Jean Engels

2e édition
PHP 5.2 et 5.3

PHP5

Cours et exercices

Cours et exercices

EYROLLES

CHEZ LE MÊME ÉDITEUR

Du même auteur

J. ENGELS. – **XHTML et CSS.** *Cours et exercices*.
N°11637, 2006, 508 pages.

Autres ouvrages

E. DASPET, C. PIERRE de GEYER. – **PHP 5 avancé.**
N°12369, 5e édition, 2008, 844 pages.

G. PONÇON, J. PAULI. – **Zend Framework.**
N°12392, 2008, 460 pages.

J.-M. DEFRANCE. – **Premières applications Web 2.0 avec Ajax et PHP.**
N°12090, 2008, 450 pages.

D. SÉGUY, P. GAMACHE. – **Sécurité PHP 5 et MySQL.**
N°12114, 2007, 240 pages.

G. PONÇON. – **Best practices PHP 5.** Les meilleures pratiques de développement en PHP.
N°11676, 2005, 480 pages.

C. PIERRE DE GEYER et G. PONÇON. – **Mémento PHP et SQL.**
N°12457, 2e édition, 2009, 14 pages.

C. PORTENEUVE. – **Bien développer pour le Web 2.0.**
Bonnes pratiques Ajax - Prototype, Script.aculo.us, accessibilité, JavaScript, DOM, XHTML/CSS.
N°12391, 2008, 674 pages.

R. GOETTER. – **CSS 2 : pratique du design web.**
N°12461, 3e édition, 2009, 340 pages.

V. ISAKSEN, T. TARDIF. – **Joomla et VirtueMart.** *Réussir sa boutique en ligne.*
N°12381, 2008, 306 pages.

A. VANNIEUWENHUYZE. – **Flex 3.**
N°12387, 2009, 532 pages.

T. AUDOUX, J.-M. DEFRANCE. – **Dreamweaver CS3.**
N°12234, 2008, 602 pages.

A. BOUCHER. – **Ergonomie web.**
N°12479, 2e édition, 2009, 456 pages.

N. CHU. – **Réussir un projet de site web.**
N°12400, 5e édition, 2008, 246 pages.

Jean Engels

PHP5

Cours et exercices

2e édition - PHP 5.2 et 5.3

EYROLLES

Il est recommandé d'imprimer ces informations et de les conserver précieusement car elles vous permettront de déterminer, au moment où vous en aurez besoin, si vous pouvez utiliser tel ou tel module ou fonction. Il serait dommage de travailler des heures à créer un script qui utilise des fonctions utilisables en local mais non disponibles sur votre serveur distant.

Structure des fichiers XHTML

Comme expliqué précédemment, la connaissance du langage XHTML est utile pour se lancer dans l'écriture de scripts PHP. Il est donc utile de connaître la structure des fichiers XHTML car une page dynamique PHP est bien un document XHTML envoyé par le serveur vers le poste client.

Pour être conforme aux recommandations XHTML du W3C *(http:// www.w3.org)*, un document XHTML doit avoir la structure suivante (fichier `pagexhtml.html`) :

```
<!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-8859-1" />
<title>Titre de la page</title>
</head>
<body>
<h2>Bienvenue sur le site PHP 5</h2>
</body>
</html>
```


Cette page primaire est écrite en XHTML pur, et tous les visiteurs de votre site verront exactement le même contenu, quel que soit le moment de leur connexion. Le fichier peut avoir l'extension `.html` ou `.htm` car il ne contient que du code XHTML, mais il pourrait tout aussi bien avoir une extension `.php` et avoir le même rendu dans un navigateur.

Vous pourriez lui apporter un brin de dynamisme en affichant la date du jour en tête de page à l'aide du code PHP suivant (fichier `codephp.php`) :

```
<!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-8859-1" />
  <title>Une page PHP</title>
 </head>
 <body>
  <?php
   echo "<h3> Aujourd'hui le ". date('d / M / Y H:m:s')."</h3><hr />";
   echo "<h2>Bienvenue sur le site PHP 5</h2>";
  ?>
 </body>
```

Examinez maintenant le code source du document tel qu'il a été reçu par le navigateur. Dans Firefox, par exemple, allez dans le menu Affichage>Source de la page. Le code suivant s'affiche :

```
<!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-8859-1" />
  <title>Une page PHP</title>
 </head>
 <body>
  <h3> Aujourd'hui le 24 / Sep / 2008 11:09:26</h3><hr />
  <h2>Bienvenue sur le site PHP 5</h2>
 </body>
</html>
```

Par rapport au code du fichier `codephp.php`, ce qui était contenu entre les éléments `<?php` et `?>`, soit :

```
<?php
echo "<h3> Aujourd'hui le ". date('d / M / Y H:m:s ') ."</h3><hr />";
echo "<h2>Bienvenue sur le site PHP 5</h2>";
?>
```

a été remplacé par :

```
<h3> Aujourd'hui le 24 / Sep / 2008 11:09:26</h3><hr />
<h2>Bienvenue sur le site PHP 5</h2>
```


L'interpréteur PHP analyse le document dans son ensemble puis renvoie le code XHTML tel quel, accompagné de l'évaluation des expressions contenues dans le code PHP. Cela fait d'ailleurs dire à certains que tout est expression dans PHP puisque tout le code peut être évalué comme une chaîne de caractères, un nombre ou une valeur booléenne.

Les parties de code contenues dans les guillemets sont renvoyées dans le flux du document XHTML, et les balises qu'elles contiennent sont interprétées en tant que telles par le navigateur. C'est le cas de la deuxième ligne. La première ligne comporte une fonction PHP qui retourne la date du jour. Cette date est concaténée avec le texte qui l'entoure puis est retournée au navigateur.

Le cycle de vie d'une page PHP est le suivant :

- Envoi d'une requête HTTP par le navigateur client vers le serveur, du type *http://www.monserveur.com/codephp.php*.
- Interprétation par le serveur du code PHP contenu dans la page appelée.
- Envoi par le serveur d'un fichier dont le contenu est purement XHTML.

Vous constatez ainsi que votre code PHP n'est jamais visible par les visiteurs de votre site.

Écriture du code PHP

Le code PHP est toujours incorporé dans du code XHTML. Vous pouvez donc incorporer autant de scripts PHP indépendants que vous le souhaitez n'importe où dans du code XHTML, du moment que ces parties sont délimitées par les balises ouvrantes et fermantes `<?php` et `"?>"` (repères ❶, ❷, ❹, ❻ et ❼) ou par la forme courte `<?=` et `?>` (repères ❺ et ❽) ou encore par l'élément XHTML `<script language="php">` (repère ❸), qui est rarement employé.

Dans un fichier `.php`, vous pouvez à tout moment passer du code PHP au code XHTML, et réciproquement. C'est ce qui donne sa grande souplesse d'utilisation à ce code.

Le listing suivant illustre cette particularité :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<?php ←❶
  $variable1=" PHP 5";
?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<?php ←❷
echo "<title>Une page pleine de scripts PHP</title>";
?>
</head>
<body>
<script language="php"> ←❸
  echo "<h1>BONJOUR A TOUS </h1>";
</script>
<?php ←❹
  echo "<h2> Titre écrit par PHP</h2>";
  $variable2=" MySQL";
?>
<p>Vous allez découvrir <?= $variable1 ?> ←❺</p>
<?php ←❻
  echo "<h2> Bonjour de $variable1</h2>";
?>
<p>Utilisation de variables PHP<br />Vous allez découvrir également
  <?php ←❼
  echo $variable2
  ?>
</p>
<?= "<div><big>Bonjour de $variable2 </big></div>" ?> ←❽
</body>
</html>
```


Huit mini-scripts PHP sont placés aussi bien dans l'en-tête (entre `<head>` et `</head>`) que dans le corps (entre `<body>` et `</body>`) ou encore même en dehors du bloc délimité par les éléments `<html>` et `</html>` du document XHTML.

remarquez l'utilisation du même nom de variable alors que les valeurs affectées sont de type différent.

Dans l'affectation par valeur à l'aide de l'opérateur =, l'opérande de gauche, c'est-à-dire la variable à affecter, prend la valeur de l'expression contenue dans l'opérande de droite, et voilà tout. Toute modification ultérieure de l'opérande de droite, même s'il est lui-même une variable, n'a aucune incidence sur la variable affectée.

Dans l'exemple suivant :

```
$mavar1="Paris";
$mavar2="Lyon";
$mavar2=$mavar1;
$mavar1="Nantes";
```

à la fin du code, la variable `$mavar2` contient la chaîne `"Paris"`, puisque vous lui avez affecté la valeur de l'expression `$mavar1`, et `$mavar1` vaut `"Nantes"`, puisque sa valeur a été modifiée à la fin du script.

Avec l'affectation par référence, toujours réalisée au moyen de l'opérateur =, l'opérande de droite est une variable qui doit être précédée du caractère `&` (esperluette).

Dans l'exemple suivant :

```
$mavar1="Paris";
$mavar2="Lyon";
$mavar2 = &$mavar1;
$mavar1="Nantes";
```

la variable `$mavar2` devient un alias de la variable `$mavar1`, et les modifications opérées sur `$mavar1` sont répercutées sur `$mavar2`. Plus déroutant encore pour le novice, et plus dangereux aussi, toute modification apportée à la valeur de `$mavar2` est répercutée dans `$mavar1` puisque `$mavar2` est un alias de `$mavar1`. C'est ce qu'illustre le script de l'exemple 2-1.

Exemple 2-1. Affectation par valeur et par référence

```
<?php
//Affectation par valeur de $mavar1 et $mavar2
$mavar1="Paris";
echo "\$mavar1= ",$mavar1,"<br />";
$mavar2="Lyon";
echo "\$mavar2= ",$mavar2,"<br />";
//Affectation par référence de $mavar2
$mavar2 = &$mavar1;
echo "Affectation par référence de \$mavar2 <br />";
echo "\$mavar1= ",$mavar1,"<br />";
echo "\$mavar2= ",$mavar2,"<br />";
echo "modification  de \$mavar1 <br />";
$mavar1="Nantes";
echo "\$mavar1= ",$mavar1,"<br />";
echo "\$mavar2= ",$mavar2,"<br />";
echo "modification  de \$mavar2 <br />";
$mavar2="Marseille";
```

Tableau 2-2 – Les opérateurs d'affectation combinée *(suite)*

`/=`	Division puis affectation : `$x /= $y` équivaut à `$x = $x / $y` `$y` peut être une expression complexe dont la valeur est un nombre différent de 0.
`%=`	Modulo puis affectation : `$x %= $y` équivaut à `$x = $x % $y` `$y` peut être une expression complexe dont la valeur est un nombre.
`.=`	Concaténation puis affectation : `$x .= $y` équivaut à `$x = $x . $y` `$y` peut être une expression littérale dont la valeur est une chaîne de caractères.

Les constantes

Vous serez parfois amené à utiliser de manière répétitive des informations devant rester constantes dans toutes les pages d'un même site. Il peut s'agir de texte ou de nombres qui reviennent souvent. Pour ne pas risquer l'écrasement accidentel de ces valeurs, qui pourrait se produire si elles étaient contenues dans des variables, vous avez tout intérêt à les enregistrer sous forme de constantes personnalisées.

PHP dispose d'un ensemble de constantes prédéfinies utilisables dans tous les scripts.

Définir ses constantes personnalisées

Pour définir des constantes personnalisées, utilisez la fonction `define()`, dont la syntaxe est la suivante :

```
boolean define(string nom_cte, divers valeur_cte, boolean casse)
```

Dans cet exemple, vous attribuez la valeur `valeur_cte` à la constante nommée `nom_cte`, dont le nom doit être contenu dans une chaîne de caractères délimitée par des guillemets. Le paramètre `casse` vaut `TRUE` si le nom de la constante est insensible à la casse et `FALSE` sinon. La fonction `define()` retourne `TRUE` si la constante a bien été définie et `FALSE` en cas de problème, par exemple, si vous essayez de redéfinir une constante existante, ce qui est interdit. Toute tentative de modifier la valeur d'une constante en la redéfinissant provoque un avertissement (warning) de la part du serveur.

> **Attention**
>
> Une constante n'étant pas précédée du signe dollar (`$`), vous ne pouvez l'incorporer telle quelle dans une chaîne comme vous le faites avec les variables. Il vous faut donc la concaténer avec une chaîne ou la séparer de ce qui précède par une virgule dans l'instruction `echo`.

La fonction `defined(string nom_cte)` permet de vérifier si une constante nommée existe. Elle retourne `TRUE` si la constante nommée `nom_cte` existe et `FALSE` sinon. Cette vérification peut être utile, car il est impossible de déclarer deux constantes de même nom.

soit la valeur $3 \times 8^3 + 2 \times 8^2 + 6 \times 8^1 + 7 \times 8^0$, donc 1719 en décimal.

Notez que PHP n'affiche pas directement les valeurs en octal et que l'utilisation de l'instruction `echo $varoct` n'affiche donc pas 03267 mais la valeur décimale 1719.

Les sections suivantes donnent les différentes fonctions de conversion entre les bases de numération.

Les nombres en base 16 commencent par les caractères `0x`, ou `0X`, au choix, suivis de un ou plusieurs chiffres (de 0 à 9) ou des lettres A (pour 10) à F (pour 15) :

```
$varhex = 0xFAC7;
echo $varhex;//Affiche 64199
```

soit, en décimal, la valeur :

$15 \times 16^3 + 10 \times 16^2 + 12 \times 16^1 + 7 \times 16^0 = 64199$.

Là encore, la deuxième ligne de code n'affiche pas la valeur hexadécimale. Pour afficher la valeur hexadécimale, utilisez les fonctions de conversion mathématiques, présentées ultérieurement dans ce chapitre.

Les flottants

Le type `double` est censé représenter les nombres réels. En fait, une représentation exacte des réels est impossible à réaliser dans la plupart des cas avec un nombre de bits limités, ici 32 bits.

En toute rigueur, le type `double` représente l'ensemble des nombres décimaux avec une précision de 14 chiffres, ce qui est suffisant dans la plupart des cas. Ce n'est pas un détail si vous voulez procéder à des calculs précis, scientifiques par exemple.

Pour effectuer des calculs plus précis qu'avec des nombres de type `double`, vous pouvez utiliser la bibliothèque BCMath. Présente par défaut dans Wampserver et sur de nombreux serveurs susceptibles d'héberger votre site, elle fournit un éventail de fonctions de calcul avec une précision choisie à l'avance (pour plus de détails voir la page *http://fr2.php.net/manual/fr/book.bc.php*).

PHP admet pour les nombres flottants la notation décimale classique, avec le point comme séparateur, et la notation exponentielle, dite scientifique, avec le symbole e ou E.

Vous pouvez donc avoir les notations suivantes :

```
<?php
$vardbl = 1952.36;
$vardbl2= 1.95236E3;//Soit 1.95236 x 1000
echo $vardbl2,"<br />";//Affiche 1952.36
$vardbl3= 1.95236e3;
echo $vardbl3,"<br />";//Affiche 1952.36
echo $vardbl3*100000000000,"<br />";//Affiche 1.95236E14
?>
```

L'affichage se fait sous forme décimale tant que le nombre a moins de 15 chiffres. Au-delà, il est fait sous forme exponentielle.

Les opérateurs numériques

PHP offre un large éventail d'opérateurs utilisables avec des nombres. Les variables ou les nombres sur lesquels agissent ces opérateurs sont appelés les opérandes.

Le tableau 2-4 donne la description de ces opérateurs, dont la plupart vous sont certainement familiers.

Tableau 2-4 – Les opérateurs numériques

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division

%	Modulo : reste de la division du premier opérande par le deuxième. Fonctionne aussi avec des opérandes décimaux. Dans ce cas, PHP ne tient compte que des parties entières de chacun des opérandes.
	`$var = 159;`
	`echo $var%7; //affiche 5 car 159=22x7 + 5`
	`$var = 11.5;`
	`echo $var%3.5;  //affiche 1 car 3 rentre 3 fois dans 11.`

--	Décrémentation : soustrait une unité à la variable. Il existe deux possibilités, la prédécrémentation, qui soustrait avant d'utiliser la variable, et la postdécrémentation, qui soustrait après avoir utilisé la variable.
	`$var=56;`
	`echo $var--; //affiche 56 puis décrémente $var.`
	`echo $var; //affiche 55.`
	`echo --$var; //décrémente $var puis affiche 54.`

++	Incrémentation : ajoute une unité à la variable. Il existe deux possibilités, la préincrémentation, qui ajoute 1 avant d'utiliser la variable, et la postincrémentation, qui ajoute 1 après avoir utilisé la variable.
	`$var=56;`
	`echo $var++; //affiche 56 puis incrémente $var.`
	`echo $var; //affiche 57.`
	`echo ++$var; //incrémente $var puis affiche 58.`

Les fonctions mathématiques

Le module de base de PHP offre un grand nombre de fonctions mathématiques utiles. Les noms des fonctions n'étant pas sensibles à la casse, vous pouvez écrire `abs()`, `Abs()` ou `ABS()` pour la fonction valeur absolue, par exemple.

Le tableau 2-5 récapitule les fonctions mathématiques offertes par PHP.

Les booléens

L'utilisation d'expressions booléennes est à la base de la création des instructions conditionnelles, qui permettent de gérer le déroulement d'un algorithme.

En plus de la définition du type `boolean`, il est important de connaître la manière dont PHP procède à l'évaluation des expressions dans un contexte booléen. Certaines évaluations ne sont pas du tout intuitives et peuvent donner des résultats inattendus au premier abord.

Le type boolean

Le type `boolean` est sûrement le plus simple puisqu'il ne peut contenir que deux valeurs différentes `TRUE` ou `FALSE`, correspondant aux valeurs vrai et faux qui peuvent être prises par une expression conditionnelle. Par exemple, `$a < 75` est évaluée à `TRUE` si `$a` vaut 74 et à `FALSE` si `$a` vaut 76.

L'exemple de code suivant :

```
<?php
$a=80;
$b= ($a<95);
echo "\$sb vaut ",$b,"<br />";
?>
```

affiche `$b vaut 1`.

La variable `$b` est de type `boolean` car elle est le résultat de l'expression `$a<95`. Sa valeur est `TRUE`. PHP assimile en interne la valeur `TRUE` à 1 et la valeur `FALSE` à 0, ce qui est un héritage de PHP 3, dans lequel le type `boolean` n'existait pas explicitement. C'est pour cette raison que l'affichage de `$b` est 1 au lieu de `TRUE` et une chaîne vide au lieu de 0 si `$b` vaut `FALSE`, ce qui peut être déconcertant.

Vous pouvez bien sûr affecter directement des variables avec des valeurs booléennes, comme ci-dessous :

```
$vart = TRUE;//ou encore $vart =true
$varf = FALSE;//ou encore $varf =false
```

Cette méthode n'est toutefois à utiliser que pour modifier explicitement la valeur d'une variable existante ou pour s'assurer de l'existence d'une valeur par défaut.

Nous manipulons généralement non pas des variables booléennes mais des expressions à valeur booléenne dans des instructions conditionnelles, comme `if($a<95)`, dans laquelle l'expression `$a<95` a une évaluation à `TRUE` ou à `FALSE` selon la valeur de `$a`.

L'expression peut être une fonction dont la valeur de retour est un booléen. PHP réalise une évaluation booléenne d'un certain nombre d'expressions qui ne comportent pas d'opérateurs de comparaison.

Tableau 2-8 – Les opérateurs logiques *(suite)*

XOR	Teste si un et un seul des opérandes a la valeur TRUE :
	`$a = true;`
	`$b = true;`
	`$c = false;`
	`$d = ($a XOR $b); //$d vaut FALSE.`
	`$e = ($b XOR $c); //$e vaut TRUE.`

AND	Teste si les deux opérandes valent TRUE en même temps :
	`$a = true;`
	`$b = true;`
	`$c = false;`
	`$d = ($a AND $b); //$d vaut TRUE.`
	`$e = ($b AND $c); //$e vaut FALSE.`

&&	Équivaut à l'opérateur AND mais n'a pas la même priorité.

!	Opérateur unaire de négation, qui inverse la valeur de l'opérande :
	`$a = TRUE;`
	`$b = FALSE;`
	`$d = !$a; //$d vaut FALSE.`
	`$e = !$b; //$e vaut TRUE.`

Attention

Une erreur classique dans l'écriture des expressions conditionnelles consiste à confondre l'opérateur de comparaison == avec l'opérateur d'affectation =.

L'usage des parenthèses dans la rédaction des expressions booléennes est souvent indispensable et toujours recommandé pour éviter les problèmes liés à l'ordre d'évaluation des opérateurs.

Les chaînes de caractères

Les chaînes de caractères sont avec les nombres les types de données les plus manipulés sur un site Web. De surcroît, dans les échanges entre le client et le serveur au moyen de formulaires, toutes les données sont transmises sous forme de chaînes, d'où leur importance.

Définir des chaînes

Une chaîne de caractères est une suite de caractères alphanumériques contenus entre des guillemets simples (apostrophes) ou doubles. Par exemple :

```
$a = 'PHP5 et MySQL';
$b = "PHP5 et MySQL";
```

Si les chaînes ne contiennent que des caractères, les deux types de notation sont parfaitement équivalents. Si une chaîne contient une variable, celle-ci est évaluée, et sa valeur incorporée à la chaîne uniquement si vous utilisez des guillemets et non des apostrophes :

```
$a = 'PHP';
```

```
$b = 'MySQL';
$c = "PHP et $b";//affiche : PHP et MySQL
$d = 'PHP et $b';
/*affiche PHP et $b car $ et b sont considérés comme des caractères sans
➥signification particulière*/
```

Se pose alors la question de l'inclusion des guillemets simples ou doubles comme caractères normaux à l'intérieur d'une chaîne.

Pour inclure une apostrophe dans une chaîne délimitée par des apostrophes, il faut les faire précéder du caractère d'échappement antislash \. Le principe est le même pour les guillemets.

L'exemple suivant :

```
$a = 'Faire l\'ouverture ';
echo $a;
```

affiche le texte « Faire l'ouverture », et le suivant :

```
$b = "Sa devise est : \"Liberté, Égalité, Fraternité\" ";
echo $b;
```

affiche « Sa devise est : "Liberté, Égalité, Fraternité" ».

Si vous voulez utiliser le caractère \ en tant que tel dans une chaîne, vous devez le faire précéder d'un autre antislash.

L'exemple suivant :

```
$path = "C:\\php\\www\\exemple.php";
echo $path;
```

affiche « C:\php\www\exemple.php ».

Le tableau 2-9 indique les séquences d'échappement utiles en PHP.

Tableau 2-9 – Les séquences d'échappement

Séquence	Signification
\'	Affiche une apostrophe.
\"	Affiche des guillemets.
\$	Affiche le signe $.
\\	Affiche un antislash.
\n	Nouvelle ligne (code ASCII 0x0A).
\r	Retour chariot (code ASCII 0x0D).
\t	Tabulation horizontale (code ASCII 0x09).
\[0-7] {1,3}	Séquence de caractères désignant un nombre octal (de 1 à 3 caractères 0 à 7) et affichant le caractère correspondant : `echo "\115\171\123\121\114"; //Affiche MySQL.`

Pour bien voir le danger de ne pas utiliser de guillemets pour définir les clés, analysez le code suivant, dans lequel l'élément de clé `"lang"` est défini avec la valeur `"PHP et MySQL"` puis un élément dont la clé est `CTE` (sans guillemets) et la valeur `"ASP.NET"`. Comme vous n'avez pas utilisé les guillemets, `CTE` ne représente pas la chaîne `"CTE"` mais la valeur d'une constante définie précédemment et dont la valeur est `"lang"`. En affichant `$tab2["lang"]` vous obtenez la valeur `"ASP.NET"` et non `"PHP ET MySQL"`, laquelle a été écrasée. De même, en affichant `$tab2["CTE"]`, vous obtenez la valeur `"JAVA"`.

L'utilisation d'éléments de tableau associatif dans des chaînes pose problème.

Le fait d'écrire :

```
echo "<p> Vous utilisez $tab2['deux'] <br />";
```

provoque une erreur et l'arrêt du script, ce qui ne se produit pas avec un tableau indicé. Pour pallier cet inconvénient, il faut que la variable soit contenue dans des accolades, comme dans l'exemple ci-dessous :

```
echo "<p> Vous utilisez {$tab2['deux']} <br />";
```

qui réalise un affichage normal, ou encore concaténer les chaînes et la variable comme ci-dessous :

```
echo "<p> Vous utilisez".$tab2['deux']. " <br />";
```

Exemple 2-2. Création de tableaux associatifs

```
<?php
$tab2["zéro"] = 2003;
$tab2["pi"] = 31.14E7;
$tab2["deux"] = "PHP";
//***La ligne suivante provoque une erreur si elle est décommentée
//echo "<p> Vous utilisez $tab2['deux'] <br />";
//***on écrira à la place:
echo "<p> Vous utilisez {$tab2['deux']} <br />";
define("CTE","lang");//Crée la constante CTE
$tab2["lang"] = " PHP ET MySQL";
$tab2[CTE] = " ASP.NET";
$tab2["CTE"] = "JAVA";
echo "Le nombre d'éléments est ", count($tab2),"<br />";
echo "L'élément \$tab2[\"CTE\"] vaut ",$tab2["CTE"],"<br / >";
echo "L'élément \$tab2[CTE] vaut ",$tab2[CTE],"<br />";
echo "<p> Le langage préféré de l _ Open source est{$tab2["lang"]} <br />";
?>
```

Le script retourne le résultat suivant :

```
Vous utilisez PHP
Le nombre d'éléments est 5
L'élément $tab2["CTE"] vaut JAVA
L'élément $tab2[CTE] vaut ASP.NET
Le langage préféré de l'Open source est ASP.NET
```

Les types divers

PHP offre également deux types particuliers qui sont utilisés dans des circonstances bien définies.

Le type resource

Le type `resource` représente une référence à des informations présentes sur le serveur. Il est le type retourné par certaines fonctions particulières. C'est le cas, entre autres, des fonctions utilisées pour accéder à une base de données lors de la connexion, qui retournent une valeur de type `resource`. Cette dernière permet d'identifier chaque connexion initiée par un utilisateur puis est utilisée pour retourner les données après interrogation de la base par l'utilisateur concerné. Cet identifiant trouve toute son utilité quand il y a plusieurs connexions simultanées sur une même base, notamment à partir d'un même script.

L'exemple suivant réalise une connexion au serveur MySQL à l'aide de la fonction `mysql_connect()` et récupère un identifiant de connexion `$connect`, qui est la valeur retournée par cette fonction. Il affiche ensuite la valeur puis le type de cette variable.

```
<?php
//***************Le type resource***************
$connect = mysql_connect("localhost","root","") or die ("ERREUR de CONNEXION");
echo "L'identifiant de connexion vaut : $connect <br />";
echo "Le type de la variable $connect est ",gettype($connect);
?>
```

Le script affiche le résultat suivant :

```
L'identificateur de connexion vaut : Resource id #1
Le type de la variable $connect est resource
```

La lecture de la valeur de la variable `$connect` n'a pas d'intérêt particulier une fois la connexion réalisée, mais sa récupération permet d'accéder à la base de données. Pour plus de détails, voir le chapitre 15, consacré à l'accès aux bases de données MySQL.

Le type NULL

Le type `NULL`, ou `null`, est celui qui est attribué à une variable qui n'a pas de contenu ou qui a été explicitement initialisée avec la valeur `NULL`. Aussitôt qu'une valeur légale est donnée à la variable, elle prend le type correspondant.

> **Attention : *NULL* et *zéro***
>
> Une variable contenant une chaîne vide ou la valeur "0" n'a pas le type `NULL` mais `string`. De même, une variable contenant la valeur 0 est du type `integer`.

Dans le code suivant, le code de création de la variable `$varvide` (repère ❶) est déconseillé dans un script, car il génère un avertissement. De même, l'utilisation de cette variable par l'instruction `echo` est à proscrire.

```
<?php
//***************le type NULL*****************************
$varvide; ←❶
echo "La variable vide vaut : $varvide <br />";
echo "Le type de la variable \$varvide est ",gettype($varvide)," <br />";
$varvide="";
echo "La variable vide vaut : $varvide <br />";
echo "Le type de la variable \$varvide est ",gettype($varvide);
?>
```

Le code affiche le résultat suivant :

```
La variable vide vaut :
Le type de la variable $varvide est NULL
La variable vide vaut :
Le type de la variable $varvide est string
```

Mémo des fonctions

`boolean define(string nom_cte, valeur [, bool casse])`

Crée la constante `nom_cte` et lui attribue une valeur. Le paramètre `casse` indique que le nom de la constante est insensible à la casse (`TRUE`) ou non.

`boolean defined(string nom_cte)`

Retourne `TRUE` si la constante `nom_cte` existe et `FALSE` dans le cas contraire.

`string gettype($nom_var)`

Retourne le type de la variable `$nom_var`.

`boolean empty($nom_var)`

Retourne `TRUE` si la variable `$nom_var` n'est pas affectée ou a une des valeurs `NULL`, `0` ou `"0"` et `FALSE` dans le cas contraire.

`boolean isset($nom_var)`

Retourne `TRUE` si la variable `$nom_var` existe et est définie avec une valeur différente de `NULL`.

`array get_defined_constants()`

Retourne un tableau contenant toutes les constantes prédéfinies et celles qui ont été créées dans le script.

`boolean settype($var, string type)`

Effectue le transtypage de `$var` dans le type précisé. Retourne `TRUE` si l'opération est réussie et `FALSE` dans le cas contraire.

Plusieurs instructions `case` différentes peuvent se succéder avant qu'intervienne un bloc d'instructions (voir repère ❶ dans l'exemple 3-4). Dans ce cas, les différentes valeurs indiquées déclenchent l'exécution du même code (repère ❷).

Le script précédent devient celui de l'exemple 3-4.

☛ **Exemple 3-4. L'instruction *switch…case***

```
<?php
$dept=75;
switch($dept)
{
//Premier cas
case 75: ←❶
case "Capitale": ←❶
echo "Paris"; ←❷
break;
//Deuxième cas
case 78:
echo "Hauts de Seine";
break;
//Troisième cas
case 93:
case "Stade de france":
echo "Seine Saint Denis";
break;
//la suite des départements...
//Cas par défaut
default:
echo "Département inconnu en Ile de France";
break;
}
?>
```


Les instructions de boucle

Les boucles permettent de répéter des opérations élémentaires un grand nombre de fois sans avoir à réécrire le même code. Selon l'instruction de boucle utilisée, le nombre d'itérations peut être défini à l'avance ou être déterminé par une condition particulière.

La boucle for

Présente dans de nombreux langages, la boucle `for` permet d'exécuter plusieurs fois la même instruction ou le même bloc sans avoir à réécrire les mêmes instructions. Sa syntaxe est la suivante :

```
for(expression1; expression2; expression3)
{
//instruction ou bloc;
}
```

`expression1` est toujours évaluée. Il s'agit généralement de l'initialisation d'une ou plusieurs variables servant de compteur pour la boucle. `expression2` est ensuite évaluée avec une valeur booléenne : si elle vaut `TRUE`, la boucle continue et les instructions comprises dans le bloc sont exécutées, sinon la boucle s'arrête. Si elle est toujours vraie on obtient une boucle infinie, vérifiez donc qu'elle peut être fausse. `expression3` n'est exécutée qu'à la fin de chaque itération. Il s'agit le plus souvent d'une instruction d'incrémentation de la variable compteur.

L'exemple 3-5 crée un document qui affiche six niveaux de titre utilisant les balises `<h1>` à `<h6>` en deux lignes de code seulement.

Exemple 3-5. Une boucle *for* simple

```
<?php
for($i=1;$i<7;$i++)
{
echo "<h$i> $i :Titre de niveau $i </h$i>";
}
?>
```

Le résultat de la boucle est illustré à la figure 3-1.

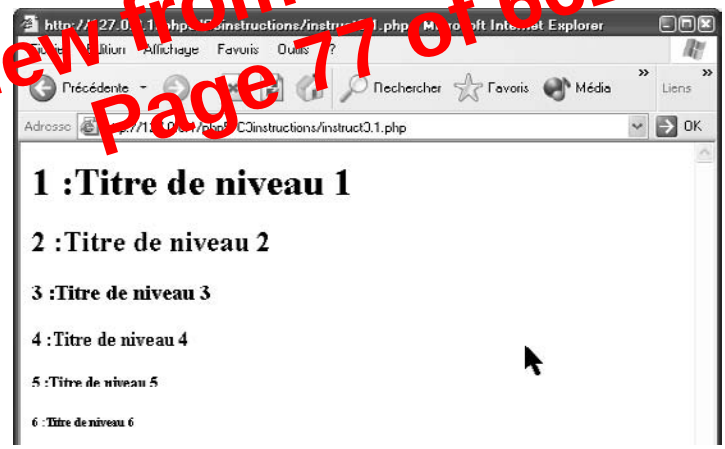


Figure 3-1
Création de titres

Les trois expressions utilisées dans la boucle `for` peuvent contenir plusieurs parties séparées par des virgules. La boucle peut en ce cas être réalisée sur plusieurs variables, comme illustré à l'exemple 3-6.

Exemple 3-6. Une boucle à plusieurs variables

```
<?php
for($i=1,$j=9;$i<10,$j>0;$i++,$j--)
//$i varie de 1 à 9 et $j de 9 à 1
{
echo "<span style=\"border-style:double;border-width:3;\"> $i + $j=10</span>";
}
?>
```

Le résultat de cette boucle double est la table d'addition de la figure 3-2.

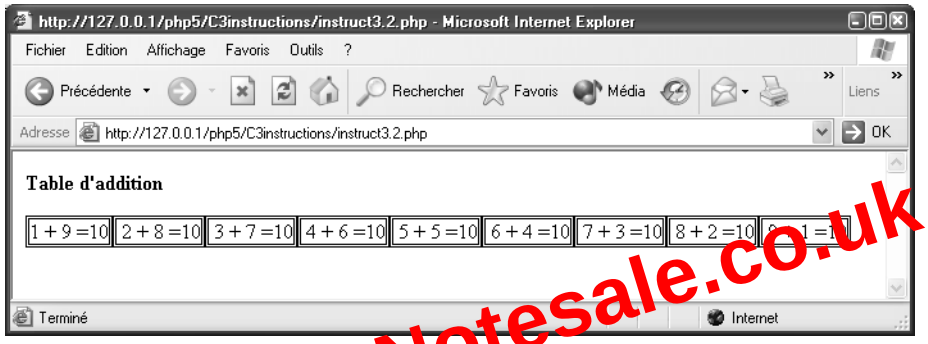


Figure 3-2
Table d'addition créée par une boucle double.

Les boucles imbriquées

Il est possible d'imbriquer des boucles `for` les unes dans les autres sur autant de niveaux que désiré, le bloc qui suit la première contenant toutes les autres. Chaque variable compteur déclarée dans une boucle n'est utilisable que dans la boucle qui la déclare et dans celles de niveau inférieur.

L'exemple 3-7 ci-après crée une table de multiplication dans un tableau XHTML à deux dimensions, chaque dimension étant gérée par une variable compteur différente, `$i` et `$j`. La première boucle `for` (repère ❶) ne sert qu'à créer la ligne d'en-tête de la table. La variable `$i` est locale à la boucle. Le fait de réutiliser le même nom de variable dans la boucle suivante n'a donc aucune importance.

Deux autres boucles imbriquées sont ensuite utilisées pour créer le corps de la table. La première (repère ❷) itère les numéros de ligne avec la variable `$i`, et la deuxième (repère ❸) le contenu des cellules du tableau avec la variable `$j`.

Exemple 3-7. Les boucles *for* imbriquées

```
<?php
echo "<h2> Révisez votre table de multiplication!</ h2>";
//Début du tableau HTML
```

Vous obtenez, par exemple, la suite de nombres 72 / 79 / 50 / 95 / 11 / 43 / 18 / 49 / (cette suite varie évidemment à chaque tirage).

La boucle do...while

La boucle `do...while` apporte une précision à la boucle `while`. Dans celle-ci, en effet, si l'expression booléenne est évaluée à `FALSE`, les instructions qu'elle contient ne sont jamais exécutées. Avec l'instruction `do...while`, au contraire, la condition n'est évaluée qu'après une première exécution des instructions du bloc compris entre `do` et `while`.

La syntaxe de la boucle `do…while` est la suivante :

```
do {
//bloc d'instructions
}
while(expression);
```

Le script de l'exemple 3-9 reprend celui de l'exemple 3-8, mais il n'est plus besoin cette fois d'initialiser la variable `$i` à 1 car la divisibilité par 7 n'est testée qu'après le premier tirage (repère ❶).

Exemple 3-9. Tirage avec une boucle *do…while*

```
<?php
do
{
$n = rand(1,100);
echo $n."  ";
}
while($n%7!=0); ←❶
?>
```


Les résultats obtenus sont similaires.

La boucle foreach

Introduite à partir de la version 4.0 de PHP, l'instruction `foreach` permet de parcourir rapidement l'ensemble des éléments d'un tableau, ce que fait aussi une boucle `for`, mais `foreach` se révèle beaucoup plus efficace.

La boucle `foreach` est particulièrement efficace pour lister les tableaux associatifs dont il n'est nécessaire de connaître ni le nombre d'éléments ni les clés. Sa syntaxe est variable selon que vous souhaitez récupérer seulement les valeurs ou les valeurs et les clés (ou les indices).

Pour lire les valeurs seules, la première syntaxe est la suivante :

```
foreach($tableau as $valeur)
{
//bloc utilisant la valeur de l'élément courant
}
```

Il est évident que pendant la phase de développement des scripts, ces méthodes doivent être désactivées pour permettre au programmeur d'être alerté sur les causes et la localisation des erreurs.

Gestion des exceptions

Une exception est un mécanisme qui permet d'intercepter une erreur générée par un script et de déclencher une action en réponse à cette erreur. Si PHP 4 ne permettait pas d'effectuer une gestion des exceptions, la version 5 fournit un mécanisme qui permet de gérer les conséquences d'une erreur.

La classe *Exception*

Si vous n'êtes pas familiarisé avec les notions de classe ou d'objet, ainsi que de propriété et de méthode, reportez vous au chapitre 9 pour acquérir ces quelques notions de base.

PHP 5 introduit la classe prédéfinie `Exception` qui offre une gestion évoluée des exceptions.

Gérer une exception consiste à délimiter un ou des blocs de code et à prévoir une action particulière qui doit être réalisée dans le cas où l'erreur prévue se produit. Ces blocs constituent les gestionnaires d'exception. Pour les créer, procédez de l'une des façons suivantes :

- Créez un bloc à l'aide de l'instruction `try` qui délimite le code dans lequel peut survenir une erreur. Il peut s'agir, par exemple, du code qui gère les saisies faites par des utilisateurs dans un formulaire (voir le chapitre 6). Ce bloc contient une instruction `throw` pour déclencher l'exception en créant un objet de type `Exception` à l'aide du mot-clé `new`.
- Créez un bloc à l'aide de l'instruction `catch` associée au bloc `try` précédent. Il comporte le code qui va gérer l'erreur si elle se produit. C'est ce bloc qui utilise l'objet `Exception` créé par l'instruction `throw`. Si aucune erreur ne se produit dans le bloc `try`, l'objet `Exception` n'est pas créé, et le code du bloc `catch` est ignoré. L'exécution se poursuit dans tous les cas après le bloc `catch`.

Un gestionnaire d'exception a donc le structure suivante :

```
try
{
  //Code à surveiller
  if(erreur prévue){throw new Exception();}
  else{// Résultat;}
}
catch(Exception $except)
{
  //Gestion de l'erreur
}
```

Le nom de l'objet utilisé dans l'instruction `catch` est sans importance. Un même script peut comporter autant de bloc `try…catch` que vous voulez. Il est également possible

```
if(!strpos ($ch,$ch4))
{ echo "Le mot $ch4 n'est pas dans \$ch";}
?>
```

Le résultat affiché est le suivant :

```
$ch = Perette et le pot au lait. C'est pas de pot ! La Fontaine
Le mot pot commence à la position 14 dans $ch
Le mot POT commence à la position 14 dans $ch
La dernière occurrence du mot pot commence à la position 40 dans $ch
Le mot fontaine n'est pas dans $ch
```

Les mêmes conventions de formatage utilisées par la fonction `printf()` sont utilisables pour extraire des sous-chaînes et affecter leurs valeurs à des variables. Vous pouvez utiliser ces variables indépendamment de la chaîne dont elles sont extraites à l'aide de la fonction `sscanf()`, dont la syntaxe est la suivante :

```
array|string sscanf (string $ch, string "format" [, $var1,$var2,…])
```

Pour utiliser cette fonction, la chaîne `$ch` doit répondre au format défini à l'aide des spécificateurs récapitulés au tableau 4-1, que vous avons déjà rencontrés avec la fonction `printf()`. Elle peut alors être bien analysée. Chacun de ces éléments est affecté aux variables dont les noms sont donnés dans les paramètres `$var1`, `$var2`, etc.

Dans l'exemple 4-6, la chaîne `$personne` contient quatre informations écrites selon le format précis suivant :

```
"date_de_naissance-date_de_décès prénom nom"
```

Dans l'exemple 4-7, la chaîne de format qui permet d'analyser et de récupérer chacun de ses éléments est la suivante (voir repère ❶) :

```
$format="%d-%d %s %s";
```

La fonction `sscanf()` affecte ces informations aux variables nommées `$ne`, `$mort`, `$prenom` et `$nom` (repère ❷) et retourne le nombre d'informations dans la variable `$nb`. Ces variables sont utilisées pour un affichage (repère ❸).

Exemple 4-7. Capture de sous-chaînes dans des variables

```
<?php
$personne = "1685-1750 Jean-Sébastien Bach";
$format="%d-%d %s %s"; ←❶
$nb = sscanf($personne,$format,$ne,$mort,$prenom,$nom); ←❷
echo "$prenom $nom né en $ne, mort en $mort <br />"; ←❸
echo "Nous lisons $nb informations";;
?>
```

Le résultat affiché est le suivant :

```
Jean-Sébastien Bach né en 1685, mort en 1750
Nous lisons 4 informations
```

Définition d'un motif élémentaire

L'écriture des motifs d'expressions régulières est la partie la plus rébarbative du travail de codage. Un grand soin est requis dans leurs écritures car cela conditionne la qualité du résultat. Les motifs sont toujours contenus dans des chaînes de caractères, et donc entre guillemets.

Recherche de un ou plusieurs caractères

Pour rechercher la présence d'un caractère particulier, il suffit de l'inclure entre des guillemets simples ou doubles. Pour rechercher le caractère `@`, vous écrivez le modèle :

```
$modele="@";
```

Pour vérifier si une chaîne contient un au moins des caractères d'une liste, vous énumérez tous ces caractères entre crochets.

Pour rechercher un ou plusieurs des caractères `xyz`, vous avez donc le motif suivant :

```
$modele="[xyz]";
```

Caractère et caractères

La présence d'un seul des caractères de la liste dans la chaîne sur laquelle s'effectue la recherche fournit un résultat positif. La présence d'autres caractères quelconques est alors possible. En d'autres termes, la présence d'au moins un des caractères définis dans le motif ne signifie pas qu'il n'y en a pas d'autres non contenus dans le motif.

Avec la même syntaxe vous pouvez définir comme motif un intervalle de lettres ou de chiffres.

Par exemple, si vous écrivez :

```
$modele = "[a-z]";
```

vous recherchez un mot d'au moins un caractère contenant n'importe quelles lettres minuscules.

De même, le motif suivant :

```
$modele = "[A-Z]";
```

recherche n'importe quel groupe de majuscules.

Le suivant :

```
$modele = "[0-9]";
```

recherche la présence d'au moins un chiffre entre 0 et 9.

Échappement des caractères spéciaux

Pour rechercher dans une chaîne les caractères qui ont une signification particulière, vous devez les échapper en les faisant précéder d'un antislash (`\`). Cela s'applique aux caractères `.`, `$`, `^`, `?`, `\`, `[`, `]`, `(`, `)`, `+` et `*`.

```
$ch2=ereg_replace ("[[:digit:]]", "5", $ch); ←1
echo $ch2,"<br />";
echo ereg_replace (" est", " fut", $ch),"<br />" ; ←2
echo ereg_replace ("\n", "<br />", $ch2),"<br />"; ←3
echo $ch,"<br />";;
//
$ch = "Quatre mariages et un enterrement ";
$ch2=eregi_replace ("MARIAGE", "divorce", $ch); ←4
$ch2=ereg_replace ("enterrement", "mariage", $ch2); ←5
echo $ch2,"<br />";
//
$ch = "Nous sommes en 2003 ";
echo ereg_replace ("[0-9]{4}", date("Y"), $ch); ←6
?>
```

L'exemple affiche le résultat suivant :

```
La 5e version est PHP 5: Viva PHP5
La 4e version fut PHP 4: Viva PHP4
La 5e
version est PHP 5:
Viva PHP5
La 4e version est PHP 4: Viva PHP4
Quatre divorces et un mariage
Nous sommes en 2004
```


Définition d'un motif complexe

Vous allez maintenant aborder la manière de créer des modèles complexes afin de réaliser des recherches et des validations d'expressions complexes, comme des montants financiers ou des adresses e-mail.

Elle consiste à combiner les différentes possibilités que vous avez découvertes à la section précédente.

Validation d'un nom

En prenant pour hypothèse qu'un nom de famille est composé uniquement de lettres et des caractères apostrophes (') et tiret (-) pour les prénoms composés, à l'exclusion de tout autre, vous créez le modèle suivant :

```
$modele="^([a-zA-Z])([- a-zA-Z]*)$";
```

Il ne recherche que les noms commençant par des lettres suivies éventuellement d'un tiret, d'une espace et de lettres, ces caractères devant constituer la fin du nom. Les chiffres et les autres caractères sont exclus.

L'exemple 4-11 affiche un message si le nom ne correspond pas à ces critères.

5

Les tableaux

Comme expliqué au chapitre 3, consacré aux types de données accessibles dans PHP, les tableaux représentés par le type `array` sont d'une utilisation courante dans les scripts. La possibilité de stocker un grand nombre de valeurs sous un seul nom de variable offre des avantages appréciables, notamment une grande souplesse dans la manipulation des données. Les nombreuses fonctions natives de PHP applicables aux tableaux permettent les opérations les plus diverses dans la gestion des tableaux.

Dans ce chapitre, vous verrez :

- les différentes façons de créer des tableaux ;
- les méthodes de lecture des éléments de tableau ;
- les fonctions de manipulation des tableaux.

Créer des tableaux

La fonction array()

La fonction `array()` permet de créer de manière rapide des tableaux indicés ou associatifs. C'est elle qui sera le plus souvent utilisée pour la création de tableaux.

Les tableaux indicés

La façon la plus élémentaire de créer un tableau indicé consiste à définir individuellement une valeur pour chacun des ses éléments, et ce de la manière suivante :

```
$tab[n] = valeur;
```

Le résultat de l'exemple 5-8 est identique à celui de la figure 5-2 réalisé avec une boucle `for`.

Indice non consécutif

Si, après avoir défini le tableau `$clients`, vous lui ajoutez un élément au moyen de l'instruction :

```
$clients[7] = array("Duval","Marseille","76");
```

créant ainsi un indice non consécutif aux trois premiers, cet élément n'est pas lu dans la boucle. Cette méthode n'est donc pas adaptée à ce cas particulier.

Lire à l'aide de la fonction each()

Pour pallier l'inconvénient signalé à la remarque précédente, il est possible d'utiliser une autre méthode de lecture. Cette dernière fait appel à la fois à une boucle `while` et à la fonction `each()`, qui reçoit comme paramètre une variable de type `array`. Cette dernière a la particularité de retourner un tableau à quatre éléments qui contient les informations sur l'élément courant du tableau passé en paramètre puis de pointer sur l'élément suivant.

La syntaxe de la fonction `each()` est la suivante :

```
$element = each($tab)
```

`$tab` est le tableau à lire et `$element` le tableau de résultats contenant les informations sur l'élément courant de `$tab`, sous la forme :

- `$element[0]`, qui contient l'indice de l'élément courant.
- `$element[1]`, qui contient la valeur de l'élément courant.
- `$element["key"]`, qui contient la clé de l'élément courant.
- `$element["value"]`, qui contient la valeur de l'élément courant.

Les couples `$element[0]-$element[1]` sont généralement utilisés pour récupérer les couples indice-valeur des tableaux indicés, et les couples `$element["key"]-$element["value"]` pour récupérer les couples clé-valeur des tableaux associatifs. Cet usage n'a toutefois d'autre justification que la force de l'habitude.

Par exemple, les deux lignes de code suivantes :

```
echo "L'élément d'indice $element[0] a la valeur $element[1]<br />";
echo "L'élément de clé {$element['key']} a la valeur {$element['value']}<br />";
```

affichent exactement le même résultat.

L'expression `$element=each($tab)` étant évaluée à `TRUE` tant que le tableau contient des éléments, placez-la dans une boucle `while` de façon à pouvoir lire l'ensemble des éléments. Arrivé à la fin du tableau, cette expression prend la valeur `FALSE`, ce qui arrête la boucle.

Pour vous assurer que le pointeur interne du tableau est positionné au début du tableau, vous pouvez appeler la fonction `reset()`, dont c'est le rôle, en utilisant comme paramètre le tableau à lire avant de commencer la lecture. L'avantage principal de cette méthode de lecture est de donner accès aussi bien à des tableaux indicés qu'à des tableaux associatifs.

Le listing de l'exemple 5-14 affiche le résultat suivant :

```
Exemple 1
Structure du tableau initial :
Array ( [UN] => Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 ) [DEUX] =>
Array ( [0] => a [1] => b [2] => c ) [0] => Array ( [0] => A [1] => B [2] => C [3] =>
D [4] => E ) [1] => Array ( [0] => 11 [1] => 12 [2] => 13 [3] => 14 [4] => 15 ) )

array_slice($tab,1,2) donne : Array ( [DEUX] => Array ( [0] => a [1] => b [2] => c )
[0] => Array ( [0] => A [1] => B [2] => C [3] => D [4] => E ) )

Exemple 2
Structure du tableau initial :
Array ( [0] => Spock [1] => Batman [2] => Dark Vador [3] => Hal [4] => Frodo [5] =>
Sky Walker [6] => Amidala [7] => Alien )

array_slice($heros,0,5)Array ( [0] => Spock [1] => Batman [2] => Dark Vador [3] =>
Hal [4] => Frodo )

array_slice($heros,-5,5) Array ( [0] => Hal [1] => Frodo [2] => Sky Walker [3] =>
Amidala [4] => Alien )

array_slice($heros,-5,3) Array ( [0] => Hal [1] => Frodo [2] => Sky Walker )

array_slice($heros,1,-2) Array ( [0] => Batman [1] => Dark Vador [2] => Hal [3] =>
Frodo [4] => Sky Walker )

array_slice($heros,-5,-2) Array ( [0] => Hal [1] => Frodo [2] => Sky Walker )
```


Ajouter et enlever des éléments

Une fois un tableau créé à l'aide de la fonction `array()` et certaines valeurs affectées à ses éléments, vous pouvez effectuer diverses manipulations d'ajout ou de retrait d'éléments selon les besoins.

La fonction :

```
int  array_push($tab, valeur1, valeur2,…, valeurN)
```

ajoute en une seule opération les `N` éléments passés en paramètres à la fin du tableau désigné par la variable `$tab`. Vous pouvez évidemment remplacer les valeurs passées en paramètres par des variables.

Les nouveaux indices ainsi créés ont pour valeur celle du plus grand indice existant (donc égal à `count($tab)-1`) incrémenté de 1 jusqu'à *N*. La fonction retourne également le nouveau nombre d'éléments du tableau modifié.

Pour ajouter des éléments au début d'un tableau, vous pouvez utiliser la fonction suivante :

```
int array_unshift($tab, valeur1, valeur2,…, valeurN)
```

```
echo "<hr />";
//Ajout au début du tableau
$poitiers=732;
$nb=array_unshift($tab,500,$poitiers);
echo "Le tableau \$tab a maintenant $nb éléments <br>";
print_r($tab);
echo "<hr />";
//Ajout à la fin du tableau
$armi=1918;
$newnb=array_push($tab,1870,1914,$armi);
echo "Le tableau \$tab a maintenant $newnb éléments <br>";
print_r($tab);
echo "<hr />";
//Suppression du dernier élément
$suppr= array_pop($tab);
echo "Le tableau \$tab a perdu l'élément $suppr <br>";
print_r($tab);
echo "<hr />";
//Suppression du premier élément
$suppr= array_shift($tab);
echo "Le tableau \$tab a perdu l'élément $suppr <br>";
print_r($tab);
echo "<hr />";
//Suppression de l'élément d'indice 4
unset($tab[4]);
print_r($tab);
?>
```


Le résultat du listing de l'exemple 5-15 permet de suivre l'évolution du tableau initial au fur et à mesure des modifications opérées :

```
Array ( [0] => 800 [1] => 1492 [2] => 1515 [3] => 1789 )
Le tableau $tab a maintenant 6 éléments
Array ( [0] => 500 [1] => 732 [2] => 800 [3] => 1492 [4] => 1515 [5] => 1789 )
Le tableau $tab a maintenant 9 éléments
Array ( [0] => 500 [1] => 732 [2] => 800 [3] => 1492 [4] => 1515 [5] => 1789 [6] =>
1870 [7] => 1914 [8] => 1918 )
Le tableau $tab a perdu l'élément 1918
Array ( [0] => 500 [1] => 732 [2] => 800 [3] => 1492 [4] => 1515 [5] => 1789 [6] =>
1870 [7] => 1914 )
Le tableau $tab a perdu l'élément 500
Array ( [0] => 732 [1] => 800 [2] => 1492 [3] => 1515 [4] => 1789 [5] => 1870 [6] =>
1914 )
L'élément d'indice 4 a été supprimé
Array ( [0] => 732 [1] => 800 [2] => 1492 [3] => 1515 [5] => 1870 [6] => 1914 )
```

Suite à une recherche dans une base de données à l'aide de critères multiples, un tableau peut être amené à contenir plusieurs fois les mêmes valeurs pour certains critères. Avant de traiter les données du tableau, il peut être préférable en ce cas d'éliminer les éléments faisant double emploi.

Elle retourne un tableau contenant les éléments présents dans le premier paramètre mais pas dans le second. Comme pour la soustraction de nombres, il est logique que l'inversion des paramètres ne fournisse pas le même résultat. Les indices associés aux valeurs dans les tableaux d'origine sont conservés.

Si vous appliquez ces deux fonctions à des tableaux associatifs, les clés sont conservées dans les mêmes conditions. De plus, si nos exemples ne montrent que l'intersection de deux tableaux, il est parfaitement licite de passer à ces fonctions un nombre quelconque de paramètres, du moment qu'il s'agit bien de variables de type `array`.

Exemple 5-17. Intersection et différence de deux tableaux

```
<?php
$tab1=array("Blanc","Jaune","Rouge","Vert","Bleu","Noir");
$tab2=array("Bleu","Rouge","Violet","Noir","Jaune","Orange");
echo"Le tableau 1 contient les éléments:<br />";
print_r($tab1);
echo "<hr />";
echo"Le tableau 2 contient les éléments:<br />";
print_r($tab2);
echo "<hr />";
echo "Intersection de \$tab1 et \$tab2 : ";
$tab3=array_intersect($tab1,$tab2);
print_r($tab3);
echo"<br />";
echo "Intersection de \$tab2 et \$tab1 : ";
$tab4=array_intersect($tab2,$tab1);
print_r($tab4);
echo"<hr />";
$tab5= array_diff($tab1,$tab2);
echo "Différence de \$tab1 et \$tab2 : ";
print_r($tab5);
echo"<br />";
$tab6= array_diff($tab2,$tab1);
echo "Différence de \$tab2 et \$tab1 : ";
print_r($tab6);
echo"<br />";
?>
```

Le script affiche les résultats suivants, qui montrent bien l'importance de l'ordre des paramètres.

```
Le tableau 1 contient les éléments:
Array ( [0] => Blanc [1] => Jaune [2] => Rouge [3] => Vert [4] => Bleu [5] => Noir )
Le tableau 2 contient les éléments:
Array ( [0] => Bleu [1] => Rouge [2] => Violet [3] => Noir [4] => Jaune [5] => Orange )
```

Cette fonction transforme toutes les clés du tableau `$tab` en minuscules si la constante `CTE` vaut `CASE_LOWER` (valeur par défaut) ou en majuscules si elle vaut `CASE_UPPER`.

```
Tableau associatif d'origine
Array ( [white2] => Blanc2 [yellow] => Jaune [red] => rouge [green] => Vert [ blue] =>
Bleu [black] => Noir [white10] => Blanc10 )
Tri en ordre alpha des clés
Array ( [ blue] => Bleu [black] => Noir [green] => Vert [red] => rouge [white10] =>
Blanc10 [white2] => Blanc2 [yellow] => Jaune )
Tri en ordre alpha inverse des clés
Array ( [yellow] => Jaune [white2] => Blanc2 [white10] => Blanc10 [red] => rouge
[green] => Vert [black] => Noir [ blue] => Bleu )
Tri selon la longueur des clés
Array ( [white10] => Blanc10 [white2] => Blanc2 [yellow] => Jaune [black] => Noir
[green] => Vert [ blue] => Bleu [red] => rouge )
```

Opérer une sélection des éléments

Lorsqu'un tableau contient un nombre important d'informations, vous pouvez réaliser une sélection de ses éléments à l'aide de la fonction `array_filter()` et ne retenir que ceux qui répondent à une condition particulière définie par le programmeur.

La syntaxe de la fonction `array_filter()` est la suivante :

```
array array_filter(array $tab, string "nom_fonction")
```

Elle retourne un nouveau tableau ne contenant que les éléments de `$tab` qui répondent à la condition définie dans la fonction dont le nom est passé en second paramètre. Le tableau initial est conservé.

L'exemple 5-24 utilise la fonction `array_filter()` pour sélectionner parmi les éléments d'un tableau contenant des noms de villes celles dont l'initiale est `"P"` ou `"p"`.

La fonction de sélection doit avoir comme paramètre une variable qui représente la valeur d'un élément courant du tableau sur lequel s'effectue la sélection et retourner cette variable si elle répond à la condition énoncée.

Exemple 5-24. Sélection dans un tableau

```
<?php
//Définition du tableau
$villes=array("Paris","Perpignan","Marseille","pau","Nantes","Lille");
//Fonction de sélection
function init($ville)
{
  if($ville[0]=="P" || $ville[0]=="p" )
  {
  return $ville;
  }
}
```

L'attribut `checked="checked"` définit le bouton coché par défaut. L'attribut `value` joue le même rôle que pour les cases à cocher et est également indispensable.

Voici un exemple d'utilisation de l'élément `"radio"` :

```
<label>Débutant</label><input type ="radio" name="capa" value="débutant" />
<label>Initié</label><input type ="radio" name="capa" value="initié" />
```

L'aspect visuel de ce composant est illustré à la figure 6-1 (repère ❺).

L'élément *<input type="submit" />*

Crée un bouton sur lequel l'utilisateur doit cliquer pour déclencher l'envoi des données de tout le formulaire vers le serveur.

Il est indispensable d'avoir au moins un bouton d'envoi par formulaire, mais il est possible d'en utiliser plusieurs. Le clic sur l'un de ces boutons est alors analysé par le script désigné par l'attribut `action` de l'élément `<form>`. Cela permet d'effectuer des tâches spécialisées en fonction de la valeur associée à chaque bouton grâce à son attribut `value`. C'est le contenu de l'attribut `value` qui constitue le texte visible du bouton dans le formulaire.

Vous pourrez voir des exemples d'utilisation de plusieurs boutons d'envoi à la fin de ce chapitre ainsi qu'au chapitre 13. L'attribut `name` n'est *a priori* pas utile, en particulier s'il n'y a qu'un seul bouton d'envoi.

Voici un exemple simple d'utilisation de l'élément `"submit"` :

```
<input type ="submit" value="Envoyer" />
```

L'aspect visuel de ce composant est illustré à la figure 6-1 (repère ❿).

L'élément *<input type="reset" />*

Crée un bouton de réinitialisation du formulaire et non d'effacement systématique, comme on le croit souvent. Si les éléments du formulaire ont des attributs `value` qui définissent des valeurs par défaut, ce sont ces valeurs qui apparaissent au démarrage de la page et qui sont réaffichées si l'utilisateur clique sur le bouton `reset`.

Le contenu de l'attribut `value` du bouton d'effacement constitue le texte visible du bouton dans le formulaire.

En voici un exemple :

```
<input type ="reset" value="Effacer" />
```

L'aspect visuel de ce composant est illustré à la figure 6-1 (repère ⓫).

L'élément *<input type="file" />*

Permet le transfert de fichiers du poste client vers le serveur. Cet élément crée un champ de saisie de même aspect qu'un champ de texte et un bouton de sélection de fichier permettant à l'utilisateur de choisir le fichier à transférer.

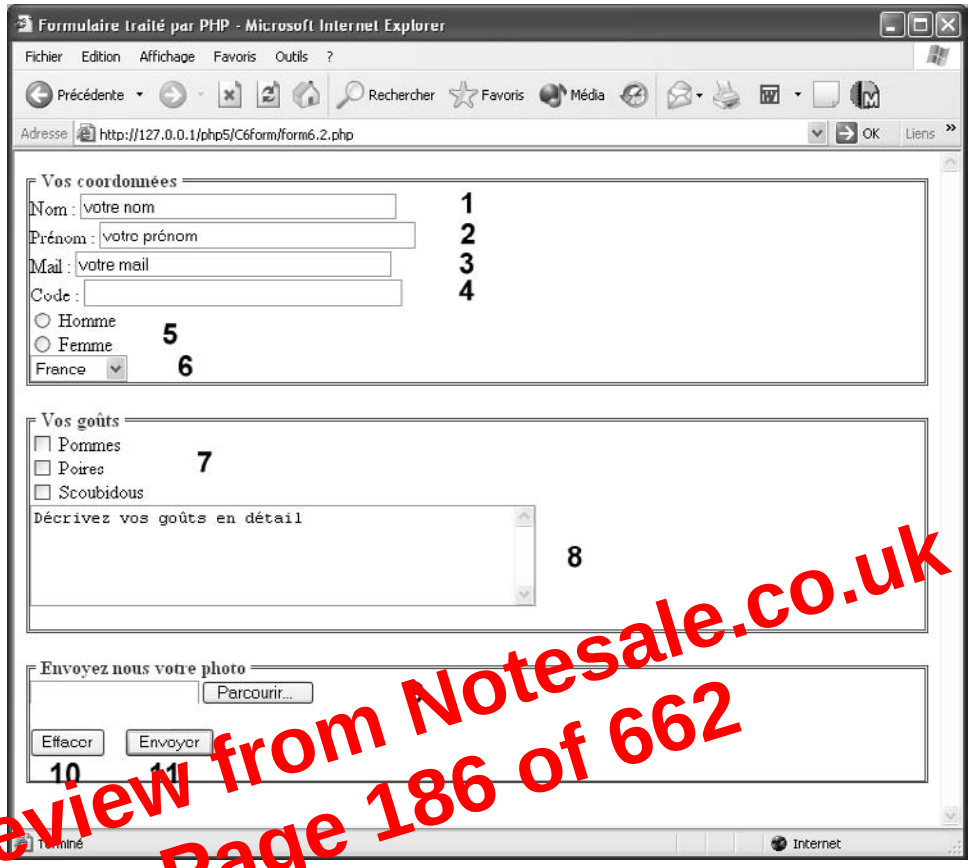


Figure 6-1

Formulaire type complet

> **Fichier XHTML ou PHP ?**
>
> Le fichier peut être enregistré sans problème avec l'extension `.html` ou `.htm`, car il ne contient aucun code PHP. C'est le fichier `cible.php`, donné comme valeur de l'attribut `action` de l'élément `<form>`, qui traite les données du formulaire coté serveur. Il n'est pas gênant de l'enregistrer avec l'extension `.php`, même s'il ne contient aucun code PHP.

Récupération des données du formulaire

Maintenant que vous savez créer de beaux formulaires, vous allez voir comment récupérer les données entrées par l'utilisateur dans les différents champs du formulaire.

Tout d'abord, que se passe-t-il lorsque l'utilisateur clique sur le bouton d'envoi ? Une requête HTTP est envoyée au serveur à destination du script désigné par l'attribut `action`

```
<td><input type="text" name="taux" /></td>
</tr>
<tr>
<td>Durée en année</td>
<td><input type="text" name="duree" /></td>
</tr>
<tr>
<td>Assurance</td>
<td>OUI : <input type="radio" name="assur" checked="checked" value="1" /> 
➥NON<input type="radio" name="assur" value="0" /></td>
</tr>
<tr>
<td><input type="reset" name="" value="Effacer"/></td>
<td><input type="submit" name="" value="Calculer"/></td>
</tr>
</table>
</fieldset>
</form>
</body>
</html>
```

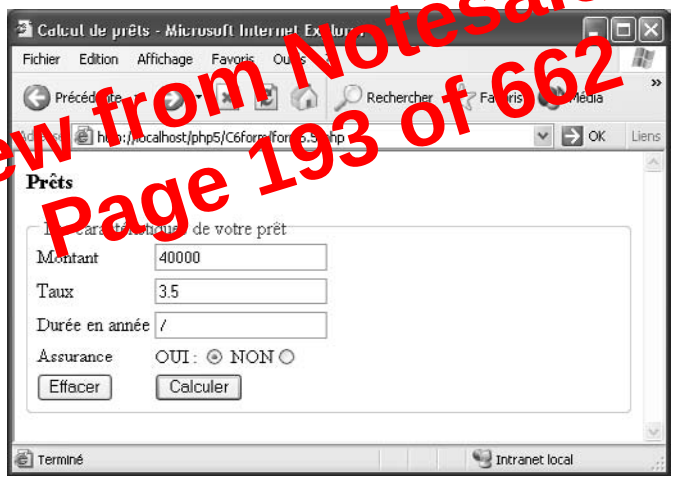



Figure 6-4

Page principale de saisie des données

Page de traitement des données et d'affichage des résultats (fichier `form4.php`) :

```
<!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>
```

Exemple 6-7. Transfert simultané de plusieurs fichiers

```
<!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-8859-1" />
<title>Transfert de plusieurs fichiers</title>
</head>
<body>
<!-- Code HTML du formulaire -->
<form action="<?= $_SERVER['PHP_SELF'] ?>" method="post"
➥enctype="multipart/form-data" >
<fieldset>
<input type="hidden" name="MAX_FILE_SIZE" value="100000" />
<legend><b>Transferts de plusieurs fichiers</b></legend>
<table>
<tbody>
<tr>
<th>Fichier 1</th>
<td> <input type="file" name="fich[]" accept="image/gif" size="50"/></td> ←1
</tr>
<tr>
<th>Fichier 2</th>
<td> <input type="file" name="fich[]" accept="image/gif" size="50"/> ←2
</td>
</tr>
<tr>
<th></th>
<td> <input type="submit" value="Envoi" /></td>
</tr>
</tbody>
</table>
</fieldset>
</form>
<!-- Code PHP -->
<?php
if(!empty($_FILES))
{
  echo "Taille maximale autorisée :",$_POST["MAX_FILE_SIZE"]," octets<hr / >";
  foreach($_FILES["fich"] as $cle => $valeur) ←3
  {
    echo "Clé : $cle <br />";
    foreach($valeur as $key=>$val) ←4
    {
      echo " Fichier : $key  valeur : $val <br />";
    }
  }
  //Déplacement et renommage des fichiers
  $result1=move_uploaded_file($_FILES["fich"]["tmp_name"][0],"image1.gif"); ←5
```

Avec le serveur local Wampserver 2.0, vous obtenez la liste suivante :

apache2handler	bcmath	calendar	com_dotnet	ctype
date	dom	filter	ftp	gd
hash	iconv	json	libxml	mbstring
mysql	mysqli	odbc	pcre	PDO
pdo_mysql	Reflection	session	SimpleXML	SPL
SQLite	standard	tokenizer	wddx	xmlxmlreader
xmlwriter	zlib			

et sur le serveur PHP 5 de l'hébergeur OVH :

bcmath	calendar	cgi	ctype	curl
date	dba	dbase	dom	exif
filter	ftp	gd	gettext	gmp
hash		iconv	imap	json
libxml	mbstring	mcrypt	mhash	ming
mysql	mysqli	openssl	pcre	pdf
PDO pdo_mysql	pdo_sqlite	pgsql	posix	pspell
Reflection	session	SimpleXML	soap	sockets
SPL	SQLite	standard	sysvsem	sysvshm
tokenizer	wddx	xml	xmlreader	xmlrpc
xmlwriter	xsl	zip	zlib	

Soit pas moins de 54 extensions !

Vous constatez qu'il existe peu d'extensions communes entre le serveur local installé avec Wampserver 2.0 et un serveur distant susceptible d'héberger votre site.

Pour chaque module, il est possible de lister l'ensemble des fonctions disponibles. Cela n'est toutefois guère utile. En effet, même si une fonction fait partie d'un module, votre hébergeur peut très bien l'avoir désactivée, notamment pour des raisons de sécurité ou d'abus d'occupation du serveur. La fonction `mail()` d'envoi de courrier, par exemple, est souvent désactivée chez les hébergeurs, en particulier les gratuits.

Pour obtenir la liste des fonctions d'un module, vous disposez de la fonction suivante :

```
array get_extensions_funcs("nom_module")
```

Cette fonction retourne un tableau indicé, dont les valeurs sont les noms des fonctions de chaque module.

Par curiosité, exécutez le petit script de l'exemple 7-1 sur votre serveur distant pour vérifier la liste, classée par ordre alphabétique, des modules et des fonctions que vous pouvez utiliser. Quoi de plus frustrant, en effet, que d'écrire de superbes scripts alors que les fonctions correspondantes sont disponibles sur le serveur local mais pas sur le serveur distant.

Exemple 7-1. Liste des modules et des fonctions affichées sur le serveur distant

```
<?php
//Tableau contenant le nom des extensions
```

affiche 1500, montrant que la valeur de la variable `$prix`, passée par valeur, n'a pas été modifiée.

L'appel de la fonction par le code suivant :

```
echo hausse(&$prix,12)
```

affiche le même résultat, mais il est nécessaire de vérifier ensuite que la variable `$prix` vaut également 1680, le deuxième appel étant fait par référence. Cette deuxième solution est préférable lorsque vous souhaitez créer des bibliothèques de fonctions dont le code ne soit pas directement visible dans le script. Ainsi, le passage par référence ne risque pas d'être involontaire.

Cette possibilité est toutefois considérée aujourd'hui comme obsolète, et le passage par référence doit faire partie de la déclaration de la fonction. En conséquence, il est important de bien documenter vos fonctions pour ne pas oublier quels sont les arguments passés par référence et ceux qui ne le sont pas.

Cas particuliers

Dans cette section, nous allons examiner divers cas particuliers qui peuvent se révéler utiles. À savoir les fonctions dynamiques, les fonctions conditionnelles, les fonctions définies à l'intérieur d'une autre fonction et les fonctions récursives.

Les fonctions dynamiques

Les fonctions que vous avez écrites jusqu'à présent ont un nom fixe et bien défini dans les scripts et la lecture du script permet de connaître immédiatement la fonction appelée. PHP offre la possibilité de travailler avec des noms de fonctions dynamiques, qui peuvent être variables et dépendants de l'utilisateur du site ou de l'interrogation d'une base de données.

Pour réaliser cette opération, il faut que le nom de la fonction — sans les parenthèses ni les paramètres — soit contenu dans une variable de type chaîne de caractères. Pour utiliser la fonction il n'y a plus ensuite qu'à faire suivre cette variable de parenthèses et de ses paramètres éventuels.

Le code suivant :

```
$ch ="phpinfo";
$ch();
```

équivaut au code :

```
phpinfo()
```

qui appelle directement la fonction `phpinfo()`.

De même, le code suivant :

```
$ch = "date";
echo $ch(" D d/M/Y H:i:s ");
```

Exemple 7-16. Fonction incluse dans une autre

```
<?php
//enfant();//ERREUR FATALE ←❶
function parent() ←❷
{
   echo "Bonjour les enfants!  <br />";
   function enfant() ←❸
  {
   echo "Bonsoir  papa !<br />";
  }
}
parent(); ←❹
enfant(); ←❺
enfant(); ←❻
?>
```

Le script affiche les résultats suivants :

```
Bonjour les enfants!
Bonsoir papa !
Bonsoir papa !
```


Les fonction récursives

Une fonction est dite récursive si, à l'intérieur de son corps, elle s'appelle elle-même avec une valeur de paramètre différent (sinon elle boucle). Chaque appel constitue un niveau de récursivité. L'exemple le plus classique est celui de la fonction qui retourne la factorielle d'un nombre entier n (notée n! que nous avons déjà calculée d'une manière différente à l'exemple 7-7). Pour calculer n!, une fonction récursive calcule $n \times (n - 1)!$, ce qui implique un nouvel appel de la fonction factorielle et ainsi de suite jusqu'à calculer 1! (par définition 0!=1) puis on remonte jusqu'à n!.

Ce qui donne, par exemple, le code suivant :

```
<?php
function facto($n)
{
  if ($n==1) return 1;
  else {return $n*facto($n-1);}
}
echo "factorielle = ",facto(150);
?>
```

Un grand autre classique de la récursivité est la programmation du jeu dit des tours de Hanoï. Imaginé par le mathématicien français Édouard Lucas, il consiste à déplacer des disques de diamètres croissants d'un piquet de « départ » à une piquet d'« arrivée » en passant par un piquet « intermédiaire » et ceci en un minimum de coups, sachant qu'on ne peut déplacer qu'un disque à la fois et que celui-ci ne peut être placé que sur un disque plus grand que lui ou sur un piquet vide. Au départ les disques sont empilés sur un des

décimales affichées pour le résultat. Il est possible de réutiliser la fonction `prod()` présentée dans ce chapitre pour calculer la factorielle *n*!.

Exercice 4

Écrivez une fonction dont le paramètre passé par référence est un tableau de chaînes de caractères et qui transforme chacun des éléments du tableau de manière que le premier caractère soit en majuscule et les autres en minuscules, quelle que soit la casse initiale des éléments, même si elle est mixte.

Exercice 5

À partir de la fonction sinus de PHP, écrivez une fonction qui donne le sinus d'un angle donné en radian, en degré ou en grade. Les paramètres sont la mesure de l'angle, et l'unité est symbolisée par une lettre. Le deuxième paramètre doit avoir une valeur par défaut correspondant aux radians.

Exercice 6

Créez une fonction de création de formulaires comprenant une zone de texte, une case d'option *(radio button)*, un bouton Submit et un bouton Reset. Choisissez comme paramètres les attributs des différents éléments XHTML en cause. Chaque appel de la fonction doit incorporer le code XHTML du formulaire à la page.

Exercice 7

Décomposez la fonction précédente en plusieurs fonctions, de façon à constituer un module complet de création de formulaire. Au total, il doit y avoir une fonction pour l'entête du formulaire, une pour le champ texte, une pour la case d'option, une pour les boutons Submit et Reset et une pour la fermeture du formulaire. Incorporez ces fonctions dans un script, et utilisez-les pour créer un formulaire contenant un nombre quelconque de champ de saisie de texte et de cases d'option.

Exercice 8

Programmez les coefficients du binôme (ou triangle de Pascal). Pour mémoire, il s'agit de la suite suivante :

1

1 2 1

1 3 3 1

1 4 6 4 1

etc.

La première colonne et la diagonale valent toujours 1 et chaque autre élément est égal à la somme de celui qui est au-dessus et de celui qui se trouve sur la diagonale gauche (par exemple 3=2+1 ou bien 6=3+3).

La fonction `time()`, que vous utiliserez souvent par la suite, retourne le timestamp de l'instant présent. Cette valeur n'est pas affichée au visiteur du site. Elle sert seulement d'intermédiaire sous-jacent pour calculer des durées et déterminer des dates futures ou passées. Le timestamp est alors passé à d'autres fonctions, qui réalisent l'affichage en clair de la date désirée. Un timestamp permet par ailleurs de stocker plus facilement une date à un seul nombre et constitue le moyen le plus sûr pour conserver une date dans une base de données.

L'exemple 8-1 montre la manière d'utiliser cette fonction pour afficher le timestamp en cours directement avec la fonction `time()` (repère ❶) ainsi que celui de dates futures (repère ❷) et passées (repère ❸). Il suffit pour cela d'ajouter ou d'enlever le nombre de secondes désiré. Pour calculer le nombre d'heures ou de jours correspondant au timestamp de l'instant en cours, il suffit de diviser la valeur donnée par la fonction `time()` par 3 600 pour le nombre d'heures (repère ❹) et par 3 600 puis 24 pour le nombre de jours (repère ❺).

Décalage horaire

Le timestamp retourné par la fonction `time()` est bien sûr celui qui est calculé côté serveur. Il n'est pas forcément identique à celui du poste client. Il faut donc tenir compte du décalage horaire éventuel.

Exemple 8-1. La fonction *time()*

```
<?php
echo "A cet instant précis le timestamp est : ", time(), "<br />"; ←❶
echo "Dans 23 jours le timestamp sera : ", time()+23*24*3600,
➥"<br />"; ←❷
echo "Il y a 12 jours le timestamp était : ", time()-12*24*3600,"<br />"; ←❸
echo"Nombre d'heures depuis le 1/1/1970 = ",round(time()/ 3600),"<br />"; ←❹
echo"Nombre de jours depuis le 1/1/1970 = ",round(time()/3600/ 24),"<br />"; ←❺
?>
```

Le résultat obtenu à l'instant du test est le suivant :

```
A cet instant précis le timestamp est : 1224522240
Dans 23 jours le timestamp sera : 1226509440
Il y a 12 jours le timestamp était : 1223485440
Nombre d'heures depuis le 1/1/1970 = 340145
Nombre de jours depuis le 1/1/1970 = 14173
```

Les fonctions abordées dans les sections qui suivent vous permettront de trouver à quelle date précise ce test a été effectué.

La fonction `microtime()` fournit également le nombre de secondes et de microsecondes de l'instant en cours, mais en retournant non pas un nombre décimal mais une chaîne de caractères commençant par le nombre de microsecondes suivi du nombre de secondes. Cela est dû au manque de précision des nombres décimaux de type double, qui ne

```
//la fonction mktime()
$timepasse= mktime(12,5,30,5,30,1969); ←❶
$timeaujour = time();
$duree = $timeaujour-$timepasse; ←❷
echo "Entre le 30/05/1969 à 12h05m30s et maintenant il s'est écoulé",$duree,
➥" secondes <br />";
echo "Soit ",round($duree/3600), " heures <br />";
echo "Ou encore ",round($duree/3600/24)," jours <br />";
$timefutur = mktime(12,5,30,12,25,2008);
$noel = $timefutur-$timeaujour; ←❸
echo "Plus que ",$noel,  "secondes entre maintenant et Noël, soit ",
➥round($noel/3600/24)," jours, Patience! <br />"; ←❹
//la fonction gmmktime()
$timepassegmt = gmmktime(12,5,30,5,30,1969); ←❺
echo "Timestamp serveur pour le 30/5/1969= ",$timepasse,"<br />";
echo "Timestamp GMT pour le 30/5/1969= ",$timepassegmt,"<br />";
echo "Décalage horaire = ",$timepasse-$timepassegmt," secondes<br />"; ←❻
?>
</div>
</body>
</html>
```

L'exemple retourne le résultat suivant sur un serveur Linux :

```
Entre le 30/05/1969 à 12h05m30s et maintenant il s'est écoulé 1243146454 secondes
Soit 345318 heures
Ou encore 14388 jours
Plus que 5679146secondes entre maintenant et Noël, soit 66 jours, Patience!
Timestamp serveur pour le 30/5/1969= -18622470
Timestamp GMT pour le 30/5/1969= -18618870
Décalage horaire = -3600 secondes
```

Sur un serveur Windows, le même script affiche l'erreur suivante, qui confirme que les timestamps négatifs n'y sont pas admis :

```
Warning: mktime(): Windows does not support negative values for this function in c:\
eyrolles\php5\c8dates\date8.3.php on line 11
```

Vérifier une date

Dans un formulaire complété par un visiteur, il n'est pas rare que celui-ci indique une date, ne serait-ce que sa date de naissance. Même si une expression régulière peut vous permettre de vérifier si la saisie répond à un format imposé, par exemple JJ/MM/AAAA, elle ne peut vérifier si la date indiquée existe ou si le nombre des jours et celui des mois sont inversés.

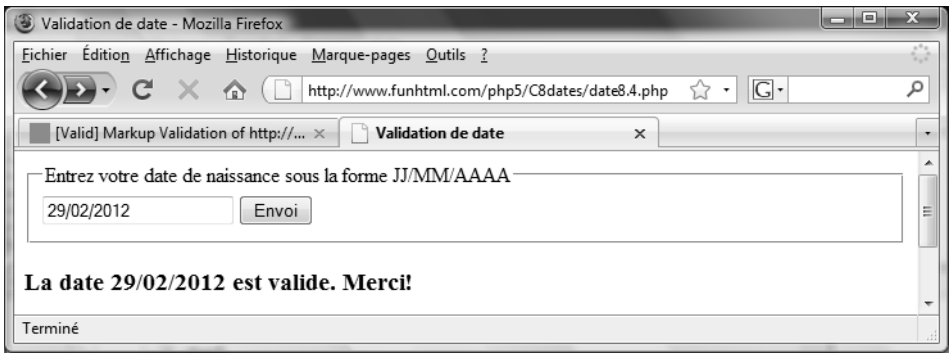


Figure 8-1

Le formulaire de vérification des dates

Afficher une date en clair

La fonction `date()` permet d'afficher une date selon des paramètres plus lisibles qu'un timestamp UNIX. Sa syntaxe est la suivante :

```
string date(string format_de_date,[int timestamp])
```

Elle retourne une chaîne contenant des informations de date dont la mise en forme est définie par des caractères spéciaux (voir leur signification au tableau 8-1). La date retournée correspond à celle du timestamp passé en deuxième paramètre ou, si ce dernier est omis, à celle de l'instant en cours.

Pour afficher un des caractères spéciaux du tableau 8-1 indépendamment de sa fonction de formatage, il faut le faire précéder d'un antislash. Par exemple, `\h` affiche le caractère « h » et non le nombre d'heure. Pour afficher les caractères « n » et « t », il faut écrire `\\n` et `\\t` car `\n` et `\t` sont employés pour le saut de ligne et la tabulation.

Par exemple, pour obtenir l'affichage :

```
Aujourd'hui Monday, 20 October 2008 il est 23:36:27
```

vous écrivez :

```
echo "Aujourd'hui ",date("l, d F Y \i\l  \e\s\\t H:i:s ");
```

La fonction `date()` tient compte de l'heure d'été.

L'exemple ci-dessous utilise la fonction `date()` pour des timestamps futurs (repère ❶) et passés (repère ❷) :

```
echo "Dans 40 jours nous serons le ",date("l, d F Y H:i:s",time()+40*3600*24); ←❶
echo "Il y 24 jours nous étions le ",date("l, d F Y H:i:s",time()-24*3600*24),
➥"<br />"; ←❷
```

L'exemple retourne le résultat suivant :

```
Dans 40 jours nous serons le Saturday, 29 November 2008 22:38:52
Il y 24 jours nous étions le Friday, 26 September 2008  23:38:52
```

Tableau 8-1 – Caractères de définition du format d'affichage

Caractère de définition	Définition et résultat affiché
y	L'année en deux chiffres (05 pour 2005)
Y	L'année en quatre chiffres (2005)
L	Affiche 1 si l'année est bissextile et 0 sinon.
m	Le mois en deux chiffres de 01 à 12
n	Le mois en un ou deux chiffres de 1 à 12
M	Le mois en trois lettres (en anglais)
F	Le mois en toutes lettres (en anglais)
t	Le nombre de jours du mois de 28 à 31
d	Le jour du mois en deux chiffres de 01 à 31
j	Le jour du mois en un chiffre de 1 à 31
D	Le jour de la semaine en trois lettres (en anglais)
l (petit L)	Le jour de la semaine en toutes lettres (en anglais)
w	Le jour de la semaine codé de 0 pour dimanche à 6 pour samedi
S	Affiche le suffixe anglais « th » ou « nd » après les chiffres du jour.
z	Le jour de l'année de 0 à 366
g	Les heures de 1à 12 (avec AM et PM)
h	Les heures de 01 à 12 (avec AM et PM)
G	Les heures de 0 à 23
H	Les heures sur deux chiffres de 00 à 23
a	Ajoute « am » pour le matin ou « pm » pour l'après-midi.
A	Ajoute « AM » pour le matin ou « PM » pour l'après-midi.
i	Les minutes en deux chiffres de 00 à 59
s	Les secondes en deux chiffres de 00 à 59
U	Affiche le timestamp UNIX.
Z	Donne le décalage horaire par rapport au temps GMT ou UTC en seconde, de –43 200 à 43 200.
T	Affiche la ville significative du fuseau horaire, par exemple « Paris, Madrid ».
I	Affiche 0 pendant l'heure d'hiver et 1 pendant l'heure d'été.
r	Affiche la date complète au format RFC 822, par exemple: « Sun, 13 Apr 2003 22:34:46 +0200 ».
B	Heure Internet Swatch : invention de la société Swatch selon laquelle 24 heures sont divisées en 1 000 éléments nommés « beats ». Par exemple, midi vaut 500 beats.

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.

Pour créer des objets représentant des actions boursières conformes au modèle de votre classe `action`, vous écrirez donc :

```
$action1 = new action();
$action2 = new action();
```

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 afin que chacun d'eux corresponde à une action boursière particulière. 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 à) sous la forme :

```
$nom_objet->prop;
```

ou encore :

```
$nom_objet->prop[n];
```

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

```
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()` (repère ❶). Vous créez ensuite une variable `$action1` représentant un objet de type `action` et définissez des valeurs pour les propriétés `nom` et `cours` (repères ❸ et ❹). Ces propriétés sont lues et utilisées pour créer un affichage (repère ❺). L'appel de la méthode de l'objet permet d'obtenir des informations sur l'ouverture des bourses (repère ❻). La fonction `var_dump()` permet d'afficher, à l'usage du programmeur uniquement, le nom, le type et la valeur de chaque propriété (repère ❼).

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

La figure 9-1 donne le résultat du script.

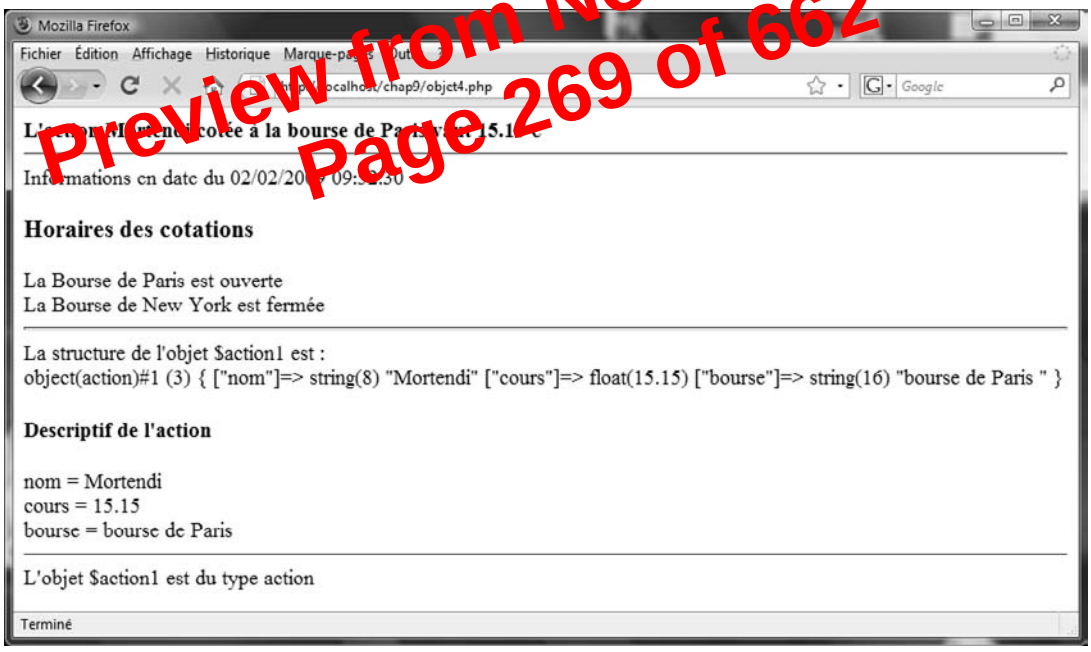


Figure 9-1

Affichage des propriétés d'un objet

Pour lire la propriété `$bourse` à l'extérieur de la classe, écrivez le code suivant :

```
info::$bourse (repère ❹).
```

L'appel des méthodes hors de tout contexte objet se fait de la même façon (repères ❺ et ❻).

Pour montrer le danger de l'utilisation d'une propriété statique dans un contexte objet, un objet de type `info` (repère ❼) est créé, puis une nouvelle valeur est affectée à sa propriété `bourse` (repère ❽). L'affichage de cette propriété montre que cette affectation est bien réalisée (repère ❾). En revanche, l'appel de la méthode de l'objet qui utilise cette propriété permet de constater que la propriété a toujours la valeur qui a été définie dans la classe (repères ❿). Le mot `static` prend alors tout son sens.

Pour pallier cet inconvénient, il faudrait ajouter à la classe une méthode spéciale qui modifierait la propriété `bourse` de la manière suivante :

```
public  function setbourse($val)
{
  info::$bourse=$val;
}
```

Exemple 9-8. Propriété et méthode statiques

```
<?php
class info
{
  //Propriété statique
  public static $bourse="Bourse de Paris";←❶
  //Méthodes statiques
  public  static function getheure()←❷
  {
    $heure=date("h : m : s");
    return $heure;
  }
  public static function afficheinfo()
  {
    $texte=info::$bourse.", il est ".self::getheure();←❸
      return $texte;
  }
}
echo info::$bourse,"<br />";←❹
echo info::getheure(),"<br />";←❺
echo info::afficheinfo(),"<hr />";←❻
//Création d'un objet info
$objet=new info();←❼
$objet->bourse="New York";←❽
echo "\$objet->bourse : ",$objet->bourse,"<hr />";←❾
echo "\$objet->getheure() : ",$objet->getheure(),"<br />";←❿
echo "\$objet->afficheinfo() : ",$objet->afficheinfo(),"<br />";←❿
?>
```

Le code suivant :

```
class triangleiso extends triangle
{
  function triangleрect()
  {
  //Instructions
  }
}
```

provoque l'apparition d'une erreur fatale et du message :

```
Fatal error: Cannot override final method triangle::triangleрect()
```

De même, pour interdire l'héritage d'une classe parente dans une classe enfant, faites précéder le mot-clé `class` du mot-clé `final`.

Si vous définissez la classe suivante :

```
final class triangle
{
  //Instructions
}
```

et si vous tentez de la dériver en créant la classe enfant suivante :

```
class triangleiso extends triangle
{
    //Instructions
}
```

vous créez également une erreur fatale accompagnée du message :

```
Fatal error: Class triangleiso may not inherit from final class (triangle)
```


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.

```
Affectation de la valeur QWERTY à la propriété code
code = QWERTY
La propriété code est définie
Effacement de la prop code
code =
```

Mémo des fonctions

`boolean class_exists (string $nomclasse)`

Détermine si la classe passée en paramètre existe ou non.

`string get_class(object $varobjet)`

Retourne le nom de la classe dont l'objet est une instance.

`array get_class_methods(string nom_classe)`
`array get_class_methods(object $varobjet)`

Retournent un tableau contenant tous les noms des méthodes de la classe ou de l'objet.

`array get_class_vars(string nom_classe)`

Retourne un tableau associatif dont les clés sont celles des propriétés de la classe et contenant toutes valeurs par défaut de ces propriétés.

`array get_declared_classes()`

Retourne un tableau de tous les noms des classes du noyau et des classes prédéfinies par PHP.

`array get_declared_interfaces()`

Retourne un tableau de tous les noms des interfaces du script.

`array get_object_vars (object $varobjet)`

Retourne un tableau de toutes les propriétés de l'objet.

`string get_parent_class(string nom_classe)`
`string get_parent_class(object $varobjet)`

Retourne le nom de la classe parente d'une classe ou d'un objet.

`bool method_exists (object $object, string $nommethode )`

Détermine si la méthode précisée en paramètre existe ou non.

`bool is_subclass_of (divers $object, string $nomclasse )`

Détermine si l'objet précisé a la classe `$nomclasse` comme parent.

Exercices

Exercice 1

Écrivez une classe représentant une ville. Elle doit avoir les propriétés `nom` et `département` et une méthode affichant « la ville X est dans le département Y ». Créez des objets `ville`, affectez leurs propriétés, et utilisez la méthode d'affichage.

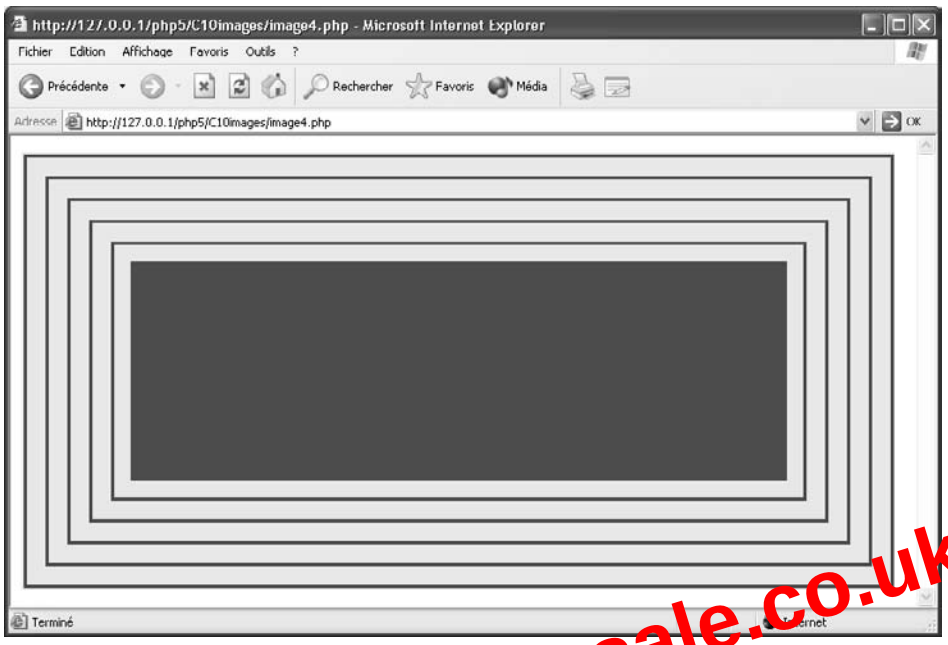


Figure 10-3

Tracé de rectangles

Contrairement au tracé des rectangles, vous n'énumérez pas les coordonnées des sommets. Celles-ci sont passées sous la forme d'un tableau `$tab`, dont les éléments sont une suite d'abscisses et d'ordonnées. Assurez-vous qu'il y ait un nombre pair de coordonnées. Le paramètre `N` indique le nombre de sommets, ce qui permet de refermer la figure sans donner à nouveau les coordonnées du premier point.

La variante suivante, dont la syntaxe est identique, permet de tracer des polygones pleins :

```
int imagefilledpolygon (resource $idimg, array $tab, int N, int $couleur)
```

L'exemple 10-5 définit un tableau de huit éléments représentant les coordonnées de quatre sommets (repère ❶) puis trace le quadrilatère correspondant (repère ❷). Les coordonnées peuvent être passées en définissant le tableau directement comme second paramètre de la fonction (repère ❸), ici pour tracer un triangle.

Exemple 10-5. Tracé de polygones

```
<?php
header ("Content-type: image/png");
$idimg=imagecreate(800,400);
$fond=imagecolorallocate($idimg,255,255,51);
$noir=imagecolorallocate($idimg,0,0,0);
$blanc=imagecolorallocate($idimg,255,255,255);
```

```
//Définition de l'épaisseur de trait de 3 pixels
imagesetthickness($idimg,3);
//Coordonnées du quadrilatère
$tab=array (100,50,500,100,750,300,50,350); ←❶
//Tracé du quadrilatère
imagepolygon($idimg,$tab,4,$noir); ←❷
//Tracé du triangle plein
imagefilledpolygon($idimg,array(150,80,500,150,250,310),3,$blanc); ←❸
imagepng($idimg,"polygon.png");
imagepng($idimg);
imagedestroy($idimg);
?>
```

Le résultat affiché est illustré à la figure 10-4.

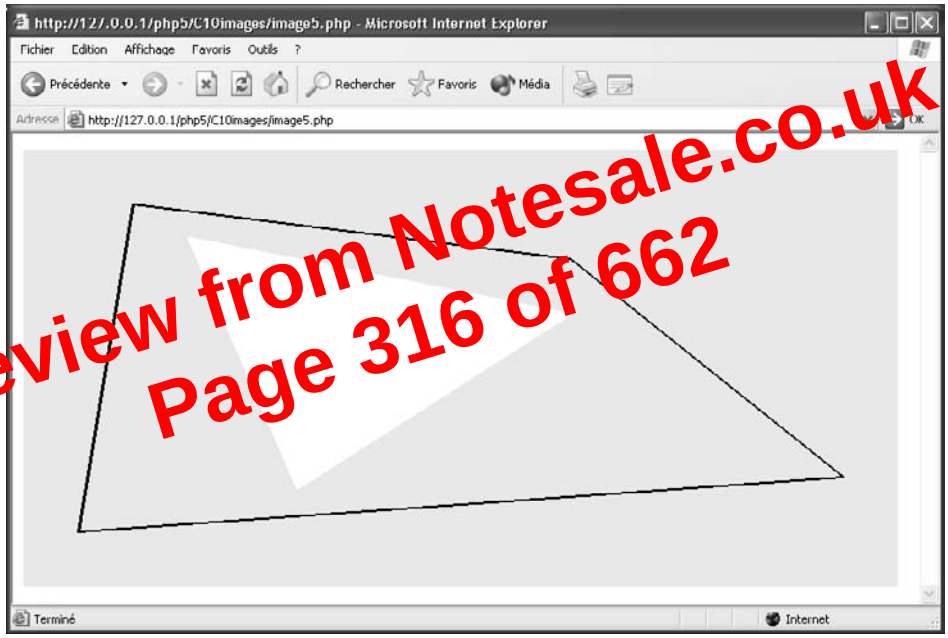


Figure 10-4
Tracé de polygones

Tracé d'arcs, de cercles et d'ellipses

Pour tracer des arcs de cercle ou des ellipses, vous disposez de la fonction `imagearc()`, dont la syntaxe est la suivante :

```
int imagearc (resource $idimg, int Xc, int Yc, int Larg, int Haut, int Ang1,
➥int Ang2, int $couleur)
```

```
//Tracé d'ellipses
imageellipse ($idimg,200,150,100,50,$noir); ←3
imagefilledellipse ($idimg,120,300,200,80,$rouge); ←4
imagepng($idimg,"cercle.png");
imagepng($idimg);
imagedestroy($idimg);
?>
```

La figure 10-5 illustre les résultats obtenus.

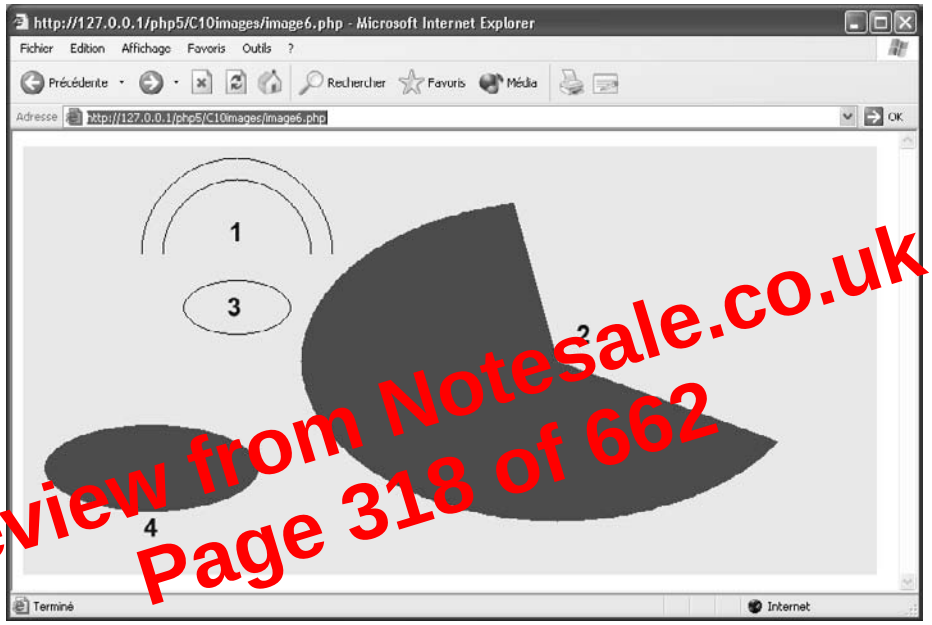


Figure 10-5
Tracé de cercles et d'ellipses

Remplissage de surfaces

Vous avez vu comment créer des surfaces pleines quand il s'agit de rectangles, de polygones ou de cercles. Quand il s'agit de remplir des formes quelconques limitées par un contour fermé avec une couleur précise, utilisez la fonction suivante :

```
int imagefill (resource $idimg, int x, int y, int $couleur)
```

Les coordonnées `(x,y)` passées en paramètre sont celles d'un point quelconque de l'intérieur de la surface à remplir.

Vous avez également la possibilité de remplir une surface à l'aide d'un motif constitué d'une image existante, qui se répète autant de fois que nécessaire. Cette fonction est très utile pour réaliser une image de fond dans un document XHTML.

- lecture des données ;
- recueil d'informations sur les fichiers ;
- modification des fichiers.

Création, ouverture et fermeture d'un fichier

Il peut être utile dans certaines circonstances de créer un fichier vide de tout contenu mais ayant une existence physique sur le serveur. Si vous envisagez d'écrire des données correspondant à plusieurs connexions différentes dans un même fichier, par exemple, il vous faut commencer par créer un fichier vide, non sans contrôler qu'il n'existe pas encore. Vous y ajoutez ensuite les données en provenance de l'utilisateur.

Pour faire cela, vous pouvez détourner la fonction `touch()` de sa vocation initiale, qui est de définir la date de la dernière modification d'un fichier. Si le fichier n'existe pas, cette fonction le crée et lui affecte la date passée en deuxième argument sous la forme d'un timestamp UNIX comme date de dernière modification.

La syntaxe générale de la fonction `touch()` est la suivante :

```
boolean touch(string "nom_fichier"[,integer timestamp])
```

Le nom du fichier vient en premier paramètre, et le timestamp de la date de création en second.

En écrivant, par exemple :

```
if(!file_exists("monfich.txt"))
{
touch("monfich.txt",time());
}
```

vous créez le fichier `monfich.txt` s'il n'existe pas encore, avec pour date de dernière modification l'instant en cours donné par la fonction `time()`. Le fichier est vide mais est disponible en écriture, comme vous le verrez à la section suivante.

La fonction `file_exists()` utilisée dans cet exemple retourne `TRUE` si le fichier existe déjà et `FALSE` dans le cas contraire.

Ouverture du fichier

Avant de réaliser des opérations de lecture ou d'écriture sur un fichier, vous devez l'ouvrir explicitement. Vous disposez pour cela de la fonction `fopen()`, qui nécessite plusieurs paramètres. Le choix de ces paramètres conditionne le mode d'accès au fichier et ce que vous allez pouvoir en faire, à savoir le lire uniquement, y écrire uniquement ou le lire et y écrire dans le même script.

La syntaxe de la fonction `fopen()` est la suivante :

```
resource fopen(string $nom, string mode, [boolean path])
```

Identifiant de fichier

Depuis PHP 4, la fonction `fopen()` retourne un identifiant de fichier de type `resource` (il était de type `integer` dans PHP 3), qui doit être utilisé comme premier paramètre de la plupart des fonctions de manipulation des fichiers. Vous devez donc impérativement récupérer cette valeur dans une variable pour pouvoir l'utiliser dans les autres opérations d'accès au même fichier. Nous noterons systématiquement cette variable `$id_file`.

En affichant la valeur de cet identifiant, vous constatez qu'il est de la forme `Resource id#n`, dans laquelle `n` est un entier incrémenté de 1 à chaque ouverture de fichier par le même script (la première valeur est toujours 1). Cet affichage n'a pour but que de satisfaire une curiosité car il ne sera jamais effectué dans un script. En cas d'échec de l'ouverture du fichier, la fonction retourne la valeur `FALSE`, ce qui peut permettre l'affichage d'un message d'erreur.

Pour ouvrir en écriture seule un fichier existant dans le même dossier que le script en cours et récupérer son identifiant, vous écrivez, par exemple :

```
$id_file = fopen("monfichier.txt","a");
if(!$id_file) echo "Erreur d'accès au fichier";
```

Rien n'empêche d'ouvrir plusieurs fichiers simultanément et de manipuler ainsi plusieurs identifiants. Vous pouvez ensuite fermer un fichier sans fermer les autres.

Fichier temporaire

Vous pouvez aussi créer un fichier temporaire sur le serveur à l'aide de la fonction :

```
resource tmpfile()
```

qui retourne également un identifiant de fichier.

Ce fichier est utilisé pour stocker des informations qui ne seront conservées que pendant la durée de la session ouverte par un client ou jusqu'à ce que le fichier soit explicitement fermé au moyen de la fonction `fclose()`.

Fermeture du fichier

Après avoir ouvert un fichier pour y effectuer des opérations de lecture ou d'écriture, il vous faut impérativement le fermer pour éviter tous les problèmes qui pourraient résulter d'une tentative d'ouverture du même fichier de la part d'un autre script ou du même script ouvert par une autre connexion.

L'opération de fermeture est réalisée à l'aide de la fonction suivante :

```
boolean fclose($id_file)
```

Cette fonction prend comme unique paramètre l'identifiant de fichier `$id_file` retourné par la fonction `fopen()`. Elle retourne la valeur booléenne `TRUE` si l'opération de fermeture s'est bien effectuée, et `FALSE` dans le cas contraire. Vous pouvez donc réaliser un test systématique pour vérifier le bon déroulement de l'opération de fermeture.

Compte tenu des fonctions que vous venez de voir, le schéma général d'utilisation d'un fichier conduit à écrire systématiquement le code suivant :

```
$id_file = fopen("monfichier.txt","mode");
flock($id_file,LOCK_SH ou LOCK_EX);
//ou encore
//flock($id_file,1 ou 2);
//opérations de lecture et/ou d'écriture
flock($id_file(LOCK_UN);
//ou encore
//flock($id_file,3);
fclose($id_file)
```

Vous devrez aussi ajouter par la suite les fonctions de lecture et d'écriture des fichiers.

Écriture dans un fichier

Une fois qu'un fichier est ouvert avec `fopen()`, vous pouvez procéder aux opérations d'écriture ou de lecture de son contenu.

Pour écrire dans un fichier avec l'un des modes définis dans la fonction `fopen()`, vous avez le choix entre plusieurs méthodes et donc entre plusieurs fonctions spécialisées de PHP. Le choix de ces fonctions s'effectue en fonction du type d'information à écrire.

Conserver une information

Les fonctions `fwrite()` et `fputs()`, alias l'une de l'autre, ont la syntaxe suivante :

```
integer fwrite(ressource $id_file,string "chaine" [,int N])
integer fputs(ressource $id_file, string "chaine" [,int N])
```

Elles écrivent toutes deux le texte contenu dans `"chaine"` dans le fichier identifié par la variable `$id_file`. Lorsque le paramètre `N` est précisé, comme ici, seuls les `N` premiers caractères de la chaîne sont écrits dans le fichier.

Votre premier exemple d'utilisation des fichiers est un compteur de visites très simple enregistrant le nombre cumulé de visites dans un fichier nommé `compteur.txt`. À chaque connexion, le script vérifie si ce fichier existe déjà (repère ❶).

S'il existe, le script l'ouvre en lecture (repère ❷) et le verrouille en écriture (repère ❸). Il lit ensuite dans la variable `$nb` la dernière valeur enregistrée à l'aide de la fonction `fread()` (repère ❹, que nous définirons plus avant dans les sections suivantes) l'incrémente d'une unité (repère ❺) et ferme le fichier (repère ❻). Vous ouvrez ensuite le fichier en écriture avec le mode `"w"`, ce qui efface l'ancienne valeur (repère ❼), et y écrivez la nouvelle valeur de `$n` (repère ❽). Vous libérez ensuite le verrou et fermez le fichier (repères ❾ et ❿).

Si le fichier n'existe pas, il est créé et ouvert en écriture (repère ⓫). Vous y enregistrez la valeur 1 ou davantage, si vous voulez faire croire que le site est très prisé. Ici `$nb` est

```
}
fclose($id_file);
echo "</tbody></table> ";
?>
</body>
</html>
```

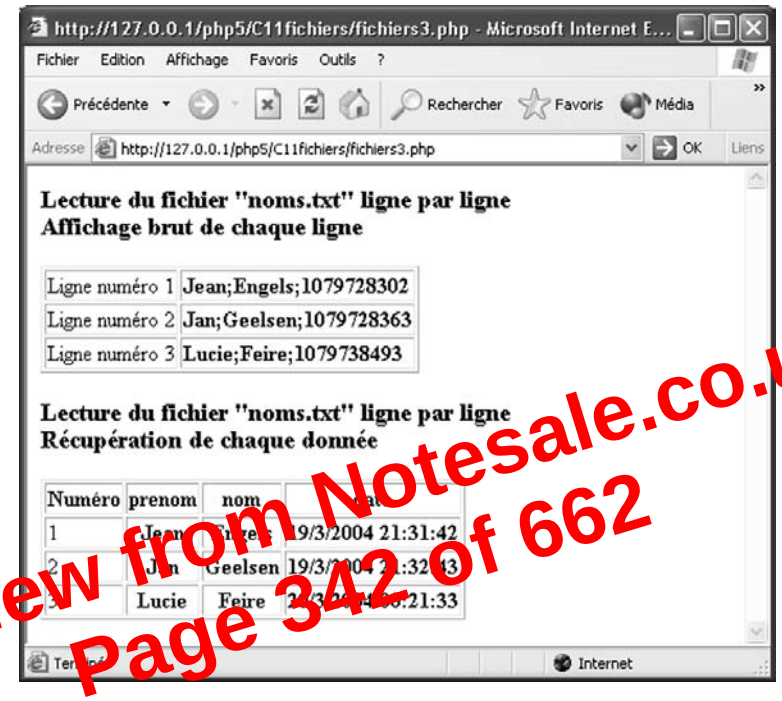


Figure 11-4

Lecture des lignes brutes et des données

Lire un nombre de caractères donné

La fonction `fread()` a pour syntaxe :

```
string fread(resource $id_file,integer nb_octets)
```

Cette fonction lit également le fichier depuis son début et retourne à chaque appel une chaîne de caractères contenant exactement le nombre de caractères précisé dans le second paramètre, sauf si la fin du fichier est atteinte ou si le caractère `"\n"` est rencontré. Son utilisation est donc adaptée à des fichiers dans lesquels vous avez préalablement enregistré des données de longueur fixe par paquets égaux.

Pour illustrer l'usage de cette fonction, vous allez créer un script permettant d'implanter sur un site un système de vote enregistrant les choix des visiteurs effectués grâce à un formulaire de saisie. Chaque proposition de vote pour un footballeur est faite à l'aide

et enregistre les données dans le tableau `$tab`. L'affichage se fait sous forme de tableau XHTML, comme l'illustre la figure 11-7. Dans ce tableau, chaque adresse e-mail est contenue dans un lien `mailto:`, ce qui permet à un visiteur de répondre à un autre (repère ❼).

Exemple 11-5. Livre d'or utilisant des données formatées

```
<!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-8859-1" />
<title>Le livre est d'or </title>
</head>
<body style="background-color: #ffcc00;">
<form action="<?php echo $PHP_SELF ?>" method="post" >
<fieldset>
<legend><b>Donnez votre avis sur PHP 5 ! </b></legend>
<label>Nom :  </label><input type="text" name="nom" /> <br />
<label>Mail :  </label><input type="text" name="mail" />  <br />
<label>Vos commentaires sur le site</label><br />
<textarea  name="comment" rows="4" cols="50">Ici </textarea>  <br />
<input type="submit" value="Envoyer " name="envoi" />
<input type="submit" value="Afficher les avis" name="affiche" />
</fieldset>
</form>
<?php
$date= time();// ←❶
//ENREGISTREMENT
if(isset($_POST['envoi']))
{
  if(isset($_POST['nom']) && isset($_POST['mail']) && isset($_POST['comment']))
  {
    echo "<h2>",$_POST['nom']," merci de votre avis </h2> ";
    if(file_exists("livre.txt") )
    {
      if($id_file=fopen("livre.txt","a"))
      {
      flock($id_file,2);
         fwrite($id_file,$_POST['nom'].":".$_POST['mail'].":".$date.":".
         ➥$_POST['comment']."\n"); ←❷
      flock($id_file,3);
      fclose($id_file);
      }
      else
      { echo "fichier inaccessible";
      }
    }
    else
    {
    $id_file=fopen("livre.txt","w"); ←❸
```

```
resource fopen(string nom_fichier, string mode, int chemin)
```

Ouvre le fichier dont vous précisez le nom et le mode d'accès choisi parmi les valeurs suivantes :
"r" : accès en lecture seule, pointeur au début du fichier.
"r+" : accès en lecture et écriture, pointeur au début du fichier.
"w" : accès en écriture seule, pointeur au début du fichier. Efface le contenu du fichier et le crée s'il n'existe pas.
"w+" : accès en lecture et écriture, pointeur au début du fichier. Efface le contenu du fichier et le crée s'il n'existe pas.
"a" : accès en lecture seule, pointeur en fin de fichier. Crée le fichier s'il n'existe pas.
"a+" : accès en lecture et écriture, pointeur en fin de fichier. Crée le fichier s'il n'existe pas.
La fonction retourne un identifiant de fichier qui est utilisé comme paramètre par un grand nombre de fonctions de lecture/écriture.
Le paramètre `chemin` permet d'élargir la recherche du fichier aux sous-dossiers s'il vaut 1.

```
int fpassthru(resource $id_file)
```

Lit le contenu du fichier situé après le pointeur et envoie le contenu vers la sortie standard.

```
string fread(resource $id_file, int nb)
```

Lit un nombre `nb` de caractères dans le fichier identifié par `$id_file`.

```
int fseek(resource $id_file, int nb)
```

Déplace le pointeur de fichier à la position `nb`.

```
int ftell(resource $id_file)
```

Retourne la position en cours du pointeur de fichier.

```
int fwrite(resource $id_file, string texte, int nb)
```

Écrit le nombre maximal `nb` de caractères de la chaîne texte dans le fichier identifié.

```
boolean is_file(string nom_fichier)
```

Retourne `TRUE` si le fichier existe et `FALSE` dans le cas contraire.

```
boolean is_readable(string nom_fichier)
```

Retourne `TRUE` si le fichier est accessible en lecture et `FALSE` dans le cas contraire.

```
boolean is_uploaded_file(string nom_fichier)
```

Retourne `TRUE` si le fichier provient d'un téléchargement par la méthode `POST` et `FALSE` dans le cas contraire.

```
boolean is_writable(string nom_fichier)
```

Retourne `TRUE` si le fichier est accessible en écriture et `FALSE` dans le cas contraire.

```
int readfile(string nom_fichier, boolean chemin)
```

Lit la totalité du fichier et l'envoie vers la sortie standard.

```
string realpath(string nom_fichier)
```

Retourne le chemin d'accès complet du fichier.

```
boolean rename(string nom_ancien, string nom_nouveau)
```

Renomme le fichier `nom_ancien` en `nom_nouveau` et retourne `TRUE` si l'opération s'est bien exécutée.

```
boolean rewind(resource $id_file)
```

Replace le pointeur de fichier au début.

12

Cookies, sessions et e-mails

Ce chapitre aborde les différentes méthodes qui permettent de conserver des informations pour améliorer le service rendu par un site. Il peut s'agir de conserver des choix fait par un visiteur entre deux visites pour adapter le contenu de la page d'accueil aux besoins de ce dernier. C'est ce que permettent les cookies. Il est également possible de conserver des informations saisies dans une page et de les rendre accessibles à toutes les autres pages d'un même site. Le mécanisme de session autorise ce type d'action, qui est à la base de la gestion de panier utilisée dans tous les sites de commerce en ligne.

Toujours dans le but d'améliorer le service rendu aux visiteurs ou aux clients d'un site, la fin du chapitre traite de l'envoi d'e-mails à partir du serveur du site vers le poste client ou tout autre destinataire, comme le font, par exemple, les sites de vente en ligne pour envoyer des confirmations de commande.

Les cookies

Les cookies sont de petits fichiers qui peuvent être écrits par un script PHP ou par d'autres langages, tel JavaScript, sur l'ordinateur du visiteur. À l'exception du piratage, c'est le seul cas où un site peut intervenir sur le disque dur d'un utilisateur.

Lors de sa création par Netscape, cette possibilité a effrayé plus d'un internaute, bien qu'elle ne présentait pas de danger réel. Chaque utilisateur garde de surcroît la possibilité de désactiver l'écriture des cookies en paramétrant son navigateur, mais certains services en ligne ne fonctionnent que si les cookies sont activés sur le poste client. Il peut aussi les effacer à sa guise puisqu'ils se trouvent sur son ordinateur.

Par défaut, Internet Explorer ou Firefox écrivent les cookies sans autorisation de l'utilisateur, tandis que Netscape (plus guère utilisé il est vrai) demandait l'accord du visiteur

pour chaque cookie à écrire. Vous n'avez donc jamais la certitude de pouvoir écrire vos cookies pour chaque visiteur.

Les cookies font l'objet de limites d'emploi. Un site donné ne peut écrire que 20 cookies sur un même poste client. Chacun d'eux ne doit pas dépasser 4 Ko, ce qui empêche le stockage d'information de taille importante. Sauf spécification contraire, un cookie n'est accessible que par le site qui l'a écrit. L'utilisation des cookies est donc généralement limitée au stockage d'information de petite taille, comme le nom, le code d'accès, l'adresse ou les préférences de l'utilisateur. L'usage courant est de stocker les coordonnées qu'un visiteur a saisies dans un formulaire et de les lire lors d'une prochaine connexion pour remplir automatiquement le même formulaire, permettant ainsi aux visiteurs de gagner du temps. Les cookies sont aussi employés par le mécanisme de session, que nous allons aborder par la suite. Il va de soi que, de par leur mode de stockage, les cookies ne sont pas récupérables si l'utilisateur se reconnecte à partir d'un poste différent, contrairement à ce qu'il est possible de faire si les informations sont inscrites dans une base de données.

Écriture des cookies

Pour écrire un cookie, comme pour envoyer des en-têtes au moyen de la fonction `header()`, il est impératif qu'aucun contenu XHTML n'ait été envoyé au poste client avant l'écriture du cookie. Autrement dit, aucune instruction PHP d'affichage, ne serait-ce que pour afficher un seul caractère, ne doit figurer dans le script avant la fonction qui va créer le cookie.

Pour écrire un cookie, vous utilisez la fonction `setcookie()`, dont la syntaxe est la suivante :

```
boolean setcookie(string nom_cookie,[string valeur, integer datefin,
➥string chemin, string domaine, integer securite] )
```

Les paramètres de la fonction `setcookie()` sont les suivants :

- `nom_cookie` est une chaîne définissant le nom du cookie. Ce nom obéissant aux mêmes règles de nommage que les variables, il faut éviter les caractères spéciaux. Ce nom sert à identifier le cookie pour les opérations de lecture de leur contenu.
- `valeur` contient la valeur associée au cookie. Il s'agit d'une chaîne de caractères, même pour un nombre. Il y a donc lieu d'effectuer au besoin un transtypage (voir le chapitre 2) pour effectuer des calculs avec cette valeur.
- `datefin` est un entier qui permet de stocker un timestamp UNIX exprimé en seconde (voir le chapitre 8) définissant la date à partir de laquelle le cookie n'est plus utilisable. Si ce paramètre est omis, le cookie n'est valable que pendant le temps de connexion du visiteur sur le site. Pour définir cette date, vous utilisez le plus souvent la fonction `time()`, qui donne le timestamp en cours, auquel vous ajoutez la durée désirée par un nombre de seconde. Pour une durée de validité de vingt-quatre heures, par exemple, vous écrivez `time() +86400`.

- `chemin` définit dans une chaîne le chemin d'accès aux dossiers qui contiennent les scripts autorisés à accéder au cookie. Les scripts contenus dans les sous-dossiers éventuels de ce dossier ont également accès au cookie. Si la valeur est `/`, le cookie est lisible sur l'ensemble du domaine `domaine`. Si la valeur est `/repertoire/`, le cookie est uniquement lisible dans le répertoire `/repertoire/` ainsi que tous ses sous-répertoires (`/repertoire/sousrep/` par exemple) du domaine `domaine`. La valeur par défaut est le répertoire qui contient le script ayant créé le cookie.
- `domaine` définit le nom entier du domaine à partir duquel vous pouvez accéder au cookie. Vous écrivez, par exemple, `www.mondomaine.com`, et non `mondomaine.com` seulement. Lorsque ce nom de domaine est le même que celui qui a créé le cookie, ce qui est le cas le plus fréquent, vous pouvez omettre ce paramètre.
- `securite` est une valeur de type `boolean` qui vaut `TRUE` (ou la valeur 1) si le cookie doit être transmis par une connexion sécurisée (avec une adresse du type `https://www.mondomaine.com`) et `FALSE` (ou la valeur 0) dans le cas contraire, qui est la valeur par défaut.

La fonction `setcookie()` renvoie une valeur booléenne `TRUE` qui permet de contrôler si l'écriture du cookie a eu lieu et `FALSE` en cas de problème (si le navigateur client refuse les cookies, par exemple).

L'exemple 12-1 écrit plusieurs cookies en utilisant les différents paramètres possibles.

Exemple 12-1. Écriture de cookies avec différents paramètres

```
<?php
//cookie valable uniquement pour la session
setcookie("prenom","Jan"); ←❶
//cookie valable 24 heures
setcookie("nom","Dessen",time()+86400); ←❷
//Ce cookie utilise tous les paramètres
setcookie("CB","5612 1234 5678 1234",time()+86400,"/client/paiement/",
➥"www.funxhtml.com",TRUE); ←❸
?>
```

Le premier appel de `setcookie()` ne définit que le nom et la valeur du cookie. Ce dernier n'étant valable que pendant la durée de la session, cette valeur n'est récupérable que pour une autre page du même site et non pour une prochaine connexion (repère ❶). Le deuxième appel crée un cookie valable vingt-quatre heures, soit 86 400 secondes (repère ❷). Le troisième appel crée un cookie utilisant tous les paramètres possibles (repère ❸).

Pour effacer le contenu d'un cookie, il suffit d'utiliser la fonction `setcookie()` en n'utilisant que le paramètre `nom_cookie` sans lui affecter de valeur.

La fonction suivante :

```
setcookie("nom")
```

efface la valeur précédente donnée au cookie nommé `"nom"`.

Dans le code de l'exemple, la durée de vie des cookies est fixée à 60 secondes, ce qui n'est guère réaliste mais tout à fait intentionnel. Cela vous permet de tester que vous pouvez enregistrer un nouveau vote après ce délai. Dans la pratique, la durée devrait être celle choisie pour le sondage.

Lorsque l'utilisateur valide le formulaire, le script commence par contrôler l'existence d'un cookie indiquant s'il existe déjà un vote, ici le cookie `"votant"` (repère ❶), et la valeur de ce vote, ici le cookie `"vote"` (repère ❷). Si un vote est déjà enregistré, une boîte d'alerte JavaScript affiche un message indiquant qu'il est impossible de voter deux fois (repère ❸) et rappelle le vote précédent.

S'il a voté, une boîte d'alerte JavaScript affiche la valeur du premier vote. Dans le cas contraire, les deux cookies sont enregistrés, puis une boîte d'alerte de remerciement s'affiche (voir la figure 12-1). Dans la réalité, chaque vote doit évidement être enregistré sur le serveur, dans un fichier texte, par exemple, de façon à pouvoir afficher les résultats du vote (voir le chapitre 11 pour réaliser cet enregistrement), ou dans une base de données (voir les chapitres 14 à 18).

Le script est découpé en plusieurs morceaux, délimités par les balises habituelles `<?php` et `?>`. Entre ces morceaux de scripts sont incorporés les éléments XHTML destinés à afficher les boîtes d'alertes.

La suite du fichier est le code XHTML créant l'interface visible du sondage, constituée de boutons radio portant tous le même nom, ce qui les rend exclusifs et n'autorise qu'un seul choix.

Exemple 12-3. Sondage avec vérification des votes

```
<?php
  1ère partie du script PHP
  if(isset($_POST['choix']))
  {
    if($_COOKIE["votant"] ←❶&& $_COOKIE["vote"]) ←❷
    {
      $vote = $_COOKIE["vote"];
?>
      <!--Code JavaScript -->
      <script type="text/javascript" >
      alert('Vous avez déja voté pour <?php echo $vote ?>!') ←❸
      </script>
<!-- 2 eme partie du script PHP-->
<?php
    }
    else
    {
      $vote = $_COOKIE["vote"];
      setcookie("votant","true",time()+300);
      setcookie("vote","{$_POST['choix']}",time()+300);
      //enregistrement du vote dans un fichier --> voir chapitre 11
      if(file_exists("sondage.txt") )
      {
```

L'usage de la constante `SID` est de loin préférable pour transmettre l'identifiant vers d'autres pages.

L'envoi de mails

Vous avez déjà vu au chapitre 6, consacré aux formulaires, la possibilité de communication par e-mail entre un internaute et un site Web pour transmettre les données d'un formulaire.

Vous allez découvrir ici la possibilité d'envoyer des e-mails du serveur vers le poste client. Pour peu que votre serveur vous l'autorise, cette fonctionnalité vous permet d'envoyer un e-mail contenant un identifiant et un code d'accès à une personne qui s'inscrit sur un forum de discussion ou un site en donnant son adresse e-mail. Ce contrôle évite, par exemple, les saisies fantaisistes ou, pire, les usurpations d'adresse e-mail.

Sur un site de commerce en ligne, cette fonction donne la possibilité d'envoyer automatiquement une confirmation de commande sitôt qu'elle est passée.

La fonction mail()

La première chose à faire est bien entendu de vous assurer auprès de votre hébergeur que vous disposez de la fonction `mail()`. De nombreux hébergeurs, en particulier les gratuits, désactivent la fonction d'envoi d'e-mail pour éviter le spam et surtout pour ne pas surcharger leurs serveurs. Certains autres réécrivent la fonction `mail()` pour en limiter les possibilités, le nombre d'envois par exemple.

Cette vérification peut être réalisée en utilisant l'exemple 7-1 du chapitre 7 (fichier `fonction1.php`), qui affiche la liste des fonctions disponibles sur le serveur. Pour l'hébergement d'un site professionnel, la possibilité d'envoi d'e-mail peut être un critère de sélection déterminant.

La fonction de base d'envoi d'e-mail se nomme donc `mail()`. Sa syntaxe est la suivante :

```
boolean mail($dest, $objet, $texte, [$entete])
```

Ses paramètres sont les suivants :

- `$dest` est une chaîne contenant l'adresse e-mail du destinataire. Pour envoyer le même e-mail à plusieurs adresses, il faut séparer chacune d'elles par une virgule.
- `$objet` est une chaîne contenant le texte qui apparaît dans la colonne Objet du logiciel de courrier du destinataire.
- `$texte` est une chaîne donnant le contenu réel du message, qui peut être au format texte ou au format HTML.
- `$entete` est une chaîne contenant les en-têtes nécessaires à l'envoi d'e-mails, lorsque ces derniers ne sont pas au format texte. Chaque en-tête se termine par la séquence `"\n"` sur un serveur Linux et `"\r\n"` sous Windows.

13

Rappels sur les SGBDR

Une base de données est un ensemble d'informations structurées sur un support et doté d'une certaine organisation. Votre carnet d'adresses en est un exemple élémentaire, l'annuaire du téléphone également, mais à une autre échelle.

L'accès rapide à l'information *via* des réseaux comme ceux des entreprises puis par Internet a nécessité la création de systèmes d'organisation des données permettant un accès rapide à l'information. Imaginez que vous deviez trouver toutes les personnes portant le même nom dans un département donné. La consultation de l'annuaire vous prendrait des heures. Face à de tels problèmes, l'informatisation du stockage des données est devenue une nécessité. C'est dans ce but qu'ont été créés les SGBDR (Système de gestion de base de données relationnelle).

L'objectif de ce chapitre n'est pas de vous fournir un cours complet sur les bases de données, loin de là. Il s'agit simplement de rappeler les notions essentielles qui vous permettront, à partir d'un besoin particulier de stockage d'information, de structurer les différentes données dans une base. Pour approfondir le sujet de la conception des bases de données et en particulier la méthode de conception UML (*Unified Modeling Language*, Langage de modélisation unifié en français), mieux adaptée à la conception orientée objet, vous pouvez vous reporter utilement à l'ouvrage de Christian Soutou, *De UML à SQL : Conception de bases de données*, paru aux éditions Eyrolles.

L'organisation des données doit permettre de répondre à des contraintes précises, notamment les suivantes :

- Les données doivent occuper le moins d'espace possible.
- Les redondances d'information doivent être évitées.

On distingue des cardinalités minimales et maximales :

- La cardinalité minimale mesure le nombre minimal de participation d'une entité dans l'association. Pour être enregistrée dans la base, une personne doit passer au moins une commande. La cardinalité minimale du coté de l'entité `personne` est donc 1. Un produit peut n'être commandé par aucune personne. La cardinalité minimale du coté de l'entité `article` est donc 0. En résumé, on indique ces deux cardinalités par la notation 0.N.
- La cardinalité maximale mesure le nombre maximal de participations d'une entité dans l'association. Une personne peut passer un nombre quelconque de commandes. La cardinalité maximale du coté de l'entité `personne` est donc notée N. Un produit peut être commandée par *N* personnes différentes. La cardinalité maximale du coté de l'entité `article` est donc N. En résumé, on indique ces deux cardinalités par la notation N.N. En pratique, on la note N.M pour montrer que les cardinalités ne sont pas nécessairement égales.

Par combinaison des différentes possibilités, on obtient quatre cardinalités possibles pour chaque entité :

- 0.1 : zéro ou une seule au maximum ;
- 1.1 : une et une seule ;
- 0.N : zéro ou plusieurs ;
- 1.N : une ou plusieurs.
- La figure 13-4 représente l'association `commande` munie de ses cardinalités.

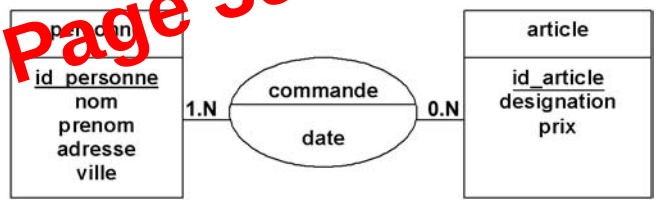


Figure 13-4

Association et cardinalités

Une association peut être définie au moyen des seules cardinalités maximales. L'association de la figure 13-4 peut être définie par la cardinalité N:M.

Après étude des cardinalités, on distingue plusieurs cas, qui vont constituer l'organisation des tables dans le modèle relationnel.

Détermination des cardinalités

Vous verrez dans les exemples qui suivent que la détermination des cardinalités peut dépendre des contraintes imposées au programmeur.

Il faut introduire ici la notion de dépendance fonctionnelle (DF), proche de la notion mathématique de fonction. On dit que deux attributs A et B sont en dépendance fonctionnelle si, à une valeur de A, correspond, au plus, une valeur de B.

On a, par exemple, les dépendances fonctionnelles suivantes :

```
numéro_client → Nom_client
commune → code postal
```

alors que les réciproques ne sont pas vraies.

Les méthodes permettant de vérifier et de normaliser le MCD sont les suivantes :

- Chaque attribut est atomique, c'est-à-dire qu'il ne peut contenir qu'une seule valeur choisie dans un domaine particulier. Par exemple, un attribut ne peut contenir plusieurs noms de personnes différentes. C'est ce qu'on nomme la première forme normale (1 N.F)
- Tous les attributs d'une entité en 1 N.F doivent dépendre complètement de la clé de l'entité et non d'une partie seulement de la clé. C'est la deuxième forme normale (2 N.F). Une entité en 1 N.F qui a une clé primaire composée d'un seul attribut est nécessairement en 2 N.F.
- Tous les attributs d'une entité en 2 N.F dépendent uniquement de la clé et non d'un autre attribut nom-clé. Si ce n'est pas le cas, il faut décomposer l'entité en deux entités distinctes, la nouvelle entité contenant les attributs nom-clé qui sont en dépendance fonctionnelle. C'est la troisième forme normale (3 N.F)
- Les attributs d'une association doivent être en 2 N.F et donc dépendre de toute la clé de l'association (constituée par la concaténation des clés des entités reliées).

La base magasin en ligne

Votre objectif premier étant de modéliser la base de données d'un site de commerce en ligne, vous devez envisager les contraintes d'utilisation de la base qui vous permettront de créer son MCD :

1. Un client enregistré dans la base a passé au moins une commande, sinon il ne figure pas dans la base.
2. Une commande peut contenir un ou plusieurs articles. Chaque commande a une date.
3. Le modèle doit permettre de retrouver toutes les commandes d'un client et la composition de chaque commande.

En fonction de ces contraintes, vous constatez immédiatement que le MCD de la figure 13-4 ne convient pas. En effet, il ne prend pas en compte le fait qu'un client peut commander plusieurs articles (contrainte n° 2).

Vous devez créer une entité séparée pour représenter une commande et les associations répondant aux besoins suivants :

Précision

Dans un premier temps, les champs `ville` et `age` ont été volontairement oubliés afin d'illustrer à la section suivante les possibilités de modification des tables offertes par phpMyAdmin.

4. Dans la page qui s'affiche, saisissez les caractéristiques des colonnes, ou champs, de la table dans un formulaire comptant autant de lignes qu'il y a de champs définis (voir la figure 14-5).

Figure 14-5

Création de la table client

Le code généré par phpMyAdmin est le suivant :

```
CREATE TABLE `magasin`.`client` (
`id_client` MEDIUMINT UNSIGNED NOT NULL AUTO_INCREMENT ,
`nom` VARCHAR( 30 ) NOT NULL ,
`prenom` VARCHAR( 30 ) NOT NULL ,
`adresse` VARCHAR( 60 ) NOT NULL ,
`mail` VARCHAR( 50 ) NULL DEFAULT 'pas de mail',
PRIMARY KEY ( `id_client` )
) ENGINE = InnoDB
```

Ce code affiche un tableau contenant la structure résumée de la table, comme illustré à la figure 14-6.

Figure 14-6

Structure de la table client

est possible de choisir le format d'exportation (repères ❶ à ❺), ainsi que l'exportation de la structure de la table (repère ❻) avec ou sans les données qu'elle contient (repère ❼).

Seule l'activation de la case Transmettre (repère ❽) permet de transmettre ces informations dans un fichier, compressé ou non (repère ❾). Si la case n'est pas cochée, phpMyAdmin affiche le code dans une nouvelle page.

Le contenu du fichier `article.sql` obtenu est le suivant :

```
-- phpMyAdmin SQL Dump
-- version 2.11.6
-- http://www.phpmyadmin.net
--
-- Serveur: localhost
-- Généré le : Lun 01 Décembre 2008 à 13:39
-- Version du serveur: 5.0.51
-- Version de PHP: 5.2.6

SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";

--
-- Base de données: `magasin`
--

-- --------------------------------------------------------

--
-- Structure de la table `article`
--

CREATE TABLE `article` (
  `id_article` char(5) collate latin1_bin NOT NULL,
  `designation` varchar(100) collate latin1_bin NOT NULL,
  `prix` decimal(8,2) NOT NULL,
  `categorie` enum('tous','photo','vidéo','informatique','divers')
  ➥collate latin1_bin NOT NULL default 'tous',
  PRIMARY KEY  (`id_article`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin;

--
-- Contenu de la table `article`
--

INSERT INTO `article` (`id_article`, `designation`, `prix`,`categorie`) VALUES
('CA300', 'Canon EOS 3000V zoom 28/80', '329.00', 'photo'),
('CAS07', 'Cassette DV60 par 5', '26.90', 'divers'),
('CP100', 'Caméscope Panasonic SV-AV 100', '1490.00', 'vidéo'),
('CS330', 'Caméscope Sony DCR-PC330', '1629.00', 'vidéo'),
('DEL30', 'Portable Dell X300', '1715.00', 'informatique'),
```

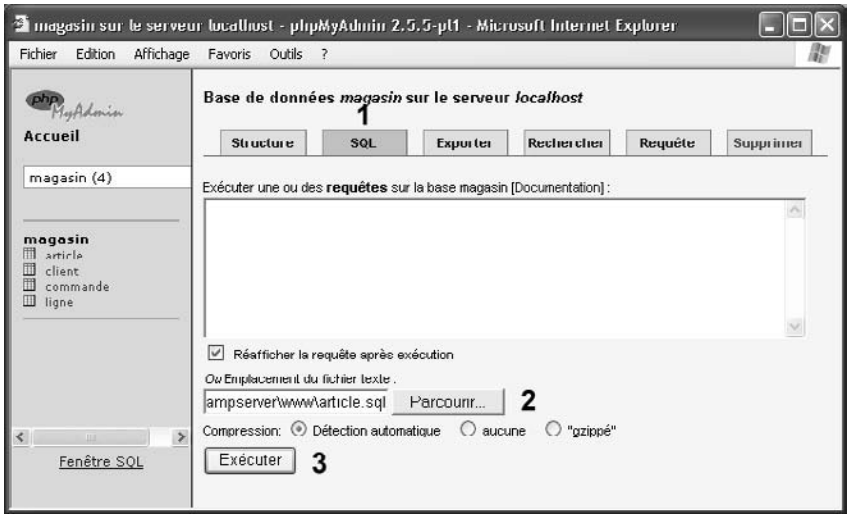


Figure 14-11

Utilisation d'un fichier .sql pour créer une table

Insertion de données

Une fois les tables créées, différents moyens permettent d'y enregistrer les données nécessaires au fonctionnement d'un site de commerce en ligne.

Il existe deux grandes catégories de données, les données statiques, qui ne dépendent que de l'administrateur du site comme celles de la table `article`, qui constituent le contenu du magasin, et les données en provenance des internautes clients du site, qui rempliront les autres tables. Ces dernières informations sont obtenues à partir d'un formulaire de saisie alors que les données statiques peuvent être enregistrées aussi bien à partir d'un formulaire en ligne qu'à l'aide de phpMyAdmin.

Insertion ligne par ligne

Vous allez commencer par insérer les données dans la table `article`. Il suffit pour cela de cliquer sur la base `magasin` dans la page d'accueil de phpMyAdmin puis, dans l'ensemble des tables de la base qui s'affichent, de sélectionner la table voulue et de cliquer sur le bouton Insérer. Un formulaire de saisie s'affiche alors comme illustré à la figure 14-12.

En cliquant sur le bouton "Enregistrer", vous générez l'affichage du code SQL de la requête, par exemple le suivant :

```
INSERT INTO 'article' ( 'id_article' , 'designation' , 'prix' , 'categorie' )
VALUES (
'CS110', 'Caméscope Sony 110', '1250.50', 'vidéo'
);
```

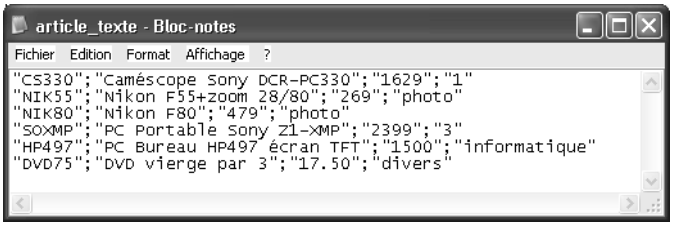


Figure 14-13

Le fichier texte visualisé dans le Bloc-notes

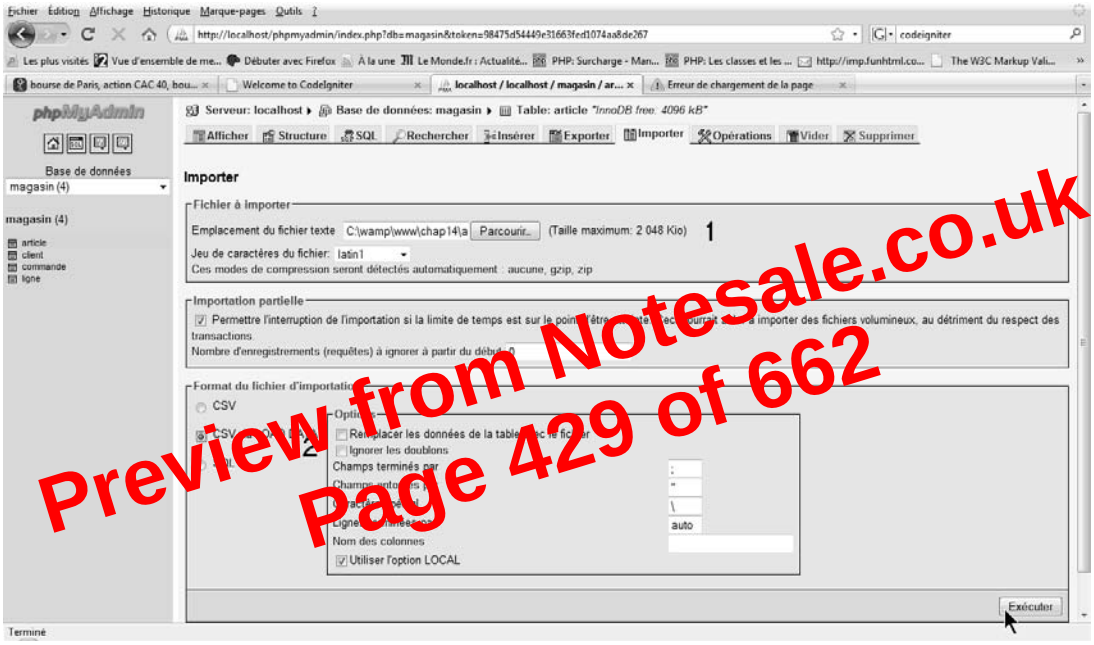


Figure 14-14

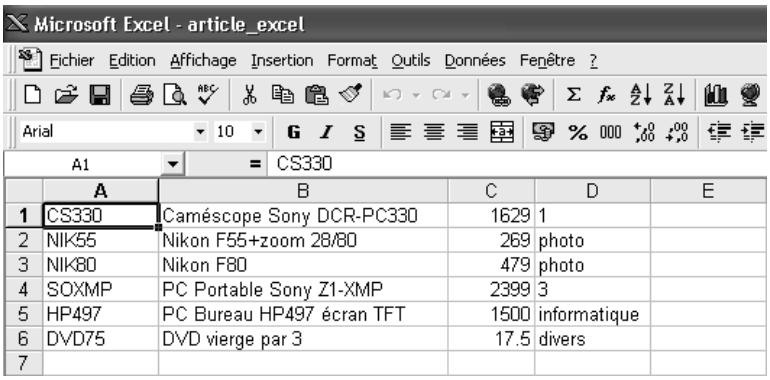
Formulaire d'insertion à partir d'un fichier texte

Si le fichier est situé sur le poste client, vous pouvez utiliser le bouton Parcourir (repère ❶) pour le localiser et conserver le choix par défaut `DATA LOCAL` (repère ❷).

Pour exécuter vous-même cette opération, vous écririez la requête SQL suivante :

```
LOAD DATA INFILE 'C:/article_texte.txt'
INTO TABLE `article`
FIELDS TERMINATED BY ';'
ENCLOSED BY '"'
ESCAPED BY '\\'
LINES TERMINATED BY '\r\n'
```

La procédure d'insertion est la même que pour un fichier texte ordinaire.



	A	B	C	D	E
1	CS330	Caméscope Sony DCR-PC330	1629	1	
2	NIK55	Nikon F55+zoom 28/80	269	photo	
3	NIK80	Nikon F80	479	photo	
4	SOXMP	PC Portable Sony Z1-XMP	2399	3	
5	HP497	PC Bureau HP497 écran TFT	1500	informatique	
6	DVD75	DVD vierge par 3	17.5	divers	
7					

Figure 14-16

Feuille de calcul Excel à insérer

Les données de la base magasin

Avant de procéder à des opérations de sélection des données à l'aide de code SQL, il vous faut mieux comprendre les résultats affichés par les différentes requêtes sur la base `magasin`. Ces résultats sont récapitulés aux tableaux 14-4 pour la table `client`, 14-5 pour la table `article`, 14-6 pour la table `commande` et 14-7 pour la table `ligne`.

Tableau 14-4 – Données de la table *client*

id_client	nom	prenom	age	adresse	ville	mail
1	Marti	Jean	36	5 av. Einstein	Orléans	mart@marti.com
2	Rapp	Paul	44	32 av. Foch	Paris	rapp@libert.com
3	Devos	Marie	18	75 bd Hochimin	Lille	grav@waladoo.fr
4	Hochon	Paul	22	33 rue Tsétsé	Chartres	hoch@fiscali.fr
5	Grave	Nuyen	18	75 bd Hochimin	Lille	grav@waladoo.fr
6	Hachette	Jeanne	45	60 rue d'Amiens	Versailles	NULL
7	Marti	Pierre	25	4 av. Henri	Paris	martin7@fiscali.fr
8	Mac Neal	John	52	89 rue Diana	Lyon	mac@freez.fr
9	Basile	Did	37	26 rue Gallas	Nantes	bas@walabi.com
10	Darc	Jeanne	19	9 av. d'Orléans	Paris	NULL
11	Gaté	Bill	45	9 bd des Bugs	Lyon	bill@microhard.be

Tableau 14-8 – Les opérateurs de comparaison *(suite)*

Opérateur	Description
`IS NULL`	Retourne `TRUE` si la valeur est `NULL`. Par exemple : `SELECT nom,prenom FROM auteurs WHERE prenom IS NULL`
`IS NOT NULL`	Retourne `TRUE` si la valeur n'est pas `NULL`. Par exemple : `SELECT nom,prenom FROM auteurs WHERE prenom IS NOT NULL`
`express BETWEEN min AND max`	Retourne `TRUE` si la valeur est comprise entre `min` et `max` (limites comprises). Par exemple, le code suivant sélectionne tous les noms dont l'initiale est comprise entre A et E : `SELECT nom FROM client WHERE nom BETWEEN 'A' and 'E'`
`express NOT BETWEEN min AND max`	Retourne `TRUE` si la valeur n'est pas comprise entre `min` et `max` (limites comprises). Par exemple, le code suivant sélectionne tous les noms dont l'initiale n'est pas comprise entre A et E : `SELECT nom FROM client WHERE nom NOT BETWEEN 'A' and 'E'`
`express LIKE 'motif'`	Retourne la valeur `TRUE` si l'expression est conforme au motif défini. Pour définir les motifs, vous utilisez deux caractères particuliers, ou jokers : le caractère de soulignement (_) pour remplacer un caractère quelconque et le caractère `%` pour remplacer un nombre variable de caractères. Si le motif est égal à `'chaine_'`, tous les mots commençant par `chaine` suivi de n'importe quel caractère conviennent. Si le motif est égal à `'%mar'`, tous les mots qui contiennent la chaîne `'mar'` précédée de n'importe quel groupe de caractères conviennent. Si le motif est `'%mar%'`, toutes les chaînes contenant simplement la chaîne `'mar'` sont valables. Par exemple, le code suivant sélectionne tous les noms qui contiennent les lettres `'tin'` : `SELECT nom,prenom FROM client WHERE nom LIKE '%tin'`
`express NOT LIKE 'motif'`	Négation de l'opérateur `LIKE`

`express REGEXP 'motif'`	Retourne `TRUE` si le motif de l'expression régulière est trouvé dans la valeur d'une colonne. Par exemple, le code suivant sélectionne tous les noms dont le prénom commence par la lettre J suivie d'une voyelle puis d'un nombre quelconque de caractères :
	`SELECT nom`
	`FROM client WHERE prenom REGEXP 'J[aeiouy].*'`

`express NOT REGEXP 'motif'`	Négation de l'opérateur `REGEXP`
`express IN(val1,val2,...)`	Retourne `TRUE` si la valeur est incluse dans l'ensemble des valeurs listées. Par exemple, le code suivant sélectionne tous les noms d'auteurs qui ont 21, 22 ou 23 ans : `SELECT nom FROM auteurs WHERE age IN (21,22,23)`
`express NOT IN(val1, val2,...)`	Retourne `TRUE` si la valeur n'est pas incluse dans l'ensemble des valeurs listées. Par exemple, le code suivant sélectionne tous les noms d'auteurs qui n'ont ni 21, ni 22 ni 23 ans : `SELECT nom FROM auteurs WHERE age NOT IN (21,22,23)`
`ISNULL(express)`	Retourne `TRUE` si la valeur de l'expression est de type `NULL`. Par exemple, le code suivant sélectionne tous les noms d'auteurs dont l'attribut `prenom` est vide : `SELECT nom FROM auteurs WHERE ISNULL(prenom)`
`COALESCE(col1,col2,...)`	Retourne la première valeur non `NULL` de la liste passée en paramètre. Par exemple, le code suivant retourne tous les e-mails non `NULL` de la table `client`, les mails `NULL` étant remplacés par les noms : `SELECT COALESCE( mail, nom ) FROM  client`

Les fonctions statistiques

Ces fonctions opèrent sur les données d'une même colonne. Elles retournent des résultats calculés sur l'ensemble des valeurs non `NULL` correspondant à cette colonne. Il est possible de combiner plusieurs fonctions statistiques à condition d'utiliser la clause `GROUP BY`, que nous détaillons à la section suivante.

Le tableau 14-11 récapitule les fonctions statistiques utilisables dans des requêtes MySQL.

Tableau 14-11 – Fonctions statistiques

Fonction	Description
`AVG(colonne)`	Retourne la moyenne des valeurs de la colonne précisée.
`COUNT(colonne)`	Retourne le nombre de lignes dont la valeur n'est pas `NULL` dans la colonne précisée. `COUNT(*)` retourne le nombre total de lignes, même si certaines ont la valeur `NULL`.
`COUNT(DISTINCT colonne)`	Retourne le nombre de lignes ayant une valeur non `NULL` et distincte (en éliminant les doublons).
`MAX(colonne)`	Retourne la valeur maximale de la colonne précisée.
`MIN(colonne)`	Retourne la valeur minimale de la colonne précisée.
`SUM(colonne)`	Retourne la somme des valeurs de la colonne précisée.

Exemples

Pour calculer l'âge moyen des clients :

```
SELECT AVG(age)FROM client
```

Vous obtenez la valeur 32.8182.

Pour calculer le nombre de clients ayant indiqué leur adresse e-mail (donc le nombre de lignes pour lesquelles la colonne `mail` n'est pas `NULL`) :

```
SELECT COUNT(mail)FROM client
```

Vous obtenez la valeur `9`, alors qu'il y a 11 clients dans la table.

Pour calculer le nombre total de clients :

```
SELECT COUNT(nom)FROM client
```

Vous obtenez la valeur `11`. Vous auriez pu appliquer la fonction `COUNT()` à n'importe quelle colonne ayant l'attribut `NOT NULL` et obtenir le même résultat.

Pour calculer le nombre de villes différentes de la table `client` :

```
SELECT COUNT(DISTINCT ville)FROM client
```

Vous obtenez la valeur `7`.

Pour déterminer le prix maximal de la table `article` :

```
SELECT MAX(prix)FROM article
```

Vous obtenez la valeur `2399.00`.

Dans un regroupement, vous pouvez indiquer une condition de restriction des lignes, comme dans une requête de sélection habituelle. Vous utilisez pour cela la clause `HAVING`, qui est l'équivalent de la clause `WHERE` mais n'est utilisée qu'après une clause `GROUP BY`. Les conditions écrites dans `HAVING` peuvent utiliser les mêmes opérateurs et fonctions que celles écrites dans `WHERE`.

Par exemple, pour calculer l'âge moyen des clients qui habitent des villes dont l'initiale est L, vous ajoutez une condition de restriction `HAVING` à l'aide de l'opérateur `LIKE` après la clause `GROUP BY`.

La requête suivante :

```
SELECT AVG( age ),ville
FROM  client
GROUP  BY ville
HAVING ville LIKE 'L%'
```

affiche le résultat ci-dessous :

AVG(age)	ville
18.0000	Lille
48.5000	Lyon

Les jointures

Faire une jointure consiste à effectuer une sélection de données sur plusieurs tables. Il faut pour cela que les tables concernées par la jointure aient au moins chacune une colonne contenant un même type d'information. C'est le cas des tables `client` et `commande` de la base `magasin`, qui ont en commun la colonne `id_client`. La commande `SELECT` peut donc s'appliquer, avec une condition `WHERE`, à cette colonne.

La syntaxe la plus courante d'une jointure est la suivante :

```
SELECT col1,col2,…
FROM table1,table2,…
WHERE condition_de_jointure
```

La condition de jointure est de la forme :

```
table1.colX = table2.colY
```

dans laquelle `colX` et `colY` contiennent des données représentant la même information, comme l'identifiant de client.

Jointure de deux tables

L'association entre un numéro de commande et le nom d'un client fait intervenir les tables `client` et `commande`. Pour retrouver toutes les commandes faites par un client, vous opérez une jointure entre ces tables en utilisant la colonne `id_client` commune aux deux tables.

Le tri des données est effectué par le nom des clients de façon à voir d'un coup toutes les commandes d'un même client :

```
SELECT commande.id_comm, nom, prenom, ville, sum( quantite * prixunit )
➥AS 'Prix total'
FROM commande,  'client' , ligne
WHERE client.id_client = commande.id_client AND commande.id_comm = ligne.id_comm
GROUP  BY id_comm
ORDER BY nom
```

La requête affiche le résultat illustré à la figure 14-23.

id_comm	nom	prenom	ville	Prix total
5	Basile	Did	Nantes	329.00
13	Basile	Did	Nantes	3998.00
2	Basile	Did	Nantes	3000.00
6	Darc	Jeanne	Paris	1570.70
4	Devos	Marie	Lille	7232.00
9	Gaté	Bill	Lyon	[illegible]
12	Grave	[illegible]	[illegible]	4994.60
[illegible]	[illegible]	Nuyen	Lille	[illegible]
10	Hochon	[illegible]	[illegible]tres	5429.00
11	[illegible]	John	Lyon	175.00
3	Marti	Jean	Orléans	2395.00
8	Marti	Pierre	Paris	3889.00
7	Rapp	Paul	Paris	9995.00

Figure 14-23
Jointure sur trois tables

Exercices

Exercice 1

Créez une base nommée `voitures`. Créez ensuite les tables de la base `voitures` selon le modèle logique défini dans les exercices du chapitre 13. Omettez volontairement certaines colonnes, et faites quelques erreurs de type de colonne. Une fois les tables créées, ajoutez les colonnes manquantes, et corrigez les erreurs. Vérifiez la structure de chaque table.

Exercice 2

Exportez les tables de la base `voitures` dans des fichiers SQL.

Envoi de requêtes SQL au serveur

Toute opération à réaliser sur une base nécessite d'envoyer au serveur une requête SQL rédigée à l'aide des commandes détaillées au chapitre précédent.

Pour envoyer une requête, vous utilisez d'abord la fonction `mysql_query()`, dont la syntaxe est la suivante :

```
resource mysql_query(string $requete [,resource $idcom][,int mode])
```

La chaîne `$requete` contient le code de la requête SQL. Elle ne doit pas se terminer par un point-virgule. Les requêtes étant souvent longues, il est recommandé, à des fins de lisibilité du code, de les écrire dans une variable chaîne `$requete` passée ensuite à la fonction.

`$idcom` est l'identifiant de connexion. Il est facultatif si une seule connexion est ouverte. `mode` est une constante entière, qui prend les valeurs `MYSQL_STORE_RESULT` (valeur par défaut) si le résultat de la requête doit être mis en buffer (mémoire tampon sur le serveur) et `MYSQL_USE_RESULT` dans le cas contraire. Dans ce dernier cas, il faut lire le résultat avant d'envoyer une nouvelle requête au serveur, faute de quoi le résultat est perdu.

La fonction retourne un identifiant de résultat de type `resource`, noté systématiquement `$result` dans ces exemples. Si la requête contient des commandes `SELECT`, cet identifiant permet d'accéder aux données fournies par la requête en utilisant certaines fonctions PHP que vous découvrirez plus loin dans ce chapitre. Pour les autres requêtes (suppression, modification, mise à jour), la fonction retourne `TRUE` si la requête est bien exécutée. Si une requête quelconque n'est pas exécutée, la fonction `mysql_query()` retourne `FALSE`. Il est recommandé d'effectuer un test pour vérifier la bonne réalisation de chaque requête SQL dans un script.

Un script d'envoi de requête a la forme suivante :

```
<?php
include("connex.inc.php"); ←❶
$idcom=connex("magasin","myparam"); ←❷
$requete="SELECT * FROM article ORDER BY categorie"; ←❸
$result=@mysql_query($requete,$idcom); ←❹
if(!$result)
{
  echo "Lecture impossible"; ←❺
}
else
{
//Lecture des résultats éventuels de la requête ←❻
  }
  ?>
```

Il effectue successivement l'inclusion du fichier `connex.inc.php` (repère ❶), la connexion au serveur grâce à la fonction `connex()` (repère ❷), l'écriture de la requête (repère ❸), l'envoi de la requête et la récupération du résultat (repère ❹) et enfin l'affichage d'un message d'erreur éventuel si la requête n'a pas abouti (repère ❺) et celui des résultats de la requête (repère ❻). Nous verrons, dans les sections suivantes, comment afficher les résultats d'une requête de sélection.

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`, il est indispensable de recueillir les données. PHP offre une grande variété de fonctions qui permettent de récupérer des données sous des formes diverses, la plus courante étant un tableau. Chacune de ces fonctions ne récupérant qu'une ligne du tableau à la fois, il faut recourir à une ou plusieurs boucles pour lire l'ensemble des données.

Lecture à l'aide d'un tableau

La fonction la plus perfectionnée pour lire les données dans un tableau est `mysql_fetch_array()`. Sa syntaxe est la suivante :

```
array mysql_fetch_array(resource $result [,int typetab])
```

Cette fonction retourne un tableau contenant autant d'éléments qu'il y a de colonnes précisées dans la requête `SELECT`. Le paramètre `$result` est celui qui a été retourné par la fonction `mysql_query()`, et le paramètre `typetab` une constante entière précisant si le tableau retourné doit être associatif (valeur `MYSQL_ASSOC`), indicé (valeur `MYSQL_NUM`) ou les deux à la fois (valeur `MYSQL_BOTH`, qui est la valeur par défaut). Si le tableau est associatif, les clés du tableau sont les noms des colonnes de la table interrogée ou des alias, si vous en avez définis dans la requête. Il n'est donc pas nécessaire de connaître l'ordre des colonnes dans la table.

Si le tableau est indicé, l'indice d'un élément est celui de la colonne dans la requête, la première colonne ayant l'indice 0. Pour récupérer la valeur d'une colonne précise, il est indispensable de connaître sa position dans la requête.

Chaque nouvel appel de la fonction `mysql_fetch_array()` retourne la ligne suivante du résultat identifié par `$result`, ou `FALSE` s'il n'y a plus de ligne à lire. Il faut effectuer une boucle pour lire chacune des lignes puis une autre boucle imbriquée dans la première pour lire chacune des valeurs des colonnes de la table.

> **Nombre de lignes et de colonnes d'un résultat**
>
> La fonction `int mysql_num_fields(resource $idresult)` permet de déterminer le nombre de colonnes du résultat et la fonction `int mysql_num_rows(resource $idresult)` le nombre de lignes.

L'exemple 15-3 met en pratique les techniques que vous venez d'aborder en effectuant la lecture de la table `article` de la base `magasin` créée au chapitre 14. Après l'appel de la fonction de connexion (repère ❶), l'écriture de la requête SQL (repère ❷) et son envoi au serveur, le résultat est identifié par la variable `$result` (repère ❸). Le script contrôle ensuite si le résultat est valide (repère ❹) et lit le nombre de colonnes et de lignes retourné par la requête (repère ❺ et ❻). La première boucle `while` de lecture du résultat

retourne une ligne à la fois dans un tableau indicé (repère ❼), puis une boucle `foreach` lit chacune des valeurs du tableau et réalise un affichage dans un tableau XHTML (repère ❽). La ressource `$result` est ensuite libérée (repère ❾).

Libération de la mémoire

Une fois les données utilisées pour réaliser un affichage dans une page XHTML, il est possible de libérer la mémoire occupée par la variable `$result` en appelant la fonction `mysql_free_result ($result)`. Cette dernière retourne `TRUE` en cas de réussite et `FALSE` dans le cas contraire. Cette opération ne doit être faite que si vous n'avez plus besoin des données dans le script.

La figure 15-1 illustre la page réalisée par ce script.

Exemple 15-3. Lecture de la table *article*

```
<!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-8859-1" />
<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("connex.inc.php");
$idcom=connex_magasin("myparam"); ←❶
$requete="SELECT * FROM article ORDER BY categorie"; ←❷
$result=@mysql_query($requete,$idcom); ←❸
if(!$result) ←❹
{
    echo "Lecture impossible";
}
else
{
    $nbcol=mysql_num_fields($result); ←❺
    $nbart=mysql_num_rows($result); ←❻
    echo "<h3> Tous nos articles par catégorie</h3>";
    echo "<h4> Il y a $nbart articles en magasin </h4>";
    echo "<table border=\"1\"><tbody>";
    echo "<tr><th>Code article</th> <th>Description</th> <th>Prix</th>
    ➥<th>Catégorie</th></tr>";
    while($ligne=mysql_fetch_array($result,MYSQL_NUM)) ←❼
    {
    echo "<tr>";
    foreach($ligne as $valeur) ←❽
    {
```

Utilisation d'un objet

À l'exemple 15-5, l'appel de la fonction `mysql_field_name($result,$i)` pourrait être remplacé par `mysql_fetch_field($result,$i)->name`. Cette dernière fonction, ayant pour syntaxe `object mysql_fetch_field($result,$i)`, retourne un objet dont la propriété `name` contient le nom de la colonne dont la position est précisée par l'entier `$i`.

Récupération des valeurs dans un objet

Vous pouvez récupérer non seulement le nom de chaque colonne, comme en appelant la fonction `mysql_fetch_field()`, mais également la valeur de chaque colonne d'un résultat sous la forme d'une propriété d'objet grâce à la fonction `mysql_fetch_object()`.

La syntaxe de cette fonction est la suivante :

```
object mysql_fetch_object(resource $result)
```

L'objet retourné par cette fonction a autant de propriétés que la requête a sélectionné de colonnes, les noms des propriétés étant ceux des colonnes ou des alias éventuels. L'exemple 15-6 réalise le même affichage que l'exemple 15-5 en utilisant cette fonction. Les seules différences résident dans l'utilisation de la fonction `mysql_fetch_field()` (repère ❶) pour afficher les en-têtes du tableau XHTML puis de `mysql_fetch_object()` pour lire chacune des lignes du résultat (repère ❷).

Un nouvel appel de `mysql_fetch_field()` permet d'enregistrer le nom de chacune des propriétés de l'objet dans la variable `$nomcol` (repère ❸) qui permet d'afficher toutes les données d'une ligne. La syntaxe `$ligne->$nomcol` permet de lire chaque propriété de l'objet (repère ❹). Le résultat affiché est identique à celui de la figure 15-3.

Exemple 15-6 : Lecture des données au moyen d'un objet

```
<!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-8859-1" />
<title>Lecture de la table client</title>
<style type="text/css" >
table {border-style:double;border-width:3px;border-color:red;
➥background-color:yellow;}
</style>
</head>
<body>
<?php
include("connex.inc.php");
$idcom=connex("magasin","myparam");
$requete="SELECT id_client AS 'Code client',nom,prenom,adresse,age,mail
FROM client
WHERE ville ='Paris'
ORDER BY nom";
```

Insertion des données

Comme vous l'avez vu au chapitre 6, consacré aux formulaires, l'outil de communication essentiel entre le poste client et le serveur est le formulaire XHTML. Ce dernier permet la saisie de données et leur envoi vers le serveur, sur lequel un script PHP enregistre les valeurs saisies dans la base MySQL.

Vous disposez déjà de la fonction personnalisée de connexion `connex()` et de `mysql_query()` pour envoyer la requête. Seule la requête qui contient la commande SQL `INSERT`, qui va être envoyée au serveur, distingue cette opération de celle de lecture des données.

Le script de l'exemple 15-7 réalise ce type d'insertion à partir d'un formulaire permettant à un client d'enregistrer ses coordonnées lors d'une commande. Comme ceux du chapitre 6, il commence par vérifier l'existence des saisies obligatoires correspondant aux variables `$_POST['nom']`, `$_POST[adresse]` et `$_POST[ville]` (repère ❶).

Dans le cas où une requête est formée en utilisant des saisies faites par l'utilisateur dans un formulaire, il est préférable d'utiliser un caractère d'échappement pour les caractères spéciaux des chaînes récupérées dans le tableau `$_POST`, en particulier les guillemets, qui peuvent poser problème. Vous disposez pour cela des fonctions `mysql_escape_string()` et `mysql_real_escape_string()`, dont les syntaxes respectives sont les suivantes :

```
string mysql_escape_string(string $chaine)
string mysql_real_escape_string(string $chaine[,resource $idcom])
```

La seconde présente l'avantage de tenir compte du jeu de caractères utilisé. Le jeu de caractères utilisé sur le client MySQL peut être lu en appelant la fonction `mysql_client_encoding()` qui retourne ce code dans une chaîne explicite, par exemple `'latin1'`. Par précaution, il est préférable d'appliquer ces fonctions à chacune des variables récupérées plutôt qu'à la chaîne de requête dans son entier. Aucune de ces deux fonctions n'échappe les caractères `%` et `"_"`.

Vous récupérez et protégez ensuite toutes les valeurs du tableau `$_POST` dans des variables de façon à obtenir un code plus court par la suite (repères ❷ à ❽).

La valeur `"\N"` représente la valeur `NULL` de la variable `$id_client`. Elle permet que la colonne `id_client` de la table `client` soit incrémentée d'une unité à chaque nouvelle insertion car elle est déclarée avec l'option `AUTO_INCREMENT`.

La fonction `mysql_insert_id()`, dont la syntaxe est la suivante :

```
int mysql_insert_id([$idcom])
```

retourne la dernière valeur insérée dans une colonne ayant cette option `AUTO_INCREMENT`. Elle sert à donner son numéro au client pour qu'il puisse s'identifier lors d'une autre connexion.

La requête SQL `INSERT` contenue dans la variable `$requete` (repère ❾) permet l'insertion de toutes ces valeurs dans la table `client`. Dans le cas d'une commande `INSERT`, le résultat de la requête envoyée par la fonction `mysql_query()` n'est qu'une valeur booléenne `TRUE` ou `FALSE` selon que l'insertion a été réalisée ou non (repère ❿). Elle vous permet d'afficher un message d'information indiquant la bonne exécution de l'insertion. Le numéro

`int mysql_num_fields(resource $result)`

Retourne le nombre de colonnes présentes dans un résultat.

`int mysql_num_rows(resource $result)`

Retourne le nombre de lignes d'un résultat.

`resource mysql_pconnect(string $host,string $user,string $pass)`

Identique à la fonction `mysql_connect()` mais crée une connexion permanente.

`boolean mysql_ping([resource $idcom])`

Vérifie si la connexion est toujours active et effectue une reconnexion dans le cas contraire. Retourne `TRUE` si la connexion est effectuée et `FALSE` dans le cas contraire.

`resource mysql_query(string $requete[,resource $idcom[,int mode]])`

Envoie une requête SQL au serveur et retourne un identifiant de résultat.

`string mysql_real_escape_string(string $chaine [, resource $idcom])`

Échappe la chaîne précisée afin de l'inclure dans une requête SQL.

`divers mysql_result( resource $result,int N [,divers M])`

Retourne la valeur du champ de la colonne `M` de la ligne `N` du résultat.

`boolean mysql_select_db(string base[,resource $idcom])`

Choisit la base sur laquelle vont s'effectuer les opérations suivantes. Retourne `TRUE` si la base existe et `FALSE` dans le cas contraire.

`string mysql_tablename( resource $result,int N)`

Utilise l'identifiant de résultat retourné par la fonction `mysql_list_tables()` et retourne le nom de la nième table de la base.

`resource mysql_unbuffered_query(string $requete[,resource $idcom])`

Envoie une requête au serveur mais ne met pas le résultat en buffer. Le résultat doit être utilisé avant l'envoi d'une autre requête.

Exercices

Tous les exercices ci-dessous portent sur la base de données `voitures` créée au chapitres 13 et 14.

Exercice 1

Créez un script permettant d'afficher le contenu de la table `modele` dans un tableau XHTML. Les résultats doivent être triés par marque.

Exercice 2

Créez un formulaire permettant l'insertion de nouvelles données dans la table `modele`.

(repère ❻). L'affichage des données sélectionnées est réalisé dans une boucle `while` avec la méthode `fetch_array()` comme dans l'exemple précédent. Le tableau retourné étant ici indicé (repère ❼), sa lecture est effectuée par une boucle `foreach`. Le résultat et l'objet connexion sont ensuite supprimés.

Exemple 16-5. Lecture des noms des colonnes

```
<!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");
$idcom=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;";
//
$result=$idcom->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)
```

L'exemple 16-6 réalise, en employant cette méthode, la recherche et l'affichage dans un tableau XHTML de tous les clients qui habitent Paris. Par contre, pour afficher les titres du tableau nous n'allons pas utiliser la méthode `fetch_fields()` comme précédemment. Dans la requête SQL nous définissons des alias qui vont être les en-têtes (repère ❶). Pour les lire, nous appelons une première fois la méthode `fetch_object()` pour l'objet `$result` (repère ❷) et récupérons un objet `$titres`. Ici, ce ne sont pas les valeurs de ses propriétés qui nous intéressent mais leurs noms. Ces derniers sont récupérables dans la variable `$colonne` en appliquant une boucle `foreach` à l'objet `$titres` (repère ❸). Nous intégrons alors ces valeurs dans des éléments `<th>` (repère ❹). Nous pourrions maintenant appeler de nouveau la méthode `fetch_object()` pour lire les données mais, à ce niveau, il se pose un problème. En effet, cette méthode ayant déjà été appelée, un deuxième appel lirait la deuxième ligne de résultat et la première serait perdue. Une solution est d'utiliser la méthode `data_seek()` (repère ❺), dont la syntaxe est :

```
boolean $result->data_seek(int N)
```

Le paramètre `N` désigne la ligne de résultat sur laquelle nous voulons pointer. Le booléen retourné permet de vérifier que l'opération est bien réalisée et que la ligne N existe bien, par exemple. Nous pouvons maintenant utiliser une boucle `while` pour lire chacune des lignes du résultat (repère ❻) et afficher les données. Remarquez que, comme nous l'avons déjà signalé, les noms des propriétés de l'objet `$titres` sont ici les alias définis dans la requête SQL (repère ❼). Le tableau XHTML obtenu est représenté à la figure 16-3.

☞ **Exemple 16-6. Lecture des données dans un objet**

```
<!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 client</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");
$idcom=connexobjet("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)
{
  echo "Lecture impossible";
}
```

Exemple 16-9 Mise à jour de données

```
<?php
  if(empty($_POST['code'])){header("Location:mysqliex8.php");} ←❶
?>
<!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-8859-1" />
<title>Modifiez vos coordonnées</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head>
<body>
<?php
include('connexobjet.inc.php');
$idcom=connexobjet('magasin','myparam');
if($_POST['modif']!='Enregistrer') ←❷
{
  $code=$idcom->escape_string($_POST['code']); ←❸
  //Requête SQL
  $requete="SELECT * FROM client WHERE id_client='$code' "; ←❹
  $result=$idcom->query($requete);
  $coord=$result->fetch_row(); ←❺
  //Création du formulaire complété des coordonnées existantes ←❻
  echo "<form action= \"".$_SERVER['PHP_SELF']."\"
  ➥method=\"post\" enctype=\"application/x-www-form-urlencoded\">";
  echo "<fieldset>";
  echo "<legend><b>Modifiez vos coordonnées</b></legend>";
  echo "<table>";
  echo "<tr><td>Nom : </td><td><input type=\"text\" name=\"nom\"
  ➥size=\"40\" maxlength=\"30\" value=\"$coord[1]\"/> </td></tr>";
  echo "<tr><td>Prénom : </td><td><input type=\"text\" name=\"prenom\"
  ➥size=\"40\" maxlength=\"30\" value=\"$coord[2]\"/></td></tr>";
  echo "<tr><td>Age : </td><td><input type=\"text\" name=\"age\"
  ➥size=\"40\" maxlength=\"2\" value=\"$coord[3]\"/></td></tr>";
  echo "<tr><td>Adresse : </td><td><input type=\"text\" name=\"adresse\"
  ➥size=\"40\" maxlength=\"60\" value=\"$coord[4]\"/></td></tr>";
  echo "<tr><td>Ville : </td><td><input type=\"text\" name=\"ville\"
  ➥size=\"40\" maxlength=\"40\" value=\"$coord[5]\"/></td></tr>";
  echo "<tr><td>Mail : </td><td><input type=\"text\" name=\"mail\"
  ➥size=\"40\" maxlength=\"50\" value=\"$coord[6]\"/></td></tr>";
  echo "<tr><td><input type=\"reset\" value=\" Effacer \"></td> <td><input
  ➥type=\"submit\" name=\"modif\" value=\"Enregistrer\"></td></tr></table>";
  echo "</fieldset>";
  echo "<input type=\"hidden\" name=\"code\"  value=\"$code\"/>"; ←❼
  echo "</form>";
  $result->close();
  $idcom->close();
}
```

On peut remarquer l'utilisation de la fonction MySQL `lower()` qui permet d'effectuer une recherche insensible à la casse. De cette façon, que l'utilisateur cherche les mots-clés « `sony` » ou « `Sony` », il obtiendra bien les résultats présents dans la table `article`.

L'utilisation d'alias donne un meilleur affichage des titres du tableau de résultats. Après la connexion au serveur, vous récupérez le résultat de la requête dans la variable `$result`. La lecture des résultats et l'affichage de toutes les lignes retournées sont réalisés avec la méthode `fetch_row()` (repère ❾). La figure 16-8 illustre la page créée après la recherche du mot-clé `portable`.

Exemple 16-10. Page de recherche d'articles

```
<!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>Rechercher un article dans le magasin</title>
 </head>
 <body>
  <form action= "<?php echo $_SERVER['PHP_SELF']?>" method="post"
  ➥enctype="application/x-www-form-urlencoded">
  <fieldset>
  <legend><b>Rechercher un article en magasin</b></legend>
  <table>
  <tbody>
  <tr> <td>Mot-cl&#233; : </td>
  <td><input type="text" name="motcle" size="40" maxlength="40"
  ➥value="<?php $_POST['motcle']?>" /></td>
  </tr>
  <tr>
  <td>Dans la cat&#233;gorie : </td>
  <td>
  <select name="categorie">
  <option value="tous">Tous</option>
  <option value="vidéo">Vid&#233;o</option>
  <option value="informatique">Informatique</option>
  <option value="photo">Photo</option>
  <option value="divers">Divers</option>
  </select>
  </td>
  </tr>
  <tr>
  <td>Trier par : </td>
  <td>
  <select name="tri">
  <option value="prix">Prix</option>
  <option value="categorie">Cat&#233;gorie</option>
  <option value="id_article">Code</option>
  </select>
  </td>
```

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 16-2.

Tableau 16-2. Signification des caractères de la chaîne $types

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()`.

L'exemple 16-11 présente une application de requête préparée dans laquelle ce sont les saisies d'un utilisateur qui déterminent les valeurs des paramètres et donc la requête finale. Un formulaire classique demande la saisie d'un nom de ville et d'un numéro de client (repères ❶ et ❷) dans le but de trouver tous les clients habitant cette ville et dont l'identifiant est supérieur ou égal à la valeur saisie. Ces saisies sont d'abord récupérées dans les variables `$ville` et `$id_client` (repères ❸ et ❹). Après la connexion à la base, nous écrivons la requête préparée avec la méthode `prepare()` (repère ❺), nous lions les paramètres de type `string` et `integer` aux variables `$ville` et `$id_client` (repère ❻), puis

```
    }
    echo "</div>";
    $reqprep->free_result(); ←⓾
    $idcom->close(); ←⓫
}
?>
```

Avec notre base de données magasin et les paramètres « Paris » et « 3 », nous obtenons par exemple l'affichage présenté à la figure 16-9

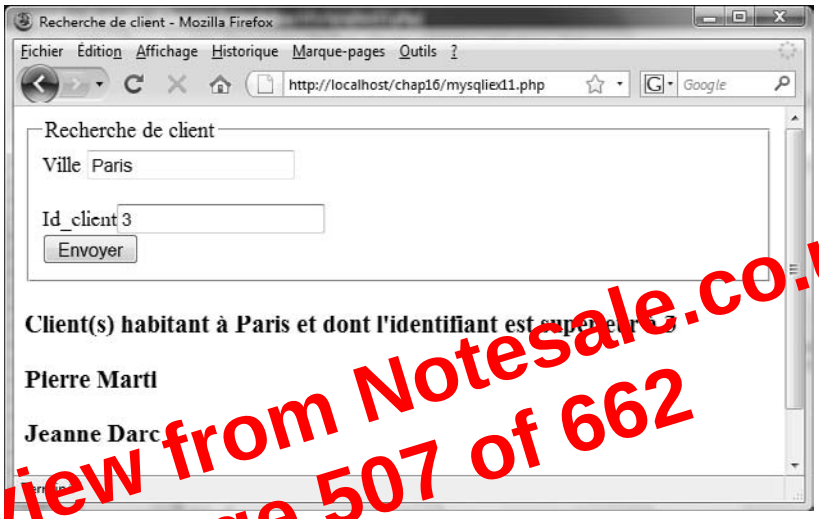


Figure 16-9
Résultats d'une requête préparée

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 car, dans ce cas, 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, signifie que si l'une des deux requêtes n'est pas exécutée, aucune ne l'est. L'intégrité de la base s'en trouve préservée.

Le langage SQL possède des commandes qui permettent de gérer les transactions, mais l'extension `mysqli` nous fournit des méthodes qui permettent de gérer les transactions sans y faire appel.

Classe mysqli_stmt : propriétés

`integer affected_rows`

Contient le nombre total de lignes concernées par la dernière requête.

`integer errno`

Contient un code erreur pour la dernière requête préparée.

`string error`

Contient la description de la dernière erreur.

`integer field_count`

Contient le nombre de colonnes dans la requête.

`integer insert_id`

Contient l'identifiant généré par la dernière requête `INSERT` pour la colonne `AUTO_INCREMENT`.

`integer num_rows`

Contient le nombre de lignes d'un résultat de requête préparée.

`integer param_count`

Contient le nombre de paramètres nécessaires dans la requête préparée.

Exercices

Tous les exercices ci-dessous portent sur la base de données `voitures` créée aux chapitres 13 et 14. Ils sont identiques à ceux du chapitre 15, mais vous devez les réaliser uniquement avec l'extension `mysqli` objet.

Exercice 1

Créez un script permettant d'afficher le contenu de la table `modele` dans un tableau XHTML. Les résultats doivent être triés par marque.

Exercice 2

Créez un formulaire permettant l'insertion de nouvelles données dans la table `modele`.

Exercice 3

Créez un formulaire permettant l'insertion simultanée des coordonnées d'une personne dans les tables `proprietaire` et `cartegrise`. Il doit contenir les zones de saisie des coordonnées de la personne et la liste des modèles d'une marque créée dynamiquement à partir de la saisie de la marque.

Exercice 4

Créez un formulaire de recherche permettant de retrouver tous les propriétaires d'un type de véhicule de marque et de modèle donnés. Affichez les résultats sous forme de tableau XHTML.

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 17-6 réalise la recherche et l'affichage dans un tableau XHTML 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 (repère ❶). Pour les lire, nous appelons la méthode `fetchObject()` pour l'objet `$result` (repère ❷) et récupérons la première ligne de résultat dans la variable `$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 XHTML (repère ❸). Les valeurs associées ne nous intéressent pas encore ici. Pour récupérer le tableau des données nous utilisons une boucle `do...while` (repère ❹), 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` (repère ❺).

Exemple 17-6. Lecture des données dans un objet

```
<!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 client</title>
<style type="text/css">
table {border-style:double;border-width:3px;border-color:red;
➥background-color:yellow;}
</style>
</head>
<body>
<?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(); ←2
  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) ←3
  {
    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>", $ligne->mail,"</td></tr>"; ←4
  }
  while ($ligne = $result->fetchObject()) ; ←5
  echo "</table>";
  $result->closeCursor();
  $idcom=null;
}
?>
</body>
</html>
```

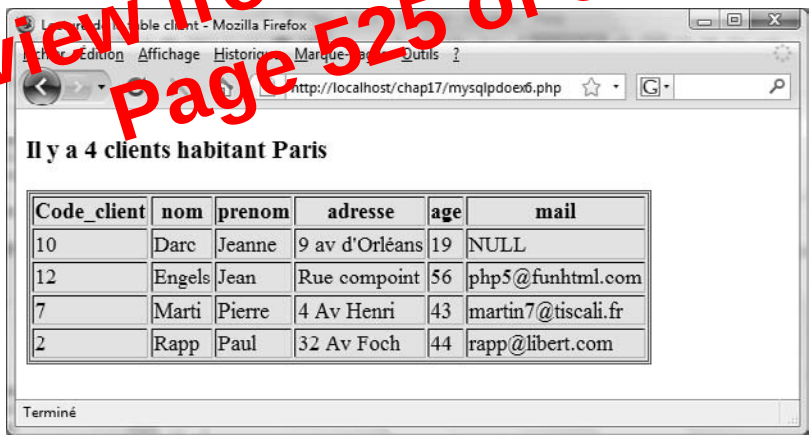



Figure 17-3

Lecture des noms des colonnes

Il est également possible de récupérer des résultats dans un objet avec la méthode `fetch()`, en utilisant la syntaxe suivante :

```
object $result->fetch(PDO::FETCH_OBJ)
```

Figure 17-6

Page de saisie du code client

La première inclusion de code PHP renvoie le client vers la page de saisie du code s'il a validé le formulaire sans avoir effectué de saisie (repère ❶). Rappelons, comme nous l'avons déjà vu par ailleurs, que cette partie de code PHP doit figurer en tête du fichier car elle utilise la fonction `header()` pour effectuer la redirection.

La suite du fichier comporte deux parties distinctes. La première crée dynamiquement un formulaire permettant la modification des données, quand à la seconde, elle enregistre ces mêmes données dans la base.

Lors du premier appel du fichier de l'exemple 17-9, la condition de l'instruction `if` (repère ❷) est nécessairement vérifiée car la variable `$_POST['modif']` n'existe pas encore. Elle correspond à la valeur associée au bouton `submit` du formulaire qui n'est pas encore créé. Le script génère une connexion au serveur MySQL pour y lire les coordonnées actuelles du client dont le code est contenu dans la variable `$code` issue de la page de saisie de l'exemple 17-8 (repère ❸).

Dans le but de compléter le formulaire avec les données actuelles, la requête SQL sélectionne toutes les colonnes de la table `client` dont l'identifiant client (colonne `id_client` de la table `client`) correspond à la valeur de la variable `$code` (repère ❹). Cela permet de ne saisir que les modifications éventuelles de coordonnées du client, sans devoir ressaisir l'ensemble. Ces coordonnées sont lues à l'aide de la méthode `fetch()` de l'objet `$result` de type `PDOStatement`. Elles sont alors contenues dans la variable `$coord` de type `array`. Pour afficher les coordonnées dans le formulaire, vous devez attribuer les valeurs de ces éléments aux attributs `value` des différents champs `<input />` (repère ❻).

La figure 17-7 montre un exemple de création dynamique de formulaire pour le client dont l'identifiant vaut 12. Le champ caché `code` du formulaire permet de passer la valeur du code client à la partie du script chargée de l'enregistrement des données modifiées (repère ❼).

L'envoi du formulaire utilise la deuxième partie du script, qui met à jour les données du visiteur dans la table `client` après avoir vérifié l'existence de valeurs pour les champs obligatoires du formulaire (repère ❽). Seules les colonnes `nom`, `adresse`, `ville` et `mail`

Les requêtes préparées

Nous avions déjà construit dynamiquement des requêtes SQL dans l'exemple précédent à partir d'informations saisies par l'utilisateur. 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, par exemple, pour des appels répétitifs avec des valeurs différentes. Une requête préparée se présente, entre autres, sous la forme suivante :

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

Dans laquelle les caractères `?` vont être remplacés par des valeurs quelconques en fonction des besoins du visiteur. C'est la méthode que nous avons employée au chapitre 16.

Ou encore avec des paramètres nommés :

```
SELECT prenom,nom FROM client WHERTE ville=:ville AND id_client=:id_client
```

C'est d'ailleurs cette méthode que nous allons utiliser ici.

Les paramètres nommés `:ville` et `:id_client`, dont les noms sont ici les mêmes que ceux des attributs de la table `client`, sont en fait arbitraires.

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 `PDO`. Cette méthode retourne un objet de type `PDOStatement` qui représente la requête préparée et qui est noté `$regprep` dans notre exemple.
2. Lier les paramètres dans l'ordre de leur apparition avec des valeurs ou des variables. Cette liaison peut se faire de plusieurs manières :
   – En créant un tableau associatif de la forme :

     ```
     $tab=array(':ville'=>$ville,'id_client'=>$id_client)
     ```

     les variables `$ville` et `$id_client` ayant déjà des valeurs à ce stade. Ce tableau sera ensuite passé en paramètre à la méthode `execute()` détaillée ci-après.
   – En appelant la méthode `bindParam()` des objets `PDOStatement` pour chaque paramètre selon le modèle :

     ```
     bindParam(':ville',$ville,PDO::PARAM_STR)
     ```

     dans lequel le troisième paramètre désigne le type de la variable et peut prendre les valeurs `PDO::PARAM_STR` pour une chaîne et `PDO::PARAM_INT` pour un entier.
3. Exécuter la requête en appelant la méthode `execute()` de l'objet `PDOStatement` avec le tableau comme paramètre, s'il a été créé à l'étape 2, ou sinon sans paramètre.
4. Pour les requêtes préparées qui retournent des résultats, comme celles qui contiennent la commande `SELECT`, il faut ensuite lier les résultats à des variables PHP, avec la méthode `bindColumn()` selon les modèles :

   ```
   $regprep->bindColumn(1,$prenom)
   ```

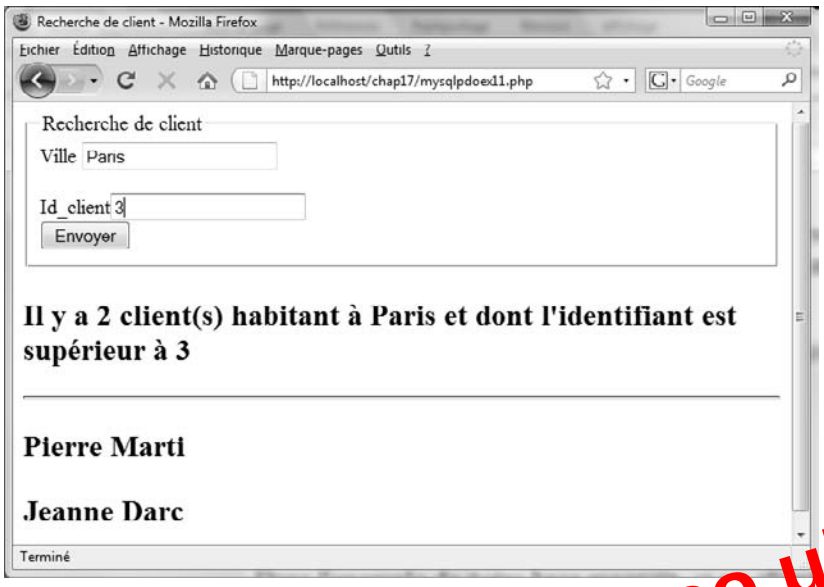


Figure 17-9

Résultats d'une requête préparée

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()`.

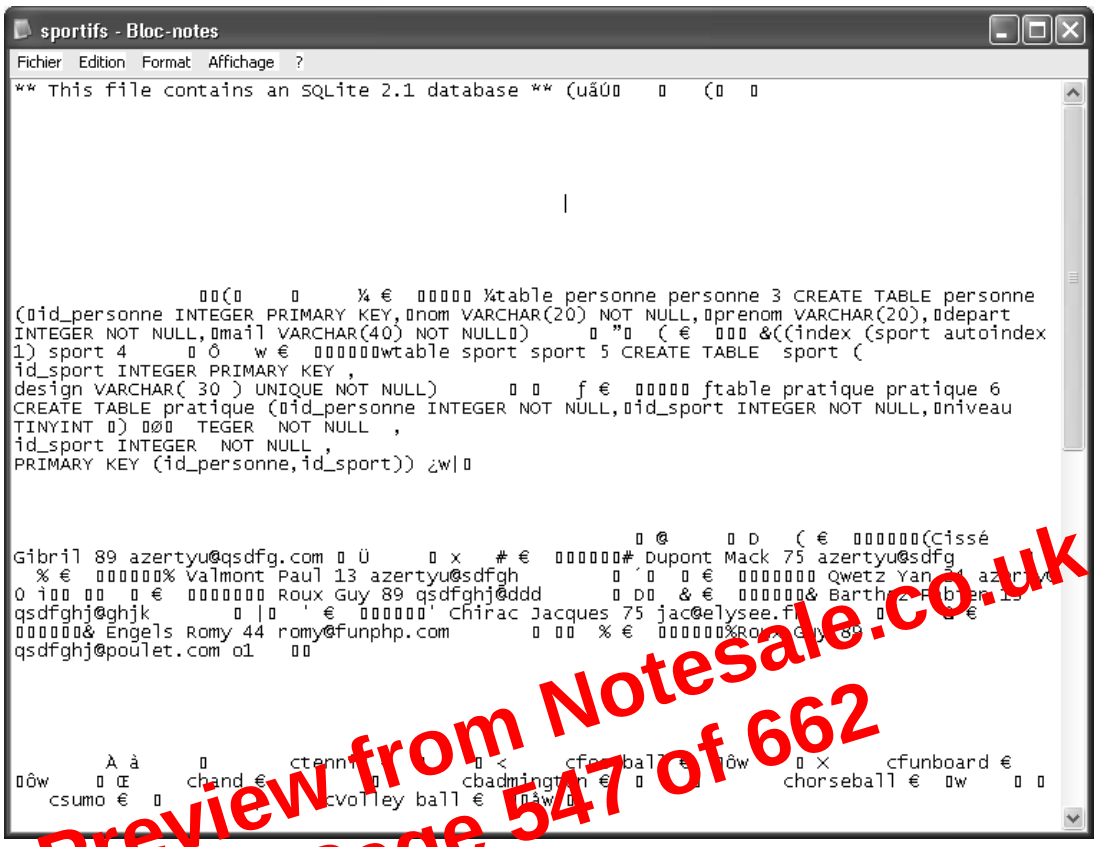


Figure 18-1

Structure d'un fichier de base SQLite

SQLite supporte l'essentiel des fonctionnalités du langage SQL, à l'exception des suivantes :

- `FOREIGN KEY` ;
- Support complet de `ALTER TABLE` ;
- support intégral des triggers ;
- transactions imbriquées ;
- `COUNT (DISTINCT X)` ;
- `VIEW` en écriture ;
- `GRANT` et `REVOKE` (pas de gestion des droits d'accès).

D'après les tests publiés sur le site SQLite (*http://www.sqlite.org*), cette base est jusqu'à trois fois plus rapide que MySQL dans les accès en lecture de données. En revanche, le fait

que toute la base soit contenue dans un seul fichier entraîne son verrouillage total lors des accès en écriture. Cela ralentit les opérations dans le cas d'accès concourants à la base.

Dans sa version actuelle 2.8, SQLite est dite « typeless », c'est-à-dire qu'elle n'oblige pas à effectuer une déclaration du type de chaque colonne lors de la création des tables, contrairement à MySQL. Vous pouvez cependant conserver vos habitudes pour l'écriture des requêtes SQL de création des tables en définissant un type pour chaque colonne.

Lors des opérations de comparaison et de tri, SQLite établit une différence entre les valeurs numériques et les textes. Les colonnes déclarées avec un type contenant les mots `CHAR`, `TEXT` ou `BLOB` sont considérées comme du texte. Toutes les autres possibilités, y compris les types personnels non normalisés par le SQL, sont considérées comme numériques.

Dans le code suivant :

```
CREATE TABLE matable(nom LETEXTE, age SONAGE)
```

La colonne `nom` est considérée comme du texte et la colonne `age` comme un nombre en 32 bits.

Ce typage des données étant peu rigoureux pour une base de données, la version 3.0 de SQLite prévoit de reconnaître explicitement les types suivants : `NULL`, `INTEGER`, `REAL`, `TEXT` et `BLOB`, ce qui la rapprochera des pratiques habituelles. Dans la suite du chapitre, vous créerez les tables de votre base en déclarant le type des colonnes.

De nombreuses tables comportent une clé primaire, qui est un entier auto-incrémenté d'une unité à chaque nouvelle insertion. SQLite ne reconnaît pas l'attribut `AUTO_INCREMENT` utilisé avec MySQL pour créer ce type de colonne. Pour obtenir le même résultat, il faut déclarer la colonne avec le type `INTEGER PRIMARY KEY`, comme ci-dessous :

```
CREATE TABLE personne(id INTEGER PRIMARY KEY,nom VARCHAR(20) );
```

L'insertion de la valeur `NULL` dans la colonne `id` permet cette incrémentation automatique.

L'interface SQLiteManager

Comme pour MySQL avec phpMyAdmin, des programmeurs bénévoles ont créé des interfaces de gestion en ligne des bases de données SQLite. Il en existe plusieurs, comme phpSQLiteAdmin et SQLiteManager. Cette dernière version est livrée d'office avec Wampserver.

Si votre hébergeur n'offre pas cette interface, vous pouvez l'installer vous même puisqu'il s'agit de fichiers PHP (voir le site *http://www.sqlitemanager.org*). La figure 18-2 montre la page d'accueil de SQLiteManager 1.2, à partir de laquelle vous pouvez créer une base et les tables qui la composent. L'application sur laquelle reposent les exemples de ce chapitre a pour but de créer un site de contact entre sportifs partageant la même passion et leur permettant de trouver des partenaires éventuels. Ce modèle pourrait être

Nous aurons, par exemple, le code d'ouverture suivant :

```
<?php
if ($id_base = sqlite_open ("ma_base", 0666, $erreur)) echo "OK" ;
else echo $erreur ;
?>
```

Si le premier paramètre est la chaîne `":memory:"`, la base est créée dans la mémoire vive du serveur. Cette possibilité permet d'effectuer des traitements provisoires. La base créée est effacée à la fin du script.

Il est également possible de réaliser une connexion persistante à l'aide de la variante `sqlite_popen()`, dont la syntaxe est similaire :

```
resource sqlite_open (string nom_base [, integer mode [, string $erreur]])
```

Pour fermer la base, vous devez appeler la fonction `sqlite_close()`, dont la syntaxe est la suivante :

```
void sqlite_close(resource $id_base)
```

et dont le seul paramètre est l'identifiant d'ouverture de la base `$id_base`. L'appel de cette fonction doit être fait le plus vite possible dans le script afin de libérer le fichier contenant la base qui est verrouillée en écriture à chaque accès.

Envoi de requêtes

L'étape suivante consiste principalement dans l'envoi à la base de requêtes écrites à l'aide du langage SQL. Les connaissances acquises au chapitre 14 vont vous permettre d'être tout de suite opérationnel. Vous n'utiliserez que certaines possibilités supplémentaires qu'offre SQLite par rapport à MySQL.

Pour envoyer une requête à la base, vous utiliserez le plus souvent la fonction `sqlite_query()`, dont la syntaxe est la suivante :

```
resource sqlite_query (resource $id_base, string $requete)
```

La variable `$id_base` est l'identifiant d'ouverture de la base et `$requete` la chaîne contenant la ou les commandes SQL appropriées.

Cette fonction retourne une variable de type `resource` si la requête SQL retourne des lignes de données, cette variable étant utilisable pour lire l'ensemble des données à l'aide de fonctions spécialisées, comme c'est le cas pour MySQL. Si la requête ne concerne pas une commande SQL fournissant des données, la fonction retourne une valeur booléenne `TRUE` si l'opération est réussie et `FALSE` dans le cas contraire. De plus, si la requête n'est pas exécutée, la fonction retourne encore `FALSE`, ce qui permet de vérifier sa bonne exécution à l'aide d'une instruction `if`.

Un script d'interrogation de la base a une structure identique à celle de l'exemple 18-1.

Exemple 18-1. Envoi de requête SQL

```
<?php
if ($id_base = sqlite_open ("C:/wamp/apps/sqlitemanager1.2.0/sportifs", 0666,
➥$erreur))
{
  $requete = "SELECT * FROM personne" ;
  if ($result = sqlite_query ($id_base, $requete))
  {
    echo "Requête effectuée <br />"  ;
    //Lecture des résultats
  }
  else echo " La requête n'a pas aboutie" ;
}
else echo $erreur ;
?>
```

Pour que le résultat ne soit pas mis en buffer sur le serveur avant son utilisation par les fonctions de lecture, vous pouvez utiliser la fonction `sqlite_unbuffered_query()`, dont la syntaxe est la suivante :

```
resource sqlite_unbuffered_query(resource $id_base,string $requete)
```

Elle utilise les mêmes paramètres que la précédente. Si elle consomme moins de mémoire sur le serveur, elle présente cependant l'inconvénient d'obliger à utiliser les données du résultat avant tout autre envoi de requête, faute de quoi les résultats sont perdus.

Pour envoyer une requête qui ne retourne pas de résultats, par exemple, une requête de création de table, d'insertion ou de mise à jour, il peut être plus efficace d'utiliser la fonction `sqlite_exec()`, dont la syntaxe est la suivante :

```
boolean sqlite_exec(resource $id_base,string $requete)
```

Elle exécute la requête `$requete` sur la base identifiée par `$id_base` et retourne `TRUE` si la requête est bien réalisée et `FALSE` dans le cas contraire. L'interface SQLiteManager n'étant pas encore exempte de bogues, il peut être plus sécurisant de créer les tables de la base `sportifs` à l'aide d'un script à n'exécuter qu'une seule fois afin d'éviter tout risque d'erreur, car la clause `IF NOT EXISTS` de MySQL n'existe pas dans SQLite. L'exemple 18-2 crée les tables de la base `sportifs` en envoyant les requêtes SQL à la base avec la fonction `sqlite_exec()` (repères ❶, ❷ et ❸).

Exemple 18-2. Création de table à l'aide d'un script

```
<?php
if ($id_base = sqlite_open('C:/wamp2/apps/sqlitemanager1.2.0/sportifstest', 0666,
➥$erreur))
{
  echo "La base sportifs est ouverte";
  //*****************************
  //Création de la table personne
  $req_personne="CREATE TABLE personne (
```

Chacun de ces objets possède des méthodes dont les actions sont similaires à celles des fonctions utilisées à la section précédente. Vous allez donc réécrire les différents exemples de cette section en utilisant ces objets et ces méthodes.

Casse des noms de méthode
Tous les noms des méthodes de ces objets sont insensibles à la casse.

Accès à la base

Pour accéder à une base SQLite, vous devez d'abord créer un objet `SQLiteDatabase` en utilisant le mot-clé `new` selon la syntaxe suivante, qui appelle le constructeur de la classe avec les mêmes paramètres que la fonction `sqlite_open()` :

```
$db=new SQLiteDatabase("C:/wamp/apps/sqlitemanager1.2.0/sportifs", 0666, $erreur);
```

Si la base n'existe pas, elle est créée. Le premier paramètre, qui indique le nom et le chemin d'accès à la base, est obligatoire. Si le premier paramètre est la chaîne `":memory:"`, la base est créée en mémoire vive du serveur. Pour fermer la base, vous détruisez la variable `$db` en appelant la fonction `unset()` aussitôt que vous n'en avez plus besoin, donc juste après l'envoi de la dernière requête du script.

Envoi de requêtes

L'objet `$db` qui vient d'être créé va être utilisé pour envoyer des requêtes à la base avec la méthode `query()`. Cette dernière est équivalente à la fonction `sqlite_query()`, dont le seul paramètre est la chaîne de la requête SQL. Pour les requêtes de sélection de données, cette méthode retourne un objet `SQLiteResult`, dont l'utilisation est détaillée à la section suivante.

Si la requête n'est pas exécutée, la méthode retourne la valeur booléenne `FALSE`. L'exemple 18-14 représente la forme générale d'un script d'ouverture de la base et d'envoi d'une requête. Il équivaut à l'exemple 18-1 mais avec la méthode objet.

Exemple 18-14. Script type d'accès à la base

```
<?php
if ($db=new SQLiteDatabase ("C:/wamp/apps/sqlitemanager1.2.0/sportifs", 0666,
➥$erreur))
{
  $requete = "SELECT nom,prenom FROM personne" ;
  if ($result=$db->query($requete))
  {
    unset($db);
    echo " La requête a été effectuée " ;
    //Lecture des résultats
  }
  else echo " La requête n'a pas aboutie" ;
}
```

variable `$mail` contenant l'information recherchée, qui est ensuite affichée (repère ❸). Si vous ne vous intéressez qu'à la première valeur, la boucle `while` est inutile, et vous pouvez écrire simplement le code suivant :

```
$mail=$result->fetchSingle();
echo "<h4>Mail : $mail </h4>";
```

Cela peut être utile pour une requête dont vous savez par avance qu'elle ne retourne qu'une seule ligne, comme la recherche du nom d'un utilisateur dont vous connaissez l'identifiant unique.

☛ Exemple 18-21. Lecture d'une seule colonne

```
<?php
if ($db=new SQLiteDatabase("C:/wamp/apps/sqlitemanager1.2.0/sportifs"))
{
  $requete = "SELECT mail FROM personne WHERE depart='13'" ; ←❶
  if($result=$db->Query($requete)) ←❷
  {
    while($mail=$result->fetchSingle()) echo "<h4>Mail : $mail </h4>"; ←❸
  }
}
else echo $erreur ;
?>
```

Pour lire une colonne précise dans la ligne en cours d'un résultat, vous disposez de la méthode `column()`, qui s'applique à l'objet `$result` dont la syntaxe est la suivante :

```
divers $result->column(string nom_col | integer num_col)
```

L'unique paramètre précise le nom de la colonne ou de son alias avec une chaîne ou sa position dans la requête SQL avec un entier, la première colonne ayant le numéro 0.

Si la requête est :

```
$requete="SELECT nom,prenom,mail AS adresse_mail FROM  personne";
```

le code suivant :

```
$mail=$result->column('adresse_mail');
```

ou encore :

```
$mail=$result->coluvvmn(2);
```

permet de ne lire que la colonne des adresses e-mail.

Informations sur le résultat

À partir d'un objet `SQLiteResult`, vous pouvez obtenir des informations sur les données retournées par la requête SQL. La méthode suivante :

```
string $result->fieldName(integer N)
```

Exemple 18-23. Création de fonctions SQL personnalisées

```
<?php
function present($mot) ←❶
{
  return ucfirst(strtolower($mot));
}
if ($db=new SQLiteDatabase("C:/wamp/apps/sqlitemanager1.2.0/sportifs",  0666,
➥$erreur))
{
  $db->createFunction('initiale','present',1); ←❷
  $requete = "SELECT upper(nom) AS nom, initiale(prenom) AS prenom,
  ➥php('strtolower',mail) AS mail  FROM personne" ; ←❸
  if ($result = $db->query ($requete))
  {
    echo "<h3>Liste des personnes enregistrées</h3>";
    while($ligne=$result->fetch(SQLITE_BOTH))
    {
      echo "Nom : ", $ligne[0] ," Prénom : ", $ligne[1] , " Mail : ", $ligne[2],
      ➥"<br />"; ←❹
      echo "Nom : ", $ligne['nom'] ," Prénom : ", $ligne['prenom'], " Mail : ",
      ➥$ligne['mail'] ,"<br />"; ←❺
    }
  }
  else echo " La requête n'a pas abouti";
  unset($db);
}
else echo $erreur ;
?>
```


L'objet SQLiteException

Vous avez déjà rencontré la classe `Exception`, qui permet la gestion des exceptions survenant dans un script (voir le chapitre 3). SQLite dispose d'une classe spéciale, nommée `SQLiteException`, dérivée de la précédente. Elles possèdent toutes deux les mêmes propriétés et les mêmes méthodes. Il n'y a donc rien de nouveau dans l'utilisation de cette classe par rapport à ce que vous avez déjà réalisé.

Pour illustrer l'emploi d'un objet `SQLiteException` lors d'un accès à la base `sportifs`, le code de l'exemple 18-24 tente d'accéder à une table qui n'existe pas (elle est appelée `personnes` alors que seule la table `personne` a été créée). Pour masquer les messages d'erreur à l'utilisateur, vous appelez la fonction `error_reporting()`, qui bloque l'affichage de tous ces messages (repère ❶). L'ensemble du code de lecture de la table est inclus dans un bloc `try` (repère ❷).

Comme l'envoi de la requête par la méthode `query()` retourne la valeur `FALSE` (repère ❸), un message est affiché, et une exception est déclenchée par l'instruction `throw`. Cette dernière provoque la création d'un objet de type `SQLiteException`. Vous attribuez à cet objet un texte de message d'erreur et un code d'erreur, qui correspondent respectivement

Exemple 19-5. Modification et enregistrement des données

```
<?php
$xml=simplexml_load_file("biblio3.xml"); ←❶
//Modification d'un élément et d'un attribut
$xml->livre[0]->titre="La haine de l'Occident "; ←❷
$xml->livre[0]->date="2008";  ←❸
//Affichage des données du fichier
foreach($xml->livre as $cle=>$val) ←❹
{
  static $i=1;
  echo ucfirst($cle)," $i : $val->titre de $val->auteur paru en $val->date<hr />";
  $i++;
}
//Enregistrement des modifications
$chxml= $xml->asxml("biblio3a.xml"); ←❺
if($chxml) echo "Enregistrement réalisé"; ←❻
?>
```

Après l'exécution du script de l'exemple 19-5, le premier élément `<livre>` du fichier XML est le suivant :

```
<livre editeur="FAYARD" prix="20.00">
    <titre>La haine de l'Occident </titre>
    <auteur>Jean Ziegler</auteur>
    <date>2008</date>
</livre>
```


Recherche dans un fichier

L'extension SimpleXML permet d'effectuer des recherches dans un fichier XML grâce à la méthode `xpath()` des objets de type `simplexml_element`.

La syntaxe de la méthode `xpath()` est la suivante :

```
array xpath(string requete_xpath)
```

La fonction retourne un tableau de tous les contenus d'éléments ou d'attributs qui correspondent à la requête qui lui est passée en paramètre. Pour effectuer une recherche, vous devez décrire l'arborescence permettant de parvenir à l'information recherchée.

Considérant que vous effectuez des recherches de titres, d'auteurs et d'éditeurs dans le fichier `biblio4.xml`, vous pouvez écrire les exemples de requêtes suivants :

- Pour trouver tous les titres de livre (seuls les éléments fils de l'élément `<livre>` ont un contenu textuel dans le fichier) :

```
$xml->xpath("/biblio/ouvrage/livre/titre");
$xml->xpath("//ouvrage/livre/titre");
$xml->xpath("//livre/titre");
```

- Pour trouver les auteurs, vous remplacez `titre` par `auteur`. Faites de même pour les dates.

```
  <tr>
    <td> Date :
     <input type="text" name="date" />
    </td>
  </tr>
  <tr>
  <td>
  <input type="submit" name="envoi" value="OK"/>
  </td>
  </tr>
  </tbody></table>
  </fieldset>
  </form>
</body>
</html>

<?php
if(isset($_POST['envoi'])&& !empty($_POST['titre'])&& !empty($_POST['auteur'])&&
➥!empty($_POST['date'])) ←❶
{
  $titre= htmlspecialchars($_POST['titre']); ←❷
  $auteur= htmlspecialchars($_POST['auteur']); ←❸
  $date= htmlspecialchars($_POST['date']); ←❹
  if(!file_exists("biblio5.xml")) ←❺
  {
    $chxml= "<?xml version=\"1.0\" encoding=\"iso-8859-1\" standalone=\"yes\"?>
    ➥\n<biblio>\n <livre>\n <titre>$titre</titre>\n <auteur>$auteur</auteur>
    ➥\n <date>$date</date>\n</livre>\n</biblio>"; ←❻
  }
  else
  {
  $xml=simplexml_load_file("biblio5.xml"); ←❼
  $chxml = $xml->asxml(); ←❽
  $chxml = str_replace("</biblio>", "", $chxml); ←❾
  $chxml.= "<livre>\n <titre>$titre</titre>\n <auteur>$auteur</auteur>\n
  ➥<date>$date</date>\n</livre>\n</biblio>"; ←❿
  }
  $verif=file_put_contents("biblio6.xml",$chxml); ←⓫
}
?>
```


Relations entre XML et une base MySQL

Les fichiers XML étant lisibles par de nombreux médias, ils constituent un excellent moyen d'échange de données entre différents systèmes. En particulier, vous allez les utiliser pour stocker les données d'une table MySQL dans un format structuré. Cela vous permettra de les transmettre à d'autres applications, qui seraient incapables d'accéder directement aux données d'une base MySQL. Vous verrez également les moyens permettant de réaliser l'opération inverse.

20

Le framework PEAR

En informatique, et particulièrement en POO, il est d'usage de dire qu'il ne faut pas réinventer la roue. En effet, à quoi bon réécrire du code qui a déjà été fait et testé par d'autres, pour parvenir au même résultat. C'est cette idée qui a prévalu lors de la création des frameworks et de PEAR en particulier. PEAR est un ensemble de très nombreux modules adaptés aux situations les plus diverses que l'on peut rencontrer lors de la création de sites Web. Chaque module ou package est un ensemble de scripts écrits en PHP. L'arrivée de PHP 5 et de son modèle objet plus élaboré que celui, très primaire, de PHP 4, permet à PEAR de proposer désormais, pour chaque module, des classes dotées de nombreuses méthodes qui vous permettent d'utiliser aisément, et surtout d'adapter à vos besoins particuliers, chaque module. Il existe d'autres frameworks Open Source que PEAR comme Zend, Symphony… mais notre choix s'est porté sur PEAR non seulement parce que c'est l'un des plus anciens, qu'il est bien établi et constamment mis à jour, mais surtout parce qu'il nous semble plus facile d'accès.

Installer PEAR

Il est possible de récupérer les différents packages de PEAR sur le site *http://pear.php.net*, mais il existe un moyen plus pratique qui consiste à utiliser un installeur qui permet non seulement d'installer les packages de base, mais aussi de gérer la recherche et l'installation des packages en fonction de vos besoins ainsi que de les mettre à jour en un clic. Pour cela, il vous faut tout d'abord récupérer le fichier nommé `go-pear.php` soit sur le site de PEAR cité plus haut, soit dans le répertoire `wamp/apps` dans lequel vous avez installé le serveur local Wampserver2. Transférez-le ensuite sur le serveur de votre site distant via un logiciel FTP, comme pour les fichiers habituels, puis lancez-le à partir d'un navigateur. Vous obtiendrez la page présentée à la figure 20-1. Laissez-vous alors guider et fournissez

21

Travaux personnels

Plutôt que de clore l'ouvrage par des études de cas, souvent difficiles à suivre pour qui ne les pas écrites, nous vous proposons quatre thèmes de travaux personnels qui constituent en quelque sorte des sujets de mémoire ou de fin d'études sur PHP. Chacun d'eux consiste à créer un site de complexité croissante. Les uns aux autres, les descriptifs sont de moins en moins détaillés afin de vous mettre progressivement dans une situation réelle de création de site.

Les corrigés de ces travaux personnels sont téléchargeables depuis le site des éditions Eyrolles *(http://www.editions-eyrolles.com)* sur la page dédiée à l'ouvrage. Les sites correspondants peuvent en outre être consultés en fonctionnement réel sur le site Web *http://www.funhtml.com/php5*.

Démarche à suivre

Pour chaque projet vous devrez suivre la démarche suivante :

1. Lire attentivement le cahier des charges proposé.
2. Décomposer le site en différents modules, auxquels correspondront autant de pages.
3. Établir un schéma général de fonctionnement mettant en relief les liens entre ces différentes pages.
4. Tracer les grandes lignes de l'organisation de chaque page sous forme de schéma, en séparant le code HTML du code PHP.
5. Écrire le code de chaque partie sans trop entrer dans les détails de design dans un premier temps. Le design relevant plus du designer que du programmeur, il ne vous concerne pas ici.

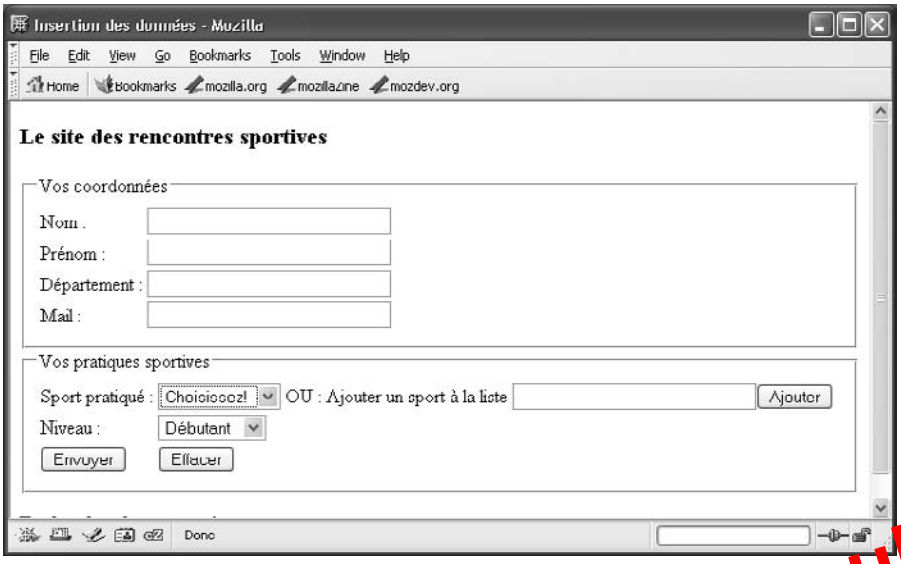


Figure 21-1

Le formulaire d'inscription

- Lien vers la page d'accueil.
- Lien vers la page d'inscription.
- Script traitant les informations saisies et affichant la liste des partenaires correspondants dans un tableau XHTML. Les données saisies sont réaffichées dans le formulaire afin de faciliter une éventuelle nouvelle recherche de l'utilisateur.

Le formulaire de recherche doit ressembler à celui illustré à la figure 21-2.

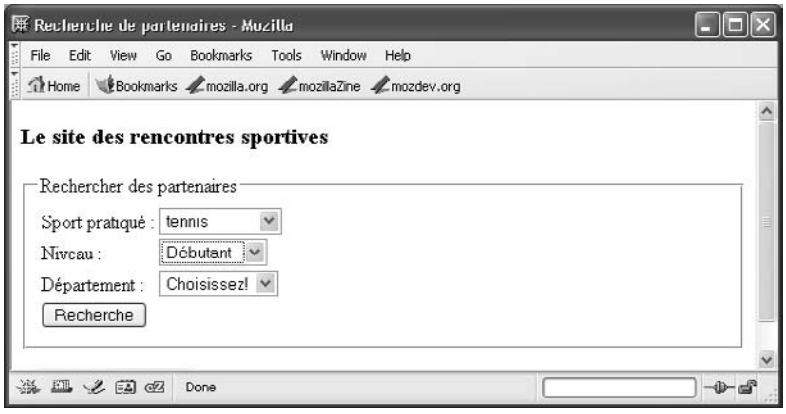


Figure 21-2

Le formulaire de recherche

Votre travail

Comme vous n'allez pas réinventer le concept de commerce en ligne et partir de zéro, vous vous inspirerez de ce qui existe déjà. Vous consulterez pour cela des sites de vente en ligne tels que *eyrolles.com* pour en étudier le fonctionnement.

Il vous faudra en dégager les points essentiels et n'en retenir que ce qui correspond au cas relativement simple de votre client.

En particulier, le nombre d'articles vendus par cette PME est relativement limité comparativement à ceux du site d'Eyrolles. Cela peut entraîner des simplifications dans la structure de la base, comme le fait de ne pas créer de table spéciale pour enregistrer le nom des marques des produits en magasin.

Pour accéder à MySQL, utilisez de préférence l'extension mysqli (voir le chapitre 16) ou, encore mieux, pour les courageux, réalisez une version mysqli puis une version PDO (voir le chapitre 17).

Fonctionnement du site

Le site répond aux conditions suivantes :

- Dans la page d'accueil, le client recherche un type de produit dans une des catégories `informatique`, `vidéo` et `divers` (pour les accessoires et consommables). Un tri par marque et par prix est proposé pour l'affichage des résultats.
- Les résultats de la recherche sont affichés en respectant le critère de tri. Chaque produit est suivi d'un lien permettant sa sélection.
- La sélection d'un produit entraîne sa mise en panier.
- Après chaque sélection, le client peut soit rechercher un autre produit, soit terminer sa commande.
- Dans ce dernier cas, l'ensemble de sa commande est affichée, et vous lui demandez ses coordonnées.
- Si le client n'est pas encore enregistré, il saisit ses coordonnées complètes. Son adresse e-mail et un mot de passe lui serviront à s'identifier par la suite. Il saisit également le cas échéant l'adresse de livraison, qui peut être différente de l'adresse du client. Ces informations sont stockées à l'aide de sessions pour être transmises aux pages suivantes de la procédure d'achat.
- Si le visiteur est déjà client, il ne saisit que son e-mail et son mot de passe. L'authenticité de ces informations est vérifiée dans la base, et les coordonnées complètes du client sont récupérées. Ces coordonnées sont utilisées pour remplir automatiquement un formulaire identique à celui du visiteur non enregistré. Ces informations sont également stockées à l'aide de sessions.
- La phase de paiement par carte bancaire n'étant pas réalisable sans une convention bancaire, vous vous contentez de demander la saisie d'un numéro de carte et d'utiliser

P

U