter une autre valeur. Par exemple, pour afficher la valeur de la propriété bourse d'un objet de type action, vous écrivez :

echo $action1–>bourse;

qui affiche la valeur par défaut « Bourse de Paris » définie dans la classe.

Pour lui affecter une nouvelle valeur, vous avez le code :

$action1–>bourse = "New York";

Pour appeler la seule méthode de l'objet :

$action1–>info()

L'exemple ci-dessous donne une illustration de la création d'objets à partir de la classe action puis de la définition des propriétés et enfin de l'utilisation de ces propriétés pour un affichage d'informations.

```
<?php
require("objet2.php"); ←❶
//Création d'une action
$action1= new action(); ←❷
//Affectation de deux propriétés
$action1->nom = "Mortendi"; ←❸
$action1->cours = 15.15; ←❹
//Utilisation des propriétés
echo "<b>L'action $action1->nom cotée à la $action1->bourse vaut $action1->cours
➥&euro;</b><hr>"; ←❺
//Appel d'une méthode
$action1->info(); ←❻
echo "La structure de l'objet \$action1 est :  <br>";
var_dump($action1); ←❼
echo "<h4>Descriptif de l'action</h4>";
foreach($action1 as $prop=>$valeur) ←❽
{
  echo "$prop = $valeur <br />";
}
if($action1 instanceof action) echo "<hr />L'objet \$action1 est du
➥type action"; ←❾
?>
```

Le code de la classe action est incorporé à l'aide de la fonction require() .

Vous créez ensuite une variable $action1 représentant un objet de type action et définissez des valeurs pour les propriétés nom et cours

Ces propriétés sont lues et utilisées pour créer un affichage.

L'appel de la méthode de l'objet permet d'obtenir des informations sur l'ouverture des bourses.

La fonction var_dump() permet d'afficher, à l'usage du programmeur uniquement, le nom, le type et la valeur de chaque propriété .

Plus élégamment, vous pouvez lire l'ensemble des propriétés de l'objet $action1 à l'aide d'une boucle foreach . L'utilisation de l'opérateur instanceof vous permet de vérifier que l'objet est bien une instance de la classe action.

6- Accès aux variables de la classe

Comme vous venez de le voir, les variables propres de la classe ne sont pas accessibles directement à l'extérieur du code qui définit la classe. Il est donc possible d'utiliser dans le script des variables qui ont les mêmes noms sans risquer de modifier les valeurs de celles de la classe. De même, l'accès habituel aux méthodes est impossible directement de l'extérieur de la classe. Cette particularité est nommée encapsulation et permet en quelque sorte de protéger la « cuisine » interne que vous avez conçue pour créer une classe.

De la même façon, si vous essayez d'utiliser dans une méthode une variable déclarée de la classe, vous n'obtenez aucun résultat.

Pour accéder à une constante de classe dans le corps d'une fonction, utilisez la syntaxe particulière suivante :

self::maconstante

ou encore :

nomclasse::maconstante

Pour accéder à cette constante à l'extérieur de la classe vous pouvez également utiliser cette dernière notation et, depuis PHP 5.3, la syntaxe suivante :

```
$classe="nomclasse";
echo $classe::maconstante;
```

Pour qu'une méthode accède aux variables déclarées dans la classe, elle doit y faire appel à l'aide de la syntaxe suivante :

```
$this–>mavar
```

dans laquelle la pseudo-variable $this fait référence à l'objet en cours, ce qui permet d'utiliser la variable $mavar dans la méthode. La méthode info() de votre classe action peut maintenant être enrichie et avoir comme fonctionnalité supplémentaire d'afficher toutes les caractéristiques d'un objet action.

7- Les modificateurs d'accessibilité

Vous avez défini jusqu'à présent des propriétés et des méthodes qui étaient accessibles librement à l'aide du mot-clé public. PHP 5 introduit des niveaux d'accessibilité différents pour vous permettre de limiter l'accès aux propriétés et aux méthodes et par là même de réduire le risque de modification des propriétés.

8- Accessibilité des propriétés

Il existe trois options d'accessibilité, qui s'utilisent en préfixant le nom de la variable de la classe. Ces options sont les suivantes :

• public. Permet l'accès universel à la propriété, aussi bien dans la classe que dans tout le script, y compris pour les classes dérivées, comme vous l'avez vu jusqu'à présent.

• protected. La propriété n'est accessible que dans la classe qui l'a créée et dans ses classes dérivées (voir la section « Héritage » de ce chapitre).

• private. C'est l'option la plus stricte : l'accès à la propriété n'est possible que dans la classe et nulle part ailleurs.

Le code ci-dessous teste les différents niveaux d'accessibilité aux propriétés. La classe acces contient trois propriétés, munies respectivement des modificateurs public, protected et private. La méthode lireprop() contenue dans la classe a accès à toutes ces propriétés, et ce quel que soit le modificateur utilisé.

La création d'un objet et l'appel de cette méthode affichent l'ensemble des propriétés.

L'appel de la propriété publique à partir de cet objet est possible et permet d'afficher sa valeur. Par contre, l'appel des propriétés protégées et privées provoquerait une erreur fatale. Par contre, une boucle foreach appliquée au tableau, contenant l'ensemble des propriétés de la classe acces, permet d'afficher l'ensemble de ces propriétés et leur valeur.

```
<?php
class acces
{
  //Variables propres de la classe
  public $varpub ="Propriété publique"; ←❶
  protected $varpro="Propriété protégée"; ←❷
  private $varpriv="Propriété privée"; ←❸
  function lireprop() ←❹
  {
    echo "Lecture publique: $this->varpub","<br />";
    echo "Lecture protégée: $this->varpro","<br />";
    echo "Lecture privée: $this->varpriv","<hr />";
  }
}
$objet=new acces(); ←❺
$objet->lireprop(); ←❻
echo $objet->varpub; ←❼
//echo $objet->varpriv; Erreur fatale ←❽
//echo $objet->varpro; Erreur fatale ←❾
echo "<hr />";
foreach(get_class_vars('acces') as $prop=>$val) ←❿
{
  echo "Propriété ",$prop ," = ",$val,"<br />";
}
?>
```

9- Accessibilité des méthodes

PHP 5 permet désormais de définir des niveaux d'accessibilité pour les méthodes des objets.

Vous retrouvez les mêmes modificateurs que pour les propriétés :

• public. La méthode est utilisable par tous les objets et instances de la classe et de ses classes dérivées.

• protected. La méthode est utilisable dans sa classe et dans ses classes dérivées, mais par aucun objet.

• private. La méthode n'est utilisable que dans la classe qui la contient, donc ni dans les classes dérivées, ni par aucun objet.

Tout appel d'une méthode en dehors de son champ de visibilité provoque une erreur fatale.

L'exemple ci-dessous illustre l'emploi de ces modificateurs dans une classe. Celle-ci contient un propriété déclarée private et trois méthodes déclarées, respectivement private, protected et public. Cette dernière appelle les deux autres méthodes. La création d'un objet et l'appel des différentes méthodes montrent que seule la méthode publique est utilisable par un objet. Le fait de décommenter les deux dernières lignes du script pour utiliser les méthodes protégées et privées provoquerait une erreur fatale. Nous verrons à la section consacrée à l'héritage des exemples d'utilisation de méthodes protégées dans une sous-classe.

```php
<?php
class accesmath
{
  //Variables propres de la classe
  private $code="Mon code privé"; ←❶
  //Méthodes
  //Méthode privée
  private function lirepriv() ←❷
  {
    echo "Lire privée ",$this->code,"<br />";
  }
  //Méthode protégée
  protected function lirepro() ←❸
  {
    echo "Lire protégée ",$this->code,"<br />";
  }
  //Méthode publique
  public function lirepub() ←❹
  {
    echo "Lire publique : ",$this->code,"<br />";
    $this->lirepro(); ←❺
    $this->lirepriv(); ←❻
  }
}
//*********************************
//Appels des méthodes
$objet=new accesmath(); ←❼
$objet->lirepub(); ←❽
//$objet->lirepro();//Erreur fatale ←❾
//$objet->lirepriv();//Erreur fatale ←❿
?>
```

10- Propriétés et méthodes statiques

PHP 5 introduit la notion de propriété et de méthode statique, qui permet d'accéder à ces éléments sans qu'il soit besoin de créer une instance de la classe. Pour déclarer une propriété ou une méthode statique, vous devez faire suivre le mot-clé définissant l'accessibilité du mot-clé static.

Comme les méthodes statiques sont utilisables sans la création d'objet, vous ne devez pas utiliser la pseudo-variable $this pour faire référence à une propriété de la classe dans le corps de la méthode. Vous devez utiliser à la place une des syntaxes suivantes :

self::$propriété

si la méthode est celle de la même classe, ou encore :

nomclasse::$propriété

si la méthode est celle d'une autre classe.

Notez qu'il faut conserver le signe $ pour désigner la propriété, contrairement à ce que vous faisiez précédemment.

De même, pour appeler une méthode statique de la classe à partir d'une autre méthode, vous utilisez les mêmes syntaxes, avec les mêmes conditions que ci-dessus :

self::$méthode()

nomclasse::$méthode()

Si vous créez un objet instance de la classe, la propriété déclarée static n'est pas accessible à l'objet en écrivant le code $objet–>propriété.

Par contre, les méthodes statiques sont accessibles par l'objet avec la syntaxe habituelle

$objet–>méthode().

Si vous modifiez la valeur d'une propriété déclarée statique à partir d'un objet, cette modification n'est pas prise en compte par les méthodes qui utilisent cette propriété. Il y a donc un danger de confusion difficile à localiser puisque aucune erreur n'est signalée.

L'exemple 9-8 présente ces différentes caractéristiques et leur emploi. Vous y créez une classe nommée info contenant une propriété statique $bourse initialisée. Une méthode statique n'utilisant aucune propriété retourne simplement l'heure en cours. Vous créez ensuite une méthode, également statique, qui utilise la propriété $bourse et la méthode précédente au moyen des notations self:: et info::.

11- Constructeur et destructeur d'objet

Dans ce qui précède, vous avez créé des objets en instanciant la classe action puis avez défini les propriétés des objets ainsi créés. Cette méthode est un peu lourde, car elle implique de définir les propriétés une par une. Il existe une façon plus élégante et plus rapide de créer des objets et de définir leurs propriétés en une seule opération. Elle consiste à créer un constructeur d'objet, qui n'est rien d'autre qu'une fonction spéciale de la classe, dont les paramètres sont les valeurs que vous voulez attribuer aux propriétés de l'objet.

PHP 5 permet désormais de créer des constructeurs unifiés avec la méthode __construct(), dont la syntaxe est la suivante :

void __construct(divers $argument1,…,argumentN)

Cette méthode, dite « méthode magique » comme toutes celles qui commencent par deux caractères de soulignement (__), porte le même nom, quelle que soit la classe, ce qui permet des mises à jour sans avoir à modifier le nom du constructeur. Elle ne retourne aucune valeur et est utilisée généralement pour initialiser les propriétés de l'objet et éventuellement pour afficher un message de bonne fin.

Elle est appelée automatiquement lors de la création d'un objet à l'aide du mot-clé new suivi du nom de la classe et des paramètres du constructeur, en utilisant la syntaxe suivante :

$mon_objet = new nom_classe(param1,param2,….)

Vous avez créé un objet nommé $mon_objet et initialisé chacune de ses propriétés avec les valeurs des paramètres passés à la fonction.

Exemple : classe action ayant un constructeur et un destructeur

```
<?php
class action
{
  private $propnom;
  private $propcours;
  protected $propbourse;
  function __construct($nom,$cours,$bourse="Paris") ←❶
  {
    $this->propnom=$nom; ←❷
    $this->propcours=$cours; ←❸
    $this->propbourse=$bourse; ←❹
  }
  function __destruct()
  {
    echo "L'action $this->propnom n'existe plus!<br />"; ←❺
  }
}
//Création d'objets
$alcatel = new action("Alcatel",10.21); ←❻
$bouch = new action("Bouch",9.11,"New York"); ←❼
$bim = new action("BIM",34.50,"New York"); ←❽
$ref=&$bim; ←❾
var_dump($alcatel); ←❿
echo "<hr />";
unset($alcatel); ← ⓫
unset($bim); ← ⓬
echo "<hr /><h4> FIN du script </h4><hr />"; ←⓭
?>
```

12- Déréférencement

nous avons vu que l'appel d'une méthode d'objet se faisait selon la syntaxe suivante :

$varobj–>methode() ;

Dans le cas où la méthode appliquée à un objet retourne elle-même un objet et que celui-ci possède ses propres méthodes, il est possible avec PHP 5 de pratiquer le déréférencement.

Cela permet d'enchaîner les appels de méthodes les uns à la suite des autres.

Vous pouvez écrire le code suivant :

$varobj–>methode1()–>methode2();

à condition que methode2() soit une méthode de l'objet obtenu par l'appel de methode1().

```
<?php
class varchar ←❶
{
  private $chaine;
  function __construct($a) ←❷
  {
    $this->chaine= (string)$a;
  }
  function add($addch)
  {
    $this->chaine.=$addch; ←❸
    return $this; ←❹
  }
  function getch()
  {
    return  $this->chaine; ←❺
  }
}
//Création d'objet
$texte=new varchar("Apache "); ←❻
echo $texte->getch(),"<hr />"; ←❼
echo $texte->add( " PHP 5 ")->getch(),"<hr />"; ←❽
echo $texte->add(" MySQL ")->add("SQLite ")->getch(),"<hr />"; ←❾
?>
```

13- Typage des paramètres

Vous savez que PHP est un langage peu typé et qu'il n'est pas possible de fixer le type d'un paramètre dans une fonction personnalisée. Cependant PHP 5 introduit une nuance en permettant désormais d'imposer un type au paramètre d'une méthode, mais uniquement pour les paramètres qui sont de type ***object ou array***

et pas encore pour les types de base de PHP, comme les types string ou integer.

Pour imposer le type d'un paramètre, vous devez faire précéder le nom de la variable par le nom de la classe dont le paramètre doit être une instance.

Vous écrivez, par exemple :

function (action $var)

{

//Corps de la fonction

}

Dans ce cas, le paramètre $var doit être un objet instancié à partir de la classe action et d'aucune autre. Si le type de la variable n'est pas conforme, une erreur fatale est générée.

14- Héritage

Vous avez considéré une classe comme un moule réutilisable à l'infini pour créer des objets. Que se passe-t-il si le moule ne convient plus, par exemple, parce que vous voulez créer des objets plus perfectionnés ? Faut-il le casser et en reconstruire entièrement un autre ? Vous pouvez aussi

vouloir faire évoluer une classe en lui ajoutant de nouvelles fonctionnalités, que ce soit des propriétés ou des méthodes, sans pour autant devoir modifier le code de la classe d'origine. À l'instar des langages objet plus perfectionnés, PHP 5 donne la possibilité de dériver de nouvelles classes à partir d'une classe donnée, dont elles seront des améliorations.

15- Enrichir un objet

La définition d'une classe contient toutes les déclarations des propriétés d'un objet instancié. Une fois l'objet créé, vous pourriez vous attendre que le nombre de propriétés soit fixé définitivement. Or il n'en est rien. Vous pouvez ajouter des propriétés à un objet en cours de script sans avoir à modifier la classe. Il vous suffit pour cela d'utiliser la notation d'affectation d'une propriété sous la forme suivante :

$objet–>propriété = "valeur"

16- Création d'une classe dérivée

Le mécanisme de l'héritage est fondamental en POO. Il vous permet, en fonction des besoins, de faire évoluer une classe sans la modifier en créant une classe dérivée, on dit aussi une classe enfant, ou une sous-classe, à partir d'une classe de base, ou classe parente. La classe dérivée hérite des caractéristiques (propriétés et méthodes) de la classe parente, et vous lui ajoutez des fonctionnalités supplémentaires. Vous pouvez ainsi créer toute une hiérarchie de classes en spécialisant chaque classe selon vos besoins. Contrairement à d'autres langages, PHP 5 n'autorise que l'héritage simple, une classe ne pouvant hériter que d'une seule classe parente.

Pour créer une classe enfant, faites suivre le nom de la nouvelle classe du mot-clé extends puis du nom de la classe parente, selon la forme suivante :

```
class classenfant extends classparent
{
//Propriétés et méthodes nouvelles
}
```

Dans le corps de la classe enfant, il est possible de redéfinir les propriétés et les méthodes de la classe parente, sauf si elles sont déclarées private ou final. Il est encore possible d'accéder aux propriétés et aux méthodes redéfinies de la classe parente en les faisant précéder du mot-clé parent::. Si elles ne sont pas redéfinies, la classe enfant possède les mêmes propriétés et les mêmes méthodes que la classe parente.

17- Les classes abstraites

PHP 5 fournit un degré supplémentaire d'abstraction des classes en permettant la création de classes et de méthodes abstraites. Une classe abstraite ne permet pas l'instanciation d'objets mais sert uniquement de classe de base pour la création de classes dérivées.

Elle définit en quelque sorte un cadre minimal auquel doivent se conformer les classes dérivées.

Une classe abstraite peut contenir des méthodes déclarées public ou protected, qu'elles soient elles-mêmes abstraites ou non. Une méthode abstraite ne doit contenir que sa signature, sans aucune implémentation. Chaque classe dérivée est chargée de créer sa propre implémentation de la

méthode. Une classe contenant au moins une méthode abstraite doit obligatoirement être déclarée abstraite, sinon elle permettrait de créer des objets qui auraient une méthode non fonctionnelle.

Pour créer une classe abstraite, faites précéder le mot-clé class du mot-clé abstract, comme ceci :

abstract class nomclasse

{

//Définition de la classe

}

Pour créer une méthode abstraite, faites également précéder le modificateur d'accès du mot-clé abstract, selon le modèle suivant :

abstract public function nomfonction() ;

Dans la classe qui dérive d'une classe abstraite, vous devez définir les modificateurs d'accessibilité des méthodes avec une visibilité égale ou plus large que celle de la méthode abstraite. Une classe abstraite définie, par exemple, protected, est implémentée dans les classes dérivées comme protected ou public.

L'exemple ci-dessous reprend la création des classes enfants action et emprunt de l'exemple précédent.

Il permet de réaliser la même opération mais à partir d'une classe abstraite nommée valeur. À la différence de l'exemple précédent, il n'est pas possible de créer des objets à partir de la classe valeur. Vous ne pouvez le faire qu'à partir de ses classes dérivées. Cette classe valeur contient deux propriétés et deux méthodes abstraites.

Chacune des classes dérivées doit donc créer sa propre implémentation complète de ces méthodes.

Il n'est plus question ici d'utiliser une méthode parente, comme dans l'exemple ci-dessous.

Le code s'en trouve alourdi, ce qui doit faire réfléchir avant d'utiliser des classes abstraites.

```
<?php
//Classe abstraite valeur
abstract class valeur ←1
{
  protected $nom;
  protected $prix;
```

```
  abstract  protected function __construct() ; ←❷
  abstract protected function getinfo(); ←❸
}
//Classe action
class action extends valeur ←❹
{
  private $bourse;
  function __construct($nom,$prix,$bourse="Paris") ←❺
  {
    $this->nom=$nom;
    $this->prix=$prix;
    $this->bourse=$bourse;
  }
  public function getinfo() ←❻
  {
     $info="Action $this->nom cotée à la bourse de $this->bourse <br />";
     $info.="Le prix de $this->nom est de $this->prix";
     return $info;
  }
}
//Classe emprunt
class emprunt extends valeur ←❼
{
  private $taux;
  private $fin;
  function __construct($nom,$prix,$taux,$fin) ←❽
  {
    $this->nom=$nom;
    $this->prix=$prix;
    $this->taux=$taux;
    $this->fin=mktime(24,0,0,12,31,$fin);
  }
  public function getinfo() ←❾
  {
    $reste=round(($this->fin-time())/86400);
    $info="Emprunt $this->nom au taux de  de $this->taux % <br />";
    $info.="Echéance : fin $fin (dans $reste jours)";
    return $info;
  }
}
//Création d'objets
$action1 = new action("Alcotel",9.75);
echo "<h4>", $action1->getinfo()," </h4>";
$action2 = new action("BMI",23.75,"New York");
echo "<h4>", $action2->getinfo() ,"</h4>";
$emprunt = new emprunt("EdF",1000,5.5,2012);
echo "<h4>", $emprunt->getinfo(),"</h4>";
?>
```

18- Les interfaces

La conception orientée objet s'accompagne d'une décomposition de l'application en modules élémentaires. L'introduction des interfaces dans PHP 5 permet cette décomposition en briques de base qu'une classe utilise comme modèle.

La notion d'interface est encore plus restrictive que celle de classe abstraite. Une interface ne doit contenir aucune déclaration de propriétés. C'est la classe qui l'implémente qui doit se charger de ces déclarations. Une interface ne contient aucune implémentation de méthode, à la différence d'une classe abstraite, qui peut contenir des méthodes entièrement définies.

Les méthodes qu'elle contient ne peuvent être déclarées qu'avec le modificateur public ou aucun modificateur, ce qui est équivalent.

L'interface ne fait que définir une structure à laquelle la classe qui l'implémente doit se conformer.

PHP 5 n'admet pas l'héritage multiple, mais une classe peut implémenter plusieurs interfaces.

La structure d'une interface doit respecter la forme suivante :

interface nom_interface

{

public function fonction1($var) ;

public function fonction2($var) ;

}

Pour implémenter une interface dans une classe, il faut faire suivre le nom de la classe du mot-clé implements à la place de extends puis des noms des interfaces séparés par des virgules.

Vous avez donc la structure suivante :

class nom_classe implements interface1,interface2

{

//Implémentation des méthodes des interfaces

}

Les interfaces définissant un cadre à respecter strictement, la classe qui les implémente doit obligatoirement définir toutes les méthodes des toutes les interfaces.

L'exemple ci-dessous crée une interface nommée abscisse qui déclare une méthode setx(), dont le rôle est d'initialiser une abscisse. Il crée également une interface nommée ordonnee déclarant une méthode sety(), dont le rôle est similaire pour l'ordonnée d'un point.

Ces interfaces sont implémentées par deux classes différentes. La classe pointaxe représente un point sur un axe, n'ayant donc qu'une seule coordonnée. Elle implémente l'interface abscisse et définit la méthode setx().

La classe pointplan représente un point du plan, avec deux coordonnées. Elle implémente donc les deux interfaces abscisse et ordonnee et définit les méthodes setx() et sety(), comme l'imposent les interfaces.

L'implémentation des interfaces dans une classe impose certes la définition de leurs méthodes mais n'empêche pas d'enrichir la classe avec d'autres méthodes. La classe pointplan possède une méthode supplémentaire module(), qui retourne la distance du point M(x,y) à l'origine du repère. Elle pourrait, par exemple, s'enrichir d'une méthode retournant l'angle (Ox,OM) et permettre ainsi la gestion des nombres

complexes.

```php
<?php
interface abscisse ←1
{
  public function setx($x); ←2
}
//
interface ordonnee ←3
{
   public function sety($y); ←4
}
//Classe
class pointaxe implements abscisse ←5
{
  public $x;
  public function setx($x) ←6
  {
    $this->x=$x;
  }
}
//
class pointplan implements abscisse,ordonnee ←7
{
  public $x;
  public $y;
  public function setx($x) ←8
  {
    $this->x=$x;
  }
  public function sety($y) ←9
  {
    $this->y=$y;
  }
  public function module() ←10
  {
    return sqrt($this->x*$this->x+$this->y*$this->y);
  }
}
//Création d'objets
$point1 = new pointaxe();
$point1->setx(21);
var_dump($point1);
echo "<hr />";
//
```

19- Méthode et classe finales

La création de sous-classes et la redéfinition des méthodes sont les éléments essentiels de la POO. Il peut cependant être nécessaire d'empêcher toute modification d'une méthode ou d'une classe, en particulier dans un projet où l'on travaille en équipe.

Pour interdire la redéfinition d'une méthode d'une classe parente dans ses classes dérivées, faits précéder le mot-clé function du mot-clé final.

Si vous définissez la classe suivante :

```php
class triangle
{
  private $x ;
  private $y ;
  private $z ;
  function __construct($x,$y,$z)
  {
    $this->x=$x ;
    $this->y=$y ;
    $this->z=$z ;
  }
  final function trianglerect()
  {
    if(($this->x*$this->x + $this->y*$this->y) ==($this->z*$this->z))
    ==return "Le triangle est rectangle";
    else echo "Triangle non rectangle";
  }
}
```

dans laquelle la fonction trianglerect() est déclarée final, il devient impossible de créer une classe dérivée qui contienne une méthode portant le même nom.

20- Clonage d'objet

La notion de clonage d'objet est une des nouveautés introduites dans PHP 5. Elle permet d'effectuer une copie exacte d'un objet mais en lui affectant une zone de mémoire différente de celle de l'objet original. Contrairement à la création d'une simple copie à l'aide de l'opérateur =

ou d'une référence sur un objet avec l'opérateur &, les modifications opérées sur l'objet cloné ne sont pas répercutées sur l'original.

Pour cloner un objet, utilisez le mot-clé clone selon la syntaxe suivante :

$objetclone = clone $objet ;

L'objet cloné a exactement les mêmes propriétés et les mêmes méthodes que l'original.

Après le clonage, les modifications opérées sur la variable $objet n'ont aucun effet sur le clone, et réciproquement.

Chapitre 3 Accès objet à MySQL avec PHP

1- Introduction

Une tendance évidente qui a prévalu dans le développement de PHP 5 est la programmation objet ; l'accès à MySQL n'y a pas échappé, et cela n'a fait que s'accentuer dans les versions successives de PHP 5. Cette possibilité est liée à l'utilisation de l'extension mysqli, dite « mysql améliorée », qui comprend les trois classes suivantes :

• La classe mysqli qui permet de créer des objets de type mysqli_object. Cette dernière possède pas moins de 33 méthodes et 15 propriétés pouvant remplacer ou compléter les fonctions de l'extension mysql comme la connexion à la base, l'envoi de requêtes, les transactions ou les requêtes préparées.

• La classe mysqli_result, permettant de gérer les résultats d'une requête SQL effectuée par l'objet précédent. Ses méthodes et propriétés sont également les équivalents des fonctions de l'extension mysql vues au chapitre 15.

• La classe mysql_stmt qui représente une requête préparée. Il s'agit là d'une nouveauté par rapport à l'extension mysql.

Nous allons maintenant reprendre des différents exemples du chapitre 15 et voir comment obtenir les mêmes résultats via un accès objet. Ce chapitre peut donc être abordé indépendamment du chapitre précédent.

2- Connexion au serveur MySQL

Comme il se doit, la première chose à faire est de se connecter au serveur MySQL. Pour cela nous créons un objet de la classe mysqli selon la syntaxe suivante :

$idcom = new mysqli (string $host, string $user, string $pass [,string $base]) ;

$idcom est un objet mysqli et $host, $user et $pass sont, comme dans le chapitre 15, le nom du serveur MySQL, le nom de l'utilisateur et le mot de passe. Le paramètre facultatif $base permet de choisir d'emblée la base sur laquelle seront effectuées les commandes SQL.

Nous pouvons également réutiliser le fichier myparam.inc.php, créé au chapitre 15, contentant les paramètres de connexions (ci-dessous en local) :

```
<?php
define("MYHOST","localhost");
define("MYUSER","root");
define("MYPASS","");
?>
```

L'objet $idcom représentant la connexion sera utilisé directement ou indirectement pour toutes les opérations à effectuer sur la base. Si la connexion n'est pas effectuée, la variable $idcom contiendra la valeur FALSE, permettant ainsi de tester si la connexion est bien réalisée.

La connexion prend fin quand l'exécution du script PHP est terminée, mais on peut y mettre fin explicitement pour observer le serveur MySQL. Si les résultats d'une requête sont entièrement

récupérés, la connexion peut en effet être coupée avec la méthode close() selon la syntaxe suivante :

boolean $idcom–>close()

Si le serveur comporte plusieurs bases de données et que le paramètre $base a été employé lors de la création de l'objet $idcom, il est possible de changer de base sans interrompre la connexion en cours en appelant la méthode select_db() avec la syntaxe suivante :

boolean $idcom–>select_db (string $base)

Cette méthode retourne un booléen qui permet de tester la bonne fin de l'opération. Si le paramètre $base n'a pas été précisé lors de la création de l'objet mysqli, c'est cette méthode qui permet de choisir la base.

En cours de script, vous pouvez à tout moment tester si la connexion est encore active en appelant la méthode ping() selon la syntaxe suivante :

boolean $idcom–>ping()

Celle-ci renvoie TRUE si la connexion est active, sinon elle renvoie FALSE et effectue une reconnexion avec les paramètres initiaux.

La structure d'un script accédant à MySQL est donc la suivante :

```
<?php
//Inclusion des paramètres de connexion
include_once("myparam.inc.php");
//Connexion au serveur
$idcom = new mysqli(MYHOST,MYUSER,MYPASS,"ma_base");
//Affichage d'un message en cas d'erreurs
if(!$idcom)
{
    echo "<script type=text/javascript>";
    echo "alert('Connexion Impossible à la base)</script>";
}
//*******************************
//Requêtes SQL sur la base choisie
//Lecture des résultats
//*******************************
//Fermeture de la connexion
$idcom->close();
?>
```

Comme nous l'avons fait pour l'accès procédural à MySQL, nous avons intérêt à créer une fonction de connexion au serveur que nous réutiliserons systématiquement dans tous les exemples qui suivent. C'est l'objet de l'exemple 16-2 qui crée la fonction connexobjet() dont les paramètres sont le nom de la base dans la variable $base et le nom du fichier .inc.php contenant les paramètres de connexion dans la variable $param.

La fonction inclut d'abord les paramètres de connexion puis crée un objet mysqli; elle vérifie ensuite que la connexion est bien réalisée, affiche un message d'alerte JavaScript

et sort de la fonction en cas de problème.

Si la connexion est bien réalisée elle retourne l'objet $idcom.

```php
<?php
function connexobjet($base,$param)
{
    include_once($param.".inc.php"); ←❶
    $idcom = new mysqli(MYHOST,MYUSER,MYPASS,$base); ←❷
    if (!$idcom) ←❸
    {
        echo "<script type=text/javascript>";
        echo "alert('Connexion Impossible à la base')</script>";
        exit(); ←❹
    }
    return $idcom; ←❺
}
?>
```

Chacun de vos scripts d'accès à la base doit donc contenir les lignes suivantes :

include("connexobjet.inc.php");

$idcom = connexobjet ("nom_base","myparam ");

L'opération de connexion est donc gérée en deux lignes.

3- Envoi de requêtes SQL au serveur

Les différentes opérations à réaliser sur la base MySQL impliquent l'envoi de requêtes SQL au serveur.

Pour envoyer une requête, il faut employer la méthode query() de l'objet mysqli dont la syntaxe est :

divers $idcom–>query (string $requete [,int mode])

La requête est soit directement une chaîne de caractères, soit une variable de même type.

Le paramètre mode est une constante qui prend la valeur MYSQLI_USE_RESULT ou MYSQLI_STORE_RESULT, cette dernière étant la valeur par défaut. Avec MYSQLI_STORE_RESULT, il est possible d'envoyer plusieurs requêtes sans libérer la mémoire associée à un premier résultat, tandis qu'avec l'autre méthode, il faut d'abord libérer cette mémoire à l'aide de la méthode free_result() appliquée à l'objet $result (de type mysqli_result).

La méthode query() retourne TRUE en cas de réussite et FALSE sinon et un objet de type mysqli_result pour les commandes SQL de sélection comme SELECT. Nous pouvons donc tester la bonne exécution d'une requête. En résumé, un script d'envoi de requête a la forme suivante :

```php
<?php
include_once("connexobjet.inc.php"); ←❶
$idcom=connexobjet("magasin","myparam"); ←❷
$requete="SELECT * FROM article ORDER BY categorie"; ←❸
$result=$idcom->query($requete); ←❹
if(!$result)
{
  echo "Lecture impossible"; ←❺
}
else
{
  //Lecture des résultats ←❻
  while ($row = $result->fetch_array(MYSQLI_NUM))
  {
   foreach($row as $donn)
   {
     echo $donn," ";
   }
   echo "<hr />";
  }
  //Destruction de l'objet $result
  $result->close();
}
// Fermeture de la connexion
$idcom->close(); ←❼
?>
```

Ce script effectue successivement l'inclusion du fichier connexobjet.inc.php, la connexion au serveur, l'écriture de la requête SQL dans la variable $requete, l'envoi de la requête et la récupération du résultat puis l'affichage d'un message d'erreur éventuel ou bien des résultats, procédure que nous détaillerons dans le paragraphe suivant et, enfin, la libération de la mémoire occupée par l'objet mysqli_result.

4- Lecture du résultat d'une requête

Pour les opérations d'insertion, de suppression ou de mise à jour de données dans une base, il est simplement utile de vérifier si la requête a bien été exécutée. Par contre, lorsqu'il s'agit de lire le résultat d'une requête contenant la commande SELECT, la méthode query() retourne un objet de type mysqli_result, identifié dans nos exemples par la variable $result. La classe mysqli_result offre une grande variété de méthodes permettant de récupérer des données sous des formes diverses, la plus courante étant un tableau.

Chacune de ces méthodes ne récupérant qu'une ligne du tableau à la fois, il faut recourir à une ou plusieurs boucles pour lire l'ensemble des données.

5- Lecture à l'aide d'un tableau

La méthode des objets instances de la classe mysqli_result la plus perfectionnée pour lire des données dans un tableau est fetch_array(), dont la syntaxe est :

array $result–>fetch_array (int type)

Elle retourne un tableau qui peut être indicé (si la constante type vaut MYSQLI_NUM), associatif (si type vaut MYSQLI_ASSOC), dont les clés sont les noms des colonnes ou les alias de la table interrogée, ou encore mixte, contenant à la fois les indices et les clés (si type vaut MYSQLI_BOTH). Pour lire toutes les lignes du résultat, il faut écrire une boucle (while par exemple) qui effectue un nouvel appel de la méthode fetch_array() pour chaque ligne.

Cette boucle teste s'il existe encore des lignes à lire, la méthode fetch_array() retournant la valeur NULL quand il n'y en a plus. Pour lire et afficher chaque ligne nous utilisons ensuite une boucle foreach. Notez encore une fois que si le tableau retourné est indicé, l'indice 0 correspond au premier attribut écrit dans la requête et ainsi de suite.

Les méthodes suivantes permettent également de récupérer une ligne de résultat à la fois :

array $result–>fetch_assoc(void)

retourne un tableau associatif dont les clés sont les noms des colonnes de la table.

array $result–>fetch_row(void)

donc les indices de 0 à N sont les positions des attributs dans la requête SQL.

L'exemple ci-dessous met cette méthode en pratique dans le but d'afficher le contenu de la table article de la base magasin dans un tableau XHTML. Après l'inclusion de la fonction de connexion puis son appel sur la base magasin nous écrivons la requête SQL de sélection.

L'envoi de la requête avec la méthode query() permet de récupérer un objet $result ou FALSE en cas d'échec. Un test sur la variable $result permet d'afficher un message d'erreur ou le contenu de la table. Nous récupérons d'abord le nombre d'articles dans la table grâce à la propriété num_rows de l'objet mysqli_result et affichons ce nombre dans un titre <h4>. Une boucle while permet de lire une ligne à la fois dans un tableau indicé, puis une boucle foreach permet d'afficher chacune des

valeurs du tableau dans un tableau XHTML. L'objet $result est alors supprimé et la connexion fermée.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Lecture de la table article</title>
  <style type="text/css" >
  table {border-style:double;border-width: 3px;border-color:red;
  ➥background-color: yellow;}
  </style>
 </head>
 <body>
<?php
include("connexobjet.inc.php"); ←1
$idcom=connexobjet("magasin","myparam"); ←2
$requete="SELECT * FROM article ORDER BY categorie"; ←3
$result=$idcom->query($requete); ←4
if(!$result) ←5
{
  echo "Lecture impossible"; ←6
}
else ←7
{
  $nbcol=$result->field_count;
  $nbart=$result->num_rows; ←8
  echo "<h3> Tous nos articles par cat&#233;gorie</h3>";
  echo "<h4> Il y a $nbart articles en magasin </h4>"; ←9
```

```
  echo "<table border=\"1\">";
  echo "<tr><th>Code article</th> <th>Description</th> <th>Prix</th
  ➥<th>Cat&#233;gorie</th></tr>";
  while($ligne=$result->fetch_array(MYSQLI_NUM)) ←10
  {
    echo "<tr>";
    foreach($ligne as $valeur) ←11
    {
      echo "<td> $valeur </td>";
    }
    echo "</tr>";
  }
  echo "</table>";
}
$result->close(); ←12
$idcom->close(); ←13
?>
</body>
</html>
```

6- Lecture des noms de colonnes.

Sa syntaxe est :

array $result–>fetch_fields(void)

Le tableau retourné contient autant d'objets qu'il existe de colonnes dans la requête SQL.

Exemple

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Lecture de la table article</title>
  <style type="text/css" >
  table {border-style:double;border-width: 3px;border-color:red;
  ➥background-color: yellow;}
  </style>
 </head>
<body>
<?php
include("connexobjet.inc.php");
$idcon=connexobjet("magasin","myparam");
//
$requete="SELECT id_article AS 'Code article',designation AS 'Désignation',prix
➥AS 'Prix Unitaire',categorie AS 'Catégorie' FROM article WHERE designation
➥LIKE '%Sony%' ORDER BY categorie"; ←❶
//
$result=$idcon->query($requete); ←❷
//
if(!$result)
{
   echo "Lecture impossible";
}
else
{
  $nbart=$result->num_rows; ←❸
  $titres=$result->fetch_fields(); ←❹
  echo "<h3> Tous nos articles de la marque Sony</h3>";
  echo "<h4> Il y a $nbart articles en magasin </h4>";
  echo "<table border=\"1\"> <tr>";
  //Affichage des titres
  foreach($titres as $colonne) ←❺
  {
     echo "<th>", htmlentities($colonne->name) ,"</th>"; ←❻
  }
   echo "</tr>";
   //Lecture des lignes de résultat
  while($ligne=$result->fetch_array(MYSQLI_NUM)) ←❼
  {
    echo "<tr>";
    foreach($ligne as $valeur)
    {
      echo "<td> $valeur </td>";
    }
    echo "</tr>";
  }
  echo "</table>";
}
$result->free_result();
$idcon->close();
?>
</body>
</html>
```

7- Récupération des valeurs dans un objet

Les objets de type mysqli_result possèdent la méthode fetch_object() dont la syntaxe est :

object $result–>fetch_object()

À chaque appel de cette méthode, l'objet retourné représente une ligne de résultat et possède autant de propriétés qu'il existe d'attributs dans la requête SQL SELECT ; les noms de ces propriétés sont ceux des colonnes de la table ou des alias éventuels.

Si nous récupérons une ligne de résultat dans la variable $ligne :

$ligne=$result–>fetch_object()

La valeur d'un attribut nom est lue avec la ligne de code : ***$ligne–>nom***

```php
<?php
include("connexobjet.inc.php");
$idcom=connexobjet('magasin','myparam');
if(!empty($_POST['nom'])&& !empty($_POST['adresse'])&&
!empty($_POST['ville'])) ←❶
{
  $id_client="\N"; ←❷
  $nom=$idcom->escape_string($_POST['nom']); ←❸
  $prenom=$idcom->escape_string($_POST['prenom']); ←❹
  $age=$idcom->escape_string($_POST['age']); ←❺
  $adresse=$idcom->escape_string($_POST['adresse']); ←❻
  $ville=$idcom->escape_string($_POST['ville']); ←❼
  $mail=$idcom->escape_string($_POST['mail']); ←❽
  //Requête SQL
  $requete="INSERT INTO client VALUES('$id_client','$nom','$prenom','$age',
  '$adresse','$ville','$mail')";
  $result=$idcom->query($requete); ←❾
  if(!$result)
  {
    echo $idcom->errno;
    echo $idcom->error;
    echo "<script type=\"text/javascript\">
    alert('Erreur : ".$idcom->error."')</script>";
  }
  else
  {
    echo "<script type=\"text/javascript\">
    alert('Vous êtes enregistré Votre numéro de client est : ".
    $idcom->insert_id."')</script>"; ←❿
  }
}
else {echo "<h3>Formulaire à compléter!</h3>";}
?>
```

8- Les requêtes préparées

Nous avons déjà construit des requêtes SQL dynamiquement à partir d'informations saisies par l'utilisateur dans l'exemple précédent. Les requêtes préparées permettent de créer des requêtes SQL qui ne sont pas directement utilisables mais qui contiennent des paramètres auxquels on peut donner des valeurs différentes en fonction des besoins, pour des appels répétitifs par exemple. Une requête préparée peut se présenter sous la forme suivante :

SELECT prenom,nom FROM client WHERE ville=? AND id_client>=?

Dans cette requête, les caractères ? vont être remplacés par des valeurs quelconques en fonction des besoins du visiteur.

La démarche à suivre pour utiliser une requête préparée est la suivante :

1. Écrire la chaîne de requête comme paramètre de la méthode prepare() de l'objet mysqli. Cette dernière retourne un objet de type mysqli_stmt qui représente la requête préparée.

2. Lier les paramètres dans l'ordre de leur apparition avec des valeurs ou des variables à l'aide de la méthode bind_param() de l'objet mysqli_stmt, selon la syntaxe :

$mysqli_stmt–>bind_param(string $types, $param1,...$paramN).

La chaîne $types est la concaténation de caractères indiquant le type de chacun des paramètres. La signification de ces caractères est présentée au tableau ci-dessous.

Caractère	Définition
i	Entier (integer)
d	Décimal
s	Chaîne de caractères (string)
b	Blob (texte long)

3. Pour trois paramètres qui seraient, dans l'ordre d'apparition dans la requête, un décimal,

une chaîne et un entier,

la chaîne $types

serait par exemple

"dsi".

4. Exécuter la requête en appelant la méthode execute() de l'objet mysqli_stmt. 5. Pour les requêtes préparées qui retournent des résultats, comme SELECT, il faut ensuite lier les résultats à des variables PHP, avec la méthode bind_result() selon la syntaxe :

mysqli_stmt–>bind_result($var1,...$varN)

avec autant de paramètres qu'il existe de colonnes lues dans la requête. L'objet mysqli_result obtenu contient alors toutes les lignes du résultat.

6. Lire ces lignes à l'aide d'une boucle en appliquant la méthode fetch() à cet objet et utiliser ces résultats pour un affichage avec les noms des variable définies à l'étape 4.

7. Libérer la mémoire occupée par l'objet mysqli_stmt avec la méthode free_result().

Chapitre 4 PDO et MySQL

1- Introduction

PDO (PHP Data Objects) est une extension qui offre une couche d'abstraction de données introduite dans PHP 5, ce qui signifie qu'elle n'est pas liée, comme les extensions mysql ou mysqli que nous avons abordées dans les deux chapitres précédents, mais indépendante de la base de données utilisée. Grâce à elle, le code devient portable très facilement, c'est-à-dire qu'elle modifie uniquement la ligne de connexion, d'une base de données MySQL à SQLite, PostgreSQL, Oracle et d'autres encore par exemple.

PDO offre donc aussi bien l'avantage de la portabilité, de la facilité d'utilisation que de la rapidité.

L'extension PDO comprend les trois classes suivantes :

• La classe PDO, qui permet de créer des objets représentant la connexion à la base et qui dispose des méthodes permettant de réaliser les fonctions essentielles, à savoir l'envoi de requête, la création de requêtes préparées et la gestion des transactions.

• La classe PDOStatement, qui représente, par exemple, une requête préparée ou un résultat de requête SELECT.

Ses méthodes permettent de gérer les requêtes préparées et de lire les résultats des requêtes.

• La classe PDOException qui permet de gérer et d'afficher des informations sur les erreurs à l'aide d'objets.

Ce chapitre reprend exactement la même démarche que le précédent en utilisant exclusivement PDO ; nous allons y reprendre les différents exemples des chapitres 15 et 16 et voir comment obtenir les mêmes résultats via un accès objet PDO.

Il peut donc être abordé indépendamment des deux chapitres précédents.

Connexion au serveur MySQL

Bien entendu, la première chose à faire est de se connecter au serveur. Pour cela nous créons un objet de la classe PDO en utilisant le constructeur de la classe PDO dont les paramètres changent selon le serveur

auquel nous nous connectons. Par exemple :

• Pour MySQL :

```
$idcom= new PDO("mysql:host=$host;dbname=$base ,$user,$pass) ;
```

• Pour SQLite :

```
$idcom= new PDO("sqlite2:/chemin/rep/$base") ;
```

• Ou encore pour PostgreSQL, que nous n'utiliserons pas, ce qui illustre la portabilité de PDO :

```
$idcom= new PDO("pgsql:host=$host port=5432 dbname=$base user=$user password=$pass").
```

Dans ces exemples, $idcom est un objet PDO. $host, $user, $pass et $base, quant à eux désignent, comme dans les chapitres précédents, le nom du serveur, l'utilisateur, le mot de passe et le nom de la base.

Nous pouvons également réutiliser le fichier myparam.inc.php créé au chapitre 15, contenant les paramètres de connexion (ci-dessous en local) :

```
<?php
define("MYHOST","localhost");
define("MYUSER","root");
define("MYPASS","");
?>
```

L'objet $idcom représentant la connexion sera utilisé directement ou indirectement pour toutes les opérations à effectuer sur la base. Si la connexion n'est pas effectuée, la variable $idcom est un booléen qui contient la valeur FALSE, ce qui permet de tester si la connexion est bien réalisée.

La connexion prend fin quand l'exécution du script PHP est terminée, mais on peut mettre fin explicitement à la connexion pour libérer le serveur MySQL. Si les résultats d'une requête sont entièrement récupérés, la connexion peut en effet être détruite en donnant à la variable $idcom la valeur NULL.

La structure de principe d'un script accédant à MySQL est donc la suivante :

```
<?php
//Inclusion des paramètres de connexion
include_once("myparam.inc.php");
//Connexion au serveur
$dsn="mysql:host=".MYHOST.";dbname=".$base;
$user=MYUSER;
$pass=MYPASS;
$idcom = new PDO($dsn,$user,$pass);
//Contrôle de la connexion
if(!$idcom)
{
  echo "Erreur";
}
//*****************************
//Requêtes SQL sur la base choisie
//Lecture des résultats
//*****************************
//Fermeture de la connexion
$idcom=NULL;
?>
```

Comme nous l'avons fait pour l'accès procédural à MySQL, nous avons intérêt à créer une fonction de connexion au serveur que nous réutiliserons systématiquement dans tous les exemples qui suivent. C'est l'objet de l'exemple ci-dessous qui crée la fonction connexpdo(), dont les paramètres sont le nom de la base dans la variable $base et le nom du fichier .inc.php qui contient les paramètres de connexion dans la variable $param.

La fonction inclut d'abord les paramètres de connexion, définit la chaîne DSN (Data Source Name) pour MySQL, puis crée un objet PDO dans un bloc try. En cas d'échec, un bloc catch crée un objet PDOException qui permet d'afficher un message d'erreur en appelant la méthode getMessage().

Si la connexion est bien réalisée elle retourne l'objet $idcom ou FALSE dans le cas contraire.

```php
<?php
function connexpdo($base,$param)
{
  include_once($param.".inc.php"); ←❶
  $dsn="mysql:host=".MYHOST."; ←❷
dbname=".$base;
  $user=MYUSER;
  $pass=MYPASS;
  try
  {
    $idcom = new PDO($dsn,$user,$pass); ←❸
    return $idcom;
  }
  catch(PDOException $except) ←❹

  {
    echo"Echec de la connexion",$except->getMessage(); ←❺
    return FALSE;
    exit();
  }
}
?>
```

Chacun de vos scripts d'accès à la base doit de ce fait contenir les lignes suivantes :

include("connexpdo.inc.php")

$idcom = connexpdo ("nom_base","myparam ") ;

L'opération de connexion est donc là aussi gérée en 2 lignes, quel que soit le script ou le type de base.

2- Envoi de requêtes SQL au serveur

Les différentes opérations à réaliser sur la base MySQL impliquent l'envoi de requêtes SQL au serveur.

Pour envoyer une requête au serveur, nous avons le choix entre plusieurs méthodes.

Pour celles qui ne retournent pas de résultats, il existe la méthode exec() des objets PDO dont la syntaxe est la suivante :

integer $idcom–>exec(string requete)

Elle est utilisée pour les requêtes INSERT, UPDATE ou DELETE par exemple. Elle retourne simplement un entier qui correspond au nombre de lignes concernées par la requête.

Pour les requêtes qui vont retourner des résultats, il faut employer la méthode query(), dont la syntaxe est :

object $idcom–>query(string $requete)

Elle retourne FALSE en cas d'erreur ou, sinon, un objet de la classe PDOStatement représentant l'ensemble des lignes de résultats – pour lequelles il faut ensuite appeler les méthodes spécialisées pour afficher les valeurs.

En résumé, un script d'envoi de requêtes se présente sous la forme suivante :

```php
include_once("connexpdo.inc.php"); ←❶
$idcon=connexpdo("magasin","myparam"); ←❷
//Requête sans résultats
$requete1="UPDATE client SET age=43 WHERE id_client=7"; ←❸
$nb=$idcon->exec($requete1); ←❹
echo "<p>$nb ligne(s)  modifi&#233;e(s)<hr /></p>"; ←❺
//Requête avec résultats
$requete2="SELECT * FROM client ORDER BY nom"; ←❻
$result=$idcon->query($requete2); ←❼
if(!$result) ←❽
{
  $mes_erreur=$idcon->errorInfo();
  echo "Lecture impossible, code", $idcon->errorCode(),$mes_erreur[2];
}
else ←❾
{
  while ($row = $result->fetch(PDO::FETCH_NUM))
  {
    foreach($row as $donn)
    {
      echo $donn," ";
    }
    echo "<hr />";
  }
  $result->closeCursor(); ←❿
}
$idcon=null;
?>
```

Il effectue successivement l'inclusion du fichier connexpdo.inc.php, la connexion au serveur, l'écriture d'une première requête SQL dans la variable $requete1 et son envoi à l'aide de la méthode exec(), qui retourne le nombre de lignes affectées dans la variable $nb et son affichage. Une seconde requête contenant la commande SELECT est envoyée via la méthode query() qui retourne un résultat dans la variable $result – qui est un objet de type PDOStatement. Un test permet de vérifier la bonne exécution de la requête et l'affichage des résultats au moyen de méthodes que nous développerons dans les sections suivantes. Enfin, pour libérer la mémoire occupée par le résultat obtenu, nous utilisons la méthode closeCursor() de l'objet PDOStatement, surtout utile avant d'envoyer une nouvelle requête SELECT au serveur.

3- Lecture du résultat d'une requête

Pour les opérations d'insertion, de suppression ou de mise à jour des données dans une base, il est utile de vérifier si la requête a bien été exécutée ou, de vérifier le nombre de lignes affectées.

En revanche, lorsqu'il s'agit de lire le résultat d'une requête contenant la commande SELECT, la méthode query() retourne un objet de type PDOStatement identifié dans nos exemples par la variable $result. La classe PDOStatement offre une variété de méthodes qui permettent de récupérer des données sous des formes diverses, la plus courante étant un tableau mais cela peut également être un objet.

4- Lecture à l'aide d'un tableau

La méthode des objets PDOStatement la plus courante pour lire des données dans un tableau est fetch() dont la syntaxe est :

array $result–>fetch(integer type)

Elle retourne un tableau qui peut être indicé (si la constante type vaut PDO::FETCH_NUM), associatif (si type vaut PDO::FETCH_ASSOC), dont les clés sont les noms des colonnes ou les alias de la table interrogée, ou encore mixte contenant à la fois les indices et les clés (si type vaut PDO::FETCH_BOTH). Pour lire toutes les lignes du résultat, il faut écrire une boucle (while par exemple) qui effectue un nouvel appel de la méthode fetch() pour chaque ligne. Cette boucle teste s'il y a encore des lignes à lire, la méthode fetch() retournant la valeur NULL quand il n'y en a plus. Pour lire et afficher chaque ligne, nous utilisons ensuite une boucle foreach. Notez encore une fois

que si le tableau retourné est indicé, l'indice 0 correspond au premier attribut écrit dans la requête, et ainsi de suite.

Nous pouvons également récupérer toutes les lignes de résultats dans un seul tableau multidimensionnel pour lequel le premier niveau est indicé de 0 à N-1 pour lire N lignes, chaque élément étant lui même un tableau qui peut être indicé, associatif ou mixte selon la valeur du paramètre type qui prend les mêmes valeurs que pour la méthode fetch(). Il s'agit de la méthode fetchAll() qui sera mise en œuvre plus loin et dont la syntaxe est :

array $result–>fetchAll(integer type)

L'exemple ci-dessous met cette méthode en pratique dans le but d'afficher le contenu de la table article de la base magasin dans un tableau XHTML. Après l'inclusion de la fonction de connexion, puis son appel sur la base magasin, nous écrivons la requête SQL de sélection (requête).
L'envoi de la requête avec la méthode query() permet de récupérer un objet $result ou FALSE en cas d'échec. Un test sur la variable $result permet d'afficher un message d'erreur ou le contenu de la table. Nous récupérons d'abord le nombre d'articles dans la table grâce à la méthode rowCount() de l'objet PDOStatement et affichons ce nombre dans un titre <h4>. Une boucle while permet de lire une ligne à la fois dans un tableau indicé, puis une boucle foreach permet d'afficher chacune des valeurs du tableau dans un tableau XHTML. L'objet $result est alors supprimé et la connexion fermée.

```
<?php
include("connexpdo.inc.php"); ←❶
if($idcom=connexpdo("magasin","myparam")) ←❷
{
  $requete="SELECT * FROM article ORDER BY categorie"; ←❸
  $result=$idcom->query($requete); ←❹
  if(!$result) ←❺
  {
   $mes_erreur=$idcom->errorInfo();
   echo "Lecture impossible, code", $idcom->errorCode(),$mes_erreur[2]; ←❻
  }
  else ←❼
  {
  $nbart=$result->rowCount(); ←❽
  echo "<h3> Tous nos articles par cat&#233;gorie</h3>";
  echo "<h4> Il y a $nbart articles en magasin </h4>"; ←❾
  echo "<table border=\"1\">";
  echo "<tr><th>Code article</th> <th>Description</th> <th>Prix</th>
  ➥<th>Cat&#233;gorie</th></tr>";
  while($ligne=$result->fetch(PDO::FETCH_NUM)) ←❿
  {
    echo "<tr>";
    foreach($ligne as $valeur) ←⓫
    {
      echo "<td> $valeur </td>";
    }
    echo "</tr>";
  }
  echo "</table>";
  }
  $result->closeCursor(); ←⓬
  $idcom=null; ←⓭
}
?>
```

5- Récupération des valeurs dans un objet

Les objets de type PDOStatement possèdent le méthode fetchObject(), dont la syntaxe est :

object $result–>fetchObject()

À chaque appel de cette méthode, l'objet retourné représente une ligne de résultat et possède autant de propriétés qu'il existe d'attributs dans la requête SQL SELECT ; les noms de celles ci sont d'ailleurs ceux des colonnes de la table ou des alias éventuels.

Si nous récupérons une ligne de résultat dans la variable :

$ligne=$result–>fetchObject()

La valeur d'un attribut nom est lue avec la syntaxe :

$ligne–>nom

L'exemple ci-dessous réalise la recherche et l'affichage dans un tableau HTML de tous les clients qui habitent Paris en utilisant cette méthode. En revanche, pour afficher les titres du tableau, nous n'allons pas utiliser la méthode développée dans l'exemple précédent.

Dans la requête SQL, nous définissons des alias qui vont correspondre aux en-têtes. Pour les lire, nous appelons la méthode fetchObject() pour l'objet $result et récupérons la première ligne de résultat dans la variable $ligne. Cette variable est un objet de type StdClass (la classe de base de PHP 5) dont les propriétés ont pour nom ceux des colonnes ou des alias de la requête. Comme il s'agit d'un objet, nous pouvons le parcourir au moyen d'une boucle foreach pour écrire les en-têtes du tableau HTML. Les valeurs associées ne nous intéressent pas encore ici. Pour récupérer le tableau des données nous utilisons une boucle do...while, ce qui nous permet de ne pas perdre les valeurs de la première ligne par un deuxième appel de la méthode fetchObject().

Chaque nouvelle ligne est obtenue dans la condition de l'instruction while .

```
<?php
include("connexpdo.inc.php");
$idcom=connexpdo("magasin","myparam");
$requete="SELECT id_client AS 'Code_client',nom,prenom,adresse,age,mail
➥FROM client WHERE ville ='Paris' ORDER BY nom"; ←❶
$result=$idcom->query($requete);
if(!$result)
{
  $mes_erreur=$idcom->errorInfo();
  echo "Lecture impossible, code", $idcom->errorCode(),$mes_erreur[2];
}
else
{
  $nbart=$result->rowCount();

  $ligne=$result->fetchObject(); ←❷
  echo "<h3> Il y a $nbart clients habitant Paris</h3>";
  //Affichage des titres du tableau
  echo "<table border=\"1\"> <tr>";
  foreach($ligne as $nomcol=>$val) ←❸
  {
    echo "<th>", $nomcol ,"</th>";
  }
  echo "</tr>";
  //Affichage des valeurs du tableau
  echo "<tr>";
  //Il faut utiliser do while car sinon on perd la première ligne de données
  do
  {
    echo"<td>", $ligne->Code_client,"</td>", "<td>", $ligne->nom,"</td>","<td>",
    ➥$ligne->prenom,"</td>","<td>", $ligne->adresse,"</td>","<td>", $ligne->age,
    ➥"</td>","<td>", $ligne->mail,"</td></tr>"; ←❹
  }
  while ($ligne = $result->fetchObject()) ; ←❺
  echo "</table>";
  $result->closeCursor();
  $idcom=null;
}
?>
```

Par exemple, le code suivant permet d'afficher les résultats d'une requête SELECT :

while($ligne=$result–>fetch(PDO::FETCH_OBJ))

{

foreach($ligne as $donnee)

{

```
    echo $donnee;
  }
  echo "<br />";
}
```

6- Insertion de données dans la base

Sur un site interactif, il faut pouvoir enregistrer dans la base de données les informations saisies par les visiteurs dans un formulaire XHTML, en vue d'une réutilisation ultérieure (par exemple, pour consulter ou réutiliser les coordonnées complètes d'un client).

Comme dans les chapitres 15 et 16, placez-vous dans la perspective d'un site de ecommerce.

Vous allez ainsi envisager la réalisation de la saisie puis de l'insertion des coordonnées d'un client dans la table client de la base magasin. Dans un second temps, vous lui permettrez de mettre à jour les informations enregistrées.

7- Insertion des données

Le formulaire XHTML est l'outil privilégié pour saisir des données et les envoyer vers le serveur PHP/MySQL. Nous disposons désormais de la fonction connexpdo() pour effectuer la connexion et des méthodes exec() et query() pour l'envoi des requêtes. Seule la commande SQL INSERT

distingue cette opération de celle de lecture de données. Le script de l'exemple ci-dessous réalise ce type d'insertion en récupérant les données saisies par le client dans un formulaire lors d'une commande. Nous commençons par vérifier l'existence des saisies obligatoires correspondant aux variables $_POST['nom'], $_POST['adresse'] et $_POST['ville']. Quand une requête est formée en utilisant les saisies faites par l'utilisateur, il est préférable d'utiliser le caractère d'échappement pour les caractères spéciaux des chaînes récupérées dans le tableau $_POST, en particulier les guillemets, qui peuvent poser problème dans la requête. Nous disposons pour cela de la méthode quote() des objets PDO dont la syntaxe est :

string $idcom–>quote(string $chaine)

La chaîne obtenue contient les caractères d'échappement "/" devant les caractères spéciaux NULL, \n, \r, ', " et Control-Z.

Le script récupère toutes les saisies et les protège. La colonne id_client de la table client ayant été déclarée avec l'option AUTO_INCREMENT, il faut y insérer la valeur NULL en lui donnant la valeur "\N". L'envoi de la requête se faisant avec la méthode exec(), la valeur retournée est le nombre de lignes insérées (ici « 1 »), nous pouvons contrôler la bonne fin de l'insertion. Le script doit

```
<?php
include("connexpdo.inc.php");
$idcom=connexpdo('magasin','myparam');
if(!empty($_POST['nom'])&& !empty($_POST['adresse'])&& !empty($_POST['ville'])) ←❶
{
  $id_client="\N"; ←❷
  $nom=$idcom->quote($_POST['nom']); ←❸
  $prenom=$idcom->quote($_POST['prenom']); ←❹
  $age=$idcom->quote($_POST['age']); ←❺
  $adresse=$idcom->quote($_POST['adresse']); ←❻
  $ville=$idcom->quote($_POST['ville']); ←❼
```

```
    $mail=$idcom->quote($_POST['mail']); ←8
    //Requête SQL
    $requete="INSERT INTO client
    VALUES($id_client,$nom,$prenom,$age,$adresse,$ville,$mail)";
    //pas de guillemets si on applique la méthode quote aux variables
    $nblignes=$idcom->exec($requete); ←9
    if($nblignes!=1) ←10
    {
      $mess_erreur=$idcom->errorInfo();
      echo "Insertion impossible, code", $idcom->errorCode(), $mess_erreur[2];
      echo "<script type=\"text/javascript\">
      alert('Erreur : ".$idcom->errorCode()."')</script>";
    }
    else
    {
      echo "<script type=\"text/javascript\">
      alert('Vous êtes enregistré Votre numéro de client est : ".
      $idcom->lastInsertId()."')</script>"; ←11
    $idcom=null;
    }
  }
else {echo "<h3>Formulaire à compléter!</h3>";}
?>
```

8- Les transactions

Dans l'exemple de notre base magasin, si un client effectue un achat, il faut réaliser simultanément deux insertions, une dans la table commande et une dans la table ligne. Si pour une raison quelconque, matérielle ou logicielle, la seconde insertion n'est pas réalisée, alors que la première l'est déjà, la base contiendra une incohérence car il existera une commande ne comportant aucune ligne. L'inverse ne serait guère préférable puisqu'il existerait une ligne qui ne serait reliée à aucune commande. Les transactions permettent de gérer ce genre de situation en effectuant des commandes « tout ou rien » ce qui, en pratique pour notre exemple ci-dessus, permet d'exécuter les deux requêtes ou bien aucune si l'une des deux insertion n'a pas été effectuée.

L'intégrité de la base est ainsi préservée.

Le langage SQL possède des commandes qui permettent de gérer les transactions, mais PDO nous fournit des méthodes qui permettent de gérer les transactions sans y faire appel.

Une transaction doit commencer par l'appel de la méthode beginTransaction() de l'objet PDO qui désactive le mode autocommit qui est automatiquement activé par défaut dans MySQL. L'envoi des requêtes au serveur se fait comme d'habitude avec les méthodes query() ou exec(), selon qu'elles retournent des résultats ou pas. Ce n'est qu'après avoir vérifié que les différentes requêtes ont été correctement effectuées que nous pouvons valider l'ensemble en appelant la méthode commit(). Dans le cas contraire, par exemple si l'une d'entre elles n'a pas été réalisée, nous annulons l'ensemble en appelant la méthode rollBack().

Dans l'exemple ci-dessous, nous illustrons la procédure à suivre pour effectuer deux commandes INSERT sur la table article. Nous envoyons ensuite ces requêtes au serveur avec la méthode exec() qui retourne le nombre de lignes affectées par chaque requête. La variable $verif cumule donc le nombre total d'insertions. Si ce nombre vaut 2, nous pouvons valider la transaction avec la méthode commit(), sinon elle est annulée avec la méthode rollBack(), un message d'erreur est affiché et le tableau indicé retourné par la méthode errorInfo() contenant le code d'erreur pour l'indice 0 et le message d'erreur en clair pour l'indice 2. Pour tester l'efficacité du script, il suffit d'introduire une erreur dans la seconde requête en écrivant, par exemple, le nom de table inexistant "clients" à la place de "client", pour constater avec phpMyAdmin qu'aucune insertion n'est effectuée alors que la première est valable.

```php
<?php
include('connexpdo.inc.php');
$idcon=connexpdo('magasin','myparam');
$idcon->beginTransaction();
echo "<hr />";
$requete1="INSERT INTO client(id_client,nom,prenom,age,adresse,ville,mail)
=>VALUES (NULL , 'Spencer', 'Marc', '32', 'rue du blues', 'Orléans',
=>'marc@spencer.be');"; ←1
echo $requete1,"<hr />";
$requete2="INSERT INTO client(id_client,nom,prenom,age,adresse,ville,mail)
=>VALUES (NULL , 'Spencer', 'Diss', '89', 'rue du Métad', 'New York',
=>'diss@metad.fr');"; ←2
echo $requete2,"<hr />";

//Insertions des données
$verif= $idcon->exec($requete1); ←3
$verif+= $idcon->exec($requete2); ←4
if($verif==2 ) ←5
{
    $idcon->commit(); ←6
    echo "Insertions réussies de $verif  lignes<br />";
}
else
{
    $idcon->rollBack(); ←7
    $tab_erreur=$idcon->errorInfo();
    echo "Insertions annulées. Erreur n° :",$tab_erreur[0],"<br />"; ←8
    echo "Info : ",$tab_erreur[2]; ←9
}
?>
```

Chapitre 5 PHP et XML

1 Introduction

L'extension SimpleXML fut une des nouveautés importantes de PHP 5. Cette nouveauté n'est pas vraiment révolutionnaire puisque l'extension DOMXML permettait déjà d'accéder au contenu d'un fichier XML à partir d'un script PHP.

À la différence de DOMXML, SimpleXML apporte, comme son nom l'indique, un accès facile à toutes sortes de documents XML, même s'ils ont une structure complexe. Elle est encore dotée de peu de fonctions, mais elle ne tardera pas à s'enrichir au fur et à mesure des besoins manifestés par ses utilisateurs et la version 3 a déjà accru le nombre de commandes SQL disponibles.

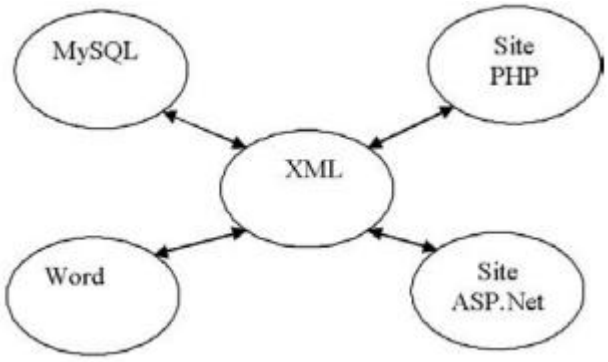
Les fonctions de SimpleXML retournent des objets qui possèdent des méthodes intéressantes d'analyse des fichiers.

Les fonctions et méthodes disponibles permettent de lire et de modifier le contenu des éléments ou des attributs d'un fichier XML, de sauvegarder ces modifications et d'effectuer des recherches sur les contenus. Cela représente déjà un large éventail d'activités.

2- Notions de XML

Le langage XML (eXtensible Markup Language) est un vecteur privilégié d'échange de données entre applications différentes. Comme l'illustre la figure 19-1, XML peut être le point commun d'applications qui ne pourraient pas communiquer entre elles sans son intermédiaire.

À l'instar du XHTML, XML est un langage de structuration de contenu au moyen de balises.



Pour délimiter des éléments ayant un contenu explicite, XML utilise les structures générales suivantes :

<balise> contenu de l'élément </balise>

Pour les éléments n'ayant pas de contenu explicite (éléments vides), dans lesquels l'information est donnée uniquement dans un ou plusieurs attributs, il utilise les balises suivantes :

<balise attribut = "valeur"/>

En XHTML, il existe 70 éléments prédéterminés. Les navigateurs implémentent ces éléments et associent à chacun d'eux un type de présentation de leur contenu.

Par exemple, les lignes suivantes :

<h1> PHP5 MySQL </h1>

écrivent dans un navigateur un gros titre suivi d'un saut de ligne, mais cette présentation peut être modifiée avec l'emploi de feuilles de styles CSS.

L'objectif de XML est différent. Il consiste à décrire un grand nombre de types d'informations différents,

allant de la description des coordonnées d'un client jusqu'à celle d'une base de données complète, ce que ne permet pas le XHTML.

L'idée essentielle à l'origine de XML est de structurer l'information en séparant le contenu de sa présentation, que ce soit dans un navigateur ou dans des médias les plus divers. Le contenu est écrit dans un fichier XML, et la présentation dans une feuille de style CSS ou XSLT.

Pour écrire un document XML, vous n'êtes plus limité par un nombre d'éléments fixe.

Le créateur d'un document peut même choisir les noms des éléments qu'il souhaite utiliser pour structurer ses informations.

L'écriture d'un document XML doit obéir à des règles syntaxiques strictes afin d'être bien formé et lisible par un parseur XML, celui d'un navigateur pour ce qui nous concerne.

Les principales règles syntaxiques de XML sont les suivantes :

• Chaque document commence par l'élément suivant :

<? xml version = "1.0" encoding="IS0-8859-1" standalone = "yes">

dans lequel l'attribut version définit la version de XML utilisée. La dernière version publiée par le W3C est la 1.1, mais elle n'est actuellement pas bien reconnue par tous les navigateurs. L'attribut encoding contient le jeu de caractères utilisé dans le document.

L'attribut ***standalone*** indique si le document est indépendant (valeur "yes") ou s'il doit faire appel à un autre document externe, comme une DTD, pour pouvoir être utilisé (valeur "no").

• Chaque document doit avoir un élément racine qui englobe tous les autres. C'est l'équivalent de l'élément <html> </html> dans un document XHTML.

• Les noms des éléments sont écrits le plus souvent en minuscules. Les majuscules sont cependant autorisées, à condition que la balise d'ouverture ait la même casse que celle de fermeture.

• Chaque élément ayant un contenu doit avoir une balise d'ouverture et une balise de fermeture, comme en XHTML selon la forme :

<element> contenu </element>

• Les éléments n'ayant pas de contenu peuvent avoir la forme suivante :

<element />

• Chaque élément peut en contenir d'autres, sans limite de nombre. Les éléments qui en contiennent d'autres doivent être fermés après tous ceux qu'ils contiennent. On dit qu'ils doivent être correctement emboîtés. Leur structure doit respecter la forme générale suivante :

<element>

<souselement>

contenu du sous élément

</souselement>

</element>

Vous obtenez une hiérarchie père-fils.

• Tous les éléments peuvent avoir des attributs contenus dans la balise d'ouverture. Les valeurs de ces attributs doivent être écrites entre guillemets :

<element attribut="valeur">contenu </element>

• Les caractères spéciaux suivants présents dans le contenu d'un élément doivent être remplacés par des entités prédéfinies :

'<' par '<'

'>' par '>'

'&' par '&'

" par '"'

' par '''

La structure type d'un document XML est la suivante :

<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>

<racine >

<element1 attribut="valeur">

<souselementA attribut="valeur">contenu A</souselementA>

<souselementB attribut="valeur">contenu B</souselementB>

</element1>

<element2 attribut="valeur">

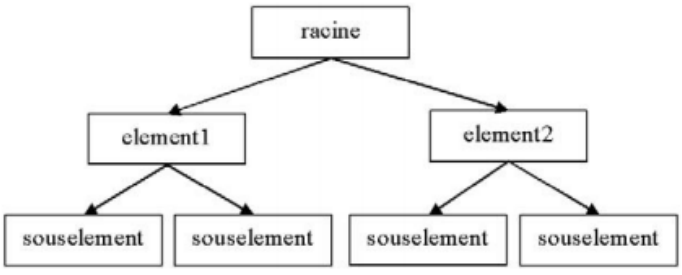
<souselementC attribut="valeur">contenu C</souselementC>

<souselementD attribut="valeur">contenu D</souselementD>

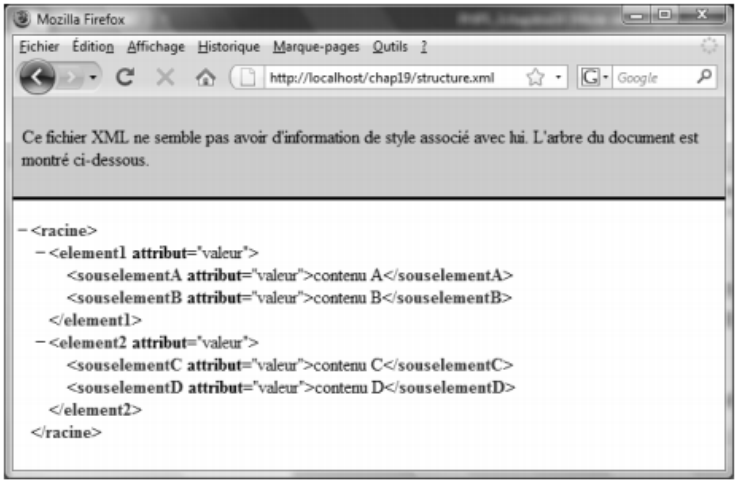
</element2>

</racine>

En conséquence de la règle d'emboîtement des éléments, un document a une structure arborescente, comme illustré à la figure ci-dessous.



Si vous ouvrez ce fichier dans un navigateur Firefox, vous obtenez un affichage sans grand intérêt pratique, comme le montre la figure ci-dessous.



Dans le navigateur Amaya créé par le W3C, vous obtenez l'affichage du contenu des éléments sans les balises qui les délimitent ni leurs attributs, comme l'illustre la figure ci-dessous.

Pour exploiter le contenu d'un fichier XML dans une page Web, il vous faut envisager les moyens de lire un fichier XML et de récupérer le contenu de ses éléments dans des variables utilisables par des scripts PHP.

3- Lecture d'un fichier XML

L'extension SimpleXML fournit des fonctions qui permettent un accès simple et rapide au contenu d'un fichier XML à l'aide d'objets de type simplexml_element. Ces objets comportent des propriétés et des méthodes qui permettent d'accéder au contenu des éléments, de le modifier ou d'effectuer des recherches dans le fichier.

4- Accéder au contenu d'un fichier XML

Pour accéder au contenu d'un fichier XML, vous disposez de la fonction simplexml_load_file()

. Cette fonction transfère le contenu du fichier dans un objet de type simplexml_element contenant l'arborescence du fichier. Les propriétés de cet objet prennent pour noms ceux de chacun des éléments XML et pour valeurs les contenus des éléments du fichier.

La syntaxe de simplexml_element est la suivante :

object simplexml_load_file (string nom_fichier)

Si l'ensemble du code XML est contenu dans une chaîne de caractères $code, vous pouvez utiliser la fonction suivante à la place de la précédente :

object simplexml_load_string (string $code)

Cette fonction retourne le même type d'objet.

Un script de lecture de fichier XML commence donc par le code suivant :

$xml = simplexml_load_file ("nom_fichier.xml");

Il est suivi de la lecture des données du fichier XML au moyen des propriétés et des méthodes de l'objet $xml de type simplexml_element ainsi obtenu.

Pour lire le contenu d'un élément nommé, par exemple, <livre> vous écrivez :

echo $xml–>livre;

Le fichier XML biblio1.xml suivant contient une bibliographie dont l'élément racine est <biblio> et dont les éléments <titre>, <auteur> et <date> contiennent les caractéristiques d'un seul livre.

Le fichier *biblio1.xml*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<biblio>
  <titre>L'empire de la honte</titre>
  <auteur>Jean Ziegler</auteur>
  <date>2005</date>
</biblio>
```

Le script de l'exemple 19-1 permet de lire le contenu du fichier biblio1.xml. Après le chargement du contenu du fichier dans l'objet $xml, le script affiche le contenu de chacun des éléments en utilisant les propriétés titre, auteur et date de l'objet $xml.

Exemple 19-1. Lecture des éléments

```
<?php
$xml=simplexml_load_file("biblio1.xml"); ←❶
echo "Titre :",$xml->titre,"<br />" ←❷;
echo "Auteur :",$xml->auteur,"<br />" ←❸;
echo "Date :",$xml->date,"<br />" ←❹;
?>
```

L'exemple retourne le résultat suivant :

```
Titre : L'empire de la honte
Auteur : Jean Ziegler
Date :2005
```

Ce premier fichier biblio1.xml est très simple et n'a pour but que d'introduire la méthode de lecture des éléments. Vous allez maintenant aborder la méthode de lecture d'un fichier XML plus proche d'un fichier réel.

Le fichier biblio2.xml contient encore une bibliographie, mais il est maintenant capable de contenir un nombre quelconque de livres. Pour cela, sa structure a été modifiée.

L'élément racine <biblio> contient autant d'éléments <livre> que nécessaire, chaque livre étant encore caractérisé par les sous-éléments <titre>, <auteur> et <date>.

Le fichier *biblio2.xml*

```
<?xml version="1.0" encoding="iso-8859-1"?>
<biblio>
  <livre>
   <titre>L'empire de la honte</titre>
  <auteur>Jean Ziegler</auteur>
  <date>2005</date>
  </livre>
  <livre>
    <titre>Ritournelle de la faim </titre>
    <auteur>J.M.G Le Clézio</auteur>
    <date>2008</date>
  </livre>
  <livre>
    <titre>Singue Sabour : La pierre de patience</titre>
    <auteur>Atiq Rahimi</auteur>
    <date>2008</date>
  </livre>
</biblio>
```

Pour accéder au contenu d'un élément <livre>, vous devez utiliser une syntaxe proche de celle des tableaux. La variable $xml–>livre[0] est maintenant un objet de type simplexml_ element . Cet objet représente le premier livre du fichier et possède autant de propriétés que l'élément <livre> a de sous-éléments.

Vous accédez aux informations utiles en écrivant le code suivant :

```
echo $xml->livre[0]->titre;
echo $xml->livre[0]->auteur;
echo $xml->livre[0]->date;
```

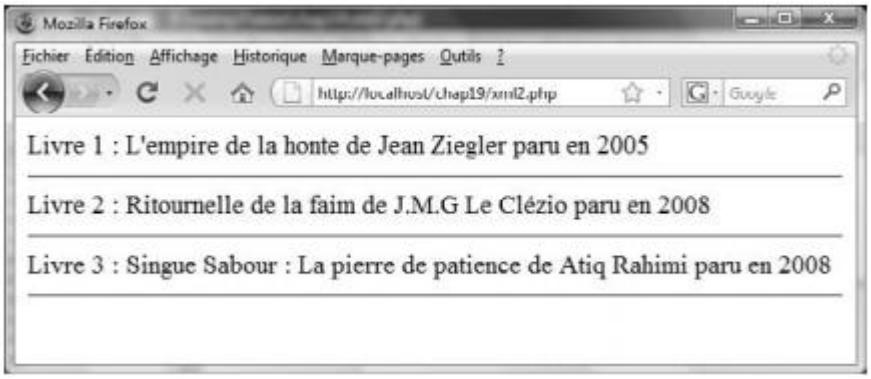
Ce code affiche le titre, l'auteur et la date de publication du premier livre. Pour lire l'ensemble des livres, il est préférable d'effectuer une boucle.

Le script de l'exemple ci-dessous commence par charger le fichier biblio2.xml. Il lit ensuite l'ensemble du fichier avec le minimum de code en effectuant une boucle foreach sur l'objet $xml–>livre. La variable $cle contient la même valeur livre à chaque itération. La variable $val est aussi un objet de type simplexml_element, dont les propriétés sont titre, auteur et date. Vous y accédez avec la même syntaxe qu'à l'exemple précédent. L'utilisation du mot-clé static pour la variable compteur $i et son incrémentation permet d'afficher le numéro de chaque livre.

Exemple 19-2. Lecture d'un ensemble d'éléments

```
<?php
$xml=simplexml_load_file("biblio2.xml"); ←❶
//Lecture du contenu des éléments
foreach($xml->livre as $cle=>$val) ←❷
{
  static $i=1; ←❸
  echo ucfirst($cle)," $i : $val->titre de $val->auteur paru en
  ➥$val->date<hr />"; ←❹
  $i++; ←❺
}
?>
```

On obtient le résultat suivant :



5- Lecture des attributs d'un élément

Chaque élément du document XML peut avoir des attributs. Les attributs constituent un complément d'information inclus dans un élément. Leur définition ressort d'un choix du programmeur, car la même information peut être enregistrée dans un sous-élément. Vous verrez que la transformation d'une base de données en fichier XML par phpMyAdmin ne crée aucun attribut et crée uniquement des éléments. Si ce choix de créer des éléments et des attributs est fait, il vous faut lire la valeur des différents attributs d'un élément.

Le fichier XML biblio3.xml comporte les mêmes éléments que le fichier biblio2.xml, mais chaque élément <livre> a désormais deux attributs, qui précisent l'éditeur et le prix de chaque livre.

Le fichier *biblio3.xml*

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
<biblio>
  <livre editeur="FAYARD" prix="20.00">
    <titre>L'empire de la honte</titre>
    <auteur>Jean Ziegler</auteur>
    <date>2005</date>
  </livre>
  <livre editeur="GALLIMARD" prix="18.00">
    <titre>Ritournelle de la faim </titre>
    <auteur>J.M.G Le Clézio</auteur>
    <date>2008</date>
  </livre>
  <livre editeur="POL" prix="15.00">
    <titre>Singue Sabour : La pierre de patience</titre>
    <auteur>Atiq Rahimi</auteur>
    <date>2008</date>
  </livre>
</biblio>
```