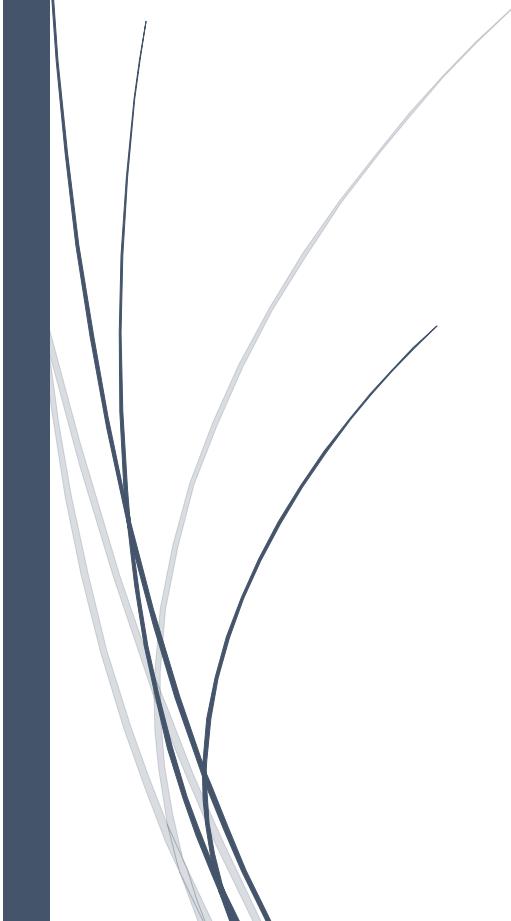




# **PROGRAMMATION WEB**

**ENEAM-UAC**



Dr Matine OUSMANE

<b>SEMESTRE N° : 2</b>					
<b>Titre de l'UE : PROGRAMMATION WEB</b>					
<b>Code :</b>	<b>Crédits :</b>				
<b>Objectif Général :</b>					
<p>Ce cours vise à permettre aux apprenants de comprendre les concepts du web, le développement des applications web et leur mise en ligne dans un domaine.</p> <p>Il fournit les différentes connaissances pour le développement et la dynamisation des applications Web. Il permettra de maîtriser les principaux concepts et les méthodes visant à concevoir des applications web performantes.</p>					
<b>Objectifs Spécifiques :</b>					
<p>Au terme des activités d'enseignement-apprentissage proposés dans le cadre de l'UE, l'apprenant est en mesure de :</p> <ul style="list-style-type: none"> <li>• Identifier dans son milieu, un projet viable de développement Web ;</li> <li>• Maîtriser les bases de la programmation en Javascript et en php ;</li> <li>• Maîtriser la gestion des évènements au niveau du web ;</li> <li>• Maîtriser le développement Web à l'aide de quelques framework (ReactJS et Laravel ou Symfony).</li> </ul>					
<b>Structure de l'UE :</b>					
<b>Contenu de l'UE :</b>					
<ul style="list-style-type: none"> <li>• Programmation côté client avec Javascript ;</li> <li>• Programmation serveur avec PHP brute ;</li> <li>• Atelier de formation sur Framework ReactJS ;</li> <li>• Atelier de formation sur Framework Laravel ou Symfony</li> </ul>					
<b>Modes d'évaluation</b>					
CC <input checked="" type="checkbox"/>	ET <input checked="" type="checkbox"/>	PP <input type="checkbox"/>			
<b>Méthodes d'enseignement</b>					
CM <input checked="" type="checkbox"/>	TD <input checked="" type="checkbox"/>	TP <input checked="" type="checkbox"/>	SC <input checked="" type="checkbox"/>	JR <input type="checkbox"/>	SP <input type="checkbox"/>
<ol style="list-style-type: none"> <li>1. <a href="#"><u>Références Bibliographiques :</u></a></li> <li>2. <a href="http://cours-fad-public.ensg.eu/pluginfile.php/1572/mod_resource/content/1/1-Introduction%20au%20fonctionnement%20du%20web-1.pdf"><u>http://cours-fad-public.ensg.eu/pluginfile.php/1572/mod_resource/content/1/1-Introduction au fonctionnement du web-1.pdf</u></a></li> <li>3. <a href="https://www.geeksforgeeks.org/difference-between-static-and-dynamic-web-pages/"><u>https://www.geeksforgeeks.org/difference-between-static-and-dynamic-web-pages/</u></a></li> <li>4. <a href="https://wpamelia.com/static-vs-dynamic-website/"><u>https://wpamelia.com/static-vs-dynamic-website/</u></a></li> <li>5. <a href="https://openclassrooms.com/fr/courses/6175841-apprenez-a-programmer-avec-javascript/"><u>https://openclassrooms.com/fr/courses/6175841-apprenez-a-programmer-avec-javascript/</u></a></li> <li>6. <a href="https://developer.mozilla.org/fr/docs/Web/Guide/AJAX"><u>https://developer.mozilla.org/fr/docs/Web/Guide/AJAX</u></a></li> <li>7. <a href="https://fr.reactjs.org/docs/getting-started.html"><u>https://fr.reactjs.org/docs/getting-started.html</u></a></li> <li>8. <a href="https://api.jquery.com/"><u>https://api.jquery.com/</u></a></li> <li>9. <a href="https://www.php.net/docs.php"><u>https://www.php.net/docs.php</u></a></li> <li>10. <a href="https://openclassrooms.com/fr/courses/1959476-administrez-vos-bases-de-donnees-avec-mysql"><u>https://openclassrooms.com/fr/courses/1959476-administrez-vos-bases-de-donnees-avec-mysql</u></a></li> <li>11. <a href="https://openclassrooms.com/fr/courses/4670706-adoptez-une-architecture-mvc-en-php"><u>https://openclassrooms.com/fr/courses/4670706-adoptez-une-architecture-mvc-en-php</u></a></li> <li>12. <a href="https://laravel.com/"><u>https://laravel.com/</u></a></li> <li>13. <a href="https://symfony.com/"><u>https://symfony.com/</u></a></li> <li>14. <a href="https://openclassrooms.com/fr/courses/6175841-apprenez-a-programmer-avec-javascript/"><u>https://openclassrooms.com/fr/courses/6175841-apprenez-a-programmer-avec-javascript/</u></a></li> <li>15. <a href="https://fr.reactjs.org/docs/getting-started.html"><u>https://fr.reactjs.org/docs/getting-started.html</u></a></li> <li>16. <a href="https://api.jquery.com/"><u>https://api.jquery.com/</u></a></li> <li>17. <a href="https://laravel.com/docs/8.x/installation"><u>https://laravel.com/docs/8.x/installation</u></a></li> <li>18. <a href="https://laravel.com/docs/8.x/configuration"><u>https://laravel.com/docs/8.x/configuration</u></a></li> <li>19. <a href="https://laravel.sillo.org/cours-laravel-8-les-bases-installation-et-organisation/"><u>https://laravel.sillo.org/cours-laravel-8-les-bases-installation-et-organisation/</u></a></li> <li>20. <a href="https://laravel.com/docs/8.x/controllers"><u>https://laravel.com/docs/8.x/controllers</u></a></li> </ol>					

## **PLAN DU COURS**

Introduction

I- Programmation en Javascript

II- Programmation en PHP

III- Atelier de formation sur ReactJS

IV- Atelier de formation sur Framework Laravel ou Symfony

# **LE JAVASCRIPT**

## **- en 25 chapitres -**

OUSMANE Matine

# SOMMAIRE

<b>INTRODUCTION</b>	<b>6</b>
<b>1.GENERALITES</b>	<b>7</b>
1.1.Le langage Javascript	7
1.2.Côté pratique	7
1.3.Détails techniques	7
<b>2.LE LANGAGE</b>	<b>8</b>
2.1.Incorporation du code	8
2.2.Spécificités du langage	8
<b>3.LES STRUCTURES DE DONNEES</b>	<b>10</b>
3.1.Les données constantes	10
3.2.Les variables en JS	10
<b>4. LES OPERATEURS</b>	<b>14</b>
4.1.Les opérateurs de calcul	14
4.2.Les opérateurs de comparaison	14
4.3.Les opérateurs associatifs	15
4.4.Les opérateurs logiques	15
4.5.Les opérateurs d'incrémentation	15
<b>5. LES FONCTIONS</b>	<b>16</b>
5.1.Définition	16
5.2.Déclaration	16
5.3.Les fonctions dans l'en-tête	16
5.4.L'appel d'une fonction	16
5.5.Fonctions manipulant des valeurs	17
5.6.Variables locales et variables globales	18
<b>6. LES STRUCTURES DE CONTROLE</b>	<b>19</b>
6.1.Les structures algorithmiques	19
6.2.La structure séquentielle	19
6.3.Les instructions conditionnelles	19
6.4.Les instructions itératives	21
6.5.Interrompre une boucle	22
6.6.Exercice	22
<b>7. LES BOITES DE MESSAGE</b>	<b>23</b>
7.1.Généralités	23
7.2.Alert()	23
7.3.Prompt()	23
7.4.Confirm()	24
7.5.Exercice : Confirmation et vérification	24
<b>8. NOTION OBJET</b>	<b>26</b>
8.1.Le concept objet	26
8.2.Création d'un objet	26
8.3.Accès aux propriétés et aux méthodes	26
8.4.L'objet this	27

<b>9. LES FORMULAIRES</b>	28
9.1.Généralités	28
9.2. Champ de texte	29
9.3.Cases à sélectionner	31
9.4.Liste de sélection	33
9.5.Bouton	34
9.6.Contrôle caché	36
9.7.Un formulaire complet	38
9.8.Les objet du formulaire	39
9.9.Exercice	40
<b>10. LES EVENEMENTS</b>	41
10.1.Généralités	41
10.2.Le clic de souris	42
10.3.Le chargement	42
10.4.Le passage de la souris	44
10.5.Le focus	45
10.6.Les changements	46
10.7.La sélection	46
10.8.L'envoi	47
10.9.Le reset	47
10.10.L'utilisation de l'objet document	47
10.11.Exemple concret	48
10.12.Exercice	49
<b>11. L'OBJET ARRAY</b>	50
11.1.Généralités	50
11.2.Création et affectation d'un tableau	50
11.3.Accès aux données d'un tableau	51
11.4.Tableau à 2 dimensions	51
11.5.Propriétés et méthodes	52
11.6.Exemple concret	54
11.7.Exercice	54
<b>12.LES OBJETS DU NAVIGATEUR</b>	55
12.1.Généralités	55
12.2.Arborescence	55
<b>13.L'OBJET NAVIGATOR</b>	56
13.1.Les propriétés de navigator	56
13.2.Les méthodes de navigator	57
13.3.Les objets de navigator	57
13.4.Exemple	58
<b>14.L'OBJET WINDOW</b>	59
14.1.Les propriétés de window	59
14.2.Les méthodes de window	59
14.3.Les évènements de window	62
14.4.Les objets de window	63
14.5.Exemple concret	66
14.6.Exercice	66

<b>15.L'OBJET DOCUMENT</b>	67
15.1.Les propriétés de document	67
15.2.Les méthodes de document	68
15.3.Les évènements de document	69
15.4.Les objets de document	69
15.5.Exercice	72
<b>16.LES OBJET DU NOYAU JAVASCRIPT</b>	73
16.1.Généralités	73
16.2.Quelques précisions	73
<b>17.L'OBJET MATH</b>	74
17.1.Fonctions	74
17.2.Constantes	75
17.3.Simplification	75
17.4.Exercice	75
<b>18.L'OBJET STRING</b>	76
18.1.Généralités	76
18.2.La propriété	76
18.3.Les méthodes	76
18.4.Manipulations sur les chaînes	85
18.5.Exercice	87
<b>19.L'OBJET DATE</b>	88
19.1.Généralités	88
19.2.Les méthodes	88
19.3.Exemple concret	90
19.4.Exercice	91
<b>20.L'OBJET IMAGE</b>	92
20.1.Rappel HTML	92
20.2.L'objet	92
20.3.Les propriétés	92
20.4.Afficher une image	95
20.5.Exemple concret : cliquer pour changer d'image	96
20.6.Exercice	97
<b>21.LA PROGRAMMATION MULTI-CADRES</b>	98
21.1.Rappel HTML	98
21.2.Généralités	98
21.3.Liens hypertexte	98
21.4.Accès aux éléments des frames	99
21.5.Exemple concret	100
21.6.Exercice	101
<b>22.LES COOKIES</b>	102
22.1.Présentation	102
22.2.Créer un cookie	102
22.3.Récupérer un cookie	103
22.4.Modifier un cookie	104
22.5.Supprimer un cookie	105

<b>23.LA PROGRAMMATION OBJET</b>	106
23.1.Présentation	106
23.2.Déclaration d'une classe	106
23.3.Utilisation de méthodes	107
23.4. Exercice	108
<b>24.LES EXPRESSION REGULIERES</b>	109
24.1.Présentation	109
24.2.Définition	109
24.3.Paramètres d'une expression régulière	110
24.4.Utilisation d'une expression régulière	113
24.5.Exemple concret	114
24.6.Exercice	115
<b>25.FONCTIONS ET PROPRIETES</b>	116
25.1.Présentation	116
25.2.Les fonctions du langage	116
25.3.Méthodes et propriétés des objets	118
25.4.Exercice	120
<b>CONCLUSION</b>	121
<b>AUTEUR</b>	122
DESCRIPTION	122
COORDONNEES	122
<b>REMERCIEMENTS</b>	123

# INTRODUCTION

Que dire à propos de ce cours ? Voilà une bien grande question...à laquelle je ne peux pas répondre. Je pourrais parler de son contenu, mais à quoi bon ? Vous allez le lire - enfin je l'espère - dans quelques instants. Je pourrais parler de moi, mais ce n'est pas intéressant... Un problème insoluble se présente donc à moi.

Ce cours s'adresse à tout programmeur, du débutant ayant soif d'apprendre à l'expert ayant besoin de se remémorer quelque détail, en passant par le programmeur qui souhaite découvrir un langage ou se perfectionner... Son but n'est pas de former quelqu'un au Javascript, car ce serait se vanter. Non, son but est de donner des bases, que le programmeur averti saura compléter par d'autres lectures.

Il - le cours - n'a pas vocation à être exhaustif. Je dirai plutôt qu'il est évolutif, c'est-à-dire qu'il n'existe que pour être amélioré. Je compte bien entendu l'enrichir au fur et à mesure. A ce propos, je vous serai reconnaissant de m'adresser vos suggestions, critiques, compléments, et autres aides.

Je ne m'engage pas à fournir un cours parfait. Il peut - et il doit - y avoir des erreurs. Je ne prétends pas posséder le cours sans fautes. Je suis tout ouïe à vos critiques et corrections. Même si vous êtes content de ce cours - mon vœu le plus cher - donnez-moi votre avis, cela me permettra d'améliorer encore ce cours.

Enfin, si vous rencontrez un problème, si vous ne comprenez pas un point, n'hésitez pas à piocher dans la liste des liens que je vous fournit, et même, si un passage ne vous semble pas clair, écrivez-moi. Mes coordonnées figurent à la fin du document. Cependant, n'abusez pas de ce service, je ne suis pas une aide en ligne, j'ai aussi une vie professionnelle - voir la partie AUTEUR, pour ceux que cela intéresse - et je ne peux passer mon temps à répondre à vos questions. Si la réponse se fait attendre, patientez encore. Peut-être n'ai-je pas le temps ou peut-être n'ai-je pas encore trouvé la solution...

Dernier point : si vous souhaitez réutiliser ce cours, ce n'est pas un problème. Je vous demande cependant de bien vouloir me contacter, car je souhaite savoir ce qu'il devient. Sachez que cela me flatterait.

Merci d'avoir lu cette introduction. Je vous souhaite une bonne lecture et bienvenue dans l'univers de Javascript !

# 1.GENERALITES

## 1.1.Le langage Javascript

Le langage Javascript a été mis au point par Netscape en 1995. Microsoft ayant sorti son propre langage quelques temps après, un standard a été créé, le Javascript<sup>1</sup>. Actuellement, la version du langage est le Javascript 1.5<sup>2</sup>.

Ce langage est interprété, c'est-à-dire qu'il n'est pas compilé en langage machine avant exécution, comme le Java ou le C++. Le Javascript est intégré au code HTML, il vous faudra donc des bases assez solides en HTML. Si ce n'est pas le cas, il est conseillé de consulter un cours HTML de toute urgence. Il s'agit, a priori, du premier langage de script créé pour le web. Il permet d'exécuter des commandes du côté utilisateur, donc au niveau du navigateur et non du serveur - comme le PHP - .

Son principal inconvénient est le fait qu'il ne dispose pas de débogueur, la détection des erreurs en devient fastidieuse. Ensuite, il le code est accessible, puisque contenu dans la page envoyée au navigateur.

## 1.2.Côté pratique

Pour programmer en Javascript, il vous faut un navigateur web assez récent et un éditeur de texte, le bloc-notes de Windows est suffisant. Une connexion Internet est souhaitable. Au niveau de l'éditeur de texte, je vous conseillerait un éditeur un peu plus évolué. Le mieux serait bien entendu Dreamweaver MX<sup>3</sup>, si vous avez les moyens, bien que ce soit surtout utile si vous faites un site web. Du côté des éditeurs gratuits, je conseillerai Editplus<sup>4</sup>, un éditeur tous langages qui propose une coloration syntaxique. Si vous devez prendre un autre éditeur, veillez à ce qu'il propose cette coloration syntaxique, ça éclaire le code.

## 1.3.Détails techniques

Par convention, l'abréviation JS, utilisée souvent tout au long de ce cours, désigne Javascript.

Chaque chapitre concerne un point du langage qui peut être pris comme une leçon. A chaque fin de chapitre, je présente un exemple concret - quand c'est possible - et un ou plusieurs exercice(s). Ces exercices sont corrigés - un lien mène au fichier HTML - dans le dossier « solutions ».

Les cases blanches correspondent à la syntaxe, c'est-à-dire la mise en forme du langage. Les cases grises correspondent aux exemples. Lorsque ces derniers ont un résultat graphique, un lien mène au fichier HTML de l'exemple. Ces derniers sont dans le dossier « Exemples »

---

<sup>1</sup> Pour plus de précisions, voir le site <http://www.commentcamarche.net>.

<sup>2</sup> Ne fonctionne qu'avec Nestcape Navigator6.X et Internet Explorer 6.X

<sup>3</sup> Édité par Macromédia

<sup>4</sup> Édité par ES-Computing, à télécharger sur <http://www.telecharger.com>.

## 2.LE LANGAGE

### 2.1.Incorporation du code

Comme son nom l'indique, il s'agit d'un langage de script. Le code s'intègre donc dans la page HTML avec les balises <script>. Il existe deux façons d'intégrer votre code. La première consiste à le placer entre les balises, tout simplement.

Syntaxe :

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

Exemple 2.1 :

```
<script language = "Javascript">
var mavariable = 12 ;
document.write (typeof(mavariable)) ;
</script>
```

Il est aussi possible de placer votre code dans un fichier Javascript, comportant l'extension « .js ». Ensuite, il faut l'insérer dans la page HTML avec la balise <script>. Ne pas oublier il est préférable de placer le fichier JS dans la même dossier que la page.

Syntaxe :

```
<script src = "chemin fichier"></script>
```

Exemple 2.2 :

```
<script src = "date.js">
</script>
```

### 2.2.Spécificités du langage

La première chose qu'il faut noter en Javascript, comme dans le C, est que chaque instruction se termine par un point-virgule ‘;’. Il est possible de mettre plusieurs instructions sur la même ligne, puisque l'interpréteur ignore les sauts de lignes. Vous pouvez aussi insérer des blancs où vous voulez - excepté dans des noms de variables ou dans des expressions - cela ne change pas le code.

Exemple 2.3 :

```
var mavariable = 12 ;
document.write ( typeof(mavariable) ) ;
```

Il est utile de commenter son code. Cela se fait à l'aide de ‘//’, tout le texte suivant le double slash jusqu'à la fin de la ligne est ignoré par l'interpréteur.

**Exemple 2.4 :**

```
var mavariable = 12 ; //commentaire  
document.write ( typeof(mavariable) ) ;
```

Il est aussi possible de mettre des commentaires au milieu d'un ligne, ou sur plusieurs ligne, en les encadrant avec « /\* » et « \*/ »

**Exemple 2.5 :**

```
var mavariable = 12 ; /*commentaire sur  
plusieurs lignes*/  
document.write ( typeof(mavariable) ) ;
```

## 3.LES STRUCTURES DE DONNEES

### 3.1.Les données constantes

Le JS fournit quatre types de constantes déjà définies :

- Les constantes **numériques**, en notation décimale ( 75,2 ) ou flottante, c'est-à-dire scientifique (752E-1).
- Les constantes **booléennes** : **true** et **false**.
- Les **chaînes de caractères**, encadrées de guillemets ou de d'apostrophes ("").
- La constante **null** qui signifie « rien », « sans valeur », et qui est attribuée au variables non définies.

### 3.2.Les variables en JS

#### 3.2.1.Les types de variable

Un nom de variable doit commencer par une lettre (alphabet ASCII) ou le signe « \_ » et se composer de lettres, de chiffres et des caractères « \_ » et « \$ » (à l'exclusion du blanc). Le nombre de caractères n'est pas précisé. Javascript est sensible à la **casse** (majuscules et minuscules). Cela signifie que « MaVariable » n'est pas équivalent à « mavariable ».

Il existe 5 types de variables en Javascript :

- Les **nombres** : **number**
- Les **chaînes de caractères** : **string**
- Les **booléens**<sup>5</sup> : **boolean**
- Les **objets** : **object**
- Les **fonctions** : **function**

Nous verrons les fonctions au chapitre 5.LES FONCTION, mais je vous présente la première que vous verrez : la fonction **typeof()**, qui retourne le type d'une variable.

**Exemple 3.1 :**

```
var mavariable = 12 ;
document.write (typeof(mavariable)) ;
```

⇒ **Résultat**

Si la variable n'est pas affectée d'une valeur, elle recevra alors le type **undefined**, comme le montre l'exemple 3.2.

<sup>5</sup> Variable ne pouvant avoir que deux valeur : true (1) et false (0).

### Exemple 3.2 :

```
document.write (typeof(mavariable)) ;
```

#### ⇒ Résultat

### 3.2.2.La déclaration et l'affectation

On distinguera ici la **déclaration** et **l'affectation**. La première consiste à donner un nom à la variable alors que la seconde consiste à donner une valeur à la variable. La différence est qu'une variable peut-être affectée d'une valeur plusieurs fois alors qu'elle ne peut être définie qu'une seule fois. L'exemple suivant illustre une définition sans affectation.

### Exemple 3.3 :

```
var Numero;
```

Dans les exemples qui suivent, par facilité, j'ai regroupé l'affectation avec la déclaration. Les variables peuvent être déclarées de deux façons :

- De façon explicite, avec le mot-clé « **var** ».

### Exemple 3.4 :

```
var Numero = 1 ;
```

- De façon implicite. On écrit directement le nom de la variable suivi de la valeur que l'on lui attribue.

### Exemple 3.5 :

```
Numero = 1 ;
```

### 3.2.3.Les noms réservés

Les mots de la liste ci-après ne peuvent être utilisés pour des noms de fonctions et de variables. Certains de ces mots sont des mots clés Javascript, d'autres ont été réservés par Netscape pour un futur usage éventuel.

Lettre	Mot-clé
A	abstract
B	boolean / break / byte
C	case / catch / char / class / const / continue
D	default / do / double
E	else / extends
F	false / final / finally / float / for / function
G	goto
I	if / implements / import / in / instanceof / int / interface
L	long
N	native / new / null
P	package / private / protected / public
R	return
S	short / static / super / switch / synchronized
T	this / throw / throws / transient / true / try
V	var / void
W	while / with

Tab. 3.1 : Noms réservés

### 3.2.4.Manipulations sur les chaînes de caractères

Javascript convertit automatiquement les entiers en chaîne de caractères. Pour les puristes, il s'agit du **transtypage automatique**. Il est ainsi possible de concaténer des entiers avec des chaînes de caractères.

Dans la fonction d'écriture dans le document courant (`document.write()`), les données peuvent être séparées par des ',' ou des '+'.

Certains caractères peuvent être insérés dans les chaînes de caractères : le retour arrière (\b), le saut de page (\f), le saut de ligne (\n), l'entrée (\r), la tabulation (\t) et l'apostrophe (\').

On peut insérer des codes HTML dans les chaînes de caractères. Ces dernières seront interprétés comme de vraies balises.

Nous reverrons tout cela plus en détail dans le chapitre 16.L'OBJET STRING.

### 3.2.5. Variables globales et variables locales

Les variables déclarées tout au début du script, en dehors et avant toutes fonctions (voir plus loin), seront toujours globales, qu'elles soient déclarées avec var ou de façon contextuelle. On pourra donc les exploiter partout dans le script.

Dans une fonction, une variable déclarée par le mot clé var aura une portée limitée à cette seule fonction. On ne pourra donc pas l'exploiter ailleurs dans le script. D'où son nom de locale. Par contre, toujours dans une fonction, si la variable est déclarée contextuellement (sans utiliser le mot var), sa portée sera globale.

## 4. LES OPERATEURS

Comme tout langage informatique, JS possède des opérateurs pour effectuer les calculs. Leur présentation est rapide, surtout pour les plus simples. Dans les exemples, la valeur initiale de x sera toujours égale à 11, et celle de y sera égale à 5.

### 4.1.Les opérateurs de calcul

Signe	Nom	Signification	Exemple	Résultat
+	plus	addition	x + 3	14
-	moins	soustraction	x - 3	8
*	multiplié par	multiplication	x * 2	22
/	divisé par	division	x / 2	5.5
%	modulo	reste de la division par	x % 5	1
=	affectation	a la valeur	x = 5	5

Tab 4.1 : opérateurs de calcul

### 4.2.Les opérateurs de comparaison

Signe	Nom	Exemple	Résultat
==	Egal	x == 11	true
<	Inférieur	x < 11	false
<=	Inférieur ou égal	x <= 11	true
>	Supérieur	x > 11	false
>=	Supérieur ou égal	x >= 11	true
!=	Different	x != 11	false

Tab 4.2 : opérateurs de comparaison

Ces opérateur seront utilisés dans les boucles conditionnelles<sup>6</sup>. Le résultat s'exprime alors en valeur booléenne.

<sup>6</sup> Voir pour cela le chapitre 6.STRUCTURES DE CONTROLE

### 4.3.Les opérateurs associatifs

Signe	Description	Exemple	Signification	Résultat
<code>+=</code>	plus égal	<code>x += y</code>	<code>x = x + y</code>	16
<code>-=</code>	moins égal	<code>x -= y</code>	<code>x = x - y</code>	6
<code>*=</code>	multiplié égal	<code>x *= y</code>	<code>x = x * y</code>	55
<code>/=</code>	divisé égal	<code>x /= y</code>	<code>x = x / y</code>	2.2

Tab 4.3 : opérateurs associatifs

Ces opérateurs permettent un raccourci d'écriture, sans pour autant changer le résultat. Il permettent un incrémentation ou une décrémentation autre que 1.

### 4.4.Les opérateurs logiques

Signe	Nom	Exemple	Signification
<code>&amp;&amp;</code>	et	<code>(condition1) &amp;&amp; (condition2)</code>	condition1 <u>et</u> condition2
<code>  </code>	ou	<code>(condition1)    (condition2)</code>	condition1 <u>ou</u> condition2

Tab 4.4 : opérateurs logiques

On utilisera ces opérateurs dans les boucles conditionnelles principalement. Chaque condition correspondant à une expression avec un opérateur de comparaison. Ces opérateurs fonctionnent comme un ET logique et un OU logique<sup>7</sup>.

### 4.5.Les opérateurs d'incrémentation

Signe	Description	Exemple	Signification	Résultat
<code>x++</code>	incrémentation	<code>y = x++</code>	<code>y = x + 1</code>	6
<code>x--</code>	décrémentation	<code>y = x--</code>	<code>y = x - 1</code>	4

Tab 4.5 : opérateurs d'incrémentation

<sup>7</sup> Il s'agit d'un OU non exclusif, pour les puristes.

## 5. LES FONCTIONS

### 5.1.Définition

C'est un groupe d'instruction prédéfini et exécuté lorsqu'il est appelé et que l'on peut appeler plusieurs fois.

En Javascript, il existe deux types de fonctions :

- les fonctions propres à Javascript, appelées **méthodes**<sup>8</sup>. Elles sont associées à un objet en particulier.
- les fonctions que vous définissez vous-même. Ce sont celles qui nous intéressent ici.

### 5.2.Déclaration

Pour déclarer ou définir une fonction, on utilise le mot-clé **function**. La syntaxe d'une déclaration de fonction est la suivante :

Syntaxe :

```
function nom_de_la_fonction (arguments) {  
    code des instructions  
}
```

Le nom d'une fonction suit les mêmes règles que celui d'une variable<sup>9</sup>. Chaque nom de fonction doit être unique dans un même script. Les parenthèses sont obligatoires même si il n'y a pas d'arguments, puisque Javascript reconnaît une fonction grâce à elles.

### 5.3.Les fonctions dans l'en-tête

Il est plus prudent de placer les déclarations de fonctions dans l'en-tête `<head>...</head>` pour qu'elles soient prises en compte par l'interpréteur avant leur exécution dans le corps de la page `<body>...</body>`

### 5.4.L'appel d'une fonction

Syntaxe :

```
nom_de_la_fonction();
```

Il faut que la fonction aie été définie avant l'appel, étant donné que l'interpréteur lit le script de haut en bas.

<sup>8</sup> Ces fonctions particulières seront l'objet d'une sous-partie du chapitre 8.NOTION OBJET.

<sup>9</sup> voir Chapitre 1.STRUCTURES DE DONNEES.

## 5.5.Fonctions manipulant des valeurs

### 5.5.1.Passer une valeur à une fonction

On peut passer des valeurs à une fonction. On parle alors de paramètres. Pour passer un paramètre à une fonction, on fournit un nom d'une variable dans la déclaration de la fonction.

**Exemple 5.1 :** une fonction affichant le texte dans la page à laquelle on fournit le texte à afficher.

```
function Exemple(texte) { //définition avec un paramètre
    document.write(texte);
}
Exemple("Salut à tous"); // appel avec un paramètre
```

⇒ Résultat

On peut passer plusieurs paramètres à une fonction, en séparant les paramètres par des virgules.

Syntaxe :

```
function nom_de_la_fonction(arg1, arg2, arg3) {
    instructions
}
```

**Exemple 5.2 :**

```
function cube(nombre) {
    y = nombre*nombre*nombre;
    return y; //retour de valeur
}
x = cube(5); //appel avec paramètre
document.write(x); //résultat
```

⇒ Résultat

### 5.5.2.Retourner une valeur

Une fonction peut retourner une valeur. Il suffit alors de placer le mot-clé **return** suivi de la valeur ou de la variable à retourner.

### Exemple 5.3 :

```
function cube(nombre) { //Définition de la fonction
    var c = nombre*nombre*nombre ;
    return c; //Retour du cube du paramètre
}
var x = cube(5) ; // appel avec paramètre
document.write (x) ; //affichage
```

NB : Le mot-clé **return** est facultatif.

## 5.6. Variables locales et variables globales

Une variable déclarée dans une fonction par le mot-clé **var** aura une portée limitée à cette seule fonction. On l'appelle donc variable locale et ne pourra être utilisé dans le reste du script.

### Exemple 5.4 :

```
function cube(nombre) {
    var c = nombre*nombre*nombre ;
}
```

Si la variable est déclarée sans utiliser le mot **var**, sa portée sera globale.

### Exemple 5.5 :

```
function cube(nombre) {
    cube = nombre*nombre*nombre ;
}
```

Les variables déclarées tout au début du script, en dehors et avant toutes fonctions, seront toujours globales, qu'elles soient déclarées avec **var** ou de façon contextuelle.

### Exemple 5.6 :

```
var cube=1
function cube(nombre) {
    var cube = nombre*nombre*nombre ;
}
```

## 6. LES STRUCTURES DE CONTROLE

### 6.1.Les structures algorithmiques

Quelque soit le langage informatique utilisé<sup>10</sup>, un programme informatique utilise 3 catégories principales d'instructions. Ce sont :

- les instructions séquentielles.
- les instructions alternatives, ou conditionnelles.
- les instructions itératives, ou répétitives.

Nous verrons chacun des types de structures, en comparant l'algorithme et le code JS.

### 6.2.La structure séquentielle

C'est une suite d'instructions exécutées dans l'ordre où elles sont écrites, l'une après l'autre. L'exemple le plus fréquent est la fonction.

Algorithme	Code JS
DEBUT Instruction1 Instruction2 FIN	{ Instruction1 Instruction2 }

### 6.3.Les instructions conditionnelles

#### 6.3.1.L'expression if ... else

Il s'agit de tester une condition et d'effectuer une série d'instructions si la condition est vraie et si elle est fausse, effectuer une autre série d'instructions.

Algorithme	Code JS
SI condition ALORS Instructions1 SINON Instructions2 FINSI	if (condition) { Instruction1 } else { Instruction2 }

Tab. 6.1 : Expression conditionnelle if...else

<sup>10</sup> Ce chapitre, excepté pour quelques expressions, est valables pour de nombreux langages, notamment le C++.

### Exemple 6.1 :

```
x = prompt ("votre age?", "age");
if ( x < 40) {
    alert ('vous êtes jeune') ;
}
else {
    alert ('vous êtes vieux') ;
}
```

#### ⇒ Résultat

Dans l'exemple 6.1, la fonction `prompt()` permet d'afficher une boîte de dialogue.

### 6.3.2.L'expression () ? :

Il s'agit d'un moyen rapide de tester une condition et de d'exécuter une instruction selon son résultat.

Algorithme	Code JS
SI condition ALORS Instructions1 SINON Instructions2 FINSI	(condition) ? instruction 1 : instruction 2

Tab. 6.2 : Expression conditionnelle () ? :

Dans cette expression, si la condition est vérifiée, l'instruction 1 est effectuée, sinon, l'instruction 2 est effectuée. Précisons, que toute l'expression doit se trouver sur la même ligne.

### Exemple 6.2 :

```
x = prompt ("votre age?", "age");
age = (x < 40) ? "jeune" : "vieux";
alert ('vous êtes ' + age) ;
```

#### ⇒ Résultat

## 6.4.Les instructions itératives

### 6.4.1.L'itération for

Elle permet de répéter des instructions en s'arrêtant à un certain nombre d'itérations (boucles).

Algorithme	Code JS
POUR i = valeur initiale A i = valeur finale REPETER instructions FIN POUR	<pre>For (val initiale ; condition ; incrémentation) {   Instructions }</pre>

Tab. 6.3 : Expression itérative for

#### Exemple 6.3 :

```
for (i = 0; i < 10; i++) {
  document.write(i + " ");
}
```

#### ⇒ Résultat

### 6.4.2.L'itération while

Elle permet de répéter des instructions tant qu'une condition est vraie.

Algorithme	Code JS
TANT QUE (condition vraie) REPETER instructions FIN TANT QUE	<pre>while (condition) {   instructions }</pre>

Tab. 6.4 : Expression itérative while

#### Exemple 6.4 :

```
i = 0;
while (i < 10) {
  document.write(i + " ");
  i++;
}
```

#### ⇒ Résultat

## 6.5. Interrompre une boucle

### 6.5.1. L'instruction break

Elle permet de mettre fin prématûrement à une instruction itérative `while` ou `for`.

**Exemple 6.5 :**

```
for (i = 0; i < 10; i++) {  
    if (i == 5) {  
        break;  
    }  
    document.write(i + " ");  
}
```

⇒ Résultat

### 6.5.2. L'instruction continue

Elle permet de mettre fin à une itération et de passer à la boucle suivante.

**Exemple 6.6 :**

```
for (i = 0; i < 5; i++) {  
    if (i == 2) {  
        continue ;  
    }  
    document.write(i + '<br>');  
}
```

⇒ Résultat

## 6.6. Exercice

Dans cet exercice, vous devez afficher tous les nombres pairs compris entre 0 et 20 inclus, dans l'ordre décroissant. Il ne faudra pas afficher 10 mais 100, et il ne faudra pas non plus afficher 14.

⇒ Solution

## 7. LES BOITES DE MESSAGE

### 7.1. Généralités

Nous abordons les boîtes de dialogue, qui sont des méthodes de l'objet window, que nous verrons plus tard, dans la partie objet du langage. Même si nous n'avons pas encore abordé la programmation objet, il me semble important de vous donner ces boîtes de dialogue, qui s'avèrent utile en JS. Il en existe 3, que nous détaillerons l'une après l'autre.

### 7.2. Alert()

Nous avons déjà utilisé cette méthode précédemment, sans savoir forcément ce qui en rentrait. Il ne comporte qu'un texte informatif et un bouton « OK ».

**Syntaxe :**

```
alert (paramètres) ;
```

**Paramètres :**

- Une chaîne de caractère
- Une variable
- Un enchaînement des deux, séparés par le signe "+"

**Exemple 7.1 :**

```
x = 5 ;
alert ('Le nombre est ' + x) ;
```

⇒ **Résultat**

### 7.3. Prompt()

Il s'agit d'une boîte d'invite, composée d'un texte, d'une zone de texte, d'un bouton « OK » et d'un bouton « Annuler ».

**Syntaxe :**

```
variable = prompt ("texte", "valeur") ;
```

La méthode retourne la valeur du champ si le bouton « OK » est choisi dans le cas inverse (bouton « Annuler »), la variable reçoit la valeur « **NULL** ».

**Paramètres :**

- Le texte à afficher dans la boîte de message
- La valeur par défaut à afficher dans la boîte d'invite.

### Exemple 7.2 :

```
x = prompt ('Votre prénom ?','prénom') ;  
alert (x) ;
```

#### ⇒ Résultat

## 7.4. Confirm()

Il s'agit d'une boite de confirmation, composée d'un texte, d'un bouton « OK » et d'un bouton « Annuler ».

#### Syntaxe :

```
variable = confirm ("texte") ;
```

La méthode retourne **true** si on clique sur « OK », et **false** si on clique sur « Annuler »

#### Paramètre :

- Le texte à afficher dans la boite de message

### Exemple 7.3 :

```
x = prompt ('Votre prénom ?','prénom') ;  
y = confirm ('Votre prénom est bien ' + x + ' ?') ;
```

#### ⇒ Résultat

## 7.5. Exercice : Confirmation et vérification

Le but est de récupérer et afficher 3 informations de l'utilisateur à l'écran. Il faut demander à l'utilisateur son login. S'il choisit annuler, on affiche la page blanche. Ensuite, on compare le login entré avec celui attendu : s'il est différent, on redemande le login, sinon, on continue le traitement. On demande ensuite le password. On vérifie qu'il est correct et on continue. S'il ne l'est pas, on redemande 2 fois ce password. S'il est toujours incorrect au 3<sup>ème</sup> essai, on affiche un message d'erreur à l'écran, et le traitement est fini. Enfin, on demande le prénom de l'utilisateur, puis on demande vérification à l'internaute. Si le prénom est correct, on l'affiche à l'écran.

Pour afficher à l'écran, utiliser la méthode `document.write()`, en passant la chaîne à afficher en paramètre.

Cet exercice utilisera vos connaissances au niveau des fonctions, des variables, des structures algorithmiques, et bien entendu des boites de dialogue.

Dans la solution, le login est « prof » et le password est « abcd ». Je suis resté classique.

⇒ **Solution**

## 8. NOTION OBJET

Bien que ce ne soit pas le but dans ce chapitre, il me semble important de rappeler le concept objet, notamment pour ceux qui ne le connaissent pas. Il ne s'agit pas d'un cours magistral, mais juste d'un résumé à des fins d'utilisation en JS.

### 8.1. Le concept objet

Jusque-là, nous avons vu des variables, avec une valeur, indépendantes des autres. Maintenant, on parlera d'objet. Un objet - en général - possède des caractéristiques et des compétences. On peut voir ces caractéristiques et utiliser les compétences. En informatique, un objet possède des variables et des fonctions, qui permettent de décrire son comportement. Ces variables sont appelées **propriétés** et ces fonctions sont appelées **méthodes**. De cette façon, on peut voir les propriétés et utiliser les méthodes.

### 8.2. Crédit d'un objet

Un objet est créé en utilisant une fonction spéciale de la classe, précédée du mot **new**. Cette fonction est appelée **constructeur**, et porte le nom de la classe. Un objet est appelé **instance** de la classe.

Syntaxe :

```
var objet = new classe ();
```

Exemple 8.1 :

```
var montableau = new Array ();
```

### 8.3. Accès aux propriétés et aux méthodes

On accède aux propriétés - et aux méthodes - d'un objet en accolant l'objet et sa propriété avec l'opérateur « . ».

Syntaxe :

```
objet.propriété  
objet.méthode()
```

Exemple 8.2: par extrapolation

```
Rémi = new Homme () ;  
Rémi.yeux = "bleus" ;  
Rémi.cheveux = "bruns" ;
```

### Exemple 8.3:

```
document.write ("Hello world !");
```

L'objet `document` est intégré au langage, et sa méthode `write()` permet d'écrire dans la page courante.

## 8.4.L'objet `this`

Il existe un objet très utile en JS objet : `this`. Même s'il sera plus utile en programmation objet, lorsque vous créerez vos classes. Il s'agit d'un objet qui représente l'objet en cours. Il possède alors les propriétés et les méthodes de l'objet. Etant donné son utilité minime avant la programmation objet, je ne donnerai pas d'exemple ici.

## 9. LES FORMULAIRES

### 9.1. Généralités

#### 9.1.1. Présentation

On attaque ici le gros morceau du JS. Sans les formulaires, les pages HTML ne proposent aucune interactivité avec l'utilisateur. En effet, la page appelée est envoyée par le serveur et le browser ne fait que l'afficher, il ne peut rien faire avec le serveur. Les champs de formulaire (form) permettent l'utilisateur d'entrer des informations auxquelles la page pourra réagir. Ces réactions seront programmées à l'avance par vous. Cela reste assez limité, puisque l'interaction ne quitte pas la machine du client, mais on peut effectuer quelques traitements. Mon but n'est pas de vous apprendre le HTML, je ne reviendrai donc pas sur le fonctionnement des balises.

Autre point, j'indique ici, par commodité, les propriétés JS, dont on se servira plus loin. Ne vous en préoccupez pas si vous ne comprenez pas de quoi il s'agit.

#### 9.1.2. La balise form

Chaque formulaire doit être encadré par les balises `<form>...</form>`. Toutes les balises seront comprises entre ces deux-là.

Si vous travaillez avec PHP ou un autre langage, vous aurez sûrement besoin d'envoyer les informations à un serveur. Dans ce cas, indiquez dans la balise `<form>` la méthode (`post` ou `get`) et l'URL du script qui traitera l'information.

#### Exemple 9.1 :

```
<form method='post' action='traitement.php'>
.....
</form>
```

Si vous désirez que les informations du formulaire vous soient envoyées par mail, précisez '`mailto:...`' dans l'action.

#### Exemple 9.2 :

```
<form method='post' action='mailto:e-mail'>
.....
</form>
```

Dans chaque cas, n'oubliez pas d'insérer un bouton submit dans votre formulaire.

## 9.2. Champ de texte

### 9.2.1. Ligne de texte

Il s'agit de l'élément d'entrée/sortie le plus courant en JS. Une simple ligne où l'utilisateur peut écrire quelques mots.

Syntaxe HTML :

```
<input type="text" name="nom" value="valeur"  
size=x maxlength=z>
```

Paramètres HTML :

- name : le nom du champ (servira lors de la programmation)
- value : la valeur que vous voulez afficher dans la ligne de texte (facultatif).
- size : la longueur de la ligne de texte. Si le texte est trop long, la ligne défilera.
- maxlength : le nombre de caractères maximum contenus dans la ligne de texte (facultatif mais conseillé).

Propriétés JS :

- name : indique le nom du contrôle.
- defaultvalue : indique la valeur par défaut affichée dans la ligne de texte.
- value : indique la valeur actuellement dans la zone de texte.

Exemple 9.3 :

```
<form method="post" action="mailto:lapape@le-  
vatican.com">  
<input type="text" name="text1" size=20  
maxlength=40><br>  
<input type="submit" name="envoi"  
value="envoyer">  
</form>
```

⇒ Résultat

### 9.2.2. Zone de texte

Il s'agit de plusieurs ligne de textes. Utile quand le texte est long.

Syntaxe HTML :

```
<textarea name="nom" rows=x cols=y>  
texte par défaut  
</textarea>
```

### Paramètres HTML :

- name : le nom du champ (servira lors de la programmation)
- rows : le nombre de lignes affichés à l'écran. Si le texte est trop long, la zone de texte défilera.
- cols : le nombre de colonnes affichées à l'écran.

### Propriétés JS :

- name : indique le nom du contrôle.
- defaultvalue : indique la valeur par défaut affichée dans la ligne de texte.
- value : indique la valeur actuellement dans la zone de texte.

### Exemple 9.4 :

```
<form method="post"
action="mailto:lapape@le-vatican.com">
<textarea name="text2" rows=5 cols=20>
zone de texte
</textarea><br>
<input type="submit" name="envoi"
value="envoyer">
</form>
```

### ⇒ Résultat

#### 9.2.3.Champ password

Il s'agit d'une ligne de texte dont les caractères sont cachés.

### Syntaxe HTML :

```
<input type="password" name="nom"
value="valeur" size=x maxlength=z>
```

### Paramètres HTML :

- name : le nom du champ (servira lors de la programmation)
- value : la valeur que vous voulez afficher dans le champ (facultatif).
- size : la longueur de la ligne de texte. Si le texte est trop long, la ligne défilera.
- maxlength : le nombre de caractères maximum contenus dans la ligne de texte (facultatif mais conseillé).

### Propriétés JS :

- name : indique le nom du contrôle.
- defaultvalue : indique la valeur par défaut affichée dans la ligne de texte.
- value : indique la valeur actuellement dans le champ password.

### Exemple 9.5 :

```
<form method="post" action="mailto:lapape@le-vatican.com">
<input type="password" value="password"
name="text1" size=20 maxlength=40><br>
<input type="submit" name="envoi"
value="envoyer">
</form>
```

#### ⇒ Résultat

## 9.3.Cases à sélectionner

### 9.3.1.Boutons radios

Il s'agit de cases à cocher à choix unique. D'une forme ronde, elle sont liées entre elles au niveau du code JS.

#### Syntaxe HTML :

```
<input type="radio" name="nom"
value="valeur"> texte
```

#### Paramètres HTML :

- name : le nom du champ (servira lors de la programmation) il doit être identique pour chaque choix.
- value : la valeur que vous voulez afficher dans le champ (facultatif).
- checked : à mettre sur un seul bouton, qui sera sélectionné par défaut.

Il est nécessaire de préciser le label après le contrôle (**texte**)

#### Propriétés JS :

- name : indique le nom du contrôle.
- checked : indique si le bouton radio est coché, actuellement.
- defaultchecked : indique si le bouton radio est coché ou non par défaut.
- value : indique la valeur du bouton radio.
- index : indique le rang du bouton radio, à partit de 0.

### Exemple 9.6 :

```
<form method="post" action="mailto:lapape@le-
vatican.com">
<input type="radio" name="choix" value="1">
choix 1<br>
<input type="radio" name="choix" value="2">
choix 2<br>
<input type="radio" name="choix" value="3">
choix 3<br>
<input type="submit" name="envoi"
value="envoyer">
</form>
```

#### ⇒ Résultat

### 9.3.2. Checkbox

Il s'agit de cases à cocher à choix multiple.

#### Syntaxe HTML :

```
<input type="checkbox" name="nom"
value="valeur"> texte
```

#### Paramètres HTML :

- name : le nom du champ (servira lors de la programmation). Il doit être différent pour chaque choix.
- value : la valeur que vous voulez afficher dans le champ (facultatif).
- checked : à mettre sur un (ou plusieurs) bouton, qui sera sélectionné par défaut.

Il est nécessaire de préciser le label après le contrôle (**texte**)

#### Propriétés JS :

- name : indique le nom du contrôle.
- checked : indique si la case est cochée, actuellement.
- defaultchecked : indique si la case est cochée ou non par défaut.
- value : indique la valeur de la case à cocher.

### Exemple 9.7 :

```
<form method="post" action="mailto:lapape@le-
vatican.com">
<input type="checkbox" name="choix1" value="1">
choix 1<br>
<input type="checkbox" name="choix2" value="2">
choix 2<br>
<input type="checkbox" name="choix3" value="3">
choix 3<br>
<input type="submit" name="envoi"
value="envoyer">
</form>
```

#### ⇒ Résultat

## 9.4. Liste de sélection

Il s'agit d'une liste déroulante dans laquelle on peut sélectionner un option.

#### Syntaxe HTML :

```
<select name="nom" size=x>
<option value="valeur"> texte
</select>
```

#### Paramètres HTML :

- name : le nom du champ (servira lors de la programmation).
- size : le nombre d'options que vous voulez afficher à l'écran. S'il n'est pas précisé, la liste n'affiche qu'une seule ligne.
- value : la valeur que vous voulez afficher dans le champ (facultatif).
- selected : à mettre dans une balise `<option>`. Cette option sera sélectionnée par défaut.
- Multiple : à mettre dans une balise `<select>`. Autorise la sélection de plusieurs options dans la liste déroulante.

Chaque option de la liste déroulante correspond à une balise `<option>`. Il est nécessaire de préciser le label après chaque option contrôle (**texte**).

#### Propriétés JS :

- name : indique le nom du contrôle.
- selected : indique si le bouton radio est sélectionné
- defaultselected : indique si le bouton radio est coché ou non par défaut.
- length : indique le nombre d'éléments de la liste.

### Exemple 9.8 :

```
<form method="post" action="mailto:lapape@le-
vatican.com">
<select name="choix" size=3
```

#### ⇒ Résultat

### Exemple 9.9 :

```
<form method="post" action="mailto:lapape@le-
vatican.com">
<select name="choix">
<option value="1">choix 1
<option value="2">choix 2
<option value="3">choix 3
</select><br>
<input type="submit" name="envoi"
value="envoyer">
</form>
```

#### ⇒ Résultat

## 9.5.Bouton

### 9.5.1.Bouton simple

#### Syntaxe HTML :

```
<input type="button" name="nom" value="valeur">
```

#### Paramètres HTML :

- name : le nom du champ (servira lors de la programmation).
- value : la valeur que vous voulez afficher sur le bouton.

#### Propriétés JS :

- name : indique le nom du contrôle.
- value : indique la valeur actuelle du bouton.
- defaultselected : indique la valeur par défaut du bouton.

### Exemple 9.10 :

```
<form method="post" action="mailto:lapape@le-
vatican.com">
<input type="button" name="click"
value="Cliquez">
</form>
```

#### ⇒ Résultat

### 9.5.2.Bouton reset

Il permet de remettre la valeur par défaut de tous les champs du formulaire.

#### Syntaxe HTML :

```
<input type="reset" name="nom" value="valeur">
```

#### Paramètres HTML :

- name : le nom du champ (servira lors de la programmation).
- value : la valeur que vous voulez afficher sur le bouton.

#### Propriétés JS :

- name : indique le nom du contrôle.
- value : indique la valeur actuelle du bouton.
- defaultselected : indique la valeur par défaut du bouton.

### Exemple 9.11 :

```
<form method="post" action="mailto:lapape@le-
vatican.com">
<select name="choix" size=3>
<option value="1">choix 1
<option value="2">choix 2
<option value="3" selected>choix 3
</select><br>
<input type="reset" name="effacer"
value="effacer">
</form>
```

#### ⇒ Résultat

### 9.5.3.Bouton submit

Il permet d'exécuter l'action prévue dans la balise <form>, comme on l'a vu dans la partie 6.1.2.

### Syntaxe HTML :

```
<input type="submit" name="nom" value="valeur">
```

#### Paramètres HTML :

- name : le nom du champ (servira lors de la programmation).
- value : la valeur que vous voulez afficher sur le bouton.

#### Propriétés JS :

- name : indique le nom du contrôle.
- value : indique la valeur actuelle du bouton.
- defaultselected : indique la valeur par défaut du bouton.

#### Exemple 9.12 :

```
<form method="post" action="mailto:lapape@le-vatican.com">
<select name="choix" size=3>
<option value="1">choix 1
<option value="2">choix 2
<option value="3" selected>choix 3
</select><br>
<input type="submit" name="envoi"
value="envoyer">
</form>
```

#### ⇒ Résultat

## 9.6. Contrôle caché

Il permet de mettre dans le script des éléments (souvent des données) qui ne seront pas affichées par le navigateur, mais qui pourront être envoyées par mail ou à un serveur.

### Syntaxe HTML :

```
<input type="hidden" name="nom" value="valeur">
```

#### Paramètres HTML :

- name : le nom du champ (servira lors de la programmation).
- value : la valeur qui sera contenue dans le contrôle.

#### Propriétés JS :

- name : indique le nom du contrôle.
- value : indique la valeur actuelle du contrôle.
- defaultselected : indique la valeur par défaut du contrôle.

#### Exemple 9.13 :

```
<form method="post" action="mailto:lapape@le-
vatican.com">
<input type="hidden" name="date"
value="30/06/2004">
<input type="button" name="click" value="Vous
pouvez cliquez!">
</form>
```

⇒ **Résultat**

## 9.7.Un formulaire complet

Voilà un formulaire complet, utilisant la plupart des contrôles que nous avons vu ci-dessus. Il pourrait s'agir d'un formulaire réel, mais je l'ai inventé de toutes pièces. De nombreux sites proposent des formulaires, rien ne vous empêche d'enregistrer la page et d'aller fouiner pour voir leur astuces...

### Exemple 9.14 :

```
<form method="post" action="traitement.php">
Inscription au stage<br>
Identité<br>
<input type="hidden" name="date" value="15/06/04">
<input type="text" name="nom" value="Nom">&ampnbsp
<input type="text" name="prénom" value="Prénom"><br>
<input type="text" name="date" value="Date de
naissance "><br>
Activité<br>
<select name="Activité">
<option value="Collégien">Collégien
<option value="Lycéen">Lycéen
<option value="Etudiant">Etudiant
<option value="Salarié">Salarié
<option value="Fonctionnaire">Fonctionnaire
<option value="Patron">Patron
</select><br>
Qualification(s) :<br>
<input type="checkbox" name="choix1" value="bafa">
BAFA<br>
<input type="checkbox" name="choix2" value="bafd">
BAFD<br>
<input type="checkbox" name="choix3" value="rien">
rien<br>
Semaine choisie:<br>
<input type="radio" name="semaine" value="1">
1ère<br>
<input type="radio" name="semaine" value="2">
2ème<br>
<input type="radio" name="semaine" value="3">
3ème<br>
Vos motivations :<br>
<textarea name="motivations" rows=5
cols=20></textarea><br>
<input type="submit" name="envoi" value="envoyer">
&ampnbsp
<input type="reset" name="effacer" value="effacer">
</form>
```

### ⇒ Résultat

## 9.8.Les objet du formulaire

Nous avons vu précédemment la notion d'objet<sup>11</sup>. Maintenant que nous avons vu tous les composants d'un formulaire, nous allons pouvoir utiliser le concept objet appliquée aux formulaires.

Même si nous n'avons pas encore vu les objets du navigateur<sup>12</sup>, nous allons introduire ici l'objet `document`. Il renvoie à la page en cours. Il contient en propriétés tous les éléments de la page. Les attributs des balises deviennent les propriétés de l'élément objet. On y a accès de la façon suivante :

**Syntaxe :**

```
document.formulaire.élément.propriété
```

Un exemple éclairera le propos. L'exemple 9.15 montre comment attribuer une valeur à une ligne de texte.

**Exemple 9.15 :**

```
<html>
<head>
<script language = "Javascript">
function f() {
    document.form1.texte.value="voici du texte";
}
</script>
</head>
<body onLoad="f() ;">
<form name="form1">
<input type="text" name="texte" value="">
</form>
</body>
</html>
```

⇒ **Résultat**

Dans l'exemple 9.15, la fonction est appelée à la fin du chargement grâce au gestionnaire d'événement<sup>13</sup> placé dans la balise `<body>`.

De cette façon, on peut accéder à chaque élément de chaque formulaire de la page. Il est important de donner un nom, à la fois au formulaire et au contrôle. Ce nom doit de préférence être explicite, même si celui du formulaire peut rester

---

<sup>11</sup> Voir, pour ceux qui auraient oublié, le chapitre 8.NOTION OBJET.

<sup>12</sup> Voir pour cela le chapitre 12.LES OBJETS DU NAVIGATEUR.

<sup>13</sup> Nous verrons cela dans le chapitre 10.LES EVENEMENTS.

classique, étant donné qu'il y en a rarement plusieurs sur une seule et même page. Lorsque nous travaillerons avec les frames, en revanche, ce nom sera important.

Cette manière de pouvoir accéder aux objets est très pratique, même si pour le moment, elle ne semble pas très utile. Cette utilité sera mise en valeur lors du prochain chapitre. Mais avant cela, il est souhaitable de faire un petit exercice.

## 9.9.Exercice

Le but de cet exercice est d'arriver à un formulaire complet et assez ergonomique. Il faudra faire attention aux choix des contrôles et à la disposition. La solution proposée n'est pas parfaite - loin de là - mais elle illustre à peu près ce qu'est un formulaire de qualité correcte.

Le formulaire à créer est un formulaire d'adhésion à une association, pour l'activité échecs. Les informations demandées sont les suivantes : nom, prénom, date de naissance, numéro de téléphone, adresse, mail, profession (choix, dont « autres : préciser »), centres d'intérêt (choix), niveau de jeu, motivations. Vous présenterez ce formulaire de façon assez esthétique, mais la police et la taille de caractères ne sont pas importantes, cela dépend plutôt du cours HTML. Bien entendu, on envoie le tout à « traitement.php », par la méthode « post » en cliquant sur un bouton.

⇒ **Solution**

# 10. LES EVENEMENTS

## 10.1. Généralités

### 10.1.1. Présentation

Les événements sont l'intérêt du JS en matière de programmation Web. Il donnent une interactivité à la page que vous consultez, ce qui n'existe pas avec le HTML, si on excepte bien entendu le lien hypertexte. Le JS permet de réagir à certaines actions de l'utilisateur. On nomme événement le mot Event, et gestionnaire d'événement le mot-clé `onEvent`.

### 10.1.2. Fonctionnement

Les événements sont indiqués dans la balise, d'un formulaire, le plus souvent. Ils apparaissent comme un paramètres supplémentaire et suivent une syntaxe particulière.

Syntaxe :

```
<balise onEvent="code">
```

- balise : désigne le nom de la balise HTML qui supporte l'événement.
- onEvent : désigne le gestionnaire d'événement associé à l'événement *Event*.
- Le code inséré entre guillemets fait le plus souvent référence à une fonction définie dans les balises `<head>...</head>`. Il peut cependant s'agir d'instructions directement.

Plusieurs gestionnaires d'événements peuvent être placés dans la même balise. Certaines balises n'appartenant pas à un formulaire peuvent supporter des gestionnaires d'événement.

Exemple 10.1 :

```
<a href="http://www.google.fr" onClick="alert('vous  
avez cliqué!');" onMouseOver="alert('Héhé') ;;">  
click?</a>
```

⇒ Résultat

## 10.2.Le clic de souris

Lorsque vous cliquez sur un élément de formulaire de la page que vous consultez.

**Gestionnaire d'événement :**

onClick

**Exemple 10.2 :**

```
<input type="button" onClick="alert('vous avez  
cliqué sur le bouton') ;">
```

⇒ **Résultat**

**Balises supportées :**

<input type="button">, <input type="checkbox">, <input type="radio">, <input type="reset">, <input type="submit">, <a href..>

## 10.3.Le chargement

### 10.3.1.Load

Lorsque la page que vous consultez finit de se charger.

**Gestionnaire d'événement :**

onLoad

**Exemple 10.3 :**

```
<body onLoad="alert('la page est chargée !') ;">
```

⇒ **Résultat**

**Balises supportées :**

<body>, <frameset>

### 10.3.2.UnLoad

Lorsque vous quittez un document ou une page web.

**Gestionnaire d'événement :**

onUnLoad

**Exemple 10.4 :**

```
<body onUnLoad="alert('Vous quittez la page !') ;">
```

⇒ **Résultat**

**Balises supportées :**

<body>, <frameset>

### 10.3.3.Error

Lorsque le chargement d'une page ou d'un image s'interrompt en erreur.

**Gestionnaire d'événement :**

onError

**Exemple 10.5 :**

```

```

**Balises supportées :**

<body>, <frameset>, <img>

### 10.3.4.Abort

Lorsque vous interrompez le chargement d'une image.

**Gestionnaire d'événement :**

onAbort

**Exemple 10.6 :**

```

```

**Balises supportées :**

<img>

## 10.4. Le passage de la souris

### 10.4.1. MouseOver

Lorsque vous survolez un lien ou une zone d'image activable. Un zone d'image activable est une partie d'image qui a été transformée en lien. Je ne reviendrai pas sur ce sujet, ce n'est pas l'objet de ce cours.

**Gestionnaire d'événement :**

onMouseOver

**Exemple 10.7 :**

```
<a href="http://www.google.fr"  
onMouseOver="alert('Pour aller sur google.fr,  
cliquer ici') ;">http://www.google.fr</a>
```

⇒ **Résultat**

**Balises supportées :**

<a href...>, <area href...>

### 10.4.2. MouseOut

Lorsque vous sortez de la zone de survol d'un lien ou d'une zone d'image activable.

**Gestionnaire d'événement :**

onMouseOut

**Exemple 10.8 :**

```
<a href="http://www.google.fr"  
onMouseOut="alert('Vous ne voulez pas y  
aller ?') ;">http://www.google.fr</a>
```

⇒ **Résultat**

**Balises supportées :**  
<a href...>, <area href...>

## 10.5.Le focus

### 10.5.1.Focus

Lorsque vous activer un contrôle texte ou sélection.

**Gestionnaire d'événement :**

onFocus

**Exemple 10.9 :**

```
<input type="text" value="votre nom" name="nom"
onFocus="alert('Ecrivez votre nom ici') ;">
```

⇒ **Résultat**

**Balises supportées :**  
<input type="text">, <select>, <textarea>, <input type="password">

### 10.5.2.Blur

Lorsque vous quitter un contrôle texte ou sélection.

**Gestionnaire d'événement :**

onBlur

**Exemple 10.10 :**

```
<input type="text" value="votre nom" name="nom"
onBlur="alert('Vous n\'avez rien oublié ?') ;">
```

⇒ **Résultat**

**Balises supportées :**  
<input type="text">, <select>, <textarea>, <input type="password">

## 10.6.Les changements

Lorsque la valeur d'un texte ou d'une option change dans un formulaire. Si vous cliquez dans la zone de texte mais que vous ne touchez pas au texte, rien ne se produit.

**Gestionnaire d'événement :**

onChange

**Exemple 10.11 :**

```
<input type="text" value="votre nom" name="nom"
onchange="alert('Vous avez changé votre nom ??') ;">
```

⇒ **Résultat**

**Balises supportées :**

<input type="text">, <select>, <textarea>, <input type="password">

## 10.7.La sélection

Lorsque vous sélectionnez du texte dans un champ de texte.

**Gestionnaire d'événement :**

onSelect

**Exemple 10.12 :**

```
<input type="text" value="votre nom" name="nom"
onselect="alert('Vous avez sélectionné un
champ') ;">
```

⇒ **Résultat**

**Balises supportées :**

<input type="text">, <textarea>

## 10.8.L'envoi

Lorsque vous cliquez sur un bouton « submit » d'un formulaire de type « post » ou « get ».

**Gestionnaire d'événement :**

```
onSubmit
```

**Exemple 10.13 :**

```
<input type="submit" value="Envoyer" name="envoi"
onSubmit="alert('C'est parti !') ;">
```

⇒ **Résultat**

**Balises supportées :**

```
<input type="submit">
```

## 10.9.Le reset

Lorsque vous sélectionnez un champ de texte.

**Gestionnaire d'événement :**

```
onReset
```

**Exemple 10.14 :**

```
<input type="reset" value="Effacer" name="effacer"
onSubmit="alert('On efface tout !') ;">
```

⇒ **Résultat**

**Balises supportées :**

```
<input type="reset">
```

## 10.10.L'utilisation de l'objet document

Avant de voir les événements, l'intérêt de l'accès au éléments à travers l'objet document était discutable. Maintenant, on voit que l'on peut changer des valeurs selon des évènements, ce qui est très intéressant. Il est aussi possible de demander confirmation, de demander si on veut vraiment changer la valeur... Tout est permis. Cette partie ne comporte pas d'exemple, car la partie 10.11.Exemple concret en présente un assez complet.

## 10.11.Exemple concret

Voici un exemple - parmi tant d'autres - de ce que pourrait être un formulaire interactif, avec tous les évènements auxquels on peut penser.

**Exemple 10.15 :**

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
var TestLog = 0; //test pour le login
function ChangeLog() {
    TestLog++; //incrémente le test
    //si le login a été changé plus d'une fois
    if (TestLog > 1) {
        alert("Vous changez de login?\n Décidez-vous!");
    }
}
function VerifPass () {
    // si les deux password sont différents
    if (document.form1.pass1.value != document.form1.pass2.value) {
        alert("Vous avez entré deux password différents !\nVérifiez
les deux.");
    }
}
function VerifMail () {
    //confirmation du mail
    var test = confirm ("Votre mail est bien : " +
    document.form1.mail.value + " ?");
    if (test == false) { //si l'internaute ne confirme pas
        alert("N'oubliez pas de changer votre mail!");
    }
}
</script>
</HEAD>
<BODY>
<form name="form1">
    <center>Formulaire d'inscription sur notre site</center><br/>
    Login : <input type="text" name="login" onChange="ChangeLog();"
value="login"><br/>
    Password : <input type="password" name="pass1"><br/>
    Password (vérification) : <input type="password" name="pass2"
onBlur="VerifPass();"><br/>
    Adresse e-mail : <input type="text" name="mail"
onBlur="VerifMail(); value="@"><br/>
    Vous voulez recevoir la newsletter? <input type="radio"
name="news" value="yes">Oui <input type="radio" name="news" value="no"
onClick="alert('Tant pis pour vous!')">Non<br/>
    <input type="submit" name="sub" value="Envoi"> <input
type="reset" name="reset" value="Effacer">
</form>
</BODY>
</HTML>
```

⇒ **Résultat**

## 10.12.Exercice

### 10.12.1.Le bouton

L'exercice consiste à faire un bouton dont la valeur s'incrémente à chaque clic. Ce n'est pas très compliqué.

⇒ **Solution**

### 10.12.2.La phrase

L'exercice consiste à demander 3 informations à un utilisateur, puis de les afficher dans un zone de texte en faisant un phrase. Il faudra demander le nom à l'aide d'une boite de message, ainsi que l'âge et la ville avec des lignes de texte. Ensuite, en cliquant sur un bouton, on affiche une phrase contenant les 3 informations dans un zone de texte.

⇒ **Solution**

## 11. L'OBJET ARRAY

### 11.1. Généralités

L'objet Array n'est rien d'autre qu'un tableau. Un tableau est en quelque sorte une liste d'éléments indexés que l'on pourra lire - chercher les données - ou écrire - entrer les données - à volonté.

### 11.2. Crédation et affectation d'un tableau

#### 11.2.1 Crédation d'un tableau

On crée un tableau comme n'importe quel objet de classe. On invoque le constructeur de classe. On indique en paramètre du constructeur le nombre maximum d'éléments (x) que l'on souhaite entrer dans le tableau. Ce nombre est cependant facultatif, car JS prend en compte le numéro du dernier élément entré comme taille du tableau si le ce nombre n'est pas indiqué.

Syntaxe :

```
variable = new Array(x) ;
```

Exemple 11.1 :

```
var MonTableau = new Array (10) ;
```

#### 11.2.2 Affectation d'un tableau

On affecte un tableau très simplement. Il suffit d'affecter une valeur (y) à la variable avec le numéro de l'élément (i) entre crochets. Ces derniers ne sont pas présents lors de la création mais sont indispensables lors de l'affectation. Le numéro commence à 0 et finit au nombre maximum moins 1.

Syntaxe :

```
variable = new Array(x) ;
variable [i] = y;
```

Exemple 11.2 :

```
var MonTableau = new Array (2) ;
MonTableau [0] = 7 ;
MonTableau [1] = 4 ;
```

Evidemment, à ce rythme-là, l'affectation est longue, surtout si votre tableau compte plus que quelques éléments. C'est pourquoi on utilise la boucle itérative `for`. L'exemple suivant entre une série de nombre en ajoutant 1 à chaque fois. Il s'agit d'un exemple rapide et simple, mais on peut imaginer faire un calcul ou demander une valeur à l'utilisateur à chaque boucle.

#### Exemple 11.3 :

```
var MonTableau = new Array (5) ;
for ( var i = 0 ; i < 5 ; i++) {
    MonTableau [i] = i;
    document.write (MonTableau[i]);
}
```

#### ⇒ Résultat

### 11.3.Accès aux données d'un tableau

On accède aux données d'un tableau en indiquant le numéro de l'élément entre crochets. On affecte cette valeur à une variable, par exemple. On peut aussi la rentrer comme paramètre d'une fonction.

#### Syntaxe :

```
variable1 = new Array(x) ;
    variable1 [i] = y ;
variable2 = variable1 [i] ;
```

#### Exemple 11.4 :

```
var MonTableau = new Array (2) ;
MonTableau [0] = 7 ;
var element = MonTableau [0] ;
document.write (MonTableau [0]) ;
```

#### ⇒ Résultat

### 11.4.Tableau à 2 dimensions

Pour créer un tableau à 2 dimensions, on utilise l'astuce suivante : on déclare chaque élément comme un nouveau tableau.

#### Syntaxe :

```
variable = new Array (x) ;
    variable [i] = new Array (y) ;
```

### Exemple 11.5 :

```
var MonTableau = new Array (2) ;  
MonTableau [0] = new Array (7) ;
```

Il est bien entendu plus rapide d'utiliser une instruction itérative pour créer ce tableau à 2 dimensions.

### Exemple 11.6 :

```
var MonTableau = new Array (5) ;  
for ( var i = 0 ; i < 5 ; i++) {  
    MonTableau [i] = new Array (3);  
}
```

Avec ce système, on peut créer un tableau à 3,4 dimensions ou plus, bien que l'utilité en soit quelque peu douteuse...

## 11.5. Propriétés et méthodes

Comme tout objet, l'objet Array possède une propriété et des méthodes qui s'avèrent assez utiles.

### 11.5.1. Propriété

L'objet Array ne possède qu'une propriété - `length` - qui désigne le nombre d'éléments du tableau.

#### Syntaxe :

```
variable = new Array (x) ;  
y = variable.length ;
```

### Exemple 11.7 :

```
var MonTableau = new Array (2) ;  
document.write (MonTableau.length) ;
```

#### ⇒ Résultat

### 11.5.2.Méthodes

L'objet Array possède plusieurs méthodes qui permettent de manier les éléments du tableau.

Syntaxe :

```
tableau.méthode() ;
```

- `join ( séparateur )` : regroupe tous les éléments du tableau en une seule chaîne. Chaque élément est séparé par un séparateur. Si celui-ci n'est pas précisé, ce sera une virgule.
- `reverse ()` : inverse l'ordre des éléments sans les trier.
- `sort ()` : Renvoie les éléments par ordre alphabétique, s'ils sont de même nature.
- `concat(tableau)` : concatène le tableau passé en paramètre avec celui de la méthode.
- `pop()` : supprime le dernier élément du tableau.
- `push(élément1,...)` : ajoute l(es) élément(s) passé(s) en paramètre à la fin du tableau.
- `shift()` : supprime le premier élément du tableau.
- `slice(début,fin)` : renvoie les éléments contenus entre la position supérieure ou égale à début et inférieure à fin.
- `splice(début, nombre, éléments)` : supprime les éléments à partir de la position début et sur nombre de position. Les éléments sont remplacés par ceux fournis en paramètre (facultatif).
- `unshift(élément1,...)` : ajoute l(es) élément(s) passé(s) en paramètre au début du tableau.

Exemple 11.8 :

```
var MonTableau = new Array (4) ;
MonTableau [0] = "Moi" ;
MonTableau [1] = "Toi" ;
MonTableau [2] = "Lui" ;
MonTableau [3] = "Elle" ;
MonTableau.reverse() ;
document.write (MonTableau.join(' ')) ;
MonTableau.sort() ;
document.write ("<br>" + MonTableau.join(' ')) ;
```

⇒ Résultat

## 11.6.Exemple concret

Le but de cet exemple est de remplir et afficher les valeurs à 0.1 près comprises entre 0 et 10 exclu. Pour cela, on utilise un tableau à deux dimensions, dont la première indique la partie entière du nombre, et la deuxième dimension indique la partie décimale. On va d'abord créer le tableau, puis le remplir et enfin l'afficher.

**Exemple 11.9 :**

```
<script language="Javascript">
var Tableau = new Array (10) ; // objet tableau
for (i = 0; i < 10; i++) { // pour chaque élément...
    Tableau[i] = new Array(10); // on redéfinit un tableau
}
// remplissage du tableau
for (i = 0; i < 10; i++) { /* pour chaque élément de la première
dimension*/
    for (j = 0; j < 10; j++) { /* pour chaque élément de la
seconde dimension*/
        Tableau[i][j] = i + "." + j; //on remplit l'élément
    }
}
// affichage du tableau
for (i = 0; i < 10; i++) { /* pour chaque élément de la première
dimension */
    for (j = 0; j < 10; j++) { /* pour chaque élément de la
seconde dimension */
        document.write( Tableau[i][j] + " ; " ); /*on affiche
l'élément, avec un point virgule */
    }
    document.write ("<br>"); /* à chaque unité, on revient à la
ligne. */
}
</script>
```

⇒ **Résultat**

## 11.7.Exercice

Le but de l'exercice est d'afficher les entiers compris entre 1 et 10 inclus dans l'ordre décroissant. Vous utiliserez pour cela un tableau de 10 éléments.

⇒ **Solution**

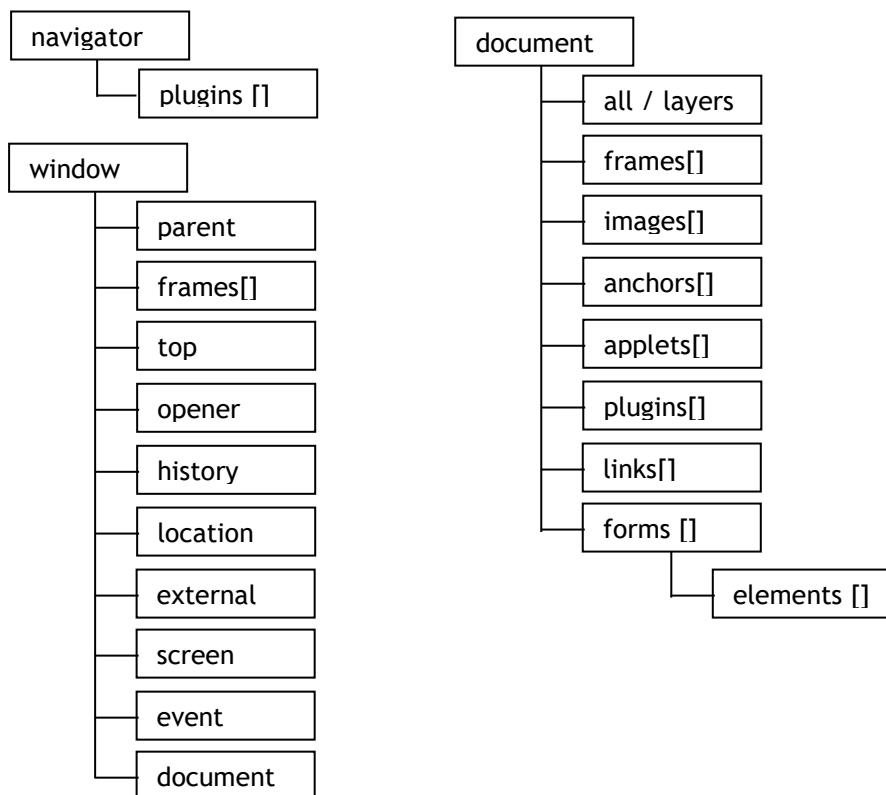
## 12.LES OBJETS DU NAVIGATEUR

### 12.1.Généralités

Nous avons vu les classes prédefinies en JS. Il existe aussi plusieurs objets en JS rattachés à la fenêtre, à la page et au navigateur. Il sont appelés **window**, **document** et **navigator**. Ce sont 3 classes distinctes - sans lien entre elle - puisque l'héritage n'existe pas en JS. Nous étudierons chaque classe l'une après l'autre. Cependant, l'objet **document** fait partie de l'objet **window**.

### 12.2.Arborescence

Je présente ici l'arborescence des objets de JS. Elle comprend les 3 objets principaux ainsi que tous les « sous-objets » contenus dans chaque objet principal.



## 13.L'OBJET NAVIGATOR

Il s'agit - comme son nom l'indique - de votre navigateur, ou votre browser. Les deux principaux sont sûrement Microsoft Internet Explorer et Netscape. L'objet navigator possède toutes les caractéristiques de votre navigateur ainsi que certaines de votre ordinateur. Cela peut s'avérer utile pour tester la compatibilité de certains codes avec un browser.

L'objet `navigator` étant intégré dans le langage, il n'est pas nécessaire de créer une instance de cette classe.

### 13.1.Les propriétés de `navigator`

Toutes ces propriétés ne font que donner des informations sur votre type de navigateur. Pour y accéder, on suit la syntaxe habituelle d'un objet.

**Syntaxe :**

```
variable = navigator.propriété ;
```

Les propriétés qui suivent fonctionnent sous tous les navigateurs. Les compatibilité ne seront évoqués que pour Netscape et Internet Explorer.

- `appCodeName` : nom de code du navigateur.
- `appName` : nom complet du navigateur.
- `appVersion` : numéro de version du navigateur ainsi que d'autres informations de plate-forme.
- `userAgent` : informations de `appCodeName` et de `appVersion` réunies.

Certaines propriétés ne fonctionnent qu'avec Microsoft Internet Explorer. Veillez à tester le type de browser avant d'exécuter le script !

- `appMinorVersion` : numéro de version mineure.
- `browserLanguage` : code langue du browser.
- `userLanguage` : code langue de l'utilisateur.
- `systemLanguage` : code langue du système d'exploitation.
- `cpuClass` : classe du processeur.
- `platform` : code type de plate-forme (pc, mac, linux ...).
- `cookieEnabled` : booléen qui indique si l'utilisateur accepte les cookies.
- `onLine` : booléen qui indique si le poste est connecté à Internet.

Une propriété existe sous Netscape pour remplacer la propriété browserLanguage d'Internet Explorer.

- language : code langue du browser.

## 13.2.Les méthodes de navigator

Dans cette partie, je vous cite les méthodes de navigator, mais malheureusement, je ne suis pas en mesure de vous expliquer leur utilité. Si vous le savez, merci de me le faire savoir.

Syntaxe :

```
variable = navigator.méthode() ;
```

- javaEnabled()
- taintEnabled()

## 13.3.Les objets de navigator

L'objet navigator contient un autre objet, qui n'hérite pas de lui. Il s'agit du tableau plugins, qui n'est reconnu que par Netscape. Il s'agit de la liste de tous les plugins installés. Personnellement, je ne vois pas l'utilité de cet objet, mais par souci didactique, je vous le décris.

Syntaxe :

```
variable = navigator.plugin.propriété ;
```

- plugins.length : nombre de plugins
- plugins[i].name : nom du plugin n° i.
- plugins[i].filename : nom de l'exécutable du plugin n° i.
- plugins[i].description : description du plugin n° i.
- plugins[i].length : nombre de types de fichiers supportés par le plugin n° i.
- plugins[i][j].type : type n° j du plugin n° i.
- plugins[i][j].suffixes : Extensions des fichiers du type n° j du plugin n° i.

## 13.4.Exemple

Le but de l'exemple 13.1 est d'afficher les informations du navigateur, en fonction de celui-ci. On commencera donc par tester le type de navigateur. L'exemple 13.1 n'est pas commenté car il n'a rien de nouveau excepté les propriétés vues ci-dessus.

### Exemple 13.1 :

```
<script language="Javascript">
document.write (navigator.appCodeName + "<br>");
document.write (navigator.appName + "<br>");
document.write (navigator.appVersion + "<br>");
document.write (navigator.userAgent + "<br>");
if (navigator.appName == "Microsoft Internet Explorer") {
    document.write (navigator.appMinorVersion + "<br>");
    document.write (navigator.browserLanguage + "<br>");
    document.write (navigator.userLanguage + "<br>");
    document.write (navigator.systemLanguage + "<br>");
    document.write (navigator.cpuClass + "<br>");
    document.write (navigator.platform + "<br>");
    document.write (navigator.cookieEnabled + "<br>");
    document.write (navigator.onLine + "<br>");
}
else {
    document.write (navigator.language + "<br>");
}
</script>
```

### ⇒ Résultat

Ce chapitre ne comporte pas d'exercice car le contenu est assez restreint. L'exemple 13.1 devrait suffire à illustrer l'objet navigator.

## 14.L'OBJET WINDOW

Il s'agit - comme son nom l'indique - de la fenêtre du browser en cours d'exécution. Tous les éléments de la fenêtre sont des propriétés ou des méthodes de `window`.

L'objet `window` étant intégré dans le langage, il n'est pas nécessaire de créer une instance de cette classe.

### 14.1.Les propriétés de `window`

Toutes ces propriétés correspondent à des éléments de la fenêtre ouverte. Elles permettent de changer quelques détails sympathiques dans le visuel d'une page web. Pour y accéder, on suit la syntaxe habituelle d'un objet.

Syntaxe :

```
window.propriété;
```

- `defaultStatus` : le texte par défaut affiché dans la barre d'état.
- `status` : le texte à afficher dans la barre d'état, prioritaire par rapport à `defaultStatus`.
- `name` : le nom de la fenêtre
- `screenTop` : ordonnée du point supérieur gauche de la fenêtre.
- `screenLeft` : abscisse du point supérieur gauche de la fenêtre.
- `closed` : booléen indiquant si la fenêtre est fermée.

### 14.2.Les méthodes de `window`

#### 14.2.1.Généralités

L'objet `window` possède de nombreuses méthodes, dont certaines que nous avons déjà vu précédemment - les boîtes de dialogue -, et qui peuvent offrir quelques atouts à votre page web. La plupart renvoient une valeur à la variable qui appelle la fonction.

Syntaxe :

```
variable = window.méthode();
```

Certaines propriétés ne nécessitent pas de préciser le suffixe `window.` pour fonctionner. C'est notamment le cas des boîtes de dialogue.

### 14.2.2.Liste des méthodes

- `alert ('texte')` : boite de message avec un bouton unique.
- `prompt ('texte', 'valeur_défaut')` : boite d'invite avec un texte informatif et une zone de texte avec une valeur par défaut facultative.
- `confirm ('texte')` : boite de confirmation avec un texte informatif et deux boutons « OK » et « annuler ».
- `print ('texte')` : afficher le texte dans la page.
- `open ('URL', 'nom', options)` : ouvre une page pop-up avec les caractéristiques données en paramètres.
- `close ()` : ferme la fenêtre.
- `focus ()` : donne le focus à la page.
- `blur ()` : enlève le focus à la page.
- `moveBy (x,y)` : déplacement relatif du coin supérieur gauche de la fenêtre.
- `moveTo (x,y)` : déplacement absolu du coin supérieur gauche de la fenêtre.
- `resizeBy (x,y)` : redimensionnement relatif de la fenêtre.
- `resizeTo (x,y)` : redimensionnement absolu de la fenêtre.
- `scrollBy (x,y)` : scroll relatif.
- `scrollTo (x,y)` : scroll absolu.
- `setTimeOut ('fonction', temps)` : déclenche une minuterie en millisecondes.
- `clearTimeout('timer')` : suspend la minuterie.
- `stopTimeOut ('timer')` : arrête une minuterie.
- `setInterval(code, délai)` : exécute le code - sous forme de String - passé en paramètre à chaque fois que le délai est écoulé.
- `clearInterval(timer)` : arrête la minuterie définie avec setInterval().
- `stop()` : arrête le chargement de la page.
- `home ()` : ouvre la page de démarrage de l'internaute.

### 14.2.3.Exemples

Voici, pour illustrer les liste proposée ci-dessus, deux façons de faire un chronomètre. La première utilise la méthode `setInterval()` - moins connue - , ce qui économise un ligne de code. La seconde utilise la méthode `setTimeout()` - plus connue - qui doit être rappelée à chaque boucle. Le reste du code est identique. Il sert à incrémenter à chaque fois les secondes, et permet de rendre l'affichage un peu plus agréable.

### Exemple 14.1 :

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
var seconds = 0;
var minutes = 0;
var hours = 0;
function chrono () {
    if (seconds == 60) {
        seconds = 0;
        minutes++;
    }
    if (minutes == 60) {
        minutes = 0;
        hours++;
    }
    h = hours;
    m = minutes;
    s = seconds;
    if (s < 10) s = "0" + s;
    if (m < 10) m = "0" + m;
    if (h < 10) h = "0" + h;
    document.form1.chrono.value = h + ":" + m + ":" + s;
    seconds++;
}
</script>
</HEAD>
<BODY onLoad="window.setInterval('chrono()',1000)">
<form name="form1">
<center><input type="text" value="" name="chrono" size=6></center>
</form>
</BODY>
</HTML>
```

### ⇒ Résultat

### Exemple 14.2 :

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
var seconds = 0;
var minutes = 0;
var hours = 0;
function chrono () {
    if (seconds == 60) {
        seconds = 0;
        minutes++;
    }
    if (minutes == 60) {
        minutes = 0;
        hours++;
    }
    h = hours;
    m = minutes;
    s = seconds;
    if (s < 10) s = "0" + s;
    if (m < 10) m = "0" + m;
    if (h < 10) h = "0" + h;
    document.form1.chrono.value = h + ":" + m + ":" + s;
    seconds++;
    window.setTimeout('chrono()',1000);
}
</script>
</HEAD>
<BODY onLoad="chrono();">
<form name="form1">
<center><input type="text" value="" name="chrono" size=6></center>
</form>
</BODY>
</HTML>
```

### ⇒ Résultat

## 14.3.Les évènements de window

On peut rattacher certains évènements à l'objet window. Il seront placés dans la balise `<body>`, grâce au gestionnaire d'événement `onEvent`.

### Syntaxe :

```
<body onEvent="code">
```

- `load` : fin de chargement de la page.
- `unLoad` : déchargement de la page.
- `focus` : prise de focus de la fenêtre ou d'un de ses éléments.
- `blur` : perte de focus de la fenêtre ou d'un de ses éléments.
- `resize` : redimensionnement de la fenêtre.

Pour plus de précisions, consultez le chapitre 7, relatif au évènements de Javascript.

## 14.4.Les objets de window

L'objet `window` contient plusieurs autres objets assez réduits que je présente dans cette partie.

**Syntaxe :**

```
variable = window.objet.propriété ;  
variable = window.objet.méthode() ;
```

### 14.4.1.L'objet frames

Il s'agit d'un tableau contenant toutes les frames déclarées dans la page. Il ne possède ni propriétés ni méthodes.

**Syntaxe :**

```
variable = window.frames[i] ;
```

### 14.4.2.L'objet parent

Il s'agit de la page principale, qui contient la déclaration de toutes les frames. Il possède les mêmes attributs que l'objet `window`.

**Syntaxe :**

```
variable = window.parent.méthode() ;  
variable = window.parent.propriété ;
```

### 14.4.3.L'objet top

Il s'agit de la page de plus haut niveau. Il possède les mêmes attributs que l'objet `window`.

**Syntaxe :**

```
variable = window.top.méthode() ;  
variable = window.top.propriété ;
```

#### 14.4.4.L'objet opener

Il s'agit de la page responsable de l'ouverture de la page courante. Il possède les mêmes attributs que l'objet window.

**Syntaxe :**

```
variable = window.opener.méthode() ;  
variable = window.opener.propriété ;
```

#### 14.4.5.L'objet history

Il s'agit de l'historique des pages consultées. Il possède une propriété et 3 méthodes. Voici la propriété et les méthodes de cet objet, qui se déclarent selon la syntaxe objet.

**Syntaxe :**

```
variable = window.history.méthode() ;  
variable = window.history.propriété ;
```

- length : le nombre d'entrées de l'historique.
- back () : permet de retourner à la page précédente.
- forward () : permet d'aller à la page suivante.
- go (numéro) : permet d'aller à la page du numéro passé en paramètre.

#### 14.4.6.L'objet location

Il s'agit de toutes les informations contenues dans l'URL de la page en cours. Voici la propriété et les méthodes de cet objet, qui se déclarent selon la syntaxe objet. L'objet window.location renvoie l'URL complète de la page en cours.

**Syntaxe :**

```
variable = window.location.méthode() ;  
variable = window.location.propriété ;
```

Les propriétés suivantes renvoient des informations plus précises concernant l'URL. Je ne précise pas à quoi elle renvoient exactement car ce n'est pas très utile dans ce cours.

- hash
- host
- hostName
- pathName
- href
- port
- protocole
- search : renvoie la partie de l'URL située après le « ? ».

Il existe deux propriétés de l'objet window.location, qui concernent toutes deux le rechargement de la page.

- reload () : recharge la page. Ne fonctionne pas sous tous les navigateurs.
- replace (page) : recharge la page en cours sans modifier l'historique.

#### 14.4.7.L'objet screen

Il s'agit de toutes les informations du système d'affichage de l'utilisateur. Il y a 4 propriétés rattachées à cet objet.

Syntaxe :

```
variable = window.screen.propriété ;
```

- availHeight : hauteur en pixels de la zone utilisable pour l'affichage.
- availWidth : largeur en pixels de la zone utilisable pour l'affichage.
- height : hauteur de la fenêtre en pixels - contient barres de menus, d'état, de titre et de scrolling - .
- width : largeur de la fenêtre en pixels - contient barres de menus, d'état, de titre et de scrolling - .
- colorDepth : Contient la profondeur de couleur en nombre de bits.

#### 14.4.8.L'objet event

Il s'agit d'un objet propre à Microsoft Internet Explorer. Il renvoie le type d'événement qui a eu lieu.

Syntaxe :

```
variable = window.event.propriété ;
```

- `altKey` : renvoie le code du caractère frappé au clavier.
- `button` : renvoie le type de clic de souris effectué.

#### 14.4.9.L'objet `external`

Il s'agit d'un objet propre à Microsoft Internet Explorer. Il permet d'accéder aux propriétés du navigateur.

**Syntaxe :**

```
variable = window.external.propriété ;
```

- `AddFavorite(URL,titre)`: Ajoute une ligne à la liste des favoris. Demande confirmation à l'internaute.

### 14.5.Exemple concret

Dans l'exemple 14.1, on a voulu afficher les attributs de la fenêtre (hauteur/largeur, place affichable) ainsi que l'URL en cours. Il n'y a pas de commentaires car les instructions utilisées sont détaillées ci-dessus.

**Exemple 14.3 :**

```
<script language="Javascript">
document.write (window.location + "<br>");
document.write (window.screen.availHeight + "<br>");
document.write (window.screen.availWidth + "<br>");
document.write (window.screen.height + "<br>");
document.write (window.screen.width + "<br>");
</script>
```

⇒ **Résultat**

### 14.6.Exercice

Le but de l'exercice est de donner à l'utilisateur la possibilité de modifier la fenêtre à l'aide de boutons. Il pourra redimensionner et revenir à la taille initiale, la déplacer et la remettre en position initiale, lui enlever le focus, et la fermer. Lors de la fermeture, l'utilisateur sera averti par une boîte de message.

⇒ **Solution**

## 15.L'OBJET DOCUMENT

Il s'agit - comme son nom l'indique - de la page en cours d'exécution. Tous les éléments de la page sont des propriétés ou des méthodes de document.

L'objet document étant intégré dans le langage, il n'est pas nécessaire de créer une instance de cette classe.

L'objet document fait partie de l'objet window, mais il n'est pas nécessaire de préciser le suffixe "window."

### 15.1.Les propriétés de document

Toutes ces propriétés correspondent à des éléments de la page ouverte. Cela permet d'uniformiser et de changer quelques détails de votre page. On les utilise grâce à la syntaxe habituelle.

Syntaxe :

`document.propriété;`

- `bgColor` : couleur du fond.
- `fgColor` : couleur du texte.
- `linkColor` : couleur des liens ni actifs ni visités.
- `alinkColor` : couleurs des liens actifs.
- `vlinkColor` : couleurs des liens visités.
- `cookie` : chaîne de caractères contenant les cookie.
- `lastModified` : date de dernière modification du fichier source
- `referrer` : adresse de la page par laquelle la page en cours a été appelée.
- `title` : titre du document, indiqué par les balises `<title>...</title>`. N'est modifiable qu'avec Microsoft Internet Explorer.
- `fileSize` : taille de la page en octets.
- `defaultCharset` : jeu de caractère du document chargé.
- `mimeType` : type du document chargé.
- `URLUnencoded` : URL complète de la page, avec les caractères spéciaux encodés.
- `URL` : URL de la page.
- `protocol` : protocole de chargement de la page.
- `domain` : domaine de l'URL complète de la page.

### Exemple 15.1 :

```
<script language="Javascript">
document.write ("Taille du fichier : " + document.fileSize + "<br>");
document.write ("Type mime : " + document.mimeType + "<br>");
document.write ("Jeu de caractères : " + document.defaultCharset +
"<br>");
document.write ("URL décodée : " + document.URLUnencoded + "<br>");
document.write ("URL : " + document.URL + "<br>");
document.write ("Protocole : " + document.protocol + "<br>");
document.write ("Dernière modification : " + document.lastModified +
"<br>");
</script>
```

#### ⇒ Résultat

## 15.2.Les méthodes de document

L'objet `document` possède de plusieurs méthodes, dont certaines que nous avons déjà vu précédemment.

#### Syntaxe :

```
variable = document.méthode();
```

Certaines propriétés ne nécessitent pas de préciser le suffixe `window`. pour fonctionner. C'est notamment le cas des boites de dialogue.

- `write ('texte')` : affiche le texte et le code HTML dans la page courante.
- `getSelection ()` : renvoie le texte qui a été sélectionné dans la page.
- `handleEvents` : créé un gestionnaire d'évènement.
- `captureEvents` : détecte un évènement.
- `open ()` : ouvre une nouvelle fenêtre de votre browser.
- `close ()` : ferme le flux d'affichage externe.
- `getElementById(ID)` : renvoie un objet HTML en fonction de son ID. A ne pas confondre avec le Name.
- `getElementsByName(nom)` : renvoie un objet HTML en fonction de son name.
- `getElementsByTagName(type)` : renvoie un tableau de toutes les balises HTML du type passé en paramètre.

### Exemple 15.2 :

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
function f() {
    var tab = document.getElementsByTagName ("input");
    var result;
    for (i = 0; i < tab.length; i++) {
        result = result + " " + (tab[i].value);
    }
    document.form1.result.value = result;
}
</script>
</HEAD>
<BODY>
<form name="form1">
<input type="text" name="1" value=" "><br/>
<input type="text" name="2" value=" "><br/>
<input type="text" name="3" value=" "><br/>
<input type="text" name="4" value=" "><br/>
<input type="text" name="5" value=" "><br/>
<input type="button" value="click!" name="click" onClick="f () ; "><br/>
<textarea rows=4 cols=40 name="result"></textarea>
</form>
</BODY>
</HTML>
```

### ⇒ Résultat

## 15.3.Les évènements de document

On peut rattacher certains évènements à l'objet `document`.

- `onClick` : clic de souris sur un élément de la page.
- `onDblClick` : le double clic de souris.
- `onKeyPress` : la frappe d'une touche de clavier.

Pour plus de précisions, consultez le chapitre 7, relatif aux évènements de Javascript.

## 15.4.Les objets de document

L'objet `document` contient plusieurs autres objets assez réduits que je présente dans cette partie.

**Syntaxe :**

```
variable = document.objet.propriété ;  
variable = document.objet.méthode() ;
```

Nous préciserons ici que tout élément de la page est en soi un objet de document. On y accède selon la syntaxe ci-dessus. Il serait trop lourd de donner la liste des éléments, qui sont répertoriés dans tout cours HTML digne de ce nom.

#### 15.4.1.L'objet `all`

Il s'agit d'un tableau contenant tous les calques déclarés dans la page dans les balises `<div>...</div>`. Il s'agit d'une particularité de Microsoft Internet Explorer. Il possède les propriétés de la balise `<div>`.

**Syntaxe :**

```
document.all["calque"].style.property = x ;
```

#### 15.4.2.L'objet `layers`

C'est l'équivalent de l'objet `all` pour Netscape, pour les calques des balises `<div>` ou `<layer>`. Il possède les propriétés de la balise `<div>`.

**Syntaxe :**

```
document.layers["calque"].style.property = x ;
```

#### 15.4.3.L'objet `forms`

Il s'agit d'un tableau contenant tous les formulaires du document. Il possède une propriété - `elements[]` - qui est un tableau de tous les éléments de formulaire.

**Syntaxe :**

```
document.forms[i].elements[j] ;
```

#### 15.4.4.L'objet `anchors`

Il s'agit d'un tableau contenant toutes les ancrés - les balises `<a>` - de la page courante. Il ne possède ni propriétés ni méthodes.

Syntaxe :

document.anchors[i] ;

#### 15.4.5.L'objet `images`

Il s'agit d'un tableau contenant toutes les images - les balises `<img>` - de la page courante. Il ne possède ni propriétés ni méthodes. Cela permet de faire des effets, par exemple des rollovers, sur les images.

Syntaxe :

document.images[i] ;

#### 15.4.6.L'objet `applets`

Il s'agit d'un tableau contenant toutes les applets java déclarés dans la page courante. Il ne possède ni propriétés ni méthodes.

Syntaxe :

document.applets[i] ;

#### 15.4.7.L'objet `plugins`

Il s'agit du tableau `plugins`, qui n'est reconnu que par Netscape. Il s'agit de la liste de tous les plugins installés. Il est aussi rattaché à l'objet `navigator`. Je ne préciserai pas une nouvelle fois les propriétés. Pour cela, voyez la section 16.3.Les objets de `navigator`.

#### 15.4.8.L'objet `frames`

Il s'agit d'un tableau contenant toutes les frames déclarées dans la page courante. Il ne possède ni propriétés ni méthodes.

Syntaxe :

document.frames[i] ;

## 15.5.Exercice

Vous devez créer un formulaire avec un texte, dans lequel on pourra entrer une couleur dans une zone de texte et avec un click, on change soit la couleur du texte, soit la couleur du fond.

⇒ **Solution**

## 16.LES OBJET DU NOYAU JAVASCRIPT

### 16.1.Généralités

Le langage JS n'est pas un langage orienté objet, mais il possède une partie des concepts d'un langage objet comme le C++. Nous avons déjà vu précédemment ce concept<sup>14</sup>. Certains de ces objets ne vous sont pas inconnus. Vous connaissez l'objet `boolean`. Nous avons vu l'objet `Array`. Nous verrons dans ci-après les 4 autres objets intégrés au JS. Il s'agit de l'objet `Math`, de l'objet `String`, de l'objet `Date` et de l'objet `Image`.

### 16.2.Quelques précisions

Certains de ces objets seront déjà définis, comme l'objet `Math`. L'objet `String` est défini par une variable - il s'agit des chaînes de caractères - et n'a pas besoin de constructeur. Les objets `Date` et `Image` nécessitent un constructeur, intégré au langage, mais qu'il faut appeler selon la syntaxe habituelle. Certains objets `Image` existent déjà dans votre page, sans que vous le sachiez. Il s'agit des balises `<img>`. Ces dernières n'ont donc pas besoin de faire appel au constructeur.

---

<sup>14</sup> Pour les étourdis, il s'agit du chapitre 8.NOTION OBJET

# 17.L'OBJET MATH

## 17.1.Fonctions

Les fonctions mathématiques usuelles ont été transcris en langage JS à travers l'objet **Math**. La plupart des fonctions s'utilisent selon la même syntaxe.

Syntaxe :

```
var1 = math.fonction(paramètres) ;
```

### 17.1.1.Fonctions diverses

Ces fonctions étant simples, je ne fournirai pas d'exemples.

- `abs(x)` : renvoie la valeur absolue de `x`.
- `ceil(x)` : renvoie l'entier supérieur à `x`.
- `floor(x)` : renvoie l'entier inférieur à `x`.
- `round(x)` : renvoie l'entier le plus proche de `x`.
- `max(x,y)` : renvoie le plus grand nombre entre `x` et `y`.
- `min(x,y)` : renvoie le plus petit nombre entre `x` et `y`.
- `pow(x,y)` : renvoie le nombre `x` à la puissance `y`.
- `random(x,y)` : renvoie un nombre aléatoire entre 0 et 1.
- `sqrt(y)` : renvoie la racine carrée de `x`.

### 17.1.2.Fonctions trigonométriques

- `sin(x)`
- `asin(x)`
- `cos(x)`
- `acos(x)`
- `tan(x)`
- `atan(x)`

### 17.1.3.Fonctions logarithmiques

- `exp(x)`
- `log(x)`

## 17.2. Constantes

- `math.PI`
- `math.LN2`
- `math.LN10`
- `math.E`
- `math.LOG2E`
- `math.LOG10E`

## 17.3. Simplification

Si vous utilisez beaucoup l'objet `math`, l'écriture deviendra vite pénible. Il est possible de ne pas avoir à écrire le mot "`math`" à chaque fois. Il suffit d'encadrer la zone de code par des accolades et l'expression `with (math)`.

Syntaxe :

```
with (math) {  
    code...  
}
```

Exemple 17.1 :

```
with (Math) {  
    x = sqrt (238) ;  
    y = sin (x) ;  
    document.write(y) ;  
}
```

⇒ Résultat

## 17.4. Exercice

L'exercice est relativement simple. Il consiste à afficher la racine carrée de tous les x entiers, compris dans [0 ; 20]. Le résultat doit être arrondi à l'entier.

⇒ Solution

## 18.L'OBJET STRING

### 18.1.Généralités

Un objet **string** est une chaîne de caractères. Il possède une propriété et 7 méthodes. Cet classe permet la **manipulation des caractères** qui s'avère très utile en JS. Il est à préciser que l'objet **string** ne se construit pas comme les autres objets. Une chaîne de caractère est en soi un objet **string**.

### 18.2.La propriété

Il s'agit de la longueur de la chaîne, "**length**".

Syntaxe :

```
variable1 = variable2.length ;  
variable = ("chaine").length ;
```

Exemple 18.1 :

```
x = "Mon château" ;  
y = x.length ;  
document.write(y) ;
```

⇒ Résultat

### 18.3.Les méthodes

#### 18.3.1.CharAt ()

Cette méthode renvoie le caractère situé à la position x fournie en paramètre. Le numéro de la position est compris entre 0 et la longueur de la chaîne -1.

Syntaxe :

```
chainel = chaine2.charAt(x) ;  
chainel = ("chaine").charAt(x) ;  
chainel = charAt(chaine2,x) ;
```

### Exemple 18.2 :

```
x = "Mon Château" ;
y = x.charAt(4) ;
document.write(y) ;
```

⇒ Résultat

### 18.3.2.FromCharCode ()

Cette méthode renvoie les nombre ASCII passés en paramètres sous forme de chaîne de caractère.

Syntaxe :

```
chaine.fromCharCode(i1,i2,i3)
```

### Exemple 18.3 :

```
document.write(y.fromCharCode(12,105,123,104)) ;
```

⇒ Résultat

### 18.3.3.charCodeAt ()

Cette méthode renvoie le code ASCII du caractère présent à la position indiquée en paramètre.

Syntaxe :

```
variable = chaine1.charCodeAt(position)
```

### Exemple 18.4 :

```
y = "lepape@le-vatican.com";
document.write(y.charCodeAt(6)) ;
```

⇒ Résultat

### 18.3.4.indexOf ()

Cette méthode renvoie la première position d'une chaîne partielle dans une autre chaîne en partant de gauche, à partir de la position x indiquée en paramètre. Si elle n'est pas présente, la méthode renvoie -1. Le numéro de la position est compris entre 0 et la longueur de la chaîne -1.

**Syntaxe :**

```
variable = chaine.indexOf  
(chaine partielle, x)
```

**Exemple 18.5 :**

```
x = "maman" ;  
y = x.indexOf("ma", 0) ;  
document.write(y) ;
```

⇒ **Résultat**

### 18.3.5.LastIndexOf ()

Cette méthode renvoie la première position d'une chaîne partielle dans une autre chaîne en partant de gauche, à partir de la position x indiquée en paramètre. Si elle n'est pas présente, la méthode renvoie -1. Le numéro de la position est compris entre 0 et la longueur de la chaîne -1.

**Syntaxe :**

```
variable = chaine.lastIndexOf  
(chaine partielle, x)
```

**Exemple 18.6 :**

```
x = "maman" ;  
y = x.lastIndexOf("ma", 4) ;  
document.write(y) ;
```

⇒ **Résultat**

### 18.3.6.Substring ()

Cette méthode renvoie la sous-chaîne comprise entre les positions x et y indiquées en paramètres, dans la chaîne principale.

**Syntaxe :**

```
variable = chaine.substring (x, y)
```

**Exemple 18.7 :**

```
x = "maman" ;  
y = x.substring(1, 3) ;  
document.write(y) ;
```

⇒ **Résultat**

### 18.3.7.Substr ()

Cette méthode renvoie le texte une sous-chaîne de la String de la méthode, à partir du début et sur n caractères..

Syntaxe :

```
variable = chaine.substr(début, n)
```

Exemple 18.8 :

```
y = "lepage@le-vatican.com";
document.write(y.substr(5, 2)) ;
```

⇒ Résultat

### 18.3.8.Slice ()

Equivalent à substring(). La fin est facultative.

Syntaxe :

```
variable = chaine.slice(début, fin)
```

Exemple 18.9 :

```
y = "lepage@le-vatican.com";
document.write(y.slice(7)) ;
```

⇒ Résultat

### 18.3.9.Split ()

Cette méthode renvoie un tableau de sous-chaînes découpées selon le séparateur passé en paramètres.

Syntaxe :

```
variable = chaine.split(séparateur)
```

Exemple 18.10 :

```
x = "lepage@le-vatican.com" ;
y = x.split("@") ;
document.write(y[0] + "<br>" + y[1]) ;
```

⇒ Résultat

### 18.3.10.Concat ()

Cette méthode renvoie la concaténation de la chaîne passée en paramètre avec celle de la méthode.

Syntaxe :

```
variable = chaine1.concat(chaine2)
```

Exemple 18.11 :

```
x = "Ecrivez-moi à " ;
y = "lepape@le-vatican.com";
z = x.concat(y) ;
document.write(z) ;
```

⇒ Résultat

### 18.3.11.ToLowerCase ()

Cette méthode renvoie la chaîne avec tous les caractères en minuscules

Syntaxe :

```
variable = chaine.toLowerCase()
```

Exemple 18.12 :

```
x = "MAMAN" ;
y = x.toLowerCase() ;
document.write(y) ;
```

⇒ Résultat

### 18.3.12.ToUpperCase ()

Cette méthode renvoie la chaîne avec tous les caractères en majuscules

Syntaxe :

```
variable = chaine.toUpperCase()
```

### Exemple 18.13 :

```
x = "Maman" ;
y = x.toUpperCase() ;
document.write(y) ;
```

#### ⇒ Résultat

### 18.3.13.FontColor ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <font color=couleur>...</font>.

#### Syntaxe :

```
variable = chaine1.fontColor(couleur)
```

### Exemple 18.14 :

```
y = "lepape@le-vatican.com";
document.write(y.fontColor("blue")) ;
document.write(y.fontColor("red")) ;
```

#### ⇒ Résultat

### 18.3.14.FontSize ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <font size=taille>...</font>.

#### Syntaxe :

```
variable = chaine1.fontSize(taille)
```

### Exemple 18.15 :

```
y = "lepape@le-vatican.com";
document.write(y.fontSize("16")) ;
document.write(y.fontSize("8")) ;
```

#### ⇒ Résultat

### 18.3.15.Fixed ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <pre>...</pre>.

---

**Syntaxe :**

```
variable = chaine1.fixed()
```

**Exemple 18.16 :**

```
y = "lepape@le-vatican.com";
document.write(y.fixed()) ;
```

⇒ **Résultat**

### 18.3.16.Bold ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises `<b>...</b>`.

**Syntaxe :**

```
variable = chaine1.bold()
```

**Exemple 18.17 :**

```
y = "lepape@le-vatican.com";
document.write(y.bold()) ;
```

⇒ **Résultat**

### 18.3.17.Strike ()

Cette méthode renvoie le texte de l'objet barré.

**Syntaxe :**

```
variable = chaine1.strike()
```

**Exemple 18.18 :**

```
y = "lepape@le-vatican.com";
document.write(y.strike()) ;
```

⇒ **Résultat**

### 18.3.18.Sub ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises `<sub>...</sub>`.

**Syntaxe :**

```
variable = chaine1.sub()
```

**Exemple 18.19 :**

```
y = "lepape@le-vatican.com";
document.write(y.sub()) ;
```

⇒ **Résultat**

### 18.3.19.Big ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <big>...</big>.

**Syntaxe :**

```
variable = chaine1.big()
```

**Exemple 18.20 :**

```
y = "lepape@le-vatican.com";
document.write(y.big()) ;
```

⇒ **Résultat**

### 18.3.20.Sup ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <sup>...</sup>.

**Syntaxe :**

```
variable = chaine.sup()
```

**Exemple 18.21 :**

```
y = "lepape@le-vatican.com";
document.write(y.sup()) ;
```

⇒ **Résultat**

### 18.3.21.Blink ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <blink>...</blink>. Ne fonctionne que sous Netscape

**Syntaxe :**

```
variable = chaine.blink()
```

**Exemple 18.22 :**

```
y = "lepape@le-vatican.com";
document.write(y.blink()) ;
```

⇒ **Résultat**

### 18.2.22.Small ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <small>...</small>.

**Syntaxe :**

```
variable = chaine.small()
```

**Exemple 18.23 :**

```
y = "lepape@le-vatican.com";
document.write(y.small()) ;
```

⇒ **Résultat**

### 18.3.23.Italic ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises <i>...</i>.

**Syntaxe :**

```
variable = chaine.italics()
```

**Exemple 18.24 :**

```
y = "lepape@le-vatican.com";
document.write(y.italics()) ;
```

⇒ **Résultat**

### 18.3.24.Link ()

Cette méthode renvoie le texte de l'objet en y ajoutant les balises `<a href=URL >...</a>`.

**Syntaxe :**

```
variable = chaine1.link(URL)
```

**Exemple 18.25 :**

```
y = "lepape@le-vatican.com";
document.write(y.link("http://www.google.fr")) ;
```

⇒ **Résultat**

### 18.3.25.Anchor ()

Cette méthode crée un ancre, et renvoie le texte de l'objet en y ajoutant les balises `<a name=ancre >...</a>`.

**Syntaxe :**

```
variable = chaine1.anchor(ancre)
```

**Exemple 18.26 :**

```
y = "lepape@le-vatican.com"
document.write(y.anchor("ancre")) ;
```

⇒ **Résultat**

## 18.4.Manipulations sur les chaînes

La manipulation des chaînes demande certaines connaissances syntaxiques. Ces dernières ont déjà été précisées dans le chapitre Structures de données, mais elles doivent être détaillées. Elles concernent l'affectation, la concaténation, et les caractères spéciaux.

### 18.4.1.Affectation

Les chaînes de caractères se présentent sous deux formes. Elles sont soit encadrées de quotes “ ”, soit encadrées de guillemets “ ”. On affecte les chaînes à leur variables comme toute variable.

**Exemple 18.27 :**

```
x = "Maman" ;  
y = 'Maman' ;
```

Dans l'exemple 12.8, les chaînes x et y sont équivalentes. Elles contiennent la même donnée.

#### 18.4.2.Concaténation

Il est possible d'ajouter des chaînes - de les mettre à la suite l'une de l'autre - grâce à l'opérateur « + ».

**Exemple 18.28 :**

```
x = "Maman" ;  
y = 'Papa' ;  
z = x + " " + y ;  
document.write (z) ;
```

⇒ **Résultat**

Dans les fonctions, comme la méthode write(), il est possible d'ajouter ces chaînes à l'aide de l'opérateur « + » ou « , ».

**Exemple 18.29 :**

```
x = "Maman" ;  
y = 'Papa' ;  
document.write (x + " " + y) ;  
document.write (x , " " , y) ;
```

⇒ **Résultat**

### 18.4.3.Caractères spéciaux

Certains caractères permettent de faire un effet dans l'affichage, d'autres doivent être précédés du caractère « \ ». Ils sont répertoriés dans le tableau suivant.

Caractère	Affichage
\b	touche de suppression
\f	formulaire plein
\n	retour à la ligne
\r	appui sur la touche <i>ENTREE</i>
\t	tabulation
\"	guillemets
\'	apostrophes
\\"	caractère antislash

Tab. 12.1 : Caractères spéciaux

Les autres caractères spéciaux - si on peut les appeler ainsi - sont les balises HTML. En effet, celles-ci peuvent être insérées dans les chaînes de caractère, et seront interprétées comme des balises par le navigateur lors de l'écriture avec la méthode `document.write()`.

**Exemple 18.30 :**

```
x = "Maman" ;
y = 'Papa' ;
document.write (x + "<br>" + y);
```

⇒ Résultat

### 18.5.Exercice

Dans cet exercice, vous devez créer un formulaire avec un bouton et une zone de texte. L'internaute entre une série de mots séparés par des espaces. Ensuite, on récupère ces mots, et on les sépare par des virgules. Enfin, on les affiche à l'écran avec `document.write()`.

⇒ Solution

## 19.L'OBJET DATE

### 19.1.Généralités

La date et l'heure sont regroupées en JS dans la classe Date. On crée alors une variable Date grâce au constructeur.

Syntaxe :

```
variable =new Date()
```

La date et l'heure en JS ne commencent qu'à partir du 1<sup>er</sup> janvier 1970, 0h 0min 0s. Toute référence à une date antérieure donnera un résultat aléatoire. Je ne mets pas d'exemple à chaque méthode, je ferai un exemple général et concret à la fin du chapitre.

### 19.2.Les méthodes

#### 19.2.1.Get

Une fois notre variable Date créée, il faut lui donner les informations sur la date et l'heure actuelle. Les fonctions suivantes remplissent la variable - qui est une chaîne - Date avec les données courantes. Elles utilisent toutes la même syntaxe, ce sont des méthodes objet.

Syntaxe :

```
variable1 =new Date()
variable2 = variable1.getInfo();
```

- `getYear()` : Retourne les 2 derniers chiffres de l'année. Il faudra donc rajouter le préfixe "20".
- `getFullYear()` : Retourne la date sur 4 chiffres.
- `getMonth()` : Retourne le mois sous la forme d'un entier compris entre 0 et 11.
- `getDate()` : Retourne le jour du mois sous forme d'un entier compris entre 1 et 31.
- `getDay()` : Retourne le jour de la semaine sous forme d'un entier compris entre 0 et 6.
- `getHours()` : Retourne l'heure sous forme d'un entier compris entre 0 et 23.
- `getMinutes()` : Retourne les minutes sous forme d'un entier compris entre 0 et 59.
- `getSeconds()` : Retourne les secondes sous forme d'un entier compris entre 0 et 59.

- `getMilliseconds()` : retourne les millisecondes de la date. A ne pas confondre avec `getTime()`.

### 19.2.2.Set

Il est aussi possible de remplir la variable Date avec nos propres données. Les fonctions suivantes remplissent la variable - qui est une chaîne - Date avec les données que vous voulez. Elles utilisent toujours la même syntaxe, ce sont des méthodes objet.

Syntaxe :

```
variable1 = new Date()  
variable2 = variable1.setInfo();
```

- `setYear()` : Assigne les 2 derniers chiffres de l'année, sous forme d'un entier supérieur à 1900.
- `setYear()` : Assigne l'année sur 4 chiffres.
- `setMonth()` : Assigne le mois sous la forme d'un entier compris entre 0 et 11.
- `setDate()` : Assigne le jour du mois sous forme d'un entier compris entre 1 et 31.
- `setDay()` : Assigne le jour de la semaine sous forme d'un entier compris entre 0 et 6.
- `setHours()` : Assigne l'heure sous forme d'un entier compris entre 0 et 23.
- `setMinutes()` : Assigne les minutes sous forme d'un entier compris entre 0 et 59.
- `setSeconds()` : Assigne les secondes sous forme d'un entier compris entre 0 et 59.
- `setMilliseconds()` : assigne les millisecondes de la date. A ne pas confondre avec `getTime()`.

### 19.2.3.L'heure

L'heure est très utile en JS, elle possède donc plusieurs méthode utiles. La syntaxe est toujours la même.

Syntaxe :

```
variable1 =new Date()  
variable2 = variable1.méthode();
```

- `getTime()` : Renvoie l'heure courante sous forme d'un entier représentant le nombre de millisecondes depuis le 1<sup>er</sup> janvier 1970 00h 00min 00s.
- `getTimezoneOffset()` : Renvoie la différence entre l'heure locale et l'heure GMT sous forme d'un entier en minutes.
- `setTime()` : Assigne l'heure courante sous forme d'un entier représentant le nombre de millisecondes depuis le 1<sup>er</sup> janvier 1970 00h 00min 00s.
- `toGMTString()` : Renvoie la valeur actuelle de la variable Date en heure GMT.
- `toLocaleString()` : Renvoie la valeur actuelle de l'heure de la variable Date. C'est plus rapide que de combiner `getHours()`, `getMinutes()`, et `getSeconds()`.

### 19.3.Exemple concret

Le but de cet exemple est d'afficher une horloge dans une ligne de texte. Pour cela, on utilise la méthode `window.setTimeout()`, qui rappelle la fonction d'affichage toute les secondes.

#### Exemple 19.1 :

```
<HTML>
<HEAD>
<script language="Javascript">
function GetTime () { //la fonction que l'on doit appeler
    var time = new Date (); // objet Date()
    var hours = time.getHours(); //on récupère les heures
    var min = time.getMinutes(); //on récupère les minutes
    var sec = time.getSeconds(); //on récupère les secondes
    if (hours < 10) hours = "0" + hours; //on rajoute un 0
    if (min < 10) min = "0" + min; //si le chiffre est
    if (sec < 10) sec = "0" + sec; //inférieur à 10
    // affichage de l'heure dans une zone de texte
    document.time.heure.value = hours + ":" + min + ":" + sec;
    window.setTimeout('GetTime()',1000); /* le timer rappelle la
fonction toutes les secondes */
}
</script>
</HEAD>
<BODY onLoad="GetTime();">
<form name="time">
Voici l'heure :
<!-- la zone de texte qui sert à l'affichage -->
<center><input type="text" name="heure" value="" size=6></center>
</form>
</BODY>
</HTML>
```

#### ⇒ Résultat

## 19.4.Exercice

### 19.4.1.Nombre de jours

Le but de l'exercice est de calculer le nombre de jours écoulés depuis l'an 2000.

⇒ **Solution**

### 19.4.2.Heure GMT

Cet exercice consiste à donner l'heure GMT de deux façons différentes.

⇒ **Solution**

## 20.L'OBJET IMAGE

### 20.1.Rappel HTML

Il semble utile de rappeler que la balise `<img>` possède de nombreux attributs dont certains qu'il est conseillé de déclarer en Javascript. Ci-dessous, la liste de ces attributs.

- `src` : URL, souvent relative, de l'image.
- `name` : nom de l'image dans la page, très important en JS.
- `width` : largeur de l'image affichée.
- `height` : hauteur de l'image affichée.
- `align` : alignement de l'image par rapport au texte.

Il est important de rajouter qu'une image peut servir d'ancre pour un lien hypertexte, c'est-à-dire être placée entre les balises `<a href>...</a>`. Cela permet certaines astuces d'affichage.

### 20.2.L'objet

La classe `Image` correspond à la balise HTML `<img>`. Son emploi est assez difficile et ce cours ne décrira pas en détail toutes ses facettes. Il permet de manipuler les images de façon dynamique, ce qui est impossible avec le HTML. On rappellera que les images sont stockées dans le tableau `images[]` de l'objet `document`.

Grâce à un objet `Image`, on peut précharger une image et la stocker en cache, contrairement au HTML. L'image ne sera cependant pas affichée. Elle le sera à l'aide de la balise `<img>`.

Un objet `Image` est appelé selon la syntaxe objet habituelle, avec le constructeur `Image()`. L'objet possède alors les propriétés de la balise HTML `<img>`, dont la liste figure dans 20.3.Les propriétés.

### 20.3.Les propriétés

#### 20.3.1.Syntaxe

Un objet `Image` possède plusieurs propriétés, que l'on peut assimiler aux attributs de la balise `<img>`.

Syntaxe :

```
variable = new Image();  
variable.propriété = x ;  
var2 = variable.propriété ;
```

### 20.3.2.Src

Il s'agit de l'URL absolue ou relative de l'image. Elle peut être modifiée. Cela permet de charger en cache l'image lors du chargement de la page.

#### Exemple 20.1 :

```

```

#### ⇒ Résultat

### 20.3.3.Name

C'est le nom défini par l'attribut name="..." de la balise <img>. A ne pas confondre avec l'ID. Permet de trouver l'image dans le tableau document.images[].

#### Exemple 20.2 :

```

```

#### ⇒ Résultat

### 20.3.4.Id

C'est l'ID défini par l'attribut id="..." de la balise <img>. A ne pas confondre avec le nom. Permet de retrouver l'image grâce à la méthode document.getElementById().

#### Exemple 20.3 :

```

```

#### ⇒ Résultat

### 20.3.5.Width

Il s'agit de la largeur de l'image. Elle contient la valeur de l'attribut width de la balise <img>. Si cet attribut n'est pas précisé, elle contient la largeur réelle de l'image. Ne peut être modifiée.

**Exemple 20.4 :**

```

```

⇒ **Résultat**

### 20.3.6.Height

Il s'agit de la hauteur de l'image. Elle contient la valeur de l'attribut height de la balise <img>. Si cet attribut n'est pas précisé, elle contient la hauteur réelle de l'image. Ne peut être modifiée.

**Exemple 20.5 :**

```

```

⇒ **Résultat**

### 20.3.7.Complete

C'est un booléen qui indique si le chargement de l'image est terminé. Renvoie true si le chargement est achevé et false dans le cas contraire.

**Exemple 20.6 :**

```

```

⇒ **Résultat**

### 20.3.8.Alt

Elle contient le texte qui affiché avant la chargement de l'image et en info-bulle lorsqu'elle est survolée. Contient la valeur de l'attribut alt="..." de la balise <img>. Ne peut être modifiée.

**Exemple 20.7 :**

```

```

⇒ **Résultat**

### 20.3.9.FileSize

Elle contient la taille en octets de l'image. N'est accessible que lorsque le chargement est terminé, d'où l'utilité de la propriété complete.

**Exemple 20.8 :**

```

```

⇒ **Résultat**

## 20.4.Afficher une image

Il suffit de prendre l'élément `<img>` voulu dans la page et de changer sa propriété. Cet élément est assimilé à un objet `Image`, faisant partie de l'objet `document`.

**Exemple 20.9 :**

```
document.image1.src = 'img2.jpg' ;
```

Dans ce cas précis, l'image change et `img2.jpg` est affiché dans le champ `image1`. L'intérêt de créer un objet `Image` reste donc discutable, puisqu'on ne l'utilise pas...

L'objet `image` permet de stocker l'image en cache au moment du chargement de la page. Il ne reste plus qu'à trouver le moyen de l'afficher... De plus, on peut créer un tableau d'objets `Image`, ce qui nous facilitera les manipulations ensuite.

### Exemple 20.10 :

```
<html>
<head>
<script language="Javascript">
var tab = new Array (4) ; //tableau
for (i = 0; i < 4; i++) {//remplissage d'images
    tab[i] = new Image();
    tab[i].src = i + ".gif";
}
function f() { //fonction d'affichage
    for (i = 0; i < 4; i++) {
        document.images[i].src = tab[i].src ;
    }
}
</script>
</head>
<body>
  


</body>
</html>
```

#### ⇒ Résultat

L'exemple 14.3 permet de charger 5 images et des les afficher dans 5 balises `<img>` différentes grâce au tableau `images[]`.

### 20.5.Exemple concret : cliquer pour changer d'image

Ici, le but est de créer une page avec une image et un bouton. Lorsque l'on clique sur le bouton, l'image change. On a au total 4 images différentes. On utilise un tableau d'objets `Image`, une balise `<img>`, une balise `<input>`, et une fonction. Lorsque les 4 images ont défilé, on recommence à la première. Pour cela, on utilise un compteur.

### Exemple 20.11 :

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
//création d'un tableau de 4 éléments
var tab_images = new Array(4);
for (i = 0; i < tab_images.length; i++) { // pour chaque élément
    tab_images[i] = new Image(); /*on crée un objet Image
    et on lui donne un fichier à charger */
    tab_images[i].src = i + ".gif";
}
var nb = 0; //on initialise un compteur

function changer() { // la fonction qui sera appelée.
    nb++; //incrémenter le compteur

    if (nb == tab_images.length) { //si on est 4
        nb = 0; //on remet à 0 le compteur
    }
    //affichage de l'image
    window.document.image.src = tab_images[nb].src; }
</script>
</HEAD>
<!-- la fonction est appelée au chargement de la page -->
<BODY onload="changer();">
<img src="" name="image"><br><!-- la balise <img> -->
<input type="button" name="bouton" value="cliquez"
onClick="changer();"> <!-- le bouton avec l'évènement -->
</BODY>
</HTML>
```

#### ⇒ Résultat

## 20.6.Exercice

Ici, le but est aussi de créer une page avec une image. Lorsque passe la souris sur l'image, l'image change. Quand on « ressort » de l'image, l'image de base revient. 4 images différentes s'afficheront en un cycle.

#### ⇒ Solution

# 21.LA PROGRAMMATION MULTI-CADRES<sup>15</sup>

## 21.1.Rappel HTML

Les frames sont un élément du langage HTML. Il s'agit du partage de la fenêtre en plusieurs cadres où l'on pourra afficher différentes pages HTML. Les frames se déclarent dans le fichier principal avec la balise `<frameset>` suivie de plusieurs balises `<frame>`. Il n'y a alors pas de balise `<body>` dans la page principale. Pour plus de précisions, consulter un cours HTML.

## 21.2.Généralités

Les frames ne sont pas très utilisées en JS, mais il est important de savoir quelques détails en cas d'utilisation de frames au niveau HTML. Il est important, lorsque le JS est utilisé, de préciser le nom de la frame dans la balise `<frame>`.

**Exemple 21.1 :**

```
<frame src="fichier.htm" name="cadre1">
```

Ce nom ne paraît pas très utile en HTML, mais il prend toute son importance en JS. Il sera utilisé dans deux cas : la cible des liens ainsi que pour accéder à un élément d'une autre frame de la page.

## 21.3.Liens hypertexte

Lorsque de l'utilisation des frames, il peut s'avérer utile d'afficher une page dans une frame différente de celle où se trouve le lien. C'est là où intervient l'attribut target de la balise `<a href...>`. Il permet d'indiquer la frame où l'on veut que la page appelée s'affiche.

**Syntaxe :**

```
<a href="lien" target="frame">texte</a>
```

La référence indiquée dans l'attribut target peut être de deux types. Dans un premier cas, il s'agit du nom de la frame déclarée dans la page principale, cas où il faut savoir ce nom. Sinon, si on fait référence au cadre parent, on indiquera « `_parent` », et si on fait référence à la fenêtre complète, on indiquera « `_top` ».

<sup>15</sup> Par programmation multi-cadres, j'entend la programmation avec frames.

### Exemple 21.2 :

```
<a href="lien1.htm" target="frame1">cliquez ici</a>
<a href="lien2.htm" target=" parent">ou ici</a>
```

## 21.4.Accès aux éléments des frames

Lorsque de l'utilisation des frames, il est souvent nécessaire en JS d'accéder aux éléments des autres frames. L'objet `window` - que nous avons vu précédemment - nous fournit de quoi le faire. Il contient l'objet `parent`, qui possède les mêmes propriétés que lui.

### 21.4.1.L'objet frames[]

La première façon d'accéder aux éléments d'une autre frame se fait bien sûr par le tableau `frames[]` - propriété de `parent` - dont le numéro des frames est attribué dans l'ordre de déclaration de celles-ci. On accède ensuite à chaque élément de la frame ainsi qu'à ses propriétés et méthodes.

#### Syntaxe :

```
parent.frames[i].objet.propriété
parent.frames[i].objet.méthode()
```

### Exemple 21.3 :

```
Parent.frames[1].form1.action = "get";
```

### Exemple 21.4 :

```
parent.frames[1].form1.button1.value = "Click";
```

Dans l'exemple 20.4, on accède à la valeur du bouton appelé « `Button1` » du formulaire nommé « `form1` ».

### 21.4.2.Le nom de la frame

L'autre façon d'accéder aux éléments d'une frame est d'utiliser son nom. Ce sera alors un objet de `parent`. Il faut donc prendre soin à donner un nom aux frames lors de leur déclaration. On accède ensuite à chaque élément de la frame ainsi qu'à ses propriétés et méthodes.

### Syntaxe :

```
parent.nom.objet.propriété  
parent.nom.objet.méthode()
```

### Exemple 21.5 :

```
parent.frame1.form1.action = "get";
```

### Exemple 21.6 :

```
parent.frame1.form1.button1.value = "Click";
```

## 21.5.Exemple concret

L'exemple 21.7 permet de faire un compteur de secondes impair sur une frame et pair sur l'autre.

### Exemple 21.7 :

```
Exemple 21.7.1.htm
```

```
<HTML>  
<HEAD>  
<TITLE>New Document</TITLE>  
<script language="Javascript">  
</script>  
</HEAD>  
<body>  
<form name="form1">  
<center><input type="text" name="out1"  
value="00:00:00"></center>  
</form>  
</body>  
</HTML>
```

```
Exemple 21.7.2.htm
```

```
<HTML>  
<HEAD>  
<TITLE>New Document</TITLE>  
<script language="Javascript">  
</script>  
</HEAD>  
<body>  
<form name="form2">  
<center><input type="text" name="out2"  
value="00:00:00"></center>  
</form>  
</body>  
</HTML>
```

Exemple 21.7.htm

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
var sec = 0;
var min = 0;
var hrs = 0;
var test = 1;
function f() {
    sec++;
    if (sec == 60) {
        sec = 0;
        min++;
    }
    if (min == 60) {
        min = 0;
        hrs++;
    }
    h = hrs;
    m = min;
    s = sec;
    if (h < 10) h = "0" + h;
    if (m < 10) m = "0" + m;
    if (s < 10) s = "0" + s;
    if (test == 1) {
        parent.frame1.form1.out1.value = h + ":" + m + ":" +
+ s;
        test = 2;
    }
    else if (test == 2) {
        parent.frame2.form2.out2.value = h + ":" + m + ":" +
+ s;
        test = 1;
    }
    window.setTimeout('f()',1000);
}
</script>
</HEAD>
<frameset cols="50%,50%" onLoad="f()">
    <frame src="exemple 21.7.1.htm" name="frame1">
    <frame src="exemple 21.7.2.htm" name="frame2">
</frameset>
</HTML>
```

⇒ Résultat

## 21.6.Exercice

Votre exercice comporte 2 frames, avec chacune un bouton. Chaque clic sur un bouton incrémente la valeur de l'autre bouton.

⇒ Solution

## 22.LES COOKIES

### 22.1.Présentation

En JS, il est possible de travailler avec les cookies. Etant donné l'absence de gestion d'écriture/lecture de fichier, les cookies sont le seul moyen de stocker des informations permanentes sur la machine de l'utilisateur. Ces dernières pourront être récupérées plus tard et réutilisées. Cela permet de compter le nombre de visites de l'internaute, de créer une liste de préférence de navigation sur le site, de conserver le login et le password afin de se connecter directement à un compte... Les applications des cookies sont nombreuses.

Le seul risque de cette méthode est la suppression ou le refus des cookies par l'utilisateur<sup>16</sup>.

Le cookie en lui-même se présente sous la forme d'une chaîne de caractère qui contient des informations concaténées : l'information que l'on veut conserver, la date d'expiration, l'auteur - path -, le nom de domaine, la sécurisation.

### 22.2.Créer un cookie

On crée un cookie avec l'objet `cookie` de l'objet `document`. Il s'agit d'une chaîne de caractère dans laquelle on indique les informations que l'on veut.

Syntaxe :

```
document.cookie = "informations"
```

Dans les informations, il y a tout d'abord la chaîne que l'on souhaite conserver. Ensuite, on met la date d'expiration, le path, le nom de domaine, et - si besoin - la fait que le cookie soit protégé. Seuls les deux premiers champs sont obligatoires.

Syntaxe :

```
document.cookie = "variable = contenu ; expires = date ;  
path = nom ; domain = domaine ; secure = true/false" ;
```

Il semble utile de décrire chaque champ indiqué ci-dessus. Si la chaîne est mal écrite, le cookie ne sera pas utilisable, et deviendra par conséquent inutile.

---

<sup>16</sup> Cette option est disponible dans les menus de votre navigateur. Pour Internet explorer, ce la se situe dans le menu Outils | Options Internet , onglet Sécurité, dans la rubrique Personnaliser le niveau.

- Le champ information : il s'agit de ce que vous voulez stocker dans le cookie. Il faut définir un nom à la variable et lui affecter une valeur, un contenu. Comme on peut le voir ci-dessus, les champs sont séparés par des points-virgule, il ne faut donc pas insérer des « ; » dans le contenu.
- expires : contient la date d'expiration, à laquelle le cookie sera détruit. La valeur est en secondes. Le plus simple consiste à utiliser un objet Date.
- path : le chemin de la page qui a créé le cookie.
- domain : le domaine de la page qui a créé le cookie.
- secure : booléen qui indique si le cookie doit utiliser le protocole HTTP (false) ou le protocole sécurisé HTTPS (true).

### Exemple 22.1 :

```
document.cookie = "visite=1; expires=31/12/2004;  
path=le-vatican.com/index.php; secure=true";
```

Dans l'exemple 9.1, il s'agit d'un cookie créé par lepape, qui expire le 31/12/2004, auquel on peut accéder uniquement par un échange sécurisé, et qui compte le nombre de visites.

## 22.3.Récupérer un cookie

La récupération d'un cookie est plus complexe, mais elle est essentielle, car sinon le cookie est inutile. Le but est de trouver le type d'information que l'on cherche, et ensuite de lire la valeur. La maîtrise de la classe String est obligatoire pour ce genre d'exercice. Cette partie explique pas à pas la manière pour récupérer un cookie. On commence par mettre tous les cookies dans une variable.

### Syntaxe :

```
variable = document.cookie ;
```

Ensuite, on cherche le nom de la « variable » dans la chaîne du cookie. Ensuite, on récupère ce qui est situé entre le signe « = » et le « ; ». Cela semble simple, mais il faut utiliser les méthodes des objets String et utiliser des instructions logiques. Ci-dessous, l'exemple 9.2 montre en détail comment récupérer un cookie.

### Exemple 22.2 :

```
//création du cookie
document.cookie = "visite=1;expires=31/12/2004";
//variable à trouver
var variable_a_trouver = "visite" ;
//on rajoute le signe =
variable_a_trouver = variable_a_trouver + "=" ;
//on récupère le cookie
var chaine = document.cookie ; var longueur = variable_a_trouver.length ;
//on récupère la longueur à trouver
var resultat ;
//si le cookie n'est pas vide, on commence à chercher.
if (chaine.length > 0) {
    //on cherche la position du début de la variable
    var debut = chaine.indexOf(chaine_a_trouver,0);
    //si on a trouvé cette position, on continue
    if (debut >=0) {
        var fin ;
        /* on rajoute la longueur de la variable, pour arriver au début
        du contenu */
        debut = debut + longueur ;
        //on cherche la fin du contenu, c'est-à-dire le premier « ; »
        fin = chaine.indexOf(";", debut) ;
        //si on trouve la position du point-virgule
        if (fin>=0) {
            //on récupère la chaîne située entre le "=" et le ";"
            resultat = unescape(chaine.substring(debut,fin)) ;
        }
        // si il n'y a pas de ";", c'est que c'est la fin de la chaîne
        else {
            /* on récupère la chaîne située après le "=" la fin de la
            chaîne */
            resultat = unescape(chaine.substring(debut,chaine.length)) ;
        }
    }
}
```

Cet exemple n'est qu'une manière parmi d'autres. On peut imaginer quantités de façons, mais celle-ci est sûrement la plus simple. Cela peut d'ailleurs constituer une exercice.

### 22.4.Modifier un cookie

Modifier un cookie est aussi simple que de le créer. En réalité, il suffit de le recréer avec un contenu différent et une date actualisée.

## 22.5. Supprimer un cookie

Pour supprimer un cookie, il faut tout simplement le recréer, avec la même valeur pour éviter de se compliquer les choses, et lui donner une date d'expiration passée. Sinon, il suffit d'attendre sa date d'expiration, ce qui n'est pas toujours satisfaisant.

## 23.LA PROGRAMMATION OBJET

### 23.1.Présentation

Bien que le Javascript ne soit pas un langage orienté objet, il donne la possibilité de créer ses propres objets. Ce chapitre s'adresse à des programmeurs possédant un niveau correct<sup>17</sup> car la programmation objet est un point assez difficile.

Les programmeurs C++ seront étonnés de voir que les classes JS sont très simplifiées comparées à celles du C++. Elles ne demandent pas une définition complète de la classe, mais il suffit de déclarer la fonction constructeur.

### 23.2.Déclaration d'une classe

Contrairement au C++ - qui demande une déclaration détaillée - déclarer une classe en JS se fait simplement en déclarant la fonction constructeur de classe. On déclare à l'intérieur les propriétés à l'aide de l'objet this. On retrouve l'objet this découvert précédemment. Comme on l'a vu, il désigne l'objet en cours.

**Syntaxe :**

```
function nom_classe ( paramètres ) {  
    this.propriété = paramètre1 ;  
}
```

On déclare l'objet avec la syntaxe habituelle et le mot-clé new. Le nom de la classe est en général le type d'objet avec une majuscule.

**Exemple 23.1 :**

```
function Eleve (Age,Sexe,Ville) {  
    this.age = Age ;  
    this.sex = Sexe ;  
    this.ville = Ville ;  
}  
  
var Laurent = new Eleve(17,'M','Grenoble') ;
```

<sup>17</sup> Il convient de préciser que ce terme n'est en aucun cas péjoratif. Il signifie simplement qu'il est nécessaire de bien maîtriser tous les chapitres précédent avant de se lancer dans la programmation objet.

### 23.3.Utilisation de méthodes

Il est bien entendu possible de déclarer et utiliser des méthodes. Pour cela, il faut déclarer une fonction indépendante de la classe. Ensuite, on déclare la fonction à l'intérieur du constructeur, en l'affectant à une méthode. Il faut faire attention lors de cette déclaration, car **il ne faut pas mettre les parenthèses des fonctions<sup>18</sup> !**

**Syntaxe :**

```
function nom_classe () {  
    this.méthode = fonction ;  
}
```

**Exemple 23.2 :**

```
function Eleve (Nom, Age) {  
    this.age = Age ;  
    this.nom = Nom ;  
    this.affich = affich_infos;  
}  
function affich_infos () {  
    document.write (this.nom + " a " + this.age + "  
ans.");  
}  
var Laurent = new Eleve(17, 'Laurent') ;  
Laurent.affich() ;
```

⇒ **Résultat**

Il est possible de simplifier l'écriture lorsqu'il y a beaucoup de propriétés, grâce à l'utilisation de `with (objet) {}`. Cela ne fonctionne qu'à l'intérieur des méthodes, et non dans le constructeur.

<sup>18</sup> Même si cela semble contradictoire, ceci est important. On ne peut faire des égalités de fonctions.

### Exemple 23.3 :

```
function Eleve (Nom, Age) {  
    age = Age ;  
    nom = Nom ;  
    affich = affich_infos;  
}  
function affich_infos () {  
    with (this) {  
        document.write (nom + " a " + age + " ans.");  
    }  
}  
var Laurent = new Eleve(17, 'Laurent') ;  
Laurent.affich() ;
```

#### ⇒ Résultat

## 23.4. Exercice

Dans cet exercice, le but est de demander 3 informations à l'utilisateur (nom, age, ville) à l'aide d'une boîte de dialogue. On regroupe ces informations à l'aide d'un objet - d'une classe créée auparavant - puis on utilise une méthode pour l'affichage dans une zone de texte.

#### ⇒ Solution

## 24.LES EXPRESSION REGULIERES

### 24.1.Présentation

Ce chapitre s'adresse à des programmeurs avertis, qui voudront bâtir un site dynamique d'un aspect assez évolué. Les expressions régulières sont assez compliquées à comprendre, et leur utilisation demande une connaissance sans failles de l'objet String.

### 24.2.Définition

Les expressions régulières existent dans la plupart des langages de programmation, mais sont peu connues du fait de leur complexité. Elles permettent de réaliser des traitements sur les chaînes de caractères. Ces traitements sont de l'ordre de la recherche de caractères, de l'extraction de sous-chaînes... beaucoup d'autres traitements existent qu'il revient à vous d'imaginer et de créer. En réalité, une expression régulière possède un motif, qui est une suite de caractères ordonnés selon un ordre, un nombre d'apparitions, une non-apparition...

Une expression régulière est avant tout un objet RegExp. Comme tout objet, il se déclare ainsi :

Syntaxe :

```
var reg = RegExp (pattern, option);
```

Comme vous l'avez sûrement remarqué, pour la première fois, il faut fournir des paramètres au constructeur. Ces paramètres concernent l'expression régulière que vous utilisez. Le pattern est le motif de l'expression. L'option est une chaîne de caractères, qui - comme son nom l'indique - contient les options de l'expression régulière. Nous allons voir leur constitution ci-après.

Il existe aussi une autre façon de déclarer une expression régulière. Elle est moins utilisée et surtout moins claire.

Syntaxe :

```
var reg = /pattern	option/;
```

Les deux notations sont absolument équivalentes mais la première est plus explicite, donc conseillée.

## 24.3.Paramètres d'une expression régulière

Les exemples sont regroupés dans le 24.3.3.Exemples car ils se doivent de contenir les deux paramètres.

### 24.3.1.L'option

La chaîne option peut prendre 4 valeurs. La première est la chaîne vide "", qui signifie l'absence d'option. Les 3 autres valeurs sont détaillées ci-dessous.

- "g" : la recherche est globale - sur toute la chaîne -.
- "i" : ne pas tenir compte de casse - majuscules/minuscules- .
- "gi" : les deux options réunies

### 24.3.2.Le pattern

Cette partie est la plus délicate. Le pattern est assez complexe et sa compréhension peut être difficile. Il faut savoir que le pattern correspond aux caractères que vous cherchez - pour une raison ou une autre - dans une chaîne.

La chaîne pattern est extensible à l'infini. Le choix de ce qu'elle contient vous appartient. Il s'agit du motif de la chaîne, c'est-à-dire des caractères que vous choisissez d'inclure ou d'exclure de votre recherche. Cette chaîne pattern contient des caractères spéciaux concaténés. Ces caractères spéciaux concernent le motif lui-même, le caractère à rechercher, le nombre d'occurrences, le groupe de caractères cherché... Leur liste se trouve dans le Tab.24.1.

Motif	Signification
^	Début du pattern - de la chaîne.
\$	Fin du pattern. Interdit tout caractère après.
.	N'importe quel caractère.
a   b	Caractère a OU b.
*	Caractère précédent présent 0 à x fois.
+	Caractère précédent présent 1 à x fois.
?	Caractère précédent présent 0 à 1 fois.
{x}	Caractère précédent présent x fois.
{x,y}	Caractère précédent présent au moins x fois.
{x,y}	Caractère précédent présent au entre x et y fois.
[abc]	Groupe de caractères : n'importe lequel contenu entre les crochets.
[a-z]	N'importe quel caractère alphabétique.
[^a-z]	Aucun caractères alphabétique.
\\	Caractère « \ ».
\d	Tous les chiffres - équivalent de [0-9] - .
\D	Aucun chiffre - équivalent de [^0-9] - .
\b	Limites des mots (espace, tab, ...).
\s	Tous les caractères d'espacements - équivalent de [\v\r\n\t\f] - .
\S	Aucun caractère d'espacements - équivalent de [^\v\r\n\t\f] - .
\w	Tous les caractères alphanumériques dont « _ » - équivalent de [A-Za-z0-9_] - .
\W	Aucun caractère alphanumérique - équivalent de [^A-Za-z0-9_] - .

Tab.24.1 : Motifs des expressions régulières.

Les différents motifs ne sont pas séparés. On les mets les uns à la suite des autres, sans espacements.

Il est à préciser que l'on peut très bien mettre une variable ou un mot entre guillemets - sans mise en forme avec ^ \$ - comme argument pattern.

### 24.3.3.Exemples

Voici différents exemples de déclaration d'objet RegExp. Ces exemples ne sont pas très complexes, ils donnent simplement une idée de la construction des motifs. Le pattern est plus détaillé que les options qui ne sont pas très importantes. Chaque exemple est commenté, pour bien comprendre le but du motif.

#### Exemple 24.1 :

```
var exp = new RegExp ("^ [A-Za-z0-9] { 4 , } $ ", " i " ) ;
```

L'exemple 24.1 présente la recherche d'une chaîne de au moins 4 caractères alphanumériques. `[A-Za-z0-9]` désigne les caractères alphanumériques, et `{ 4 , }` désigne 4 fois ou plus. Les caractères sont au choix majuscules ou minuscules.

#### Exemple 24.2 :

```
var exp = / ^ [A-Za-z0-9] { 4 , } $ / i ;
```

L'exemple 24.2 est équivalent à l'exemple 24.1, mais il utilise l'autre notation.

#### Exemple 24.3 :

```
var exp = new RegExp (" ^ [A-Za-z0-9] + [A-Za-z0-9] \ . \ - _ ] * @ [A-  
Za-z0-9] \ - _ ] + \ . [A-Za-z0-9] \ . \ - _ ] { 1 , } $ " , "" );
```

L'exemple 24.2 présente la vérification d'une adresse e-mail. Elle doit commencer par au moins un caractère alphanumérique : `[A-Za-z0-9] +`. Ensuite elle peut comporter autant de caractères alphanumériques que l'on veut, plus le point, le tiret et l'underscore : `[A-Za-z0-9] \ . \ - _ ] *`. Tout cela doit être suivi d'un `@` : `@`. Ensuite, il peut y avoir n'importe quel nombre de caractères alphanumériques, plus le point, le tiret et l'underscore : `[A-Za-z0-9] \ - _ ] +`. Cela doit être suivi d'un point : `\ .`. Ce dernier doit être suivi d'au moins deux caractères alphanumériques dont le point et le tiret : `[A-Za-z0-9] \ . \ - _ ] { 1 , }` . Il n'y a aucune option.

Ces exemples ne traitent que de la création de l'objet RegExp. Ce dernier est bien entendu inutile si on ne l'utilise pas ensuite. C'est le sujet de la partie suivante.

## 24.4.Utilisation d'une expression régulière

### 24.4.1.Les méthode de l'objet RegExp

Il existe trois méthodes de l'objet RegExp : `test()`, `exec()` et `compile()`. Elles s'utilisent selon la syntaxe objet habituelle. La première prend en paramètre la chaîne à tester selon le motif de l'expression régulière. Elle renvoie un booléen qui indique si la chaîne correspond au motif ou non. La deuxième méthode prend aussi la chaîne à tester en paramètre. Elle renvoie un tableau des occurrences du motif à tester. La dernière permet de modifier le motif de l'expression régulière, sans en créer un nouveau.

#### Exemple 24.3 :

```
adresse = prompt ("Votre mail ?","mail@fai.com") ;
var exp = new RegExp ("^ [A-Za-z0-9]+[A-Za-z0-9\\._]*@[A-
Za-z0-9\\._]+\\. [A-Za-z0-9\\._]{1,} $","");
document.write ( exp.test(adresse) );
```

L'exemple 24.3 propose de tester la correction de votre mail. On retrouve le motif de l'exemple 24.2. La méthode `test()` est appliquée à l'adresse e-mail. Le reste vous est - ou devrait vous être - familier.

### 24.4.2.Les méthode de l'objet String

Les méthodes de l'objet String vont aussi aider à la manipulation des expressions régulières. Nous introduirons ici 3 nouvelles méthodes de l'objet String<sup>19</sup>. A ces trois méthodes, nous rajouterons la plupart des méthodes String.

- `search()` : trouver les occurrences répondant aux critères du motif.
- `replace ()` : trouver remplacer les occurrences répondant aux critères du motif.
- `match()` : trouver les occurrences répondant aux critères du motif.

Ces trois méthodes prennent en paramètre un objet RegExp. Voici la syntaxe correspondante ci-après.

#### Syntaxe :

```
var reg = RegExp (pattern, option);
chaîne.méthode(reg);
```

<sup>19</sup> Elles n'ont pas été introduites précédemment du fait de leur inutilité en dehors des expressions régulières.

Il est à noter que la méthode `split()` possède une syntaxe similaire. De plus, il est possible d'indiquer simplement le motif, avec la notation peu utilisée des slash, comme paramètre.

**Syntaxe :**

`chaîne.méthode(/pattern option);`

Les méthodes `split()` et `match()` renvoient chacune un tableau de toutes les occurrences trouvées.

**Exemple 24.4 :**

```
var nom = prompt('Votre nom?\nMettez les accents...', 'nom') ;
nom = nom.replace (/[\éèêë]/g, "e") ;
nom = nom.replace (/[\àäâ]/g, "a") ;
nom = nom.replace (/[\üûù]/g, "u") ;
nom = nom.replace (/[\òöô]/g, "o") ;
nom = nom.replace (/[\ìïî]/g, "i") ;
alert(nom) ;
```

⇒ **Résultat**

L'exemple 24.4 propose de supprimer les accents d'une chaîne. On remplace tout simplement les lettres accentuées par la lettre sans accent.

## 24.5.Exemple concret

Ci-dessous est proposé un exemple dans lequel vous entrez une suite de mots séparés par des caractères de ponctuation. En cliquant sur un bouton, la liste de tous les noms est affichée. De plus, deux méthodes sont proposées. Cet exemple vous montre aussi comment intégrer le traitement au code avec une fonction.

### Exemple 24.5 :

```
<HTML>
<HEAD>
<TITLE>New Document</TITLE>
<script language="Javascript">
function TraitBySplit()  {
    var chaine = document.form1.input.value; //récupération de la chaîne
    var exp = new RegExp ('[.,;:/!? ]',"g"); //motif avec la ponctuation
    tab = chaine.split(exp); //séparation de la chaîne
    var result = "Voici les noms :"; //affichage
    for (i = 0 ; i < tab.length ; i++)  {
        result = result + "\n" + tab[i] ;
    }
    document.form1.output.value = result;
}
function TraitByMatch()  {
    var chaine = document.form1.input.value;//récupération de la chaîne
    //motif avec les lettres
    var exp = new RegExp ('[A-Za-zïîòöüùàääéèëé]+',"g");
    tab = chaine.match(exp); //séparation de la chaîne
    var result = "Voici les noms :"; //affichage
    for (i = 0 ; i < tab.length ; i++)  {
        result = result + "\n" + tab[i] ;
    }
    document.form1.output.value = result;
}
</script>
</HEAD>
<BODY>
<center>
<form name="form1">
<textarea name="input" rows=5 cols=50>Entrez une suite de noms séparés
indifféremment par les signes de ponctuation .,:/!? et
espace</textarea><br/>
<input type="button" name="match" value="Avec match()" 
onClick="TraitByMatch();">&ampnbsp
<input type="button" name="split" value="Avec split()" 
onClick="TraitBySplit();"><br/>
<textarea name="output" rows=5 cols=50>Résultat</textarea><br/>
</form>
</center>
</BODY>
</HTML>
```

#### ⇒ Résultat

## 24.6.Exercice

Faites un formulaire avec 3 lignes de texte et un bouton. Dans les lignes de texte, l'internaute entre son adresse, code postal et ville. En cliquant sur le bouton, on teste leur validité.

#### ⇒ Solution

# 25.FONCTIONS ET PROPRIÉTÉS

## 25.1.Présentation

Ce chapitre présente - un peu tardivement - les fonctions intégrées au langage, et ne dépendant d'aucun objet. On verra aussi les méthodes et propriétés que l'on peut associer à tous les objets. Ce chapitre arrive - j'en conviens - un peu tard, car j'avais omis ce sujet dans mon plan. Je le rajoute donc ici.

## 25.2.Les fonctions du langage

### 25.2.1.Escape()

Cette fonction encode les caractères spéciaux d'une chaîne, selon le code %xx, et retourne cette chaîne encodée.

**Syntaxe :**

```
chaine1 = escape (chaine2)
```

**Exemple 25.1 :**

```
var chaine = "Voici des caractères spéciaux !" ;  
document.write( escape(chaine) ) ;
```

⇒ **Résultat**

### 25.2.2.Unescape()

Cette fonction décide les caractères spéciaux codé par escape(), et retourne cette chaîne encodée.

**Syntaxe :**

```
chaine1 = escape (chaine2)
```

**Exemple 25.2 :**

```
var chaine = "Voici des caractères spéciaux !" ;  
var chaine2 = escape(chaine) ;  
document.write( unescape(chaine2) ) ;
```

⇒ **Résultat**

### 25.2.3.ParseFloat ()

Cette fonction convertit une chaîne passée en paramètre en nombre décimal. Renvoie NaN si la conversion est impossible.

**Syntaxe :**

```
decimal = parseFloat (chaine)
```

**Exemple 25.3 :**

```
var chaine = "Voici des caractères spéciaux !" ;
var chaine2 = "35.7" ;
document.write( parseFloat(chaine) + "<br/>" ) ;
document.write( parseFloat(chaine2) ) ;
```

⇒ **Résultat**

### 25.2.4.ParseInt ()

Cette fonction convertit une chaîne passée en paramètre en nombre entier. Renvoie NaN si la conversion est impossible. Le paramètre facultatif base permet de faire une conversion en une autre base que décimale.

**Syntaxe :**

```
decimal = parseInt (chaine, base)
```

**Exemple 25.4 :**

```
var chaine = "35.7" ;
document.write( parseInt(chaine) + "<br/>" ) ;
document.write( parseInt(chaine, 8) ) ;
```

⇒ **Résultat**

### 25.2.5.IsFinite ()

Cette fonction renvoie true si le nombre est fini, sinon, elle renvoie false.

**Syntaxe :**

```
booleen = isFinite (nombre)
```

### Exemple 25.5 :

```
var chaine = "35.7" ;
var chaine2 = "Math";
document.write( isFinite(chaine) + "<br/>" ) ;
document.write( isFinite(chaine2) ) ;
```

#### ⇒ Résultat

### 25.2.6. isNaN ()

Cette fonction renvoie true si la chaîne n'est pas un nombre, sinon, elle renvoie false.

#### Syntaxe :

```
booleen = isNaN (chaine)
```

#### Exemple 25.6 :

```
var chaine = "35.7" ;
var chaine2 = "Math";
document.write( isNaN(chaine) + "<br/>" ) ;
document.write( isNaN(chaine2) ) ;
```

#### ⇒ Résultat

## 25.3. Méthodes et propriétés des objets

Les objets de Javascript ou que vous avez créés possèdent deux propriétés et deux méthodes en commun. Elles permettent de manipuler ces objets et de connaître certaines de leur caractéristiques.

### 25.3.1. Prototype

Cette propriété permet de rajouter une propriété ou une méthode à un objet déjà existant, que vous avez créé ou qui existe dans JS. On l'utilise selon la syntaxe suivante :

#### Syntaxe :

```
classe.prototype.nom = valeur ;
```

L'exemple 25.7 rajoute une propriété et une méthode à l'objet Array.

### Exemple 25.7 :

```
var tab = new Array(5);
for (i = 0; i < 5; i++) tab[i] = i+1;

Array.prototype.comment = null; // on rajoute un commentaire
tab.comment = "Tableau de 5 chiffres";

function FirstElement () { /* fonction rentrant le premier
élément */
    return this[0];
}
Array.prototype.firstElement = FirstElement;
document.write (tab.comment + " : premier élément : " +
tab.firstElement());
```

#### ⇒ Résultat

### 25.3.2.Constructor

Cette propriété renvoie le constructeur de l'objet.

#### Syntaxe :

```
variable = objet.constructor ;
```

### Exemple 25.8 :

```
var tab = new Array(5);
var s = "string";
var d = new Date ()
var e = new RegExp();
document.write ("Constructeur Array : " + tab.constructor + "<br/>");
document.write ("Constructeur String : " + s.constructor + "<br/>");
document.write ("Constructeur Date : " + d.constructor + "<br/>");
document.write ("Constructeur RegExp : " + e.constructor + "<br/>");
```

#### ⇒ Résultat

### 25.3.3.ValueOf()

Cette méthode renvoie tout simplement la valeur de l'objet.

#### Syntaxe :

```
variable = objet.valueOf ;
```

### Exemple 25.9 :

```
var tab = new Array(5);
for (i = 0; i < 5; i++) tab[i] = i + 1;
var s = "string";
document.write ("Valeur du Tableau : " + tab.valueOf() + "<br/>");
document.write ("Valeur de la Chaîne : " + s.valueOf() + "<br/>");
```

#### ⇒ Résultat

### 25.3.4.ToString()

Cette méthode retourne la description de l'objet.

#### Syntaxe :

```
variable = objet.toString ;
```

### Exemple 25.9 :

```
var tab = new Array(5);
for (i = 0; i < 5; i++) tab[i] = i + 1;
var s = "string";
document.write ("Description du Tableau : " + tab.toString() + "<br/>");
document.write ("Description de la Chaîne : " + s.toString() + "<br/>");
```

#### ⇒ Résultat

### 25.4.Exercice

Dans cet exercice, vous devez rajouter 3 méthodes à la classe String. La première renverra une chaîne avec la valeur et le constructeur. La seconde renverra la chaîne codée. Et la dernière renverra le dernier caractère de la chaîne.

#### ⇒ Solution

## CONCLUSION

Et voilà ce cours est déjà terminé ! J'espère qu'il vous a plu et qu'il vous a apporté ce que vous souhaitiez. J'attends avec impatience vos commentaires. N'hésitez pas à critiquer mon travail.

Pour ceux qui ont découvert le Javascript, j'espère que ce langage vous a conquis et que vous l'utiliserez à l'avenir. N'oubliez pas qu'il peut s'avérer utile, même en dehors de la programmation Internet.

Merci encore et bonne programmation. Pour ceux qui souhaitent me joindre, allez voir la rubrique suivante.

## AUTEUR

### DESCRIPTION

Pour ceux que ça intéresse, voici un petit descriptif de ma personne. Je m'appelle Michaël, j'ai 17ans, je vis près de Grenoble. Je suis en 1<sup>ère</sup> S, et je vais passer en Terminale. J'ai découvert la programmation l'an dernier grâce à mon professeur de ISI<sup>20</sup> - je le remercie, d'ailleurs - et depuis, je ne peux plus m'en passer. Je programme en HTML/Javascript bien sûr, mais aussi en PHP et en C++.

### COORDONNEES

Pour ceux qui souhaitent me joindre, mon adresse e-mail est [banzaichico@yahoo.fr](mailto:banzaichico@yahoo.fr). Si certains veulent me parler par l'intermédiaire de MSN Messenger, mon contact est [banzaichico@hotmail.com](mailto:banzaichico@hotmail.com). Voilà.

---

<sup>20</sup> Initiation aux Sciences de l'Ingénieur.

## REMERCIEMENTS

Je remercie tous mes amis qui m'ont aidé à progresser en programmation, notamment Arnaud et Dimitri. Je remercie Philippe et M.Allardin pour m'avoir donné goût à ce passe-temps.

Je remercie tous les sites que j'ai consultés pour apprendre le Javascript, dont [www.commentcamarche.com](http://www.commentcamarche.com), [www.toutjavascript.com](http://www.toutjavascript.com), et tant d'autres.

Je remercie mon bon vieux Pentium II et ses 450MHz qui m'on permis de réaliser la plus grande partie de ce cours. Mille mercis à Aiwa, pour avoir construit la chaîne Hi-Fi qui m'accompagne. Merci aussi aux Red Hot Chili Peppers qui ont réalisés les chansons qui m'ont accompagnées tout au long de ce cours.

Merci à Microsoft, pour le nombre de redémarrage suite à un plantage de Windows. Merci à ES-Computing pour avoir créé EditPlus2, logiciel qui m'a permis de faire tous les exemples de ce cours.

Jean Engels

# PHP 7

Cours et exercices



Corrigés des exercices et leur code source

EYROLLES

## Résumé

### **Un cours idéal pour assimiler la syntaxe et les concepts objet de PHP 7 et s'initier au développement d'applications web professionnelles**

Ce manuel d'initiation vous conduira des premiers pas en PHP jusqu'à la réalisation d'un site web complet interagissant avec une base de données MySQL ou SQLite.

Après avoir appris à installer PHP et à créer quelques pages simples, vous étudierez en détail la syntaxe du langage (variables, types de données, opérateurs, instructions, fonctions, tableaux...), avant de progresser rapidement vers des sujets de niveau plus avancé : programmation objet, manipulation des chaînes de caractères et expressions régulières, gestion des mails, sessions et cookies, accès objet aux bases de données MySQL et SQLite, traitements XML, etc.

Successeur de PHP 5 – Cours et exercices du même auteur, cet ouvrage met en avant les nouveautés de PHP 7 : typage des paramètres des fonctions et des valeurs qu'elles retournent, fonctions anonymes, générateurs, classes anonymes, suppression de l'accès procédural à MySQL au profit des méthodes objet, etc.

### **Des exercices corrigés et des travaux pratiques pour une mise en oeuvre immédiate de vos connaissances**

Pour vous aider à valider et mettre en oeuvre vos connaissances, vous trouverez en fin de chaque chapitre une série d'exercices dont les corrigés et le code source sont disponibles sur [www.editions-eyrolles.com](http://www.editions-eyrolles.com) et [www.funhtml.com](http://www.funhtml.com). Vous découvrirez également en fin d'ouvrage trois exemples de sites web dynamiques présentés sous forme de travaux pratiques : à vous de développer ces applications à partir du cahier des charges et des indications fournies dans l'énoncé, en résistant à la tentation de télécharger trop rapidement les solutions données sur le site des éditions Eyrolles !

### **À qui s'adresse cet ouvrage ?**

- Aux étudiants en cursus d'informatique ou de design web.
- À toute personne ayant des bases de programmation web (HTML, JavaScript...) et souhaitant s'autoformer à PHP.
- Aux enseignants et formateurs à la recherche d'une méthode pédagogique pour enseigner PHP. PHP 7

Enseignant en mathématiques et consultant web, **Jean Engels** est auteur de nombreux ouvrages portant sur les technologies du Web : HTML, CSS, JavaScript, PHP et MySQL.

## Sommaire

---

Premier contact avec PHP • Variables, constantes et types • Les instructions de contrôle (if-else, for, while...) • Les chaînes de caractères et les expressions régulières • Les tableaux • Les formulaires • Les fonctions • Dates et calendriers • La programmation objet (classes et instances, héritage, namespaces...) • Les images dynamiques • La gestion des fichiers • Cookies, sessions et emails • Rappels sur les bases de données relationnelles • Le langage SQL et phpMyAdmin • Accès à MySQL avec mysqli • Accès à MySQL avec la couche d'abstraction PDO • La base de données SQLite • PHP et SimpleXML • Travaux dirigés : site de rencontres, dictionnaire de citations interactif, site de commerce électronique.

# AUX ÉDITIONS EYROLLES

## *Du même auteur*

J. ENGELS. – **HTML 5 et CSS 3. Cours et exercices.**  
N°13400, 2012, 550 pages.

## *Autres ouvrages sur PHP et ses frameworks*

P. MARTIN, J. PAULI, C. PIERRE DE GEYER et E. DASPET. – **PHP 7 avancé.**  
N°14357, 2016, 728 pages.

M. CHAVELLI. – **Découvrez le framework PHP Laravel.**  
N°14398, collection OpenClassrooms, 2016, 326 pages.

A. BACCO, préface de F. POTENCIER. – **Développez votre site web avec le framework Symfony3.**  
N°14403, collection OpenClassrooms, 2016, 536 pages.

C. CAMIN. – **Développer avec Symfony2.**  
N°14131, 2015, 450 pages.

## *Dans la même collection*

G. SWINNEN. – **Programmer avec MySQL.**  
N°14302, 4<sup>e</sup> édition, 2015, 480 pages.

C. DELANNOY. – **Programmer en Java.**  
N°11889, 9<sup>e</sup> édition, 2014, 940 pages (réédition avec nouvelle présentation, 2016).

G. SWINNEN. – **Apprendre à programmer avec Python 3.**  
N°13434, 3<sup>e</sup> édition, 2012, 435 pages.

P. ROQUES. – **UML 2 par la pratique.**  
N°12565, 7<sup>e</sup> édition, 2009, 396 pages.

G. SWINNEN. – **Apprendre à programmer avec Python 3.**  
N°13434, 3<sup>e</sup> édition, 2012, 435 pages.

P. ROQUES. – **UML 2 par la pratique.**  
N°12565, 7<sup>e</sup> édition, 2009, 396 pages.

Jean Engels



EYROLLES

**Attention :** pour lire les exemples de lignes de code, réduisez la police de votre support au maximum.

© Groupe Eyrolles, 2017, ISBN : 978-2-212-67360-9.

61, bd Saint-Germain

75240 Paris Cedex 05

[www.editions-eyrolles.com](http://www.editions-eyrolles.com)

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans l'autorisation de l'Éditeur ou du Centre français d'exploitation du droit de copie, 20, rue des Grands-Augustins, 75006 Paris.

# Table des matières

---

## Avant-propos

### CHAPITRE 1

## Introduction

### Avant de commencer

- Compétences requises
- Installation d'un serveur local

### Premier contact avec PHP

- Organisation de PHP
- Structure des fichiers HTML
- Écriture du code PHP
- Ajout de commentaires

### CHAPITRE 2

## Variables, constantes et types

### Les variables

- Affectation par valeur et par référence
- Les variables prédéfinies
- Les opérateurs d'affectation combinée

### Les constantes

- Définir ses constantes personnalisées
- Les constantes prédéfinies

### Les types de données

#### Déterminer le type d'une variable

- La conversion de type
- Contrôler l'état d'une variable

### Les entiers

### Les flottants

### Les opérateurs numériques

## **Les fonctions mathématiques**

### **Les booléens**

Le type boolean

Les opérateurs booléens

### **Les chaînes de caractères**

Définir des chaînes

Concaténer des chaînes

### **Les tableaux**

### **Les objets**

### **Les types divers**

Le type resource

Le type NULL

### **Mémo des fonctions**

### **Exercices**

## CHAPITRE 3

## **Les instructions de contrôle**

### **Les instructions conditionnelles**

L'instruction if

L'instruction if...else

Les opérateurs ? et ??

L'instruction switch...case

### **Les instructions de boucle**

La boucle for

La boucle while

La boucle do...while

La boucle foreach

Sortie anticipée des boucles

### **Gestion des erreurs**

Suppression des messages d'erreur

Gestion des exceptions

### **Exercices**

## CHAPITRE 4

## **Les chaînes de caractères**

## **Affichage des chaînes**

Affichage formaté

Longueur d'une chaîne et codes des caractères

## **Mise en forme des chaînes**

Modification de la casse

Gestion des espaces

## **Entités HTML et caractères spéciaux**

## **Recherche de sous-chaînes**

## **Comparaison de chaînes**

## **Transformation de chaînes en tableaux**

## **Les expressions régulières**

Définition d'un motif élémentaire

Les fonctions de recherche PHP

Définition d'un motif complexe

## **Mémo des fonctions**

## **Exercices**

# **CHAPITRE 5**

## **Les tableaux**

### **Créer des tableaux**

La fonction array()

Créer des suites

Créer un tableau à partir d'une chaîne

Compter le nombre de valeurs d'un tableau

### **Lire les éléments des tableaux**

Lire avec une boucle for

Lire avec une boucle while

Lire à l'aide de la fonction each()

Lire avec each() et list()

L'instruction foreach

### **Manipuler des tableaux**

Extraire une partie d'un tableau

Ajouter et enlever des éléments

Opérations sur plusieurs tableaux

### **Trier les éléments d'un tableau**

Trier des tableaux indicés  
Trier des tableaux associatifs

## Opérer une sélection des éléments

## Appliquer une fonction à un tableau

### L'objet ArrayObject

Création d'un objet tableau

## Les méthodes de tri des éléments

## Mémo des fonctions

## Exercices

# CHAPITRE 6

## Les formulaires

### Création d'un formulaire HTML

L'élément <input />  
L'élément <textarea>  
L'élément <select>  
Exemple de code <form>

### Récupération des données du formulaire

Valeurs uniques  
Les valeurs multiples

### Transfert de fichiers vers le serveur

Transfert de plusieurs fichiers

### Gérer les boutons d'envoi multiples

## Exercices

# CHAPITRE 7

## Les fonctions

### Les fonctions natives de PHP

### Créer ses propres fonctions

Définir une fonction  
Les fonctions qui ne retournent pas de valeur  
Typage des paramètres  
Les fonctions qui retournent une valeur  
Retourner plusieurs valeurs  
Les paramètres par défaut

## **Les fonctions avec un nombre de paramètres variable**

- Les paramètres de type array
- Les fonctions particulières de PHP
- L'opérateur ...

## **Les générateurs**

- Création d'un générateur
- Délégation d'un générateur

## **Portée des variables**

- Variables locales et globales
- Les variables statiques

## **Passer des arguments par référence**

## **Cas particuliers**

- Les fonctions dynamiques
- Les fonctions conditionnelles
- Fonction définie dans une autre fonction
- Les fonctions récursives
- Les fonctions anonymes (closure)

## **Exercices**

# **CHAPITRE 8**

## **Dates et calendriers**

### **Les dates**

- Définir une date
- Vérifier une date

### **Afficher une date en clair**

- La fonction getdate()
- Afficher la date en français

### **Les fonctions de calendrier**

### **Mémo des fonctions**

## **Exercices**

# **CHAPITRE 9**

## **La programmation objet**

### **Terminologie des objets**

### **Classe et instance**

- Création d'une classe
- Créer un objet
- Accès aux variables de la classe
- Les modificateurs d'accessibilité
- Propriétés et méthodes statiques
- Constructeur et destructeur d'objet
- Déréférencement

## **Héritage**

- Enrichir un objet
- Création d'une classe dérivée
- Les traits
- Late Static Binding
- Les classes abstraites
- Les interfaces
- Méthode et classe finales

## **Les classes anonymes**

## **Clonage d'objet**

## **Les namespaces**

- Création et utilisation
- Utilisation des alias

## **Méthodes magiques**

## **Mémo des fonctions**

## **Exercices**

# **CHAPITRE 10**

## **Les images dynamiques**

### **Principes généraux**

- Création du cadre de l'image
- Création des couleurs
- Tracé de formes géométriques
- Écriture de texte

### **Utilisation pratique**

### **Mémo des fonctions**

### **Exercices**

## CHAPITRE 11

# Les fichiers

## Création, ouverture et fermeture d'un fichier

Ouverture du fichier

Fermeture du fichier

Verrouillage des fichiers

## Écriture dans un fichier

Conserver une information

Formatage des données

## Lecture de fichiers

Lire une ligne à la fois

Lire un nombre de caractères donné

Lire un caractère à la fois

Lecture d'une partie d'un fichier

Lecture de données formatées

Lecture de la totalité d'un fichier

## Modifications de fichiers

Copier un fichier

Renommer un fichier

Effacer un fichier

## Informations sur les fichiers

Existence d'un fichier

Taille des fichiers

## Mémo des fonctions

## Exercices

## CHAPITRE 12

# Cookies, sessions et e-mails

## Les cookies

Écriture des cookies

Lecture des cookies

Exemple de page avec cookies

## Les sessions

Le mécanisme des sessions

Session avec cookie

La gestion de panier

Les sessions sans cookie

## L'envoi d'e-mails

La fonction mail()

Envoi d'e-mails au format texte

Envoi d'e-mails au format HTML

## Mémo des fonctions

## Exercices

CHAPITRE 13

## Rappels sur les SGBDR

### Le modèle entité/association

Les entités

Les attributs

Les associations

Les cardinalités

Conception du MCD

Normalisation du MCD

La base magasin en ligne

### Passage au modèle relationnel

Le modèle relationnel

Conception du MLD

Le MLD de la base magasin en ligne

Modèle physique de données

## Exercices

CHAPITRE 14

## Le langage SQL et phpMyAdmin

### L'interface phpMyAdmin

### Création d'une base de données

### Création de tables

Les types de données MySQL

Création des tables

Modification des tables

### Insertion de données

Insertion ligne par ligne

Mise à jour des données  
Importation à partir d'un fichier texte  
Insertion à partir d'un fichier Excel  
Les données de la base magasin

## Sélection des données

Sélection dans une table

## Les jointures

Jointure de deux tables  
Jointure de plus de deux tables

## Exercices

# CHAPITRE 15

## Accès objet à MySQL avec PHP

### Connexion au serveur MySQL

### Envoi de requêtes SQL au serveur

### Lecture du résultat d'une requête

Lecture à l'aide d'un tableau

### Lecture des noms de colonnes

Récupération des valeurs dans un objet

### Insertion de données dans la base

Insertion des données

### Mise à jour d'une table

### Recherche dans la base

### Les requêtes préparées

### Les transactions

### Mémo des méthodes et propriétés

Classe mysqli : méthodes  
Classe mysqli : propriétés  
Classe mysqli\_result : méthodes  
Classe mysqli\_result : propriétés  
Classe mysqli\_stmt : méthodes  
Classe mysqli\_stmt : propriétés

## Exercices

## CHAPITRE 16

# PDO et MySQL

**Connexion au serveur MySQL**

**Envoi de requêtes SQL au serveur**

**Lecture du résultat d'une requête**

Lecture à l'aide d'un tableau

**Lecture des noms de colonnes**

Récupération des valeurs dans un objet

**Insertion de données dans la base**

Insertion des données

**Mise à jour d'une table**

**Recherche dans la base**

**Les requêtes préparées**

**Les transactions**

**Mémo des méthodes**

Classe PDO

Classe PDOStatement

Classe PDOException

**Exercices**

## CHAPITRE 17

# La base SQLite

**Caractéristiques générales**

L'interface SQLiteManager

Méthodes d'accès à SQLite

**La méthode SQLite3**

Connexion à la base

Envoi des requêtes

Insertion de données

Les transactions

Lecture des résultats d'une requête

Création de fonctions SQL personnalisées

Les requêtes préparées

**Accès à SQLite avec PDO**

## **Mémo des méthodes des objets**

- Classe SQLite3
- Classe SQLite3Result
- Classe SQLite3Stmt

## **Exercices**

CHAPITRE 18

# **PHP et SimpleXML**

## **Notions de XML**

### **Lecture d'un fichier XML**

- Accéder au contenu d'un fichier XML
- Lecture des attributs d'un élément
- Lecture d'un fichier à structure complexe
- Modification des valeurs des éléments et des attributs
- Recherche dans un fichier

### **Création d'un fichier XML à partir d'un formulaire**

## **Relations entre XML et une base MySQL**

- Création d'un fichier XML à partir d'une table MySQL
- Création d'une table MySQL à partir d'un fichier XML

## **Mémo des fonctions et méthodes**

## **Exercices**

CHAPITRE 19

# **Travaux personnels**

## **Démarche à suivre**

### **TP n° 1. Un site de rencontres**

- L'interface
- La base de données SQLite

### **TP n° 2. Dictionnaire de citations interactif**

- L'interface
- La base de données MySQL

### **TP n° 3. Commerce en ligne**

- Les besoins du client
- Votre travail

# **Index**

## Introduction

Le sigle PHP signifiait à l'origine *Personal Home Page*. Pour Rasmus Lerdorf, l'auteur de ce qui allait devenir le langage de script côté serveur incorporable dans tout document HTML que nous connaissons, il s'agissait alors d'ajouter quelques fonctionnalités à ses pages personnelles. PHP signifie aujourd'hui *Php Hypertext Preprocessor* car il renvoie à un navigateur un document HTML construit par le moteur de script Zend Engine 2 de PHP, dont nous allons voir le fonctionnement. Il permet de créer des pages web dynamiques et interactives.

Imaginez que vous soyez fan de moto et que vous vouliez présenter les photos de vos modèles préférés et leurs caractéristiques techniques. La création de quelques pages HTML statiques, agrémentées de liens pour naviguer d'une page à l'autre, peut suffire. Imaginez maintenant que vous soyez rejoint par d'autres personnes qui partagent la même passion et que votre site présente des centaines de modèles et une rubrique de petites annonces et de contacts entre membres. La quantité d'informations à présenter ne permet plus de naviguer dans le site au moyen de liens mais réclame, dès la page d'accueil, un moteur de recherche. L'utilisateur saisit un ou plusieurs critères de recherche, à partir desquels le code d'un script PHP crée une page contenant les informations recherchées et seulement elles. Chaque visiteur et chaque besoin particulier génèrent donc des pages différentes, personnalisées, construites dynamiquement.

PHP permet en outre de créer des pages interactives. Une page interactive permet à un visiteur de saisir des données personnelles. Ces dernières sont ensuite transmises au serveur, où elles peuvent rester stockées dans une base de données pour être diffusées vers d'autres utilisateurs. Un visiteur peut, par exemple, s'enregistrer et retrouver une page adaptée à ses besoins lors d'une visite ultérieure. Il peut aussi envoyer des e-mails et des fichiers sans avoir à passer par son logiciel de messagerie. En associant toutes ces caractéristiques, il est possible de créer aussi bien des sites de diffusion et de collecte d'information que des sites d'e-commerce, de rencontres ou des blogs.

Pour contenir la masse d'informations collectées, PHP s'appuie généralement sur une base de données, généralement MySQL mais aussi SQLite, et sur des serveurs Apache. PHP, MySQL et Apache forment d'ailleurs le trio ultradominant sur les serveurs

Internet. Quand ce trio est associé sur un serveur à Linux, on parle de système LAMP (Linux, Apache, MySQL, PHP). PHP est utilisé aujourd’hui par plus des trois quarts des sites dynamiques de la planète et par les trois quarts des grandes entreprises françaises. Pour un serveur Windows, on parle de système WAMP, mais ceci est beaucoup moins courant.

Vous passerez en revue dans le cours de cet ouvrage tous les outils nécessaires à la réalisation d’un site dynamique et interactif à l’aide de PHP et d’une base de données MySQL ou SQLite. Les principaux avantages de ces outils sont la facilité d’apprentissage, la grande souplesse d’utilisation, l’excellent niveau de performance et, ce qui ne gâte rien, la gratuité.

Pour parvenir à la réalisation des types de site que nous venons de voir nous allons aborder successivement les points suivants :

- La syntaxe et les caractéristiques du langage PHP, dont la connaissance est la base indispensable à toute la suite.
- Les notions essentielles du langage SQL permettant la création et la gestion des bases de données et la réalisation des requêtes sur ces bases.
- Le fonctionnement et la réalisation de bases de données MySQL puis SQLite et les moyens d’y accéder à l’aide des fonctions spécialisées de PHP ou d’objets.

Pour progresser rapidement il vous sera nécessaire de lire ce livre de manière linéaire au moins pour le début et de ne pas brûler les étapes. N’essayez donc pas de commencer par la fin en abordant les bases de données sans connaissance préalable de PHP ou de SQL.

## Avant de commencer

Avant d’envisager d’écrire votre premier script, il vous faut faire le point sur les connaissances nécessaires à cette réalisation. Il n’est pas envisageable de commencer cet apprentissage sans aucune connaissance d’Internet et de la création de pages HTML. Du point de vue matériel, vous devez de surcroît disposer des quelques outils qui vous permettront d’écrire et surtout de tester vos scripts sur un ordinateur personnel.

## *Compétences requises*

L’objectif de cet ouvrage étant de permettre un apprentissage progressif de PHP, la connaissance d’un langage de programmation quelconque n’est pas vraiment indispensable. Cependant, quelques notions de programmation en langage C, Java ou en JavaScript, par exemple, ne peuvent que rendre l’accès à PHP plus facile. En revanche, la connaissance du langage HTML est recommandée puisque le serveur PHP renvoie les pages HTML que vous programmez.

Pour ce qui concerne la méthode, commencez par télécharger et tester les exemples du

livre, puis modifiez-en certains paramètres afin d'évaluer le rôle de chacun d'eux. Cela vous permettra de mieux apprécier l'effet réel d'une instruction, par exemple.

## Les outils de création

Puisqu'il s'agit de construire des pages web et de produire un document HTML lisible par un navigateur, un éditeur HTML peut convenir pour créer la structure générale des pages, y compris s'il est WYSIWYG. Le code des scripts PHP peut quant à lui être écrit dans n'importe quel éditeur de texte, tel que le Bloc-notes de Windows.

Si les éditeurs tels que Dreamweaver privilégient l'aspect visuel en cachant le code, d'autres outils de création très simples, comme HTML Kit, obligent le programmeur à voir en permanence les éléments HTML utilisés. Un bon compromis consiste à utiliser un éditeur WYSIWYG pour créer le design et la mise en page générale des pages web puis de récupérer le fichier HTML réalisé dans un éditeur PHP spécialisé afin d'effectuer les tests facilement après avoir installé le serveur local PHP.

Le tableau 1-1 présente une liste d'outils de développement de scripts.

**Tableau 1-1 – Éditeurs HTML et PHP**

Produit	Statut	Description	Adresse
HTML Kit	Gratuit	Éditeur HTML	<a href="http://www.htmlkit.com">http://www.htmlkit.com</a>
HTMLPad Rapid PHP	Payants	Éditeurs HTML et PHP	<a href="http://www.blumentals.net">http://www.blumentals.net</a>
EditPlus	Payants	Éditeur HTML permettant l'écriture et l'exécution de scripts PHP	<a href="http://www.editplus.com">http://www.editplus.com</a>
Zend Studio	Payants	Très complet	<a href="http://www zend.com">http://www zend.com</a>

## Installation d'un serveur local

Faute de disposer d'un serveur local sur votre ordinateur personnel, vous seriez obligé pour tester vos pages PHP de les transférer sur le serveur distant de votre hébergeur puis d'appeler ces pages en vous connectant au site à l'aide de votre navigateur. La moindre erreur de code ou la moindre modification vous obligera à répéter toute cette procédure, d'où une importante perte de temps.

Il est donc indispensable d'installer sur votre poste de travail un serveur local simulant votre serveur distant et vous permettant d'effectuer en direct tous les tests désirés. Vous aurez alors dans votre navigateur exactement le même aspect pour toutes ces pages que les visiteurs de votre site quand vous aurez opéré le transfert de vos fichiers sur le serveur distant qui l'hébergera.

Le serveur local comprend les éléments suivants, disponibles séparément aux adresses entre parenthèses :

- Serveur Apache (<http://www.apache.org>).

- Interpréteur de code PHP (<http://www.php.net>).
- Base de données MySQL (<http://www.mysql.com>).
- Base de données SQLite (<http://www.sqlite.org>).
- Utilitaire phpMyAdmin, qui permet de créer et de gérer bases et tables de données MySQL(<http://www.phpmyadmin.net>).
- Utilitaire SQLiteManager, qui permet de créer et de gérer bases et tables de données SQLite (<http://www.sqlitemanager.org>).

On peut trouver sur le Web divers packages complets pour Windows, Linux ou Mac, qui permettent d'installer en une seule opération tous ces éléments, évitant du même coup les problèmes de configuration.

Un installateur est apparu à l'occasion de la sortie de PHP 5. Son auteur, Romain Bourdon, se montre très réactif en publant une nouvelle version à chaque évolution. Son package, nommé Wampserver, téléchargeable à l'adresse <http://www.wampserver.com>, est destiné aux ordinateurs sous Windows.

Une fois la procédure de téléchargement terminée, il vous suffit de lancer l'exécutable, qui installe automatiquement Apache, PHP, MySQL, SQLite phpMyAdmin et SQLiteManager sur votre ordinateur. Si, pendant la phase d'installation, vous avez choisi d'installer PHP en tant que service Windows, le serveur est lancé automatiquement à chaque démarrage du système d'exploitation.

Sous Mac il existe un très bon installateur de serveur local nommé MAMP, très facile d'emploi – c'est celui-ci que nous présentons ici mais WampServer est aussi accessible.

Pour pouvoir être exécutés par le serveur local, tous les scripts que vous écrivez doivent être enregistrés dans le sous-dossier `www` sous Windows ou `MAMP/htdocs` sous Mac. Dans ce dernier, vous pouvez créer un ou plusieurs sous-dossiers correspondant à chaque site que vous voulez tester.

La page de démarrage du serveur vous permet de lancer les serveurs Apache et MySQL (voir figure 1-1). L'icône « Ouvrir la page Webstart » déclenche l'ouverture de la page d'accueil dans votre navigateur par défaut. Cette page d'administration du serveur local (voir figure 1-2) vous donne accès à différents paramètres, l'onglet « Mon site web » affiche la liste de tous les fichiers présents dans le dossier `htdocs` et l'onglet « Outils » donne accès à phpMyAdmin pour gérer les bases MySQL et à phpLiteAdmin pour les bases SQLite.



**Figure 1-1**  
*La page de démarrage des serveurs*



**Figure 1-2**  
*Page d'administration du serveur local Apache PHP MySQL*

## Premier contact avec PHP

Étant désormais doté de tous les outils nécessaires, vous pouvez aborder le fonctionnement de PHP et les différentes méthodes de travail que vous devrez utiliser par la suite.

### *Organisation de PHP*

PHP ne repose pas sur une hiérarchie de classes regroupées en sous-ensembles (namespace), comme ASP.Net ou Java, mais sur des modules. Le module de base, dit standard, permet d'accéder aux instructions élémentaires, aux différents types de données et à un grand nombre de fonctions. Des modules additionnels spécialisés permettent d'ajouter des fonctionnalités particulières, comme l'accès aux diverses bases de données et leur gestion. Chaque module donne accès à un grand nombre de fonctions

spécialisées pour un domaine particulier.

La liste des modules disponibles actuellement est visible dans la documentation générale du langage sur le site officiel de PHP, à l'adresse <http://www.php.net>.

Vous pouvez télécharger sur le même site la documentation officielle de PHP, qui donne, y compris en français, la définition de toutes les fonctions existantes. Le document compte quelque deux mille pages au format Acrobat PDF.

Pour savoir quels modules vous pouvez utiliser sur votre serveur local, il vous suffit de cliquer sur l'onglet `phpinfo` de la page d'accueil.

Pour obtenir la même information pour le serveur qui héberge votre site, procédez de la façon suivante :

1. Écrivez le script PHP suivant, d'une simplicité enfantine (vous n'en écrirez jamais d'aussi court donnant autant d'informations), à l'aide de l'éditeur que vous avez choisi :

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

2. Enregistrez le script sous le nom `info.php`. Sous PHP, tous les scripts commencent par la ligne `<?php` et se terminent par `?>`. Notez que, sauf recommandation spéciale de votre hébergeur, tous les fichiers qui contiennent des instructions PHP sont enregistrés avec l'extension `.php`. Les extensions `.php3`, `.php4`, `.php5` ou `.phtml` se rencontrent sur certains serveurs, suivant la configuration effectuée par l'administrateur.
3. Transférez le fichier `info.php` sur votre serveur distant à l'aide d'un logiciel FTP. Si vous n'en avez pas, vous pouvez télécharger FileZilla, un logiciel gratuit, dont le fonctionnement est aussi simple que convivial, à l'adresse <http://www.sourceforge.net/projects/filezilla>.
4. Saisissez l'adresse <http://www.votresite.com/info.php> dans votre navigateur.

Un grand nombre d'informations utiles concernant votre serveur et l'ensemble des modules qui y sont installés apparaissent alors sur toute la hauteur de la page (voir figure 1-3).

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 HTML***

Comme expliqué précédemment, la connaissance du langage HTML est utile pour se lancer dans l'écriture de scripts PHP. Il est donc utile de connaître la structure des

fichiers HTML car une page dynamique PHP est bien un document HTML envoyé par le serveur vers le poste client.

The screenshot shows the output of the `phpinfo()` function. At the top, it displays "PHP Version 7.0.0". Below this is a table containing various PHP configuration parameters and their values. Key entries include:

- System**: Darwin Air-de-ENGELS 16.3.0 Darwin Kernel Version 16.3.0: Thu Nov 17 20:23:58 PST 2016; root:xnu-3789.31.2~1/RELEASE\_X86\_64 x86\_64
- Build Date**: Dec 8 2015 16:35:17
- Configure Command**: A long command line showing the configuration options used to build PHP, including paths to MAMP libraries and various extensions like MySQL, PDO, and OpenSSL.
- Server API**: Apache 2.0 Handler
- Virtual Directory Support**: disabled
- Configuration File (php.ini) Path**: /Applications/MAMP/bin/php/php7.0.0/conf
- Loaded Configuration File**: /Applications/MAMP/bin/php/php7.0.0/conf/php.ini
- Scan this dir for additional .ini files**: (none)
- Additional .ini files parsed**: (none)
- PHP API**: 20151012
- PHP Extension**: 20151012
- Zend Extension**: 320151012
- Zend Extension Build**: API20151012,NTS
- PHP Extension Build**: API20151012,NTS
- Debug Build**: no
- Thread Safety**: disabled
- Zend Signal Handling**: disabled
- Zend Memory Manager**: enabled
- Zend Multibyte Support**: provided by mbstring

**Figure 1-3**

*Informations concernant le serveur fournies par phpinfo()*

Pour être conforme aux recommandations HTML 5, un document doit avoir la structure suivante (fichier `pagehtml.html`) :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Titre de la page</title>
</head>
<body>
<h2>Bienvenue sur le site PHP 7 </h2>
</body>
</html>
```

Cette page primaire est écrite en HTML 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 HTML, 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>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<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 7</h2>";
?>
</body>
</html>
```

Le code de votre nouvelle page contient les nouveaux éléments suivants, qui ne sont pas du HTML :

```
<?php
echo "<h3> Aujourd'hui le ". date('d / M / Y H:m:s')." </h3><hr />";
echo "<h2>Bienvenue sur le site PHP 7</h2>";
?>
```

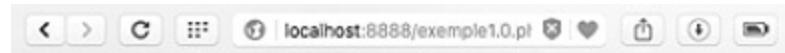
Les éléments `<?php` et `?>` marquent respectivement le début et la fin de tout script PHP, qu'il soit inclus dans du code HTML ou isolé dans un fichier ne contenant que du code PHP. Vous pouvez inclure autant de blocs de code PHP que vous le désirez dans un document HTML, à condition que chacun d'eux soit délimité par ces marqueurs.

Entre ces éléments figure le code PHP proprement dit :

```
echo "<h3> Aujourd'hui le ". date('d / M / Y H:m:s')." </h3><hr />";
echo "<h2>Bienvenue sur le site PHP 7</h2>";
```

L'instruction `echo` permet d'écrire dans le document final le contenu qui la suit, que ce soit du texte ou le résultat retourné par une fonction, comme dans les deux lignes précédentes. Notez que les lignes de code PHP se terminent toujours par un point-virgule.

Si vous recopiez et exécutez ce fichier dans votre navigateur, vous obtenez le résultat illustré à la figure 1-5, qui donne un aperçu de ce qu'est une page dynamique élémentaire. Vous pourriez faire la même chose à l'aide d'un script JavaScript exécuté non pas sur le serveur mais par le navigateur du poste client. La différence est que la date et l'heure affichées ici sont celles du serveur et pas celle de votre ordinateur, comme le ferait JavaScript. L'un des avantages de PHP est cependant que vous n'avez pas à tenir compte des capacités du navigateur du visiteur.



**Figure 1-4**  
*Résultat de votre première page PHP*

Examinez maintenant le code source du document tel qu'il a été reçu par le navigateur. Dans Opera, par exemple, allez dans le menu Développeur>Afficher la source. Le code suivant s'affiche :

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Une page PHP</title>
</head>
<body>
  <h3> Aujourd'hui le 22 / Jul / 2016 22:07:53</h3><hr />
  <h2>Bienvenue sur le site PHP 7</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 7</h2>";
?>
```

a été remplacé par :

```
<h3> Aujourd'hui le 22 / Jul / 2016 22:07:53</h3><hr />
<h2>Bienvenue sur le site PHP 7</h2>
```

L'interpréteur PHP analyse le document dans son ensemble puis renvoie le code HTML 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 HTML, 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 HTML.

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 souvent incorporé dans du code HTML. Vous pouvez donc incorporer autant de scripts PHP indépendants que vous le souhaitez n'importe où dans du code HTML, du moment que ces parties sont délimitées par les balises ouvrantes et fermantes `<?php` et `"?>"` (repères 1, 2, 4, 6 et 7). Les formes `<?=` et `?>` (repères 5 et 8) ou encore l'élément HTML `<script language="php">` (repère 3), qui est rarement employé, sont désormais considérés comme obsolètes et déconseillés. Citée pour mémoire car certains l'utilisaient, cette dernière forme n'est même plus reconnue du tout par PHP 7 (voir figure 1-5), donc à proscrire.

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

Le listing suivant illustre cette particularité :

```
<!DOCTYPE html>
<?php ← 1
    $variable1=" PHP 7";
?>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<?php ← 2
echo "<title>Une page pleine de scripts PHP</title>";
?>
</head>
<body>
<script language="php"> ← 3
    echo"<h1>BONJOUR A TOUS </h1>";
</script>
<?php ← 4
    echo "<h2> Titre écrit par PHP</h2>";
    $variable2=" MySQL";
?>
<p>Vous allez découvrir <?= $variable1 ?> ← 5 </p>
<?php ← 6
    echo "<h2> Bonjour de $variable1</h2>";
?>
<p>Utilisation de variables PHP<br />Vous allez découvrir également
```

```
<?php← 7
echo $variable2
?>
</p>
<?= "<div><big>Bonjour de $variable2 </big></div>" ?> ← 8
</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 HTML.

Certains de ces scripts interviennent comme contenu d'un élément HTML avec une syntaxe particulière. Par exemple :

```
<?= $variable1 ?>
```

peut être utilisé pour des instructions courtes. Il est équivalent à :

```
<?php echo $variable1 ?>.
```

À partir de ce document, vous obtenez le résultat illustré à la figure 1-5.



**Figure 1-5**  
*Résultat des mini-scripts*

Comme précédemment, la consultation du code source dans le navigateur montrerait que le résultat de chaque mini-script est purement HTML et qu'aucun code PHP ne subsiste.

## Inclure des fichiers externes

Comme en JavaScript, il est possible d'écrire du code PHP ou HTML dans des fichiers séparés puis de les incorporer dans du code HTML ou d'autres scripts PHP en fonction des besoins. Cela peut constituer un début de modularisation du code, permettant d'écrire une seule fois certaines parties de code et de les réutiliser dans plusieurs pages différentes, avec économie de temps. Cette possibilité permet notamment de créer une bibliothèque de fonctions d'utilisation courante.

On donne généralement aux fichiers de code PHP l'extension `.inc` ou `.inc.php`, cette dernière ayant l'avantage de protéger les données confidentielles que peut contenir le

code, comme les paramètres de connexion à la base de données (login et mot de passe). Le contenu du fichier est interprété par le serveur. Si le fichier ne contient que vos paramètres dans des variables, le serveur ne renvoie rien au poste client si quelqu'un tente de l'exécuter, alors qu'un navigateur affiche le contenu d'un fichier avec l'extension .inc seule.

Pour inclure le contenu d'un fichier externe dans du code PHP, vous disposez des fonctions recensées au tableau 1-2.

**Tableau 1-2 – Fonctions d'inclusion de code externe**

Fonction	Description
<code>include("nom_fichier.ext")</code>	Lors de son interprétation par le serveur, cette ligne est remplacée par tout le contenu du fichier précisé en paramètre, dont vous fournissez le nom et éventuellement l'adresse complète. En cas d'erreur, par exemple si le fichier n'est pas trouvé, <code>include()</code> ne génère qu'une alerte, et le script continue.
<code>require("nom_fichier.ext")</code>	A désormais un comportement identique à <code>include()</code> , à la différence près qu'en cas d'erreur, <code>require()</code> provoque une erreur fatale et met fin au script.
<code>include_once("nom_fichier.ext")</code> <code>require_once("nom_fichier.ext")</code>	Contrairement aux deux précédentes, ces fonctions ne sont pas exécutées plusieurs fois, même si elles figurent dans une boucle ou si elles ont déjà été exécutées une fois dans le code qui précède.

L'exemple suivant utilise les possibilités d'inclusion fournies par ces fonctions pour créer une page HTML à partir de quatre fichiers indépendants. Il s'agit d'un début de modularisation du code d'un site. Notre hypothèse est que chaque page du site a le même en-tête et le même pied de page et que chacune des pages ne diffère des autres que par son contenu.

L'exemple comprend les fichiers suivants :

- `tete.inc.php`. Contient le début du code HTML d'une page normale (`<html>`, `<head>`, `<body>`) et trois petits scripts PHP. Le dernier de ces scripts (repère ①) affiche le bandeau commun à toutes les pages (repère ②) ainsi que le nom du fichier exécuté et celui du fichier inclus (repère ③).
- `corps.inc.php`. Ne contient que du code PHP affichant deux lignes de texte (repère ④).
- `corps.html`. Ne contient que du code HTML affichant deux lignes de texte (repère ⑤).
- `pied.inc.php`. Contient un script affichant un bandeau de pied de page et deux liens vers des sites dignes d'intérêt (repère ⑥).
- `principal.php`. Script utilisant les quatre précédents à l'aide des fonctions `include()` (repère ①), `include_once()` (repère ②), `require()` (repère ③) et `require_once()` (repère ④).

(repère ④). C'est le seul qui doit être appelé directement. Les autres fichiers n'étant que des composants, ils ne doivent normalement pas être utilisés seuls.

La figure 1-6 donne un aperçu du résultat obtenu.

## Exemple 1-1. Inclusion de fichiers externes

Le fichier `tete.inc.php` :

```
<!DOCTYPE html>
<?php
    $variable1=" PHP 7";
?>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<?php
echo "<title>Une page pleine d'inclusions $variable1</title>";
?>
</head>
<body>
<?php← 1
$variableext="Ce texte provient du fichier inclus";
echo "    <div><h1 style=\"border-width:5px; border-style:double; background-color:#ffcc99;\">
Bienvenue sur le site $variable1 </h1>";← 2
echo "<h3> $variableext</h3>";
echo "Nom du fichier ex&#233;cut&#233; : ", $_SERVER['PHP_SELF'], " &ampnbsp&ampnbsp";
← 3
echo " Nom du fichier inclus : ", __FILE__ , "</div> "; ← 3
?>
```

Le fichier `corps.inc.php` :

```
<?php
echo "<h1> Ceci est le corps du document </h1>"; ← 4
echo "<h2> Ceci est le corps du document </h2>";
?>
```

Le fichier `corps.html` :

```
<h1> Ceci est le corps du document : Avec PHP on progresse vite et avec MySQL le
site devient vite très dynamique.....</h1>
<h2> On s'y met tout de suite!!!! </h2> ← 5
```

Le fichier `pied.inc.php` :

```
<hr />
<?php
echo "<div><h1 style=\"border-width:3px; border-style:groove; background-color:#ffcc99;\"> Fin de la page PHP Liens utiles : <a href=\"php.net\">php.net</a>
&ampnbsp <a href=\"mysql.org\">mysql.org</a></h1>"; ← 6
echo "Nom du fichier ex&#233;cut&#233; : ", $_SERVER['PHP_SELF'], " &ampnbsp&ampnbsp";
echo "Nom du fichier inclus: ", __FILE__ , "</div>";
?>
</body>
```

```
| </html>
```

Le fichier `principal.php` :

```
<?php  
include("tete.inc.php"); ←①  
echo "<hr />";  
include_once("corps.inc.php"); ←②  
require("corps.html"); ←③  
require_once("pied.inc.php"); ←④  
?>
```



**Figure 1-6**

*Un page composée de fichiers inclus*

## Ajout de commentaires

Il est toujours utile de commenter les scripts que vous écrivez. Lors de l'écriture, tout peut paraître évident, mais à la relecture, plusieurs mois plus tard, lorsqu'il s'agit d'effectuer des mises à jour, par exemple, autant éviter de perdre du temps à redécouvrir la logique adoptée auparavant.

Les commentaires ne sont pas pris en compte par l'analyseur PHP. S'ils alourdisSENT un peu le fichier PHP en termes d'octets sur le serveur, ils ne sont pas présENTS dans le code HTML renvoyé au navigateur client. Leur poids est donc sans importance pour la rapidité de transmission des pages.

PHP supporte les trois syntaxes de commentaires suivantes :

- commentaires sur une seule ligne introduits par les caractères `//` :

```
| //Ceci est un commentaire court sur une ligne
```

- commentaires sur plusieurs lignes introduits par les caractères `/ *` et fermés par les caractères `*/` :

```
/* Ceci est commentaire abondant  
qui va occuper plusieurs lignes  
et va expliquer le code qui suit..... */
```

- commentaires de type Unix, ne comportant qu'une seule ligne introduite par le caractère # :

```
*****  
# commentaires de type Unix  
*****
```

## Variabes, constantes et types

Comme tout langage, PHP manipule des données. Pour un site dynamique, ces données sont variables. De plus, elles peuvent être de types différents, tel du texte sous forme de chaîne de caractères, comme vous en avez utilisé avec l'instruction `echo`, sous forme de nombres entiers ou décimaux ou encore sous forme de valeurs booléennes vrai ou faux (`TRUE` ou `FALSE`). Ces types de base sont les plus employés, mais il en existe d'autres, qui peuvent être des types composés, comme les tableaux et les objets, ou des types particuliers, comme `resource` ou `NULL`.

### Les variables

Une variable est le conteneur d'une valeur d'un des types utilisés par PHP (entiers, flottants, chaînes de caractères, tableaux, booléens, objets, ressource ou `NULL`).

Chaque variable possède un identifiant particulier, qui commence toujours par le caractère dollar (\$) suivi du nom de la variable. Les règles de création des noms de variable sont les suivantes :

- Le nom commence par un caractère alphabétique, pris dans les ensembles `[a-z]`, `[A-Z]` ou par le caractère de soulignement (`_`).
- Les caractères suivants peuvent être les mêmes plus des chiffres.
- La longueur du nom n'est pas limitée, mais il convient d'être raisonnable sous peine de confusion dans la saisie du code. Il est conseillé de créer des noms de variable le plus « parlant » possible. En relisant le code contenant la variable `$nomclient`, par exemple, vous comprenez davantage ce que vous manipulez que si vous aviez écrit `$x` ou `$y`.
- La déclaration des variables n'est pas obligatoire en début de script. C'est là une différence notable avec les langages fortement typés comme Java ou C. Vous pouvez créer des variables n'importe où, à condition bien sûr de les créer avant de les utiliser, même s'il reste possible d'appeler une variable qui n'existe pas sans provoquer d'erreur.
- L'initialisation des variables n'est pas non plus obligatoire et une variable non

initialisée n'a pas de type précis.

- Les noms des variables sont sensibles à la casse (majuscules et minuscules). \$mavar et \$MaVar ne désignent donc pas la même variable.

Les noms de variables suivants sont légaux :

```
$mavar  
$_mavar  
$mavar2  
$M1  
$_123
```

Les suivants sont illégaux :

```
$5mamar  
$*mavar  
$mavar+
```

## Affectation par valeur et par référence

L'affectation consiste à donner une valeur à une variable. Comme expliqué précédemment, lors de la création d'une variable, vous ne déclarez pas son type. C'est la valeur que vous lui affectez qui détermine ce type. Dans PHP, vous pouvez affecter une variable par valeur ou par référence. Vous verrez que les méthodes et les conséquences de ces deux types d'affectation sont différentes et peuvent amener des résultats inattendus, si vous n'y prenez garde.

L'affectation par valeur se fait à l'aide de l'opérateur `=`, soit après la création de la variable, soit en même temps.

Dans l'exemple suivant :

```
$mavar = expression;
```

la variable `$mavar` prend la valeur de l'expression, qui peut être une valeur numérique, par exemple, une chaîne de caractères littérale, mais aussi une autre variable ou encore une expression PHP valide contenant des fonctions.

Dans les affectations suivantes :

```
$mavar=75;  
$mavar="Paris";  
$mavar=7*3+2/5-91%7; //PHP évalue l'expression puis affecte le résultat  
$mavar=mysql_connect($a,$b,$c); //la fonction retourne une ressource  
$mavar=isset($var&&($var==9)); //la fonction retourne une valeur booléenne
```

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 />";
$mavar2="Marseille";
echo "\$mavar1= ",$mavar1,"<br />";
echo "\$mavar2= ",$mavar2,"<br />";
?>
```

Le résultat de l'exécution de ce script montre l'évolution des valeurs des deux variables après plusieurs affectations.

---

```
$mavar1= Paris
$mavar2= Lyon
affectation par référence de $mavar2
$mavar1= Paris
```

```
$mavar2= Paris  
modification de $mavar1  
$mavar1= Nantes  
$mavar2= Nantes  
modification de $mavar2  
$mavar1= Marseille  
$mavar2= Marseille
```

---

Lorsque vous utilisez ce type d'affectation, il est important de ne pas oublier ses effets en cours de script car chacune de ces deux variables change de valeur de manière sousjacente chaque fois que vous intervenez sur l'autre, sans que la modification soit explicitement écrite.

Les variables peuvent aussi être définies dynamiquement, c'est-à-dire que leur nom est contenu dans une variable dont la valeur peut changer en cours de script. En écrivant le code suivant :

```
| $a="PHP";  
| $$a="Open Source";
```

nous créons une variable nommée "PHP" (équivalente à `$PHP`) et dont la valeur est "Open source". Si nous voulons utiliser le contenu de la variable dynamique dans une chaîne, nous devons écrire la syntaxe `$$a` pour obtenir la chaîne "Open Source". Par exemple, le code qui suit :

```
| echo "$a est $$a";  
affiche : "PHP est Open Source".
```

## ***Les variables prédéfinies***

PHP dispose d'un grand nombre de variables prédéfinies, qui contiennent des informations à la fois sur le serveur et sur toutes les données qui peuvent transiter entre le poste client et le serveur, comme les valeurs saisies dans un formulaire (voir le chapitre 6), les cookies ou les sessions (voir le chapitre 13).

Depuis PHP 4.1, ces variables se présentent sous la forme de tableaux, accessibles en tout point de n'importe quel script. On appelle ces tableaux *superglobaux*. Le tableau 2-1 donne une brève description de ces variables sans en détailler le contenu car un certain nombre d'entre elles ne présentent pas un intérêt pratique immédiat. Vous aurez toute précision nécessaire sur leur utilisation à mesure que vous les utiliserez dans le cours de l'ouvrage. Reportez-vous à la section concernant les tableaux pour lire le contenu de ces variables.

**Tableau 2-1 – Les variables serveur PHP**

\$GLOBALS	Contient le nom et la valeur de toutes les variables globales du script. Les noms des variables sont les clés de ce tableau.  \$GLOBALS["mavar"] récupère la valeur de la variable <code>\$mavar</code> en dehors de sa zone de visibilité (dans les fonctions, par exemple).
	Contient le nom et la valeur des cookies enregistrés sur le poste client. Les

<code>\$_COOKIE</code>	noms des cookies sont les clés de ce tableau (voir le chapitre 13). Avant PHP 4.1, cette variable se nommait <code>\$HTTP_COOKIES_VARS</code> .
<code>\$_ENV</code>	Contient le nom et la valeur des variables d'environnement qui sont changeantes selon les serveurs. Avant PHP 4.1, cette variable se nommait <code>\$HTTP_ENV_VARS</code> .
<code>\$_FILES</code>	Contient le nom des fichiers téléchargés à partir du poste client. Avant PHP 4.1, cette variable se nommait <code>\$HTTP_FILES_VARS</code> .
<code>\$_GET</code>	Contient le nom et la valeur des données issues d'un formulaire envoyé par la méthode <code>GET</code> . Les noms des champs du formulaire sont les clés de ce tableau (voir le chapitre 6). Avant PHP 4.1, cette variable se nommait <code>\$HTTP_GET_VARS</code> .
<code>\$_POST</code>	Contient le nom et la valeur des données issues d'un formulaire envoyé par la méthode <code>POST</code> . Les noms des champs du formulaire sont les clés de ce tableau. Avant PHP 4.1, cette variable se nommait <code>\$HTTP_POST_VARS</code> .
<code>\$_REQUEST</code>	Contient l'ensemble des variables superglobales <code>\$_GET</code> , <code>\$_POST</code> , <code>\$_COOKIE</code> et <code>\$_FILES</code> . Avant PHP 4.1, cette variable n'existe pas.
<code>\$_SERVER</code>	Contient les informations liées au serveur web, tel le contenu des en-têtes HTTP ou le nom du script en cours d'exécution. Retenons les variables suivantes :  <code>\$_SERVER["HTTP_ACCEPT_LANGUAGE"]</code> , qui contient le code de langue du navigateur client. <code>\$_SERVER["HTTP_COOKIE"]</code> , qui contient le nom et la valeur des cookies lus sur le poste client. <code>\$_SERVER["HTTP_HOST"]</code> , qui donne le nom de domaine. <code>\$_SERVER["SERVER_ADDR"]</code> , qui indique l'adresse IP du serveur. <code>\$_SERVER["PHP_SELF"]</code> , qui contient le nom du script en cours. Nous l'utiliserons souvent dans les formulaires. <code>\$_SERVER["QUERY_STRING"]</code> , qui contient la chaîne de la requête utilisée pour accéder au script.
<code>\$_SESSION</code>	Contient l'ensemble des noms des variables de session et leurs valeurs.

## ***Les opérateurs d'affectation combinée***

En plus de l'opérateur classique d'affectation `=`, il existe plusieurs opérateurs d'affectation combinée. Ces opérateurs réalisent à la fois une opération entre deux opérandes et l'affectation du résultat à l'opérande de gauche.

Le tableau 2-2 décrit l'ensemble de ces opérateurs.

**Tableau 2-2 – Les opérateurs d'affectation combinée**

Opérateur	Description
<code>+=</code>	Addition puis affectation : <code>\$x += \$y</code> équivaut à <code>\$x = \$x + \$y</code> <code>\$y</code> peut être une expression complexe dont la valeur est un nombre.

--	Soustraction puis affectation : \$x -= \$y équivaut à \$x = \$x - \$y \$y peut être une expression complexe dont la valeur est un nombre.
*=	Multiplication puis affectation : \$x *= \$y équivaut à \$x = \$x * \$y \$y peut être une expression complexe dont la valeur est un nombre.
**=	Puissance puis affectation \$x**=2 équivaut à \$x=(\$x)²
/=	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.

### Exemple 2-2. Crédit et lecture de constantes

```
<?php
// Définition insensible à la casse
define("PI",3.1415926535,TRUE); ←①
// Utilisation
echo "La constante PI vaut ",PI,"<br />";
echo "La constante PI vaut ",pi,"<br />";
// Vérification de l'existence
if (defined( "PI")) echo "La constante PI est déjà définie","<br />";
if (defined( "pi")) echo "La constante pi est déjà définie","<br />";
// Définition sensible à la casse, vérification de l'existence et utilisation
if(define("site","http://www.funhtml.com",FALSE)) ←②
{
echo "<a href=\" " ,site, " \">>Lien vers mon site </ a>";
}
?>
```

La constante `PI` étant déclarée insensible à la casse (repère ①), elle peut être utilisée sous la forme `PI` ou `pi` ou encore `Pi` ou toute autre variante. Par contre, la constante `site` est déclarée sensible à la casse (repère ②) et ne peut être utilisée qu'en minuscules.

Nous pouvons également définir des constantes à l'aide du mot-clé `const` comme c'était le cas à l'intérieur d'une classe, mais il est maintenant utilisable en dehors de celle-ci. De plus, la définition d'une constante peut contenir une expression scalaire contenant des nombres, des chaînes ou d'autres constantes. Par exemple :

```
const PI= 3.14;
const PI_SUR_4 = PI/4;
const 2PI = 2*PI;
```

Une constante peut aussi être un tableau en écrivant par exemple :

```
const NOM = ['Bach','Saint Matthieu'];
//ou encore : define('NOM', array('bach','Saint Matthieu'));
```

et les utiliser comme des éléments de tableau :

```
echo NOM[1]; // qui affiche 'Saint Matthieu'
```

## Les constantes prédefinies

Il existe dans PHP un grand nombre de constantes prédefinies, que vous pouvez

notamment utiliser dans les fonctions comme paramètres permettant de définir des options. Nous ne pouvons les citer toutes tant elles sont nombreuses, mais nous les définirons au fur et à mesure de nos besoins.

Le tableau 2-3 définit quelques constantes utiles à connaître. Si, par curiosité, vous voulez afficher l'ensemble des constantes existantes, vous pouvez écrire le code suivant :

```
<?php  
print_r(get_defined_constants());  
?>
```

Vous obtenez une liste impressionnante, dont un grand nombre des valeurs sont des entiers.

**Tableau 2-3 – Quelques constantes prédéfinies**

PHP_VERSION	Version de PHP installée sur le serveur
PHP_OS	Nom du système d'exploitation du serveur
DEFAULT_INCLUDE_PATH	Chemin d'accès aux fichiers par défaut
__FILE__	Nom du fichier en cours d'exécution
__LINE__	Numéro de la ligne en cours d'exécution

En complément, vous trouverez à la section consacrée aux fonctions mathématiques, ultérieurement dans ce chapitre, une liste de constantes mathématiques classiques.

## Les types de données

Dans PHP, il n'existe pas de déclaration explicite du type d'une variable lors de sa création. Même PHP 7 reste un langage pauvrement typé comparé à Java ou au C.

PHP permet la manipulation d'un certain nombre de types de données différents dans lequel on distingue :

- Les types scalaires de base :
  - Entiers, avec le type `integer`, qui permet de représenter les nombres entiers dans les bases 10, 8 et 16.
  - Flottants, avec le type `double` ou `float`, au choix, qui représentent les nombres réels, ou plutôt décimaux au sens mathématique.
  - Chaînes de caractères, avec le type `string`.
  - Booléens, avec le type `boolean`, qui contient les valeurs de vérité `TRUE` ou `FALSE` (soit les valeurs 1 ou 0 si on veut les afficher).
- Les types composés :
  - Tableaux, avec le type `array`, qui peut contenir plusieurs valeurs.

- Objets, avec le type `object`.
- Les types spéciaux :
  - Type `resource`.
  - Type `null`.

Vous verrez dans les sections suivantes de quelle manière vous pouvez définir des variables pour qu'elles aient un des types ci-dessus.

## Déterminer le type d'une variable

Avant de manipuler des variables, en utilisant, par exemple, des opérateurs, il peut être utile de connaître leur type. Cela permet de s'assurer que le résultat obtenu est conforme à ce qui est attendu et qu'il n'y a pas d'incompatibilité entre les types de ces variables. L'opérateur d'incrémentation appliqué à une chaîne, par exemple, peut donner des résultats curieux.

La principale fonction permettant de déterminer le type d'une valeur est `gettype()`, dont la syntaxe est la suivante :

```
| string gettype($mavar)
```

Elle retourne une chaîne de caractères contenant le type de la variable en clair.

Les fonctions suivantes permettent de vérifier si une variable est d'un type précis :

- `is_integer($var)` ou `is_int($var)`
- `is_double($var)`
- `is_string($var)`
- `is_bool($var)`
- `is_array($var)`
- `is_object($var)`
- `is_resource($var)`
- `is_null($var)`

Elles retournent la valeur booléenne `TRUE` si la variable est du type recherché et `FALSE` dans le cas contraire.

Vous pouvez savoir si une variable contient une valeur scalaire en appelant la fonction `is_scalar($var)` et, plus précisément, si elle contient une valeur numérique de type `integer` ou `double` en appelant la fonction `is_numeric($var)`.

Dans le code suivant, la variable `$var` est incrémentée d'une unité uniquement si elle contient une valeur numérique (repère ①), et la variable `$var2` est concaténée avec la chaîne "à tous" uniquement si elle est de type `string` (repère ②) :

```

<?php
$var = 73;
if(is_int($var)) ←❶
{
$var++;
echo "La variable vaut $var <br />";
}
// Affiche : La variable vaut 74
$var2="Bonjour ";
if(is_string($var2)) ←❷
{
$var2.=" à tous!";
echo $var2;
}
// Affiche : Bonjour à tous !
?>

```

## ***La conversion de type***

Malgré la grande souplesse, ou le grand laxisme, selon les opinions, de PHP à l'égard des types des variables, il peut être indispensable de convertir explicitement une variable d'un type dans un autre. C'est particulièrement vrai pour les variables issues d'un formulaire, ce dernier étant l'outil essentiel de communication du poste client au serveur. Ces variables sont toujours de type `string`.

Pour convertir une variable d'un type dans un autre, utilisez la syntaxe suivante :

```
| $result = (type_désiré) $mavar;
```

Si vous créez bien de la sorte une nouvelle variable du type désiré à partir de la première variable, celle-ci conserve son type initial. Si vous n'avez pas de raison de craindre de perdre la valeur initiale, il vous suffit de donner le même nom aux deux variables.

Dans l'exemple suivant, vous transformez une chaîne de caractères successivement en nombre décimal puis en entier et enfin en booléen :

```

<?php
$var="3.52 kilomètres";
$var2 = (double) $var;
echo "\$var2= ",$var2,"<br />";//affiche "$var2=3.52"
$var3 = (integer) $var2;
echo "\$var3= ",$var3,"<br />";//affiche "$var3=3"
$var4 = (boolean) $var3;
echo "\$var4= ",$var4,"<br />";//affiche "$var4=1" soit la valeur true
?>

```

Vous avez également la possibilité de modifier le type de la variable elle-même au moyen de la fonction `settype()`, dont la syntaxe est la suivante :

```
| boolean settype($var,"type _désiré")
```

Elle retourne la valeur `TRUE` si l'opération est réalisée et `FALSE` dans le cas contraire. Avec cette fonction, le code précédent devient :

```
<?php
$var="3.52 kilomètres";
settype($var,"double");
echo "\$var= ",$var,"<br />";//affiche "$var=3.52"
settype($var,"integer");
echo "\$var= ",$var,"<br />";//affiche "$var=3"
settype($var,"boolean");
echo "\$var= ",$var,"<br />";//affiche "$var=1" soit la valeur true
?>
```

## Contrôler l'état d'une variable

Lors de l'envoi de données d'un formulaire vers le serveur, le script qui reçoit les informations doit pouvoir détecter l'existence d'une réponse dans les champs du formulaire. Les fonctions `isset()` et `empty()` permettent ce type de contrôle.

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

```
boolean isset($var)
```

retourne la valeur `FALSE` si la variable `$var` existe mais n'est pas initialisée ou si elle a la valeur `NULL`, et la valeur `TRUE` si elle a une valeur quelconque. Il peut y avoir plusieurs paramètres dans cette fonction mais ils doivent tous répondre à cette même condition pour que la fonction retourne la valeur `TRUE`.

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

```
boolean empty(expression)
```

retourne la valeur `TRUE` si l'expression passée en paramètre n'est pas initialisée, a une des valeurs suivantes : `0`, `NULL`, `FALSE`, la chaîne "`0`", ou est un tableau vide, et la valeur `FALSE` si elle a une quelconque autre valeur.

Notez que l'expression peut être quelconque, comme une fonction par exemple.

L'exemple suivant illustre les différences subtiles entre ces deux fonctions :

```
<?php
$a=null;
if(isset($a)){echo "\$a existe déjà<br />";}
else {echo "\$a n'existe pas<br />";}
if(empty($a)){echo "\$a est vide <br />";}
else {echo "\$a a la valeur \$a<br />";}
//Affiche "$a n'existe pas" et "$a est vide"
$b=0;
if(isset($b)){echo "\$b existe déjà<br />";}
else {echo "\$b n'existe pas<br />";}
if(empty($b)){echo "\$b est vide <br />";}
else {echo "\$b a la valeur \$b<br />";}
//Affiche "$b existe déjà" et "$b est vide"
$c=1;
if(isset($c)){echo "\$c existe déjà<br />";}
else {echo "\$c n'existe pas<br />";}
if(empty($c)){echo "\$c est vide <br />";}
else {echo "\$c a la valeur \$c<br />";}
//Affiche "$c existe déjà" et "$c a la valeur 1"
?>
```

Pour la variable `$a` qui a la valeur `NULL`, `isset()` retourne également `FALSE` et `empty()` `TRUE`. Pour `$b`, qui a la valeur `0`, `isset()` permet de détecter l'existence de cette variable bien que `empty()` la déclare vide. Il en irait de même si `$b` était une chaîne vide.

Pour une valeur numérique affectée à la variable `$c`, les deux fonctions retournent `TRUE`. Ces fonctions, et en particulier `isset()`, vous permettront de vérifier si un utilisateur a bien rempli tous les champs d'un formulaire (voir le chapitre 6).

## Les entiers

Le type `integer` est affecté aux variables qui contiennent des valeurs entières positives ou négatives en base 10 (décimal), en base 2 (binaire), en base 8 (octal) ou en base 16 (hexadécimal).

Les entiers sont codés sur 32 bits sur la plupart des plates-formes, mais cela peut varier en fonction des serveurs. L'intervalle de valeur des entiers est donc dans ce cas de - 2 147 483 648, soit  $-2^{31}$  à + 2 147 483 647, soit  $2^{31} - 1$  en base 10.

Si une opération sur une variable de type `integer` l'amène à contenir une valeur en dehors de cet intervalle, elle est automatiquement convertie en type `double` et conserve sa nouvelle valeur.

Les nombres en base 10 s'écrivent de la manière, que chacun connaît :

```
$varint = 1789;  
$varint = -758;
```

Les nombres en base 2 doivent commencer par les caractères `0b`, précédés éventuellement d'un signe et suivis de un ou plusieurs chiffres strictement inférieurs à 2 :

```
$varbin = 0b1101;  
echo $varbin; // va afficher 13
```

soit la valeur  $1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$ , donc 13 en décimal.

Les nombres en base 8 doivent commencer par le chiffre 0, précédé éventuellement d'un signe et suivi de un ou plusieurs chiffres strictement inférieurs à 8 :

```
$varoct = 03267;  
echo $varoct; // va afficher 1719
```

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 ou binaire et que l'utilisation de l'instruction `echo $varoct`, par exemple, 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.

La fonction `bin2hex(string)` transforme la chaîne en sa représentation hexadécimale et la fonction `hex2bin(string)` effectue l'opération inverse. Par exemple :

```
echo bin2hex('PHP et MySQL'); //Affiche 504850206574204D7953514C  
echo hex2bin(504850206574204D7953514C); //Affiche 'PHP et MySQL'
```

Veillez absolument à ce que le nombre de caractères passés à la fonction `hex2bin()` soit pair.

## 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 × 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
**	Puissance (associatif à droite)  \$a=3; echo \$a**2; //Affiche 9 echo \$a**2**4; //Affiche 43046721 soit 3**2**4 ou 316
/	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 = 10.5; echo \$var%3.5; //affiche 1 et non pas 0.
--	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 préincrémantation, qui ajoute 1 avant d'utiliser la variable, et la postincrémantation, 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.

**Tableau 2-5 – Les fonctions mathématiques**

double/integer <code>abs (double/ integer X)</code>	Valeur absolue de X :  echo <code>abs(-543); // affiche 543.</code>
double <code>acos (double X)</code>	Arc cosinus de X, qui doit être compris entre - 1 et + 1. Le résultat est en radians :  echo <code>acos(0.5); // affiche 1.0471975511966.</code>
double <code>acosh (double X)</code>	Arc cosinus hyperbolique de X. Ne fonctionne pas sous Windows.
double <code>asin (double X)</code>	Arc sinus de X, qui doit être compris entre - 1 et + 1. Le résultat est en radians :  echo <code>asin(0.5); // affiche 0.5235987755983.</code>
double <code>asinh (double X)</code>	Arc sinus hyperbolique de X. Ne fonctionne pas sous Windows.
double <code>atan (double X)</code>	Arc tangente de X. Le résultat est en radians :  echo <code>atan(5); // affiche 0.46364760900081.</code>
double <code>atan2 (double Y, double X )</code>	Arc tangente du rapport Y/X. Le résultat est en radians. Il faut que Y soit différent de 0.
double <code>atanh (double X)</code>	Arc tangente hyperbolique de X.
string <code>base_convert (string N, integer B1, integer B2)</code>	Convertit le nombre N contenu dans une chaîne de la base B1 dans la base B2.
integer <code>bindec (string X)</code>	Convertit un nombre binaire X contenu dans une chaîne en base 10.
double <code>ceil (double X)</code>	Retourne l'entier immédiatement supérieur à X.
double <code>cos (double X)</code>	Cosinus de X qui doit être exprimé en radians.
double <code>cosh (double X)</code>	Cosinus hyperbolique de X.
string <code>decbin (integer X)</code>	Convertit X de la base 10 en binaire.
string <code>dechex (integer X)</code>	Convertit X de la base 10 en hexadécimal.
string <code>decoct (integer X)</code>	Convertit X de la base 10 en octal.
double <code>deg2rad (double X)</code>	Convertit X de degrés en radians.
double <code>exp (double X)</code>	Exponentielle de X, soit $e^x$ .
double <code>expm1 (double X)</code>	Retourne l'exponentielle de X - 1, soit $e^x - 1$ .
double <code>floor (double X)</code>	Retourne la partie entière de X, soit l'entier immédiatement inférieur à X.
double <code>fmod (double X, double Y)</code>	Retourne le reste de la division de Y par X pour des opérandes de type double.
integer <code>getrandmax (void)</code>	Indique la valeur maximale renournée par la fonction <code>rand()</code> .
integer <code>hexdec (string CH)</code>	Convertit la chaîne hexadécimale CH en décimal.

double hypot (double X, double Y)	Retourne la valeur de l'hypoténuse d'un triangle rectangle dont les côtés de l'angle droit sont X et Y, donc la valeur de la racine carrée de ( $X^2 + Y^2$ ).
integer intdiv(int \$a,int \$b)	Retourne le quotient de la division entière de \$a par \$b.
boolean is_finite ( double X)	Retourne TRUE si la valeur X est finie, c'est-à-dire dans l'intervalle des valeurs admises pour un double, et FALSE dans le cas contraire.
boolean is_infinite ( double X)	Retourne TRUE si la valeur X est supérieure à la valeur maximale admise pour un double, et FALSE dans le cas contraire.
boolean is_nan (double X)	Retourne TRUE si la valeur X n'est pas un nombre, et FALSE dans le cas contraire.
double lcg_value (void)	Retourne un nombre aléatoire compris entre 0 et 1.
double log (double X, double B)	Logarithme népérien (de base e) du nombre X.
double log10 (double X)	Logarithme décimal (de base 10) de X.
double log1p (double X)	Logarithme népérien de (1 + X).
double/integer max (double/ integer X, double/integer Y)	Retourne la valeur maximale de X et de Y.
double/integer min (double/ integer X, double/integer Y)	Retourne la valeur minimale de X et de Y.
integer mt_getrandmax (void)	Retourne la plus grande valeur aléatoire que peut retourner la fonction mt_rand().
integer mt_rand ( integer Min, integer Max)	Génère un résultat compris entre Min et Max ou entre 0 et la constante RAND_MAX si vous omettez les paramètres.
void mt_srand ( integer N)	Initialise le générateur de nombres aléatoires pour la fonction mt_rand(). Le paramètre N est un entier quelconque.
integer octdec (string CH)	Convertit un nombre octal contenu dans la chaîne CH en base 10.
double pi (void)	Retourne la valeur de pi.
double/integer pow (double/ integer X, double/integer Y)	Calcule X à la puissance Y. Les paramètres peuvent être entiers ou décimaux.
double rad2deg (double X)	Convertit X de radians en degrés.
integer rand (integer Min, integer Max)	Retourne un nombre aléatoire compris entre Min et Max y compris les bornes.
integer random_int(int Min,int Max)	Retourne un entier aléatoire sécurisé compris entre Min et Max qui peut être négatif.
double round (double X, integer N)	Arrondit X avec N décimales.
double sin (double X)	Sinus de X exprimé en radians.
double sinh (double X)	Sinus hyperbolique de X.

double sqrt (double x)	Racine carrée de X (qui doit être positif).
void srand (integer N)	Initialise le générateur de nombres aléatoires de la fonction <code>rand()</code> . Le paramètre <i>N</i> est un entier quelconque.
double tan (double x)	Tangente de X qui doit être en radians.
double tanh (double x)	Tangente hyperbolique de X.

## 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 "\$b 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'existe 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.

Vous pouvez donc écrire :

```
| $a=15;  
| if($a) {echo "$a existe et vaut $a";}
```

L'expression entre parenthèses qui ne contient que la variable `$a` est évaluée à `TRUE` car la variable `$a` existe et a une valeur non nulle. Dans le contexte d'évaluation booléenne de l'instruction `if` la valeur de `$a` n'a pas d'importance.

Vous pouvez traduire `if($a)` par « si `$a` existe et a une valeur », ce qui est vrai dans l'exemple ci-dessus. Chaque expression simple ou complexe peut donc être évaluée par une valeur booléenne en dehors de sa valeur propre, numérique ou autre.

## Évaluation booléenne des expressions

Le tableau 2-6 indique la manière dont sont évaluées les expressions PHP dans un contexte booléen. Il est important de bien connaître ces règles.

**Tableau 2-6 – Règles d'évaluation booléenne des expressions**

Expressions évaluées à <code>FALSE</code>	<ul style="list-style-type: none"><li>– Le mot-clé <code>FALSE</code></li><li>– La valeur entière <code>0</code> de type <code>integer</code></li><li>– La valeur décimale <code>0.0</code> de type <code>double</code></li><li>– La chaîne <code>"0"</code> de type <code>string</code></li><li>– Une variable de type <code>NULL</code></li><li>– Une variable non initialisée</li><li>– Un tableau vide</li><li>– Un objet sans propriété ni méthode</li><li>– Une expression logique fausse utilisant un ou plusieurs opérateurs</li></ul>
Expressions évaluées à <code>TRUE</code>	Toutes les autres possibilités, y compris l'entier <code>-1</code> , car il est non nul, et la chaîne <code>"false"</code> , car elle est non vide. Les variables de type <code>resource</code> sont également évaluées à <code>TRUE</code> .

## *Les opérateurs booléens*

Quand ils sont associés, les opérateurs booléens servent à écrire des expressions simples ou complexes, qui sont évaluées par une valeur booléenne `TRUE` ou `FALSE`.

Typiquement utilisés dans les instructions conditionnelles (voir le chapitre 3), ils se décomposent en deux catégories : les opérateurs de comparaison (voir tableau 2-7), qui testent, par exemple, l'égalité de deux valeurs, et les opérateurs logiques proprement dits, qui servent à écrire des expressions composées (voir tableau 2-8).

Il est important de bien manipuler ces opérateurs car ils sont à la base de l'élaboration des expressions conditionnelles complexes. En règle générale, les opérandes de ces opérateurs sont des expressions plus ou moins complexes.

**Tableau 2-7 – Les opérateurs de comparaison**

Opérateur	Description
<code>==</code>	<p>Teste l'égalité de deux valeurs. L'expression <code>\$a == \$b</code> vaut <code>TRUE</code> si la valeur de <code>\$a</code> est égale à celle de <code>\$b</code> et <code>FALSE</code> dans le cas contraire :</p> <pre>\$a = 345; \$b = "345"; \$c = (\$a==\$b);</pre> <p><code>\$c</code> est un booléen qui vaut <code>TRUE</code> car dans un contexte de comparaison numérique, la chaîne <code>"345"</code> est évaluée comme le nombre 345. Si <code>\$b="345 éléphants"</code> nous obtenons le même résultat.</p>
<code>!= ou &lt;&gt;</code>	<p>Teste l'inégalité de deux valeurs. L'expression <code>\$a != \$b</code> vaut <code>TRUE</code> si la valeur de <code>\$a</code> est différente de celle de <code>\$b</code> et <code>FALSE</code> dans le cas contraire.</p>
<code>===</code>	<p>Teste l'identité des valeurs et des types de deux expressions. L'expression <code>\$a === \$b</code> vaut <code>TRUE</code> si la valeur de <code>\$a</code> est égale à celle de <code>\$b</code> et que <code>\$a</code> et <code>\$b</code> sont du même type. Elle vaut <code>FALSE</code> dans le cas contraire :</p> <pre>\$a = 345; \$b = "345"; \$c = (\$a === \$b);</pre> <p><code>\$c</code> est un booléen qui vaut <code>FALSE</code> car si les valeurs sont égales, les types sont différents (integer et string).</p>
<code>!==</code>	<p>Teste la non-identité de deux expressions. L'expression <code>\$a !== \$b</code> vaut <code>TRUE</code> si la valeur de <code>\$a</code> est différente de celle de <code>\$b</code> ou si <code>\$a</code> et <code>\$b</code> sont d'un type différent. Dans le cas contraire, elle vaut <code>FALSE</code> :</p> <pre>\$a = 345; \$b = "345"; \$c = (\$a !== \$b);</pre> <p><code>\$c</code> est un booléen qui vaut <code>TRUE</code> car si les valeurs sont égales, les types sont différents (integer et string).</p>
<code>&lt;</code>	Teste si le premier opérande est strictement inférieur au second.
<code>&lt;=</code>	Teste si le premier opérande est inférieur ou égal au second.
<code>&gt;</code>	Teste si le premier opérande est strictement supérieur au second.
<code>&gt;=</code>	Teste si le premier opérande est supérieur ou égal au second.
<code>&lt;=&gt;</code>	Avec <code>\$a&lt;=&gt;\$b</code> , retourne <code>-1</code> , <code>0</code> ou <code>1</code> respectivement si <code>\$a &lt; \$b</code> , <code>\$a == \$b</code> ou <code>\$a &gt; \$b</code> ( <code>\$a</code> et <code>\$b</code> peuvent être des chaînes).

**Tableau 2-8 – Les opérateurs logiques**

Opérateur	Description
OR	<p>Teste si l'un au moins des opérandes a la valeur <code>TRUE</code> :</p> <pre>\$a = true; \$b = false; \$c = false; \$d = (\$a OR \$b); // \$d vaut TRUE.</pre>

	<pre>\$e = (\$b OR \$c); // \$e vaut FALSE.</pre>
	Équivaut à l'opérateur OR mais n'a pas la même priorité.
XOR	Teste si un et un seul des opérandes a la valeur TRUE :  <pre>\$a = true; \$b = true; \$c = false; \$d = (\$a XOR \$b); // \$d vaut FALSE. \$e = (\$b XOR \$c); // \$e vaut TRUE.</pre>
AND	Teste si les deux opérandes valent TRUE en même temps :  <pre>\$a = true; \$b = true; \$c = false; \$d = (\$a AND \$b); // \$d vaut TRUE. \$e = (\$b AND \$c); // \$e vaut FALSE.</pre>
&&	Équivaut à l'opérateur AND mais n'a pas la même priorité.
!	Opérateur unaire de négation, qui inverse la valeur de l'opérande :  <pre>\$a = TRUE; \$b = FALSE; \$d = !\$a; // \$d vaut FALSE. \$e = !\$b; // \$e vaut TRUE.</pre>

## 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 = 'PHP7 et MySQL';
$b = "PHP7 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.
\x[0-9 A-F a-f] {1,2}	Séquence de caractères désignant un nombre hexadécimal (de 1 à 2 caractères 0 à 9 et A à F ou a à f) et affichant le caractère correspondant :

```
echo "\x4D\x79\x53\x51\x4C"; // Affiche MySQL
```

## Concaténer des chaînes

L'opérateur PHP de concaténation est le point (.), qui fusionne deux chaînes littérales ou contenues dans des variables en une seule chaîne.

Le code suivant :

```
$a = "PHP";
$b = "MySQL";
$c = "Utilisez ".$a." et ".$b." pour construire un site dynamique";
echo $c;
```

affiche :

```
Utilisez PHP et MySQL pour construire un site dynamique
```

Lors de l'affichage avec l'instruction echo, cette concaténation peut être simulée en séparant chaque chaîne ou variable par une virgule.

L'exemple suivant :

```
echo "Utilisez ",$a," et ",$b, " pour construire un site dynamique";
```

affiche le même résultat, mais aucune chaîne ne contient l'ensemble du texte.

De nombreuses fonctions permettent d'effectuer toutes sortes de manipulations sur les chaînes de caractères. Le chapitre 4 leur est entièrement consacré.

## Les tableaux

Les tableaux représentent un type composé car ils permettent de stocker sous un même nom de variable plusieurs valeurs indépendantes d'un des types de base que vous venez de voir. C'est comme un tiroir divisé en compartiments. Chaque compartiment, que nous nommerons un élément du tableau, est repéré par un indice numérique (le premier ayant par défaut la valeur 0 et non 1). D'où l'expression de tableau indicé.

Chaque élément peut aussi être identifié par une étiquette, qui est une chaîne de caractères ou une variable de type string, nommée clé, associée à l'élément du tableau. Ce type de tableau est appelé tableau associatif.

Les éléments de ces tableaux peuvent être de type integer, double, boolean, string ou même array, ce qui permet de créer des tableaux de tableaux, c'est-à-dire des tableaux multidimensionnels, ce que PHP ne permet pas explicitement, contrairement à d'autres langages.

Les éléments d'un tableau pourraient aussi être des types object ou resource, qui sont présentés dans les sections suivantes.

D'une manière primaire, vous définissez la valeur d'un élément de tableau indicé à l'aide de la syntaxe à crochets [], avec un nom de variable, suivi des crochets, qui

contiennent l'indice ou la variable de type `integer` :

```
$tab[0] = 2004;
$tab[1] = 31.14E7;
$tab[2] = "PHP7";
$tab[35] = $tab[2]. "et MySQL";
$tab[] = TRUE; //voir les paragraphes suivants
$ind = 40;
$tab[$ind] = "Dernier élément";
echo "Nombre d'éléments = ", count($tab);
```

La variable `$tab` est un tableau par le simple fait que son nom est suivi de crochets et d'un indice. Il contient maintenant six éléments de types variés.

Les trois premiers éléments sont affectés en utilisant des indices incrémentés d'une unité. Pour le quatrième élément, l'indice utilisé ne succède pas aux précédents. Cela implique que les éléments d'indice 3 à 34 sont non seulement vides mais n'existent pas. En effet, la fonction `count($tab)`, qui retourne le nombre d'éléments du tableau qui lui est passé en paramètre, retourne ici la valeur 6 (en réalité, ces éléments sont de type `NULL`).

L'élément suivant est affecté sans qu'aucun indice soit précisé. Dans ce cas, il a automatiquement l'indice suivant celui de l'élément précédemment affecté, soit ici l'indice 36. L'avantage de cette syntaxe est de permettre d'ajouter un nouvel élément à la fin d'un tableau sans connaître la valeur du premier indice disponible.

Le dernier élément est créé en lui donnant comme indice la valeur d'une variable de type `integer` qui est de 40.

Pour lire la valeur d'un élément de tableau dans un script, il suffit d'utiliser la même syntaxe en précisant l'indice de la valeur désirée.

L'exemple suivant :

```
echo "<p> Le langage préféré de l'open source est $tab[2] <br />";
echo " Utilisez $tab[35] </p>";
```

affiche :

---

```
Le langage préféré de l'open source est PHP7
Utilisez PHP7 et MySQL
```

---

Si une variable dynamique est un tableau, il y a une ambiguïté qu'il faut lever selon ce qu'on désire obtenir. Par exemple, si nous avons :

```
$b=['PHP', 'MySQL']; // $b n'a pas de sens et $$b[1] non plus
```

Par contre, si on a :

```
$MySQL="BASE";
echo ${$b[1]}; // est la variable $MySQL qui contient "BASE"
echo ${$b[1]}[1] //Affiche $MySQL[1] soit le caractère "A"
```

La syntaxe permettant de définir les éléments de tableaux associatifs est similaire, mais vous remplacez l'indice numérique par une chaîne de caractères quelconques ou par une variable ou une constante de type `string`. Il ne faut donc pas oublier d'inclure cette chaîne dans des apostrophes ou des guillemets, faute de quoi vous vous exposez à

quelques problèmes dans des cas particuliers.

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. Crédation de tableaux associatifs

```
<?php
$tab2["zéro"] = 2003;
$tab2['un'] = 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
```

---

Remarquez que la dernière ligne ne correspond pas du tout à vos attentes, pas plus d'ailleurs qu'à la réalité.

## Attention

Les clés des tableaux associatifs étant sensibles à la casse,

\$tab["cle"]  
est différent de  
\$tab["CLE"]

De plus, les chaînes définissant les clés ne doivent pas comporter d'espaces.

L'exemple 2-3 crée dynamiquement une liste de liens à partir des valeurs des éléments d'un tableau associatif. Dans la pratique, les valeurs des éléments du tableau devraient provenir d'une base de données pour que la page soit réellement dynamique. Les liens sont affichés dans un liste à puces créée avec les balises HTML `<ul>` et `<li>`, auxquelles est appliqué un style CSS défini dans l'en-tête du document.

## Exemple 2-3. Utilisation des tableaux

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Les tableaux</title>
<style type="text/css">
ul {list-style-image:url("etoile.gif");}
</style>
</head>
<body>
<?php
//création des éléments du tableau
$tab["php"] = "php.net";
$tab["mysql"] = "mysql.com";
$tab["xhtml"] = "w3.org";
//création des liens
echo"<h2> Mes liens préférés </h2>";
echo "<ul><li><a href=\" http://www.{$tab['php']} \" title=\"Le site php.net\">
    &nbsp; PHP </a> </li>";
echo "<li><a href=\" http://www.{$tab['mysql']} \" title=\"Le site mysql.com\">
    &nbsp; MySQL </a> </li>";
echo "<li><a href=\" http://www.{$tab['xhtml']} \" title=\"Le site du W3C\">
    &nbsp; HTML </a> </li></ul>";
?>
</body>
</html>
```

Le script affiche le résultat illustré à la figure 2-1.



**Figure 2-1**  
*Création dynamique de liens*

Comme les chaînes, les tableaux offrent de nombreuses possibilités de manipulation des données. PHP fournit en standard un grand nombre de fonctions spécialisées permettant d'améliorer cette gestion. Elles seront abordées en détail au chapitre 5.

## Les objets

PHP permet l'utilisation des classes et utilise le type `object` pour toute variable créée en tant qu'instance d'une classe. Nous reviendrons plus en détail sur les notions de classe et d'objet au chapitre 9.

La version 7 de PHP offre un éventail beaucoup plus large et rigoureux que PHP 4 de possibilités de programmation objet.

Le script suivant :

```
<?php
class myclass{
// Définition de la classe (ici elle est vide)
}
$varcl = new myclass; ←①
echo "Le type de la variable \$varcl est :", gettype($varcl); ←②
?>
```

crée une classe nommée `myclass` puis une variable `$varcl` à l'aide de l'opérateur particulier `new` (repère ①).

La ligne suivante (repère ②) affiche :

---

Le type de la variable \$varcl est : object

---

Vous venez de créer une variable d'un type nouveau.

## 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 renvoyée par cette fonction. Il affiche ensuite la valeur puis le type de cette variable.

```
<?php
//*****Le type resource*****
$connect = mysql_connect("localhost","root","");
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 1) 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; ← 1
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

divers constant (string \$nom )

Retourne la valeur de la constante dont le nom est contenu dans \$nom.

```
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(expression)
```

Retourne TRUE si l'expression 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.

```
boolean is_array($var)
boolean is_bool($var)
boolean is_double($var)
boolean is_integer($var)
boolean is_null($var)
boolean is_object($var)
```

```
boolean is_resource($var)
boolean is_string($var)
```

Ces fonctions retournent `TRUE` si la variable est du type testé et `FALSE` dans le cas contraire.

## Exercices

### Exercice 1

Parmi les variables suivantes, lesquelles ont un nom valide : `mavar`, `$mavar`, `$var5`, `$_mavar`, `$_5var`, `$_élément1`, `$hotel4*` ?

### Exercice 2

Donnez les valeurs de `$x`, `$y`, `$z` à la fin du script suivant :

```
$x="PostgreSQL";
$y="MySQL";
$z=&$x;
$x="PHP 5";
$y=&$x;
```

### Exercice 3

Lisez les valeurs des variables du script de l'exercice 2 à l'aide du tableau `$GLOBALS`.

### Exercice 4

Déterminez le numéro de version de PHP, le nom du système d'exploitation de votre serveur ainsi que la langue du navigateur du poste client.

### Exercice 5

Donnez la valeur de chacune des variables pendant et à la fin du script suivant, et vérifiez l'évolution du type de ces variables :

```
$x="PHP7";
$a[]=&$x;
$y=" 7e version de PHP";
$z=$y*10;
$x.=$y;
$y*=$z;
$a[0]="MySQL";
```

### Exercice 6

Donnez la valeur des variables `$x`, `$y`, `$z` à la fin du script :

```
$x="7 personnes";
$y=(integer) $x;
$x="9E3";
$z=(double) $x;
```

### Exercice 7

Donnez la valeur booléenne des variables `$a`, `$b`, `$c`, `$d`, `$e` et `$f` :

```
$a="0";
```

```
$b="TRUE";
$c=FALSE;
$d=($a OR $b);
$e=($a AND $c);
$f=($a XOR $b);
```

## Les instructions de contrôle

On retrouve dans PHP la plupart des instructions de contrôle des scripts. Indispensables à la gestion du déroulement d'un algorithme quelconque, ces instructions sont présentes dans tous les langages. PHP utilise une syntaxe très proche de celle du langage C.

Ceux qui ont déjà pratiqué un langage tel que le C ou plus simplement JavaScript seront en pays de connaissance. Pour les autres, une adaptation sera sans doute nécessaire. La version 7 de PHP a vu l'apparition de nouvelles instructions dédiées à la gestion des exceptions, comme `try...catch...finally` ou `throw`, qui lui faisaient défaut jusqu'à présent. La version 5.4 avait même vu le retour de l'ancestrale instruction `goto`, très controversée dans tous les langages.

### Les instructions conditionnelles

Comme tout langage, PHP dispose d'instructions conditionnelles qui permettent d'orienter le déroulement d'un script en fonction de la valeur de données.

#### *L'instruction if*

L'instruction `if` est la plus simple et la plus utilisée des instructions conditionnelles. Présente dans tous les langages de programmation, elle est essentielle en ce qu'elle permet d'orienter l'exécution du script en fonction de la valeur booléenne d'une expression.

Sa syntaxe est la suivante :

```
| if (expression) instruction;
```

Si l'expression incluse dans les parenthèses est évaluée à la valeur booléenne `TRUE`, l'instruction qui suit est exécutée. Dans le cas contraire, l'exécution passe directement à la ligne suivante.

L'instruction `if` peut être suivie d'un bloc d'instructions délimité par des parenthèses qui sera entièrement exécuté dans les mêmes conditions :

```
| if(expression)
```

```
{  
//bloc de code  
}
```

La rédaction de l'expression est importante. Elle peut devenir complexe lorsqu'elle comprend des opérateurs logiques associant ses différents composants.

Dans le code suivant :

```
<?php  
$a=6;  
if(is_integer($a) && ($a<10 && $a>5) && ($a%2==0) ) {echo "Conditions satisfaites";}  
?>
```

l'expression composée :

```
(is_integer($a) && ($a<10 && $a>5) && ($a%2==0))
```

est évaluée à `TRUE` si `$a` répond simultanément aux trois conditions suivantes : être un entier, être compris entre 5 et 10 et être divisible par 2, soit pour `$a` les valeurs possibles de 6 et 8 uniquement. Le message ne s'affiche donc que dans ces cas.

PHP réalise une évaluation booléenne d'un grand nombre d'expressions qui ne contiennent pas en elles-mêmes de variables booléennes. Il admet, par exemple, des expressions du genre :

```
$a = 25;  
if($a) {echo "La condition est vraie <br />";}
```

Dans ce cas, ce n'est pas la valeur de la variable `$a` qui est prise en compte mais son évaluation booléenne, qui vaut `TRUE`. Nous avons déjà abordé ce point au chapitre 2. Le lecteur pourra se reporter au tableau 2-6 pour revoir les conditions d'évaluation dans les différents cas.

### Ne pas se tromper d'opérateur

Une erreur courante consiste à confondre l'opérateur de comparaison `==` avec l'opérateur d'affectation `=`. Dans les expressions conditionnelles, pour tester l'égalité de deux valeurs il faut employer l'opérateur `==` ou encore `===` pour tester l'identité (même valeur et même type).

## *L'instruction if...else*

L'instruction `if...else` permet de traiter le cas où l'expression conditionnelle est vraie et en même temps d'écrire un traitement de rechange quand elle est évaluée à `FALSE`, ce que ne permet pas une instruction `if` seule. L'instruction ou le bloc qui suit `else` est alors le seul à être exécuté. L'exécution continue ensuite normalement après le bloc `else`.

l'exemple 3-1 suivant calcule le prix net après une remise variable en fonction du montant des achats selon les critères suivants :

- Si le prix total est supérieur à 100 euros, la remise est de 10 %. Cette condition est traitée par l'instruction `if` (repère ①).

- Pour les montants inférieurs ou égaux à 100 euros, la remise est de 5 %. Cette condition est traitée par l'instruction `else` (repère ②).

### Exemple 3-1. L'instruction `if...else`

```
<?php
$prix=55;
if($prix>100) ← ①
{
    echo "<b>Pour un montant d'achat de $prix &euro;, la remise est de 10 % </b><br />";
    echo "Le prix net est de ",$prix*0.90;
}
else ← ②
{
    echo "<b>Pour un montant d'achat de $prix &euro;, la remise est de 5 %</b><br />";
    echo "<h3>Le prix net est de ",$prix*0.95,"</h3>";
}
?>
```

Compte tenu de la valeur attribuée ici à la variable `$a`, le script affiche le résultat suivant :

---

```
Pour un montant d'achat de 55 &euro;, la remise est de 5 %
Le prix net est de 52.25 &euro;
```

---

Le bloc qui suit les instructions `if` ou `else` peut contenir toutes sortes d'instructions, y compris d'autres instructions `if...else`. Nous obtenons dans ce cas une syntaxe plus complexe, de la forme :

```
if(expression1)
{ //Bloc 1}
elseif(expression2)
{ //Bloc 2}
else
{ //Bloc 3}
```

Cette construction s'interprète de la façon suivante : si l'expression 1 est évaluée à `TRUE`, le bloc 1 est exécuté ; dans le cas contraire, si l'expression 2 qui suit l'instruction `elseif` est évaluée à `TRUE`, le bloc 2 est exécuté. Dans les autres cas, c'est le bloc 3 qui est exécuté. Quelle que soit la situation, un seul bloc est exécuté.

Dans l'exemple 3-11, vous voulez afficher le montant d'une remise calculée selon les modalités suivantes :

- Si vous achetez un PC de plus de 1 000 euros, la remise est de 15 %.
- Pour un PC de 1 000 euros et moins, la remise est de 10 %.
- Pour les livres, la remise est de 5 %.
- Pour tous les autres articles, la remise est de 2 %.

La première instruction `if...elseif...else` (repères ①, ② et ③) contrôle la catégorie du produit. La deuxième instruction `if...else` détermine le montant de la remise si le produit

est un PC (repères ④ et ⑤). L'indentation du code permet une lecture plus facile. Faute d'une telle indentation, il est difficile de distinguer à quelle instruction `if` se rapporte une instruction `else`.

## Exemple 3-2. Les instructions `if` imbriquées

```
<?php
// ****if...elseif...else*****
$cat="PC";
$prix=900;
if($cat=="PC") ←①
{
    if($prix >= 1000) ←④
    {
        echo "<b>Pour l'achat d'un PC d'un montant de $prix &euro;, la remise est de
        ↪15 %</b><br />";
        echo "<h3> Le prix net est de : ",$prix*0.85, "&euro; </h3>";
    }
    else ←⑤
    {
        echo "<b>Pour l'achat d'un PC d'un montant de $prix &euro;, la remise est de
        ↪10 %</b><br />";
        echo "<h3> Le prix net est de : ",$prix*0.90, "&euro; </h3>";
    }
}
elseif($cat=="Livres") ←②
{
    echo "<b>Pour l'achat de livres la remise est de 5 %</ b><br />";
    echo "<h3> Le prix net est de : ",$prix*0.95, "&euro; </h3>";
}
else ←③
{
    echo"<b>Pour les autres achats la remise est de 2 %</ b><br />";
    echo "<h3> Le prix net est de : ",$prix*0.98, "&euro; </h3>";
}
?>
```

Le résultat de ce script pour `$cat` à la valeur "PC" et la variable `$prix` à la valeur 900 donne l'affichage suivant :

---

```
Pour l'achat d'un PC d'un montant de 900 €, la remise est de 10 %
Le prix net est de : 810€
```

---

## *Les opérateurs ? et ??*

L'opérateur `?` permet de remplacer avantageusement une instruction `if...else` en évaluant une expression et en attribuant à une variable une première valeur si la condition est vraie ou une autre valeur si elle est fausse.

Sa syntaxe est la suivante :

```
| $var = expression ? valeur1 : valeur2
```

Elle est équivalente à :

```
| if(expression) {$var=valeur1;}  
| else {$var=valeur2;}
```

Le premier exemple de calcul de remise pourrait s'écrire :

```
| $var = ($prix>100) ? "la remise est de 10 %":"la remise est de 5 %";  
| echo "<b>Pour un montant d'achat de $prix &euro;: $var </b><br />";
```

au lieu de :

```
| if($prix>100)  
{  
|   echo "<b>Pour un montant d'achat de $prix &euro;, la remise est de 10 %</b><br />";  
| }  
| else  
{  
|   echo "<b>Pour un montant d'achat de $prix &euro;, la remise est de 5 %</b><br />";  
| }
```

Cet opérateur est généralement employé avec des expressions booléennes courtes. L'exemple 3-3 adapte un texte en fonction de la valeur d'une variable, soit pour une formule de politesse en fonction du sexe du visiteur, soit pour mettre au pluriel un mot en fonction d'un nombre.

### Exemple 3-3. L'opérateur ?

```
<?php  
$ch = "Bonjour ";  
$sexe="M";  
$ch .= ($sexe=="F") ?"Madame":"Monsieur";  
echo "<h2>$ch</h2>";  
$nb = 3;  
$pmu ="Il faut trouver ".$nb;  
$mot = ($nb==1)?" cheval":" chevaux";  
echo "<h3> $pmu $mot </h3>";  
?>
```

Compte tenu des valeurs des variables \$sexe et \$nb le résultat retourné est le suivant :

---

```
Bonjour Monsieur  
Il faut trouver 3 chevaux
```

---

En complément de l'opérateur précédent, PHP 7 ajoute l'opérateur ?? qui retourne la valeur du premier opérande s'il existe et s'il n'a pas la valeur NULL ou celle du second opérande dans le cas contraire.

Sa syntaxe est :

```
| $var = $valeur1 ?? $valeur2 ;
```

Notez que le second opérande peut être une chaîne et constituer la valeur par défaut.

Ce code est équivalent à :

```
| $var = isset($valeur1) ? $valeur1 : $valeur2;
```

On peut également enchaîner plusieurs fois cet opérateur sous la forme :

```
| $var=$valeur1 ?? $valeur2 ?? $valeur3 ;
```

L'affectation se faisant dans ce cas avec la première variable existante et non nulle. Une utilisation typique est la gestion des saisies dans un formulaire (voir le chapitre 6), car elle permet de récupérer la valeur d'un champ ou de lui attribuer une valeur par défaut (voir aussi les variables globales `$_POST` ou `$_GET` au chapitre 6). Dans l'exemple ci-dessous :

```
| $nom=$_POST['nom'] ?? 'Anonyme' ;
```

si l'utilisateur saisit son nom dans le champ de formulaire nommé '`nom`', la variable `$nom` le contiendra, sinon elle aura la valeur '`Anonyme`'.

## ***L'instruction switch...case***

Supposez que vous vouliez associer un code de département avec son nom réel. Avec une suite d'instructions `if`, vous écririez le script suivant :

```
<?php  
$dept=75;  
if($dept==75) echo "Paris";  
if($dept==78) echo "Hauts de Seine";  
if($dept==91) echo "Yvelines";  
if($dept==93) echo "Seine Saint Denis";  
?>
```

dans lequel la variable `$dept` proviendrait d'un formulaire, par exemple.

Ce code peut être simplifié sans multiplier les instructions `if` grâce à l'instruction `switch...case`. Cette dernière permet de comparer la valeur d'une expression avec une liste de valeurs prédéterminées par le programmeur et d'orienter le script en fonction de la valeur de cette expression.

La syntaxe de cette instruction est la suivante :

```
switch(expression)  
{  
    case valeur1:  
        bloc d'instructions 1;  
        break;  
    case valeur2:  
        bloc d'instructions 2;  
        break;  
    .....  
    case valeurN:  
        bloc d'instructions N;  
        break;  
  
    default:  
        bloc d'instructions par défaut;  
        break;  
}
```

Si l'expression qui suit le mot-clé `switch` vaut `valeur1`, les instructions qui suivent la première instruction `case` sont exécutées, après quoi l'exécution passe à la fin du bloc `switch`. Il en va de même pour les valeurs suivantes. Si aucune concordance n'est

trouvée, ce sont les instructions qui suivent l'instruction `default` qui sont exécutées.

La présence de cette instruction n'est pas obligatoire, mais elle est conseillée pour faire face à toutes les éventualités, telles les erreurs de saisie, par exemple. Chaque groupe `case` doit se terminer par une instruction `break`, faute de quoi les autres blocs `case` sont aussi exécutés.

La valeur qui suit chaque instruction `case` peut être une constante littérale ou une constante nommée, déclarée précédemment à `switch` à l'aide du mot-clé `define`.

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;
}
```

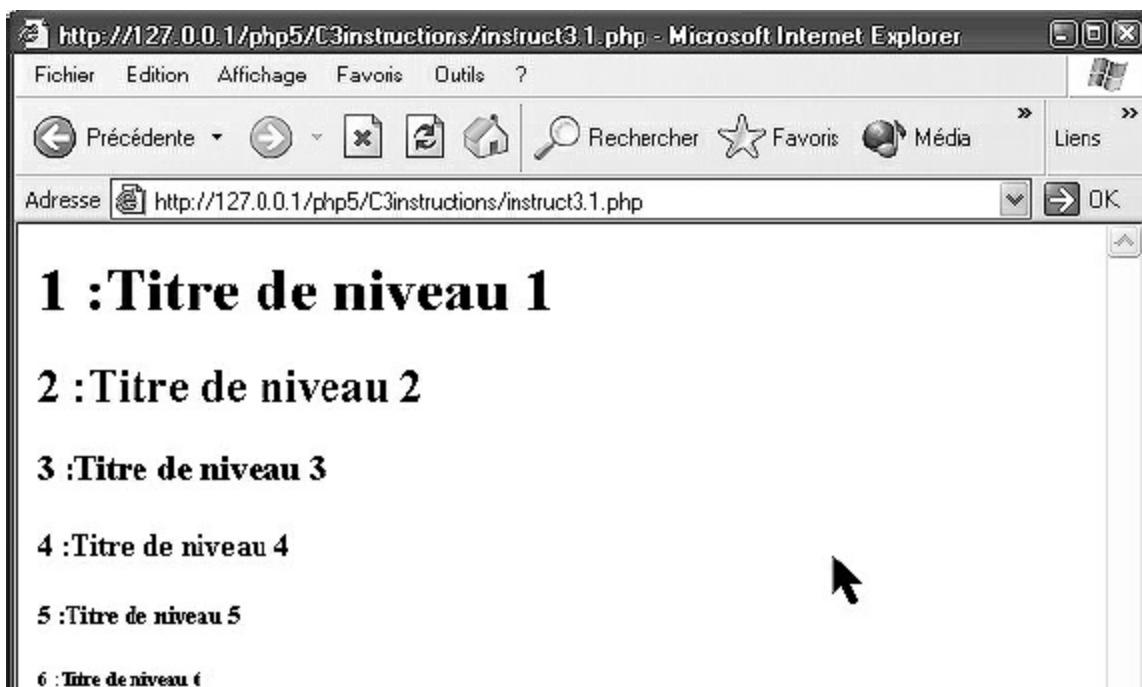
`expression1` est toujours évaluée. Il s'agit généralement de l'initialisation d'une ou plusieurs variables servant de compteur pour la boucle. `expression2` est ensuite évaluée avec une valeur booléenne : si elle vaut `TRUE`, la boucle continue et les instructions comprises dans le bloc sont exécutées, 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.



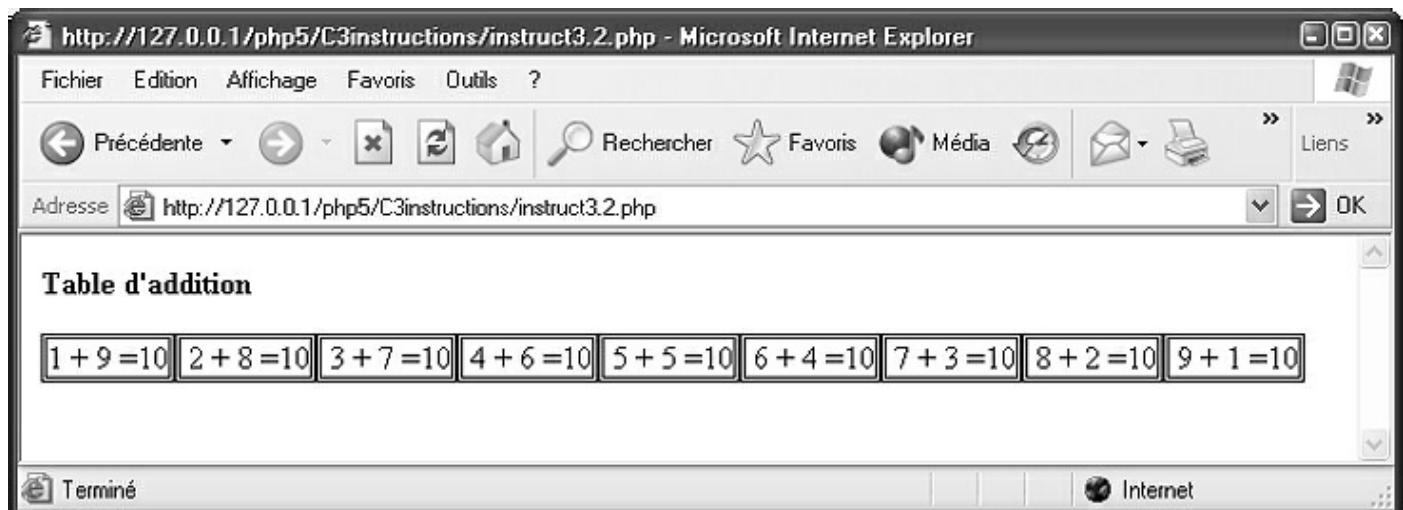
**Figure 3-1**  
*Création de titres*

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 HTML à deux dimensions, chaque dimension étant gérée par une variable compteur différente, `$i` et `$j`. La première boucle `for` (repère ①) 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 ②) itère les numéros de ligne avec la variable `$i`, et la deuxième (repère ③) le contenu des cellules du tableau avec la variable `$j`.

### Exemple 3-7. Les boucles `for` imbriquées

```

<?php
echo "<h2> Révisez votre table de multiplication!</ h2>";
//Début du tableau HTML
echo "<table border=\"2\" style=\"background-color:yellow\> <th>
    &nbsp;X &nbsp;</th>"; ←①
//Création de la première ligne
for($i=1;$i<10;$i++)
{
echo "<th>&nbsp;$i&nbsp;</th>"; ←②
}
//Fin de la boucle 1
//*****
//Création du corps de la table
//*****
//Boucles de création du contenu de la table
for($i=1;$i<10;$i++) ←③
{
    //Création de la première colonne
    echo "<tr><th>&nbsp;$i&nbsp;</th>"; ←④
    //Remplissage de la table
    for($j=1;$j<10;$j++) ←⑤
    {
        echo "<td style=\"background-color:red;color:white\> &nbsp;&nbsp; <b>". $i*$j.
            "&nbsp;&nbsp; </td>"; ←⑥
    }
    echo "</tr>"; ←⑦
}
echo "</table>"; ←⑧
?>

```

La figure 3-3 présente la table de multiplication créée par ce script.

The screenshot shows a web browser window with the URL `http://127.0.0.1/php5/C3instructions/instruct3.3.php`. The page title is "Révisez votre table de multiplication!". Below the title is a 9x9 multiplication table. The first row and column are highlighted with a yellow background. The first column is labeled with the numbers 1 through 9, and the first row is labeled with the numbers 1 through 9. All other cells in the table contain the product of the row and column indices, such as 2, 4, 6, 8, etc., up to 81.

X	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

**Figure 3-3**  
*Notre table de multiplication*

Vous trouverez de nombreux exemples d'utilisation de boucles `for` tout au long de cet ouvrage, notamment au chapitre 5 pour la lecture de l'ensemble des éléments d'un tableau.

## La boucle `while`

La boucle `for` oblige à préciser les valeurs limites pour lesquelles la boucle va s'arrêter. À moins d'utiliser une instruction `if` pour la stopper à l'aide de l'instruction `break` (voir le paragraphe « Sortie anticipée des boucles » dans les sections suivantes), il faut connaître ces valeurs limites. La boucle `while` permet d'affiner ce comportement en réalisant une action de manière répétitive tant qu'une condition est vérifiée ou qu'une expression quelconque est évaluée à `TRUE` et donc de l'arrêter quand elle n'est plus vérifiée (évaluée à `FALSE`).

La boucle `while` permet, par exemple, d'afficher tous les résultats fournis après interrogation d'une base de données, sans en connaître à l'avance le nombre exact. La syntaxe de cette instruction est la suivante :

```
while(expression)
{
    //Bloc d'instructions à répéter
}
```

L'expression précisée doit pouvoir être évaluée de façon booléenne par PHP et pouvoir changer de valeur au cours du script, faute de quoi la boucle serait infinie.

L'exemple suivant :

```
$a = "oui";
while ($a) {echo $a; }
```

constitue une boucle infinie, car `$a` est évaluée à `TRUE` en tant que chaîne non vide.

De même, dans le code suivant :

```
$a = 54;
while ($a>100) {echo $a; }
```

l'instruction `echo $a` n'est jamais exécutée car l'expression `$a>100` contenue dans `while` est toujours fausse.

L'exemple 3-8 ci-dessous effectue une suite de tirages de nombres aléatoires compris entre 1 et 100 grâce à la fonction `rand()` (repère ②), avec comme condition supplémentaire que le nombre tiré soit un multiple de 7 (repère ①). Le script affiche les nombres tirés jusqu'à trouver un multiple de 7.

### Exemple 3-8. Tirage d'un multiple de 7 avec une boucle `while`

```
<?php
$n=1;
while($n%7!=0 ) ←①
{
    $n = rand(1,100); ←②
```

```
echo $n,"nbsp; /";
}
?>
```

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 ①).

### Exemple 3-9. Tirage avec une boucle `do...while`

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

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  
| }
```

La variable `$valeur` contient successivement chacune des valeurs du tableau. Il importe cependant de ne pas utiliser un nom de variable existant, faute de quoi sa valeur est écrasée. Les variables utilisées dans une boucle `foreach` ne sont pas locales à la boucle et gardent donc la valeur du dernier élément lu dans tout le script.

Pour lire les valeurs et les clés (ou les indices), la deuxième syntaxe est la suivante :

```
| foreach($tableau as $cle=>$valeur)  
{  
//bloc utilisant la valeur et la clé de l'élément courant  
}
```

Ici, la variable `$cle` contient successivement l'indice ou la clé de chacun des éléments du tableau dans l'ordre numérique pour les tableaux indicés et dans l'ordre de création des éléments pour les tableaux associatifs. La variable `$valeur` contient la valeur associée à la clé ou à l'indice courant.

Dans ces deux cas, la variable `$tableau` peut être remplacée par une expression dont la valeur est du type `array`, comme ce pourrait être le cas, par exemple, d'une fonction retournant un tableau. Cette instruction est abondamment utilisée pour lire tous les résultats obtenus après interrogation d'une base de données.

L'exemple 3-10 crée d'abord un tableau indicé contenant les puissances de 2 à l'aide d'une simple boucle `for` (repère ①) puis lit l'intégralité des éléments à l'aide d'une boucle `foreach` (repère ②).

### Exemple 3-10. Lecture des valeurs d'un tableau indicé

```
<?php  
//Création du tableau de 9 éléments  
for($i=0;$i<=8;$i++)  
{  
    $tab[$i] = pow(2,$i); ←①  
}  
$val ="Une valeur";  
echo $val,"<br />";  
//Lecture des valeurs du tableau  
echo"Les puissances de 2 sont :";  
foreach($tab as $val) ←②  
{echo $val." : ";}  
?>
```

Le résultat suivant est affiché :

---

```
Les puissances de 2 sont :1 : 2 : 4 : 8 : 16 : 32 : 64 : 128 : 256:
```

---

L'exemple 3-11 lit les indices et les valeurs du même tableau en utilisant la deuxième syntaxe (de la forme `$tab as $ind=>$val`) pour récupérer les indices dans la variable `$ind` et les valeurs dans `$val` (repère ①). Il est possible de vérifier que les variables `$ind` et `$val` sont toujours visibles à l'extérieur de la boucle en affichant leurs valeurs

après la fin de la boucle `foreach` (repère ②).

### Exemple 3-11. Lecture des indices et des valeurs

```
<?php
//Création du tableau
for($i=0;$i<=8;$i++)
{
    $tab[$i] = pow(2,$i);
}
//Lecture des indices et des valeurs
foreach($tab as $ind=>$val) ← ①
{echo " 2 puissance $ind vaut $val <br />";}
echo "Dernier indice ",$ind, " ,dernière valeur ",$val; ← ②
?>
```

L'exemple affiche la liste suivante :

---

```
2 puissance 0 vaut 1
2 puissance 1 vaut 2
.....
2 puissance 7 vaut 128
2 puissance 8 vaut 256
Dernier indice 8, dernière valeur 256
```

---

L'exemple 3-12 crée un tableau associatif dont les clés sont des identifiants de clients et associe à chacun un code aléatoire compris entre 100 et 1 000 (repère ①) puis lit et affiche les clés et les valeurs du tableau (repère ②).

### Exemple 3-12. Lecture des clés et des valeurs

```
<?php
//Création d'un tableau associatif
for($i=0;$i<=8;$i++)
{
    $tabass["client".$i] = rand(100,1000); ← ①
}
//Lecture des clés et des valeurs
foreach($tabass as $cle=>$val) ← ②
{echo " Le client de login <b>$cle</b> a le code <b>$val</b><br />";}
?>
```

Les résultats suivants sont affichés :

---

```
Le client de login client0 a le code 638
Le client de login client1 a le code 569
.....
Le client de login client6 a le code 135
Le client de login client7 a le code 786
Le client de login client8 a le code 406
```

---

#### **foreach** et les objets

Depuis PHP 5, vous pouvez lire l'ensemble des noms et des valeurs des propriétés d'un

objet à l'aide d'une boucle `foreach` comme nous le faisons pour un tableau. Vous rencontrerez aux chapitres 9, 15 et 16 des illustrations de ce type de lecture.

## Sortie anticipée des boucles

Vous pouvez avoir besoin d'arrêter une boucle avant son terme normal. Pour cela, vous disposez des instructions `break` et `continue`, qui permettent de réaliser un arrêt partiel ou total.

### L'instruction `break`

Il est possible d'arrêter complètement une boucle `for`, `foreach` ou `while` avant son terme normal si une condition particulière est vérifiée, à l'aide de l'instruction `break`. Le script n'est pas arrêté, comme avec l'instruction `exit`, et seule la boucle en cours se termine.

Si plusieurs boucles sont imbriquées, seule celle qui contient l'instruction `break` se termine. Pour arrêter plusieurs boucles en même temps, on emploie la syntaxe suivante :

```
break n; // n doit être non nul
```

dans laquelle `n` désigne le nombre de boucles les plus internes que vous souhaitez fermer. Celles de niveau supérieur à `n` continuent à s'exécuter normalement.

L'exemple 3-13 crée un tableau de noms (repère ①), puis une boucle `for` lit le tableau (repère ②). Cette boucle contient une instruction `if` qui vérifie que le nom commence par la lettre `A` (repère ③). Si la condition est remplie, le script affiche le nom. La boucle est fermée grâce à une instruction `break` (repère ④).

### La notation `$tab[$i][0]`

La notation `$tab[$i][0]` permet de récupérer la première lettre de la chaîne de caractères contenue dans la variable `$tab[$i]`, `$tab[$i][1]` la deuxième lettre, et ainsi de suite.

### La fonction `count($tab)`

La fonction `count($tab)` retourne le nombre d'éléments du tableau `$tab`, ce qui permet de connaître le nombre maximal d'itération de la boucle. Cette expression pourrait être supprimée car c'est l'instruction `break` qui met fin à la boucle. Il est toutefois préférable de l'utiliser pour arrêter la boucle, car si le tableau ne contient aucun nom commençant par la lettre `A`, vous avez une boucle infinie, ce que le serveur n'appréciera pas.

### Exemple 3-13. L'instruction `break`

```
<?php  
//Création d'un tableau de noms ←①  
$tab[1]="Basile";  
$tab[2]="Conan";  
$tab[3]="Albert";  
$tab[4]="Vincent";  
//Boucle de lecture du tableau
```

```

for($i=1;$i<count($tab);$i++) ←②
{
    if ($tab[$i][0]=="A") ←③
    {
        echo "Le premier nom commençant par A est le n° $i: ",$tab[$i];
        break; ←④
    }
}
?>

```

## L'instruction continue

À la différence de l'instruction `break`, l'instruction `continue` n'arrête pas la boucle en cours mais seulement l'itération en cours. La variable `compteur` est incrémentée immédiatement, et toutes les instructions qui suivent le mot-clé `continue` ne sont pas exécutées lors de l'itération en cours.

L'exemple 3-14 donne deux illustrations de ce mécanisme :

- La première utilise une boucle `for` pour afficher tous les nombres de 1 à 20, à l'exception des multiples de 5 grâce au code `if($i%5==0) { continue;}` (repère ①).
- La deuxième utilise une boucle `foreach` qui parcourt un tableau de départements et permet de n'afficher que ceux qui commencent par la lettre `E` (repère ②).

### Exemple 3-14. Interruption partielle d'une boucle

```

<?php
//Interruption d'une boucle for
for($i=0;$i<20;$i++)
{
    if($i%5==0) { continue;} ←①
    echo $i,"<br />";
}
//*****
//Interruption d'une boucle foreach
$tab[1]="Ain";
$tab[2]="Allier";
$tab[27]="Eure";
$tab[28]="Eure et Loir";
$tab[29]="Finistère";
$tab[33]="Gironde";
foreach($tab as $cle=>$valeur)
{
    if($tab[$cle][0]!="E") { continue;} ←②
    echo "code $cle : département ",$tab[$cle],"<br />";
}
?>

```

Le résultat affiché par cet exemple est le suivant :

---

```

1 : 2 : 3 : 4 : 6 : 7 : 8 : 9 : 11 : 12 : 13 : 14 : 16 : 17 : 18 : 19 :
code 27 : département Eure
code 28 : département Eure et Loir

```

---

De même que pour l'instruction `break`, si plusieurs boucles sont imbriquées, il est possible d'interrompre les  $n$  boucles les plus internes en utilisant la syntaxe suivante :

| continue N

Cette possibilité est illustrée à l'exemple 3-15, qui contient trois boucles `for` imbriquées (repères 1, 2 et 3). La plus interne contient une instruction `if` qui stoppe les itérations en cours pour les trois boucles si la somme des variables `compteur $i, $j` et `$k` est un multiple de 3. C'est l'instruction `continue 3` (repère 4). Si tel est le cas, l'exécution passe directement à l'incrémentation de la variable `$i`, terminant ainsi de manière anticipée les itérations des trois boucles en cours.

## Exemple 3-15. Interruption de plusieurs boucles

```
<?php
for ($i=0;$i<10;$i++) ← 1
{
    for ($j=0;$j<10;$j++) ← 2
    {
        for ($k=0;$k<10;$k++) ← 3
        {
            if(($i+$j+$k)%3==0) continue 3; ← 4
            echo "$i : $j : $k <br /> ";
        }
    }
}
```

## L'instruction `goto`

À la demande générale, je suppose (?!), les concepteurs de PHP ont ajouté l'instruction `goto` à la version 5.4. Comme son nom l'indique, elle renvoie l'exécution en un point précis du script marqué par un label (dit aussi étiquette), ce dernier étant constitué par une chaîne de caractères terminée par le caractère : (deux-points). Un exemple typique peut être le suivant dans lequel le mot « Cacher » ne sera bien sûr jamais affiché au profit du mot « Voir ».

```
<?php
goto a;
echo 'Cacher';

a:
echo 'Voir';
?>
```

Cette instruction est généralement utilisée avec une instruction conditionnelle ce qui nécessite au moins une précaution. Dans l'exemple 3-16, nous reprenons la seconde partie de l'exemple 3-14 en utilisant un `goto`. Le but est ici d'afficher uniquement le premier département dont la première lettre est « E » (repère 1) en renvoyant l'exécution au label « info » (repère 4). Signalons qu'un problème se pose si aucune solution n'est trouvée (si on cherche un département commençant par la lettre K, par exemple). Dans ce cas, le message contenu dans la variable `$result` va s'afficher

(repère ②) mais le contenu du label aussi ! Pour éviter cela, nous insérons une instruction `exit` (repère ③) pour terminer proprement en cas de recherche infructueuse.

### Exemple 3-16. L'instruction goto

```
<?php
$tab[1]="Ain";
$tab[2]="Allier";
$tab[27]="Eure";
$tab[28]="Eure et Loir";
$tab[29]="Finistère";
$tab[33]="Gironde";
foreach($tab as $cle=>$valeur)
{
    if($tab[$cle][0]=="E") {$result=$tab[$cle]; goto info;} ← 1
    else $result="Pas de résultat";
}
echo $result; ← 2
exit; ← 3
info: ← 4
echo "Premier trouvé = ",$result;
?>
```

## Gestion des erreurs

Un bon script ne devrait théoriquement pas générer d'erreur. Certaines erreurs sont inévitables comme celles qui sont dues à des problèmes de connexion défaillante ou de bogue sur le serveur. Nous nous intéressons ici davantage aux erreurs qui peuvent être provoquées par des actions de l'utilisateur comme des saisies erronées qui provoquent l'arrêt du script ou encore à celles qui peuvent survenir lors de la tentative d'accès à une ressource inexistante.

La gestion des erreurs a pour but de signaler « proprement » les problèmes au visiteur et d'éviter l'affichage des messages d'erreur bruts tels que PHP les envoie au navigateur.

### *Suppression des messages d'erreur*

Si vous écrivez le code ci-dessous :

```
<?php
$a=10;
$b=0;
echo $a/$b; ← 1
fopen("inconnu.txt","r"); ← 2
?>
```

les messages suivants apparaissent :

**Warning:** Division by zero in `c:\wamp5\www\php5\c3instructions\instruct3.15a.php`  
on line 4

```
| Warning: fopen(inconnu.txt) [function.fopen]: failed to open stream: No such file or
| directory in c:\wamp5\www\php5\c3instructions\instruct3.15a.php on line 5
```

Le premier message correspond à la division par 0 (repère 1), et le second à la tentative d'ouverture d'un fichier qui n'existe pas (repère 2). Ces messages affichés dans le cours d'une page HTML sont du plus mauvais effet pour les visiteurs du site.

Pour éviter l'affichage des messages d'erreur de PHP dans le navigateur, il existe les moyens élémentaires suivants :

- Faire précédé l'appel d'une fonction du caractère @ en écrivant par exemple @fopen("inconnu.txt", "r").
- Utiliser la fonction `error_reporting()`, qui permet de n'afficher que certains messages selon le type d'erreur. Sa syntaxe est `int error_reporting ( [int niveau])`. C'est le paramètre `niveau` qui permet de choisir le niveau d'affichage des messages d'erreur. Ses principales valeurs sont indiquées au tableau 3-1. Vous pouvez combiner plusieurs valeurs avec l'opérateur |, comme dans le code suivant :

```
| error_reporting(E_WARNING | E_PARSE) ;
```

qui permet de n'afficher que les alertes et les erreurs de syntaxe. Pour bloquer tous les messages d'erreur, il faut passer à la fonction le paramètre 0. Cette fonction doit être utilisée dès le début du script.

**Tableau 3-1 – Liste des niveaux d'erreur courants**

Constante	Valeur	Niveau d'affichage
E_ERROR	1	Erreurs fatales qui provoquent l'arrêt du script, par exemple, l'appel d'une fonction qui n'existe pas.
E_WARNING	2	Avertissements ne provoquant pas l'arrêt du script, par exemple, une division par 0.
E_PARSE	4	Erreurs de syntaxe détectées par l'analyseur PHP et provoquant l'arrêt du script, par exemple l'oubli du point-virgule en fin de ligne.
E_NOTICE	8	Avis que le script a rencontré un problème simple qui peut ne pas être une erreur.
E_ALL	4095	Toutes les erreurs

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'exceptions. 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`.
- Créez un bloc `finally` qui contient le code à exécuter obligatoirement, qu'une erreur se soit produite ou non.

Un gestionnaire d'exceptions a donc la structure suivante :

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

Le nom de l'objet utilisé dans l'instruction `catch` est sans importance. Un même script peut comporter autant de blocs `try...catch...finally` que vous voulez. Il est également possible d'imbriquer des blocs `try` les uns dans les autres, à condition que chacun ait un bloc `catch` associé.

Le constructeur de l'objet `Exception` créé dans l'instruction `throw` reçoit deux paramètres,

correspondant aux propriétés `message` et `code` de l'objet. Le premier est une chaîne contenant le message d'erreur et le second un entier qui définit un code d'erreur facultatif. Cet objet est utilisé dans le bloc `catch` en appelant ses méthodes pour afficher des informations sur l'exception. Le tableau 3-2 donne la liste et le rôle des méthodes de l'objet `Exception`.

**Tableau 3-2 – Les méthodes de l'objet `Exception`**

Méthode	Définition
<code>__construct()</code>	Constructeur de l'objet. Il est appelé automatiquement lors de la création d'un objet avec le mot-clé <code>new</code> . Il définit les propriétés <code>message</code> et <code>code</code> de l'objet. Cette méthode ne doit pas être appelée explicitement.
<code>getMessage()</code>	Retourne la valeur de la propriété <code>message</code> dans une chaîne.
<code>getCode()</code>	Retourne la valeur de la propriété <code>code</code> sous la forme d'un entier.
<code>getFile()</code>	Retourne la valeur de la propriété <code>file</code> qui contient le nom et le chemin d'accès du fichier dans lequel s'est produite l'erreur.
<code>getLine()</code>	Retourne la valeur de la propriété <code>line</code> qui indique le numéro de ligne à laquelle a été créée l'exception.
<code>__toString()</code>	Retourne une chaîne contenant toutes les informations sur l'exception.

L'exemple 3-17 présente une première mise en œuvre du mécanisme des exceptions et des méthodes ci-dessus. Vous y définissez deux variables dans le but d'afficher le résultat de la division (repères ① et ②). Ces valeurs proviendraient en pratique des saisies d'un utilisateur. L'absence de gestion de l'erreur provoquerait l'affichage d'une alerte (warning). Supprimer l'affichage de cette alerte en utilisant la fonction `error_reporting()` avec le paramètre 0 laisserait l'utilisateur dubitatif car le résultat attendu ne serait pas affiché, et ce sans aucune explication.

Vous créez donc un bloc `try` (repère ③) pour gérer ce type d'erreur. Le contrôle de la valeur de la division permet de déclencher une exception en définissant un message approprié et un code d'erreur (repère ④). Si la valeur de `$b` était valide, le résultat de la division serait affiché normalement, et le bloc `catch` serait ignoré (repère ⑤).

Comme ici `$b` vaut 0, le bloc `catch` (repère ⑥) affiche successivement le message d'erreur (repère ⑦), le nom du fichier dans lequel se produit l'erreur (repère ⑧), la ligne de l'instruction `throw` à laquelle l'objet `$except` a été créé (repère ⑨), le code d'erreur (repère ⑩) et les informations sur l'exception (repère ⑪). Le bloc `finally` affiche dans tous les cas un message précisant que tout est bien géré (repère ⑫). L'affichage du message "Vraie Fin" prouve que l'exécution du script continue normalement, que le bloc `catch` soit exécuté ou non (repère ⑬).

### Exemple 3-17. Crédit d'un gestionnaire d'exceptions

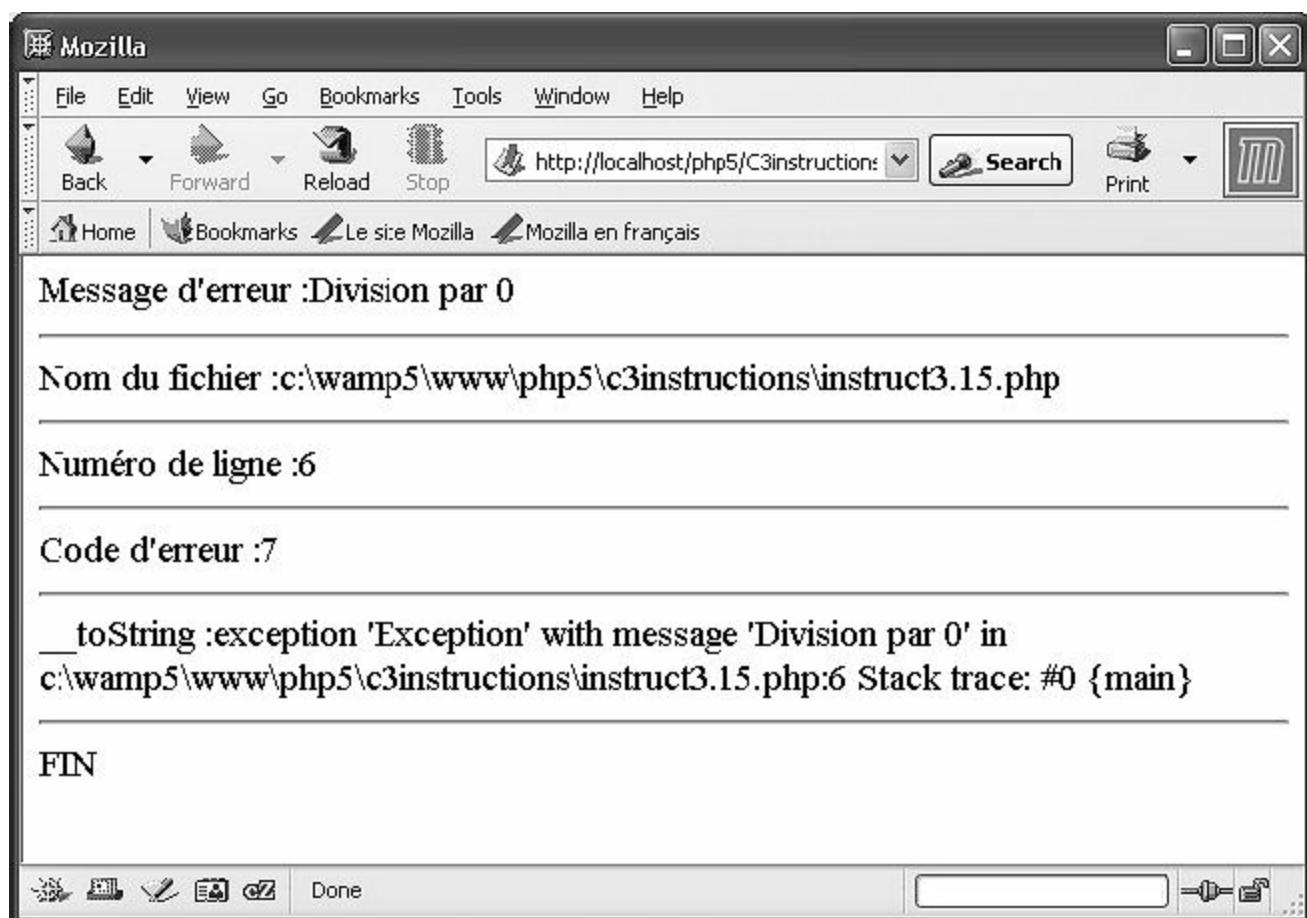
```
<?php
$a=100; ← ①
$b=0; ← ②
```

```

try ←③
{
    if($b==0){throw new Exception("Division par 0",7); } ←④
    else{echo "Résultat de : $a / $b = ",$a/$b;} ←⑤
}
catch(Exception $except) ←⑥
{
    echo "Message d'erreur :",$except->getMessage(),"<hr>"; ←⑦
    echo "Nom du fichier :",$except->getFile(),"<hr>"; ←⑧
    echo "Numéro de ligne :",$except->getLine(),"<hr>"; ←⑨
    echo "Code d'erreur :",$except->getCode(),"<hr>"; ←⑩
    echo "__toString :",$except->__toString(),"<hr>"; ←⑪
}
finally ←⑫
{
    echo "L'exception a été levée, le script continue<hr/>";
}
echo "Vraie Fin";←⑬
?>

```

La figure 3-4 montre le résultat obtenu.



**Figure 3-4**  
*Affichages créés par l'objet Exception*

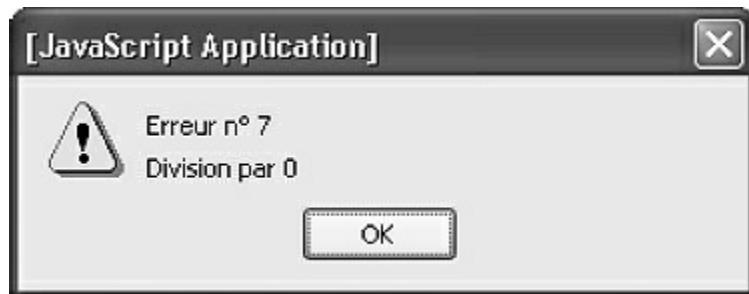
Cet exemple n'a pour but que d'illustrer l'utilisation des méthodes de l'objet `Exception`.

En production, vous n'afficheriez pas toutes ces informations, qui n'intéressent pas le visiteur.

Vous allez donc modifier ce script pour afficher une boîte d'alerte signalant simplement le problème à l'utilisateur. L'exemple 3-18 montre les modifications opérées. Dans le bloc `catch`, vous créez une boîte d'alerte utilisant la fonction JavaScript `alert()`, qui affiche uniquement le message et le code d'erreur (repère ①).

### Exemple 3-18. Affichage pour le visiteur

```
<?php
$a=10;
$b=0;
try
{
    if($b==0) {throw new Exception("Division par 0",7);}
    else{echo "Résultat de : $a / $b = ",$a/$b;}
}
catch(Exception $except)
{
    echo "<script type=\"text/javascript\"> alert(' Erreur n°
    ↪ ",$except ->getCode()," \n  ",$except ->getMessage() ,'' ) </script> "; ← ①
}
finally
{
    echo "Tout est sous contrôle";
}
echo "FIN";
?>
```



**Figure 3-5**  
*Message d'erreur pour le visiteur*

### Personnalisation de l'objet `Exception`

Le mécanisme de l'héritage (voir le chapitre 9) permet d'étendre une classe pour lui ajouter des méthodes ou des propriétés. Vous pouvez donc enrichir l'objet `Exception` avec de nouvelles méthodes. Il serait même envisageable de créer des objets personnalisés permettant de gérer des types d'erreurs spécifiques.

L'exemple 3-19 illustre cette possibilité d'extension en créant une classe qui hérite de la classe `Exception` (repère ①). Une nouvelle méthode, nommée `alerte()` (repère ②), est ajoutée pour créer une boîte d'alerte JavaScript. Le simple appel de la méthode `alerte()`

pour l'objet `MonException` (repère ③) retourne le code qui crée cette boîte (repère ④) sans avoir à réécrire systématiquement ce code dans chaque script à chaque utilisation.

Pour montrer qu'un même bloc `try` peut gérer plusieurs erreurs différentes, vous créez deux conditions `if` qui correspondent à des situations différentes. La première déclenche une exception si `$b` vaut 0 (repère ⑤), et la seconde une autre exception si le résultat de `$a/$b` n'est pas un entier, c'est-à-dire si `$a` n'est pas un multiple entier de `$b` (repère ⑥). En fonction de la valeur de `$b`, l'appel de la méthode `alerte()` contenue dans le bloc `catch` (repère ⑦) de l'objet de type `MonException` qui sera créé affiche une boîte d'alerte identique à celle de la figure 3-5 (si `$b` vaut 0) ou celle de la figure 3-6 (comme c'est le cas dans l'exemple où `$b` vaut 3).

### Exemple 3-19. Exception personnalisée

```
<?php
//Création d'une classe dérivée de Exception
class MonException extends Exception ← ①
{
    public function alerte() ← ②
    {
        $this->message ="<script type=\"text/javascript\"> alert(' Erreur n° ".$this->
        getCode()." \n ".$this->getMessage()." ') </script> "; ← ③
        return $this->getMessage(); ← ④
    }
}
//Utilisation de la classe
$a=100;
$b=3;
try
{
    if($b == 0) {throw new MonException("Division par 0 ",7); } ← ⑤
    elseif($a%$b != 0) {throw new MonException("Quotient entier impossible",55); } ← ⑥
    else{echo "Résultat de : $a / $b = ",$a/$b;}
}
catch(MonException $except)
{
    echo $except->alerte(); ← ⑦
}
finally
{
    echo "Le script continue sans problème <br/>";
}
echo "FIN";
?>
```



## **Figure 3-6**

### *Affichage pour une erreur de divisibilité*

Vous pouvez envisager de créer un grand nombre d'objets personnalisés permettant de gérer rapidement et d'une manière adaptée toutes sortes d'erreurs de types très différents.

## **Exercices**

### **Exercice 1**

Rédigez une expression conditionnelle pour tester si un nombre est à la fois un multiple de 3 et de 5.

### **Exercice 2**

Écrivez une expression conditionnelle utilisant les variables `$age` et `$sexe` dans une instruction `if` pour sélectionner une personne de sexe féminin dont l'âge est compris entre 21 et 40 ans et afficher un message de bienvenue approprié.

### **Exercice 3**

Effectuez une suite de tirages de nombres aléatoires jusqu'à obtenir une suite composée d'un nombre pair suivi de deux nombres impairs.

### **Exercice 4**

Créez et affichez des numéros d'immatriculation automobile (pour Paris, par exemple) en commençant au numéro 100 PHP 75. Effectuez ensuite la même procédure en mettant en réserve les numéros dont le premier groupe de chiffres est un multiple de 100. Stockez ces numéros particuliers dans un tableau.

### **Exercice 5**

Choisissez un nombre de trois chiffres. Effectuez ensuite des tirages aléatoires, et comptez le nombre de tirages nécessaire pour obtenir le nombre initial. Arrêtez les tirages, et affichez le nombre de coups réalisés. Réalisez ce script d'abord avec l'instruction `while` puis avec l'instruction `for`.

### **Exercice 6**

Créez un tableau dont les indices varient de 11 à 36 et dont les valeurs sont des lettres de A à Z. Lisez ensuite ce tableau avec une boucle `for` puis une boucle `foreach`, et affichez les indices et les valeurs (la fonction `chr(n)` retourne le caractère dont le code ASCII vaut `n`).

### **Exercice 7**

Utilisez une boucle `while` pour déterminer le premier entier obtenu par tirage aléatoire qui soit un multiple d'un nombre donné. Écrivez la variante utilisant la boucle `do...while`.

## **Exercice 8**

Recherchez le PGCD (plus grand commun diviseur) de deux nombres donnés. Gérez au moyen d'une exception le cas où au moins un des nombres n'est pas entier.

# 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. Depuis la version 5.4, elle peut être remplacée par une syntaxe courte en écrivant simplement `[]` à la place de la fonction `array()`.

### 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;
```

où `n` est un indice entier quelconque, et `valeur` un scalaire ou une variable de type `integer`, `double`, `boolean`, `string` ou `array`.

Cette manière de procéder se révèle rapidement rébarbative dès qu'il s'agit de définir

un nombre plus important d'éléments. Pour créer un tableau composé de plusieurs éléments en une seule opération, vous disposez heureusement de la fonction `array()`, dont la syntaxe est la suivante :

```
| $tab = array(valeur0,valeur1,...,valeurN);
```

Ou encore avec la syntaxe courte :

```
| $tab=[valeur0,valeur1,...,valeurN];
```

La variable `$tab` est ici un tableau indicé dont les valeurs d'indice varient de `0` à `N`. Ce tableau a donc `N + 1` éléments, accessibles par la notation habituelle `$tab[0]`, `$tab[1]`, ..., `$tab[N]`, dont les valeurs respectives peuvent avoir l'un quelconque des types précités.

Nous pouvons aussi lire une valeur en déréférençant un tableau. En écrivant par exemple : `[val1,val2,val3][1]`, nous lisons la deuxième valeur d'indice `1` "val2".

### Premier indice

Avec ce mode de création de tableau, le premier indice a, une fois encore, toujours la valeur `0`. Il ne faut pas oublier d'en tenir compte lors des opérations de lecture des éléments.

## Les tableaux associatifs

La même fonction `array()` permet aussi de créer rapidement un tableau associatif en définissant pour chacun de ses éléments une clé et une valeur.

La syntaxe avec la fonction `array()` est la suivante :

```
| $tabasso = array("cléA"=>valeurA, "cléB"=>valeurB,... "cléZ"=>valeurZ)
```

La syntaxe courte est :

```
| $tabasso = ["cléA"=>'valeurA', "cléB"=>'valeurB', "cléZ"=>'valeurZ'];
```

Comme vous l'avez vu au chapitre 2, chaque clé est une chaîne de caractères délimitée par des guillemets.

Pour lire `valeurA`, vous écrivez :

```
| $tabasso["cléA"]
```

de la même façon que lorsque chaque élément est créé individuellement.

Dans un tableau associatif, la notion d'ordre des éléments perd la valeur qu'elle peut avoir dans un tableau indicé. Les clés ne sont pas numérotées, par exemple. Vous pourriez énumérer clés et valeurs dans un ordre différent sans que cela gène la lecture individuelle de chaque élément.

Vous auriez donc pu créer le même tableau en écrivant :

```
| $tabasso = array("cléZ"=>valeurZ, "cléY"=>valeurY,... "cléA"=>valeurA)
```

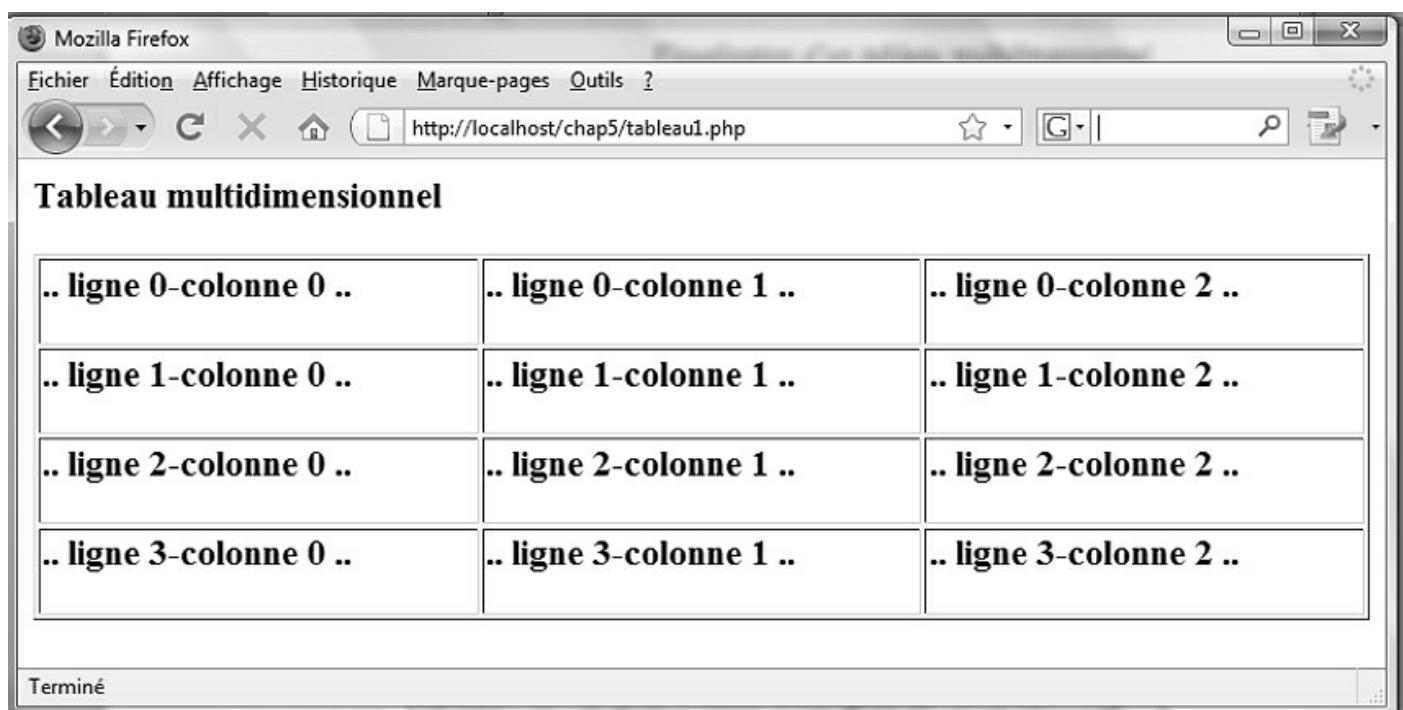
qui correspond à l'ordre inverse. Vous pouvez en fait effectuer la création dans un ordre quelconque, sans changer quoi que ce soit à l'accès aux valeurs à l'aide de leur clé.

Cela confirme la souplesse d'utilisation des tableaux associatifs en comparaison des tableaux indicés. Cette souplesse se révèle particulièrement utile dans les opérations de suppression d'éléments ou de tri sur des tableaux, lesquelles peuvent faire perdre les associations entre indices et valeurs.

## Les tableaux multidimensionnels

Contrairement aux langages dans lesquels vous déclarez les variables et leur type, comme ASP.Net utilisé avec C#, PHP ne comporte pas de méthode explicite de création de tableaux multidimensionnels. C'est tout l'avantage de ce langage que d'autoriser qu'un élément de tableau puisse être un tableau lui-même. La création de tableau comportant un nombre quelconque de dimension en est d'autant facilitée.

Un tableau multidimensionnel est similaire à une matrice, au sens mathématique du terme. La structure d'un tableau à deux dimensions peut se représenter sous la forme d'un tableau à double entrée, comme l'exprime le listing de l'exemple 5-1, dont le résultat est illustré à la figure 5-1.



The screenshot shows a Mozilla Firefox browser window. The title bar says "Mozilla Firefox". The menu bar includes "Fichier", "Édition", "Affichage", "Historique", "Marque-pages", "Outils", and a question mark icon. The address bar shows the URL "http://localhost/chap5/tableau1.php". Below the address bar are standard browser controls for back, forward, stop, and search. The main content area has a title "Tableau multidimensionnel". Inside is a 3x4 grid of text:

.. ligne 0-colonne 0 ..	.. ligne 0-colonne 1 ..	.. ligne 0-colonne 2 ..	
.. ligne 1-colonne 0 ..	.. ligne 1-colonne 1 ..	.. ligne 1-colonne 2 ..	
.. ligne 2-colonne 0 ..	.. ligne 2-colonne 1 ..	.. ligne 2-colonne 2 ..	
.. ligne 3-colonne 0 ..	.. ligne 3-colonne 1 ..	.. ligne 3-colonne 2 ..	

In the bottom left corner of the browser window, there is a status bar with the text "Terminé".

**Figure 5-1**  
*Visualisation d'un tableau multidimensionnel*

Les valeurs des éléments ont été choisies de manière à montrer clairement par la suite comment accéder à une valeur particulière. Le premier chiffre est celui de la ligne, et le second celui de la colonne. Chaque ligne est à la fois un élément du tableau principal, qui contient quatre éléments, et est elle-même un tableau à trois éléments.

Vous obtenez au total douze valeurs dans le tableau, mais il n'a que quatre éléments.

De même que pour repérer un élément dans un espace à  $n$  dimensions il faut utiliser  $N$  coordonnées, pour lire une valeur dans un tableau à  $n$  dimensions, il faut utiliser  $N$

indices, ou clés, différents.

Par exemple, pour récupérer dans la variable \$a la deuxième valeur (indice 1) de la troisième ligne (indice 2) du tableau précédent \$tabmulti, écrivez :

```
$a = $tabmulti[2][1]
```

Dans cet exemple, chaque élément du tableau est un tableau ayant autant d'éléments que les autres, ce qui réalise une matrice rectangulaire  $4 \times 3$ . Il est évidemment possible de créer des tableaux multidimensionnels non symétriques, dans lesquels le premier élément pourrait être, par exemple, une valeur entière ou une chaîne de caractères, le deuxième élément un tableau à dix éléments, le troisième un tableau avec un nombre d'éléments différents puis tout autre combinaison possible. Quoique parfaitement réalisable, ce type de tableau, qui est irrégulier, risque de se révéler difficile à lire avec une boucle, comme vous le verrez dans les exemples présentés dans la suite du chapitre.

### Pour en savoir plus

Les boucles sont abordées en détail au chapitre 3.

Le listing 5-1 utilise deux boucles de lecture pour afficher le contenu du tableau. Vous pouvez très bien envisager de créer de la même façon des tableaux à trois, quatre dimensions et bien plus encore, mais la visualisation en devient vite difficile.

Pour créer le tableau de la figure 5-1, utilisez la fonction `array()` de la façon suivante :

```
$tabmulti=array(  
    array("ligne 0-colonne 0","ligne 0-colonne 1","ligne 0-colonne 2"),  
    array("ligne 1-colonne 0","ligne 1-colonne 1","ligne 1-colonne 2"),  
    array("ligne 2-colonne 0","ligne 2-colonne 1","ligne 2-colonne 2"),  
    array("ligne 3-colonne 0","ligne 3-colonne 1","ligne 3-colonne 2")  
) ;
```

Ou encore avec la syntaxe courte :

```
$tabmulti=[ ["ligne 0-colonne 0","ligne 0-colonne 1","ligne 0-colonne 2"],  
          ["ligne 1-colonne 0","ligne 1-colonne 1","ligne 1-colonne 2"],  
          ["ligne 2-colonne 0","ligne 2-colonne 1","ligne 2-colonne 2"],  
          ["ligne 3-colonne 0","ligne 3-colonne 1","ligne 3-colonne 2"]];
```

Afin de visualiser la structure complète du tableau \$tabmulti, vous pouvez, dans les phases de test des scripts, utiliser la fonction `print_r($tabmulti)`, qui affiche la structure suivante :

```
Array (  
    [0] => Array ( [0] => ligne 0-colonne 0 [1] => ligne 0-colonne 1 [2] => ligne 0-colonne 2 )  
    [1] => Array ( [0] => ligne 1-colonne 0 [1] => ligne 1-colonne 1 [2] => ligne 1-colonne 2 )  
    [2] => Array ( [0] => ligne 2-colonne 0 [1] => ligne 2-colonne 1 [2] => ligne 2-colonne 2 )  
    [3] => Array ( [0] => ligne 3-colonne 0 [1] => ligne 3-colonne 1 [2] => ligne 3-colonne 2 ) )
```

Vous retrouvez bien les différents éléments du tableau et leur contenu.

La fonction `var_dump($tabmulti)` permet d'afficher un ensemble d'informations encore plus complètes sur le tableau. Elle donne le nombre d'éléments et la description de chacun d'eux, ainsi que le type et le contenu de chaque valeur. Vous obtenez ainsi pour le même tableau :

```
array(4) { [0]=> array(3) { [0]=> string(17) "ligne 0-colonne 0" [1]=> string(17)  
"ligne 0-colonne 1" [2]=> string(17) "ligne 0-colonne 2" }  
[1]=> array(3) { [0]=> string(17) "ligne 1-colonne 0" [1]=> string(17) "ligne 1-colonne  
1" [2]=>  
string(17) "ligne 1-colonne 2" }  
[2]=> array(3) { [0]=> string(17) "ligne 2-colonne 0" [1]=> string(17) "ligne 2-colonne  
1" [2]=>  
string(17) "ligne 2-colonne 2" }  
[3]=> array(3) { [0]=> string(17) "ligne 3-colonne 0" [1]=> string(17) "ligne 3-colonne  
1" [2]=>  
string(17) "ligne 3-colonne 2" } }
```

Dans cette description, `array(4)` signifie que la variable `$tabmulti` est un tableau à quatre éléments. Vient ensuite la description de tous les éléments sur le même principe. Les expressions du type :

[0]=> array(3)

signifient que le premier élément du tableau est lui-même un tableau à trois éléments. Les accolades qui suivent donnent le type et la valeur de chacun des éléments de ce dernier tableau. Dans la suite de la description, nous trouvons :

[0]=> string(17) "ligne 0-colonne 0"

qui indique que l'élément d'indice 0 est une chaîne de dix-sept caractères, dont la valeur est "ligne 0-colonne 0".

L'utilisation de ces fonctions sert au programmeur à des fins de débogage des scripts. Il serait maladroit de les employer pour créer un affichage à destination de l'utilisateur final, qui les trouverait pour le moins obscures et peu « parlantes ».

## Exemple 5-1. Création d'un tableau multidimensionnel

```
<?php  
$tabmulti=[ ["ligne 0-colonne 0","ligne 0-colonne 1","ligne 0-colonne 2"],  
["ligne 1-colonne 0","ligne 1-colonne 1","ligne 1-colonne 2"], ["ligne 2-colonne 0",  
"ligne 2-colonne 1","ligne 2-colonne 2"], ["ligne 3-colonne 0","ligne 3-colonne 1",  
"ligne 3-colonne 2"]];  
echo "<h3>Tableau multidimensionnel</h3><table border='1' width=\"100%\"> <tbody>";  
for ($i=0;$i<count($tabmulti);$i++)  
{  
    for ($j=0;$j<count($tabmulti[$i]);$j++)  
    {  
        echo "<td><h3> .. ",$tabmulti[$i][$j]," .. </h3></td>";  
    }  
    echo "</tr>";  
}  
echo " </tbody> </table> ";  
?>
```

Le résultat obtenu est celui de la figure 5-1.

## Créer des suites

Pour créer des tableaux dont les éléments sont des suites de nombres et éventuellement de lettres, comme vous le verrez à l'exemple 5-2, vous pourriez envisager d'utiliser une boucle `for`. PHP propose toutefois une fonction `range()`, qui permet de réaliser cette opération en une seule ligne de code. Sa syntaxe est la suivante :

```
| array range(int mini,int maxi)
```

Cette fonction retourne un tableau indicé contenant tous les entiers compris entre les valeurs `mini` et `maxi`.

Pour créer des suites de lettres, le moyen le plus classique serait d'écrire une boucle utilisant la fonction `chr(n)`, qui retourne le caractère dont le code ASCII est `n`. Pour créer un tableau contenant la suite de lettres de "A" à "Z", vous utiliseriez donc les valeurs de `n` comprises entre 65 et 90, ou 97 et 122 pour la suite de "a" à "z". Au lieu de cela, PHP vous permet d'utiliser la fonction `range()`, dont la syntaxe est alors la suivante :

```
| $tabalpha = range("A","Z")
```

Cette possibilité n'est pas mentionnée dans la documentation officielle, d'où l'utilité de l'expérimentation personnelle.

### Script de tableau

Dans les résultats affichés par le script, pour les tableaux créés à l'aide de la fonction `range()`, le premier indice est `0` alors qu'avec une boucle `for` il serait possible de choisir l'indice `1`.

### Exemple 5-2. La création de suites

```
<?php
// Suite de nombres de 1 à 10
$tabnombre= range(1,10);
print_r($tabnombre);
echo "<hr>";
// Suite de lettres de a à z avec une boucle
for($i=97;$i<=122;$i++)
{
$tabalpha[$i-96]=chr($i);
}
print_r($tabalpha);
echo "<hr>";
// Suite de lettres de A à M avec range()
$tabalpha2 = range("A","M");
print_r($tabalpha2);
?>
```

Le script de l'exemple 5-2 affiche les résultats suivants :

---

```
Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 [5] => 6 [6] => 7 [7] => 8 [8]
=> 9 [9] => 10 )
```

```
Array ( [1] => a [2] => b [3] => c [4] => d [5] => e [6] => f [7] => g [8] => h [9] =>
i [10] =>
j [11] => k [12] => l [13] => m [14] => n [15] => o [16] => p [17] => q [18] => r [19]
```

```
=> s [20]
=> t [21] => u [22] => v [23] => w [24] => x [25] => y [26] => z )
Array ( [0] => A [1] => B [2] => C [3] => D [4] => E [5] => F [6] => G [7] => H [8] =>
I [9] =>
J [10] => K [11] => L [12] => M )
```

---

## ***Créer un tableau à partir d'une chaîne***

Une dernière façon utile de créer des tableaux consiste à décomposer une chaîne de caractères en un ensemble de sous-chaînes, dont chacune devient un élément de tableau.

Le critère de coupure de la chaîne est un caractère quelconque à choisir par le programmeur. Cela peut être une espace, pour décomposer une phrase en mots au sens courant du terme, ou le caractère "@", pour séparer le nom d'utilisateur de celui de son serveur de courrier dans une adresse e-mail.

Cette opération peut être effectuée très simplement au moyen de la fonction `explode()`, dont la syntaxe est la suivante :

```
| array explode(string "coupe",string $chaine,[int nbmax])
```

Cette fonction retourne un tableau composé des sous-chaînes de `$chaine` créées avec le critère de coupure contenu dans la chaîne "`coupe`", l'entier `nbmax` facultatif donnant le nombre maximal de sous-chaîne désiré.

Le code suivant :

```
<?php
$chaine="La cigale et la fourmi";
$tabmot = explode(" ",$chaine);
print_r($tabmot);
?>
```

affiche la structure du tableau `$tabmot` :

```
Array ( [0] => La [1] => cigale [2] => et [3] => la [4] => fourmi )
```

---

En passant comme premier paramètre le caractère "@", vous pouvez décomposer une adresse e-mail avec le code suivant :

```
<?php
$adresse="machin@wanadoo.fr" ;
$tabsite=explode("@",$adresse);
echo "L'utilisateur est : {$tabsite[0]} et son serveur mail est {$tabsite[1]} ";
?>
```

qui affiche :

```
| L'utilisateur est : machin et son serveur mail est wanadoo.fr
```

## ***Compter le nombre de valeurs d'un tableau***

Vous venez de voir la distinction entre le nombre d'éléments d'un tableau et le nombre

de valeurs qu'il contient. Vous allez maintenant découvrir comment déterminer le nombre exact de valeurs d'un tableau.

Vous avez déjà employé la fonction `count()` pour déterminer le nombre d'éléments d'un tableau. Quand un tableau n'a qu'une seule dimension, la valeur obtenue correspond au nombre de valeurs contenues dans le tableau. Si le tableau est multidimensionnel, en revanche, la valeur renvoyée par `count()` n'est pas égale au nombre de valeurs.

L'exemple 5-3 fournit une méthode permettant de déterminer le nombre de valeurs d'un tableau à l'aide d'une boucle. Cette dernière examine le type de chacun des six éléments du tableau `$stabdiv`, qui contient des chaînes de caractères, des tableaux et des entiers.

Si l'un des éléments est un tableau, vous réutilisez la fonction `count()` pour trouver le nombre de valeurs qu'il contient et incrémentez le compteur `$nb_val` d'autant, sauf pour les entiers et les chaînes, pour lesquels vous l'incrémentez d'une unité seulement. Vous obtenez de la sorte le nombre total de valeurs.

Si le tableau avait trois dimensions au lieu de deux, il vous faudrait ajouter une boucle `for` supplémentaire pour arriver au même résultat.

Le listing de l'exemple 5-3 fournit le résultat suivant :

---

```
Le tableau $stabdiv contient 6 éléments
Le tableau $stabdiv contient 11 valeurs
```

---

## Exemple 5-3. Comptage du nombre de valeurs

```
<?php
// Comptage du nombre d'éléments
$stabdiv=array("Bonjour","Web",array("1-0","1-1","1-2"),1970,2013,array("3-0",
  "3-1","3-2","3-3"));
echo "Le tableau \$stabdiv contient ",count($stabdiv)," éléments <br />";
//ou encore: echo "Le tableau \$stabdiv contient ",sizeof($stabdiv)," éléments <br />";
// Comptage du nombre de valeurs
$nb_val=0;
for ($i=0;$i<count($stabdiv);$i++)
{
    if(gettype($stabdiv[$i])=="array")
    {
        $nb_val+=count($stabdiv[$i]);
    }
    else
    {
        $nb_val++;
    }
}
echo "Le tableau \$stabdiv contient ",$nb_val," valeurs <br />";
?>
```

### La fonction `sizeof()`

La fonction `sizeof()` est un alias de la fonction `count()`. Vous pouvez l'employer à la place dans tous les exemples précédents.

Dans le même ordre d'idée, vous pouvez être amené à vouloir compter non pas le nombre total de valeurs d'un tableau mais le nombre de valeurs différentes qu'il contient.

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

```
| $result = array_count_values($tab)
```

retourne le tableau associatif `$result`, ayant pour clés les valeurs du tableau `$tab` et pour valeur associée à chaque clé le nombre de fois que chacune apparaît dans le tableau `$tab`.

Cette fonction peut vous permettre de réaliser une analyse statistique des données contenues dans un tableau. Cela se révèle pratique lorsque les données sont nombreuses, comme après une requête de sélection dans une base de données.

Le listing 5-4 offre une illustration de l'emploi de cette fonction. La fonction `count()` affiche le nombre d'éléments du tableau `$tab`. La fonction `array_count_values()` affiche le nombre de valeurs différentes que contient le tableau. Enfin, la fonction `$result` donne des informations statistiques sur le nombre d'occurrences de chaque valeur.

## Exemple 5-4. Comptage du nombre de valeurs

```
<?php
$tab= array("Web","Internet","PHP","JavaScript","PHP","ASP","PHP","ASP");
$result=array_count_values($tab);
echo "Le tableau \$tab contient ",count($tab)," éléments <br>";
echo "Le tableau \$tab contient ",count($result)," valeurs différentes <br>";
print_r($result);
?>
```

Le listing de l'exemple 5-4 fournit le résultat suivant :

---

```
Le tableau $tab contient 8 éléments
Le tableau $tab contient 5 valeurs différentes
Array ( [Web] => 1 [Internet] => 1 [PHP] => 3 [Javascript] => 1 [ASP] => 2 )
```

---

### La fonction `array_count_values()`

La fonction `array_count_values()` ne s'applique que si les éléments sont de type `integer`, `double` ou `string` mais pas de type `array`. Elle n'est pas adaptée pour compter le nombre de valeurs d'un tableau multidimensionnel.

En comparaison des fonctions `print_r()` et `var_dump()`, que vous avez utilisées pour afficher l'ensemble des éléments d'un tableau, les boucles de lecture des éléments de tableau, de leur indice et de leur clé offrent un meilleur affichage, sous forme de tableau HTML, par exemple.

Ce sont ces méthodes que vous mettrez en pratique dans les sections suivantes et dans la suite de l'ouvrage pour lire les données d'un tableau et les restituer dans des pages HTML.

# Lire les éléments des tableaux

Comme vous venez de le voir, il est souvent utile d'afficher l'ensemble des informations contenues dans un tableau, que ce soit la valeur des éléments qu'il contient ou les couples indice-valeur et clé-valeur des tableaux respectivement indicés ou associatifs.

Cette section présente un large éventail des possibilités de lecture des tableaux. Ces dernières répondent à tous les besoins, y compris la lecture intégrale des tableaux multidimensionnels.

## *Lire avec une boucle for*

La boucle `for` a besoin d'un paramètre d'arrêt. Vous allez employer pour cela la fonction `count()`, qui retourne le nombre total d'éléments de la boucle. L'utilisation de la variable `$i` comme compteur de la boucle permet de parcourir l'ensemble des valeurs du tableau unidimensionnel.

Le listing de l'exemple 5-5 donne un exemple de lecture d'un tableau indicé.

### Exemple 5-5. Lecture d'un tableau indicé à l'aide de la boucle `for`

```
<?php
$montab=array("Paris","London","Brüssel");
for ($i=0;$i<count($montab);$i++)
{ echo "L'élément $i est $montab[$i]<br />";}
?>
```

Le listing fournit le résultat suivant :

---

```
L'élément 0 est Paris
L'élément 1 est London
L'élément 2 est Brüssel
```

---

La boucle `for` peut permettre la lecture de tableaux multidimensionnels, à condition d'écrire autant de niveaux de boucles qu'il y a de dimensions dans le tableau.

Le listing de l'exemple 5-6 donne un exemple de lecture de ce type de tableau à l'aide de deux boucles imbriquées. La première parcourt les éléments du tableau `$clients` à l'aide d'un compteur `$i`. Comme chacun de ces éléments est un tableau, la seconde boucle de compteur `$j` lit l'ensemble des éléments contenus dans ces tableaux.

L'affichage s'effectue sous la forme d'un tableau HTML avec en-tête et pied à l'aide des éléments HTML `<thead>` et `<tfoot>`. Ces derniers sont très utiles pour améliorer la présentation des tableaux, notamment des longs tableaux.

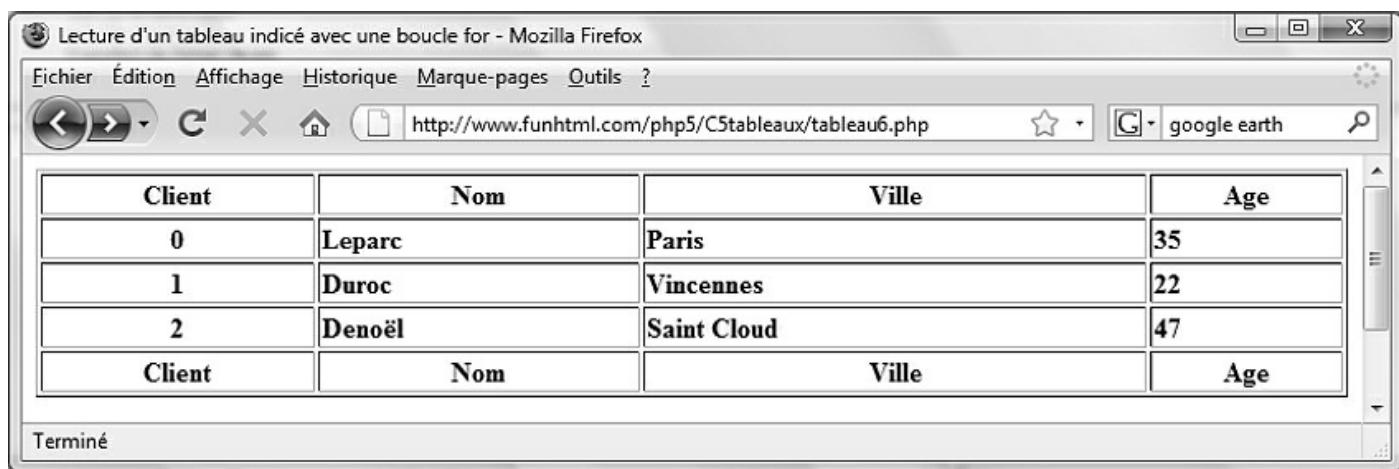
### Exemple 5-6. Lecture d'un tableau multidimensionnel indicé à l'aide de la boucle `for`

```
<!DOCTYPE html>
<html lang="fr">
<head>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Lecture d'un tableau indicé avec une boucle for</title>
</head>
<body>
<div>
<?php
// Création du tableau
$clients = array(array ("Leparc", "Paris", "35"), array("Duroc", "Vincennes", "22"),
array("Denoël","Saint Cloud","47"));
/* Alternative à la création du même tableau
$stab1= array("Leparc","Paris","35");
$stab2= array("Duroc","Vincennes","22");
$stab3= array("Denoël","Saint Cloud","47");
$clients=array($stab1,$stab2,$stab3); */
echo "<table border=\"1\" width=\"100%\" >";
// En-tête du tableau
echo "<thead><tr> <th> Client </th><th> Nom </th><th> Ville </th><th> Age </th>
</tr></thead>";
// Pied de tableau
echo "<tfoot> <tr><th> Client </th><th> Nom </th><th> Ville </th><th> Age </th>
</tr></tfoot><tbody>";
// Lecture des indices et des valeurs
for ($i=0;$i<count($clients);$i++)
{
    echo "<tr><td align=\"center\"><b>$i </b></td>";
    for($j=0;$j<count($clients[$i]);$j++)
    {
        echo "<td><b>",$clients[$i][$j]," </b></td>";
    }
    echo "</tr>";
}
?>
</tbody>
</table>
</div>
</body>
</html>

```



**Figure 5-2**

*Lecture d'un tableau à deux dimensions et affichage sous forme de tableau HTML*

## ***Lire avec une boucle while***

La boucle `for` nécessite par définition de connaître le nombre d’itérations à effectuer tel que fourni par la fonction `count()`. Ce n’est pas le cas avec la boucle `while`, qui se révèle de ce fait plus efficace dans le cas d’un tableau retourné après une requête sur une base de données, le nombre de réponses étant bien évidemment inconnu.

Pour un tableau à une seule dimension, l’expression booléenne contenue dans l’instruction `while` est `isset($tab[$i])`. Cette expression prend la valeur `TRUE` tant que l’élément désigné par `$tab[$i]` existe. Sinon, elle prend la valeur `FALSE`, ce qui est le cas en fin de tableau.

La variable `$i` étant incrémentée dans la boucle, `isset($tab[$i])` prend la valeur `FALSE` quand `$i` dépasse le nombre d’éléments du tableau `$tab`, ce qui provoque l’arrêt de la boucle.

Par précaution, vous pouvez initialiser la variable `$i` à `0` avant de démarrer la boucle de lecture au cas où elle aurait été utilisée auparavant dans le script et conserverait une valeur. L’exemple 5-7 illustre la lecture d’un tableau indicé à une dimension qui fournit le même affichage que le listing de l’exemple 5-5.

### Exemple 5-7. Lecture d’un tableau indicé à l’aide de la boucle `while`

```
<?php
$montab=array("Paris", "London", "Brüssel");
$i=0;
while(isset($montab[$i]) )
{
    echo "L'élément $i est $montab[$i]<br />";
    $i++;
}
?>
```

Si le tableau est multidimensionnel, vous opérez de même au moyen de deux boucles `while` imbriquées. L’expression booléenne de la seconde boucle est `isset($tab[$i] [$j])`. Elle est évaluée de la même façon que précédemment.

Le compteur `$j` du nombre d’éléments est également initialisé à `0` avant le début de la boucle – c’est ici indispensable pour pouvoir lire les lignes suivantes – et est incrémenté après chaque affichage d’un élément.

### Exemple 5-8. Lecture d’un tableau indicé multidimensionnel à l’aide de la boucle `while`

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Lecture d'un tableau indicé avec une boucle while</title>
</head>
<body>
    <div>
        <?php
        // Création du tableau
        $clients = array(array ("Leparc", "Paris", "35"), array("Duroc", "Vincennes", "22"),
```

```

array("Denoël","Saint Cloud","47"));
/* Ajout d'un élément */
$clients[7] = array("Duval","Marseille","76");
// Création du tableau HTML
echo "<table border=\"1\" width=\"100%\" ><thead><tr> <th> Client </th><th> Nom
</th><th> Ville </th><th> Age </th></tr></thead><tfoot> <tr><th> Client </th>
<th> Nom </th><th> Ville </th><th> Age </th></tr></tfoot><tbody>";
// Lecture des éléments
$i=0;
while(isset($clients[$i]))
{
    echo "<tr><td align=\"center\"><b>$i </b></td>";
    $j=0;
    while(isset($clients[$i][$j]))
    {
        echo "<td><b>", $clients[$i][$j], " </b></td>";
        $j++;
    }
    echo "</tr>";
    $i++;
}
?>
</tbody> </table>
</div>
</body>
</html>

```

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']}
```

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.

Comme vous pouvez le constater au listing 5-9, l'ajout d'un élément après la création du tableau, en particulier avec un indice non consécutif aux précédents, ne perturbe pas la lecture de l'intégralité du tableau, à la différence de l'exemple 5-8.

### La notation `{$element['key']}`

Dans le listing 5.9, la notation `{$element['key']}` permet que l'élément soit évalué à l'intérieur de la chaîne de caractères. Le fait d'écrire à la place `$element['key']` provoque une erreur.

## Exemple 5-9. Lecture à l'aide de la fonction `each()`

```
<?php
//*****Lecture d'un tableau indicé*****
$montab=array("Paris","London","Brüssel");//indices 0,1,2
//Ajout d'un élément au tableau
$montab[9]="Berlin";
//Lecture des éléments
reset($montab);
while($element=each($montab))
{
echo "L'élément d'indice $element[0] a la valeur $element[1]<br />";
//$_i++;
}
echo "<hr>";
//*****Lecture d'un tableau associatif*****
```

```

$montab=array("France"=>"Paris","Great Britain"=>"London","België"=>"Brüssel");
//Ajout d'un élément au tableau
$montab["Deutschland"]="Berlin";
//Lecture des éléments
reset($montab);
while($element=each($montab))
{
echo "L'élément de clé {$element['key']} a la valeur {$element['value']}<br />";
//$_i++;
}
?>

```

Le listing 5-9 affiche le résultat suivant :

---

```

L'élément d'indice 0 a la valeur Paris
L'élément d'indice 1 a la valeur London
L'élément d'indice 2 a la valeur Brüssel
L'élément d'indice 9 a la valeur Berlin

```

---

```

L'élément de clé France a la valeur Paris
L'élément de clé Great Britain a la valeur London
L'élément de clé België a la valeur Brüssel
L'élément de clé Deutschland a la valeur Berlin

```

---

La fonction `each()` offre une lecture encore plus perfectionnée du fait qu'elle s'applique à des tableaux multidimensionnels indicés ou même associatifs d'une manière plus simple que les méthodes précédentes.

Il suffit pour cela d'utiliser deux boucles `while` imbriquées. La première récupère les indices de chacun des éléments du tableau. Comme chaque élément est lui-même un tableau, la seconde récupère les clés et valeurs contenues dans chacun d'eux.

Le listing 5-10 donne un exemple de lecture de tableaux multidimensionnels, l'un indicé et l'autre associatif. Malgré l'ajout d'un élément après la création du tableau, la lecture est intégrale.

## Exemple 5-10. Lecture de tableaux multidimensionnels à l'aide de la fonction `each()`

```

<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Lecture d'un tableau indicé avec une boucle while</title>
</head>
<body>
<div>

<?php
//*****
//Tableau indicé multidimensionnel
//*****
// Créeation du tableau
$clients = array(
array ("Leparc", "Paris", "35"),

```

```

array("Duroc", "Vincennes", "22"),
array("Denoël", "Saint Cloud", "47"));
// Ajout d'un élément
$clients[7] = array("Duval", "Marseille", "76");
echo "<table border=\"1\"><tbody>";
while($element=each($clients))
{
    echo "<tr><td> élément <b> $element[0] </b></td>";
    while($val=each($element[1]))
    {
        echo "<td><b>$val[1], " </b></td>";
    }
    echo "</tr>";
}
echo " </tbody> </table> <hr />";
//*****
//Tableau associatif multidimensionnel
//*****
// Création du tableau
$clients = array(
array("client1"=>"Leparc", "ville1"=>"Paris", "age1"=>"35"),
array("client2"=>"Duroc", "ville2"=>"Vincennes", "age2"=>"22"),
array("client3"=>"Denoël", "ville3"=>"Saint Cloud", "age3"=>"47"));
// Ajout d'un élément
$clients[7] = array("client7"=>"Duval", "ville7"=>"Marseille", "age7"=>"76");
echo " <table border=\"1\"><tbody> ";
// Lecture des éléments
while($element=each($clients))
{
    echo "<tr><td> élément <b> $element[0] </b></td>";
    while($val=each($element[1]))
    {
        echo "<td> clé :<b>$val[0], " </b></td><td><b>$val[1], " </b></td>";
    }
    echo "</tr>";
}
echo " </tbody> </table>";
?>
</div>
</body>
</html>

```

La figure 5-3 illustre le résultat de ce listing sous la forme d'un tableau HTML affichant à la fois les indices, ou les clés selon le cas, et toutes les valeurs contenues dans les tableaux \$clients.

élément 0	Leparc	Paris	35
élément 1	Duroc	Vincennes	22
élément 2	Denoël	Saint Cloud	47
élément 7	Duval	Marseille	76

élément 0	clé :client1	Leparc	clé :ville1	Paris	clé :age1	35
élément 1	clé :client2	Duroc	clé :ville2	Vincennes	clé :age2	22
élément 2	clé :client3	Denoël	clé :ville3	Saint Cloud	clé :age3	47
élément 7	clé :client7	Duval	clé :ville7	Marseille	clé :age7	76

Terminé

**Figure 5-3**

*Lecture de tableaux multidimensionnels à l'aide de la fonction each*

## *Lire avec each() et list()*

La fonction `list()` permet d'affecter à  $N$  variables la valeur des  $N$  premiers éléments d'un tableau indicé. Sa syntaxe est la suivante :

```
list($x, $y, $z, ...) = $tab
```

La variable `$x` prend la valeur du premier élément du tableau `$tab` (d'indice `0` ou de première clé). `$y` prend la valeur du deuxième élément, et ainsi de suite.

Le code suivant :

```
$stab=array("Paris","London","Brüssel");
list($x,$y) = $stab;
echo "Les deux premiers éléments sont : $x et $y <br />";
```

affiche uniquement les valeurs "Paris" et "London", sans les indices, qui ne sont pas récupérés.

La fonction `list()` ne déplace pas le pointeur interne du tableau sur les éléments suivants. Si vous appelez de nouveau `list($x,$y)`, vous obtenez les mêmes valeurs.

Dans le code suivant :

```
list($x,$y) = $tab;
```

si la variable `$tab` est un tableau multidimensionnel, les variables `$x` et `$y` contiendront les deux premiers tableaux qui composent `$tab`.

L'intérêt de cette fonction dans la lecture des tableaux peut donc paraître limité. Si vous l'associez cependant à la fonction `each()`, son rôle appréciable devient plus évident. En effet, `each()` déplace le pointeur interne sur les éléments suivants, tandis que `list()` permet de lire les deux premiers éléments du tableau retourné par la fonction `each()`,

éléments qui contiennent respectivement l'indice et la valeur du tableau à lire.

Si vous écrivez le code :

```
| list($x,$y) = each($tab)
```

la variable \$x contient l'indice ou la clé, et \$y la valeur associée.

L'ensemble `list()` et `each()` placé comme expression booléenne dans une boucle `while` vous permet donc de lire l'intégralité du tableau en récupérant les indices, ou les clés selon les cas.

Le listing 5-11 donne un exemple d'utilisation de la fonction `list()` ainsi que de lecture de tableaux indicés et associatifs.

### Plusieurs virgules

Si vous écrivez :

```
list($x,,$y,,$z) = $tab
```

en plaçant plusieurs virgules de suite, \$x contient bien le premier élément de \$tab, mais \$y contient le troisième et \$z le cinquième. Cette particularité peut se révéler utile, par exemple, pour ne récupérer que les éléments d'indice pair ou impair.

### Attention

La fonction `list()` ne s'applique pas aux tableaux associatifs, desquels elle ne récupère ni clés ni valeurs.

## Exemple 5-11. Lecture avec `list()` et `each()`

```
<?php
//list() avec un tableau indicé
$tab=array("Paris","London","Brüssel");
list($x,$y) = $tab;
echo "Les deux premiers éléments sont : $x et $y <hr />";
// list() avec un tableau associatif (ne fonctionne pas)
$tab=array("France"=>"Paris","Great Britain"=>"London","België"=>"Brüssel");
list($x,$y) = $tab;
echo "Les deux premiers éléments sont : $x et $y <hr />";
//*****
//Lecture de tableau indicé
//*****
$tab=array("Paris","London","Brüssel");
while(list($indice,$valeur) = each($tab) )
{
echo "L'élément d'indice <b>$indice</b> a la valeur <b>$valeur</b><br>";
}
echo"<hr />";
//*****
//Lecture de tableau associatif
//*****
$tab=array("France"=>"Paris","Great Britain"=>"London",
"België"=>"Brüssel");
while(list($cle,$valeur) = each($tab) )
{
echo "L'élément de clé <b>$cle</b> a la valeur <b>$valeur</b><br />";
```

```
}
```

```
?>
```

La lecture des tableaux affiche les résultats suivants :

---

```
Les deux premiers éléments sont : Paris et London
Notice: Undefined offset: 1 in c:\eyrolles\php5\tableaux\tableau11.php on line 8
```

```
Notice: Undefined offset: 0 in c:\eyrolles\php5\tableaux\tableau11.php on line 8
```

```
Les deux premiers éléments sont : et
```

```
L'élément d'indice 0 a la valeur Paris
```

```
L'élément d'indice 1 a la valeur London
```

```
L'élément d'indice 2 a la valeur Brüssel
```

```
L'élément de clé France a la valeur Paris
```

```
L'élément de clé Great Britain a la valeur London
```

```
L'élément de clé Belgïë a la valeur Brüssel
```

---

Remarquez l'avis d'erreur si vous utilisez la fonction `list()` seule pour un tableau associatif.

## ***L'instruction foreach***

Plus pratique encore que les méthodes précédentes, l'instruction `foreach()` n'est utilisable qu'à partir des versions 4 de PHP. Elle se révèle particulièrement efficace pour les tableaux associatifs mais fonctionne également pour les tableaux indicés.

Contrairement à la boucle `for`, l'instruction `foreach()` ne nécessite pas de connaître par avance le nombre d'éléments du tableau à lire. Sa syntaxe varie en fonction du type de tableau.

Pour les tableaux indicés, vous écrivez le code suivant :

```
foreach($tab as $valeur)
{
//bloc de code utilisant les valeurs de la variable $valeur;
}
```

Indispensable, le mot-clé `as` permet de récupérer successivement toutes les valeurs des éléments du tableau `$tab` dans la variable `$valeur`, mais sans les indices correspondants.

Pour les tableaux associatifs, vous disposez d'une syntaxe plus perfectionnée.

Le code suivant :

```
foreach($tab as $cle=>$valeur)
{
//bloc de code utilisant les valeurs des variables $cle et $valeur;
}
```

permet de récupérer dans la variable `$cle` les valeurs et les clés successives des éléments. De plus, si le tableau est indicé numériquement, la variable `$cle` contient cet indice.

Si vous disposez d'un serveur équipé des versions 4 et suivantes de PHP, ces méthodes de lecture sont particulièrement recommandées du fait de leur simplicité d'écriture et de leur rapidité d'exécution.

Vous allez maintenant envisager un ensemble d'exemples d'utilisation de l'instruction `foreach` appliquée à la lecture de tableaux de formes diverses.

## Lecture de tableaux indicés ou associatifs

Le listing 5-12 effectue une lecture de tableaux indicés et associatifs à l'aide de l'instruction `foreach` avec et sans récupération des indices ou clés des éléments. Le résultat est similaire à celui obtenu avec les fonctions `list()` et `each()` de la section précédente, mais le code est plus élégant.

### Exemple 5-12. Lecture de tableaux à l'aide de l'instruction `foreach`

```
<?php
//*****
//Lecture de tableau indicé sans récupération des indices
//*****
$tab=array("Paris","London","Brüssel");
echo "<H3>Lecture des valeurs des éléments </H3>";
foreach($tab as $ville)
{
echo "<b>$ville</b> <br>";
}
echo"<hr>";
//*****
//Lecture de tableau indicé avec récupération des indices
//*****
echo "<h3>lecture des indices et des valeurs des éléments </h3>";
foreach($tab as $indice=>$ville)
{
echo "L'élément d'indice <b>$indice</b> a la valeur <b>$ville</b><br>";
}
echo"<hr>";
//*****
//Lecture de tableau associatif avec récupération des clés
//*****
$tab2=array("France"=>"Paris","Great Britain"=>"London","België"=>"Brüssel");
echo "<h3>lecture des clés et des valeurs des éléments</h3>";
foreach($tab2 as $cle=>$ville)
{
echo "L'élément de clé <b>$cle</b> a la valeur <b>$ville</b> <br>";
}
echo"<hr>";
?>
```

## Lecture d'un tableau multidimensionnel

L'exemple 5-13 illustre la lecture du tableau multidimensionnel et associatif `$clients` répertoriant un ensemble de clients dont chaque élément de premier niveau est lui-même un tableau associatif contenant les caractéristiques de chaque client.

Ce tableau multidimensionnel contient deux boucles `foreach` imbriquées. La première

récupère la clé de chacun des éléments du tableau `$clients` dans la variable `$cle` et son contenu de type `array` dans la variable `$tab`. La seconde boucle lit chaque tableau `$tab` en récupérant chaque clé contenue dans la variable `$key` et la valeur de chaque élément dans la variable `$valeur`. L'affichage se fait dans un tableau HTML.

La figure 5-4 donne un aperçu du résultat affiché par ce script.

### Exemple 5-13. Lecture de tableaux multidimensionnels avec `foreach()`

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Lecture d'un tableau multidimensionnel avec foreach()</title>
</head>
<body>
<div>
<?php
// Création du tableau
$clients = array(
"client 1"=>array("nom 1"=>"Leparc", "ville 1"=>"Paris", "age 1"=>"35"),
"client 2"=>array("nom 2"=>"Duroc", "ville 2"=>"Vincennes", "age 2"=>"22"),
"client 3"=>array("nom 3"=>"Denoël", "ville 3"=>"St Cloud", "age 3"=>"47"));
// Ajout d'un élément
$clients["client 7"] = array("nom 7"=>"Duval", "ville 7"=>"Marseille", "age 7"=>"76");
echo "<table border=\"1\" width=\"100%\"><thead><tr> <th> Client </th><th> Nom </th>
<th> Ville </th><th> Age </th></tr></thead><tbody>";
foreach($clients as $cle=>$tab)
{
    echo "<tr><td align=\"center\"><b> $cle </b></td>";
    foreach($tab as $key=>$valeur)
    {
        echo "<td> $key : <b> $valeur </b></td>";
    }
    echo "</tr>";
}
?>
</tbody>
</table>
</div>
</body>
</html>
```

Client	Nom	Ville	Age
client 1	nom 1 : Leparc	ville 1 : Paris	age 1 : 35
client 2	nom 2 : Duroc	ville 2 : Vincennes	age 2 : 22
client 3	nom 3 : Denoël	ville 3 : St Cloud	age 3 : 47
client 7	nom 7 : Duval	ville 7 : Marseille	age 7 : 76

Figure 5-4

## Lecture d'un tableau multidimensionnel associatif avec foreach()

Si nous voulons ne récupérer que les valeurs d'un tableau multidimensionnel, il est maintenant possible d'associer `list()` et `foreach()` dans ce but. Dans l'exemple ci-dessous, la boucle `foreach` va parcourir le tableau `$clients` et `list()` va en extraire les valeurs dans les variables `$nom`, `$ville` et `$age` correspondant aux colonnes du tableau de la figure 5-4.

```
<?php
$clients=array(
    array("Leparc","Paris","35"),
    array("Duroc","Vincennes","22"),
    array("Denoel","St Cloud","47"),
    array("Duval","Marseille","76")
);
foreach($clients as list($nom,$ville,$age))
{
    echo "$nom $ville $age <br/>";
}
?>
```

# Manipuler des tableaux

PHP dispose d'un grand nombre de fonctions permettant d'effectuer toutes sortes de manipulations de tableaux existants. Citons notamment l'extraction, l'ajout ou la suppression d'une partie des éléments, la fusion ou l'intersection de plusieurs tableaux, diverses opérations de tri des éléments ou des clés ou encore l'application d'une fonction à l'ensemble des éléments.

## Extraire une partie d'un tableau

À partir d'un tableau donné, il est possible de créer un nouveau tableau comme sousensemble du tableau initial et ne contenant qu'un nombre déterminé de ses éléments. Cette opération est réalisée à l'aide de la fonction `array_slice()`, qui permet d'effectuer divers types d'extractions.

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

```
$sous_tab = array_slice(array $tab,int ind, int nb)
```

Cette fonction ne modifie pas le tableau initial mais retourne le sous-tableau dans la variable `$sous_tab`. Sa manipulation pouvant se révéler relativement complexe en fonction des valeurs des paramètres `ind` et `nb`, nous envisageons ci-après tous les cas possibles :

- Si `ind` et `nb` sont positifs, le tableau `$sous_tab` contient `nb` éléments du tableau initial extrait en commençant à l'indice `ind`.

Par exemple, `array_slice($tab,2,3)` retourne un tableau comprenant trois éléments extraits à partir de l'indice 2. Il contient donc les éléments d'indice 2, 3 et 4 du

tableau \$tab.

- Si le paramètre `ind` est négatif et que `nb` est positif, le compte des éléments se fait en partant de la fin du tableau `$tab`, le dernier se trouvant affecté virtuellement de l'indice `-1`, l'avant-dernier de l'indice `-2`, et ainsi de suite. Le paramètre `nb` désigne encore le nombre d'élément à extraire.

Par exemple, `array_slice($tab,-5,4)` retourne quatre éléments de `$tab` extraits en commençant au cinquième à partir de la fin.

- Si `ind` est positif et `nb` négatif, le tableau `$sous_tab` contient les éléments de `$tab` extraits en commençant à l'indice `ind` et en s'arrêtant à celui qui a l'indice négatif virtuel `nb` (toujours en commençant par la fin).

Par exemple, `array_slice($tab,2,-4)` retourne tous les éléments à partir de l'indice 2 jusqu'à la fin, sauf les quatre derniers.

- Si `ind` et `nb` sont négatifs, le tableau `$sous_tab` contient les éléments de `$tab` extraits en commençant à l'indice négatif `ind` et en s'arrêtant à celui d'indice négatif `nb`.

Par exemple, `array_slice($tab,-5,-2)` retourne trois éléments compris entre les indices virtuels `-5` compris et `-2` non compris.

La mise en pratique de l'exemple 5-14 donne deux types d'utilisation de la fonction `array_slice()`. Le premier n'utilise que des paramètres positifs et s'applique à un tableau multidimensionnel dont les éléments sont des tableaux indicés ou associatifs. Le second envisage toutes les possibilités de valeurs pour les paramètres `ind` et `nb` en les appliquant à un tableau simple.

## Exemple 5-14. Utilisation de la fonction `array_slice()`

```
<?php
echo "Exemple 1<br />";
$tab= array("UN"=>range(1,5),"DEUX"=>range("a","c"),range("A","E"),range(11,15));
echo "Structure du tableau initial :<br />";
print_r($tab);
echo "<hr />";
$soustab = array_slice($tab,1,2);
echo "array_slice(\$tab,1,2) donne : ";
print_r($soustab);
echo "<hr />";
echo "Exemple 2<br />";
$heros= array("Spock","Batman","Dark Vador","Hal","Frodo","Sky Walker","Amidala",
"Alien");
echo "Structure du tableau initial :<br />";
print_r($heros);
echo "<hr />";
//Extrait des 5 premiers
echo "array_slice(\$heros,0,5)";
$prem=array_slice($heros,0,5);
print_r($prem);
echo "<hr />";
//Extrait des 5 derniers (le dernier est considéré comme ayant l'indice -1 et non
pas 0)
$der=array_slice($heros,-5,5);
```

```

echo"array_slice(\$heros,-5,5)    ";
print_r($der);
echo "<hr />";
//Extrait de 3 noms en commençant à la position -5
$der=array_slice($heros,-5,3);
echo"array_slice(\$heros,-5,3)    ";
print_r($der);
echo "<hr />";
//Extrait des éléments de l'indice 1 jusqu'à la fin hormis les deux derniers
$der=array_slice($heros,1,-2);
echo"array_slice(\$heros,1,-2)    ";
print_r($der);
echo "<hr />";
//Extrait des éléments de l'indice -5 jusqu'à la fin hormis les deux derniers
$der=array_slice($heros,-5,-2);
echo"array_slice(\$heros,-5,-2)    ";
print_r($der);
echo "<hr />";
?>

```

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)
```

Cette fonction ajoute également au tableau `$tab` les  $N$  éléments passés en paramètres mais cette fois au début du tableau. Les indices existants sont tous décalés de la valeur  $N$ , et la fonction retourne le nouveau nombre d'éléments du tableau.

Réciproquement, vous pouvez supprimer des éléments d'un tableau à l'aide de la fonction suivante :

```
| array_pop($tab)
```

qui supprime le dernier élément du tableau `$tab` et retourne cet élément s'il existe ou la valeur `NULL` dans le cas contraire, par exemple si le tableau est vide ou si le paramètre `$tab` n'est pas de type `array`. Avec les fonctions `array_push()` et `array_pop()`, le tableau se comporte comme une pile dotée respectivement de fonctions d'empilement et de dépilement.

Dans ce cas, un avertissement du type :

```
| Warning: array_pop(): The argument should be an array in c:\eyrolles\php5\tableaux\tableau15.php on line 18
```

est affiché, ce qui n'est pas du meilleur effet sur les utilisateurs. Dans le doute, il est préférable d'utiliser la fonction `gettype()` pour s'assurer que le paramètre `$tab` est bien de type `array`.

Pour supprimer le premier élément d'un tableau, utilisez la fonction suivante :

```
| array_shift($tab)
```

qui retourne la valeur de l'élément supprimé.

Enfin, il est possible de supprimer un élément d'indice ou de clé quelconque du tableau

`$tab` à l'aide de la fonction `unset()` en précisant explicitement le nom de l'élément et son indice ou sa clé.

Par exemple :

```
| unset($tab[4])  
| supprime l'élément d'indice 4 du tableau $tab et  
| unset($tab["quatre"])  
l'élément dont la clé est "quatre".
```

Cette fonction n'a pas d'effet sur les autres indices du tableau, qui conservent tous la valeur qu'ils avaient avant la suppression.

L'exemple 5-15 illustre toutes ces fonctions de modification des tableaux et affiche la structure du tableau après chacune d'elles.

## Exemple 5-15. Ajout et suppression d'éléments

```
<?php  
$tab= array(800,1492, 1515, 1789);  
print_r($tab);  
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($tab[4]);  
echo "L'élément d'indice 4 a été supprimé <br>";  
print_r($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.

La fonction `array_unique($tab)` retourne un nouveau tableau ne contenant que la dernière occurrence de chaque valeur présente plusieurs fois dans le tableau `$tab`. Les indices ou les clés associés à chaque élément sont conservés, et le tableau retourné comporte des « trous » dans la suite des indices si ces derniers sont numériques.

Par exemple, le code suivant :

```
<?php
$tab = array("Jacques", "Paul", "Pierre", "Alban", "Paul", "Jack", "Paul");
$tab2 = array_unique($tab);
print_r($tab2);
?>
```

supprime les éléments d'indices 1 et 4 qui ont la valeur "Paul" pour ne conserver que l'élément d'indice 6. Vous obtenez l'affichage suivant de la structure du tableau résultant `$tab2` :

```
Array ( [0] => Jacques [2] => Pierre [3] => Alban [5] => Jack [6] => Paul )
```

---

## ***Opérations sur plusieurs tableaux***

Il est possible d'effectuer différents types d'opérations faisant intervenir plusieurs tableaux, qu'il s'agisse de les fusionner ou d'effectuer des opérations ensemblistes, comme leur intersection ou leur différence.

### **Fusionner des tableaux**

Si des données figurent dans plusieurs tableaux différents, vous pouvez être amené à vouloir effectuer des opérations sur ces différents tableaux afin d'obtenir un tableau unique contenant soit la réunion des éléments de chacun en un seul, soit les éléments communs aux deux, soit encore les éléments présents dans l'un et pas dans l'autre.

Vous pouvez, par exemple, réunir deux ou plusieurs tableaux en un seul à l'aide de la

fonction `array_merge()`, dont la syntaxe est la suivante :

```
| $tab = array_merge($tab1,$tab2,...,$tabN)
```

Cette fonction retourne dans `$tab` l'ensemble des éléments présents dans les tableaux `$tab1`, `$tab2`, ..., `$tabN`. Les tableaux passés en paramètres sont tous sauvegardés tels qu'ils étaient avant l'appel de la fonction.

La fusion des tableaux est réalisée dans les conditions suivantes :

- Si les tableaux à fusionner sont indicés, les éléments du tableau passé en premier paramètre sont conservés, ceux des autres paramètres ayant les indices suivants. Les éléments présents dans plusieurs des paramètres sont présents en double dans le tableau final. Vous pouvez utiliser la fonction `array_unique()` pour les éliminer.
- Si les tableaux à fusionner sont associatifs, les clés et les associations clé-valeur sont préservées. Par contre, si plusieurs des paramètres ont des clés communes, seule l'association clé-valeur du dernier paramètre est conservée, et celle du tableau précédent est perdue.

Pour ne pas perdre les informations correspondant à une même clé, il est possible d'utiliser la fonction `array_merge_recursive()`. Cette dernière n'efface pas la première valeur associée à une clé double mais associe à chaque clé présente plusieurs fois un tableau indicé contenant toutes les valeurs ayant la même clé.

Le listing de l'exemple 5-16 donne deux exemples de fusion de tableaux, le premier pour les tableaux indicés et le second pour les tableaux associatifs. Remarquez la disparition de la valeur "75" associée à la clé "Paris" du tableau `$tabass1`, remplacée par la valeur "Capitale" présente dans le tableau `$tabass2` après la fusion effectuée avec `array_merge()`. La fonction `array_merge_recursive()` préserve les deux valeurs, comme le montre le résultat du script.

## Exemple 5-16. Fusion de tableaux à l'aide de `array_merge()`

```
<?php
// Fusion de tableaux indicés
echo "Tableaux indicés <br>";
$tab1= array("Paris","Lyon","Marseille");
$tab2 = array("Nantes","Orléans","Tours","Paris");
$tab = array_merge($tab1,$tab2);
echo "array_merge donne: ";
print_r($tab);
echo "<hr />";
// Fusion de tableaux associatifs
echo "Tableaux associatifs <br>";
$tabass1= array("Paris" => "75","Lyon" => "69","Marseille" => "13");
$tabass2 = array("Nantes" => "44","Orléans" => "45","Tours" => "37","Paris"
"Capitale");
echo "array_merge donne: ";
$tabass = array_merge($tabass1,$tabass2);
print_r($tabass);
echo "<hr />";
// Fusion
echo "array_merge_recursive donne : ";
```

```
$tabass3 = array_merge_recursive($tabass1,$tabass2);
print_r($tabass3);
?>
```

Le script affiche le résultat suivant, qui montre la structure des tableaux résultant de la fusion :

---

```
Tableaux indicés
array_merge donne: Array ( [0] => Paris [1] => Lyon [2] => Marseille [3] => Nantes [4]
=>
Orléans [5] => Tours [6] => Paris )

Tableaux associatifs
array_merge donne: Array ( [Paris] => Capitale [Lyon] => 69 [Marseille] => 13 [Nantes]
=> 44
[Orléans] => 45 [Tours] => 37 )

array_merge_recursive donne : Array ( [Paris] => Array ( [0] => 75 [1] => Capitale)
[Lyon] => 69
[Marseille] => 13 [Nantes] => 44 [Orléans] => 45 [Tours] => 37 )
```

---

Vous pouvez également créer un tableau associatif à partir de deux autres tableaux. Le premier contient les clés du tableau à créer et le second les valeurs qui sont associées aux clés. Cette opération est réalisable grâce à la fonction `array_combine()` introduite dans PHP 5. Sa syntaxe est la suivante :

```
array array_combine(array $tabcle, array $tabval)
```

Le code suivant :

```
$tabcle = array('F','D','B');
$tabval = array('France','Allemagne','Belgique');
$tabasso = array_combine($tabcle, $tabval);
print_r($tabasso);
```

affiche le résultat :

---

```
Array ( [F] => France [D] => Allemagne [B] => Belgique )
```

---

## Intersection et différence de deux tableaux

Quelques souvenirs de manipulation des ensembles devraient vous permettre de mieux saisir l'utilité des fonctions permettant d'obtenir soit l'intersection de deux tableaux considérés comme des ensembles de valeurs, soit leur différence.

L'intersection de deux ensembles est constituée par les éléments qui appartiennent à la fois aux deux ensembles. La différence de deux ensembles désigne les éléments qui appartiennent au premier et pas au second. En d'autres termes, sont enlevés du premier tableau les éléments qui appartiennent aussi au second.

PHP offre deux fonctions pour réaliser ces opérations sur des tableaux, `array_intersect()` et `array_diff()`.

Pour l'intersection de deux tableaux, la syntaxe de `array_intersect()` est la suivante :

```
| array array_intersect($tab1,$tab2)
```

Cette fonction retourne un tableau contenant tous les éléments communs aux tableaux \$tab1 et \$tab2. Les indices associés aux valeurs du tableau retourné comme résultat correspondent à ceux du tableau passé en premier paramètre. L'inversion de ces deux paramètres ne fournit pas les mêmes indices (voir le résultat du listing de l'exemple 5-17 ci-après).

Pour la différence de deux tableaux, la syntaxe de la fonction `array_diff()` est la suivante :

```
| array_diff($tab1,$tab2)
```

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.

```
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 )
```

---

```
Intersection de $tab1 et $tab2 : Array ( [1] => Jaune [2] => Rouge [4] => Bleu [5] =>
Noir )
Intersection de $tab2 et $tab1 : Array ( [0] => Bleu [1] => Rouge [3] => Noir [4] =>
Jaune )
Différence de $tab1 et $tab2 : Array ( [0] => Blanc [3] => Vert )
Différence de $tab2 et $tab1 : Array ( [2] => Violet [5] => Orange )
```

---

## Trier les éléments d'un tableau

Quand une fonction retourne un tableau de valeurs, comme le font, par exemple, les fonctions de recherche sur une base de données MySQL, les valeurs des éléments apparaissent dans un ordre qui n'est pas nécessairement celui souhaité pour l'affichage des informations.

Pour améliorer la présentation des données, il est souvent utile d'effectuer un tri des valeurs contenues dans le tableau avant de les utiliser pour créer un affichage dans une page web. PHP fournit nombre de fonctions natives permettant de réaliser les opérations de tri les plus diverses. Ces opérations peuvent concerner les valeurs comme les clés des éléments de tableau, aussi bien en ordre alphabétique ASCII, direct ou inversé, que selon l'ordre dit « naturel » ou encore d'après des critères personnalisés définis par le programmeur lui-même.

La quasi-totalité de ces fonctions agit directement sur le tableau qui leur est passé en paramètre en modifiant l'ordre de ses éléments ou de ses clés. Le tableau initial n'est pas récupérable. C'est pour cette raison que les exemples qui suivent créent dès le début du script une copie du tableau initial qui est réutilisée pour chaque fonction.

Certaines fonctions sont plus appropriées à des tableaux indicés et d'autres à des tableaux associatifs. La présentation de ces fonctions est donc divisée en plusieurs sections.

### *Trier des tableaux indicés*

Les fonctions natives offertes par PHP permettent les opérations de tri des éléments des tableaux selon les critères les plus variés.

#### **Trier selon l'ordre ASCII**

L'ordre ASCII n'a rien d'évident pour qui est habitué à l'ordre lexicographique, qui est celui du dictionnaire. Dans un tri ASCII, "rouge" se trouve après "vert" car la lettre "r" se trouve après la lettre "v".

Les fonctions de tri dans l'ordre ASCII proposées par PHP sont les suivantes :

- `array sort($tab)`. Trie les valeurs du tableau `$tab` en ordre croissant des codes ASCII

des caractères qui les composent (donc en tenant compte de la casse des caractères). Les correspondances entre les indices et les valeurs des éléments sont perdues après le tri.

- `array rsort($tab)`. Trie les valeurs du tableau `$tab` en ordre décroissant des codes ASCII des caractères qui les composent. Les correspondances entre les indices et les valeurs des éléments sont perdues après le tri.
- `array array_reverse($tab)`. Inverse l'ordre des valeurs des éléments de `$tab`. Les indices sont évidemment perdus.

L'exemple 5-18 illustre l'emploi de ces fonctions et affiche le tableau initial puis le tableau une fois trié.

### Exemple 5-18. Tri selon l'ordre ASCII

```
<?php
//*****
//TABLEAU INDICÉ
//*****
//Définition du tableau
$stabind=array("Blanc2","Jaune","rouge","Vert","Bleu","Noir","Blanc10");
$copie= $stabind;
echo "<b>Tableau indicé d'origine</b><br />";
print_r($stabind);
//Fonction sort()
echo "<hr />Tri en ordre ASCII sans sauvegarde des indices<br />";
$stabind=$copie;
sort($stabind);
print_r($stabind);
//Fonction rsort()
echo "<hr /> Tri en ordre ASCII inverse sans sauvegarde des indices<br />";
$stabind=$copie;
rsort($stabind);
print_r($stabind);
//Fonction array_reverse()
echo "<hr />Inversion de l'ordre des éléments<br />";
$stabind=$copie;
$stabrev=array_reverse($stabind);
print_r($stabrev);
?>
```

Le script donne les résultats suivants :

---

```
Tableau indicé d'origine
Array ( [0] => Blanc2 [1] => Jaune [2] => rouge [3] => Vert [4] => Bleu [5] => Noir [6]
=>
Blanc10 )
Tri en ordre ASCII sans sauvegarde des indices

Array ( [0] => Blanc10 [1] => Blanc2 [2] => Bleu [3] => Jaune [4] => Noir [5] => Vert
[6] =>
rouge )

Tri en ordre ASCII inverse sans sauvegarde des indices
Array ( [0] => rouge [1] => Vert [2] => Noir [3] => Jaune [4] => Bleu [5] => Blanc2 [6]
=>
```

```
Blanc10 )
Inversion de l'ordre des éléments
Array ( [0] => Blanc10 [1] => Noir [2] => Bleu [3] => Vert [4] => rouge [5] => Jaune
[6] =>
Blanc2 )
```

---

## Trier selon l'ordre naturel

L'ordre dit naturel est plus proche de ce que chacun connaît dans la vie courante. Par exemple, pour un tri en ordre croissant, la chaîne "Blanc10" se trouve après "Blanc2" et "1ZZ" avant "2AA", les chiffres étant considérés comme précédant les lettres.

Il existe deux variantes de fonctions de tri selon l'ordre naturel, suivant qu'il est tenu compte ou non de la casse des caractères :

- `array natsort($tab)`. Trie les valeurs du tableau \$tab selon l'ordre naturel croissant des caractères qui les composent. Le tri étant effectué en tenant compte de la casse, les majuscules sont placées avant les minuscules, par exemple "Vert" avant "rouge". Les correspondances entre les indices ou les clés et les valeurs des éléments sont sauvegardées après le tri, ce qui rend la fonction également applicable aux tableaux associatifs.
- `array natcasesort($tab)`. Trie les valeurs du tableau \$tab selon l'ordre naturel croissant, sans tenir compte de la casse, ce qui correspond davantage à l'ordre courant du dictionnaire, dans lequel "rouge" se trouve avant "Vert". Les correspondances entre les indices ou les clés et les valeurs des éléments sont sauvegardées après le tri.

### **for ou foreach ?**

Du fait que les fonctions `array natsort()` et `array natcasesort()` conservent les correspondances entre les indices ou les clés et les valeurs, il est déconseillé d'utiliser une boucle `for` pour lire l'ensemble des données, au risque de perdre l'ordre créé par le tri. Une boucle `foreach` est indispensable, même pour des tableaux indicés.

### **Exemple 5-19. Tri selon l'ordre naturel**

```
<?php
//*****
//TABLEAU INDICÉ
//*****
// Définition du tableau
$tabind=["Blanc2","Jaune","rouge","Vert","Bleu","Noir","Blanc10","1ZZ","2AA"];
$copie= $tabind;
echo "<b>Tableau indicé d'origine</b><br />";
print_r($tabind);
//*****
echo "<hr />Tri en ordre naturel avec sauvegarde des indices<br />";
$tabind=$copie;
natsort($tabind);
print_r($tabind);
//*****
echo "<hr />Tri en ordre naturel insensible à la casse avec sauvegarde des indices<br />
```

```

">>";
$tabind=$copie;
natcasesort($tabind);
print_r($tabind);
foreach ($tabind as $cle=>$val)
{
echo "<br />$cle => $val ";
}
?>

```

Les résultats des tris sont les suivants :

---

```

Tableau indicé d'origine
Array ( [0] => Blanc2 [1] => Jaune [2] => rouge [3] => Vert [4] => Bleu [5] => Noir [6]
=>
Blanc10 [7] => 1ZZ [8] => 2AA )

```

```

Tri en ordre naturel avec sauvegarde des indices
Array ( [7] => 1ZZ [8] => 2AA [0] => Blanc2 [6] => Blanc10 [4] => Bleu [1] => Jaune [5]
=> Noir
[3] => Vert [2] => rouge )

```

```

Tri en ordre naturel insensible à la casse avec sauvegarde des indices
Array ( [7] => 1ZZ [8] => 2AA [0] => Blanc2 [6] => Blanc10 [4] => Bleu [1] => Jaune [5]
=> Noir
[2] => rouge [3] => Vert )

```

---

## Trier selon un critère personnel

Si les fonctions de tri précédentes ne nous conviennent pas, vous pouvez définir vous-même un critère de tri. Il vous suffit pour cela de créer une fonction de comparaison. Cette dernière effectue un test contenant un opérateur impliquant une relation d'ordre sur les valeurs des éléments.

### Pour en savoir plus

Pour savoir comment définir une fonction personnalisée, reportez-vous au chapitre 7.

La fonction doit retourner une valeur entière positive, négative ou nulle selon le résultat du test. Vous choisirez généralement les valeurs `1`, `-1` et `0`, mais vous pourriez tout aussi bien choisir `N`, `-N` et `0`). Le tri s'effectue dans les conditions suivantes :

- Si le test est évalué à `TRUE` et que la fonction retourne un nombre négatif, les éléments sont triés dans l'ordre défini par le critère du test.
- Si le test est évalué à `TRUE` et que la fonction retourne un nombre positif, les éléments sont triés dans l'ordre inverse de celui défini par le critère du test.
- Si le test est évalué à `TRUE` et que la fonction retourne `0`, les éléments sont de même rang dans l'ordre du test.

Dans l'exemple 5-20, la fonction de test `long()`, dont le code figure ci-dessous, compare la longueur des chaînes de caractères à l'aide de la fonction `strlen()`. Si la chaîne

contenue dans \$mot1 est plus longue que celle contenue dans \$mot2, la fonction retourne -1, et \$mot1 est placé avant \$mot2.

```
function long($mot1,$mot2)
{
    if(strlen($mot1)==strlen($mot2)) return 0;
    elseif(strlen($mot1)>strlen($mot2)) return -1;
    else return 1;
}
```

La fonction de tri utilisable pour les tableaux indicés est la suivante :

```
void usort($tab,"nom_fonction")
```

Elle trie les valeurs des éléments de \$tab selon le critère défini dans la fonction dont le nom est passé en second paramètre. Les associations entre les indices ou les clés du tableau et les valeurs ne sont pas sauvegardées. La fonction ne retourne aucune valeur et agit sur le tableau initial, lequel est donc perdu.

Le listing de l'exemple 5-20 définit une fonction de tri selon la longueur des chaînes, utilise la fonction usort() et affiche le tableau trié.

## Exemple 5-20. Tri personnalisé

```
<?php
//*****
// TRI SUR UN CRITÈRE PERSONNALISÉ
//*****
// Définition de la fonction de tri
function long($mot1,$mot2)
{
    if(strlen($mot1)==strlen($mot2)) return 0;
    elseif(strlen($mot1)>strlen($mot2)) return -1;
    else return 1;
}
// Tableau à trier
$tab=["Blanc","Jaune","rouge","Vert","Orange","Noir","Emeraude"];
// Utilisation de la fonction de tri
echo "Tri selon la longueur des chaînes de caractères<br>";
echo "Tableau initial<br />";
print_r($tab);
usort($tab,"long");
echo "<br />Tableau trié selon la longueur décroissante des mots<br />";
print_r($tab);
?>
```

Nous obtenons le résultat suivant :

---

Tri selon la longueur des chaînes de caractères

```
Tableau initial
Array ( [0] => Blanc [1] => Jaune [2] => rouge [3] => Vert [4] => Orange [5] => Noir
[6] =>
Emeraude )
```

```
Tableau trié selon la longueur décroissante des mots
Array ( [0] => Emeraude [1] => Orange [2] => Blanc [3] => rouge [4] => Jaune [5] =>
Vert [6] =>
```

## Mélanger les valeurs de façon aléatoire

Certaines applications gèrent des nombres aléatoires, pour des tirages au sort, par exemple. D'autres affichent des informations dans un ordre différent pour chaque visiteur. Dans tous ces cas, vous pouvez utiliser un tableau contenant les informations à afficher et mélanger ces éléments de manière aléatoire au moyen de la fonction `shuffle()`, dont la syntaxe est la suivante :

```
| void shuffle(array $tab)
```

Cette fonction modifie le tableau `$tab`, dont les valeurs des éléments sont mélangées de façon aléatoire. Là encore, le tableau initial est perdu.

Il est recommandé d'initialiser le générateur de nombres aléatoires de PHP en appelant la fonction `srand()` avec un paramètre entier avant son utilisation. Les associations des indices et des valeurs ne sont pas sauvegardées.

### Exemple 5-21. Mélange aléatoire des éléments

```
<?php
//Création du tableau de nombres
$tab = range(1,10);
echo "<h4> Tableau initial</h4>";
print_r($tab);
echo "<h4> Mélange en ordre aléatoire</h4>";
//Initialisation du générateur de nombres aléatoires
srand(time());
//Mélange des éléments puis affichage du tableau
shuffle($tab);
print_r($tab);
?>
```

Le listing affiche le résultat suivant :

```
Tableau initial
Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 [5] => 6 [6] => 7 [7] => 8 [8] =>
9 [9] => 10 )
Mélange en ordre aléatoire
Array ( [0] => 4 [1] => 7 [2] => 5 [3] => 6 [4] => 8 [5] => 10 [6] => 1 [7] => 3 [8] =>
9 [9] => 2 )
```

Le mélange étant aléatoire, vous avez évidemment peu de chance de retrouver le même si vous testez le script, le paramètre passé à `srand()` étant l'instant présent en secondes fourni par la fonction `time()`.

## Trier des tableaux associatifs

Les fonctions que vous venez d'étudier pourraient s'appliquer également aux tableaux associatifs, mais elles ont l'inconvénient de ne pas conserver les associations entre les clés et les valeurs, ce qui, pour ce genre de tableau, est rédhibitoire. Vous pourrez donc

trier soit les clés soit les valeurs.

## Trier des valeurs

Les fonctions suivantes sont spécialement dédiées aux tableaux associatifs car elles préservent toutes les associations entre les clés et les valeurs :

- void `asort(array $tab)`. Trie les valeurs du tableau `$tab` selon l'ordre croissant des codes ASCII des caractères qui les composent en préservant les associations clévaleur.
- void `arsort(array $tab)`. Trie les valeurs du tableau `$tab` selon l'ordre décroissant des codes ASCII des caractères qui les composent en préservant les associations clévaleur.
- void `uasort(array $tab, string "nom_fonction")`. Joue le même rôle que la fonction `usort()`, déjà présentée pour les tableaux indicés, en effectuant un tri selon le critère défini dans la fonction de comparaison. Elle conserve en outre les clés associées aux valeurs.

Les fonctions `natsort()` et `natcasesort()`, qui permettent de trier les éléments selon l'ordre naturel respectivement en tenant compte ou pas de la casse, sont également utilisables dans les tableaux associatifs car elles préservent les associations entre les clés et les valeurs.

### Exemple 5-22. Tri de tableaux associatifs

```
<?php
//TABLEAUX ASSOCIATIFS
$tabass= array("white2"=>"Blanc2","yellow"=>"Jaune","red"=>"rouge","green"=>"Vert",
               "blue"=>"Bleu","black"=>"Noir","white10"=>"Blanc10");
$copieass=$tabass;
echo "<h4>Tableau associatif d'origine</h4>";
print_r($tabass);
echo "<h4>Tri en ordre alpha des valeurs avec sauvegarde des clés</h4>";
$tabass=$copieass;
asort($tabass);
print_r($tabass);
echo "<h4>Tri en ordre alpha inverse avec sauvegarde des clés</h4>";
$tabass=$copieass;
arsort($tabass);
print_r($tabass);
echo "<h4>Tri en ordre naturel avec sauvegarde des clés</h4>";
$tabass=$copieass;
natsort($tabass);
print_r($tabass);
echo "<h4>Tri en ordre naturel insensible à la casse avec sauvegarde des clés</h4>";
$tabass=$copieass;
natcasesort($tabass);
print_r($tabass);
?>
```

Le listing affiche le résultat suivant :

```
Array ( [white2] => Blanc2 [yellow] => Jaune [red] => rouge [green] => Vert [ blue] => Bleu  
[black] => Noir [white10] => Blanc10 )  
Tri en ordre ASCII des valeurs  
Array ( [white10] => Blanc10 [white2] => Blanc2 [ blue] => Bleu [yellow] => Jaune  
[black] =>  
Noir [green] => Vert [red] => rouge )  
Tri en ordre ASCII inverse  
Array ( [red] => rouge [green] => Vert [black] => Noir [yellow] => Jaune [ blue] => Bleu  
[white2] => Blanc2 [white10] => Blanc10 )  
Tri en ordre naturel avec sauvegarde des clés  
Array ( [white2] => Blanc2 [white10] => Blanc10 [ blue] => Bleu [yellow] => Jaune  
[black] =>  
Noir [green] => Vert [red] => rouge )  
Tri en ordre naturel insensible à la casse avec sauvegarde des clés  
Array ( [white2] => Blanc2 [white10] => Blanc10 [ blue] => Bleu [yellow] => Jaune  
[black] =>  
Noir [red] => rouge [green] => Vert )
```

---

## Trier les clés

La caractéristique des tableaux associatifs est d'utiliser une clé à la place d'un indice numérique. Il peut dès lors être utile d'opérer des tris non plus sur les valeurs mais sur les clés des éléments pour afficher les résultats de différentes façons.

La fonction suivante :

```
| boolean ksort(array $tab)
```

trie les clés de \$tab selon l'ordre croissant des codes ASCII des caractères. Les associations clé-valeur sont conservées. Cette fonction retourne une valeur booléenne indiquant si l'opération de tri a réussi ou non.

La fonction suivante permet de trier les clés selon l'ordre décroissant des codes ASCII des caractères :

```
| boolean krsort(array $tab)
```

Elle conserve également les associations clé-valeur et retourne une valeur booléenne indiquant si l'opération de tri a réussi ou non.

Comme pour les tableaux indicés, la fonction suivante :

```
| void uksort(array $tab ,string "nom_fonction")
```

permet de trier les clés des éléments de \$tab selon le critère personnalisé défini dans la fonction passée en second paramètre. La fonction de tri doit retourner -1 ou 1 selon que le critère est vérifié ou non ou 0 en cas d'égalité dans les mêmes conditions que la fonction usort() présentée précédemment.

L'exemple 5-23 utilise ces fonctions.

### Exemple 5-23. Tri des clés

```
| <?php
```

```

//*****
//TABLEAU ASSOCIATIF
$tabass= array("white2"=>"Blanc2","yellow"=>"Jaune","red"=>"rouge""green"=>"Vert",
"blue"=>"Bleu","black"=>"Noir","white10"=>"Blanc10");
$copieass=$tabass;
echo "<h4>Tableau associatif d'origine</h4>";
print_r($tabass);
echo "<h4>Tri en ordre ASCII des clés</h4>";
$tabass=$copieass;
ksort($tabass);
print_r($tabass);
echo "<h4>Tri en ordre ASCII inverse des clés</h4>";
$tabass=$copieass;
krsort($tabass);
print_r($tabass);
//*****
//TRI SUR UN CRITÈRE PERSONNALISÉ
//*****
function long($mot1,$mot2)
{
    if(strlen($mot1)>strlen($mot2)) return -1;
    elseif(strlen($mot1)<strlen($mot2)) return 1;
    else return 0;
}
echo "<h4>Tri selon la longueur des clés </h4>";
uksort($tabass,"long");
print_r($tabass);
?>

```

Le listing fournit les résultats ci-dessous, dans lesquels le tri des clés se fait selon l'ordre ASCII et non pas naturel. C'est pour cette raison qu'il est préférable que les clés aient toutes la même casse lors de la création du tableau.

Il est possible de transformer la casse des clés avant le tri en appliquant au tableau la fonction `array_change_key_case()`, dont la syntaxe est la suivante :

```
| array array_change_key_case (array $tab, int CTE)
```

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;
    }
}
//Utilisation de array_filter()
$select=array_filter($villes,"init");
print_r($select);
?>
```

Ce script retourne le tableau `$select`, dont la structure est la suivante :

---

```
Array ( [0] => Paris [1] => Perpignan [3] => Pau )
```

---

# Appliquer une fonction à un tableau

Si vous souhaitez appliquer une même opération ou fonction à l'ensemble des valeurs des éléments d'un tableau, vous pouvez envisager d'effectuer une boucle `for` ou `while` et d'appliquer la fonction à chacun des éléments lus. Outre l'allongement du code qui en résulterait, cette méthode présenterait l'inconvénient de multiplier les appels à la fonction de calcul et entraînerait une perte de temps.

Une façon plus élégante et plus rapide de parvenir au même résultat consiste à utiliser la

fonction `array_walk()`, qui peut avoir deux syntaxes différentes selon le nombre de paramètres qui lui sont passés.

Avec deux paramètres la syntaxe de la fonction `array_walk()` est la suivante :

```
| int array_walk($tab,"nom_fonction")
```

La fonction dont vous précisez le nom est appliquée à toutes les valeurs des éléments du tableau `$tab`, que ce dernier soit indicé ou associatif. La fonction appliquée aux valeurs doit être une fonction personnalisée et non une fonction native de PHP. Elle doit de surcroît avoir au moins deux paramètres, le premier faisant référence à la valeur de l'élément de tableau à traiter et le second à l'indice ou la clé de cet élément.

Vous pouvez évidemment détourner l'interdiction d'utilisation d'une fonction native en appelant celle-ci dans la fonction personnalisée. Le listing de l'exemple 5-25 utilise cette syntaxe pour afficher sous forme de tableau HTML le tableau de valeurs de la fonction `cos(pi/x)` appliquée à toutes les valeurs du tableau `$tabx`, lequel contient les valeurs entières de 1 à 10.

La figure 5-5 donne le résultat de cet exemple.

Avec trois paramètres, la syntaxe de la fonction `array_walk()` est la suivante :

```
| array array_walk(array $tab,string "nom_fonction",divers param)
```

Elle permet d'utiliser la valeur du troisième paramètre `param` comme troisième paramètre de la fonction `"nom_fonction"`. Vous pouvez de la sorte personnaliser l'affichage, comme dans le deuxième exemple du listing de l'exemple 5-25, qui utilise ce paramètre pour créer la couleur de fond des cellules de la colonne affichant le prix TTC.

## Exemple 5-25. Application d'une fonction aux éléments d'un tableau

```
<?php
//*****
//array_walk() avec deux paramètres
//*****
$tabx= range(1,10);
function tabval($val,$ind)
{
echo "<tr><td><b>", $ind+1,"</b></td><td>".cos (M_PI/$val) ."</td></tr>";
}
echo"<table border=\"3\"><tr>
<td> <h2>tableau de valeurs</h2>
<table border=\"1\" >
<thead><th> x </th><th> cos(pi/x)</th></thead>";
array_walk($tabx,"tabval");
echo"</table></td> ";
//*****
//array_walk() avec trois paramètres
//*****
$prix=array("22"=>"5.50","32.50"=>"19.60","80.00"=>"19.60","319"=>"19.60");
//fonction de calcul ht et ttc
function taxe($taux,$prix,$col)
{
```

```

echo "<tr><td > $prix </td><td > $taux </td><td>". $prix*($taux/100) .  

" </td><td style=\"background-color:$col \">".$prix*(1+$taux/100). "</td></ tr>";  

}  

echo"<td><h2>facture détaillée</h2>  

<table border=\"1\" >  

<thead><th> h.t</th><th> taux</th><th> t.v.a.</th><th> t.t.c..</th></thead>";  

array_walk($prix,"taxe","red");  

echo"</td></tr></table></td>";  

?>

```

PHP propose un autre type de fonction, la fonction `array_reduce()`, non pas pour appliquer une fonction à chacun des éléments d'un tableau, comme précédemment, mais pour retourner un seul résultat à partir de l'ensemble des valeurs contenues dans le tableau. Cela peut permettre, par exemple, de calculer la somme ou le produit de l'ensemble des valeurs du tableau.

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

```
| divers array_reduce(array $tab, string "nom_fonction"[,divers param])
```

Comme le ferait une boucle `for`, elle applique de façon itérative la fonction dont le nom est passé en paramètre à l'ensemble des valeurs du tableau `$tab`. Le troisième paramètre facultatif est considéré comme étant la première valeur du tableau. C'est la valeur retournée par défaut si `$tab` est vide.

Le code de l'exemple 5-26 illustre l'utilisation de cette fonction pour calculer d'abord le produit d'un nombre de valeurs entières en l'appliquant au calcul de la factorielle d'un entier  $N$ .

Le deuxième exemple permet ensuite d'opérer la concaténation répétée de toutes les chaînes de caractères contenues dans les éléments d'un tableau. Il utilise comme troisième paramètre la chaîne de caractères "`Salut à`", qui constituera les premiers mots de tous les résultats retournés, quel que soit le tableau.

### Rappel

La factorielle de  $N$  est notée  $N!$ . Elle est égale à  $1 \times 2 \times 3 \times \dots \times N$ . Voir à ce sujet la section « Les fonctions récursives » au chapitre 7.

**Tableau de valeurs d'une fonction**

x	cos(pi/x)
1	-1
2	6.1230317691119E-17
3	0.5
4	0.70710678118655
5	0.80901699437495
6	0.86602540378444
7	0.90096886790242
8	0.92387953251129
9	0.93969262078591
10	0.95105651629515

**Facture détaillée**

h.t	taux	t.v.a.	t.t.c..
22.00	5.50	1.21	23.21
32.50	19.60	6.37	38.87
80.00	19.60	15.68	95.68
319.00	19.6	62.524	381.524

Terminé

**Figure 5-5**  
*Application de fonctions à des tableaux de données*

### Exemple 5-26. La fonction *array\_reduce()*

```
<?php
//Définition de la fonction produit
function multi($a,$b)
{
if($a==0) $a=1;
return $a*$b;
}
//array_reduce avec deux paramètres
$n=10;
$tabn= range(1,$n);
$prod=array_reduce($tabn, "multi");
echo "<hr />Produit des éléments = factorielle $n = $n! = ",$prod;
//Définition de la fonction de concaténation
function concat($a,$b)
{
$a.=$b;
return $a;
}
// array_reduce avec trois paramètres
$tabch= array("messieurs "," Hulot", " et "," Tati");
$chaine=array_reduce($tabch,"concat","Salut à ");
echo "<hr>Concaténation des éléments : ",$chaine;
?>
```

Le script de l'exemple 5-26 affiche les résultats suivants :

---

```
Produit des éléments = factorielle 10 = 10! = 3628800
Concaténation des éléments : Salut à messieurs Hulot et Tati
```

---

## L'objet *ArrayObject*

### *Création d'un objet tableau*

À l'instar de ASP.Net, PHP 5 introduit un objet prédéfini `ArrayObject` représentant un tableau. Il permet de créer des tableaux qui ne sont pas du type `array` mais `object`. Ces objets possèdent des méthodes qui permettent d'effectuer diverses opérations. Si vous n'êtes pas familiarisé avec les notions d'objet et de méthode reportez-vous au préalable au chapitre 9.

Pour créer un objet tableau, utilisez le mot-clé `new` selon la syntaxe suivante :

```
| $objtab=new ArrayObject();
```

qui appelle le constructeur de l'objet. Le tableau créé est alors vide. Pour lui affecter des éléments au moment de sa création, vous pouvez passer au constructeur un paramètre qui est un tableau PHP classique de type `array` selon le modèle suivant :

```
| $tab = array("Linux","Apache");
| $objtab = new arrayObject($tab);
```

Le tableau passé en paramètre au constructeur peut également être un tableau associatif, selon le modèle suivant (repères ① et ② de l'exemple 5-27) :

```
| $tab = array('a'=>"Linux",'b'=>"Apache"); ←①
| $objtab = new arrayObject($tab); ←②
```

Il est également possible d'ajouter des éléments selon la syntaxe :

```
| $objtab['clé']=valeur
```

Sachant que si on omet la clé, la valeur dans le tableau sera indiquée avec un indice supérieur d'une unité au plus grand déjà existant (repère ③, s'il n'y en a aucun, la première valeur sera insérée avec l'indice 0).

La valeur donnée au nouvel élément peut aussi être un tableau de type `array`, ce qui crée un tableau multidimensionnel.

Enfin, la méthode `append()` permet d'ajouter des éléments au tableau un par un (repères ④ et ⑤). Sa syntaxe est la suivante :

```
| $objtab->append("MySQL");
```

L'élément ajouté a l'indice immédiatement supérieur au dernier existant dans le tableau. Ici, l'élément de valeur "MySQL" aura l'indice 1.

Là aussi, le paramètre de la méthode peut aussi être un tableau selon le modèle :

```
| $objtab->append(array('HTML 5', 'CSS 3'));
```

qui crée également un tableau multidimensionnel.

La boucle `foreach` étant applicable aux objets, elle permet de lire l'ensemble des clés (ou des indices) et les valeurs associées contenues dans le tableau (repère 8).

Les tableaux ainsi créés étant des objets, il est également possible de lire la valeur des éléments en tant que valeurs d'une propriété de l'objet dont le nom est la clé ou l'indice numérique de l'élément de tableau. Au préalable, il faut créer cette possibilité à l'aide de la méthode `setFlags()` selon la syntaxe suivante :

```
| $objtab->setFlags(ArrayObject::ARRAY_AS_PROPS);
```

dans laquelle '`ARRAY_AS_PROPS`' est une constante prédéfinie. Pour annuler cette opération, il faut utiliser la même méthode avec pour paramètre la constante '`ArrayObject::STD_PROP_LIST`'.

En créant ces propriétés, il est ensuite facile d'accéder aux propriétés en lecture et en écriture selon le modèle de l'exercice 5-27 (repères 6 et 7) :

```
| echo $objtab->prop1; ← 6  
| $objtab->prop2=valeur; ← 7
```

Notez que même si la clé '`prop2`' n'existe pas encore dans le tableau, le code (repère 7) crée la propriété et lui affecte une valeur.

L'équivalent de la fonction `count()` applicable au type `array` est la méthode `count()`, qui permet de lire le nombre d'éléments de l'objet tableau (repère 9).

Il est encore possible de récupérer les clés et les valeurs contenues dans l'objet dans un tableau associatif habituel de type `array` en appliquant la méthode `getArrayCopy()` à l'objet (repère 10). L'affichage de ce tableau avec la fonction `print_r()` confirme que la variable obtenue est du type `array` et non plus `object`. Vous pourriez alors appliquer à cette variable une des fonctions applicables aux tableaux étudiées dans les sections précédentes.

Dans le même ordre d'idée, il est possible de remplacer les éléments d'un objet tableau par ceux d'un autre tableau de type `array` grâce à la méthode `exchangeArray()` selon la syntaxe :

```
| $tab1=$objtab->exchangeArray($tab2);
```

Dans ce cas, l'objet possède désormais les éléments du tableau `$tab2` et ses anciens éléments sont récupérés dans la variable `$tab1`.

Vous pouvez vérifier l'existence d'un élément d'indice ou de clé donné en utilisant la méthode `offsetExists()`, qui retourne `TRUE` si l'élément existe et `FALSE` sinon (repère 11) en écrivant par exemple :

```
| if($objtab->offsetExists('a')){echo $objtab['a'];}
```

Pour lire un élément du tableau, connaissant son indice ou sa clé, vous pouvez utiliser la méthode `offsetGet()`, dont la syntaxe est la suivante :

```
| echo $objtab->offsetGet(cle);
```

Le paramètre est un nombre ou une chaîne de caractères selon les cas (repère ⑫).

Pour ajouter un élément en précisant sa clé numérique ou alphabétique, vous devez appeler la méthode `offsetSet()`, dont la syntaxe est la suivante :

```
| $objtab->offsetSet(cle,valeur);
```

Le paramètre `cle` est également un nombre ou une chaîne de caractères selon les cas (repères ⑬ et ⑭). L'affichage du tableau (repère ⑮) permet de contrôler l'insertion des éléments. Ici aussi, le paramètre "valeur" peut être un tableau créant encore une fois un tableau multidimensionnel.

Vous pouvez enfin supprimer un élément au moyen de la méthode `offsetUnset()` en lui donnant comme paramètre l'indice ou la clé de l'élément à supprimer (repère ⑯). Là aussi, vous affichez le tableau pour vérifier la suppression de l'élément (repère ⑰).

## Exemple 5-27. Crédation et manipulation d'un objet `ArrayObject`

```
<?php
$tab = array('a'=>"Linux",'b'=>"Apache"); ← ①
// Crédation de l'objet ArrayObject
$objtab =new ArrayObject($tab); ← ②
$objtab['c']="WAMP";
$objtab[]="PHP 5"; ← ③
$objtab->append("MySQL"); ← ④
$objtab->append("SQLite"); ← ⑤
// Crédation des propriétés
$objtab->setFlags(ArrayObject::ARRAY_AS_PROPS);
echo $objtab->a,"<br />"; ← ⑥
echo "<hr />";
$objtab->c="WampServer"; ← ⑦
echo $objtab->c,"<br />";
$objtab->d="HTML 5";
echo $objtab->d;
echo "<hr />";
// Lecture des éléments
foreach($objtab as $cle=>$val) ← ⑧
{
    echo "$cle : $val<br />";
}
echo "<hr />";
// Affichage du nombre d'éléments
echo "Nombre d'éléments = ",$objtab->count(),"<br />"; ← ⑨
// Copie dans un tableau array
$tab2=$objtab->getArrayCopy(); ← ⑩
print_r($tab2);
echo "<hr />";
// Vérification de l'existence d'un élément d'indice ou de clé donné
$val='a';
if($objtab->offsetExists($val)){echo $objtab['a'],'<br />';} ← ⑪
// Lecture
```

```

echo "L'élément de clé '$val' a la valeur :",$objtab->offsetGet($val), "<br />";← 12
echo "<hr />";
// Ajout d'un élément d'indice ou de clé donné
$objtab->offsetSet(3,'SimpleXML');← 13
$objtab->offsetSet('c','ArrayObject'); ← 14
print_r($objtab); ← 15
echo "<hr />";
// Suppression d'un élément d'indice ou d'une clé donné
$objtab->offsetUnset(3);← 16
print_r($objtab); ← 17
?>

```

Cet exemple affiche les résultats suivants, qui illustrent les diverses possibilités offertes par les méthodes de l'objet `ArrayObject`.

```

Linux
WampServer
HTML 5
a : Linux
b : Apache
c : WampServer
0 : PHP 5
1 : MySQL
2 : SQLite
d : HTML 5
Nombre d'éléments = 7
Array ( [a] => Linux [b] => Apache [c] => WampServer [0] => PHP 5 [1] => MySQL [2]
=> SQLite [d] => HTML 5 )
Linux
L'élément de clé 'a' a la valeur :Linux
ArrayObject Object ( [storage:ArrayObject:private] => Array ( [a] => Linux [b] =>
Apache [c] =>
ArrayObject [0] => PHP 5 [1] => MySQL [2] => SQLite [d] => HTML 5 [3]
=> SimpleXML ) )
ArrayObject Object ( [storage:ArrayObject:private] => Array ( [a] => Linux [b]
=> Apache [c] => ArrayObject [0] => PHP 5 [1] => MySQL [2] => SQLite [d] => HTML 5 ) )

```

Actuellement, il n'existe pas autant de méthodes pour l'objet `ArrayObject` que de fonctions applicables au type `array`. Cependant, le nombre de méthodes va augmenter à l'avenir et atteindra celui des fonctions spécifiques du type `array`.

## Les méthodes de tri des éléments

Comme le type `array`, l'objet `ArrayObject` dispose depuis la version PHP 5.4 de nouvelles méthodes de tri des éléments selon leur valeur, celle des clés ou selon un critère personnalisé appliqué aux valeurs ou aux clés.

### Tris alphabétiques des valeurs

Plusieurs méthodes permettent un tri des valeurs des éléments mais elles ne conservent pas le tableau initial car elles ne retournent aucune variable. Il faut donc effectuer une sauvegarde du tableau de départ pour conserver son ordre.

La méthode `asort()` effectue un tri selon l'ordre ASCII : les majuscules sont placées avant les minuscules et pour les chiffres, le 11 est avant le 2, par exemple. Dans l'exemple 5-28, nous écrivons :

```
| $objtab->asort(); ← 1
```

Dans ce cas, « 11 » est avant « 2 » et les majuscules sont avant les minuscules.

La méthode `natesort()` effectue un tri en ordre « naturel » des valeurs comme pour le type `array` mais en tenant compte de la casse, les majuscules étant alors avant les minuscules. Par exemple :

```
| $objtab->natesort(); ← 2
```

Dans ce cas, « 2 » est avant « 11 » et les majuscules avant les minuscules.

Pour rendre le tri naturel insensible à la casse, il faut appeler la méthode `natecasesort()` pour l'objet :

```
| $objtab->natecasesort(); ← 3
```

Dans ce cas, « 2 » est avant « 11 » et il n'y a plus de distinction entre majuscules et minuscules.

L'exemple 5-28 utilise ces trois méthodes et affiche les résultats suivants :

---

```
TRI ASCII des VALEURS
s : 11 apache
c : 2 WAMP
x : MySQL
11 : PHP 5
2 : linux
e : SQLite
TRI NATUREL des VALEURS
c : 2 WAMP
s : 11 apache
x : MySQL
11 : PHP 5
2 : linux
e : SQLite
TRI NATUREL SANS CASSE des VALEURS
c : 2 WAMP
s : 11 apache
2 : linux
x : MySQL
11 : PHP5
e : sQLite
```

---

## Exemple 5-28. Tris alphabétiques des valeurs

```
<?php
$tab           =           array('2'=>"linux", 's'=>"11
WAMP", '11'=>"PHP5", 'x'=>"MySQL", 'e'=>"sQLite");
// Cr?ation de l'objet ArrayObject
$objtab =new ArrayObject($tab);
echo "TRI &nbsp;ASCII des VALEURS <br />";
```

```

$objtab->asort(); ←①
foreach($objtab as $cle=>$val)
{
    echo "$cle : $val<br />";
}
echo "<hr />";
//*****
echo "TRI &nbsp;NATUREL des VALEURS <br />";
$objtab->natsort(); ←②
foreach($objtab as $cle=>$val)
{
    echo "$cle : $val<br />";
}
echo "<hr />";
//*****
echo "TRI &nbsp;NATUREL SANS CASSE des VALEURS <br />";
$objtab->natcasesort(); ←③
foreach($objtab as $cle=>$val)
{
    echo "$cle : $val<br />";
}
echo "<hr />";
?>

```

## Tri des clés

Nous pouvons effectuer un tri sur la valeur des clés des éléments à l'aide de la méthode `ksort()` selon le modèle suivant :

```
| $objtab->ksort()
```

illustré dans l'exemple suivant :

### Exemple 5-29. Tri des clés

```

<?php
$tab           =           array('2'=>"linux", 'S'=>"11
                           apache", 'c'=>"2
                           WAMP", '11'=>"PHP5", 'x'=>"MySQL", 'e'=>"sQLite");
// Création de l'objet ArrayObject
$objtab =new ArrayObject($tab);
echo "TRI &nbsp; des CLES <br />";
$objtab->ksort(); // 1
foreach($objtab as $cle=>$val)
{
    echo "$cle : $val<br />";
}
echo "<hr />";
?>

```

Il affiche le résultat suivant dans lequel on peut remarquer que les majuscules sont avant les minuscules mais que « 11 » est après « 2 ».

---

TRI des CLES  
S : 11 apache  
c : 2 WAMP  
e : sQLite  
x : MySQL

## Tris personnalisés

Comme nous l'avons vu dans l'exemple 5-20 pour le type `array`, il est maintenant possible d'effectuer un tri selon un critère personnalisé contenu dans une fonction et que nous pouvons appliquer sur les valeurs ou sur les clés d'un objet tableau. Pour trier les valeurs, nous utilisons la méthode `uasort()`, et pour les clés, la méthode `uksort()` :

```
$objtab->uasort('nom_fonction');  
$objtab->uksort('nom_fonction');
```

Dans l'exemple 5-30, nous appliquons ces deux méthodes à un objet tableau après avoir défini une fonction "long" déjà utilisée dans l'exemple 5-20, qui teste la longueur des chaînes. Pour plus de détails sur la manière de coder ce genre de fonction, reportez-vous à l'exemple 5-20.

### Exemple 5-30. Tris personnalisés

```
<?php  
//*****  
// TRI SUR UN CRITÈRE PERSONNALISÉ  
//*****  
// Définition de la fonction de tri  
function long($mot1,$mot2)  
{  
    if(strlen($mot1)==strlen($mot2)) return 0;  
    elseif(strlen($mot1)>strlen($mot2)) return -1;  
    else return 1;  
}  
// Tableau à trier  
$tab = array('abc'=>"Blanc", 'cdef'=>"Outremer", 'c'=>"Rouge", 'abcdef'=>"Vert", 'x'  
=>"Orange", 'ef'=>"Noir");  
// Création de l'objet ArrayObject  
$objtab =new ArrayObject($tab);  
// Utilisation de la fonction de tri  
echo "Tableau initial<br />";  
print_r($objtab);  
$objtab->uasort('long');  
echo "<br />Tri selon la longueur des valeurs<br />";  
print_r($objtab);  
echo "<br />Tri selon la longueur des clés<br />";  
$objtab->uksort('long');  
print_r($objtab);  
?>
```

L'exemple 5-30 affiche les résultats suivants :

---

```
Tableau initial  
ArrayObject Object ( [storage:ArrayObject:private] => Array ( [abc] => Blanc [cdef]  
=> Outremer [c] => Rouge [abcdef] => Vert [x] => Orange [ef] => Noir ) )  
Tri selon la longueur des valeurs  
ArrayObject Object ( [storage:ArrayObject:private] => Array ( [cdef] => Outremer [x]  
=> Orange [c] => Rouge [abc] => Blanc [ef] => Noir [abcdef] => Vert ) )  
Tri selon la longueur des clés  
ArrayObject Object ( [storage:ArrayObject:private] => Array ( [abcdef] => Vert [cdef]
```

```
=>
Outremer [abc] => Blanc [ef] => Noir [c] => Rouge [x] => Orange ) )
```

---

## Mémo des fonctions

Les tableaux, qu'ils soient indicés, associatifs ou multidimensionnels, fournissent un type de donnée riche de possibilités pour le stockage d'informations. Pour bien utiliser ces tableaux, vous devez en avoir une bonne connaissance car vous les retrouverez dans tous les types de scripts.

PHP fournit un grand nombre de fonctions natives permettant les opérations les plus diverses, allant de la création aux manipulations les plus variées. Là encore, la connaissance de ces possibilités est gage d'efficacité dans l'écriture de scripts.

---

```
array_change_key_case(array $tab, int cte)
```

Modifie la casse des clés du tableau `$tab`. La constante `cte` vaut `CASE_UPPER` pour les majuscules ou `CASE_LOWER` pour les minuscules.

---

```
array_chunk(array $tab, int NB [, boolean CLE])
```

Scinde le tableau `$tab` en un tableau multidimensionnel dont chaque élément est un tableau indicé contenant `NB` éléments de `$tab`. Le paramètre `CLE` indique s'il faut conserver les indices initiaux (`TRUE` pour oui).

---

```
array array_combine(array $tabclé, array $tabval)
```

Crée un tableau associatif dont les clés sont les éléments de `$tabclé` et les valeurs ceux des `$tabval`.

---

```
array_count_values(array $tab)
```

Compte le nombre d'occurrences de chaque valeur des éléments de `$tab` et retourne un tableau dont les clés sont les valeurs du tableau `$tab` et les valeurs le nombre d'occurrences.

---

```
array array_diff (array $tab1, array $tab2 [, array $tabN])
```

Retourne un tableau qui est la différence ensembliste des tableaux `$tab1` et `$tab2`. Il contient les éléments de `$tab1` moins ceux qui sont communs aux deux tableaux. Vous pouvez généraliser avec `N` tableaux.

---

```
array array_fill(integer départ, int N, divers valeur)
```

Retourne un tableau dont tous les éléments d'indices compris entre `départ` et `départ+N` ont la valeur contenue dans le paramètre `valeur`.

---

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

Supprime tous les éléments de `$tab` pour lesquels la fonction de tri passée en deuxième paramètre retourne `FALSE`.

---

```
array array_flip(array $tab)
```

Retourne un tableau associatif qui a pour clés les valeurs des éléments de `$tab` et pour valeurs les clés de `$tab`.

---

```
array array_intersect_assoc (array $tab1, array $tab2 [, array $tabN])
```

Retourne un tableau associatif contenant les éléments ayant même clé et même valeur pour les tableaux `$tab1`,

---

`$tab2, ..., $tabN`.

---

```
array array_intersect (array $tab1, array $tab2 [, array $tabN])
```

Retourne un tableau contenant les éléments ayant la même valeur dans les tableaux `$tab1`, `$tab2`, ..., `$tabN`.

---

```
bool array_key_exists (divers cle, array $tab)
```

Retourne `TRUE` si la clé (ou l'indice) de valeur `cle` existe dans `$tab`.

---

```
array array_keys (array $tab [, divers val])
```

Retourne un tableau contenant uniquement les clés de `$tab`. Si le paramètre `val` est précisé, le tableau retourné contient la position des clés ayant la valeur `val`.

---

```
array array_map (string nom_fonction, array $tab1 [, array $tab2,..., $tabN])
```

Applique la fonction indiquée à tous les éléments des tableaux `$tab1`, `$tab2`, ..., `$tabN`.

---

```
array array_merge (array $tab1, array $tab2 [, array $tabN])
```

Retourne un tableau rassemblant tous les éléments des tableaux `$tab1`, ..., `$tabN`. Si les clés sont identiques, c'est celle du dernier tableau qui l'emporte.

---

```
bool array_multisort (array $tab1 [, cte1,cte2 [,... [, array $tabN,cte1,cte2]]])
```

Trie plusieurs tableaux `$tab1`, ..., `$tabN` selon les critères précisés pour chacun d'eux par les constantes `cte1` et `cte2`. Ces constantes ont pour valeurs :

`cte1` `SORT_ASC` : tri en ordre croissant (par défaut) ou `SORT_DESC` : tri en ordre décroissant.

`cte2` `SORT_REGULAR` : comparaison alphabétique ou `SORT_NUMERIC` : comparaison numérique ou `SORT_STRING` : comparaison des chaînes.

---

```
array array_pad (array $tab, int N, divers val)
```

Retourne un tableau contenant les éléments de `$tab` auxquels sont ajoutés plusieurs fois la valeur `val` pour obtenir `N` éléments au total. Si  $N > 0$  les éléments sont ajoutés au début sinon à la fin.

---

```
divers array_pop (array $tab)
```

Supprime le dernier élément de `$tab` et retourne sa valeur.

---

```
int array_push (array $tab, divers $var1[, divers $varN])
```

Ajoute les valeurs `$var1`, ..., `$varN` à la fin du tableau `$tab` et retourne la taille du nouveau tableau.

---

```
divers array_rand (array $tab [, int N])
```

Choisit `N` éléments au hasard dans `$tab` et retourne un tableau contenant les clés des éléments choisis. Si `N` vaut 1 la fonction retourne une chaîne contenant sa clé.

---

```
divers array_reduce (array $tab, string nom_fonction [, int N])
```

Applique la fonction précisée à l'ensemble des éléments de `$tab` et retourne une valeur unique calculée à partir de ces valeurs (par exemple, la somme de tous les éléments). Si le paramètre `N` est passé, il est inclus dans les calculs comme valeur initiale.

---

```
array array_reverse (array $tab [, bool cle])
```

Inverse l'ordre des éléments du tableau. Préserve les clés si le paramètre `cle` vaut `TRUE`.

---

```
mixed array_shift (array $tab)
```

Supprime le premier élément du tableau et retourne cet élément.

---

```
array array_slice (array $tab, int N [, int L])
```

Retourne un tableau contenant `L` éléments extraits de `$tab` et dont le premier a l'indice `N`.

---

```
array array_splice (array $tab, int N [, int L [, array $tab2]])
```

Supprime `L` éléments de `$tab` à partir de l'indice `N` et les remplace éventuellement par ceux du tableau `$tab2`.

---

```
mixed array_sum (array $tab)
```

Retourne la somme des éléments du tableau.

---

```
array array_unique (array $tab)
```

Élimine les doublons et retourne un nouveau tableau.

---

```
int array_unshift (array $tab, divers $var1 [, divers $varN])
```

Ajoute les valeurs contenues dans les variables \$var1, ..., \$varN à la fin du tableau et retourne le nombre d'éléments ajoutés.

---

```
array array_values (array $tab)
```

Retourne un tableau ne contenant que les valeurs du tableau associatif \$tab. Crée des indices numériques de 0 à N.

---

```
bool array_walk (array $tab, string nom_fonction)
```

Applique la fonction indiquée à tous les éléments de \$tab. Si la fonction a plusieurs paramètres, les éléments sont passés comme premier paramètre. Retourne TRUE si l'opération est réussie et FALSE sinon.

---

```
bool arsort (array $tab )
```

Trie le tableau associatif \$tab en ordre décroissant des valeurs et en préservant les clés. Retourne TRUE si l'opération est réussie et FALSE sinon.

---

```
bool asort (array $tab [, int sort_flags])
```

Tri le tableau associatif \$tab en ordre croissant des valeurs et en préservant les clés. Retourne TRUE si l'opération est réussie et FALSE sinon.

---

```
array compact (divers $var1 [,divers $varN])
```

Crée un tableau dont les éléments sont les valeurs des variables \$var1, ..., \$varN. Ces variables peuvent être des tableaux.

---

```
int count (array $tab [, int COUNT_RECURSIVE])
```

Retourne le nombre d'éléments de \$tab. Pour obtenir le nombre d'éléments d'un tableau multidimensionnel, il faut utiliser le deuxième paramètre.

---

```
mixed current (array $tab)
```

Retourne la valeur de l'élément pointé par le pointeur du tableau.

---

```
array each (array $tab)
```

Retourne la clé (ou l'indice) et la valeur de l'élément en cours dans un tableau à quatre éléments. Les éléments d'indice 0 et key contiennent la clé de l'élément de \$tab. Les éléments 1 et value contiennent la valeur associée.

---

```
divers end (array array)
```

Place le pointeur de tableau sur le dernier élément et retourne sa valeur.

---

```
int extract (array $tab [, int N [, string prefix]])
```

Crée des variables dont les noms sont les clés de \$tab et les valeurs celles de ses éléments.

---

```
bool in_array (divers val, array $tab [, bool type])
```

Retourne TRUE si la valeur val est présente dans le tableau \$tab. Si le paramètre type vaut TRUE les types de la valeur recherchée et de l'élément doivent être identiques.

---

```
divers key (array $tab)
```

Retourne la clé de l'élément actuellement pointé dans \$tab.

---

```
bool krsort (array $tab)
```

Trie \$tab selon les clés en ordre décroissant. Les associations clé/valeur sont préservées.

---

```
bool ksort (array $tab)
```

Trie \$tab selon les clés en ordre croissant. Les associations clé/valeur sont préservées.

---

```
void list ($var1,...,$varN)
```

Permet d'affecter une liste de variables avec les éléments du tableau \$tab.

---

```
void natcasesort (array $tab)
```

Trie le tableau en ordre naturel sans tenir compte de la casse.

---

```
divers next (array $tab)
```

Retourne l'élément suivant de \$tab ou FALSE en fin de tableau.

---

```
mixed prev (array $tab)
```

Retourne l'élément précédent de \$tab ou FALSE en fin de tableau.

---

```
array range (int min, int max [, int pas])
```

Crée un tableau contenant une suite d'entiers incrémentée de une unité par défaut, en commençant à la valeur min et en finissant à max. Le paramètre éventuel ne modifie pas cet incrément.

---

```
mixed reset (array $tab)
```

Replace le pointeur de tableau sur le premier élément et retourne cette valeur.

---

```
bool rsort (array $tab [, int sort_flags])
```

Trie les éléments de \$tab en ordre décroissant.

---

```
void shuffle (array $tab)
```

Mélange tous les éléments du tableau au hasard.

---

```
bool sort (array $tab [, int sort_flags])
```

Trie les éléments de \$tab en ordre croissant.

---

```
bool uasort (array $tab, string nom_fonction)
```

Trie les éléments du tableau associatif \$tab en fonction d'un critère de comparaison défini par la fonction personnalisée nom\_fonction. Les clés sont préservées.

---

```
bool uksort (array $tab, string nom_fonction)
```

Trie les clés de \$tab en fonction d'un critère de comparaison défini par la fonction personnalisée nom\_fonction.

---

```
bool usort (array $tab, string nom_fonction)
```

Trie les éléments de \$tab selon un critère de comparaison défini par la fonction personnalisée nom\_fonction. Les clés sont préservées.

## Tableau 5-1 – Les méthodes de l'objet *ArrayObject*

---

```
void append('valeur')
```

Ajoute un élément indicé de valeur donnée (qui peut être un tableau)

---

```
void asort()
```

Trie les éléments du tableau en ordre ASCII

---

```
int count()
```

Retourne le nombre d'éléments du tableau

---

```
array exchangeArray($tab)
```

Remplace les éléments du tableau par ceux de \$tab

---

```
array getArrayCopy($tab)
```

Retourne un tableau de type array qui est une copie de \$tab

---

```
int getFlags()  
Lit la propriété de comportement définie par setFlags()  
void ksort()  
Trie les clés du tableau  
void natcasesort()  
Effectue un tri naturel des éléments, sans prise en compte de la casse  
void natsort()  
Effectue un tri naturel des éléments  
bool offsetExists($cle)  
Retourne TRUE s'il existe un élément de clé précis dans $clé  
divers offsetGet($cle)  
Retourne la valeur de l'élément dont la clé est donnée  
void offsetSet($cle,$valeur)  
Crée un élément dont la clé et la valeur sont données  
void offsetUnset($cle)  
Détruit l'élément dont la clé est passée en paramètre  
void setFlags(STD_PROPS_LIST | ARRAY_AS_PROPS)  
Définit le comportement de l'objet en termes de propriété; avec la constante ARRAY_AS_PROPS les  
éléments du tableau deviennent des propriétés de l'objet  
void uasort('nom_fonction')  
Trie les valeurs selon un critère personnalisé défini par une fonction  
void uksort('nom_fonction')  
Trie les clés selon un critère personnalisé défini par une fonction
```

---

## Exercices

### Exercice 1

Écrivez un tableau multidimensionnel associatif dont les clés sont des noms de personne et les valeurs des tableaux indicés contenant le prénom, la ville de résidence et l'âge de la personne.

### Exercice 2

Écrivez un tableau multidimensionnel associatif dont les clés sont des noms de personne et les valeurs des tableaux associatifs dont les clés sont le prénom, la ville de résidence et l'âge de la personne avec une série de valeurs associées.

### Exercice 3

Utilisez une boucle `foreach` pour lire les tableaux des exercices 1 et 2.

### Exercice 4

Utilisez une boucle `while` pour lire les tableaux des exercices 1 et 2.

## **Exercice 5**

Créez un tableau contenant une liste d'adresses de sites recommandés, puis créez un lien aléatoire vers le premier site de la liste après avoir trié le tableau en ordre aléatoire.

## **Exercice 6**

Créez un tableau d'entiers variant de 1 à 63 puis, à partir de celui-ci, un autre tableau de nombres variant de 0 à 6.3. Créez ensuite un tableau associatif dont les clés X varient de 0 à 6.3 et dont les valeurs sont  $\sin(X)$ . Affichez le tableau de valeurs dans un tableau HTML.

## **Exercice 7**

Créez un tableau contenant une liste d'adresses e-mail. Extrayez le nom de serveur de ces données, puis réalisez des statistiques sur les occurrences de chaque fournisseur d'accès.

## Les formulaires

Les formulaires introduits dans le HTML depuis ses plus anciennes versions sont l'élément essentiel qui permet l'interactivité entre un site et ses visiteurs. Ils constituent pour cette raison la base de la création de sites dynamiques. L'envoi d'informations du poste client vers le serveur *via* le protocole HTTP concerne aussi bien l'envoi des données personnelles d'un internaute qui souhaite passer une commande que le déclenchement d'une requête dans une base de données ou la création de page dynamique en réponse à cette demande.

Tout échange entre visiteur et serveur passe par un formulaire, dans lequel l'utilisateur peut saisir textes ou mots de passe, opérer des choix grâce à des boutons radio, des cases à cocher ou des listes de sélection, voire envoyer ses propres fichiers depuis le poste client. Il est donc important d'en maîtriser la création à la fois avec HTML 5, pour obtenir des formulaires présentables, et avec PHP, pour gérer les informations fournies par le formulaire au script côté serveur.

### Création d'un formulaire HTML

Avant toute chose, il faut créer la structure HTML d'un formulaire.

Pour être conforme au HTML 5, le document contenant le formulaire doit avoir la structure minimale suivante :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Titre de la page</title>
</head>
<body>
<form method="post" action="nomdefichier.php"> ←❶
  <fieldset>←❷
    <legend>Titre du formulaire</legend>
    <!-- Corps du formulaire contenant les différentes composants-->
  </fieldset>
</form>
</body>
```

```
| </html>
```

L'élément `<form>` possède certains attributs obligatoires. D'autres sont utilisés dans des circonstances particulières en fonction des besoins.

L'attribut `action="nom_de_fichier.php"` (repère 1) est obligatoire. Il désigne le fichier qui va traiter, sur le serveur, les informations saisies dans le formulaire. Il est recommandé que ce fichier soit présent dans le même répertoire que celui contenant le formulaire, mais ce n'est pas obligatoire. Si le fichier se trouve dans un autre dossier, voire sur un autre serveur, il faut utiliser une adresse absolue, en écrivant, par exemple :

```
| action= "http://www.funhtml.com/dossier/nom_de_fichier.php"
```

Dans le cadre de ce livre, il s'agira toujours d'un fichier PHP. Il existe d'autres solutions sur des serveurs non-PHP, comme ASP.Net, qui est un concurrent de PHP.

Pour que le fichier qui traite les données soit à coup sûr celui qui contient le formulaire, vous pouvez utiliser la variable `$_SERVER["PHP_SELF"]`, qui contient le nom du fichier en cours d'exécution comme valeur de l'attribut `action`. Pour les versions antérieures à PHP 4.1, il fallait utiliser la variable `$PHP_SELF`. En cas de maintenance du code entraînant le changement du nom du fichier, il n'est pas nécessaire de modifier la valeur de l'attribut `action`.

Vous pouvez avoir, par exemple, le code suivant :

```
| <form method="post" action="<?= $_SERVER["PHP_SELF"] ?>">
```

L'attribut `action="nom_de_fichier.php"` peut aussi avoir la valeur `"mailto:"`, qui provoque l'envoi des données vers une adresse e-mail, qu'il faut préciser à la suite du mot `mailto` en écrivant, par exemple :

```
| action="mailto:nom@funhtml.com"
```

Cette méthode ne peut servir qu'à envoyer des informations vers un e-mail mais pas vers une base de données. Comme elle ne permet pas de les traiter facilement, elle ne nous est pas très utile en PHP.

`method="post|get"` détermine la méthode d'envoi des données vers le serveur. La méthode `get`, qui est la méthode par défaut, présente l'inconvénient d'ajouter les données du formulaire à l'adresse URI du fichier qui les traite, ce qui les rend visibles par le visiteur. Cet inconvénient peut être exploité pour passer des données à un script dès son appel. De plus, il existe une limite à la longueur des URI et donc à la quantité de données à transmettre. Ces problèmes ne se retrouvent pas avec la valeur `post`, que vous utiliserez dans la plupart des cas.

`name="chaine_de_caractères"` attribue un nom au formulaire. Cet attribut est surtout utilisé pour accéder aux éléments du formulaire via un script JavaScript.

`enctype="type_d'encodage"` détermine le type d'encodage des données transmises au serveur. Sa valeur par défaut, `"application/x-www-form-urlencoded"`, est utilisable dans la plupart des cas, à l'exception du transfert de fichiers du poste client vers le serveur, pour lequel elle doit être `"multipart/form-data"`. Si l'attribut `action` a la valeur `"mailto:"`,

l'attribut `enctype` a pour valeur "text/plain" ou "text/html", selon que le contenu est envoyé à une adresse e-mail au format texte ou HTML.

L'élément `<fieldset>` (repère ②) permet, à l'intérieur d'un même formulaire, de créer des blocs visuels contenus entre les balises `<fieldset>` et `</fieldset>` et donc de structurer le formulaire en fonction des champs qu'il contient, ce qui améliore la présentation. L'élément `<legend>` contient le titre de chacun de ces blocs. À l'intérieur de chaque bloc se trouvent les éléments HTML qui créent les champs visibles ou invisibles du formulaire.

Les sections qui suivent rappellent les différents composants HTML d'un formulaire et leurs rôles respectifs.

## **L'élément `<input />`**

La balise unique `<input />` permet de créer les composants classiques des formulaires que vous connaissez déjà et dont les aspects et les rôles sont très différents. Depuis HTML 5, elle permet également de créer de nouveaux composants, plus ergonomiques pour les visiteurs. La différenciation de ces composants s'effectue simplement en définissant la valeur de leurs attributs, et notamment de l'attribut `type`.

L'attribut `name` est obligatoire, car c'est lui qui permet d'identifier les champs côté serveur et ainsi de récupérer leur contenu. Les sections qui suivent détaillent les éléments de type "text", "password", "email", "tel", "date", "number", "checkbox", "radio", "submit", "reset", "file" et "hidden".

### **L'élément `<input type="text" />`**

Cet élément crée un champ de saisie de texte d'une seule ligne. En plus de l'attribut `name`, vous pouvez apporter des précisions supplémentaires à l'aide des attributs suivants :

- `size="nombre"`. Détermine la largeur de la zone en nombre de caractères.
- `maxlength="nombre"`. Détermine le nombre maximal de caractères que l'utilisateur est autorisé à saisir.
- `value="texte"`. Définit un texte par défaut tant que l'utilisateur ne l'a pas modifié. C'est cette valeur qui est transmise au serveur si l'internaute ne saisit aucun texte, comme dans l'exemple ci-dessous :

```
| <input type = "text" name="ville" size="30" maxlength="40" value="Votre ville"/>
```

L'aspect visuel de ce composant est illustré à la figure 6-1 (repère ①), page 168.

#### **L'attribut `value`**

Pour des raisons d'ergonomie, il est préférable que le texte par défaut défini à l'aide de l'attribut `value` s'efface tout seul au moment où l'utilisateur clique dessus car cela lui évite de devoir l'effacer.

Il suffit pour cela d'utiliser une instruction JavaScript très simple :

Pour réagir à l'événement clic :

```
<input type="text" name="ville" size="30" maxlength="40" value="Votre ville"
onclick="this.value=' '" />
```

Pour que le texte s'efface dès que la zone de texte reçoit le focus (le curseur passe dans la zone) :

```
<input type="text" name="ville" size="30" maxlength="40" value="Votre ville"
onfocus="this.value=' '" />
```

## L'élément *<input type="email" />*

Crée un champ de saisie d'adresse e-mail identique à un champ de texte, mais si l'on définit son attribut `pattern`, qui contient un motif d'expression régulière, le navigateur vérifie la validité de la saisie et avertit l'utilisateur en cas d'erreur.

Les attributs `size` et `maxlength` y jouent le même rôle que précédemment.

En voici un exemple :

```
<input type="email" name="mail" size="50" pattern="(^[\a-z0-9]+@[a-z0-9]+\.(.\{2,4\})")/>
```

Ici, les adresses e-mail doivent être de la forme habituelle (`machin34@truc24.com`) avec des suffixes de domaine de deux à quatre lettres (`{2,4}`) que vous pouvez modifier selon les besoins. L'aspect visuel de ce composant est semblable à une zone de texte (figure 6-1 repère ②).

## L'élément *<input type="tel" />*

Crée un champ de saisie de numéro de téléphone qui a aussi l'aspect d'une zone de saisie de texte. On peut également lui ajouter l'attribut `pattern` contenant une expression régulière. Par exemple, nous pouvons écrire :

```
<input type="tel" name="tel" pattern="^0[0-9]{9}" />
```

L'expression régulière utilisée implique que la saisie doit comporter 10 chiffres dont le premier est un zéro, sinon l'erreur est signalée par les navigateurs qui gèrent ce composant. L'aspect visuel de ce composant est aussi semblable à une zone de texte (figure 6-1 repère ③).

## L'élément *<input type="password" />*

Crée un champ de saisie de mot de passe semblable à un champ texte mais dans lequel les caractères saisis sont invisibles et remplacés par des astérisques (figure 6-1, repère ④).

Les attributs `size` et `maxlength` y jouent le même rôle que précédemment.

En voici un exemple :

```
| <input type="password" name="code" size="10" maxlength="6"/>
```

## L'élément **<input type="number" />**

Crée un champ de saisie de nombre pour lequel on peut définir les attributs `min` et `max` afin de créer un intervalle de valeurs autorisées, ainsi que l'attribut `required` pour rendre une saisie obligatoire. Par exemple, nous pouvons écrire :

```
| <input type="number" name="num" min="1" max="10" step="2" />
```

Ici, le nombre saisi doit varier de 1 à 11 par pas de deux unités. L'aspect visuel de ce composant est aussi semblable à une zone de texte (figure 6-1 repère 5).

## L'élément **<input type="date" />**

Crée un champ de saisie de date au format AAA-MM-JJ. Certains navigateurs comme Opera affichent un calendrier qui permet d'éviter les erreurs de saisie. En ajoutant les attributs `min` et `max`, on peut définir des dates minimale et maximale. Par exemple, nous pouvons écrire :

```
| <input type="date" name="ladate" min="2016-09-01" max="2017-12-31" />
```

Ici, la date saisie doit obligatoirement être comprise entre le 1<sup>er</sup> septembre 2016 et le 31 décembre 2017.

Notez qu'il existe des variantes de ce composant qui permettent de saisir l'heure, le mois, la semaine ou encore la date et l'heure simultanément, en remplaçant la valeur de l'attribut `type` par `time`, `month`, `week` et `datetime` respectivement. Certains navigateurs créent une interface qui facilite la saisie en affichant un calendrier complet (figure 6-1 repère 6).

## L'élément **<input type="radio" />**

Crée un bouton radio. Employé seul, un bouton radio peut servir à valider des choix. Employé en groupe, il implique, à la différence des cases à cocher, qu'un seul choix est autorisé. Dans ce cas, tous les boutons radio du groupe doivent avoir une même valeur pour l'attribut "name". Le fait d'en activer un désactive celui qui l'était auparavant.

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 7).

## L'élément **<input type="checkbox" />**

Crée une case à cocher utilisée pour effectuer un ou plusieurs choix parmi ceux qui sont préétablis par le programmeur. L'attribut `value` contient le texte qui sera transmis au

serveur si l'utilisateur coche la case. Il est obligatoire.

Dans les groupes de cases à cocher, il est possible de cocher plusieurs cases simultanément, ce qui n'est pas le cas des cases d'option (boutons radio). Dans ce cas, il faut que le nom de tous les composants soit le même et qu'il soit suivi de crochets ouvrants et fermants de façon à récupérer les valeurs dans un tableau, comme vous le verrez en détail par la suite.

En voici un exemple :

```
| <input type ="checkbox" name="lang[]" value="français" />  
| <input type ="checkbox" name="lang[]" value="anglais" />
```

L'aspect visuel de ce composant est illustré à la figure 6-1 (repère 9).

### 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 13).

### 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 12).

## 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.

L'attribut `name` est obligatoire. Vous pouvez utiliser en plus les attributs `size` limitant la taille de la zone de saisie, et plus particulièrement l'attribut `accept`, qui définit le ou les types de fichier acceptés en transfert. Par sécurité, l'utilisation de cet attribut est recommandée car il permet de limiter le transfert à certains types de fichiers bien précis et de refuser les autres.

Dans l'exemple ci-dessous, le serveur n'accepte que le transfert de fichiers ayant les extensions `.gif` ou `.jpeg` et refuse tout autre type :

```
| <input type="file" name="monfichier" accept="image/gif,image/jpeg" size="30"/>
```

L'aspect visuel de ce composant est illustré à la figure 6-1 (repère 11).

## L'élément `<input type="hidden" />`

Crée un champ caché n'ayant, comme son nom l'indique, aucun rendu visuel dans le formulaire mais permettant de transmettre des informations invisibles pour l'utilisateur.

Les informations sont contenues dans l'attribut `value`. L'attribut `name` est obligatoire.

En voici un exemple :

```
| <input type="hidden" name="MAX_FILE_SIZE" value="20000"/>
```

## L'élément `<textarea>`

À l'instar de l'élément `<input type="text" />` avec l'attribut `type=text`, l'élément `<textarea>` crée un champ de saisie de texte mais, contrairement au précédent, en permettant la saisie sur plusieurs lignes.

Cet élément comporte une balise de fermeture `</textarea>` et un contenu de texte par défaut affiché dans la zone de texte. Les attributs `cols` et `rows`, qui donnent respectivement le nombre de colonnes et de lignes de la zone de texte, doivent être définis.

L'exemple suivant :

```
| <textarea name="commentaire" cols="45" rows="8" >  
| Tapez vos commentaires ici  
</textarea>
```

crée une zone de saisie de texte de 45 colonnes et 8 lignes au maximum.

L'aspect visuel de ce composant est illustré à la figure 6-1 (repère 10).

## L'élément `<select>`

Crée une liste de sélection d'options parmi lesquelles l'utilisateur fait un choix, chaque option devant être définie par un élément `<option>` séparé.

L'élément `<select>` a la structure suivante :

```
<select name="maliste">
  <option value="valeur 1"> Texte choix 1</option>
  <option value="valeur 2"> Texte choix 2</option>
  <option value="valeur 3"> Texte choix 3</option>
</select>
```

Il comporte les attributs suivants :

- `name="nom_select"`. Obligatoire. Donne le nom de la liste.
- `size="Nombre"`. Détermine le nombre de choix visibles simultanément. Par défaut, sa valeur est 1.
- `multiple="multiple"`. Autorise la sélection de plusieurs options simultanément.

L'élément `<option>` comporte les attributs suivants :

- `value="chaine"`. Obligatoire. Définit la valeur transmise au serveur si l'option est sélectionnée.
- `selected="selected"`. Définit l'option qui est sélectionnée par défaut dans la liste si l'utilisateur ne fait pas de choix.

L'aspect visuel de ce composant est illustré à la figure 6-1 (repère 8).

## ***Exemple de code `<form>`***

L'exemple 6-1 fournit le code d'un formulaire contenant tous les éléments possibles répartis en trois groupes :

- saisie des données personnelles du visiteur dans une zone de texte pour le nom (repère 1), puis des composants email et tel (repère 2 et 3), champ de saisie de mot de passe (repère 4), de département par un nombre (repère 5), date de naissance avec le composant date (repère 6), boutons radio (repère 7) et de sélection (repère 8) à quatre options ;
- cases à cocher (repère 9) et zone de texte multiligne (repère 10) ;
- envoi d'un fichier du client vers le serveur (repère 11), champ caché contenant la taille maximale des fichiers transférables et boutons de réinitialisation (repère 12) et d'envoi (repère 13).

### **Exemple 6-1. Formulaire type**

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Formulaire traité par PHP</title>
```

```

<style type="text/css">
fieldset {border: double medium red}
</style>
</head>
<body>
<form action="cible.php" method="post" enctype="application/x-www-form-urlencoded">
<!-- Premier groupe de composants--&gt;
&lt;fieldset&gt;
&lt;legend&gt;&lt;b&gt;Vos coordonnées&lt;/b&gt;&lt;/legend&gt;
&lt;label&gt;Nom : &lt;/label&gt;&lt;input type="text" name="nom" size="40" maxlength="256" /&gt;
&lt;br /&gt;← 1
&lt;label&gt;E-mail : &lt;/label&gt;&lt;input type="email" name="mail" pattern="(^ [a-z0-9]+
@ ([a-z0-9]+(\.) ([a-z]{2,4}))" /&gt;&lt;br /&gt;← 2
&lt;label&gt;Téléphone : &lt;/label&gt;&lt;input type="tel" name="tel" pattern="^0[0-9]{9}" /&gt;
&lt;br /&gt;← 3
&lt;label&gt;Code : &lt;/label&gt;&lt;input type="password" name="code" size="40" maxlength="6" /&gt;
&lt;br /&gt;← 4
&lt;label&gt;Département : &lt;/label&gt;&lt;input type="number" name="num" min="1" max="101" /&gt;
&lt;br /&gt;← 5
&lt;label&gt;Né(e) le : &lt;/label&gt;&lt;input type="date" name="date" min="1960_01_01"
max="1994-01-01" /&gt; &lt;br /&gt;← 6

&lt;input type="radio" name="sexe" value="homme" /&gt; Homme&lt;br /&gt;← 7
&lt;input type="radio" name="sexe" value="femme" /&gt; Femme&lt;br /&gt;
&lt;select name="pays" size="1"&gt;← 8
&lt;option value="France"&gt; France&lt;/option&gt;
&lt;option value="Belgique"&gt; Belgique&lt;/option&gt;
&lt;option value="Suisse"&gt; Suisse&lt;/option&gt;
&lt;option value="Canada"&gt; Canada&lt;/option&gt;
&lt;/select&gt;
&lt;br /&gt;
&lt;/fieldset&gt;
<!-- Deuxième groupe de composants--&gt;
&lt;fieldset&gt;
&lt;legend&gt;&lt;b&gt;Vos goûts&lt;/b&gt;&lt;/legend&gt;
&lt;input type="checkbox" name="pomme" value="pomme" /&gt; Pommes&lt;br /&gt;← 9
&lt;input type="checkbox" name="poire" value="poire" /&gt; Poires&lt;br /&gt;
&lt;input type="checkbox" name="scoubidou" value="scoubidou" /&gt; Scoubidous&lt;br /&gt;
&lt;textarea name="gouts" cols="50" rows="5" onclick="this.value=''"&gt;← 10
Décrivez vos goûts en détail
&lt;/textarea&gt;
&lt;br /&gt;&lt;br /&gt;
&lt;/fieldset&gt;
<!-- Troisième groupe de composants--&gt;
&lt;fieldset&gt;
&lt;legend&gt;&lt;b&gt;Envoyez-nous votre photo&lt;/b&gt;&lt;/legend&gt;
&lt;input type="file" name="fichier" accept="image/jpeg" /&gt;← 11
&lt;input type="hidden" name="MAX_FILE_SIZE" value="10000" /&gt;
&lt;br /&gt;&lt;br /&gt;
&lt;input type="reset" value="Effacer" /&gt;← 12
    &amp;nbsp;&amp;nbsp;&amp;nbsp;&lt;input type="submit" value="Envoyer" /&gt;← 13
&lt;br /&gt;&lt;br /&gt;
&lt;/fieldset&gt;
</pre>

```

```
</form>
</body>
</html>
```

La figure 6-1 fournit une image du formulaire que vous venez de créer avec les repères correspondant au code HTML.

The screenshot shows a web browser window with the title 'Opera'. The address bar displays 'localhost / localhost / ...' and the URL 'localhost/CH6/exemple6.1.html'. The page content is a form titled 'Vos coordonnées' containing the following fields:

- Nom :  (labeled 1)
- Mail :  (labeled 2)
- Téléphone :  (labeled 3)
- Code :  (labeled 4)
- Département :  (labeled 5)
- Né le :  (labeled 6)
- Homme (labeled 7)
- Femme (labeled 8)
- France

Below this section is another titled 'Vos goûts' containing:

- Pommes (labeled 9)
- Poires
- Scoubidous

Underneath is a text area labeled 'Décrivez vos goûts en détail' (labeled 10).

The final section is titled 'Envoyez nous votre photo' containing:

- (labeled 11)
- 
- (labeled 12)
- (labeled 13)

**Figure 6-1**  
*Formulaire type complet*

#### Fichier HTML 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 côté serveur. Il n'est pas gênant (mais pas vraiment utile) 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` de l'élément `<form>`. La requête contient toutes les associations entre les noms des champs et leur valeur. Ces associations se trouvent dans l'en-tête HTTP si la méthode `POST` est utilisée et dans l'URL s'il s'agit de la méthode `GET`.

### Valeurs uniques

Les valeurs uniques proviennent des champs de formulaire dans lesquels l'utilisateur ne peut entrer qu'une valeur, un texte par exemple, ou ne peut faire qu'un seul choix (bouton radio, liste de sélection à choix unique).

Depuis PHP 4.1, ces valeurs sont contenues sur le serveur dans des tableaux associatifs dits superglobaux appelés `$_POST` et `$_GET`, selon la méthode choisie. Les clés de ces tableaux sont les noms associés aux champs par l'attribut `name`.

Voyons tout de suite ce mécanisme à l'œuvre avec un formulaire élémentaire ne contenant qu'un seul champ de saisie de texte. Dans la suite de l'ouvrage, nous privilégions l'emploi de la méthode `post` pour l'envoi des formulaires. Dans ce chapitre, en revanche, nous envisageons la méthode `get` à titre informatif.

### Cas de la méthode `POST`

Dans le code de l'exemple 6-2, l'élément `<form>` est écrit de la façon suivante (repère ①) :

```
| action= "<?= $_SERVER["PHP_SELF"] ?>"
```

Cela désigne le script lui-même comme cible pour le traitement des données.

Le code PHP proprement dit est le suivant (repère ②) :

```
<?php  
if(isset($_POST["nom"]) && isset($_POST["niveau"]))  
{  
echo "<h2> Bonjour ". stripslashes($_POST["nom"]). " vous êtes ".$_POST["niveau"].  
" en PHP</h2>";  
}  
?>
```

Il contrôle d'abord l'existence des variables `$_POST["nom"]` et `$_POST["niveau"]`, qui représentent respectivement le texte saisi et la valeur associée à la case cochée de façon à n'afficher le message qu'après l'envoi des données (repère 3). Sans ces précautions, le message s'afficherait même en l'absence de saisie de données par le visiteur. Ces variables n'existent que si l'utilisateur a complété les champs et a cliqué sur le bouton d'envoi. Lorsqu'elles existent, elles sont utilisées pour créer un affichage de bienvenue, comme illustré à la figure 6-2.

La fonction `stripslashes()` est utilisée pour supprimer les caractères d'échappement ajoutés automatiquement devant les caractères spéciaux éventuellement utilisés dans les données saisies avant de les afficher dans la page.

## Exemple 6-2. Récupération des valeurs dans un formulaire élémentaire

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Formulaire traité par PHP</title>
</head>
<body>
<form action="<?=_SERVER[\"PHP_SELF\"] ?>" method="post"
      enctype="application/x-www-form-urlencoded"> ← 1
  <fieldset>
    <legend><b>Infos</b></legend>
  <div>
    Nom : <input type="text" name="nom" size="40" />
    <br />
    Débutant : <input type="radio" name="niveau" value="débutant" />
    Initié : <input type="radio" name="niveau" value="initié" /><br />
    <input type="reset" value="Effacer" />
    <input type="submit" value="Envoyer" />
  </div>
  </fieldset>
</form>
<?php ← 2
if(isset($_POST["nom"]) && isset($_POST["niveau"])) ← 3
{
  echo "<h2> Bonjour ". stripslashes($_POST["nom"]) . " vous êtes ".
    $_POST["niveau"] . " en PHP</h2>";
}
?>
</body>
</html>
```

Après avoir saisi « Jan Geelsen » dans le champ texte et coché la case « débutant », vous obtenez l'affichage illustré à la figure 6-2. Remarquez que les saisies ne sont plus visibles car la page a été entièrement réaffichée dans le navigateur ; pour conserver les données saisies dans le formulaire, consultez la section « Maintien de l'état du formulaire » plus loin dans ce chapitre.



**Figure 6-2**  
*Formulaire élémentaire et résultat*

## Cas de la méthode GET

Avec la méthode `GET`, vous récupérez les données du formulaire dans les variables `$_GET["nom"]` et `$_GET["niveau"]`, comme ci-dessous :

```
<?php
if(isset($_GET["nom"]) && isset($_GET["niveau"]))
{
    echo "<h2> Bonjour ". stripslashes($_GET["nom"]). " vous êtes ".$_GET["niveau"]. 
    " en PHP</h2>";
}
?>
```

Contrairement à ce qui se passe avec la méthode `POST`, vous constatez que lors du clic sur le bouton d'envoi l'adresse de la page cible désignée par l'attribut `action` est suivie par le caractère `?` puis par le nom de chaque champ et la valeur qui y est associée.

Par exemple, si le nom de l'utilisateur est « Jean-René d'Orléans » et que la case « débutant » est cochée, la plupart des navigateurs (mais pas tous) affichent l'adresse complète suivante :

```
http://localhost/CH6/exemple6.2b.php?nom=Jean+Ren%C3%A9+d%27Orl%C3%A9ans&niveau
    =d%C3%A9butant
```

Les caractères accentués (é) et l'apostrophe sont codés en hexadécimal, par `%C3%A9` et `%27` respectivement. Les espaces sont remplacées par le signe `+`, et chaque paire `nom=valeur` est séparée par le signe `&`.

Dans les versions antérieures à PHP 4.1, les données étaient récupérables dans les variables `$HTTP_POST_VARS` ou `$HTTP_GET_VARS` ou encore directement dans des variables

globales portant le nom des champs du formulaire (la valeur de leur attribut `name`), soit ici `$nom` et `$niveau`. Ces méthodes sont à considérer comme obsolètes.

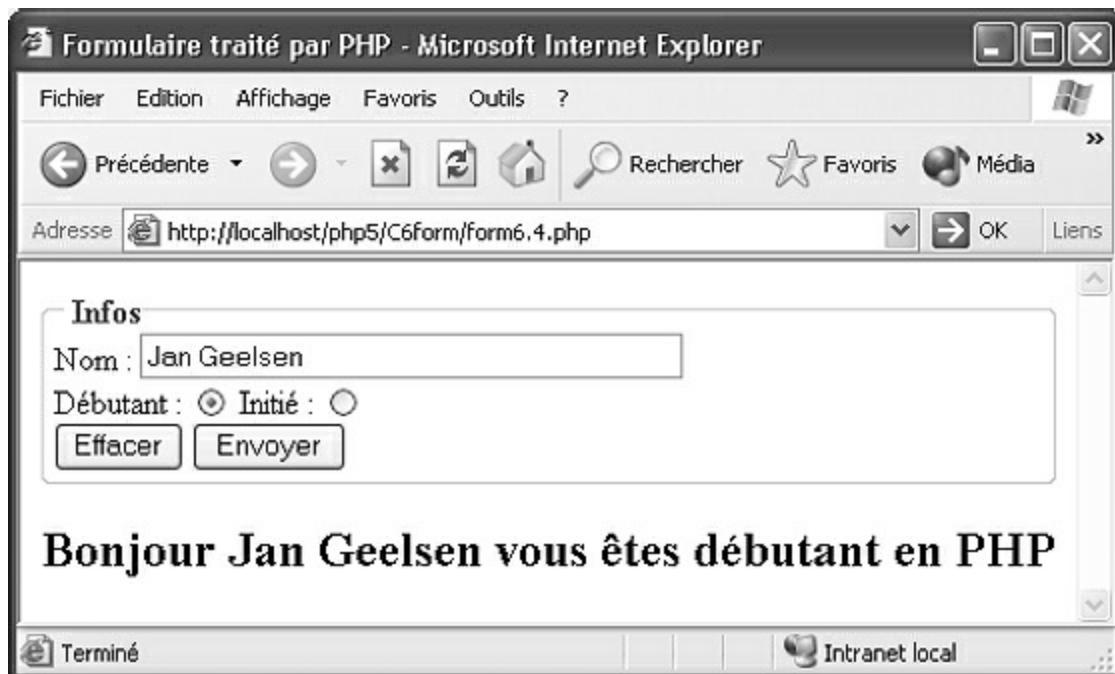
Pour simplifier la manipulation des valeurs issues du formulaire, vous pouvez récupérer chaque valeur dans des variables scalaires dès le début du script de traitement, comme ci-dessous :

```
$sonnom= $_POST["nom"];
$sonniveau=$_POST["niveau"];
```

puis les réutiliser sous cette forme par la suite.

## Maintien de l'état du formulaire

Lorsque le script contenant le formulaire est chargé du traitement des données, l'ensemble de la page est réaffiché après traitement, de même que l'ensemble du formulaire. Le formulaire se retrouve alors dans son état initial, et toutes les saisies effectuées sont effacées (voir figure 6-2).



**Figure 6-3**  
*Maintien de l'état du formulaire*

En cas d'erreur de saisie sur un seul champ, l'utilisateur est obligé de recommencer l'ensemble de la saisie. S'il s'agit d'un long questionnaire, il y a de quoi s'impatienter. Pour éviter cela, il est utile de maintenir l'état du formulaire après traitement en réaffichant l'ensemble des saisies. Vous allez effectuer cette opération à partir du code de l'exemple 6-2 :

- Pour la zone de saisie de texte dont l'attribut `name` a la valeur "`nom`", il suffit de définir l'attribut `value` avec la variable `$_POST["nom"]`, non sans avoir au préalable contrôlé l'existence de cette variable. Lors du premier affichage de la page, la zone est donc vide ou contient le dernier nom saisi (repère ①). Le code PHP est donc :

```
<?php if(isset($_POST["nom"])) echo $_POST["nom"]?>
```

- Pour les boutons radio dont l'attribut `name` a la valeur "niveau" et qui permettent le choix entre les valeurs "Débutant" et "Initié", il faut définir l'attribut `checked` du bouton choisi à la valeur "checked" en fonction de la valeur de la variable `$_POST["niveau"]` (repère ②). Pour le premier bouton, le code PHP devient :

```
<?php if(isset($_POST["niveau"]) && $_POST["niveau"]=="débutant")  
    echo "checked=\\"checked\\\" ?>
```

Après une saisie identique à celle de la figure 6-2, vous obtenez un écran identique à celui de la figure 6-3, dans lequel toutes les données saisies sont encore visibles après le traitement du formulaire.

### Exemple 6-3. Maintien de l'état du formulaire

```
<!DOCTYPE html>  
<html lang="fr">  
<head>  
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
    <title>Formulaire traité par PHP</title>  
</head>  
<body>  
    <form action="<?= $_SERVER["PHP_SELF"] ?>" method="post" enctype  
        ="application/x-www-form-urlencoded">  
        <fieldset>  
            <legend><b>Infos</b></legend>  
            Nom : <input type="text" name="nom" size="40" value="<?php if(isset($_POST["nom"]))  
                echo $_POST["nom"]?>"/> ← ①  
            <br />  
            Débutant : <input type="radio" name="niveau" value="debutant"  
            <?php if(isset($_POST["niveau"]) && $_POST["niveau"]=="debutant") echo "checked  
                =\\"checked\\\" ?> /> ← ②  
            Initie : <input type="radio" name="niveau" value="initie"  
            <?php if(isset($_POST["niveau"]) && $_POST["niveau"]=="initie") echo "checked  
                =\\"checked\\\" ?>/><br />  
            <input type="reset" value="Effacer" />  
            <input type="submit" value="Envoyer" />  
        </fieldset>  
    </form>  
    <?php  
        if(isset($_POST["nom"]) && isset($_POST["niveau"]))  
        {  
            echo "<h2> Bonjour ". stripslashes($_POST["nom"]). " vous êtes ".$_POST["niveau"].  
                " en PHP</h2>";  
        }  
    ?>  
</body>  
</html>
```

### Exemple pratique

L'exemple 6-4 offre une illustration pratique d'utilisation de formulaire. Il s'agit d'un site d'annonces immobilières proposant un plan de financement aux visiteurs.

L'application est constituée de deux fichiers, `exemple6.4.html` et `exemple6.4.php`.

Le fichier `exemple6.4.html` affiche le formulaire de saisie des données nécessaires au calcul du prêt (voir figure 6-4). Il ne contient aucun code PHP et peut donc être enregistré avec l'extension `.html`. L'attribut `action` de l'élément `<form>` (repère 1) désigne le fichier `exemple6.4.php`, qui est chargé du traitement des données et de l'affichage des résultats.

The screenshot shows a web browser window with the title "Opera" and the address bar displaying "localhost". The main content area is titled "Prêts" and contains a form for calculating a loan. The form is labeled "Les caractéristiques de votre prêt". It includes input fields for "Montant" (amount) set to 120000, "Taux" (interest rate) set to 3.2, and "Durée en année" (duration in years) set to 10. There is also a section for "Assurance" (insurance) with radio buttons for "OUI" (Yes) and "NON" (No), where "OUI" is selected. Below the form are two buttons: "Effacer" (Clear) and "Calculer" (Calculate).

**Figure 6-4**  
*Page principale de saisie des données*

Le fichier `exemple6.4.php` vérifie d'abord l'existence des variables `$_POST["capital"]`, `$_POST["taux"]` et `$_POST["duree"]`, toutes nécessaires au calcul du prêt. La variable `$_POST["assur"]` est nécessaire dans tous les cas. Elle a la valeur 1 puisque le bouton radio « OUI » est coché par défaut. `$capital` correspond au capital emprunté (repère 2). `$taux` désigne le taux mensuel sous forme décimale. Si l'utilisateur saisit 6 pour le taux annuel, la variable `$taux` vaut  $6/100/12$ , soit 0,005, ou 0,5 % par mois (repère 3). `$duree` est la durée en mois (repère 4). `$assur` renvoie au montant de l'assurance mensuelle, soit 0,035 % du capital emprunté. Cette variable prend la valeur 0 si `$_POST["assur"]` vaut 0 (repère 5).

Vient ensuite le calcul de la mensualité selon la formule financière suivante (repère 6) :

```
| $mens=($capital*$taux) / (1-pow((1+$taux), $duree))
```

Le script affiche la mensualité hors assurance (repère 7) ainsi que le tableau d'amortissement contenant, entre autres, le capital restant dû et les intérêts de chaque période (repère 8). La figure 6-5 donne un exemple de résultat.

Si le formulaire est incomplet, l'instruction `header()` affiche à nouveau la page de saisie

exemple6.4.html, obligeant l'utilisateur à effectuer une saisie complète (repère 9).

The screenshot shows a browser window with the title "Tableau d'amortissement..." and the URL "localhost/CH6/exemple6.4.php". The main content displays a heading: "Pour un prêt de 120000 € à 3.2%, sur 10 ans la mensualité est de 1169.84 € hors assurance". Below this, a caption reads "Tableau d'amortissement du prêt". A table follows, showing monthly amortization details for 15 months. The columns are labeled: Mois, Capital restant, Mensualité Hors Ass., Amortissement, Intérêt, Assurance, and Mensualité Ass. cis. The data shows a decreasing capital balance and constant monthly payments of 1169.84 €.

Mois	Capital restant	Mensualité Hors Ass.	Amortissement	Intérêt	Assurance	Mensualité Ass. cis
1	120000	1169.84	849.84	320	42	1211.84
2	119150.16	1169.84	852.11	317.73	42	1211.84
3	118298.05	1169.84	854.38	315.46	42	1211.84
4	117443.67	1169.84	856.66	313.18	42	1211.84
5	116587.02	1169.84	858.94	310.9	42	1211.84
6	115728.08	1169.84	861.23	308.61	42	1211.84
7	114866.84	1169.84	863.53	306.31	42	1211.84
8	114003.32	1169.84	865.83	304.01	42	1211.84
9	113137.48	1169.84	868.14	301.7	42	1211.84
10	112269.34	1169.84	870.46	299.38	42	1211.84
11	111398.89	1169.84	872.78	297.06	42	1211.84
12	110526.11	1169.84	875.1	294.74	42	1211.84
13	109651.01	1169.84	877.44	292.4	42	1211.84
14	108773.57	1169.84	879.78	290.06	42	1211.84
15	107892.79	1169.84	882.12	287.72	42	1211.84

**Figure 6-5**  
Page d'affichage des résultats

### Exemple 6-4. Calcul de prêt bancaire

Page de saisie des données (fichier exemple6.4.html) :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Calcul de prêts</title>
</head>
<body>
<h3>Prêts</h3>
<form method="post" action="exemple6.4.php">← 1
<fieldset>
<legend>Les caractéristiques de votre prêt</legend>
<table>
<tr>
<td>Montant</td>
<td><input type="number" name="capital" step="1000"/></td>
</tr>
<tr>
<td>Taux</td>
```

```

<td><input type="number" name="taux" step="0.1"/></td>
</tr>
<tr>
<td>Durée en années</td>
<td><input type="number" name="duree" step="1"/></td>
</tr>
<tr>
<td>Assurance</td>
<td>OUI : <input type="radio" name="assur" checked="checked" value="1" />&ampnbsp;
    ↪ 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>

```

## Page de traitement des données et d'affichage des résultats (fichier `exemple6.4.php`) :

```

<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Tableau d'amortissement</title>
  </head>
  <body>
    <div>

    <?php
    if(isset($_POST["capital"]) && $_POST["taux"] && $_POST["duree"])
    {
      $capital=$_POST["capital"]; ← ②
      $taux=$_POST["taux"]/100/12; ← ③
      $duree=$_POST["duree"]*12; ← ④
      $assur=$_POST["assur"]*$capital*0.00035; ← ⑤
      $mens=($capital*$taux) / (1-pow((1+$taux),-$duree)); ← ⑥
      echo "<h3>Pour un prêt de $capital &euro; à ", $_POST["taux"] , "%, sur
      ", $_POST["duree"],
      ↪ " ans, la mensualité est de ",round($mens,2)," &euro; hors assurance</h3>"; ← ⑦
      echo "<h4>Tableau d'amortissement du prêt</h4>";
      echo "<table border=\"1\"> <tr><th>Mois </th><th>Capital restant</th><th>
      Mensualité
      ↪ Hors Ass.</th><th>Amortissement </th><th>Intérêt</th><th> Assurance</th>
      <th>Mensualité
      ↪ Ass. cis </th>"; ← ⑧
    // Boucle d'affichage des lignes du tableau
    for($i=1;$i<=$duree;$i++)
    {
      $int=$capital*$taux;
      $amort=$mens-$int;
      echo "<tr>";
      echo "<td>$i</td>";
      echo "<td>",round($capital,2),"</td>";

```

```

echo "<td>", round($mens,2), "</td>";
echo "<td>", round($amort,2), "</td>";
echo "<td>", round($int,2), "</td>";
echo "<td>$assur</td>";
echo "<td>", round($mens+$assur,2), "</td>";
echo "</tr>";
$capital=$capital-$amort;
}
echo "</table>";
}
else
{
    header("Location:exemple6.4.php"); ← ⑨
}
?>
</div>
</body>
</html>

```

## Les valeurs multiples

Certains champs de formulaire peuvent permettre aux visiteurs de saisir plusieurs valeurs sous un même nom de composant.

Cela peut concerter un groupe de cases à cocher ayant le même attribut `name`, par exemple, dont il est possible de cocher une ou plusieurs cases simultanément. Ce peut également être le cas d'une liste de sélection ayant toujours un nom unique mais dans laquelle l'attribut `multiple="multiple"` est défini. Il est enfin possible de donner le même nom à des éléments de saisie de texte différents, mais cela présente moins d'intérêt.

Dans tous les cas, ce n'est pas une valeur scalaire mais un tableau qui est récupéré côté serveur. Il faut pour cela faire suivre le nom du composant de crochets, comme pour créer une variable de type `array`.

Dans l'exemple suivant :

```

Bleu:<input type="checkbox" name="choix[]" value="bleu" />
Blanc:<input type="checkbox" name="choix[]" value="blanc" />

```

l'utilisateur peut cocher les deux cases simultanément. Le programmeur récupère ces valeurs dans les variables suivantes :

```

$_POST["choix"][0] qui contient la valeur "bleu"
$_POST["choix"][1] qui contient la valeur "blanc"

```

La variable `$_POST` est un tableau multidimensionnel, en l'occurrence à deux dimensions.

L'exemple 6-5 illustre la méthode de récupération des valeurs multiples. Le formulaire créé par le fichier `exemple6.5.html` contient trois zones de saisie de texte portant le même nom, une liste de sélection avec l'attribut `multiple` et quatre cases à cocher ayant le même nom. Dans les listes de sélection, l'utilisateur doit maintenir la touche Ctrl enfoncee pour faire plusieurs choix. Il peut être utile de lui rappeler cette fonctionnalité.

L'objet du formulaire est de faire saisir une fiche de renseignements par l'utilisateur

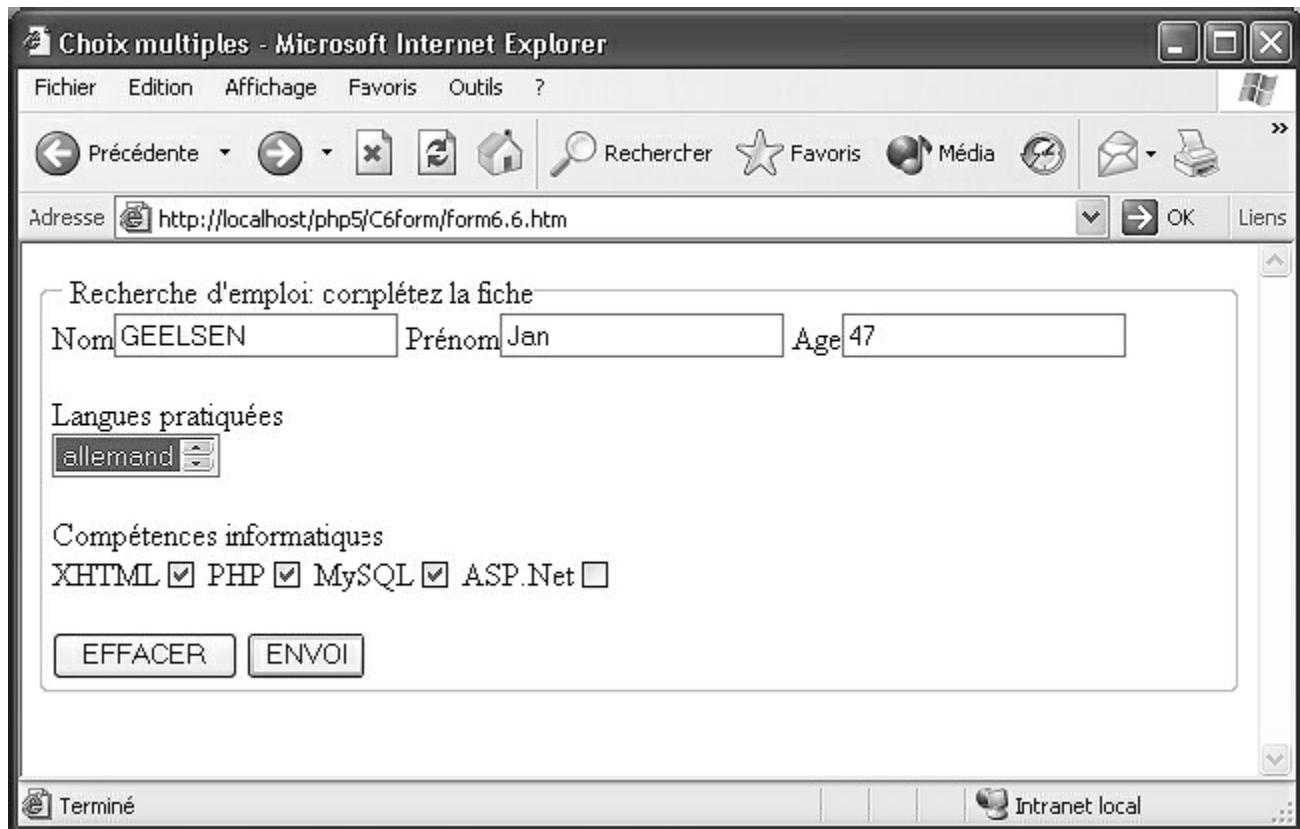
puis d'afficher l'ensemble de ces informations. Le script cible du formulaire contenu dans le fichier `exemple6.5.php` récupère les données et réalise une fiche récapitulative des renseignements personnels si les variables du tableau `$_POST` existent (repère 1) ou, dans le cas contraire, une boîte d'alerte, à l'aide de la fonction JavaScript `alert()` (repère 2), et une redirection vers la page de saisie, via la fonction JavaScript `window.history.back()` (repère 3).

La figure 6-6 illustre le formulaire de saisie et la figure 6-7 un exemple de fiche de résultat.

## Exemple 6-5. Récupération des valeurs multiples

Le fichier `exemple6.5.html` :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Listes à choix multiples</title>
</head>
<body>
<form method="post" action="exemple6.5.php" >
<fieldset>
<legend>Recherche d'emploi : complétez la fiche</legend>
<div>
<span>
Nom<input type="text" name="ident[]" />
Prénom<input type="text" name="ident[]" />
Âge<input type="number" step="1" name="ident[]" />
<br /><br />
Langues pratiquées<br />
<select name="lang[]" multiple="multiple">
<option value="français"> français</option>
<option value="anglais"> anglais</option>
<option value="allemand"> allemand</option>
<option value="espagnol"> espagnol</option>
</select><br /><br />
Compétences informatiques<br />
HTML 5<input type="checkbox" name="competent[]" value="HTML 5" />
CSS 3<input type="checkbox" name="competent[]" value="CSS 3" />
PHP<input type="checkbox" name="competent[]" value="PHP" />
MySQL<input type="checkbox" name="competent[]" value="MySQL" />
</span><br /><br />
<input type="reset" value="EFFACER"/>
<input type="submit" value="ENVOI"/>
</div>
</fieldset>
</form>
</body>
</html>
```



**Figure 6-6**  
*Formulaire de saisie*

Le fichier exemple6.5.php :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Compétences informatiques</title>
</head>
<body>

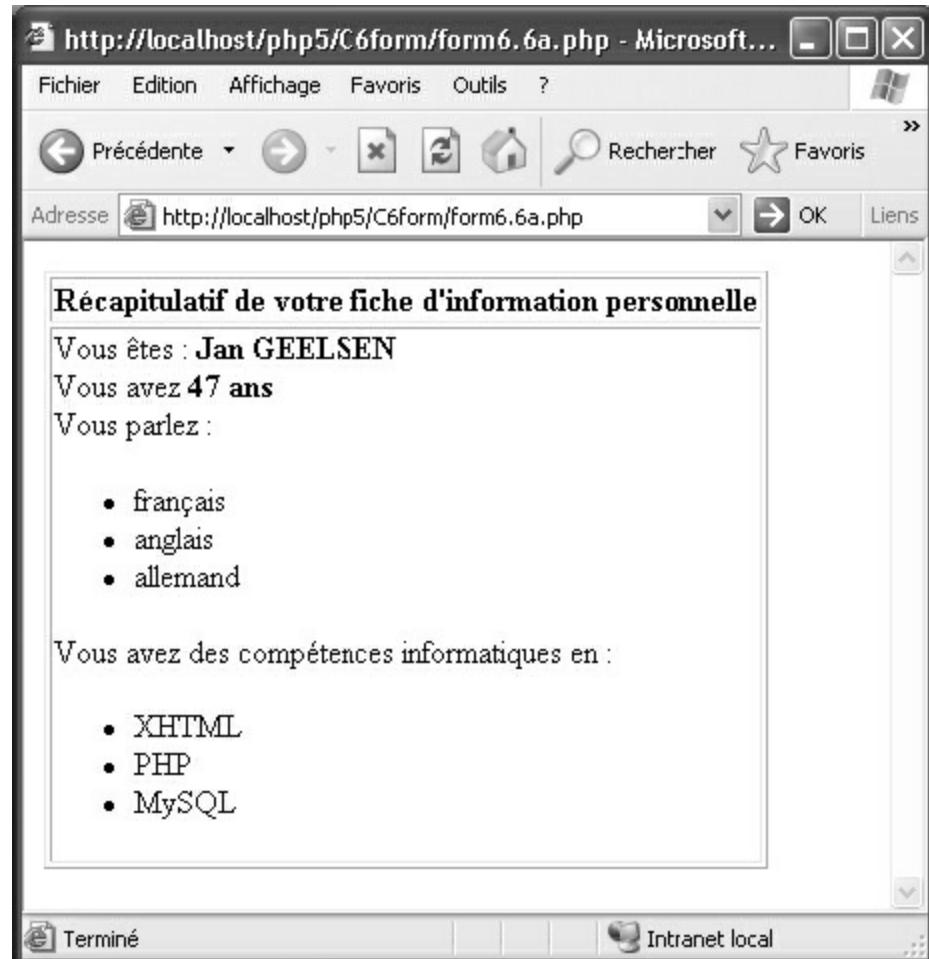
<?php
if(isset($_POST["ident"])) && isset($_POST["lang"]) && isset($_POST["competent"])) ←
①
{

echo "<table border=\"1\"><tr><th> Récapitulatif de votre fiche d'informations
personnelles
</th></tr><tr><td>";
$nom=$_POST["ident"][0];
$prenom=$_POST["ident"][1];
$age=$_POST["ident"][2];
$lang = $_POST["lang"];
$competent=$_POST["competent"];
echo"Vous êtes :<b> $prenom ", stripslashes($nom) , "</b><br />Vous avez <b>$age ans
</b> ";
echo "<br />Vous parlez :";
echo "<ul>";
foreach($lang as $valeur)
{
```

```

echo " <li> $valeur </li>";
}
echo "</ul>";
echo "Vous avez des compétences informatiques en :";
echo "<ul>";
foreach($competent as $valeur)
{
    echo "<li> $valeur </li> ";
}
echo "</ul> </td></tr>";
}
else
{
echo"<script type=\"text/javascript\">";
echo "alert('Saisissez votre nom et cochez au moins une compétence !');";←②
echo "window.history.back();";←③
echo "</script>";
}
?>
</body>
</html>

```



**Figure 6-7**  
*Page de résultat*

## Transfert de fichiers vers le serveur

L'inclusion d'un élément HTML `<input type="file" />` dans un formulaire crée une situation particulière. Il ne s'agit plus de transmettre une ou plusieurs valeurs scalaires au serveur mais l'intégralité d'un fichier, lequel peut avoir une taille importante et un type quelconque. Ce fichier doit évidemment être présent sur l'ordinateur du visiteur.

Par rapport au transfert de données, le transfert de fichiers présente un problème de sécurité pour votre site puisque des fichiers vont être écrits et éventuellement exécutés sur le serveur.

Supposez qu'un utilisateur mal intentionné transfère un fichier nommé `index.php` ou `index.html` sur le serveur. Ce fichier est exécuté automatiquement à l'appel de l'URL de votre nom de domaine. Pour éviter tout problème, il est prudent de prévoir une vérification de l'extension du fichier préalablement à la définition des extensions autorisées lors de la création du formulaire.

Pour cette définition, vous utilisez l'attribut `accept` de l'élément `<input />`. Cet attribut peut prendre un grand nombre de valeurs, correspondant aux types MIME des fichiers acceptés, par exemple `"image/gif"`, comme dans l'exemple 6-6, ou `"text/html"`. Un fichier d'un type différent est rejeté, ce qui confère une protection contre les utilisateurs mal intentionnés.

Contrairement aux exemples précédents, l'élément `<form>` doit avoir l'attribut `method` à la valeur `post` et l'attribut `enctype` à la valeur `multipart/form-data`.

Précaution supplémentaire, vous pouvez envisager de limiter la taille des fichiers à télécharger en ajoutant au formulaire un champ caché nommé `MAX_FILE_SIZE`, dont l'attribut `value` contienne la taille maximale admise en octet. Cette valeur est récupérée dans la variable `$_POST["MAX_FILE_SIZE"]` (repère ②) lorsque le champ caché est défini par le code suivant (repère ①) :

```
| <input type="hidden" name="MAX_FILE_SIZE" value="100000" />
```

L'utilisation de ce champ n'est pas obligatoire puisque le fichier `php.ini` du serveur contient la directive `"upload_max_filesize"`, dont la valeur est un entier indiquant la taille maximale en octet admise par défaut par le serveur. En local avec Wampserver, cette valeur est de 2 Mo. L'hébergeur peut toutefois définir une valeur très différente, qu'il vous appartient de vérifier à l'aide de la fonction `phpinfo()`.

L'ajout d'un bouton d'envoi à votre formulaire est bien sûr toujours indispensable.

Lors de l'affichage de la page, l'utilisateur se retrouve face à ce qui ressemble à une zone de saisie de texte accompagnée d'un bouton dont l'intitulé est « Parcourir... ». Il peut saisir manuellement l'emplacement exact du fichier à transférer ou le choisir sur son disque en cliquant sur le bouton, comme il le ferait pour ouvrir un fichier à partir d'un traitement de texte, par exemple. Il n'est donc pas nécessaire de lui expliquer la procédure en détail.

Une fois le fichier sélectionné, un clic sur le bouton `submit` provoque l'envoi du fichier au serveur et son traitement par un script, que vous devez encore écrire. À ce moment,

le fichier est bien transféré sur le serveur, mais il se trouve dans un répertoire tampon défini par la directive "upload\_tmp\_dir" du fichier `php.ini`. Si vous n'en faites rien, il est perdu lors de la déconnexion du client. De plus, il est enregistré sous un nom différent de celui qu'il avait sur le poste client, et vous ne savez pas encore sous quel nom, puisque celui-ci est créé arbitrairement par le serveur.

De même que pour le transfert de données simples, vous allez utiliser les tableaux superglobaux `$_POST` ou `$_GET`. Vous disposez également depuis la version PHP 4.1 du tableau associatif multidimensionnel `$_FILES`, qui contient les informations nécessaires au traitement du fichier transféré.

Si, comme dans le code de l'exemple 6-6, le nom de l'élément `<input type="file" .. />` est `name="fich"`, vous pouvez lire les valeurs suivantes :

- `$_FILES["fich"]["name"]` : contient le nom qu'avait le fichier sur le poste client.
- `$_FILES["fich"]["type"]` : contient le type MIME initial du fichier et permet un contrôle et une censure éventuelle du fichier transféré.
- `$_FILES["fich"]["size"]` : donne la taille réelle en octet du fichier transféré.
- `$_FILES["fich"]["tmp_name"]` : donne le nom temporaire que le serveur attribue automatiquement au fichier en l'enregistrant dans le répertoire tampon.
- `$_FILES["fich"]["error"]` : donne le code d'erreur éventuel associé au fichier téléchargé et permet d'afficher un message d'erreur en clair en créant un tableau indicé de 0 à 4 contenant les messages appropriés. Ces codes sont définis par les constantes entières suivantes depuis PHP 4.3 :
  - `UPLOAD_ERR_OK` : de valeur 0, indique que le transfert est bien réalisé.
  - `UPLOAD_ERR_INI_SIZE` : de valeur 1, indique que la taille du fichier est supérieure à celle définie dans la `php.ini`.
  - `UPLOAD_ERR_FORM_SIZE` : de valeur 2, indique que la taille est supérieure à celle définie dans le champ caché `MAX_FILE_SIZE`.
  - `UPLOAD_ERR_PARTIAL` : de valeur 3, indique que le fichier n'a été que partiellement téléchargé.
  - `UPLOAD_ERR_NO_FILE` : de valeur 4, indique qu'aucun fichier n'a été téléchargé.

En dernier lieu, vous devez procéder à l'enregistrement et au renommage éventuel du fichier sur le serveur. Vous disposez pour cela de la fonction `move_uploaded_file()`, dont la syntaxe est la suivante :

```
boolean move_uploaded_file( string "fichtmp", string "fichfinal")
```

"fichtmp" est le chemin d'accès du fichier temporaire et "fichfinal" le nom sous lequel sera enregistré définitivement le fichier. La fonction retourne `TRUE` si l'opération est bien réalisée et `FALSE` dans le cas contraire.

Dans l'exemple 6-6, le code suivant (repère ③) :

```
| move_uploaded_file($_FILES["fich"]["tmp_name"], "imagephp.gif");
```

enregistre le fichier transféré désigné par `$_FILES["fich"]["tmp_name"]` dans le répertoire du fichier `exemple6.6.php` du serveur sous le nom "imagephp.gif". Le fichier est alors pleinement utilisable dans une page du site, comme si vous l'aviez transféré vous-même à l'aide d'un logiciel FTP.

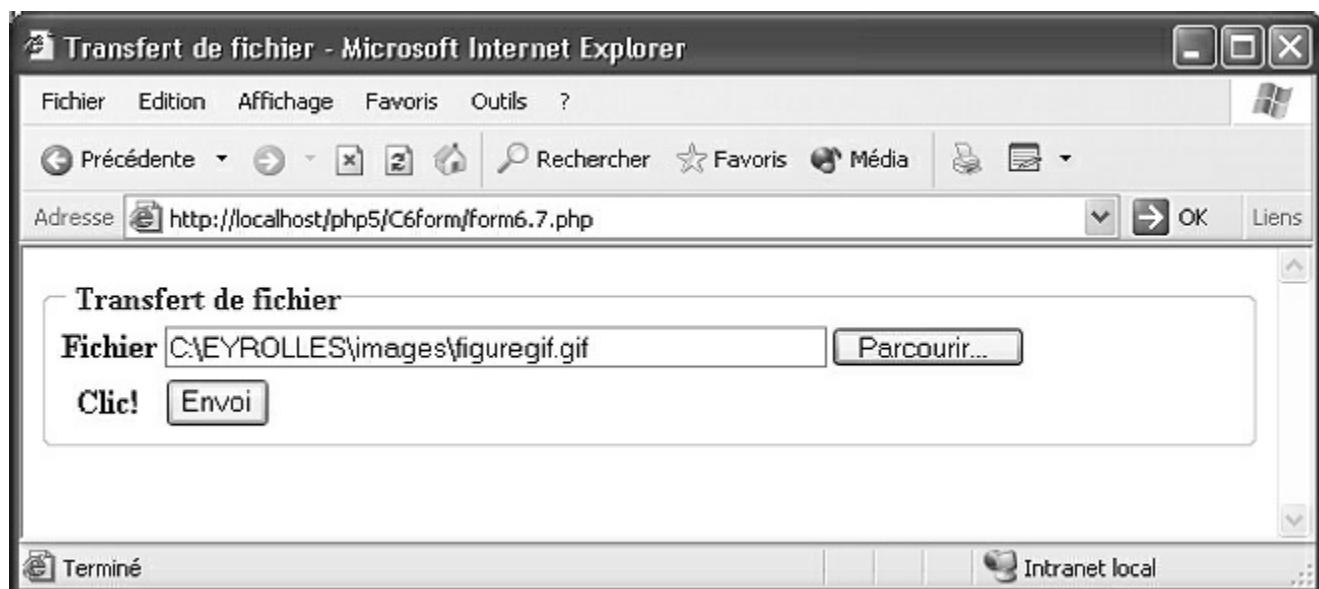
Si le transfert est bien réalisé, vous affichez un message de bonne fin (repère 4). En cas de problème de transmission ou de renommage du fichier, le script affiche le code de l'erreur générée à l'aide de la variable `$_FILES["fich"]["error"]` (repère 5).

## Exemple 6-6. Transfert de fichiers

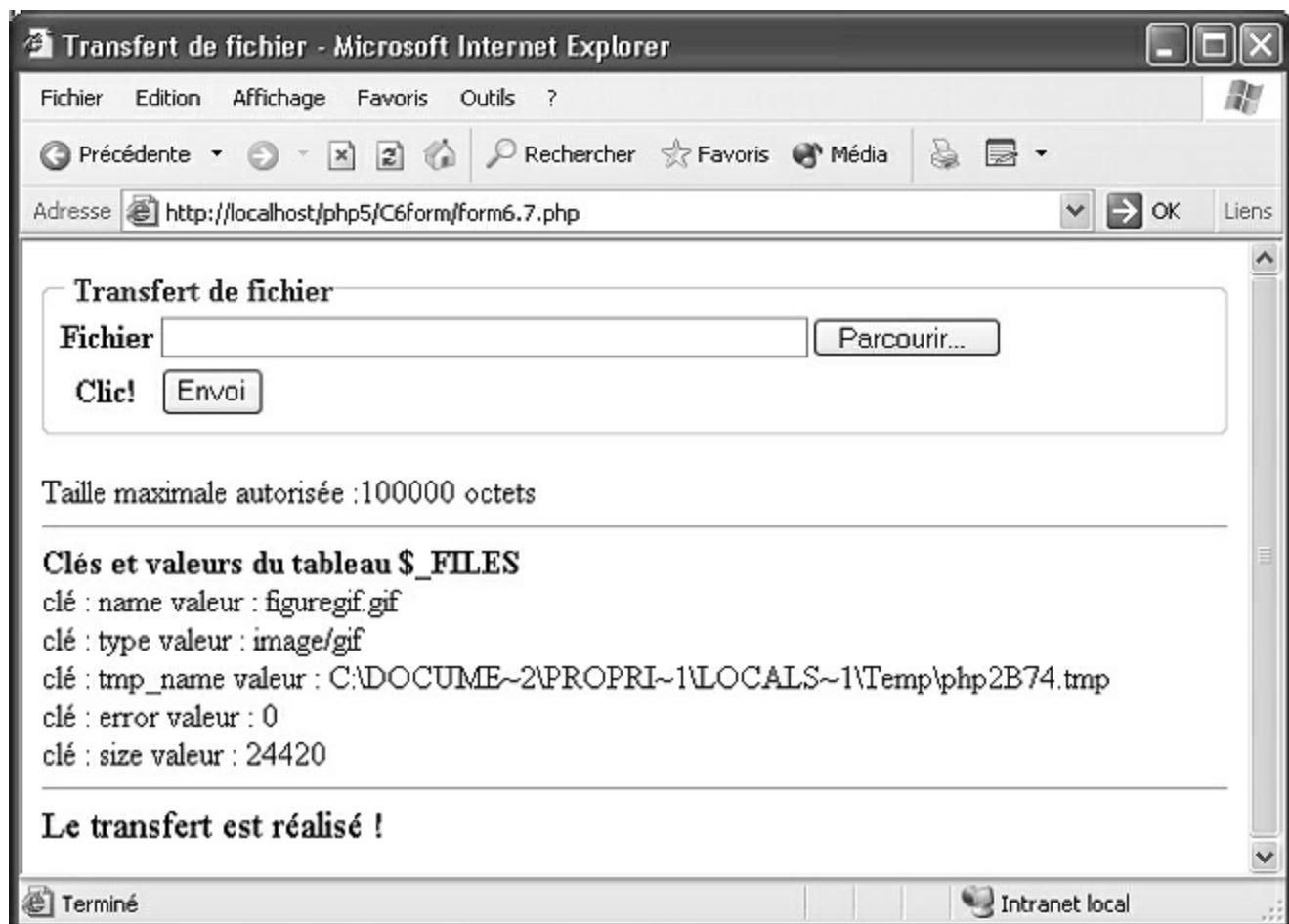
```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Transfert de fichiers</title>
</head>
<body>
<form action="exemple6.6.php" method="post" enctype="multipart/form-data" >
<fieldset>
<input type="hidden" name="MAX_FILE_SIZE" value="100000" /> ←①
<legend><b>Transfert de fichiers</b></legend>
<table>
<tbody>
<tr>
<th>Fichier</th>
<td><input type="file" name="fich" accept="image/gif" size="50"/></td>
</tr>
<tr>
<th>Clic !</th>
<td> <input type="submit" value="Envoi" /></td>
</tr>
</tbody>
</table>
</fieldset>
</form>
<!-- Code PHP -->
<?php
if(isset($_FILES['fich']))
{
    echo "Taille maximale autorisée : ", $_POST["MAX_FILE_SIZE"], " octets<br />"; ←②
    echo "<b>Clés et valeurs du tableau \$_FILES </b><br />";
    foreach($_FILES["fich"] as $cle => $valeur)
    {
        echo "clé : $cle valeur : $valeur <br />";
    }
    // Enregistrement et renommage du fichier
    $result=move_uploaded_file($_FILES["fich"]["tmp_name"], "imagephp.gif"); ←③
    if($result==TRUE)
        {echo "<br /><big>Le transfert est effectué !</big>"; } ←④
    else
        {echo "<br /> Erreur de transfert n°", $_FILES["fich"]["error"]; } ←⑤
    }
?>
```

```
</body>  
</html>
```

La figure 6-8 illustre l'aspect de la page d'envoi des fichiers et la figure 6-9 celui de la page de confirmation affichant le résultat de l'opération de transfert et les clés et valeurs du tableau `$_FILES`.



**Figure 6-8**  
*Formulaire d'envoi de fichier*



**Figure 6-9**

*Page de confirmation du transfert*

## **Transfert de plusieurs fichiers**

Il est possible de proposer à l'utilisateur de transférer plusieurs fichiers simultanément pour lui éviter de recommencer plusieurs fois la même opération à partir de l'exemple précédent. Vous devez en ce cas faire en sorte que les éléments `<input type="file"...>` soient tous écrits de la manière suivante :

```
| <input type="file" name="fich[]" accept="image/gif" size="50"/>
```

avec le même nom suivi de crochets (repères ① et ② de l'exemple 6-7) ; l'attribut `accept` peut prendre des valeurs différentes pour chaque élément `<input />`.

La page de transfert correspondante est illustrée à la figure 6-10.

The screenshot shows a Microsoft Internet Explorer window with the title "Transfert de plusieurs fichiers - Microsoft Internet Explorer". The address bar displays the URL "http://localhost/php5/C6form/form6.8.php". The main content area contains a form titled "Choisissez les fichiers à transférer". Inside the form, there is a section titled "Transferts de plusieurs fichiers" with two file input fields. The first field is labeled "Fichier 1" and the second is labeled "Fichier 2", each accompanied by a "Parcourir..." button. Below these fields are two buttons: "Clic!" and "Envoi". At the bottom of the browser window, the status bar shows "Terminé" and "Intranet local".

**Figure 6-10**

*Formulaire de transfert de plusieurs fichiers*

Vous récupérez les informations sur les fichiers transférés dans la tableau `$_FILES`, devenu multidimensionnel à trois dimensions.

Par exemple, la variable suivante :

```
| $_FILES["fich"]["name"][0]
```

permet de connaître le nom du premier fichier transféré.

Vous pouvez lire les caractéristiques de chacun des fichiers transférés à l'aide des boucles `foreach` imbriquées (repères ③ et ④). Ce code n'a toutefois d'intérêt que pour le programmeur lors des tests et est à supprimer dans un site réel.

Vous procédez enfin au renommage et à la sauvegarde des fichiers transférés (repères 5 et 6) puis au contrôle du bon déroulement de l'ensemble des opérations (repère 7).

## Exemple 6-7. Transfert simultané de plusieurs fichiers

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Transfert de plusieurs fichiers</title>
</head>
<body>

<!-- Code HTML du formulaire --&gt;
&lt;form action=&lt;?= $_SERVER['PHP_SELF'] ?&gt;" method="post" enctype="multipart/form-data" &gt;
&lt;fieldset&gt;
&lt;input type="hidden" name="MAX_FILE_SIZE" value="100000" /&gt;
&lt;legend&gt;&lt;b&gt;Transferts de plusieurs fichiers&lt;/b&gt;&lt;/legend&gt;
&lt;table&gt;
&lt;tbody&gt;
&lt;tr&gt;
&lt;th&gt;Fichier 1&lt;/th&gt;
&lt;td&gt; &lt;input type="file" name="fich[]" accept="image/gif" size="50"/&gt;&lt;/td&gt;← 1
&lt;/tr&gt;
&lt;tr&gt;
&lt;th&gt;Fichier 2&lt;/th&gt;
&lt;td&gt; &lt;input type="file" name="fich[]" accept="image/gif" size="50"/&gt;← 2
&lt;/td&gt;
&lt;/tr&gt;
&lt;tr&gt;
&lt;th&gt;Clic !&lt;/th&gt;
&lt;td&gt; &lt;input type="submit" value="Envoi" /&gt;&lt;/td&gt;
&lt;/tr&gt;
&lt;/tbody&gt;
&lt;/table&gt;
&lt;/fieldset&gt;
&lt;/form&gt;
<!-- Code PHP --&gt;
&lt;?php
if(!empty($_FILES))
{
    echo "Taille maximale autorisée :,".$_POST["MAX_FILE_SIZE"]," octets&lt;br / &gt;";
    foreach($_FILES["fich"] as $cle =&gt; $valeur)← 3
    {
        echo "Clé : $cle &lt;br /&gt;";
        foreach($valeur as $key=&gt;$val)← 4
        {
            echo " Fichier : $key valeur : $val &lt;br /&gt;";
        }
    }
    // Déplacement et renommage des fichiers
    $result1=move_uploaded_file($_FILES["fich"]["tmp_name"][0],"image1.gif");← 5
    $result2=move_uploaded_file($_FILES["fich"]["tmp_name"][1],"image2.gif");← 6
    if($result1==true &amp;&amp; $result2==true)← 7
    {echo "&lt;br /&gt; Transferts effectués !&lt;br /&gt;";}
}</pre>
```

```

    else
    {echo "<br /> Transferts non effectués <br />";}
}
else echo "<h4>Choisissez les fichiers à transférer </h4>";
?>
</body>
</html>

```

Les informations écrites par les boucles `foreach` ont l'aspect suivant en cas de réussite des transferts :

---

```

Taille maximale autorisée : 100 000 octets
Clé : name
Fichier : 0 valeur : VOEUX2013-v3.gif
Fichier : 1 valeur : figuregif.gif
Clé : type
Fichier : 0 valeur : image/gif
Fichier : 1 valeur : image/gif
Clé : tmp_name
Fichier : 0 valeur : C:\wamp\tmp\php8FB1.tmp
Fichier : 1 valeur : C:\wamp\tmp\php8FC2.tmp
Clé : error
Fichier : 0 valeur : 0
Fichier : 1 valeur : 0
Clé : size
Fichier : 0 valeur : 45902
Fichier : 1 valeur : 40657

```

---

```
Transferts effectués !
```

---

## Gérer les boutons d'envoi multiples

L'utilisation de plusieurs boutons `submit` dans un même formulaire permet de déclencher des actions différentes en fonction du bouton activé par l'utilisateur. Il est nécessaire pour cela que les boutons aient le même nom et que la sélection de l'action se fasse en fonction de la valeur associée à chaque bouton *via* l'attribut `value`.

L'exemple 6-8 crée une calculatrice en ligne proposant les opérations d'addition, de soustraction, de division et de puissance. Il est évidemment possible d'ajouter autant d'opérations que nécessaire. À chaque opération est associé un bouton `submit` (repères 1, 2, 3 et 4).

Deux zones de saisie de nombre (élément `<input/>` de type `number`) permettent d'entrer les deux opérandes X et Y. Dans cet exemple, vous utilisez le maintien de l'état du formulaire de façon que ces derniers restent visibles lors de l'affichage du résultat (repères 5 et 6).

Le code PHP commence comme d'habitude par vérifier l'existence des variables contenues dans le tableau `$_POST` en fonction de la valeur contenue dans la variable `$_POST ["calcul"]` de l'attribut `value` des boutons `submit`. Selon l'opération désirée, une instruction `switch` effectue le calcul approprié, dont le résultat est contenu dans la variable `$resultat`. Cette valeur est affichée dans le champ de `text` nommé `result`

(repère 7).

## Exemple 6-8. Une calculatrice en ligne

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Calculatrice en ligne</title>
  </head>
  <body>

    <!-- Code PHP -->
    <?php
    if(isset($_POST["calcul"])&&isset($_POST["nb1"])&&isset($_POST["nb2"]))
    {
      switch($_POST["calcul"])
      {
        case "Addition x+y":
          $resultat= $_POST["nb1"]+$_POST["nb2"];
          break;
        case "Soustraction x-y":
          $resultat= $_POST["nb1"]-$_POST["nb2"];
          break;
        case "Division x/y":
          $resultat= $_POST["nb1"]/$_POST["nb2"];
          break;
        case "Puissance x^y":
          $resultat= pow($_POST["nb1"],$_POST["nb2"]);
          break;
      }
    }
    else
    {
      echo "<h3>Entrez deux nombres : </h3>";
    }
    ?>
    <!-- Code HTML du formulaire -->
    <form action=<?=$_SERVER['PHP_SELF']?>" method="post" >
      <fieldset>
        <legend><b>Calculatrice en ligne</b></legend>
        <table>
          <tbody>
            <tr>
              <th>Nombre X</th>
              <td> <input type="number" step="0.1" name="nb1" size="30" value="<?php if(isset($_POST["nb1"])) echo $_POST['nb1'];else echo '' ?>" />← 5
                </td>
            </tr>
            <tr>
              <th>Nombre Y</th>
              <td> <input type="number" step="0.1" name="nb2" size="30" value="<?php if(isset($_POST["nb2"])) echo $_POST['nb2'];else echo '' ?>" />← 6
                </td>
            </tr>
            <tr>
              <th>Résultat </th>
              <td> <input type="number" step="0.5" name="result" size="30" value="

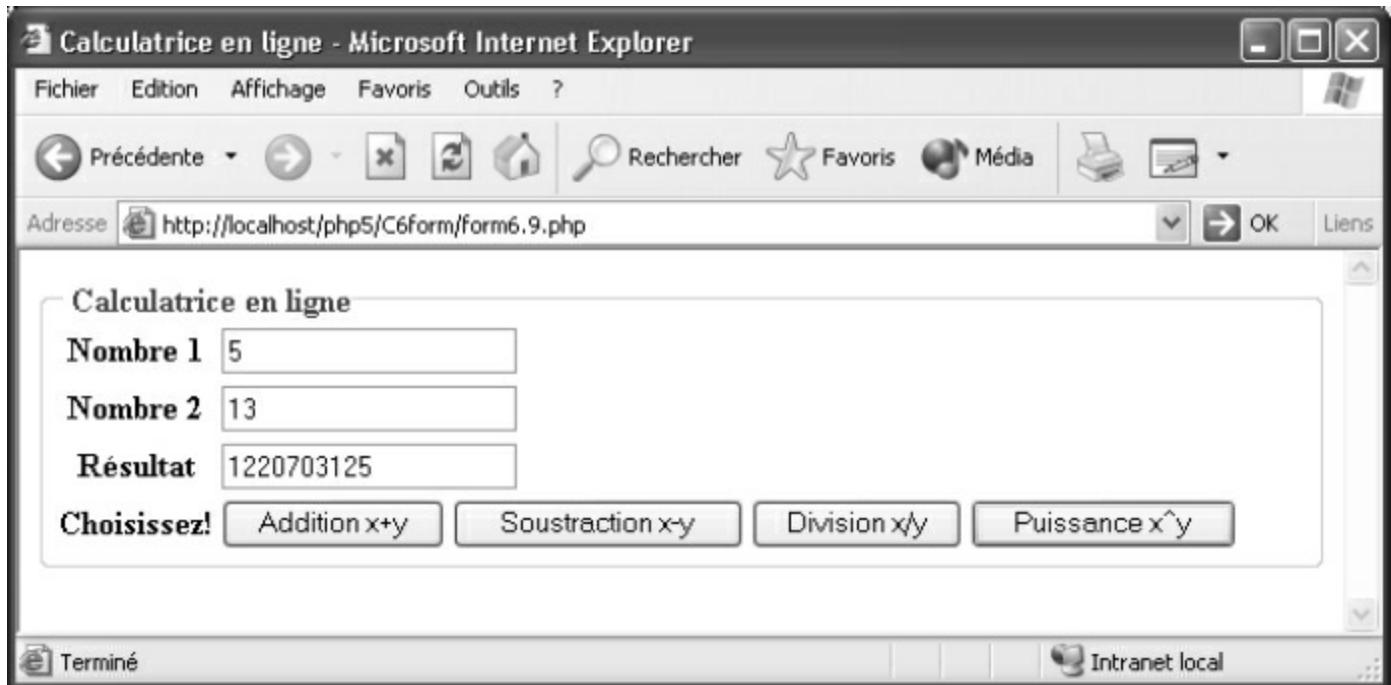

```

```

    <?php if(isset($resultat)) echo $resultat;else echo ''?>" />← 7
</td>
</tr>
<tr>
<th>Choisissez !</th>
<td>
    <input type="submit" name="calcul" value="Addition x+y" />← 1
    <input type="submit" name="calcul" value="Soustraction x-y" />← 2
    <input type="submit" name="calcul" value="Division x/y" />← 3
    <input type="submit" name="calcul" value="Puissance x^y" />← 4
</td>
</tr>
</tbody>
</table>
</fieldset>
</form>
</body>
</html>

```

La figure 6-11 illustre une page de résultat pour la fonction Puissance.



**Figure 6-11**  
*Une calculatrice en ligne*

## Exercices

### Exercice 1

Créez un formulaire comprenant un groupe de champs ayant pour titre "Adresse client". Le groupe doit permettre la saisie du nom, du prénom, de l'adresse, de la ville et du code postal. Les données sont ensuite traitées par un fichier PHP séparé récupérant les données et les affichant dans un tableau HTML.

## Exercice 2

Améliorez le script précédent en vérifiant l'existence des données et en affichant une boîte d'alerte JavaScript si l'une des données est manquante.

## Exercice 3

Le fichier suivant peut-il être enregistré avec l'extension .php ou .html ? Où se fait le traitement des données ?

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Insertion des données</title>
</head>
<body>
<form method="post" action="ajout.php" >
// Suite du formulaire
</form>
</body>
</html>
```

## Exercice 4

Comment faire en sorte que les données soient traitées par le même fichier que celui qui contient le formulaire ? Proposez deux solutions.

## Exercice 5

Créez un formulaire de saisie d'adresse e-mail contenant un champ caché destiné à récupérer le type du navigateur de l'utilisateur. Le code PHP affiche l'adresse et le nom du navigateur dans la même page après vérification de l'existence des données.

## Exercice 6

Créez un formulaire demandant la saisie d'un prix HT et d'un taux de TVA. Le script affiche le montant de la TVA et le prix TTC dans deux zones de texte créées dynamiquement. Le formulaire maintient les données saisies.

## Exercice 7

Créez un formulaire n'effectuant que le transfert de fichiers ZIP et d'une taille limitée à 1 Mo. Le script affiche le nom du fichier du poste client ainsi que la taille du fichier transféré et la confirmation de réception.

## Exercice 8

Dans la perspective de création d'un site d'agence immobilière, créez un formulaire comprenant trois boutons `submit` nommés Vendre, Acheter et Louer. En fonction du choix effectué par le visiteur, redirigez-le vers une page spécialisée dont le contenu réponde au critère choisi.

## Les fonctions

Une fonction est un bloc de code qui n'est pas exécuté de manière linéaire dans un script. Ce code ne le sera que lors de l'appel explicite de la fonction. Écrit une seule fois, ce code peut être exécuté aussi souvent que nécessaire. Cela allège d'autant l'ensemble du code.

### Les fonctions natives de PHP

PHP propose en standard une multitude de fonctions natives écrites en langage C, ainsi que quantité de modules d'extension qu'il est possible d'associer à la distribution standard. Les modules sont tous décrits dans la documentation officielle, et il est recommandé d'utiliser les fonctions qu'ils contiennent plutôt que de les réinventer vous-même.

Avant d'utiliser un module, il convient de vérifier qu'il est bien installé sur le serveur de votre hébergeur. On recense aujourd'hui une bonne centaine d'extensions. Or je ne connais personnellement aucun hébergeur qui en ait installé ne serait-ce que la moitié, loin s'en faut. Pour vérifier la disponibilité d'un module, vous disposez de la fonction `get_loaded_extensions()`, qui retourne un tableau contenant les noms de tous les modules d'extension installés sur le serveur.

En écrivant le script suivant, vous obtenez la liste des modules classée par ordre alphabétique puis affichée à l'aide d'une boucle `foreach` :

```
<?php
$tabext = get_loaded_extensions();
natcasesort($tabext);
foreach($tabext as $cle=>$valeur)
{
    echo " &ampnbsp$valeur ";
}
?>
```

Avec le serveur local MAMP sous MAC, vous obtenez la liste suivante :

---

apache2handler	bcmath	calendar	core	ctype
date	dom	exif	fileinfo	filter
ftp	gd	hash	iconv	imap
intl	json	ldap	libxml	mbstring
mcrypt	mysqli	mysqlnd	openssl	pcre
PDO	pdo_mysql	pdo_pgsql	pdo_sqlite	pgsql
phar	posix	Reflection	session	SimpleXML
soap	sockets	SPL	sqlite3	standard
tokenizer	wddx	xml	xmlreader	xmlwriter
xls	zip	zlib		

---

et sur le serveur PHP 7 de l'hébergeur OVH :

---

bcmath	bz2	calendar	cgi_fcg	ctype
curl	date	dba	dom	exif
fileinfo	filter	ftp	gd	gettext
gmp	hash	iconv	imap	intl
json	libxml	mbstring	mcrypt	mysqli
openssl	pcre	pdo_mysql	PDO	pdo_mysql
pdo_pgsql	pdo_sqlite	pgsql	Phar	posix
pspell	Reflection	session	SimpleXML	soap
sockets	SPL	SQLite3	standard	sysvmsg
sysvsem	sysvshm	tokenizer	wddx	xml
xmlreader	xmlrpc	xmlwriter	xsl	ZendOPcache
zip	zlib			

---

Soit pas moins de 57 extensions !

Vous constatez que toutes les extensions ne sont pas communes entre le serveur local installé avec MAMP 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, car 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
$stabext = get_loaded_extensions();
natcasesort($stabext); //tri par ordre alphabétique
//Lecture des extensions
foreach($stabext as $cle=>$valeur)
{
    echo "<h3>Extension  &nbsp;{$valeur} </h3> ";
    //Tableau contenant le nom des fonctions
    $fonct = get_extension_funcs($valeur);
    //Tri alphabétique des noms de fonction
    sort($fonct);
    //Lecture et affichage du nom des fonctions des extensions
    for($i=0;$i<count($fonct);$i++)
    {
        echo $fonct[$i], "  &nbsp; &nbsp; \n";
        echo "<hr />";
    }
}
?>
```

La figure 7-1 illustre la liste impressionnante de fonctions que vous obtenez. Par simple copier-coller, vous obtenez à bon marché un mémo des fonctions utilisables sur votre serveur.

Pour vérifier la disponibilité d'une fonction native de PHP ou d'une fonction personnalisée, vous pouvez tester sa présence à l'aide de la fonction `function_exists("nom_fonction")`, qui renvoie la valeur `TRUE` si la fonction passée en paramètre existe et `FALSE` dans le cas contraire.

Pour vérifier, par exemple, que la fonction `mysql_pconnect()` est utilisable, vous pouvez écrire :

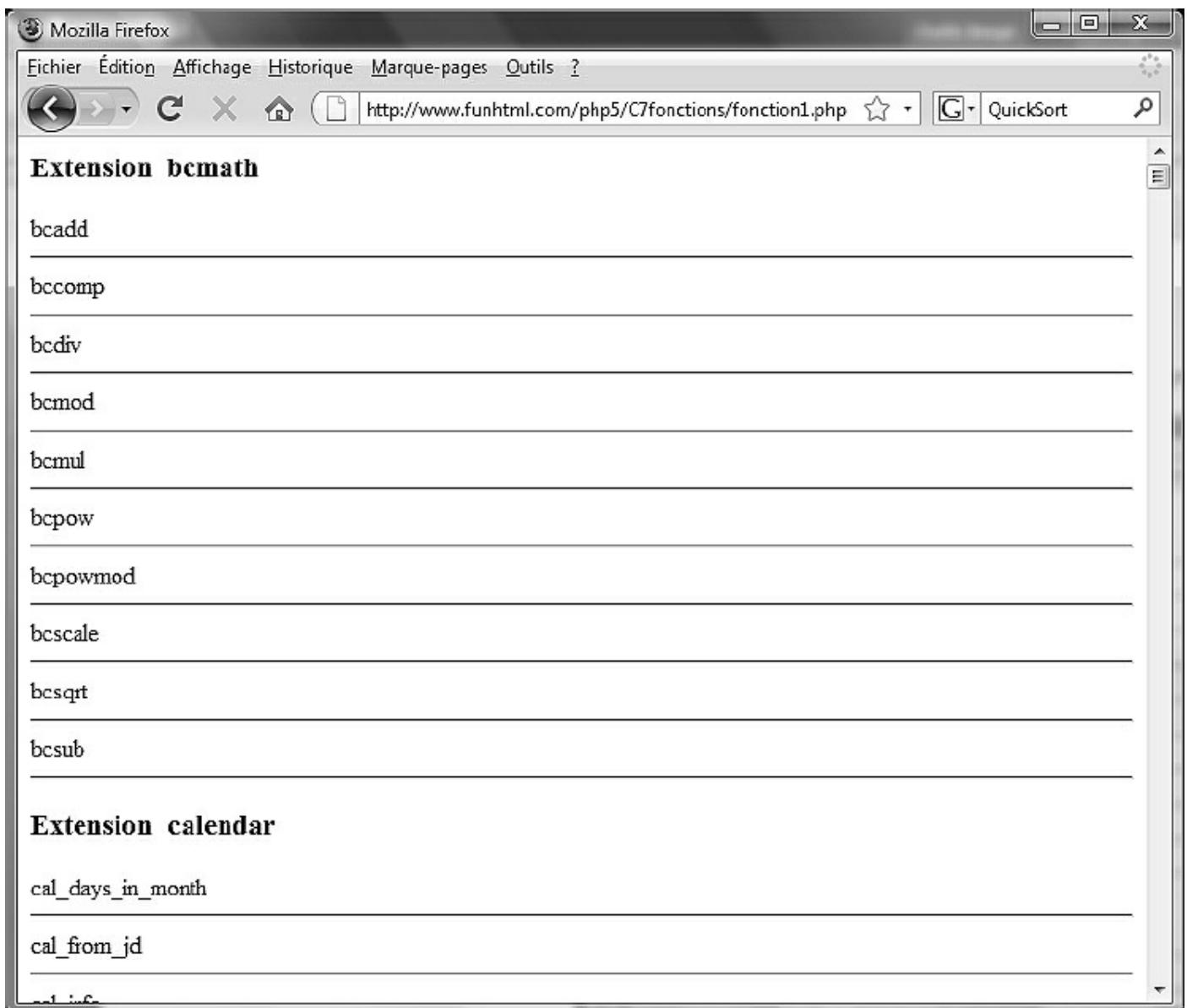
```
if (function_exists('mysql_pconnect'))  
{  
    echo "La fonction est utilisable.<br />";  
}
```

## Créer ses propres fonctions

Malgré les quelques deux mille fonctions contenues dans l'ensemble des modules existants à ce jour, il arrive, faute de fonction disponible, de devoir effectuer plusieurs fois le même traitement dans un script. Heureusement, il est possible de créer des fonctions personnalisées, qui permettent tout à la fois de gagner du temps de saisie et d'alléger votre script en ne répétant pas plusieurs fois un même code.

Au fur et à mesure de l'écriture de vos scripts, vous pouvez de la sorte réutiliser du code déjà écrit dans un script précédent. Il vous faut pour cela identifier, dès la conception du ou des sites que vous souhaitez réaliser, les tâches susceptibles de se retrouver dans d'autres situations et qui ne sont pas prises en compte par une fonction

existante de PHP.



**Figure 7-1**

*Liste des fonctions disponibles sur le serveur de l'hébergeur OVH*

Ces fonctions personnalisées doivent être écrites dans un ou des scripts séparés, qu'il vous suffit ensuite d'inclure dans de nouveaux scripts au moyen d'une instruction `include()` ou `require()`.

## **Définir une fonction**

En règle générale, une fonction peut être définie n'importe où dans un script, y compris après avoir été utilisée. Il existe cependant des exceptions, rares, comme nous le verrons dans la section « Cas particuliers » en fin de chapitre.

La procédure la plus simple de déclaration d'une fonction doit suivre les règles de syntaxe suivantes, mais nous verrons dans la suite qu'on peut les perfectionner en typant les paramètres et les valeurs retournés.

- Commencez par définir l'en-tête de la fonction à l'aide du mot-clé `function` suivi du nom de la fonction et de parenthèses. Les noms de fonctions suivent les règles énoncées pour les variables : caractères alphabétiques et signe « `_` » pour le premier caractère puis caractères alphanumériques pour la suite. Il est déconseillé de commencer un nom de fonction par deux caractères de soulignement, car cette convention est réservée pour définir des fonctions natives de PHP. La redéfinition de fonction étant interdite dans PHP, contrairement à ce qui se fait dans d'autres langages, il est interdit de créer une fonction dont le nom soit identique à une fonction existante dans un des modules installés. En contrevenant à cette règle, vous vous exposez à une erreur fatale et au méchant message suivant, qui met fin au script :

```
| "Fatal error: Cannot redeclare date() in c:\eyrolles\php5\fonctions\fonction2.php
|   on line 3"
```

Dans cet exemple, vous souhaitez définir une fonction nommée `date()` au lieu de `ladata()`, comme à l'exemple 7-2, alors qu'il existe déjà une fonction portant ce nom dans le module standard.

- Les parenthèses qui suivent le nom de la fonction peuvent contenir différents noms de variables comme paramètres de la fonction. Les noms des paramètres n'ont aucune importance en soi, et vous pouvez même leur donner des noms qui existent déjà dans le script, quoique ce ne soit pas conseillé. L'utilisation des paramètres n'est pas obligatoire.
- Ouvrez un bloc de code limité par des accolades contenant l'ensemble du code de définition de la fonction. Cette partie, qui constitue le corps de la fonction, peut contenir toutes les instructions de PHP ainsi que n'importe quelle fonction native.
- Si la fonction doit retourner une valeur, ce qui n'est pas obligatoire, il faut faire précéder la variable ou l'expression qui contient cette valeur du mot-clé `return`.

Vous obtenez la structure générale suivante :

```
| function mafonction($x,$y,...)
{
    //code de définition de la fonction
    return $var;
}
```

Pour appeler la fonction, vous écrivez :

```
| mafonction($a,$b,...)
```

ou encore :

```
| mafonction(4,5,...)
```

Les paramètres peuvent être ici indifféremment des scalaires ou des variables. Il importe dans les deux cas de donner à la fonction autant de paramètres que dans sa déclaration, sauf si vous avez défini des paramètres par défaut (voir plus loin).

La position de la déclaration d'une fonction dans le script n'a pas d'importance depuis PHP 4. Vous pouvez ainsi appeler une fonction au début du script alors qu'elle n'est

définie qu'en fin de script. Il est toutefois préférable de définir les fonctions en début de script, comme dans PHP 3, car cela améliore la présentation et la lisibilité du code.

Pour les fonctions définies dans des scripts séparés, il est préférable de les inclure dès le début du script qui les utilise, et ce au moyen de l'instruction `include()` ou `require()`.

## ***Les fonctions qui ne retournent pas de valeur***

Dans l'exemple 7-2, vous créez deux fonctions qui ne retournent pas de valeur. La première, `ladate()`, crée un affichage de la date et de l'heure sans aucun paramètre, et la seconde `ladate2()`, y ajoute un message qui est passé en paramètre unique.

Ces deux fonctions créent un affichage dans une cellule de tableau HTML munie d'un style particulier, qui peut être le même dans toutes les pages du même site.

### **Pour en savoir plus**

Pour plus de détails sur la syntaxe de la fonction `date()` utilisée dans le script, reportez-vous au chapitre 8.

Lors des appels de ces fonctions, les trois premiers appels fonctionnent sans problème. En revanche, l'appel de la fonction `ladate2()` sans paramètre, alors même que sa définition en comporte un, peut éventuellement provoquer un avertissement, ou "warning" – ceci étant variable selon les serveurs –, mais sans mettre fin pour autant au script, à la différence d'une erreur fatale. Pour éviter ce genre de problème, vous pouvez faire précéder, lors de son appel, le nom de la fonction par le caractère `@`, qui empêche l'apparition du message d'alerte.

La figure 7-2 illustre le résultat de ces différents appels.

### **Exemple 7-2. Fonctions ne retournant pas de valeur**

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Fonctions ne retournant pas de valeur</title>
</head>

<body>
<div>

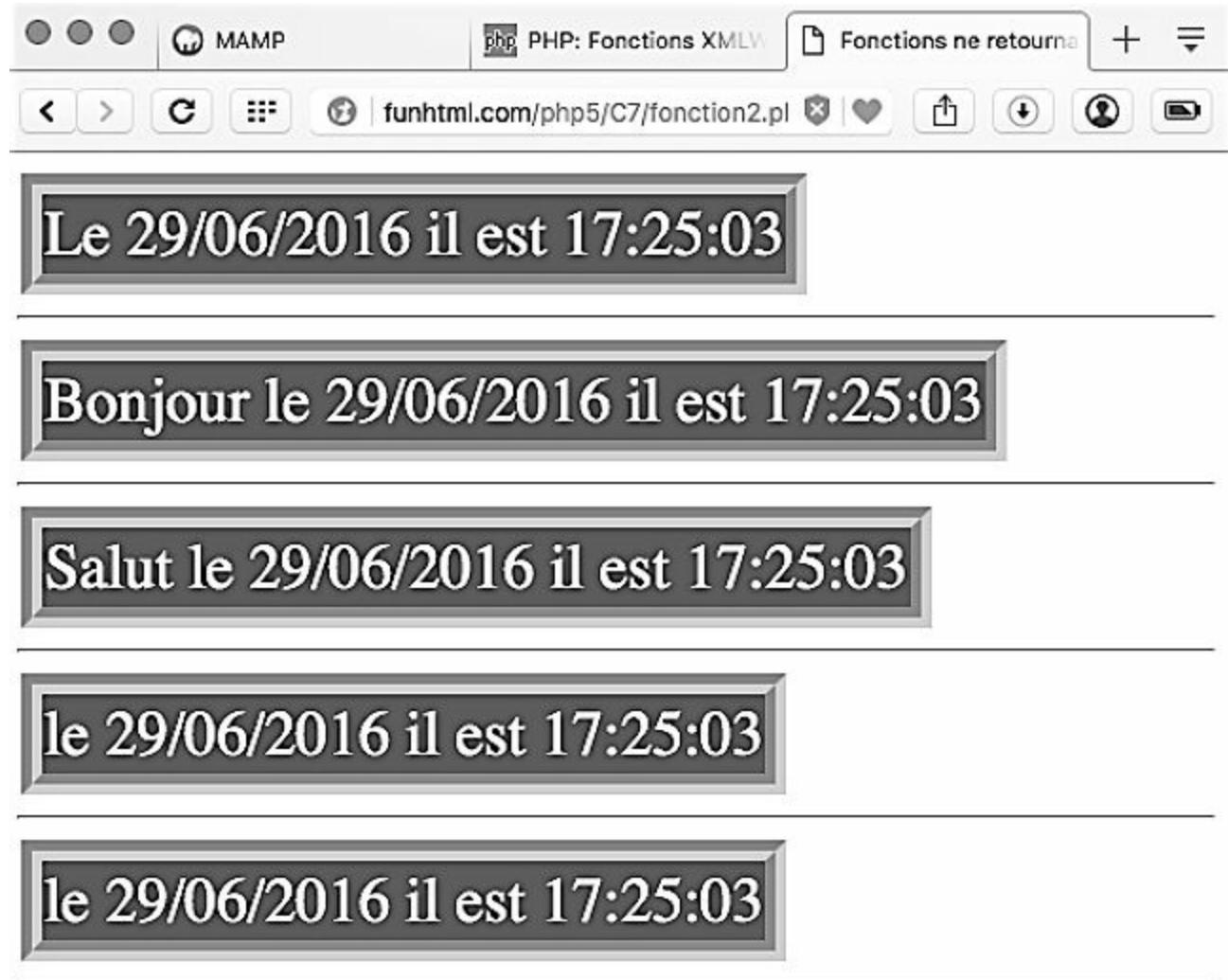
<?php
// Fonction sans argument
function ladate()
{
    echo "<table ><tr><td style=\"background-color:blue;color:yellow;
border-width:10px; border-style:groove; border-color:#FFCC66;
font-style:fantasy;font-size:30px\"> ";
    echo date("\l\le d/m/Y \i\l \\e\s\\t H:i:s");
    echo "</td></tr></table><hr />";
}


```

```

// Fonction avec un argument
function ladate2($a)
{
    echo "<table><tr><td style=\"background-color:blue;color:yellow;
border-width:10px; border-style:groove;border-color:#FFCC66; font-style:fantasy;
font-size:30px\">";
    echo "$a ",date("l\\e d/m/Y \i\l \\e\s\t H:i:s");
    echo "</td></tr></table><hr />";
}
// Appels des fonctions
echo ladate();
echo ladate2("Bonjour");
echo ladate2("Salut");
echo ladate2(); // Provoque un avertissement (Warning)
echo @ladate2(); // Empêche l'apparition du message d'avertissement
?>
</div>
</body>
</html>

```



**Figure 7-2**  
*Fonctions d'affichage de la date*

Vous avez vu en détail au chapitre 5 la manière de lire l'intégralité d'un tableau. Si vous utilisez souvent ce tableau, vous avez tout intérêt à créer une fonction de lecture du tableau et d'affichage de leurs éléments sous la forme d'un tableau HTML à deux

colonnes par exemple.

Cet exemple peut être réutilisé ou adapté pour afficher les résultats de l'interrogation d'une base de données, lorsque les résultats sont récupérés sous forme de tableau, ce qui est souvent le cas. C'est l'objet de l'exemple 7-3, qui lit un tableau unidimensionnel.

Les paramètres de la fonction sont, dans l'ordre, le nom du tableau, la largeur de la bordure des cellules HTML et les libellés des colonnes. Le corps de la fonction est essentiellement composé d'une boucle chargée de lire les clés et les valeurs des éléments du tableau et ne présente pas de difficultés. Comme la première, cette fonction ne retourne pas de valeur mais crée un affichage HTML.

La figure 7-3 donne un exemple du résultat obtenu pour des tableaux de données.

### Exemple 7.3. Fonction de lecture de tableaux

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Fonction de lecture de tableaux</title>
</head>
<body>
<div>

<?php
// Définition de la fonction
function tabuni($tab,$bord,$lib1,$lib2)
{
    echo "<table border=\"$bord\" width=\"100%\"><tbody><tr><th>$lib1</th> <th>$lib2
</th></
tr>";
    foreach($tab as $cle=>$valeur)
    {
        echo "<tr><td>$cle</td> <td>$valeur </td></tr>";
    }
    echo "</tbody> </table><br />";
}
// Définition des tableaux
$tab1 = array("France"=>"Paris","Allemagne"=>"Berlin","Espagne"=>"Madrid");
$tab2 = array("Poisson"=>"Requin","Cetacophyline"=>"Dauphin","Oiseau"=>"Aigle");
// Appels de la fonction
tabuni($tab1,1,"Pays","Capitale");
tabuni($tab2,6,"Genre","Espèce");
?>
</div>
</body>
</html>
```

### *Typage des paramètres*

Il est désormais possible de définir un typage des paramètres d'une fonction. Les types qu'on peut exiger sont `int`, `float`, `string`, `bool`, `array`, un nom de classe et d'interface.

Il existe deux sortes de typage en PHP 7, le typage faible et le typage strict (dit aussi « fort »).

En écrivant par exemple la définition de fonction suivante en typage faible (qui est le choix par défaut) :

The screenshot shows a web browser window titled "Fonction de lecture de tableau". The address bar indicates the URL is "localhost:8888/fonction3.php". Below the title bar are standard browser controls. The main content area displays two tables side-by-side.

Pays	Capitale
France	Paris
Allemagne	Berlin
Espagne	Madrid

Genre	Espèce
Poisson	Requin
Cétacé.	Dauphin
Oiseau	Aigle

**Figure 7-3**  
Affichages réalisés par la fonction tabuni()

```
function somme(int $a, int $b) {  
    return $a+$b; }
```

la fonction attend deux paramètres entiers. Si ce n'est pas le cas, un transtypage est effectué quand c'est possible. Par exemple:

```
echo somme(5,9); //Pas de problème, retourne 14  
echo somme('5.4',9.7);  
/*5.4 et 9.7 sont convertis en entiers 5 et 9 donc retourne 14*/  
echo somme('5.4',9);  
/*La chaîne '5.4' est convertie en entier 5 donc affiche 14 */  
echo somme(5.4, FALSE);  
/* FALSE est évalué à 0 donc affiche 5*/  
echo somme(5.4, TRUE);  
/* TRUE est évalué à 1 donc affiche 6*/
```

Les types ne sont pas nécessairement les mêmes dans la définition d'une fonction, comme dans l'exemple suivant :

```
function somme2(string $a,float $b){  
    return $a+$b;  
}
```

La fonction attend dans cet ordre, une chaîne puis un décimal. De même que précédemment, si ces conditions ne sont pas remplies il y aura transtypage dans le but ici d'effectuer une addition, par exemple :

```
echo somme2('5.4',9.7);  
/* la chaîne '5.4' est convertie en décimal 5.4 donc la somme est 15.1 */  
echo $s=somme2('5.4',TRUE);
```

```
/* La chaîne '5.4' est encore convertie en décimal 5.4 et TRUE évalué à 1 donc affiche 6.4 */
```

En typage strict, la fonction va exiger le strict respect du type de chaque paramètre défini dans la fonction, sinon il se produit une erreur. Pour choisir le typage strict, il faut placer une directive en tête du fichier (et surtout pas ailleurs) qui contient la définition de la fonction, sous la forme :

```
declare(strict_types=1);
```

En écrivant le code suivant :

```
<?php  
declare(strict_types=1);  
  
function somme2(int $a, float $b)  
{  
    return $a+$b;  
}  
echo somme2(5,9.7) // Affiche 14.7  
?>
```

Dans ce cas, aucune valeur non entière pour le premier paramètre et autre que décimale (attention un entier est aussi un décimal !) pour le second provoquera une erreur.

## ***Les fonctions qui retournent une valeur***

Au sens mathématique du terme, une fonction se doit de retourner une valeur calculée à partir des paramètres qui lui sont passés. Vous allez illustrer cette possibilité en créant une fonction qui calcule et retourne la mensualité à payer pour un prêt en fonction du capital emprunté, du taux d'intérêt et de la durée du prêt.

Ce type de fonction financière se retrouve couramment, dans Excel par exemple. En revanche, elle n'existe pas nativement dans PHP ni dans aucun module additionnel. Sa définition nécessite quelques commentaires pour en comprendre l'écriture.

La formule de calcul d'une mensualité  $M$  est la suivante :

$$M = (C \times T) / [1 - (1 + T)^{-n}]$$

$C$  est le capital emprunté, désigné par le paramètre `$capital`.

$T$  est le taux mensuel sous forme de nombre décimal — tel que ne nous l'indique jamais notre banquier, qui préfère nous donner un taux annuel. Ce paramètre annuel est passé à la fonction et désigné par la variable `$tauxann`.  $n$  est la durée en nombre de mois. Le paramètre correspondant fourni à la fonction est exprimé en années au moyen du paramètre `$dureeann` puis est converti en mois. Le quatrième paramètre de la fonction, `$assur`, permet de calculer le montant de l'assurance. Il vaut `1` si le client désire s'assurer et `0` dans le cas contraire.

Dans la définition de la fonction, nous définissons un typage strict en introduisant une

directive au début du fichier (repère 1) pour que le capital et la durée soient des entiers, le taux un décimal, et le choix de l'assurance un booléen (repère 2). Dans la pratique, ces données proviendraient d'un formulaire de saisie et il faudrait gérer les éventuelles erreurs de l'utilisateur.

Le corps de la fonction commence par adapter les données bancaires usuelles à la formule du calcul financier. Le taux annuel, par exemple, est converti en taux mensuel décimal. Par exemple, 5 % l'an devient 5/100/12, soit 0.00416666. La fonction calcule ensuite la mensualité au moyen de la formule ci-dessus puis retourne cette valeur arrondie au centime près à l'aide de la fonction `round()`.

Le script de l'exemple 7-4 s'achève par l'utilisation de cette fonction avec des paramètres scalaires puis avec des variables.

## Exemple 7-4. Fonction de calcul de prêt

```
<?php
declare(strict_types=1);←①
?>
<!DOCTYPE html > <html> <head> <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1" /> <title>CALCUL DE PRÊTS</title> </head> <body> <?php
function mensualite(int $capital, float $tauxann, int $dureeann, bool $assur)←②
{
    // Calcul du taux mensuel décimal
    $tauxmensuel=$tauxann/100/12;
    // Calcul de la durée en mois
    $duree=$dureeann*12;
    // Calcul de l'assurance soit 0,035 % du capital par mois
    $ass=$assur*$capital*0.00035; //vaut 0 si $assur vaut 0
    // Calcul de la mensualité
    $mens=($capital*$tauxmensuel)/(1-pow((1+$tauxmensuel),-$duree)) + $ass;
    return round($mens,2);
}
$mens = mensualite(100000,5,3,TRUE);
echo "<h3>Pour un prêt de 100 000 € à 5 % l'an sur 3 ans, la mensualité est de
".$mens ." €
assurance comprise</h3>"; //*****
$cap=800000;$taux=3.5;$duree=10; $mens = mensualite($cap,$taux,$duree,FALSE);
echo "<h3>Pour un prêt de ".$cap," € à ".$taux, " % l'an sur ".$duree." ans, la
mensualité
est de ".$mens ." € sans assurance </h3>";
?>
</body>
</html>
```

Le script retourne le résultat suivant :

---

Pour un prêt de 100 000 € à 5 % l'an sur 3 ans, la mensualité est de 3 032.09 €  
assurance  
comprise

Pour un prêt de 800 000 € à 3.5 % l'an sur 10 ans, la mensualité est de 7 910.87 € sans  
assurance

---

Au chapitre 6, vous avez déjà réalisé un formulaire de saisie de données pour un calcul

de prêt. La fonction que vous venez de créer peut être incorporée à ce script pour rendre l'ensemble plus élégant.

Un autre point ajouté à PHP 7 consiste à pouvoir définir le type de la valeur renvoyée par une fonction. Les types acceptés sont les mêmes que pour le typage des paramètres. Le typage peut également être faible (par défaut) ou strict avec la même directive que précédemment.

Pour typer les valeurs de retour, il faut faire suivre la liste des paramètres du type voulu précédé du caractère : (deux-points). Dans l'exemple suivant, nous définissons un typage strict (repère 1) pour une fonction `somme()` dont les deux paramètres sont des chaînes et dont la valeur de retour est un `float` (repère 2)

```
<?php
declare(strict_types=1);←①

function somme(string $a,string $b):float←②
{
    return $a+$b;
}

echo $s=somme('7','15.8'); //Affiche 22.8
echo "<hr/>",var_dump($s); //Affiche float(22.8)
?>
```

Le contenu des variables `$a` et `$b` qui doivent être des chaînes, sera transposé en nombre et le résultat devra être un décimal, ce qui est évident ici pour une opération arithmétique.

## Retourner plusieurs valeurs

PHP n'offre pas la possibilité de retourner explicitement plusieurs variables à l'aide d'une syntaxe du type :

```
return $a,$b,$c,...
```

Pour pallier cet inconvénient, il suffit de retourner une variable de type `array` contenant autant de valeurs que désiré. Vous pourriez aussi envisager de retourner un objet doté de plusieurs propriétés, mais il serait quelque peu fastidieux de créer une classe et des objets spécialement pour la circonstance si la création d'objets ne faisait pas partie de la conception générale du script.

L'exemple 7-5 illustre le retour de plusieurs valeurs par le biais de la création d'une fonction appliquée à un nombre complexe qui retourne à la fois le module et l'argument du nombre.

### Rappel

Un nombre complexe s'écrit  $a + ib$ , avec  $i^2 = -1$ . Son module est égal à la racine carrée de  $(a^2 + b^2)$ . Son argument est l'angle  $(Ox, OM)$  en radian si le point M a pour coordonnées

( $a, b$ ).

La fonction `modarg()` reçoit comme paramètres les nombres  $a$  et  $b$  passés dans les variables `$reel` et `$imag` et retourne un tableau associatif dont les clés sont les chaînes "module" et "argument" avec les valeurs associées correspondantes.

## Exemple 7-5. Fonction de calcul sur des nombres complexes

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Nombres complexes</title>
</head>
<body>
<div>
<?php
function modarg($reel,$imag)
{
    // $mod= hypot($reel,$imag);
    // ou encore si vous n'avez pas la fonction hypot() du module standard
    $mod =sqrt($reel*$reel + $imag*$imag);
    $arg = atan2($imag,$reel);
    return array("module"=>$mod, "argument"=>$arg);
}
// Appels de la fonction
$a= 5;
$b= 8;
$complex= modarg($a,$b);
echo "<b>Nombre complexe $a + $b i:</b><br /> module = ", $complex["module"] ,
    "<br />argument = ", $complex["argument"], " radians<br />";
?>
</div>
</body>
</html>
```

Le script retourne le résultat suivant :

---

```
Nombre complexe 5 + 8 i:
module = 9.4339811320566
argument = 1.0121970114513 radians
```

---

Grâce à l'artifice consistant à retourner un tableau de valeurs, vous pouvez créer des fonctions qui retournent autant de valeurs que désiré. Vous utiliserez cette dernière méthode systématiquement pour retourner plusieurs valeurs.

## *Les paramètres par défaut*

Il n'est pas rare dans l'écriture de code HTML de ne pas définir certaines valeurs d'attributs d'un élément donné car vous savez qu'il prendra automatiquement une valeur par défaut.

Par exemple, le formulaire écrit de la façon suivante :

```
<form action="machin.php">
```

est l'équivalent de celui-ci :

```
| <form action="machin.php" method="get" enctype="application/x-www-form-urlencoded">
```

Les attributs `method` et `enctype` ayant les valeurs par défaut ci-dessus il n'est pas nécessaire de les préciser si ces valeurs vous conviennent.

Lorsque vous créez une fonction personnalisée, vous pouvez procéder de même et donner des valeurs par défaut aux paramètres de la fonction. Il suffit pour cela d'affecter une valeur au paramètre dans la définition de la fonction (repère 1 de l'exemple 7-6).

Si vous ne donnez pas de valeur à ces paramètres lors de l'appel de la fonction, ils auront automatiquement la valeur définie dans la déclaration de la fonction (repère 2).

L'exemple 7-6 illustre la définition d'une fonction élémentaire qui retourne un prix hors taxes en utilisant comme paramètres le prix TTC et le taux de TVA.

En réalisant les appels :

```
| echo ht(154,19.6)
```

ou :

```
| echo ht(154)
```

vous obtenez le même résultat, à savoir 123.82, le paramètre `$tax` désignant le taux de TVA valant 19.6 par défaut.

En revanche, l'appel suivant :

```
| echo ht(154, 5.5)
```

retourne la valeur 145.53 car le paramètre `$tax` est défini explicitement à la valeur 5.5.

## Exemple 7-6. Fonction avec une valeur par défaut

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Fonction avec une valeur de paramètre par défaut</title>
</head>
<body>
<div>

<?php
function ht($prix,$tax=19.6)←①
{
    return "Prix Hors Taxes :". round($prix*(1-$tax/100),2);
}
$prix=154;
echo "Prix TTC= $prix &euro; ",ht($prix)," &euro;<br />";←②
echo "Prix TTC= $prix &euro; ",ht($prix,19.6)," &euro;<br />";
echo "Prix TTC= $prix &euro; ",ht($prix,5.5)," &euro;<br />";
?>
</div>
</body>
```

```
</html>
```

## Attention

Dans la définition d'une fonction, tous les paramètres qui ont une valeur par défaut doivent figurer en dernier dans la liste des variables.

La fonction `ht()` de l'exemple 7-6 peut être appelée en omettant le deuxième paramètre avec `ht(154)`. En revanche, si vous la définissez par le biais de :

```
function ht($tax=19.6,$prix) {....}
```

et que vous tentiez de l'appeler à l'aide du code `ht(,154)` ou `ht(154)`, elle ne fonctionne pas, le premier code provoquant une erreur fatale et un arrêt du script et le second un avertissement indiquant qu'il manque un argument.

## Les fonctions avec un nombre de paramètres variable

Dans les définitions précédentes de fonctions, vous aviez l'obligation de définir clairement le nombre de paramètres utilisés par la fonction. Cela pose un problème si le nombre de paramètres n'est pas connu à l'avance. Il existe heureusement plusieurs méthodes, adaptées à différentes circonstances, pour créer des fonctions acceptant un nombre variable de paramètres.

### *Les paramètres de type array*

En passant un tableau comme paramètre à une fonction, cette dernière n'a en apparence qu'un seul paramètre. Ce sont en fait les éléments du tableau qui sont utilisés et traités par la fonction, chacun devenant comme un paramètre particulier. C'est donc dans le corps de la fonction que vous pourrez déterminer le nombre d'éléments du tableau et utiliser cet ensemble de valeurs, qui seront lues à l'aide d'une boucle.

La fonction créée à l'exemple 7-7 réalise le produit de  $N$  nombres qui lui sont passés en tant qu'éléments du tableau `$tab`, affiche le nombre de paramètres et retourne leur produit. Elle peut, par exemple, servir au calcul de la factorielle d'un entier sans pour autant être récursive, pour peu que le tableau contienne une suite de nombres créée au moyen de la fonction `range()`.

### Exemple 7-7. Produit des éléments d'un tableau

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Nombre de paramètres variable</title>
</head>
<body>
<div>
<?php
function prod($tab)
{

```

```

$n=count($tab);
echo "Il y a $n paramètres :";
$prod = 1;
foreach($tab as $val)
{
    echo "$val, ";
    $prod *= $val;
}
echo " le produit vaut ";
return $prod;
}

$tab1= range(1,10);
echo "Produit des nombres de 1 à 10 : ", prod($tab1), "<br />";
$tab2 = array(7,12,15,3,21);
echo "Produit des éléments : ", prod($tab2), "<br />";
?>
</div>
</body>
</html>

```

Le script retourne le résultat suivant :

---

```

Produit des nombres de 1 à 10 : Il y a 10 paramètres : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
le produit
vaut 3628800
Produit des éléments : Il y a 5 paramètres : 7, 12, 15, 3, 21, le produit vaut 79380

```

---

## ***Les fonctions particulières de PHP***

Vous pouvez obtenir des informations sur les paramètres d'une fonction à l'aide de fonctions spécialisées de PHP.

La fonction suivante :

| integer func\_num\_args()

s'utilise sans argument et seulement dans le corps même d'une fonction. Elle retourne le nombre d'arguments passés à cette fonction.

Pour accéder à chacun des paramètres, vous utilisez ensuite la fonction :

| divers func\_get\_arg(integer \$N)

qui retourne la valeur du paramètre passé à la position `$N`. Comme dans les tableaux, le premier paramètre a l'indice `0`. Vous pouvez donc lire chacun des paramètres à l'aide d'une boucle.

La fonction suivante :

| array func\_get\_args()

s'utilise sans paramètre et retourne un tableau indicé contenant tous les paramètres passés à la fonction personnalisée.

Pour illustrer l'utilisation de ces deux fonctions, vous allez réécrire la fonction `prod()` de l'exemple 7-7 pour détecter le nombre et la valeur des paramètres sous deux formes

différentes et en faire le produit.

La fonction `prod1()` détermine le nombre de paramètres à l'aide de `func_num_args()` puis les lit un par un à l'aide d'une boucle `for`. La fonction `prod2()` récupère le tableau contenant l'ensemble des paramètres à l'aide de `func_get_args()` puis les récupère à l'aide d'une boucle `foreach`.

### Remarque

La définition des fonctions `prod1()` et `prod2()` ne contient plus l'énumération des paramètres mais ce n'est pas obligatoire. Vous pouvez très bien préciser un nombre minimal de paramètres et en passer davantage — mais jamais moins — lors de l'appel

## Exemple 7-8. Produit d'un nombre indéterminé de nombres

```
<!DOCTYPE html>
<html lang="fr">
</head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Produit d'un nombre indéterminé d'arguments</title>
</head>
<body>
<div>
<?php
// Utilisation de func_num_arg() et func_get_arg()
function prod1()
{
    $prod = 1;
    // Détermine le nombre d'arguments
    $n=func_num_args();
    for($i=0;$i<$n;$i++)
    {
        echo func_get_arg($i);
        ($i==$n-1)?print(" = "):print(" * ");
        $prod *=func_get_arg($i);
    }
    return $prod;
}
// Appels de la fonction prod1()
echo "Produit des nombres :", prod1(5,6,7,8,11,15), "<br />";
$a=55;$b=22;$c=5;$d=9;
echo "Produit de ",prod1($a,$b,$c,$d), "<hr />";
//*****//
//Utilisation de func_get_args() seule
//*****//
function prod2()
{
    $prod = 1;
    // Récupération des paramètres dans un tableau
    $tabparam = func_get_args();
    foreach($tabparam as $cle=>$val)
    {
        // Présentation
        echo $val;
        ($cle==count($tabparam)-1)?print(" = "):print(" * ");
        // Calcul du produit
        $prod *=val;
    }
}
```

```

    }
    return $prod;
}
echo "Produit des nombres : ", prod2(5,6,7,8,11,15), "<br />";
$a=55;$b=22;$c=5;$d=9;
echo "Produit de ",prod2($a,$b,$c,$d),"<hr />";
?>
</div>
</body>
</html>

```

Vous obtenez deux fois l'affichage suivant pour les deux fonctions :

---

```

Produit des nombres : 5 * 6 * 7 * 8 * 11 * 15 = 277200
Produit de 55 * 22 * 5 * 9 = 54450

```

---

## L'opérateur ...

Il existe maintenant une alternative à la fonction `func_get_args()` que nous venons de voir. Il s'agit de l'opérateur `...` qui permet de passer à la fonction un nombre variable de paramètres, sans passer explicitement en paramètre un tableau qui les contient ni sans avoir à le définir, comme nous l'avons fait dans l'exemple 7-7, en utilisant la syntaxe suivante :

```

function mafonction($var,...$tab)
function mafonction($var,...$tab)

```

où `$var1` a une valeur et où `$tab` est traité comme un tableau contenant un nombre aléatoire d'autres valeurs. Dans l'exemple 7-9, nous réécrivons la fonction `prod()` vue plus haut à l'aide de cette méthode. La liste des paramètres comprend une variable `$a` puis l'opérateur `...`, suivi de la variable `$tab` (repère 1). Dans le corps de la fonction, une boucle `foreach` lit le contenu de la liste des autres variables comme si c'était un tableau (repère 2), effectue le produit et retourne le résultat classiquement. Pour les deux premiers appels, on remarque qu'on peut passer 4 puis 5 valeurs sans avoir à créer un tableau comme dans l'exemple 7-7 (repères 3 et 4). La plus grande souplesse de cette méthode se trouve démontrée dans les appels suivants dans lesquels nous passons d'abord un tableau seul (repère 5) – mais déclaré auparavant –, puis une valeur et un tableau (repère 6), et même pour la démonstration uniquement, une seule valeur (repère 7).

### Exemple 7-9. Produit d'un nombre indéterminé de nombres avec l'opérateur ...

```

<?php
function prod($a,...$tab)←①
{
    foreach($tab as $nb)←②
    {
        $a *= $nb;
    }
    echo " Le produit vaut ";
    return $a;
}

```

```

}
echo "Produit 1",prod(2,3,5,7),"<br/>";←③ //210
echo "Produit 2",prod(4,9,11,15,20),"<br/>";←④ //118800
$tab1= range(1,10);
echo "Produit 3",prod(...$tab1),"<br/>";←⑤ //3628800
$tab2 = array(7,12,15,3,21);
echo "Produit 4",prod(2,...$tab2),"<br/>";←⑥ //158760
echo prod(7);←⑦ // 7 bien sûr!
?>

```

## Les générateurs

### *Création d'un générateur*

Afin de retourner plusieurs valeurs successives, les générateurs, ajoutés dans la version 5.5, permettent de créer des itérateurs sans créer un objet spécialisé et surtout sans créer aucun tableau dans la fonction comme c'était le cas dans l'exemple 7-5. Cela est réalisé à l'aide du mot-clé `yield` qui permet à la fonction de retourner plusieurs valeurs et de les lire via une boucle `foreach` comme s'il s'agissait d'un tableau. Cette méthode, si elle n'est pas nécessairement plus rapide, a l'avantage de consommer beaucoup moins de mémoire sur le serveur – particulièrement quand le nombre de valeurs à retourner est important.

L'exemple 7-10 illustre cette possibilité en créant une fonction nommée `suite()` qui retourne une suite arithmétique dont la raison est contenue dans la variable `$pas`, entre un minimum `$min` et un maximum `$max` (repère ①). L'appel de la fonction retourne un objet de type `Generator` qui est lu par la boucle `foreach` (repère ②) et permet l'affichage de tous les résultats itérés grâce à `yield`. L'expression qui suit `yield` peut être composée comme : `yield "Nombre".$i` qui crée la concaténation d'une chaîne et d'une variable.

### ➤ Exemple 7-10. Suite numérique à l'aide d'un générateur

```

<?php
function suite($min,$max,$pas)←①
{
    for($i=$min;$i<=$max;$i+=$pas)
    {
        yield $i;
    }
}
echo "Suite : ";
foreach(suite(13,50,5)as $nb)←②
{echo $nb, " /// ";}
?>

```

Les résultats affichés sont :

---

Multiples : 13 /// 18 /// 23 /// 28 /// 33 /// 38 /// 43 /// 48 ///

---

Le générateur se lisant comme un tableau, il est possible de récupérer les indices dans la boucle `foreach` en écrivant :

```
| foreach(suite(13,50,5)as $indice=>$nb)
| {echo $indice, ":", $nb, " /// ";}
```

En complément de l'instruction `yield`, un générateur peut contenir une instruction `return`, comme n'importe quelle fonction. L'expression retournée ne sera pas lue avec la boucle `foreach` comme nous l'avons vu plus haut, mais avec la méthode `getReturn()` de l'objet générateur (de type `Generator`). Dans l'exemple suivant, nous reprenons la fonction `suite()` en lui ajoutant une instruction `return` (repère ②) qui va fournir le total de tous les nombres retournés (ceci n'est rien d'autre que la somme des termes de la suite arithmétique créée) – la variable `$total` étant auparavant initialisée à 0 (repère ①). Pour récupérer la valeur retournée, il faut créer un objet de type `Generator` dans la variable `$gener`, en appelant la fonction `suite()` (repère ③) puis lire cette valeur à l'aide de la méthode `getReturn()` de cet objet (repère ④).

## Exemple 7-11. Somme d'une suite numérique

```
<?php
function suite($min,$max,$pas)
{
    $total=0;←①
    for($i=$min;$i<=$max;$i+=$pas)
    {
        yield $i;
        $total+=$i;
    }
    return $total;←②
}
echo "Suite : ";
$gener= suite(0,45,5);←③
foreach($gener as $nb)
{echo $nb, " ";}
echo "<br/> Total = ",$gener->getReturn();←④
?>
```

Le résultat obtenu est le suivant :

---

```
Suite : 0 5 10 15 20 25 30 35 40 45
Total = 225
```

---

Comme il est possible d'utiliser plusieurs `yield` dans le même générateur, nous pourrions obtenir toutes les sommes partielles en remplaçant la ligne `$total+=$i;` par `yield $total+=$i;.`

## **Délégation d'un générateur**

Introduite dans PHP 7, la délégation de générateur permet d'utiliser le contenu d'un générateur dans un autre générateur, en l'occurrence de ses instructions `yield` en écrivant dans le premier générateur l'instruction `yield from` suivie du nom du second générateur

dont on veut récupérer les valeurs. Nous pouvons aussi multiplier les instructions `yield from` pour obtenir les valeurs issues de plusieurs autres générateurs. Dans l'exemple 7-12, nous créons trois générateurs (repères ①, ④ et ⑤) qui seraient utilisables individuellement dans une boucle `foreach`. Mais le premier utilise deux délégations des générateurs `adresse()` (repère ④) et `code()` (repère ⑤). Les valeurs "Paris", "France" et "75006" seront donc lues avec une boucle `foreach` (repère ⑥) opérée sur le générateur `nom()` sans y être présentes directement mais à partir des deux autres.

## Exemple 7-12. Délégation d'un générateur

```
<?php
function nom($prenom, $nom){←①

    yield $prenom;
    yield $nom;
    yield from adresse();←②
    yield from code();←③
}

function adresse(){←④
    yield "Paris";
    yield "France";
}

function code(){←⑤
{
    yield "75006";
}

foreach(nom("Jan", "Geelsen") as $coord){←⑥
{
    echo $coord, " ";
}
?>
```

Le résultat obtenu est le suivant :

---

Jan Geelsen Paris France 75006

---

## Portée des variables

Toutes les variables ont une portée déterminée selon le contexte.

### *Variables locales et globales*

Toutes les variables utilisées dans la déclaration d'une fonction sont, sauf indication contraire, locales au bloc de définition de la fonction. Cela permet d'utiliser n'importe quel nom de variable dans le corps de la fonction et comme valeur de retour, même si ce

nom est déjà utilisé dans le reste du script (il vaut toutefois éviter ces répétitions de noms de variables). Toutes les variables qui sont définies en dehors d'une fonction ou d'une classe sont globales et accessibles partout dans le script qui les a créées.

En conséquence, toute modification du paramètre d'une fonction opérée dans le corps de celle-ci n'a aucun effet sur une variable externe à la fonction et portant le même nom.

Dans l'exemple 7-13, la fonction `waytu()` utilise le paramètre `$interne`, qui est local à la fonction, et les deux variables globales `$interne` et `$externe`, initialisées en dehors de la fonction. Quelle que soit la variable utilisée lors des appels de la fonction, les deux variables globales `$externe` et `$interne` ne sont pas modifiées, comme le montrent les résultats affichés.

Le corps de la fonction comprend également la variable locale `$externe` — une exception à la règle énoncée précédemment. L'affectation d'une valeur à cette variable ne se répercute pas en dehors du corps de la fonction, la variable globale `$externe` (repère ①) gardant sa valeur initiale.

### Exemple 7-13. Variables locales et globales

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Variables locales et variables globales</title>
</head>
<body>
<div>
<?php
$externe="dehors";←①
$interne ="dedans";
function waytu($interne)
{
    $interne = "Si tu me cherches, je suis ".$interne ."  
";  

    $externe = "n'importe quoi !";  

    return $interne;  

}
echo waytu($interne); // Affiche "Si tu me cherches, je suis dedans"  

echo $interne," <br />"; // Affiche "dedans"  

echo waytu($externe); // Affiche "Si tu me cherches, je suis dehors"  

echo $externe," <br />"; // Affiche "dehors"  

?>
</div>
</body>
</html>
```

Le script affiche donc :

---

```
Si tu me cherches, je suis dedans  
dedans  
Si tu me cherches, je suis dehors  
dehors
```

---

Si vous souhaitez utiliser la valeur d'une variable globale dans le corps d'une fonction,

il vous faut déclarer cette variable dans le corps de la fonction en la faisant précédé du mot-clé `global` pour les versions de PHP antérieures à la 4.1 et, de manière plus élégante depuis, en utilisant le tableau associatif superglobal `$GLOBALS`. Les clés de ce dernier sont les noms des variables globales du script sans le signe `"$"`.

Pour utiliser la valeur de la variable globale `$mavar`, vous écrivez par exemple :

```
$GLOBALS["mavar"]
```

L'exemple 7-14 illustre l'emploi de variables globales dans une fonction pour rédiger un message. La fonction `message()` utilise le paramètre `$machin`, qui contient le nom d'une ville, et les valeurs des variables globales `$truc` et `$intro` déclarées de deux manières différentes.

#### **Exemple 7-14. Utilisation de variables globales dans une fonction**

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Fonctions utilisant des variables globales</title>
</head>
<body>
<div>
<?php
function message ($machin)
{
    global $struc;
    $machin = $GLOBALS['intro']." je suis $struc $machin <br />";
    $struc = "zzzzzzzzzzzzzzzzzzzz !";
    return $machin;
}
$intro= "Ne me cherches pas,";
$struc = " parti ";
echo message(" à Londres");
$intro= "Si tu me cherches,";
$struc=" revenu ";
echo message(" de Nantes");
echo $struc;
?>
</div>
</body>
</html>
```

Le script retourne le résultat suivant :

Chaque modification de la valeur des variables `$truc` et `$intro` est sensible dans la fonction `message()`. En particulier, la modification de la variable globale `$truc` dans le corps de la fonction est répercutée dans le reste du script, ce qui peut créer un danger si vous n'y prenez garde.

Par précaution, il est recommandé de n'utiliser les variables globales dans une fonction

que comme source de donnée et de ne pas modifier leur valeur. Si, comme il convient, vous prenez la bonne habitude de constituer des bibliothèques de fonctions externes, le code de celles-ci n'est plus visible directement. L'utilisation de variables globales risque d'opérer des modifications sans qu'elles soient visibles. La recherche d'erreurs peut alors devenir difficile.

## ***Les variables statiques***

Lors de l'appel d'une fonction, les variables locales utilisées dans le corps de la fonction ne conservent pas la valeur qui leur est affectée par la fonction. La variable redevient en quelque sorte vierge après chaque appel.

Pour conserver la valeur précédemment affectée entre deux appels d'une même fonction, il faut déclarer la variable comme statique en la faisant précéder du mot-clé `static`, et ce avant de l'utiliser dans le corps de la fonction. Le deuxième appel de la fonction peut réutiliser la valeur qu'avait la variable après le premier appel de la fonction, et ainsi de suite à chaque nouvel appel.

L'utilisation typique des variables statiques concerne les fonctions qui effectuent des opérations de cumul. Une variable déclarée comme `static` ne conserve toutefois une valeur que pendant la durée du script. Lors d'une nouvelle exécution de la page, elle reprend sa valeur initiale. Il ne faut donc pas compter sur cette méthode pour transmettre des valeurs d'une page à une autre, même si ces dernières appellent la même fonction contenant des variables statiques.

Dans l'exemple suivant :

```
function acquis($capital,$taux)
{
    static $acquis;
    //corps de la fonction
}
```

la variable `$acquis` n'a pas de valeur initiale et doit être affectée dans la suite du code. Par contre, vous pouvez lui donner une valeur initiale lors de sa déclaration en écrivant :

```
function acquis($capital,$taux)
{
    static $acquis=1;
    //corps de la fonction
}
```

L'exemple 7-15 crée une fonction `acquis()` qui affiche successivement les valeurs acquises d'un placement réalisé à un taux fixe. L'utilisateur choisit le capital et le taux dans deux champs texte d'un formulaire de saisie, nommés `capital` et `taux`.

Vous utilisez dans le corps de la fonction deux variables statiques, `$acquis` et `$an`, qui représentent le coefficient multiplicateur du prêt et la durée du placement. Entre chaque appel de la fonction, ces variables conservent leur valeur pour permettre d'afficher la

valeur acquise année après année à l'aide d'une boucle.

La fonction retourne la valeur acquise totale à chaque appel et crée une boîte d'alerte en JavaScript contenant la même valeur du capital acquis.

La partie HTML du fichier consiste uniquement en la création du formulaire de saisie des informations nécessaires au calcul du placement.

## → Exemple 7-15. Utilisation d'une variable statique

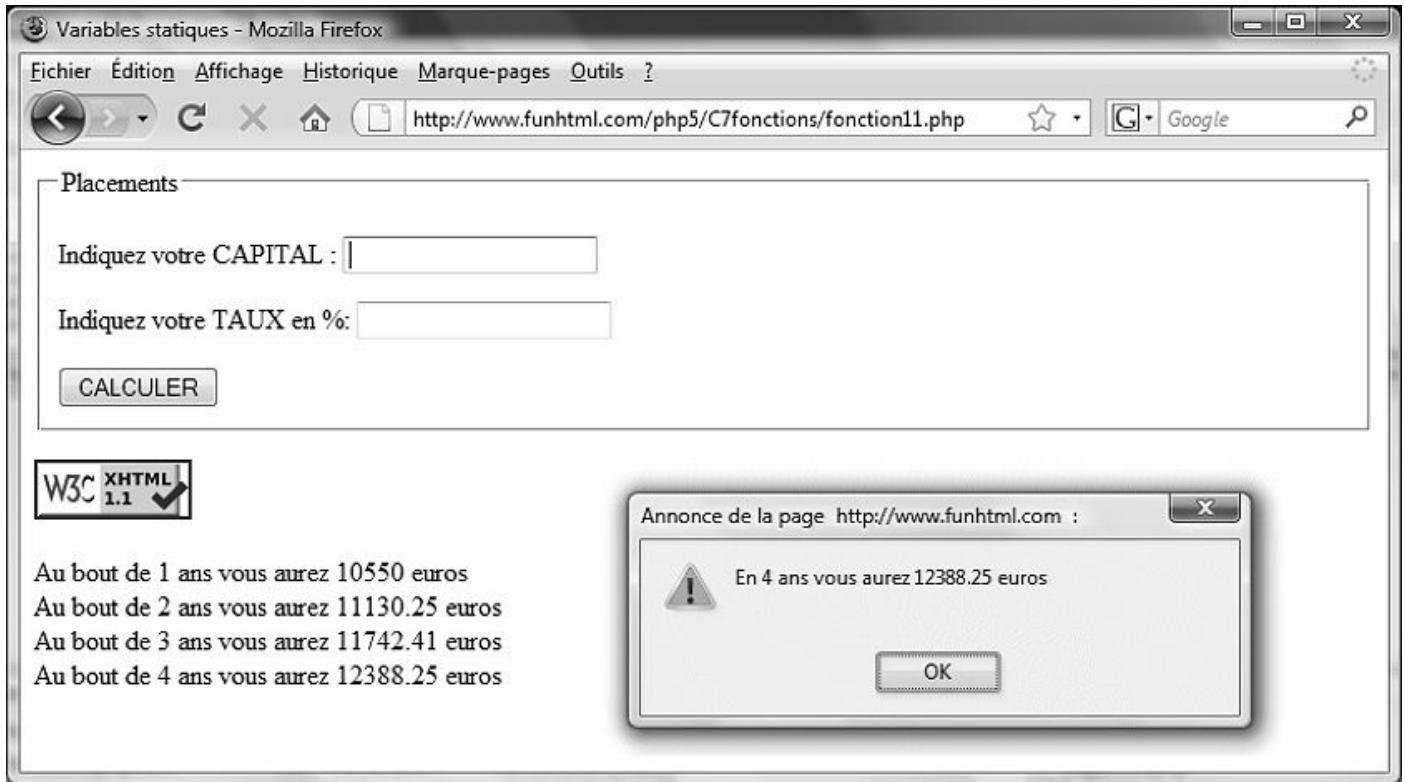
```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Variables statiques</title>
</head>
<body>
<form method="post" action="exemple7.11.php" >
<fieldset>
<legend>Placements</legend>
<p>Indiquez votre CAPITAL :<br/>
<input type="number" name="capital" step="1000"/></p>
<p>Indiquez votre TAUX en % :<br/>
<input type="number" name="taux" step="0.1"/></p>
<input type="submit" name="calcul" value="CALCULER"/><br />
</fieldset>
</form>
</body>
</html>

<?php
// Définition de la fonction
function acquis($capital,$taux)
{
    static $acquis=1;
    static $an=1;
    $coeff = 1+$taux/100;
    $acquis *= $coeff;
    echo "<script type=\"text/javascript\"> alert('En $an ans vous aurez " .
        round($capital*$acquis,2) ." euros') </script>";
    $an++;
    return round($capital*$acquis,2);
}

// Utilisation de la fonction
if(isset($_POST["taux"])&& isset($_POST["capital"])&& is_numeric($_POST["capital"])
    && is_numeric($_POST["taux"]))
{
    for($i=1;$i<5;$i++)
    {
        echo "Au bout de $i ans vous aurez ". acquis($_POST["capital"],$_POST["taux"])
            . " euros <br />";
    }
}
?>
```

La figure 7-4 donne un aperçu du formulaire de saisie ainsi que de l'affichage des

résultats dans la page.



**Figure 7-4**  
*Utilisation des variables statiques dans un formulaire de saisie*

## Passer des arguments par référence

Dans les définitions des fonctions que vous venez de créer, les arguments sont passés par valeur. C'est donc une copie de ces variables qui est utilisée par la fonction. En aucun cas les modifications de la valeur des paramètres ne sont visibles à l'extérieur de la fonction.

Comme vous l'avez vu au chapitre 2 pour l'affectation des variables par référence, il est possible de passer à une fonction un argument par référence. Dans ce cas, les modifications effectuées dans le corps de la fonction sont répercutées à l'extérieur.

Vous avez le choix entre deux possibilités, passer des arguments par référence de façon systématique (voir l'exemple 7-16) et passer des arguments par référence occasionnellement (voir l'exemple 7-17).

Si vous voulez que le passage des arguments par référence soit fait systématiquement, il vous faut faire précéder les paramètres que vous désirez passer par référence du signe & dans la définition de la fonction elle-même.

Vous utilisez pour cela la syntaxe suivante :

```
function nom_fonction(&$param1, &$param2, ....)
{
//corps de la fonction
}
```

Il n'y a pas obligation de passer tous les paramètres par référence dans la même fonction, et vous pouvez n'en passer qu'une partie. Dans l'exemple 7-16, la fonction `prod()` reçoit comme paramètre un tableau passé par référence et un coefficient passé par valeur. La fonction vérifie d'abord que chaque élément du tableau est numérique. Si tel est le cas, il est multiplié par le coefficient passé en second paramètre.

La fonction retourne ensuite un tableau contenant ces nouvelles valeurs, ou, si un seul des éléments n'est pas numérique, retourne `FALSE`.

L'ensemble des résultats du script montre que le tableau passé en paramètre a été modifié et qu'il est le même que celui retourné par la fonction (la variable `$result`). En pratique, il serait inutile de retourner une valeur dans le corps de la fonction. Remarquez bien en revanche que toutes les valeurs initiales sont perdues.

## Exemple 7-16. Passage d'arguments par référence

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Passage par référence</title>
</head>
<body>
<div>

<?php
// Définition de la fonction
function prod(&$tab,$coeff)
{
    foreach($tab as $cle=>$val)
    {
        if(is_numeric($val))
            {$tab[$cle]*=$coeff;}
        else
        {
            echo "Erreur : Le tableau est non numérique <br />";
            return FALSE;
        }
    }
    return $tab;
}
echo "Tableau numérique <br />";
$tabnum = range(1,7);
echo "Tableau avant l'appel <br />",print_r( $tabnum ), "<br />";
// Passage du tableau à la fonction
$result= prod($tabnum,3.5);
echo "Tableau résultat <br />",print_r( $result ), "<br />";
echo "Tableau initial après l'appel <br />",print_r( $tabnum ), "<br />";
echo "Tableau alphabétique <br />";
$stabalpha= range("A","F");
$resultal=prod($stabalpha,3); // Retourne FALSE
echo "Tableau après l'appel <br />",print_r( $stabalpha ), "<br />";
?>
</div>
</body>
</html>
```

Le script retourne le résultat suivant :

---

```
Tableau numérique
Tableau avant l'appel
Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 [5] => 6 [6] => 7 ) 1
Tableau résultat
Array ( [0] => 3.5 [1] => 7 [2] => 10.5 [3] => 14 [4] => 17.5 [5] => 21 [6] => 24.5 ) 1
Tableau initial après l'appel
Array ( [0] => 3.5 [1] => 7 [2] => 10.5 [3] => 14 [4] => 17.5 [5] => 21 [6] => 24.5 ) 1
Tableau alphabétique
Erreur : Le tableau est non numérique
Tableau après l'appel
Array ( [0] => A [1] => B [2] => C [3] => D [4] => E [5] => F ) 1
```

---

## 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. 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 donc 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 ");
```

permet d'appeler la fonction `date()` avec comme paramètre la chaîne "D d/M/Y H:i:s".

Il est envisageable de créer un tableau de chaînes contenant des noms de fonctions et d'appeler celles-ci en écrivant le nom de l'élément du tableau suivi de parenthèses et de

paramètres s'il en existe.

Le code suivant :

```
<?php  
$tabfonc= array("phpinfo", "date");  
$tabfonc[0]();  
echo $tabfonc[1]("D d m Y H:i:s");  
?>
```

serait ainsi l'équivalent des précédents.

Pour illustrer les possibilités des fonctions dynamiques, l'exemple 7-17 crée un formulaire de saisie permettant à l'utilisateur d'entrer un nom de fonction native de PHP et la valeur d'un paramètre. L'envoi du formulaire provoque l'affichage de la valeur désirée.

Le formulaire contient deux zones de saisie de texte. La première, nommée "fonction", permet la saisie du nom de la fonction, et la seconde, nommée "param", de la valeur du paramètre.

Le code contenu dans l'attribut `value` des éléments `<input />` (repères ①) permet de conserver l'état du formulaire avant son envoi et donc de réafficher les données saisies par l'utilisateur.

Le code PHP de gestion du formulaire vérifie d'abord l'existence d'une saisie de nom de fonction et d'un paramètre (repère ②) puis, s'ils existent, vérifie que la fonction choisie existe en PHP (ou dans le script lui-même repère ③) puis, selon les cas, affiche la valeur désirée (repère ④) ou un message d'erreur (repère ⑤).

## Exemple 7-17. Fonctions dynamiques

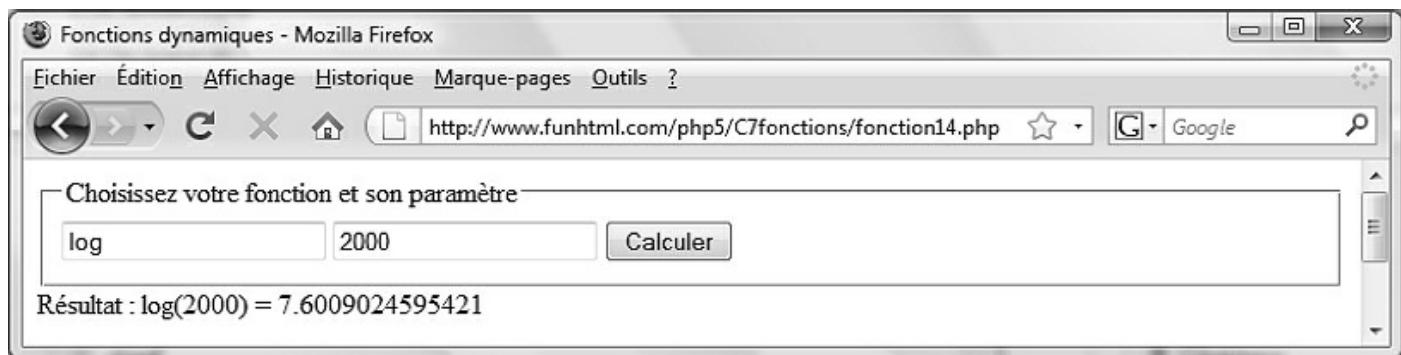
```
<!DOCTYPE html>  
<html lang="fr">  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
<title>Fonctions dynamiques</title>  
</head>  
<body>  
<form method="post" action="exemple7.13.php" >  
<fieldset>  
<legend>Choisissez votre fonction et son paramètre</legend>  
<input type="text" name="fonction" value="<?= isset($_POST["fonction"]) ?  
$_POST["fonction"] : "" ?>" />← ①  
<input type="text" name="param" value="<?= isset($_POST["param"]) ?  
$_POST["param"] : "" ?>" />← ①  
<input type="submit" value="Calculer"/>  
</fieldset>  
</form>  
<!-- Code PHP : gestion du formulaire-->  
<?php  
if((isset($_POST["fonction"])) && $_POST["fonction"] != "") && $_POST["param"] != "")← ②  
{  
    $fonction = $_POST["fonction"];  
    $param = $_POST["param"];  
}
```

```

if(function_exists($fonction)) ← ③
{
    echo "Résultat : $fonction($param) = ",$fonction($param); ← ④
}
else echo "ERREUR DE FONCTION!"; ← ⑤
}
?>
</body>
</html>

```

La figure 7-5 montre le résultat obtenu en choisissant la fonction log.



**Figure 7-5**  
*Utilisation de fonctions dynamiques*

## ***Les fonctions conditionnelles***

Une fonction est dite conditionnelle si elle est définie à l'intérieur d'un bloc `if`. Sa création se réalise alors de la même façon que pour une fonction ordinaire, mais elle n'est utilisable que si l'instruction `if` qui la contient a été exécutée et, bien sûr, si l'expression conditionnelle contenue dans `if` a la valeur booléenne `TRUE`. Ces conditions étant remplies, la fonction peut être appelée normalement dans tout le code qui suit la condition `if`.

L'exemple 7-18 illustre cette possibilité. L'appel de la fonction ordinaire `test()` (repère ①) montre que l'on peut l'appeler alors même qu'elle n'est définie qu'en fin de script (repère ⑦), ce que nous avons déjà vu. Par contre, l'appel de la fonction `salut()` en début de script (repère ②) provoquerait une erreur fatale (message: `Fatal error: Call to undefined function salut()`) et donc l'arrêt immédiat du script. Cette fonction est en effet conditionnelle et n'est accessible que lorsque le script atteint la ligne du `if` (repère ④) qui vérifie si l'heure obtenue par la fonction `date()` (repère ③ : voir le chapitre 8 pour plus de détails sur son fonctionnement) est inférieure à 18. Dans ce cas la fonction `salut()` est définie (repère ⑤). Notre exemple pourrait s'arrêter là mais, dans le but d'éviter l'erreur fatale signalée ci-dessus, nous utilisons un bloc `else` (repère ⑥) qui crée une autre version de la fonction `salut()` quand la condition `if` n'est pas vérifiée. L'appel de cette fonction est alors possible en fin de script (repère ⑧).

### **Exemple 7-18. Fonctions conditionnelles**

```

<?php
test();←①
//salut(); // Cet appel provoquerait une erreur fatale←②
$heure=date("H");←③
// Définition d'une fonction conditionnelle
if($heure<18)←④
{
    function salut()←⑤
    {
        echo "Bonjour : Fonction accessible seulement avant 18h00 <br />";
    }
}
else←⑥
{
    function salut()
    {
        echo "Bonsoir : Fonction accessible seulement après 18h00 <br />";
    }
}
// Définition d'une fonction ordinaire
function test()←⑦
{
    echo "Fonction accessible partout <br />";
    return TRUE;
}
salut();←⑧
?>

```

Testé après 18h00, le script affichera :

---

```
Fonction accessible partout
Bonsoir : Fonction accessible seulement après 18h00
```

---

## ***Fonction définie dans une autre fonction***

Dans le cas d'une fonction définie à l'intérieur d'une autre fonction, le code de création de la fonction n'est plus dans un bloc if mais dans le bloc qui constitue le corps d'une autre fonction. La fonction incluse n'est alors utilisable que si celle qui la contient a été appelée une fois, sinon PHP lève une erreur fatale. L'inconvénient de cette méthode est que la fonction « conteneur » ne doit être appelée qu'une seule et unique fois, sinon nous obtenons à nouveau une erreur fatale pour cause de redéfinition de la fonction incluse ce qui, avouons-le, rend cette fonctionnalité un peu dangereuse. L'exemple 7-19 illustre cette possibilité de création de fonction. Le script contient la définition de la fonction `parent()` (repère ②) qui contient elle-même la définition de la fonction `enfant()` (repère ③). L'appel de cette dernière en début de script provoquerait une erreur fatale (repère ①) mais, lorsque que la fonction `parent()` a été appelée une fois (repère ④), la fonction `enfant()` peut être appelée autant de fois que l'on veut (repères ⑤ et ⑥).

### **Exemple 7-19. Fonction incluse dans une autre**

```

<?php
//enfant() ; //ERREUR FATALE ← 1
function parent() ← 2
{
    echo "Bonjour les enfants ! <br />";
    function enfant() ← 3
    {
        echo "Bonsoir papa !<br />";
    }
}
parent(); ← 4
enfant(); ← 5
enfant(); ← 6
?>

```

Le script affiche les résultats suivants :

---

```

Bonjour les enfants !
Bonsoir papa !
Bonsoir papa !

```

---

## *Les fonctions 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  $n!$ .

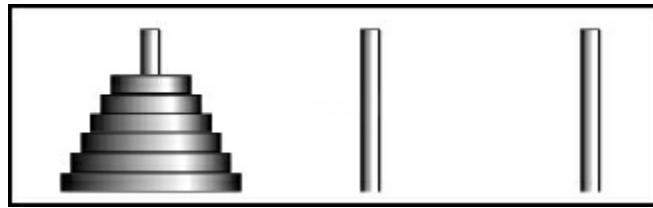
Ce qui donne, par exemple, le code suivant :

```

<?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 piquets en ordre décroissant. La figure 7-6 présente une configuration de départ avec six disques.



**Figure 7-6**  
*Les tours de Hanoï*

Pour programmer le jeu, les piquets seront numérotés de 1 à 3 de gauche à droite, et on remarquera que le piquet intermédiaire a le numéro 6 – « départ » – « arrivée ». Quand le nombre de disques est faible (au minimum 3) il est assez aisé de gagner mais, lorsqu'il augmente, ceci demande un peu de réflexion car il faut parvenir au résultat en un minimum de coups. Pour une quantité N de disques, ce nombre de coups minimum est toujours égal à  $2^N - 1$ . Comme pour la factorielle, l'idée générale pour effectuer une action pour N disques est de la réaliser d'abord pour N - 1 disques, puis pour N - 2 et ainsi de suite jusqu'à un seul disque ce qui constitue une action élémentaire facile. On remonte alors pas à pas jusqu'à N. Pour déplacer, par exemple, trois disques du piquet 1 vers le piquet 2 on effectue les opérations suivantes :

---

Déplacez un disque du piquet 1 vers le piquet 2  
 Déplacez un disque du piquet 1 vers le piquet 3  
 Déplacez un disque du piquet 2 vers le piquet 3  
 Déplacez un disque du piquet 1 vers le piquet 2  
 Déplacez un disque du piquet 3 vers le piquet 1  
 Déplacez un disque du piquet 3 vers le piquet 2  
 Déplacez un disque du piquet 1 vers le piquet 2

---

L'exemple 7-20 donne le code de résolution du jeu. S'il reste un disque à déplacer nous avons l'action élémentaire du départ vers l'arrivée (repère ①) sinon il faut déplacer N - 1 disques du départ vers l'intermédiaire (repère ②) puis N-1 disques de l'intermédiaire vers l'arrivée (repère ③).

### Exemple 7-20. Les tours de Hanoï

```
<?php
function hanoi($nb,$dep,$arr)
{
    if ($nb==1) echo "Déplacez un disque du piquet $dep vers le piquet
$arr <br />"; ←①
    else
    {
        $inter=6-$dep-$arr;
        hanoi($nb-1,$dep,$inter); ←②
        echo "Déplacez un disque du piquet $dep vers le piquet $arr <br />";
        hanoi($nb-1,$inter,$arr); ←③
    }
}
hanoi(4,1,2);
?>
```

L'affichage des opérations réalisées pour déplacer quatre disques du piquet 1 vers le piquet 2 est le suivant :

---

```
1 Déplacez un disque du piquet 1 vers le piquet 3
2 Déplacez un disque du piquet 1 vers le piquet 2
3 Déplacez un disque du piquet 3 vers le piquet 2
4 Déplacez un disque du piquet 1 vers le piquet 3
5 Déplacez un disque du piquet 2 vers le piquet 1
6 Déplacez un disque du piquet 2 vers le piquet 3
7 Déplacez un disque du piquet 1 vers le piquet 3
8 Déplacez un disque du piquet 1 vers le piquet 2
9 Déplacez un disque du piquet 3 vers le piquet 2
10 Déplacez un disque du piquet 3 vers le piquet 1
11 Déplacez un disque du piquet 2 vers le piquet 1
12 Déplacez un disque du piquet 3 vers le piquet 2
13 Déplacez un disque du piquet 1 vers le piquet 3
14 Déplacez un disque du piquet 1 vers le piquet 2
15 Déplacez un disque du piquet 3 vers le piquet 2
```

---

Il comporte bien 15 déplacements (soit  $2^4 - 1$ ) et on peut remarquer que les lignes paires du déplacement hanoi(4,1,2) correspondent au déplacement hanoi(3,1,2).

## Les fonctions anonymes (*closure*)

### Définition générale

Les fonctions anonymes, dites aussi *closures*, ont la particularité de pouvoir être créées en dérogeant aux règles vues au début de ce chapitre, car on ne leur donne pas de nom mais on assigne leur définition à une variable convertie automatiquement en objet de type `Closure`. L'appel de la fonction se fait à l'aide du nom de la variable suivi des parenthèses et des paramètres éventuels de deux manières différentes – selon que la fonction retourne ou affiche un résultat comme nous le faisons pour une fonction classique. C'est ce qu'illustre l'exemple 7-21 dans lequel nous définissons deux fonctions anonymes : la première qui affiche le résultat d'une somme directement (repères ① et ②) et la seconde qui retourne la valeur d'un quotient et est utilisable avec l'instruction `echo` (repères ③ et ④)

### Exemple 7-21. Fonctions anonymes

```
<?php
//Création
$varadd=function($a,$b){ echo "Somme = ",$a+$b; };  
①
//Appel
$varadd(22,7); //Somme = 29  
②
echo "<br/>";
//Création
$vardiv = function($a,$b){ return $a/$b; };  
③
//Appel
echo "Quotient = ",$vardiv(22,7); //3.1428571428571  
④
?>
```

## Héritage d'une variable

Dans sa définition, une fonction anonyme peut aussi hériter d'une variable créée avant sa définition en utilisant le mot-clé `use` suivi du nom de la variable entre parenthèses selon le modèle suivant :

```
| $text="Résultat = ";
| $varadd=function($a,$b) use ($text) {echo $text,$a+$b;}
```

En écrivant le code `$varadd(22,7);`, nous obtenons l'affichage "Résultat = 29". Notez bien que si la définition de la variable `$text` est faite après celle de la fonction, la variable n'est pas héritée et nous n'obtenons que la valeur 29.

La modification de la valeur de la variable héritée – si elle est faite après la définition de la fonction –, ne sera pas prise en compte. Pour qu'elle le soit, comme dans l'exemple suivant, il faut définir dans la fonction l'héritage de la variable par référence (repère ②) où la variable a d'abord la valeur "Résultat" (repères ① et ③). Le premier appel de la fonction affiche donc cette valeur (repère ③). Par la suite, nous pouvons modifier plusieurs fois cette valeur qui est bien prise en compte à chaque fois (repères ④, ⑤, ⑥ et ⑦).

```
<?php
//héritage de variable
$text="Résultat = ";←①
$varadd=function ($a,$b) use(&$text) {echo $text, $a+$b;};←②
$varadd(22,7); // Affiche: Résultat = 29←③
$text="Addition = ";←④
echo "<br/>";
$varadd(22,7); // Affiche: Addition = 29←⑤
$text="Total = ";←⑥
echo "<br/>";
$varadd(22,7); // Affiche: Total = 29←⑦
?>
```

## Accès aux propriétés d'un objet

### Objets

Pour vous familiariser avec la notion d'objet, vous pouvez vous reporter avant d'aller plus loin à l'introduction du chapitre 9.

Dans la définition d'une fonction anonyme, nous pouvons aussi accéder aux propriétés d'un objet à l'aide de la pseudo-variable `$this` selon les syntaxes suivantes.

Sans paramètre :

```
| $var=function() {return $this->prop;};
```

Avec un paramètre :

```
| $var=function($a) {return $this->prop+ $a;};
```

dans lesquelles "prop" est le nom de la propriété d'un objet dont on veut récupérer la valeur dans la fonction pour l'ajouter dans le second cas au paramètre \$a.

Dans l'exemple 7-22, nous définissons tout d'abord une classe élémentaire (repère ①) qui a deux propriétés (repères ② et ③), puis nous créons la fonction anonyme \$varadd (repère ④). Rappelons à ce stade que \$varadd est un objet de type `Closure`, et qu'en tant que tel il a des méthodes. C'est la méthode `call()` qui nous permet d'utiliser la fonction en lui faisant récupérer la propriété `valeur1` de la classe. Le premier paramètre de cette méthode doit être un objet, ici de type `Nombre`. Un premier moyen est de créer un objet `Nombre` avec la syntaxe "`new Nombre`" comme premier paramètre, suivi de la valeur du paramètre \$a (repère ⑤). L'expression `$this->valeur1` a alors la valeur 22 et la somme est affichée. L'autre possibilité consiste à créer un objet du type `Nombre` dans la variable \$nb (repère ⑥) avant l'appel de la fonction et de le passer en premier paramètre (repère ⑦) – la fonction récupérant la valeur de sa propriété `valeur1` comme auparavant. Si en cours de script nous modifions la valeur de la propriété récupérée (repère ⑧), un nouvel appel de la fonction tiendra compte de cette modification (repère ⑨).

## Exemple 7-22. Accès aux propriétés d'un objet

```
<?php
class Nombre {←①
    public $valeur1=22;←②
    public $valeur2=75;←③
}

$varadd = function ($a) { return $this->valeur1 + $a; };←④
//Appel en créant un objet
echo "Somme = ",$varadd->call(new Nombre,7); //Affiche 29←⑤
echo "<br/>";
//Appel avec un objet déjà créé
$nb = new Nombre();←⑥
echo "Somme = ",$varadd->call($nb,7); //Affiche 29←⑦
echo "<br/>";
$nb->valeur1=55;←⑧
echo "Somme = ",$varadd->call($nb,7); //Affiche 62←⑨
?>
```

Les résultats affichés sont évidents :

---

```
Somme = 29
Somme = 29
Somme = 62
```

---

## Exercices

### Exercice 1

Créez une fonction PHP qui affiche une boîte d'alerte à partir de la fonction JavaScript

dont la syntaxe est `alert("chaine_de caractères")`. Cette fonction peut être appelée avec comme paramètre le texte du message à afficher. Elle est particulièrement utile pour afficher des messages d'erreur de manière élégante, sans que ces derniers restent écrits dans la page.

## Exercice 2

Écrivez une fonction de lecture de tableaux multidimensionnels en vous inspirant de l'exemple 7-3. L'affichage se fait sous forme de tableau HTML dont les titres sont les clés des tableaux.

## Exercice 3

Écrivez une fonction qui retourne la somme de la série de terme général  $u_n = x^{2n+1}/n!$ . Les paramètres de la fonction sont  $n$  pour le nombre d'itérations et  $d$  pour le nombre de 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 HTML en cause. Chaque appel de la fonction doit incorporer le code HTML 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'en-tête 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  $6=3+3$ ).

### Exercice 9

Créez une fonction anonyme qui retourne la division entière de deux nombres.

## Rappels sur les SGBDR

Une base de données est un ensemble d'informations stockées 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 uniifié 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.
- Les mises à jour ou la suppression de données doivent laisser la base intègre et ne pas créer d'incohérences.
- La recherche d'informations doit être rapide et sûre.

Vous allez donc aborder successivement les phases suivantes :

1. Élaboration du modèle d'organisation des données à l'aide de la méthode entité/association. Cela entraîne la création d'un modèle conceptuel de données

(MCD), qui est une représentation abstraite des données à stocker et des liens entre elles.

2. Passage du modèle ainsi créé au modèle relationnel, qui est actuellement le plus courant. Cela entraîne la création d'un modèle logique de données (MLD), qui est la représentation d'un modèle implantable dans un système particulier.
3. Implémentation dans un SGBDR (Système de gestion de base de données relationnelle) particulier, comme MySQL ou SQLite, qui font l'objet des chapitres suivants.

L'exemple qui servira de socle à tout ce chapitre est la modélisation de la base de données nécessaire à la gestion d'un site de commerce en ligne.

## Le modèle entité/association

Le nom anglais du modèle entité/association (*Entity/Relationship*) est à l'origine de la confusion que l'on retrouve souvent dans la définition d'une base de données relationnelle (BDR). Certains auteurs définissent un BDR comme une base ayant des relations entre tables. Or une base est dite relationnelle si elle repose sur la notion de table, qui est la traduction du concept mathématique de relation entre ensembles. Une base de données peut en effet être qualifiée de relationnelle et ne comporter qu'une seule table.

Le modèle entité/association permet la modélisation abstraite d'une base de données. Il utilise un ensemble de conventions de représentation graphique pour modéliser les différents concepts contenus dans une information et les liens qui existent entre eux. Les schémas réalisés sont similaires à ceux de la méthode Merise et donc différents de la notation UML.

### *Les entités*

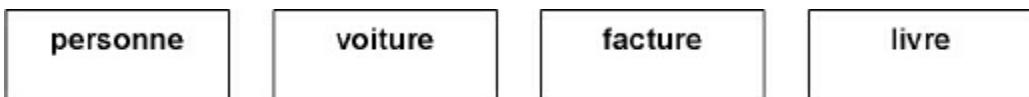
On appelle entité une représentation d'un ensemble d'objets réels ou abstraits qui ont des caractéristiques communes. Une information du monde réel peut correspondre à plusieurs entités. Cette décomposition de l'information en plusieurs entités, dont chacune a une nature différente, constitue la première étape du travail de conception. Si vous voulez modéliser une commande faite par un client, il apparaît en première lecture au moins deux entités, une personne d'un côté et les articles qu'elle commande de l'autre. Vous avez bien fait apparaître deux entités de nature différente. Chaque personne réelle est une occurrence de l'entité générale `personne`.

On distingue deux sortes d'entités :

- Les entités fortes, qui ne dépendent pas de l'existence d'une autre entité. C'est le cas, par exemple, d'une entité représentant une personne ou un produit.
- Les entités faibles, dont l'existence dépend d'une autre entité. C'est le cas d'une

entité représentant une commande qui dépend de l'entité *personne* (pas de commande s'il n'y a pas de client).

Les entités sont représentées graphiquement par des rectangles (voir figure 13-1).



**Figure 13-1**  
*Représentation des entités*

## ***Les attributs***

Chaque entité a des caractéristiques particulières, que l'on retrouve dans toutes ses occurrences. Un client a, par exemple, nécessairement un nom, un prénom, une adresse, etc. Ces caractéristiques sont nommées attributs de l'entité.

Chaque entité doit avoir au moins un attribut, qui permet de distinguer une occurrence d'une autre. Cet attribut particulier est la clé primaire de l'entité. Cette clé doit être unique dans l'entité. Pour une personne, il est évidemment possible de trouver deux personnes de même nom. Le nom est donc un mauvais candidat pour effectuer cette distinction.

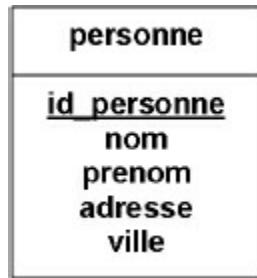
Pour distinguer deux clients, on utilise habituellement un numéro de client pour chaque personne. La clé primaire peut être constituée de la réunion de plusieurs attributs, par exemple, le nom et l'adresse, si nous admettons qu'il n'est pas possible que deux personnes homonymes habitent au même endroit. L'utilisation d'une clé primaire abstraite, comme un identifiant numérique (par exemple le numéro INSEE propre à chaque personne), représente une solution plus sûre que le choix d'un autre attribut, et il est conseillé de l'utiliser systématiquement.

Un attribut peut être défini comme obligatoire ou facultatif dans l'entité. Il peut être élémentaire ou décomposable en plusieurs autres attributs. L'adresse complète d'une personne peut constituer un seul attribut ou être décomposée en rue, ville, département et code postal. Il n'y a pas d'obligation en ce domaine. Cela relève d'un choix du programmeur en fonction des besoins du client. S'il n'envisage pas de réaliser un jour des recherches de personnes par ville ou par département, il est inutile de décomposer l'attribut *adresse*.

Chaque attribut possède un domaine de valeurs possibles. Un nom est représenté par une chaîne de caractères de longueur variable et le code postal par un nombre de cinq chiffres. À chaque attribut correspond un type de donnée particulier. C'est à prendre en compte lors de l'implantation de la base de données sur un SGBDR particulier.

Une entité munie de ses attributs est représentée par un rectangle contenant le nom de l'entité suivi de celui de ses attributs. Dans la représentation graphique de l'entité, le nom de l'attribut qui constitue la clé primaire est placé en premier et souligné (voir

figure 13-2).

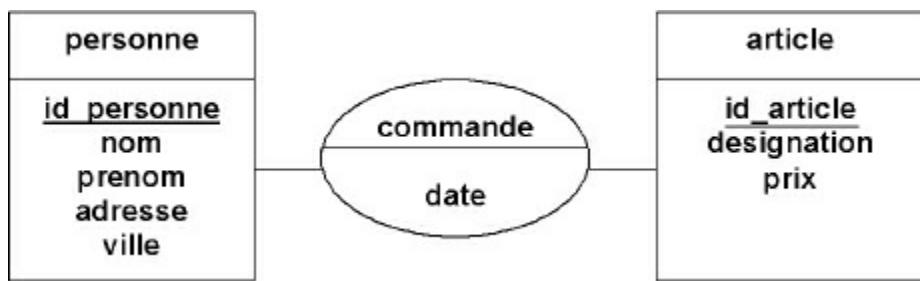


**Figure 13-2**  
*Représentation graphique d'une entité*

## **Les associations**

Le concept d'association permet de représenter le lien existant entre deux ou plusieurs entités. Une association reliant deux entités est dite binaire. Celle qui en relie plusieurs est dite *n*-aire. Dans la décomposition de l'information en entités vous avez toujours intérêt à créer des associations binaires et à redécomposer celles qui seraient ternaires.

Une association est généralement nommée à l'aide d'un verbe d'action (à la forme active d'une entité vers l'autre et passive dans l'autre sens). Par exemple, la phrase « Un client commande un article » résume l'association *commande* entre l'entité *client* et vers l'entité *article*. Une association est représentée graphiquement par une ellipse contenant son nom. La figure 13-3 donne la représentation graphique de cette association. Une association peut, comme les entités, posséder des attributs. La clé de l'association est la concaténation des clés des entités qu'elle relie. Une association peut être réflexive, c'est-à-dire relier une entité à elle-même. Par exemple, l'association *conjoint* relie une personne avec une autre personne.



**Figure 13-3**  
*Représentation d'une association binaire*

## **Les cardinalités**

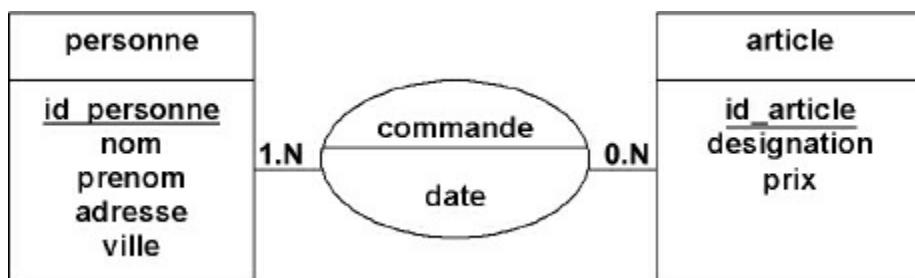
La cardinalité d'une association mesure le nombre d'occurrences d'une entité qu'il est possible d'associer à une occurrence de l'autre entité associée. Elles sont indiquées de chaque côté de l'association pour chaque entité.

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 côté de l'entité `personne` est donc 1. Un produit peut n'être commandé par aucune personne. La cardinalité minimale du côté 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 côté de l'entité `personne` est donc notée N. Un produit peut être commandé par N personnes différentes. La cardinalité maximale du côté 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.

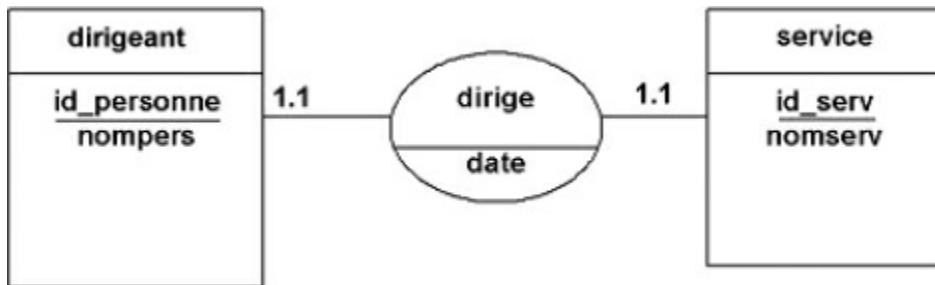
Après étude des cardinalités, on distingue plusieurs cas, qui vont constituer l'organisation des tables dans le modèle relationnel.

## Détermination des cardinalités

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

## Exemple 1

Un ministère comprend plusieurs services. Chaque service est dirigé par une seule personne. Pour tenir compte des changements de directeur, l'association `dirige` doit avoir un attribut `date` qui contienne la date de nomination. La figure 13-5 donne une représentation de cette association.



**Figure 13-5**

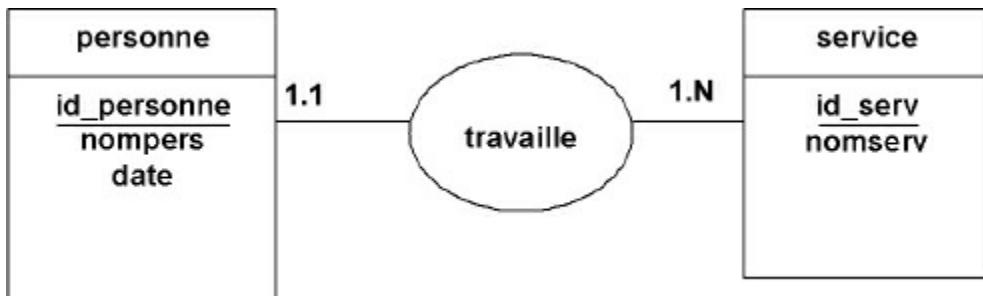
*Association 1:1*

Une et une seule personne dirige un service, ce qui correspond à une cardinalité 1.1 pour l'entité `personne`. Chaque service est dirigé par une et une seule personne, ce qui se traduit par une cardinalité 1.1 pour l'entité `service`.

## Exemple 2

Reprendons le même ministère qu'à l'exemple 1 mais en représentant l'association entre tous les employés du ministère, quel que soit leur grade ou le service dans lequel ils travaillent. Une personne travaille dans un et un seul service. La cardinalité du côté de l'entité `personne` est donc 1.1. Un service emploie de une à plusieurs personnes. La cardinalité du côté de l'entité `service` est donc 1.N.

La figure 13-6 donne une représentation de cette association.



**Figure 13-6**

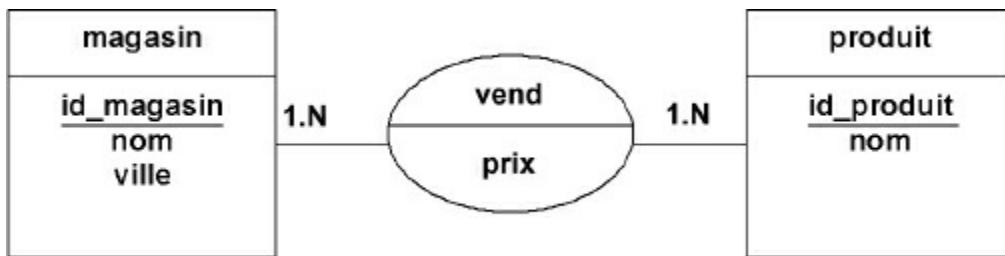
*Association 1:N*

## Exemple 3

Pour suivre l'évolution des prix, on recense un nombre de produits vendus dans un ensemble de magasins. Chaque magasin peut vendre un ou plusieurs produits. La cardinalité du côté de l'entité `magasin` est donc 1.N. Chaque produit peut être vendu par un ou plusieurs magasins. La cardinalité du côté de l'entité `produit` est donc 1.N. Chaque

magasin vend le produit à un prix donné. L'association doit donc posséder un attribut prix.

La figure 13-7 donne une représentation de cette association.



**Figure 13-7**  
*Association N.M*

## ***Conception du MCD***

Pour établir le modèle conceptuel de données (MCD) correspondant à l'ensemble d'informations à stocker, il faut procéder de la façon suivante :

1. Décomposer l'information globale en entités indépendantes représentant des concepts différents.
2. Pour chaque entité, faire l'inventaire des attributs qui permettent de la décrire. Le degré de précision de cette description dépend des besoins du client et de l'utilisation qui sera faite de la base de données. Chaque attribut doit être utile. Il faut, par exemple, envisager quels seront les critères de recherche utilisés sur la base. Chaque attribut doit pouvoir être exprimé clairement par une chaîne de caractères, un nombre ou une date, par exemple.
3. Pour chaque entité, choisir l'attribut ou le groupe d'attributs qui peut constituer une clé primaire, en évitant les clés primaires composites dans la mesure du possible. L'utilisation d'une clé primaire numérique est souvent la meilleure solution.
4. Définir les associations qui relient les différentes entités. Il faut privilégier les associations binaires, quitte à créer des entités supplémentaires. La gestion et l'interrogation de la base s'en trouvent facilitées.
5. Déterminer les attributs de l'association en ne retenant que les attributs vraiment nécessaires caractérisant l'association. Si un attribut n'est pas nécessaire, mieux vaut le placer dans une entité.
6. Exprimer les cardinalités de l'association pour chaque entité qui y est reliée.

## ***Normalisation du MCD***

Le MCD élaboré à l'étape précédente doit être vérifié et normalisé au moyen de méthodes particulières.

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 :

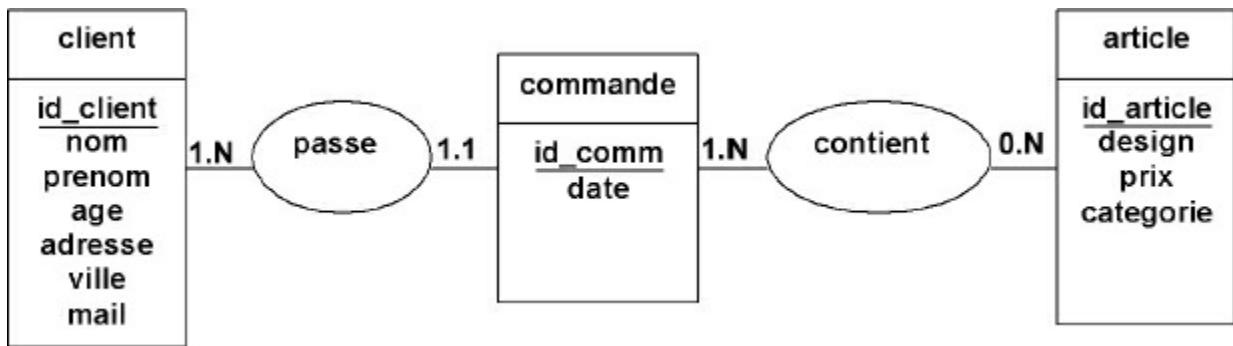
1. Un client enregistré dans la base a passé au moins une commande, sinon il ne figure pas dans la base.
2. Une commande peut contenir un ou plusieurs articles. Chaque commande a une date.
3. Le modèle doit permettre de retrouver toutes les commandes d'un client et la composition de chaque commande.

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 :

- Un client passe une ou plusieurs commandes. L'association `passee` relie les entités `client` et `commande`. Un client passe une ou plusieurs commandes, ce qui implique une cardinalité 1.N du côté de l'entité `client`. Une commande est passée par un seul client, ce qui implique une cardinalité 1.1 du côté de l'entité `commande`.
- Une commande contient un ou plusieurs articles. Vous créez donc une association `contient`, qui relie les entités `commande` et `article`. La cardinalité du côté de l'entité `commande` est 1.N. Un article pouvant se trouver dans aucune ou plusieurs commandes, la cardinalité du côté de l'entité `article` est 0.N.

Vous obtenez finalement le MCD représenté à la figure 13-8.



**Figure 13-8**  
*Le MCD de la base magasin*

## Passage au modèle relationnel

La phase précédente vous a permis de créer le MCD de la base. C'est la partie la plus importante de votre travail. Il vous faut maintenant créer le modèle logique de données (MLD), qui permettra l'implémentation physique de la base sur des SGBDR comme MySQL ou SQLite.

### *Le modèle relationnel*

La théorie des SGBDR s'appuie sur les notions mathématiques de la théorie des ensembles et des relations entre ensembles. À chaque attribut correspond un ensemble de valeurs possibles (parfois très grand) nommé domaine.

Supposez, par exemple, que vous deviez étudier les notes attribuées à une classe de 26 élèves nommés par des lettres de A à Z et qu'il existe, pour simplifier, quatre catégories de notes de 1 à 4. À la rentrée scolaire, où tous les espoirs sont permis, il y a donc théoriquement  $26 \times 4 = 104$  associations (couples) possibles entre l'ensemble des élèves et celui des notes correspondant au produit cartésien des deux ensembles, soit l'énumération  $\{(A,1),(A,2),(A,3),\dots,(B,1),(B,2),\dots,(Z,1),(Z,2),(Z,3),(Z,4)\}$ . À la fin de l'année, quand les moyennes annuelles sont faites, il n'y a plus que 26 couples réels représentant la réalité de la classe. Cette réalité est représentée par une relation entre l'ensemble des élèves et celui des notes. Dans le cas présent, à un élève ne correspond

qu'une note, mais à une note correspondent plusieurs élèves.

Chaque relation est représentée par une table. Vous pouvez comparer aisément une table à une feuille de tableur que chacun doit connaître. Dans une table, on stocke les informations représentatives d'un type particulier d'objet, réel ou abstrait, comme une personne ou une commande. Tous les objets représentés dans une même table doivent représenter un même concept.

La représentation des notes de la classe serait donc une table à deux colonnes, chaque colonne étant un attribut de la table. La première contient le nom et la seconde la note de l'élève. Elle aurait au total vingt-six lignes.

La figure 13-9 donne une représentation d'une table représentant des articles. La première ligne d'en-tête donne le nom de la table, la deuxième contient la liste des attributs, et les suivantes contiennent les valeurs de plusieurs occurrences, chacune représentant un tuple (ou  $n$ -uplet).

article			
idarticle	designation	prix	categorie
515	PC de bureau	958.35	Informatique
535	Appareil photo	253.58	Photo
647	Portable HB 25	1547.00	Informatique

Diagramme explicatif :

- Le tableau est nommé **table**.
- Les colonnes sont nommées **attributs**.
- Une ligne est nommée **tuple**.

**Figure 13-9**  
*Représentation d'une table*

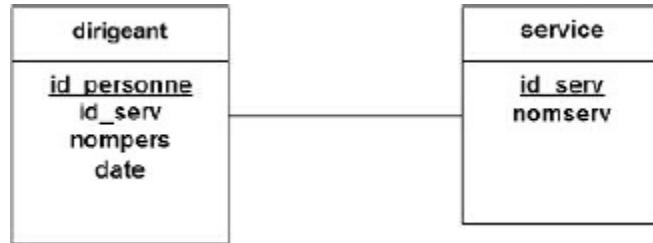
## **Conception du MLD**

Pour passer du MCD au MLD, il vous faut appliquer les règles suivantes :

1. Toute entité devient une table, avec pour corollaires que la table porte le nom de l'entité, que les attributs de l'entité deviennent les colonnes de la table et que la clé primaire de l'entité devient la clé de la table.
2. Pour une association binaire ayant des cardinalités maximales 1:1 (par exemple 1.1-1.1), une des tables reçoit comme attribut supplémentaire une copie de la clé primaire de l'autre table. Cet attribut devient une clé étrangère pour la table qui la reçoit. Les attributs éventuels de l'association sont reportés dans cette même table. Au MCD présenté à la figure 13-5 correspond le MLD de la figure 13-10 pour l'association `dirige`.

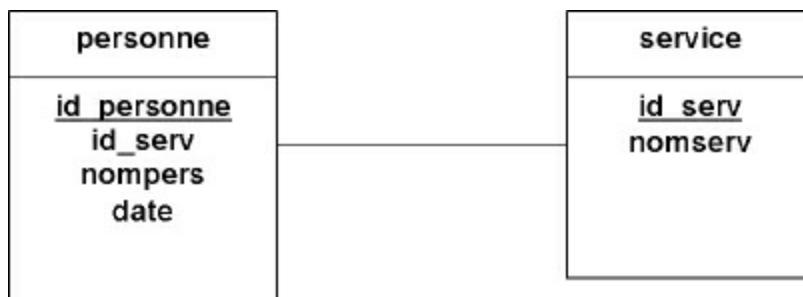
Si vous voulez par exemple conserver l'historique des différents responsables d'un service, vous devez créer une nouvelle table représentant l'association. Elle

contient les clés des deux entités reliées, et la concaténation des ces clés en constitue la clé primaire. Cette table contient également l'attribut de l'association.



**Figure 13-10**  
MLD d'une association 1:1

- Pour une association binaire ayant des cardinalités maximales de type 1:N, par exemple, 1.1-1.N ou 0.1-0.N, la table représentant l'entité ayant la cardinalité 1.1 reçoit la clé de l'autre entité comme clé étrangère. Les attributs de l'association sont ajoutés à cette même table. Le MCD de la figure 13-6 devient le MLD présenté à la figure 13-11 pour les employés d'un service.



**Figure 13-11**  
MLD d'une association 1:N

- Pour une association binaire ayant des cardinalités maximales de type N:M, par exemple, 1.N-1.N ou 0.N-1.N, l'association est toujours traduite par une table. La clé primaire de cette table est la concaténation des clés primaires des entités reliées par cette association. Les attributs de l'association sont ajoutés à cette nouvelle table. Le MCD de la figure 13-7 correspond au MLD de la figure 13-12. Il s'agit d'un MCD primaire, car il suppose qu'une personne ne peut commander qu'un article par commande et une seule fois le même article, faute de quoi vous seriez en infraction avec les règles définies (à une même clé primaire correspondrait deux dates différentes).



**Figure 13-12**  
MLD d'une association N:M

### Association réflexive

Si l'association est réflexive, elle devient une table dont la clé est la concaténation des clés des deux éléments associés. Ce serait le cas de l'association conjointe envisagée plus haut.

## Le MLD de la base magasin en ligne

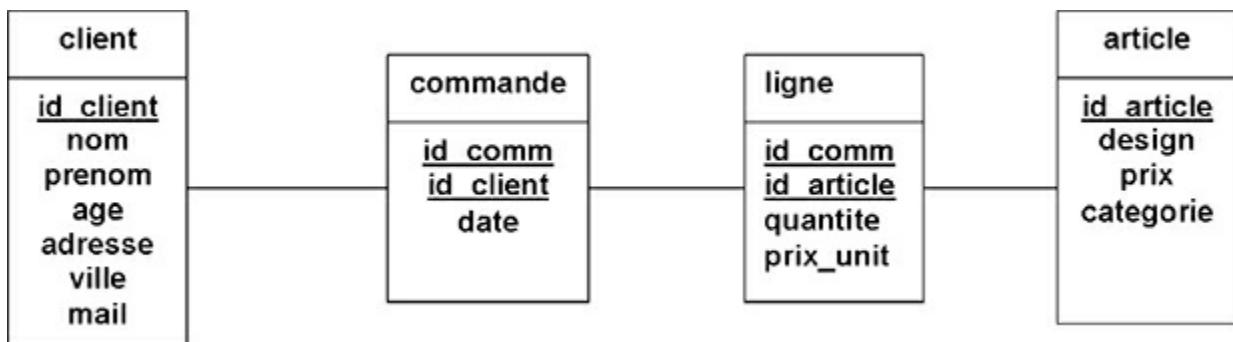
En appliquant les règles énoncées précédemment, le MLD de la base magasin du site de commerce en ligne est construit à partir du MCD représenté à la figure 13-8 de la manière suivante :

- Les entités client, commande et article deviennent des tables.
- L'association passe qui relie les entités client et commande ne devient pas une table. La clé primaire de la table commande reçoit la clé primaire de la table client comme clé étrangère.
- L'association contient, dont les cardinalités maximales sont de type N:M, devient une table nommée ligne, dont chaque occurrence représente une ligne d'une commande. La clé primaire de cette table est la concaténation des clés id\_comm de la table commande et id\_article de la table article. Elle reçoit les attributs quantite et prix\_unit.

### L'attribut prix\_unit

L'attribut prix\_unit n'existe que pour permettre de retrouver la réalité d'une commande ancienne, même après modification d'un tarif dans la table article.

La figure 13-13 présente le MLD obtenu.



**Figure 13-13**  
MLD de la base magasin en ligne

## Modèle physique de données

Le modèle physique de données est l'implémentation matérielle du MLD créé sur un

système (SGBDR) donné. Il est réalisé à l'aide du langage SQL (*Structured Query Language*) et des particularités de chaque système, tel que MySQL, SQLite ou Microsoft Access. L'apprentissage du langage SQL spécifiquement orienté vers MySQL fait l'objet du chapitre suivant.

## Exercices

### Exercice 1

Créez le MCD d'une base de données `voiture` qui enregistre les certificats d'immatriculation des véhicules en circulation (carte grise). Elle doit répondre aux contraintes suivantes :

- Un véhicule est d'un modèle donné identifié par un numéro de type.
- Un véhicule peut avoir un ou plusieurs propriétaires simultanément (copropriété).
- Les recherches effectuées sur la base doivent permettre de retrouver, par exemple, tous les véhicules d'une personne, la ou les personnes propriétaires d'un véhicule dont on connaît l'immatriculation et tous les propriétaires d'un modèle de voiture donné.

### Exercice 2

Créez le MLD de la base `voiture` à partir du MCD de l'exercice 1. Vérifiez la conformité du modèle par rapport aux formes normales.

### Exercice 3

Créez le MCD d'une base de données `tournoi` permettant d'enregistrer les participants à un tournoi de tennis et l'ensemble des matches joués en trois sets au maximum. La base doit enregistrer les participants d'un match donné, ainsi que le gagnant et le score de chaque set.

### Exercice 4

Créez le MLD de la base `tournoi`, et vérifiez sa conformité.

### Exercice 5

Créez le MCD d'une base permettant à un groupe de gérer les droits d'auteur des livres publiés par ses différentes maisons d'édition. Elle doit répondre aux contraintes suivantes :

- Un livre peut être écrit par un ou plusieurs auteurs. Un auteur peut écrire un ou plusieurs livres. Chaque auteur touche un pourcentage des droits totaux d'un livre en fonction de sa participation.
- Un livre est publié par un seul éditeur.

### Exercice 6

Créez le MLD correspondant à la base de l'exercice 5, et vérifiez sa conformité.

# Le langage SQL et phpMyAdmin

Créé par IBM il y a plus de trente cinq ans en tant que langage de manipulation de données, le SQL (*Structured Query Language*) est aujourd’hui le standard de la plupart des SGBDR, notamment de MySQL et de SQLite, présentés en détail dans cet ouvrage. Ce langage a fait l’objet de normalisations successives de la part de l’ANSI (*American National Standards Institute*) pour aboutir en 1992 à SQL 2 puis en 1999 à SQL 3 et enfin à SQL 2003, ce qui montre une constante évolution afin de s’adapter aux besoins.

Malgré cet effort de standardisation, chaque SGBD adapte le langage SQL, en n’utilisant que certaines fonctionnalités et en en ajoutant d’autres, non standards. Si vous avez déjà une connaissance de SQL acquise pour un autre système que MySQL vous ne serez pas désorientés. Il vous faudra simplement effectuer quelques adaptations non fondamentales. Toutes les commandes utilisées dans ce chapitre sont celles de MySQL.

## L’interface phpMyAdmin

Même si le couple PHP-MySQL est le plus répandu actuellement sur le Web, MySQL est accessible à d’autres langages, notamment Java. La totalité des hébergements PHP-MySQL offrent une interface nommée phpMyAdmin, à partir de laquelle il est possible, pour l’administrateur seulement, d’effectuer toutes les opérations de création de base et de tables, d’insertion ainsi que de sélection de données. L’interface phpMyAdmin installée sur vos serveurs local et distant est en réalité un formulaire écrit en PHP permettant d’agir sur la base.

L’interface phpMyAdmin vous permet d’envoyer au serveur des requêtes SQL de création et d’administration de base sans avoir à les écrire. L’interface affiche de surcroît le code SQL de la requête qui vient d’être exécutée.

Pour y accéder à partir d’un navigateur, il vous suffit de saisir les adresses suivantes :

en local :

■ <http://localhost/phpmyadmin>

ou :

■ <http://127.0.0.1/phpmyadmin>

pour des serveurs distants :

| <http://sql.votresite.com>

ou

| <http://www.votresite.com/phpmyadmin>

ou encore parfois :

| <http://www.votresite.com/phpMyAdmin>

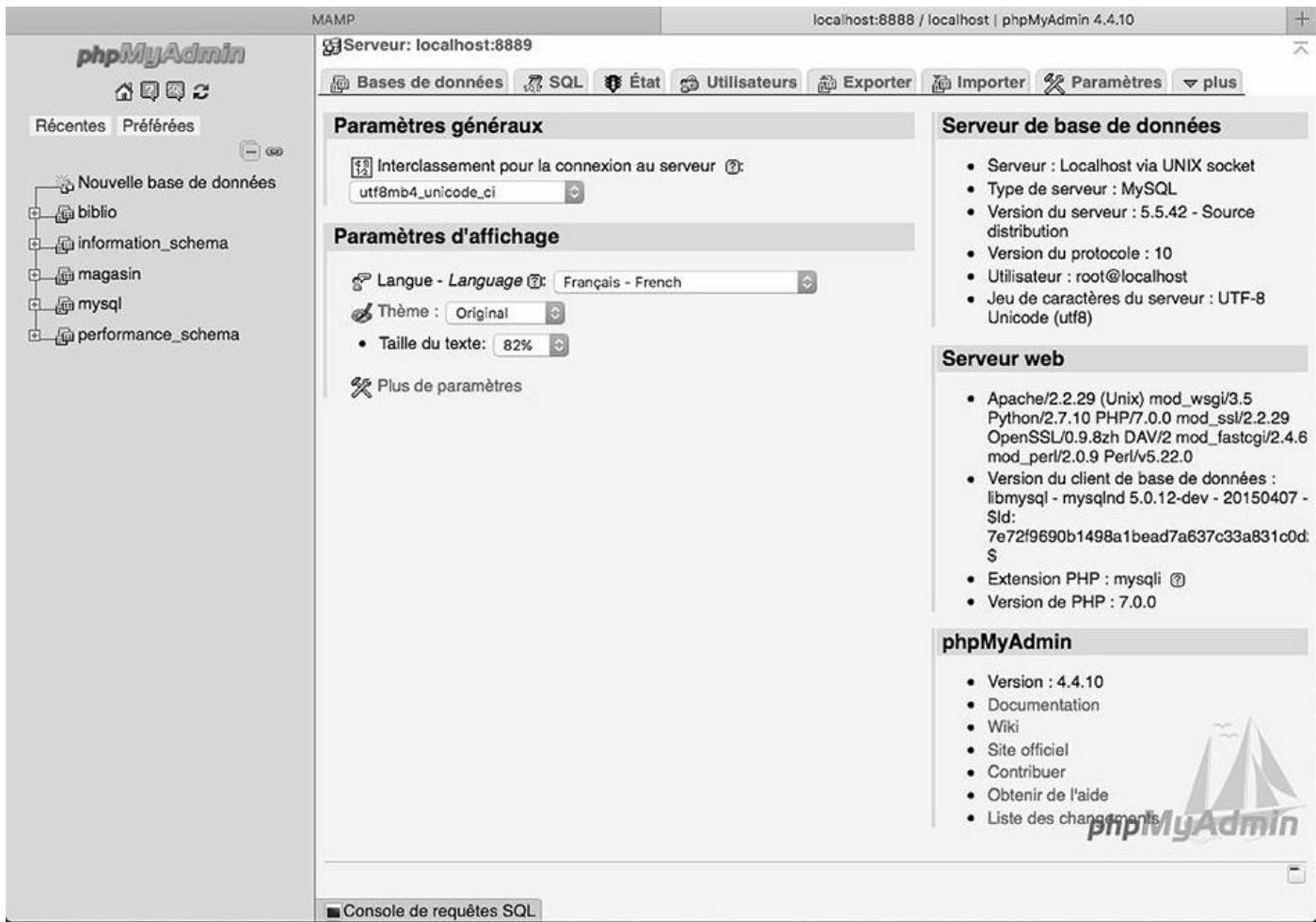
selon la sensibilité à la casse de votre serveur et la casse employée pour le répertoire correspondant.

Les options d'accès sont communiquées par l'hébergeur lors de la souscription d'un abonnement.

En local, vous obtenez la page d'accueil illustrée à la figure 14-1. La version actuelle de phpMyAdmin fournie sur le serveur local installé avec WampServer étant la 3.5, c'est avec cette version que vous allez travailler. Pour une version différente, certaines actions doivent être adaptées en saisissant les requêtes SQL correspondantes.

Certains hébergeurs, tel ovh.net, autorisent l'accès au répertoire contenant le code de phpMyAdmin. Il est de la sorte possible d'installer la dernière version en la téléchargeant à l'adresse <http://www.phpmyadmin.net>.

Les sections qui suivent expliquent comment créer une base de données à l'aide de phpMyAdmin et détaillent les tables qui la composent, ainsi que les commandes permettant d'y insérer des informations, de les modifier ou de les mettre à jour et d'effectuer les différentes formes de sélection de données destinées à la création de pages web dynamiques. Vous verrez, dans les chapitres suivants, comment interfaçer PHP et MySQL au moyen des fonctions spécialisées ou d'objets et méthodes disponibles dans les différents modules spécialisés fournis par PHP.



**Figure 14-1**  
Page d'accueil de phpMyAdmin

## Création d'une base de données

De nombreux hébergeurs, qu'ils soient gratuits ou payants, n'offrent la possibilité de créer qu'une seule base de données sur leur serveur. Vous verrez cependant qu'il est possible de créer en local plusieurs bases différentes.

Pour attribuer un nom à une base, il faut respecter les conventions suivantes, lesquelles s'appliquent également aux noms des tables, attributs (colonnes), index et alias utilisés :

- Les caractères autorisés sont les caractères alphanumériques et de soulignement (\_) ainsi que le signe dollar (\$).

### Slash et addslashes

Pour éviter de confondre ces noms avec les noms des variables PHP utilisées dans un même code, il est déconseillé d'utiliser les caractères slash (/), addslashes (\) et le point.

- Le nombre de caractères est limité à 64, extensible à 255 pour les noms d'alias. Plus le nom est long, plus il y a de risque d'erreur en le recopiant.
- Un nom peut commencer par un chiffre mais ne doit pas contenir que des chiffres.

- La sensibilité à la casse dépend du système d'exploitation. Autant considérer que MySQL est sensible à la casse et surtout éviter de changer de casse dans une même requête, notamment pour les noms d'alias de table.

Pour créer la base, il suffit de cliquer sur l'onglet Bases de données, de saisir ensuite le nom de la base désirée (ici `magasin`) dans la zone de saisie (repère 1 de la figure 14-2) et de cliquer sur le bouton Créer (repère 2) en précisant les jeux de caractères à utiliser pour la connexion et les données. Une fois la base créée, un message de confirmation apparaît (repère 3) et le nom de la base apparaîtra dans la colonne de gauche de l'interface (repère 4).

Le code de la requête SQL correspondant à cette création est le suivant :

```
| CREATE DATABASE magasin DEFAULT CHARACTER SET latin1 COLLATE latin1_bin;
```

En ajoutant l'option `IF NOT EXISTS` avant le nom de la base, cette dernière n'est créée que si elle n'existe pas déjà. Cela évite les messages d'erreur, surtout si la requête est envoyée par un script PHP.

Avec cette option, la requête précédente devient :

```
| CREATE DATABASE IF NOT EXISTS magasin DEFAULT CHARACTER SET latin1  
|   COLLATE latin1_bin;
```

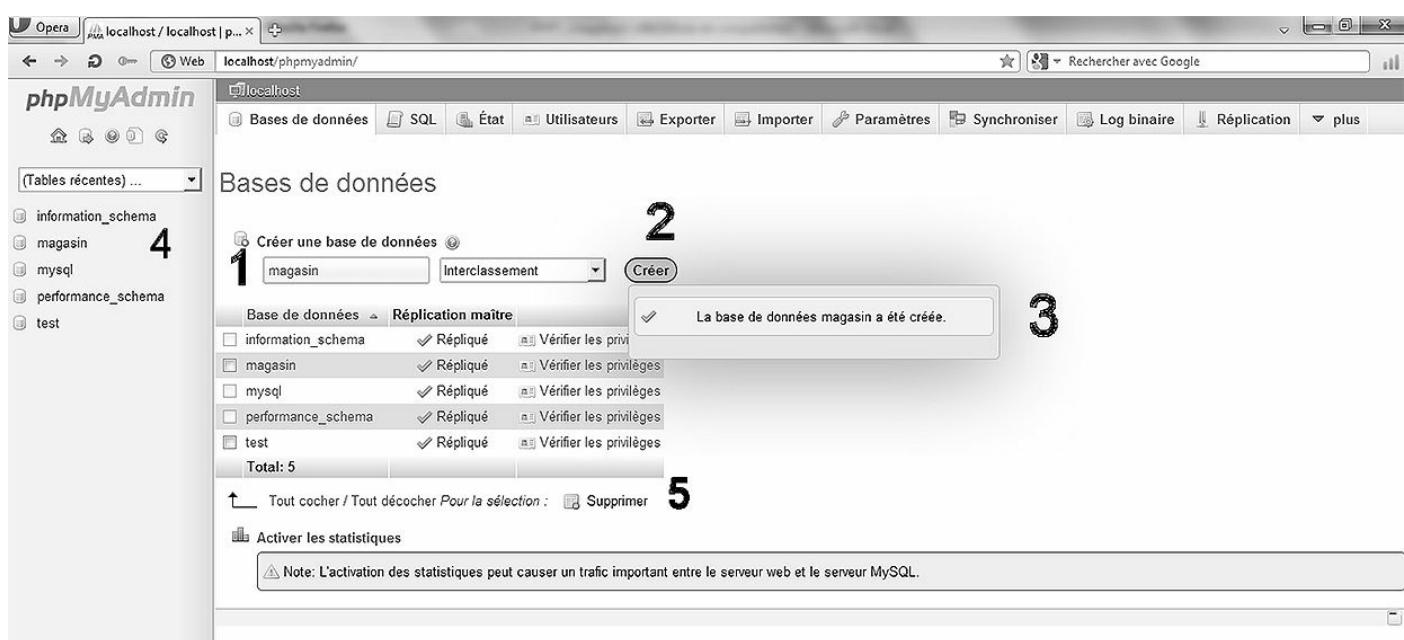
Pour détruire la base, il suffit de la sélectionner puis de cliquer sur le bouton Supprimer (repère 5). Le code SQL correspondant s'affiche :

```
| DROP DATABASE magasin;
```

Comme précédemment, l'usage de l'option `IF EXISTS` avant le nom de la base empêche l'apparition d'une erreur si vous tentez de supprimer une base qui n'existe pas.

Avec cette option, la requête devient :

```
| DROP DATABASE IF EXISTS magasin
```



## Figure 14-2

*La page de confirmation de création de la base*

### Destruction

La destruction d'une base entraîne aussi celle de toutes les tables et de toutes les données qu'elle contient. Il convient donc d'être prudent avant d'utiliser cette requête.

Avant d'effectuer d'autres opérations, il vous faut sélectionner la base sur laquelle vous voulez travailler.

## Création de tables

Une table est composée de colonnes, ou champs. Chacune de ces colonnes est dotée d'un type particulier, censé correspondre le mieux possible à l'information qu'elle contient. Avant de créer une table, il importe de connaître les différents types de données proposés par MySQL. Le choix judicieux de ce type réduit l'espace disque utilisé.

Cette section présente les différents types de données reconnus par MySQL ainsi que les commandes SQL utilisables pour créer les tables.

### *Les types de données MySQL*

MySQL permet de stocker des informations selon de nombreux types de données. Cela donne la possibilité d'adapter le plus précisément possible le type de donnée à l'information qui est enregistrée dans une table. Le choix du type doit se faire de telle sorte que l'information occupe le moins d'octets possible.

### Les types numériques

Les types numériques permettent de stocker toutes sortes de valeurs numériques entières ou décimales avec des intervalles de valeurs très étendus. Comme expliqué précédemment, il est important d'adapter le type choisi au format de l'information stockée afin de ne pas occuper plus d'espace serveur que nécessaire et, par voie de conséquence, de ne pas allonger inutilement les temps de réponse des recherches effectuées.

Le tableau 14-1 présente les différents types de données numériques.

**Tableau 14-1 – Les types de données numériques**

Type	Définition et caractéristiques
TINYINT	Un très petit entier prenant des valeurs de $-128 (-2^7)$ à $127 (2^7 - 1)$ . S'il est suivi de l'option <code>UNSIGNED</code> , les valeurs sont positives et varient de $0$ à $255 (2^8 - 1)$ . Chaque valeur occupe 1 octet.

SMALLINT	Un petit entier prenant des valeurs de $-32768$ ( $-2^{15}$ ) à $32767$ ( $2^{15} - 1$ ). S'il est suivi de l'option <code>UNSIGNED</code> , les valeurs sont positives et varient de $0$ à $65535$ ( $2^{16} - 1$ ). Chaque valeur occupe 2 octets.
MEDIUMINT	Entier moyen prenant des valeurs de $-8388608$ ( $-2^{23}$ ) à $8388607$ ( $2^{23} - 1$ ). S'il est suivi de l'option <code>UNSIGNED</code> , les valeurs sont positives et varient de $0$ à $16777215$ ( $2^{24} - 1$ ). Chaque valeur occupe 3 octets.
INT OU INTEGER	Entier prenant des valeurs de $-2147483648$ ( $-2^{31}$ ) à $2147483647$ ( $2^{31} - 1$ ). S'il est suivi de l'option <code>UNSIGNED</code> , les valeurs sont positives et varient de $0$ à $4294967295$ soit $2^{32} - 1$ . Chaque valeur occupe 4 octets.
BIGINT	Grand entier prenant des valeurs de $-9223372036854775808$ ( $-2^{63}$ ) à $9223372036854775807$ ( $2^{63} - 1$ ). S'il est suivi de l'option <code>UNSIGNED</code> , les valeurs sont positives et varient de $0$ à $18446744073709551615$ , soit $2^{64} - 1$ . Chaque valeur occupe 8 octets.
FLOAT	Nombre à virgule flottante en simple précision prenant des valeurs de $-3.402823466E+38$ à $-1.75494351E-38$ pour les nombres négatifs et de $1.75494351E-38$ à $3.402823466E+38$ pour les positifs. S'il est suivi de l'option <code>UNSIGNED</code> , les valeurs sont uniquement positives. Avec les options <code>FLOAT (M, D)</code> , l'affichage s'effectue avec $M$ chiffres dont $D$ décimales. Chaque valeur occupe 4 octets.
DOUBLE	Nombre à virgule flottante en double précision prenant des valeurs de $-1.7976931348623157E+308$ à $-2.2250738585072014E-308$ pour les nombres négatifs et de $2.2250738585072014E-308$ à $1.7976931348623157E+308$ pour les positifs, auxquelles s'ajoute la valeur exacte de $0$ . S'il est suivi de l'option <code>UNSI-GNED</code> , les valeurs sont uniquement positives. Avec les options <code>DOUBLE (M, D)</code> l'affichage se fait avec $M$ chiffres dont $D$ décimales. Chaque valeur occupe huit octets.
DECIMAL	Nombre à virgule flottante qui doit être signé. La valeur est stockée comme une chaîne de caractères dont chaque caractère est un chiffre. Les valeurs sont les mêmes que pour le type <code>DOUBLE</code> . S'il est suivi de l'option <code>UNSIGNED</code> , les valeurs sont uniquement positives. Avec les options <code>DECIMAL (M)</code> , l'affichage s'effectue avec $M$ chiffres (par défaut 10 chiffres), ce qui limite l'intervalle de valeurs. Avec les options <code>DECIMAL (M, D)</code> , l'affichage s'effectue avec $M$ chiffres dont $D$ décimales. Chaque valeur occupe autant d'octets qu'il y a de caractères dans le nombre.

## Les types chaînes de caractères

Ces types permettent de stocker des chaînes de caractères de longueurs très diverses, allant du simple caractère au discours fleuve d'un visiteur prolix dans un livre d'or. À nouveau, il vous faut penser à adapter le type de donnée à ce que vous désirez stocker afin d'utiliser le moins d'octet possible.

Le tableau 14-2 présente les différents types de chaînes de caractères.

Tableau 14-2 – Les types de chaînes de caractères

Type	Définition et caractéristiques
CHAR (M)	Chaîne de caractères de longueur fixe de $M$ caractères complétée par des espaces si la donnée stockée est plus petite. Les espaces sont supprimées lors de la lecture. La longueur indiquée varie de 0 à 255 caractères. L'option <code>CHAR(M) BINARY</code> rend la chaîne sensible à la casse lors des recherches. Une colonne de type <code>CHAR(0)</code> n'occupe qu'un octet et peut contenir les valeurs <code>NULL</code> et <code>""</code> (chaîne vide), ce qui permet de simuler une valeur booléenne. La chaîne stockée occupe toujours $M$ octets, même si elle ne contient qu'un seul caractère significatif.
VARCHAR (M)	Chaîne de caractères de longueur variable comprise entre 1 et $M$ caractères. La valeur de $M$ varie de 1 à 255 caractères. L'option <code>VARCHAR(M) BINARY</code> rend la chaîne sensible à la casse lors des recherches. La chaîne stockée occupe $N + 1$ octets quand elle comprend $N$ caractères.
TINYTEXT TINYBLOB	Texte d'une longueur comprise entre 1 et 255 caractères. Le type <code>TINYBLOB</code> est sensible à la casse. La chaîne stockée occupe $N + 1$ octets quand elle comprend $N$ caractères.
TEXT BLOB	Texte d'une longueur comprise entre 1 et 65 535 caractères. Le type <code>BLOB</code> est sensible à la casse. La chaîne stockée occupe $N + 2$ octets quand elle comprend $N$ caractères.
MEDIUMTEXT MEDIUMBLOB	Texte d'une longueur comprise entre 1 et 16 777 215 caractères. Le type <code>MEDIUMBLOB</code> est sensible à la casse. La chaîne stockée occupe $N + 3$ octets quand elle comprend $N$ caractères.
LONGTEXT LONGBLOB	Texte d'une longueur comprise entre 1 et 4 294 967 295 caractères. Le type <code>LONGBLOB</code> est sensible à la casse. La chaîne stockée occupe $N + 4$ octets quand elle comprend $N$ caractères.
ENUM('chaine1', ..., 'chaineN')	Permet le choix d'une seule valeur parmi l'énumération des $N$ chaînes de caractères définies dans le type. La valeur <code>NULL</code> est toujours admise, même si elle ne figure pas dans la liste. Le type peut définir jusqu'à 65 535 valeurs. À chaque chaîne correspond une valeur numérique de 1 à 65 535, correspondant à son ordre d'apparition dans la définition du type. La valeur 0 correspond à une chaîne vide. La définition de <code>ENUM ('bleu', 'blanc', 'rouge')</code> pour une colonne ne permet de stocker qu'une valeur parmi les trois couleurs de la liste ou la valeur <code>NULL</code> . Pour le HTML, ce type correspond à une liste de sélection <code>&lt;select&gt;</code> de $N+1$ <code>&lt;option&gt;</code> (les $N$ valeurs proposées plus le choix <code>NULL</code> par défaut) à choix unique.
SET('chaine1', ..., 'chaineN')	Permet le choix de une ou plusieurs valeurs simultanément parmi l'ensemble des $N$ chaînes de caractères définies dans le type. L'ensemble peut contenir jusqu'à 64 valeurs. À chaque choix correspond une valeur numérique entière égale à $2^{n-1}$ si $n$ est la position de la chaîne dans l'ensemble (soit 1 pour la première, 2 pour la deuxième, 4 pour la troisième, etc.) ou encore une valeur binaire dans laquelle chaque bit est à 1 si la valeur est choisie (soit <code>0001</code> pour la première, <code>0010</code> pour la deuxième, <code>0100</code> pour la troisième, etc.). Si

plusieurs valeurs sont choisies, la valeur numérique correspondante est la somme des valeurs de chacune (par exemple 5 pour la première et la troisième valeur). En HTML, ce type correspond à une liste de sélection `<select>` de `N <option>` à choix multiple (avec l'attribut `multiple`).

## Les types de dates et d'heures

Les types de dates et d'heures permettent de stocker des dates dans des formats différents, allant de la simple année seule à l'ensemble date complète plus heure complète à la seconde près.

Pour stocker un timestamp Unix tel que détaillé au chapitre 8, il est préférable d'utiliser une colonne de type entier `INT(10)`, qui facilite les opérations.

Le tableau 14-3 présente les différents types de données de dates et d'heures.

**Tableau 14-3 – Les types de dates et d'heures**

Type	Définition et caractéristiques														
DATE	Une date au format <code>AAAA-MM-JJ</code> dans l'intervalle de <code>1000-01-01</code> à <code>9999-12-31</code> . Chaque enregistrement occupe 3 octets.														
DATETIME	Contient la date et l'heure au format <code>AAAA-MM-JJ HH:MM:SS</code> dans l'intervalle de <code>1000-01-01 00:00:00</code> à <code>9999-12-31 23:59:59</code> . Chaque enregistrement occupe 8 octets.														
TIMESTAMP [ (M) ]	Stocke une date complète sous la forme <code>AAAAMMJJHHMMSS</code> sans rapport direct avec un timestamp Unix tel que retourné par la fonction <code>time()</code> détaillée au chapitre 8.  Il faut toutefois prendre des précautions pour effectuer des calculs avec ce type de valeur. En effet, le calcul <code>20030506232009 + 1030</code> n'ajoute pas 1 030 secondes mais 10 minutes et 30 secondes. De plus, l'addition peut conduire à des dates invalides par dépassement des valeurs admises (25 heures, par exemple). Le paramètre <code>M</code> facultatif détermine le nombre de caractères utilisé pour afficher la date. Ce doit être un nombre pair. Sa valeur par défaut est 14. En fonction de la valeur de <code>M</code> , vous obtenez les formats d'affichage suivants :  <table> <tbody> <tr> <td><code>TIMESTAMP(14)</code></td> <td><code>AAAAMMJJHHMMSS</code></td> </tr> <tr> <td><code>TIMESTAMP(12)</code></td> <td><code>AAMMJJHHMMSS</code></td> </tr> <tr> <td><code>TIMESTAMP(10)</code></td> <td><code>AAMMJJHHMM</code></td> </tr> <tr> <td><code>TIMESTAMP(8)</code></td> <td><code>AAAAMMJJ</code></td> </tr> <tr> <td><code>TIMESTAMP(6)</code></td> <td><code>AAMMJJ</code></td> </tr> <tr> <td><code>TIMESTAMP(4)</code></td> <td><code>AAMM</code></td> </tr> <tr> <td><code>TIMESTAMP(2)</code></td> <td><code>AA</code></td> </tr> </tbody> </table> Chaque enregistrement occupe quatre octets.	<code>TIMESTAMP(14)</code>	<code>AAAAMMJJHHMMSS</code>	<code>TIMESTAMP(12)</code>	<code>AAMMJJHHMMSS</code>	<code>TIMESTAMP(10)</code>	<code>AAMMJJHHMM</code>	<code>TIMESTAMP(8)</code>	<code>AAAAMMJJ</code>	<code>TIMESTAMP(6)</code>	<code>AAMMJJ</code>	<code>TIMESTAMP(4)</code>	<code>AAMM</code>	<code>TIMESTAMP(2)</code>	<code>AA</code>
<code>TIMESTAMP(14)</code>	<code>AAAAMMJJHHMMSS</code>														
<code>TIMESTAMP(12)</code>	<code>AAMMJJHHMMSS</code>														
<code>TIMESTAMP(10)</code>	<code>AAMMJJHHMM</code>														
<code>TIMESTAMP(8)</code>	<code>AAAAMMJJ</code>														
<code>TIMESTAMP(6)</code>	<code>AAMMJJ</code>														
<code>TIMESTAMP(4)</code>	<code>AAMM</code>														
<code>TIMESTAMP(2)</code>	<code>AA</code>														
TIME	Stocke l'heure au format <code>HH:MM:SS</code> ou <code>HHH:MM:SS</code> pour un intervalle de valeurs allant de <code>-838:59:59</code> à <code>838:59:59</code> permettant d'effectuer des calculs de durée excédant 24 heures. Chaque enregistrement occupe 3 octets.														
	Représente les années au format <code>YYYY</code> pour un intervalle allant de 1901														

à 2155. Si l'année est fournie avec deux chiffres, les valeurs de 00 à 69 correspondent aux années 2000 à 2069 et les valeurs 70 à 99 aux années 1970 à 1999. Chaque enregistrement occupe 1 octet.

## Les options des attributs

Chaque attribut d'une table peut être précisé à l'aide des options suivantes :

- NOT NULL pour que chaque enregistrement de l'attribut ait obligatoirement une valeur ou NULL pour autoriser l'absence de valeur (cette dernière option est interdite pour une clé primaire). Nous écrivons, par exemple :

```
| nom_attr VARCHAR(10) NOT NULL
```

- DEFAULT 'valeur\_défaut' permet de définir une valeur par défaut pour l'attribut si aucune valeur n'y est enregistrée. Cette option est impossible pour les types BLOB et TEXT. Les valeurs par défaut doivent être des constantes, et il n'est pas possible d'utiliser de fonction pour les définir.
- AUTO\_INCREMENT pour qu'un attribut numérique entier soit automatiquement incrémenté d'une unité à chaque insertion d'un enregistrement. Pour que cette contrainte soit valable, il faut que l'attribut soit indexé ou qu'il soit la clé primaire. Par exemple, le code suivant :

```
| nom tinyint NOT NULL auto_increment, INDEX indnom (nom)
```

crée un attribut nom auto-incrémenté NOT NULL et indexé sous le nom indnom.

- PRIMARY KEY pour définir l'attribut comme clé primaire de la table. Il est recommandé de faire cette déclaration après la définition de tous les attributs. Par exemple, le code suivant :

```
| nom tinyint(4) NOT NULL AUTO_INCREMENT, PRIMARY KEY (nom)
```

définit un attribut nom et une clé primaire sur un attribut.

Le code suivant :

```
| PRIMARY KEY (nom,prenom)
```

crée une clé primaire composée des deux attributs, nom et prenom.

De même, pour définir le ou les index de la table, vous devez faire figurer les contraintes suivantes après les définitions d'attributs :

- UNIQUE(nom\_attribut1,nom\_attribut2,...) pour que chaque enregistrement ait une valeur unique dans la colonne des attributs précisés. Par exemple, le code suivant :

```
| UNIQUE (nom,prenom)
```

empêche d'enregistrer deux personnes ayant le même nom et le même prénom. Par contre, deux noms identiques avec deux prénoms différents sont acceptés.

- INDEX[nom\_index] (nom\_attribut1,nom\_attribut2,...) crée un index pour la table à partir des colonnes précisées. La création d'index sur des colonnes de la table facilite les recherches quand ces colonnes sont utilisées comme critère de recherche.

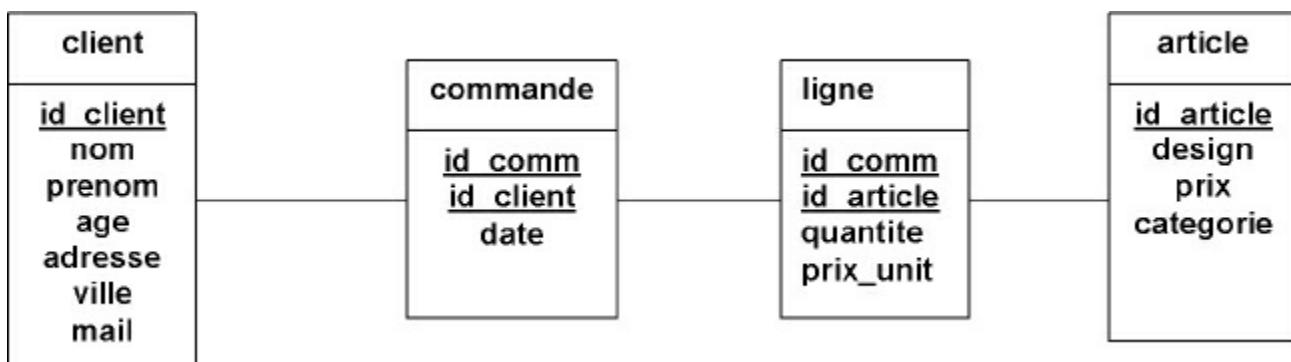
Par exemple, le code suivant :

```
| INDEX mon_index (nom, prenom)
```

crée un index nommé `mon_index` sur les colonnes `nom` et `prenom`.

## Création des tables

Vous allez maintenant créer les tables destinées à contenir les données. Les sections qui suivent détaillent les quatre tables à créer pour la base `magasin`, les tables `client`, `commande`, `article` et `ligne`. La figure 14-3 rappelle le MLD de la base `magasin` qui va nous permettre de créer les différentes tables qui la composent.



**Figure 14-3**  
Le MLD de la base `magasin`

## La table client

Le code suivant définit les colonnes de la table `client`.

```
| client(id_client, nom, prenom, age, adresse, ville, mail)
```

La colonne `id_client` est de type `MEDIUMINT`, avec l'option `UNSIGNED` sélectionnée. Sa valeur est donc un entier positif, avec 16 777 215 clients possibles. Cet attribut étant la clé primaire de la table, choisissez dans phpMyAdmin les options `PRIMARY KEY` (colonne Index de la figure 14-5) et `NOT NULL` car le champ `id_client` doit avoir obligatoirement une valeur (ne pas cocher la case dans la colonne `Null` de la figure 14-5). Ce champ étant incrémenté automatiquement d'une unité à chaque nouvelle insertion de donnée, il a également l'option `AUTO_INCREMENT` choisie dans la colonne `A.I.` de la figure 14-5.

Les colonnes `nom`, `prenom`, `adresse` et `ville` sont des chaînes de caractères de longueur variable, donc de type `VARCHAR`, avec l'option `NOT NULL` sélectionnée.

La colonne `age` est un entier positif de petite taille, donc de type `TINYINT UNSIGNED`. L'intervalle des valeurs est de 0 à 255. L'option `NULL` permet à un client de ne pas saisir son âge.

La colonne `mail` est également de type `VARCHAR`, mais cette fois avec l'option `NULL` sélectionnée, car un client peut ne pas donner son adresse e-mail.

## La table commande

La table `commande` a la structure suivante :

```
| commande(id_comm, id_client, date)
```

La colonne `id_comm` est de type `MEDIUMINT`, avec les options `UNSIGNED`, `NOT NULL` et `AUTO_INCREMENT` sélectionnées, de façon que chaque commande ait un numéro différent.

La colonne `id_client` est une clé étrangère issue de la table `client`. Elle a les mêmes caractéristiques que `id_comm`. La clé primaire de la table est composée des champs `id_comm` et `id_client`.

La colonne `date` est de type `DATE`, avec l'option `NOT NULL` sélectionnée, de façon que les valeurs soient insérées automatiquement sans intervention du client et qu'il n'y ait pas à craindre un oubli.

## La table article

La table `article` a la structure suivante :

```
| article(id_article, designation, prix, categorie)
```

La colonne `id_article` correspond au code de chaque article sur cinq caractères. Elle est donc de type `CHAR(5)`, avec les options `NOT NULL` et `PRIMARY KEY` sélectionnées.

La colonne `designation` est de type `VARCHAR(100)`, avec l'option `NOT NULL` sélectionnée.

La colonne `prix` est de type `DECIMAL(8,2)`. Elle est donc sur huit chiffres, dont deux décimales, avec l'option `NOT NULL` sélectionnée.

La colonne `categorie` est de type `ENUM`. Elle comporte une liste de cinq valeurs possibles ('tous', 'vidéo', 'photo', 'informatique', 'divers') avec l'option `NOT NULL` sélectionnée et la valeur par défaut 'tous'.

## La table ligne

La table `ligne` a la structure suivante :

```
| ligne(id_comm,id_article,quantite)
```

La colonne `id_comm` est une clé étrangère issue de la table `commande` et la colonne `id_article` une clé étrangère issue de la table `article`, ces colonnes doivent donc être déclarées avec les mêmes types de données que celles auxquelles elles se réfèrent dans les tables `commande` et `article`. La colonne `quantite` est un petit entier de type `TINYINT`, avec les options `UNSIGNED` et `NOT NULL` sélectionnées. L'intervalle de valeurs est de 1 à 255. La clé primaire de la table est composée des attributs `id_comm` et `id_article`.

### Hiérarchie `base.table.colonne`

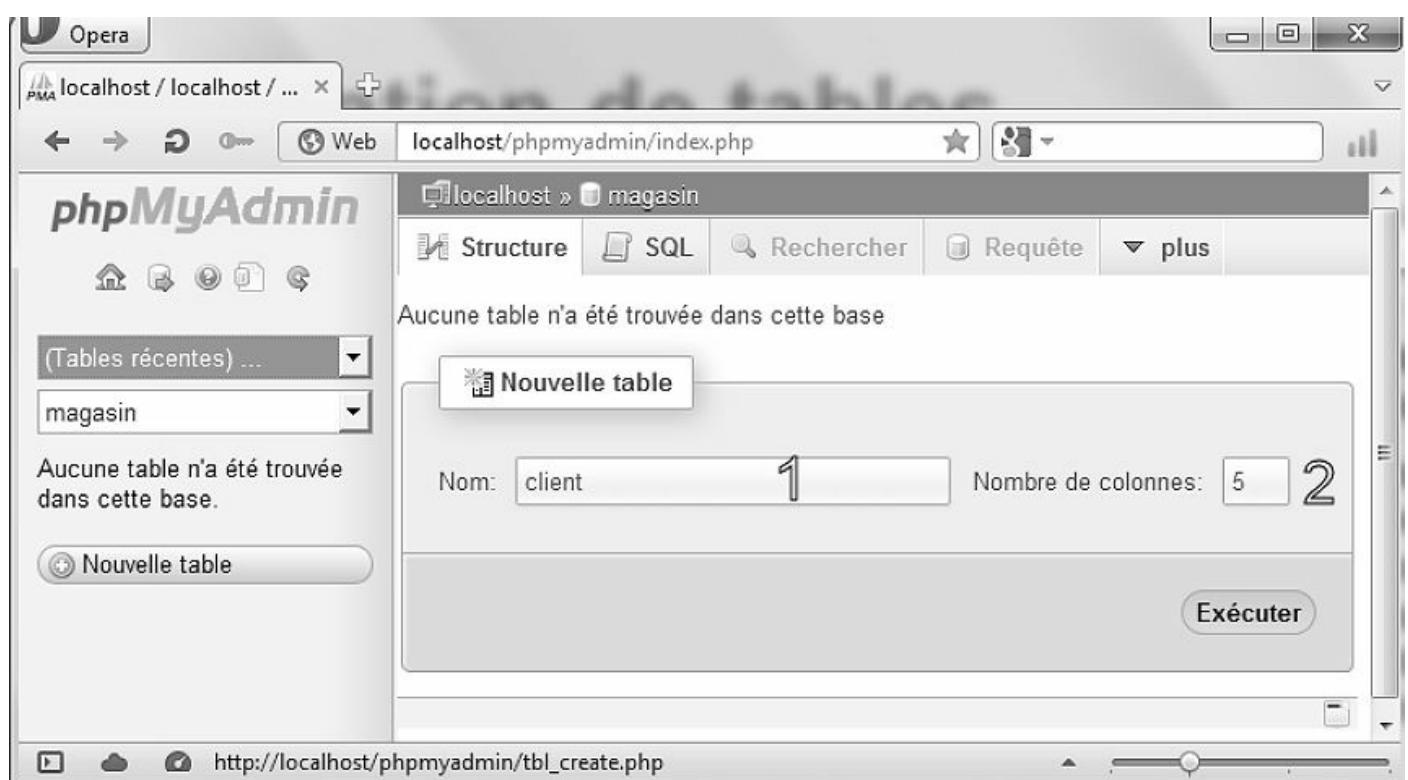
Il est possible que plusieurs bases aient des tables de même nom et que des tables différentes aient des colonnes de même nom. Pour accéder sans ambiguïté à une colonne précise il faut

employer la syntaxe suivante : nom\_base.nom\_table.nom\_colonne

## En résumé

Pour créer une table, par exemple la table `client`, procédez de la façon suivante :

1. Sélectionnez la base dans la page d'accueil de phpMyAdmin.
2. Saisissez un nom pour la table et le nombre de champs qui la composent dans les zones de saisie illustrées aux repères 1 et 2 de la figure 14-4.
3. Cliquez sur le bouton Exécuter.
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 14-4**  
*Création d'une table avec phpMyAdmin*

### 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.

Nom de la table: client Ajouter 1 colonne(s) Exécuter

Nom	Type	Taille/Valeurs*	Défaut	Interclassement	Attributs	Null	Index	A_I Comment
id_client	MEDIUMINT		Aucune		UNSIGNED	<input type="checkbox"/>	PRIMARY	<input checked="" type="checkbox"/>
nom	VARCHAR	30	Aucune	latin1_bin		<input type="checkbox"/>	...	<input type="checkbox"/>
prenom	VARCHAR	30	Aucune	latin1_bin		<input checked="" type="checkbox"/>	...	<input type="checkbox"/>
adresse	VARCHAR	60	Aucune	latin1_bin		<input type="checkbox"/>	...	<input type="checkbox"/>
mail	VARCHAR	50	NULL			<input checked="" type="checkbox"/>	...	<input type="checkbox"/>

Commentaires sur la table: Moteur de stockage: Interclassement:

InnoDB latin1\_bin

Sauvegarder Annuler

**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.

Afficher Structure SQL Rechercher Insérer Exporter Importer Opérations Déclencheurs

La table client a été modifiée avec succès

ALTER TABLE `client` CHANGE `mail` `mail` VARCHAR( 50 ) CHARACTER SET latin1 COLLATE latin1\_bin NULL

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	id_client	mediumint(8)		UNSIGNED	Non	Aucune	AUTO_INCREMENT	<input type="checkbox"/> Modifier <input checked="" type="checkbox"/> Supprimer <input type="checkbox"/> Affiche les valeurs distinctes <input type="checkbox"/> Primaire <input type="checkbox"/> Unique <input type="checkbox"/> plus
2	nom	varchar(30)	latin1_bin		Non	Aucune		<input type="checkbox"/> Modifier <input checked="" type="checkbox"/> Supprimer <input type="checkbox"/> Affiche les valeurs distinctes <input type="checkbox"/> Primaire <input type="checkbox"/> Unique <input type="checkbox"/> plus
3	prenom	varchar(30)	latin1_bin		Oui	NULL		<input type="checkbox"/> Modifier <input checked="" type="checkbox"/> Supprimer <input type="checkbox"/> Affiche les valeurs distinctes <input type="checkbox"/> Primaire <input type="checkbox"/> Unique <input type="checkbox"/> plus
4	adresse	varchar(60)	latin1_bin		Non	Aucune		<input type="checkbox"/> Modifier <input checked="" type="checkbox"/> Supprimer <input type="checkbox"/> Affiche les valeurs distinctes <input type="checkbox"/> Primaire <input type="checkbox"/> Unique <input type="checkbox"/> plus
5	mail	varchar(50)	latin1_bin		Oui	NULL		<input type="checkbox"/> Modifier <input checked="" type="checkbox"/> Supprimer <input type="checkbox"/> Affiche les valeurs distinctes <input type="checkbox"/> Primaire <input type="checkbox"/> Unique <input type="checkbox"/> plus

Tout cocher / Tout décocher Pour la sélection :  Afficher  Modifier  Supprimer  Primaire  Unique  Index

Version imprimable  vue relationnelle  Suggérer des optimisations quant à la structure de la table

Ajouter 1 colonne(s)  En fin de table  En début de table  Après id\_client  Exécuter

**Figure 14-6**  
*Structure de la table client*

Vous pouvez créer de la même façon les autres tables de la base magasin en prenant soin

de respecter les définitions données précédemment.

Pour chacune de ces tables, le code de création est le suivant :

- **table commande :**

```
CREATE TABLE IF NOT EXISTS 'commande' (
    'id_comm' mediumint(8) unsigned NOT NULL AUTO_INCREMENT,
    'id_client' mediumint(8) unsigned NOT NULL,
    'date' date NOT NULL,
    PRIMARY KEY ('id_comm','id_client')
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin AUTO_INCREMENT=1 ;
```

- **table article :**

```
CREATE TABLE IF NOT EXISTS 'commande' (
    'id_comm' mediumint(8) unsigned NOT NULL AUTO_INCREMENT,
    'id_client' mediumint(8) unsigned NOT NULL,
    'date' date NOT NULL,
    PRIMARY KEY ('id_comm','id_client')
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin AUTO_INCREMENT=1;
```

- **table ligne :**

```
CREATE TABLE IF NOT EXISTS 'ligne' (
    'id_comm' mediumint(8) unsigned NOT NULL AUTO_INCREMENT,
    'id_article' char(5) COLLATE latin1_bin NOT NULL,
    'quantite' tinyint(3) unsigned NOT NULL,
    'prix_unit' decimal(8,2) NOT NULL,
    PRIMARY KEY ('id_comm','id_article')
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COLLATE=latin1_bin AUTO_INCREMENT=1 ;
```

La syntaxe simplifiée de la commande `CREATE TABLE` est la suivante :

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nom_table (
    nom_colonne1 NOM_TYPE [options],
    nom_colonne2 NOM_TYPE [options],
    FOREIGN KEY (nom_colonne,...)
    ....)
```

Le mot-clé `TEMPORARY` indique que la table n'est créée que le temps de la connexion à la base et qu'elle est ensuite effacée.

Le mot-clé `IF NOT EXISTS` permet de ne créer la table que si une autre table du même nom n'existe pas déjà. En l'absence de cette précision, une erreur se produit si une telle table existe déjà.

## ***Modification des tables***

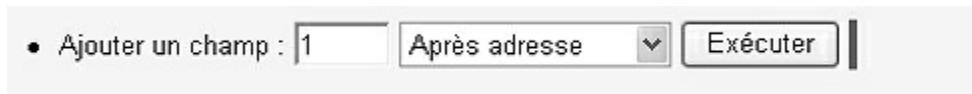
Après sa création, une table n'est pas figée. Elle peut évoluer en fonction des besoins, soit pour ajouter ou ôter des attributs, soit pour modifier ou ajouter des contraintes, des index, par exemple, soit encore pour supprimer toute la table.

### **Ajout d'un champ**

Comme expliqué précédemment, vous avez volontairement oublié deux champs de la table `client` pour illustrer les possibilités de modification offertes par phpMyAdmin.

Vous allez maintenant ajouter ces champs, `ville` et `age`.

Pour ajouter un champ, il suffit, après avoir choisi la table désirée, de saisir le nombre de champs et leur position dans la table, comme l'illustre la figure 14-7 pour l'ajout du champ `ville` après l'adresse. L'ordre des champs n'a pas d'importance pour le stockage des données mais relève plutôt d'une convention personnelle de présentation.



**Figure 14-7**

Ajout du champ `ville` à la table client

Le code généré est le suivant :

```
ALTER TABLE client ADD ville VARCHAR( 40 ) NOT NULL AFTER adresse ;
```

Ajoutez de la même façon le champ `age` après le champ `prenom`.

Le code généré est le suivant :

```
ALTER TABLE 'client' ADD 'age' TINYINT UNSIGNED AFTER 'prenom' ;
```

Pour obtenir automatiquement la structure finale de la table après exécution de la modification ou à tout moment, il suffit de cliquer sur le bouton Structure (voir figure 14-8).

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	<code>id_client</code>	mediumint(8)		UNSIGNED	Non	Aucune	AUTO_INCREMENT	<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/> <input type="button" value="Affiche les valeurs distinctes"/> <input checked="" type="checkbox" value="Primaire"/> Primaire <input checked="" type="checkbox" value="Unique"/> Unique <input type="checkbox"/> Index ▾ plus
2	<code>nom</code>	varchar(30)	latin1_bin		Non	Aucune		<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/> <input type="button" value="Affiche les valeurs distinctes"/> <input checked="" type="checkbox" value="Primaire"/> Primaire <input checked="" type="checkbox" value="Unique"/> Unique <input type="checkbox"/> Index ▾ plus
3	<code>prenom</code>	varchar(30)	latin1_bin		Oui	NULL		<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/> <input type="button" value="Affiche les valeurs distinctes"/> <input checked="" type="checkbox" value="Primaire"/> Primaire <input checked="" type="checkbox" value="Unique"/> Unique <input type="checkbox"/> Index ▾ plus
4	<code>age</code>	tinyint(3)		UNSIGNED	Non	Aucune		<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/> <input type="button" value="Affiche les valeurs distinctes"/> <input checked="" type="checkbox" value="Primaire"/> Primaire <input checked="" type="checkbox" value="Unique"/> Unique <input type="checkbox"/> Index ▾ plus
5	<code>adresse</code>	varchar(60)	latin1_bin		Non	Aucune		<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/> <input type="button" value="Affiche les valeurs distinctes"/> <input checked="" type="checkbox" value="Primaire"/> Primaire <input checked="" type="checkbox" value="Unique"/> Unique <input type="checkbox"/> Index ▾ plus
6	<code>ville</code>	varchar(40)	latin1_bin		Non	Aucune		<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/> <input type="button" value="Affiche les valeurs distinctes"/> <input checked="" type="checkbox" value="Primaire"/> Primaire <input checked="" type="checkbox" value="Unique"/> Unique <input type="checkbox"/> Index ▾ plus
7	<code>mail</code>	varchar(50)	latin1_bin		Oui	NULL		<input type="button" value="Modifier"/> <input type="button" value="Supprimer"/> <input type="button" value="Affiche les valeurs distinctes"/> <input checked="" type="checkbox" value="Primaire"/> Primaire <input checked="" type="checkbox" value="Unique"/> Unique <input type="checkbox"/> Index ▾ plus

**Figure 14-8**

Structure finale de la table client

Vous pouvez vérifier qu'elle répond bien à la définition qui en a été donnée lors de sa conception.

## Modification des propriétés d'une table

Le tableau représenté à la figure 14-8 contient six colonnes regroupées sous la catégorie Action. Ces colonnes permettent de réaliser les modifications suivantes sur les champs de la table :

- Repère 1 : modification des caractéristiques du champ. Par exemple, le code pour renommer le champ `id_client` en `num_client` et pour qu'il soit de type `SMALLINT` en conservant les autres caractéristiques est le suivant :

```
ALTER TABLE 'client' CHANGE 'id_client' 'num_client' SMALLINT NOT
NULL AUTO_INCREMENT
```

- Repère 2 : suppression du champ de la table. Par exemple, le code pour supprimer le champ `ville` est le suivant :

```
ALTER TABLE 'client' DROP 'ville'
```

- Repère 3 : définition du champ comme clé primaire. Par exemple, le code permettant que le champ `nom` fasse aussi partie de la clé primaire est le suivant :

```
ALTER TABLE 'client' DROP PRIMARY KEY ,
ADD PRIMARY KEY ( id_client, 'nom' )
```

Remarquez que la clé primaire est effacée par la commande `DROP PRIMARY KEY` avant d'être recréée avec deux champs au moyen de la commande `ADD PRIMARY KEY(id_client,nom)`.

- Repère 4 : création d'un index pour ce champ. Par exemple, le code pour créer un index sur le champ `adresse` est le suivant :

```
ALTER TABLE 'client' ADD INDEX ('adresse' )
```

Le code pour supprimer cet index est le suivant :

```
ALTER TABLE 'client' DROP INDEX 'adresse'
```

- Repère 5 : définition du champ comme étant à valeur unique. Par exemple, le code pour attribuer l'option `UNIQUE` au champ `mail` est le suivant :

```
ALTER TABLE 'client' ADD UNIQUE ('mail')
```

Le code pour supprimer cette option est le suivant :

```
ALTER TABLE 'client' DROP INDEX 'mail'
```

- Repère 6 : création d'un index sur le texte entier du champ (sauf s'il est numérique). Par exemple, le code pour créer un index sur le champ `adresse` est le suivant :

```
ALTER TABLE 'client' ADD FULLTEXT ('adresse' )
```

Le code pour supprimer l'index est le suivant :

```
ALTER TABLE 'client' DROP INDEX 'adresse'
```

## Suppression ou renommage d'une table

Pour supprimer une table, il suffit de cliquer sur l'onglet Opérations, puis sur la zone Supprimer les données ou la table après avoir sélectionné la base, puis la table. Le code de suppression de la table est le suivant :

```
DROP TABLE 'client'
```

La commande SQL complète de suppression d'une ou de plusieurs tables simultanément est le suivant :

```
| DROP TABLE [IF EXISTS] nom_table1 [, nom_table2,...]
```

L'option `IF EXISTS` évite de provoquer une erreur au cas où vous tentez d'effacer une table qui n'existe pas.

La commande suivante permet de renommer une ou plusieurs tables :

```
| RENAME TABLE ex_nom_table1 TO new_nom_table1 [, ex_nom_table2 TO new_nom_table2]
```

Par exemple, le code pour renommer la table `client` en `clientbis` est le suivant :

```
| RENAME TABLE client TO clientbis
```

## Exportation d'une table

Une fonctionnalité très intéressante offerte par phpMyAdmin est la possibilité d'exporter dans un fichier tout le code SQL de création d'une table. Ce fichier, dont l'extension est `.sql`, permet par exemple, en phase de test, de créer rapidement sur un serveur distant les mêmes tables que celles créées sur le serveur local.

Il est en outre possible d'exporter les données insérées dans la table locale en même temps que la structure de la table et de les retrouver telles quelles dans la base de l'hébergeur.

Pour exporter une table et ses données, il suffit de cliquer sur le bouton Exporter après avoir choisi la base et la table concernées. Dans la page qui s'affiche (voir figure 14-9), conservez les options par défaut qui permettent d'exporter la structure et les données de la table dans un fichier texte `.sql`. Pour exporter uniquement la structure ou les données, ou une partie seulement de celles-ci, choisissez l'option Personnalisée qui permet d'effectuer de nombreux choix.

Le contenu du fichier `article.sql` obtenu est le suivant avec l'option Rapide :

```
-- phpMyAdmin SQL Dump
-- version 3.5.1
-- http://www.phpmyadmin.net
--
-- Client: localhost
-- Généré le: Jeu 27 Décembre 2012 à 11:36
-- Version du serveur: 5.5.16-log
-- Version de PHP: 5.4.3
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
--
-- Base de données: 'magasin'
--
-----
--Structure de la table 'article'
--
CREATE TABLE IF NOT EXISTS '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','video','informatique','divers') COLLATE
latin1_bin NOT NULL,
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', 'video'),
('CS330', 'Caméscope Sony DCR-PC330', '1629.00', 'video'),
('DEL30', 'Portable Dell X300', '1715.00', 'informatique'),
('DVD75', 'DVD vierge par 3', '17.50', 'divers'),
('HP497', 'PC Bureau HP497 écran TFT', '1500.00', 'informatique'),
('NIK55', 'Nikon F55+zoom 28/80', '269.00', 'photo'),
('NIK80', 'Nikon F80', '479.00', 'photo'),
('SAX15', 'Portable Samsung X15 XVM', '1999.00', 'informatique'),
('SOXMP', 'PC Portable Sony Z1-XMP', '2399.00', 'informatique');

```

Le fichier précédent contient la structure et les données pour remplir la table après sa création.

**Figure 14-9**  
*Page d'exportation des tables*

Pour réutiliser ce fichier sur un serveur distant, il suffit de se connecter au site puis, dans le phpMyAdmin du serveur, de sélectionner la base et de cliquer sur l'onglet Importer (repère ❶ de la figure 14-10). Dans la page qui s'affiche, cliquez sur le bouton Parcourir (repère ❷) pour retrouver le fichier `article.sql` sur le disque du poste. Après un clic sur le bouton Exécuter en bas de page, le serveur exécute le code SQL contenu dans le fichier et crée la table correspondante sur le serveur distant. Vous pouvez aussi effectuer un copier-coller dans la fenêtre des requêtes de l'onglet SQL.

Fichier à importer :

Le fichier peut être comprimé (gzip, zip) ou non.  
Le nom du fichier comprimé doit se terminer par **.[format].[compression]**. Exemple: **.sql.zip**

Parcourir : "C:\wamp\www\CH14\arti" Choisir... (Taille maximum: 2 048Kio)

Jeu de caractères du fichier : utf-8

**Figure 14-10**  
*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 14-11.

Colonne	Type	Fonction	Null	Valeur
id_article	char(5)			CS110
designation	varchar(100)			Caméscope Sony 110
prix	decimal(8,2)			1250.50
categorie	enum	-		video

**Exécuter**

**Figure 14-11**  
*Insertion de données ligne par ligne*

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

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

La syntaxe générale de la commande `INSERT` se décline sous trois formes.

Dans une première forme, le nom des colonnes est facultatif si les valeurs sont insérées dans le même ordre que celui de la table :

```
INSERT [DELAYED | LOW_PRIORITY] [IGNORE]
INTO table [col1,col2,...]
VALUES (val1,val2,...)
```

Le mot-clé `DELAYED` indique que les données ne seront insérées qu'après exécution des autres commandes SQL. Le mot-clé `LOW_PRIORITY` indique au serveur d'attendre qu'il n'y ait plus aucun client connecté pour insérer les données. Le mot-clé `IGNORE` permet de ne pas insérer de lignes qui existent déjà.

La deuxième forme n'est qu'une variante de la première, dans laquelle vous définissez explicitement la valeur de chaque colonne. Elle est de préférence employée pour n'insérer des valeurs que dans certaines colonnes :

```
INSERT [DELAYED | LOW_PRIORITY] [IGNORE]
INTO table
SET col1= val1,col2= val2,...
```

La dernière forme permet d'insérer des données à partir du résultat d'une sélection opérée dans une autre table. Vous pourriez, par exemple, créer une table ne contenant que le nom et l'adresse e-mail des clients et la remplir à l'aide de cette commande à partir de la table client :

```
INSERT [DELAYED | LOW_PRIORITY] [IGNORE]
INTO table [col1,col2,...]
SELECT expression
```

### ***INSERT et REPLACE***

Vous pouvez utiliser la commande `REPLACE` à la place de `INSERT` avec la même syntaxe. Elle sert à modifier les valeurs d'une ligne sans modifier son identifiant.

## ***Mise à jour des données***

Lors du cycle de vie d'une base de données, il peut être indispensable de mettre à jour certaines données sans avoir à supprimer une ligne complète puis réécrire les nouvelles informations.

La première solution est bien sûr d'utiliser phpMyAdmin, de sélectionner la base, d'afficher la table et de cliquer sur l'icône `Modifier`, ce qui entraîne l'affichage d'une page de saisie. Vous pouvez alors modifier une ou plusieurs valeurs de la ligne

concernée.

Pour réaliser cette opération avec une requête SQL, vous disposez de la commande UPDATE, dont la syntaxe générale est la suivante :

```
UPDATE[LOW_PRIORITY] [IGNORE] nom_table  
SET colonne1 = valeur1,colonne2=valeur2,...  
[WHERE condition  
[LIMIT N]
```

La condition qui suit la commande WHERE peut permettre d'opérer une mise à jour uniquement pour les lignes répondant à une condition particulière. La clause LIMIT permet de limiter la mise à jour aux  $N$  premières lignes.

Par exemple, pour modifier la date d'une commande dans la table commande de la base magasin, si l'identifiant de commande a la valeur 2 et l'identifiant de client vaut 9, vous écrivez la requête suivante :

```
UPDATE 'magasin'.'commande' SET 'date' = '2013-01-25' WHERE 'commande'.'id_comm'  
=2 AND 'commande'.'id_client' =9;
```

## Importation à partir d'un fichier texte

L'insertion ligne par ligne devient rapidement rébarbative lorsqu'il s'agit de saisir un grand nombre de données. Si ces données sont récupérables à partir d'un catalogue, il est possible de les inclure en une seule opération à condition qu'elles soient formatées selon les mêmes critères que ceux que vous avez utilisés pour les fichiers texte (voir le chapitre 11).

Chaque donnée doit être délimitée par un caractère particulier (par défaut des guillemets) et séparée des autres par un autre caractère (par défaut le point-virgule). Chaque groupe de données correspondant à une ligne de la table doit être délimité par un troisième caractère (par défaut la séquence \n).

Pour importer une liste de données dans la table, vous devriez, par exemple, créer le fichier article\_texte.txt dont le contenu visualisé dans le Bloc-notes de Windows est illustré à la figure 14-12.

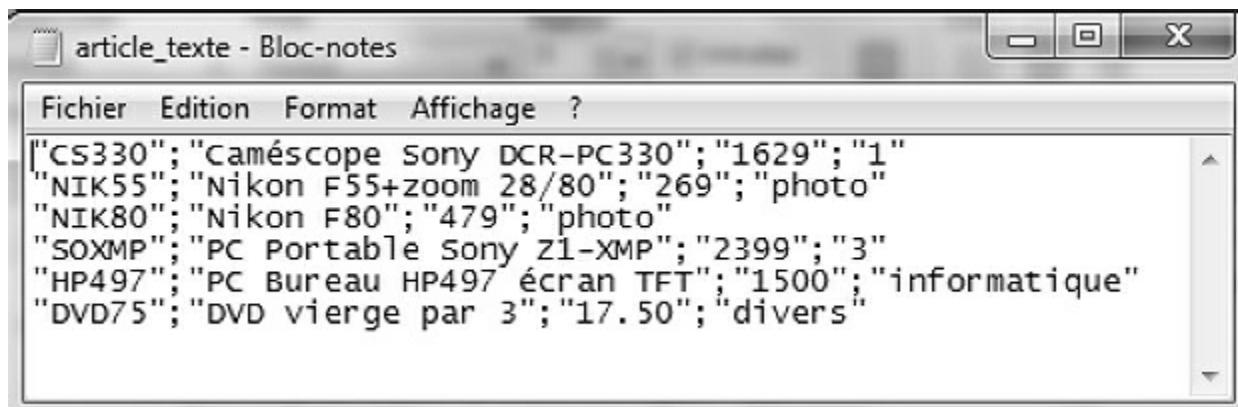


Figure 14-12

Le fichier texte visualisé dans le Bloc-notes

Pour réaliser ce type d'importation, vous devez sélectionner la base, puis la table et cliquer sur l'onglet Importer. Vous obtenez alors une page identique à celle de la figure 14-13.

The screenshot shows the MySQL Importer interface. At the top, there are several tabs: Afficher, Structure, SQL, Rechercher, Insérer, Exporter, Importer (which is selected), Opérations, and Déclencheurs. A checked checkbox says: "Permettre l'interruption de l'importation si la limite de temps configurée dans PHP est sur le point d'être atteinte. (Ceci pourrait aider à importer des fichiers volumineux, au détriment du respect des transactions.)". Below this is a text input field for "Nombre de lignes à ignorer à partir de la première ligne:" with the value "0".

**Format:** CSV via LOAD DATA

**Options spécifiques au format:**

- Remplacer les données de la table avec le fichier
- Ne pas arrêter l'importation lors d'une erreur d'énoncé INSERT

Colonnes terminées par: ;

Colonnes entourées par: "

Caractère d'échappement: \

Lignes terminées par: \n

Nom des colonnes: (empty)

Utiliser l'option LOCAL

**Exécuter**

**Figure 14-13**

*Formulaire d'insertion à partir d'un fichier texte*

Si le fichier est situé sur le poste client, vous pouvez utiliser le bouton Choisir (comme précédemment) pour le localiser et choisir CSV via LOAD DATA dans la liste déroulante Format, puis spécifier le caractère de fin de ligne (ici "\n") dans la case Lignes terminées par.

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 '\n'
```

Vous avez la possibilité de définir vos propres valeurs pour les options suivantes.

- Colonnes terminées par, qui correspond à l'option SQL FIELDS TERMINATED BY.
- Colonnes entourées par, qui correspond à l'option SQL ENCLOSED BY.
- Caractère d'échappement, qui correspond à l'option SQL ESCAPED BY qui permet de choisir le caractère d'échappement pour les caractères spéciaux ayant déjà une signification par ailleurs.

- Lignes terminées par, qui correspond à l'option SQL `LINE TERMINATED BY`.

Il est généralement préférable de conserver les paramètres par défaut, excepté pour le caractère de fin de colonne.

En cliquant sur le bouton Afficher, vous pouvez vérifier que l'insertion des données s'est effectuée correctement et en conformité avec le fichier texte. Vous obtenez l'ensemble de la table, dans la limite de 30 lignes à la fois, comme illustré à la figure 14-14.

<code>id_article</code>	<code>designation</code>	<code>prix</code>	<code>categorie</code>
CA300	Canon EOS 3000V zoom 28/80	329.00	photo
CAS07	Cassette DV60 par 5	26.90	divers
CP100	Cam&eacute;scope Panasonic SV-AV 100	1490.00	video
CS330	Caméscope Sony DCR-PC330	1629.00	
DEL30	Portable Dell X300	1715.00	informatique
DVD75	DVD vierge par 3	17.50	divers
HP497	PC Bureau HP497 écran TFT	1500.00	
NIK80	Nikon F80	479.00	
SAX15	Portable Samsung X15 XVM	1999.00	informatique
SOXMP	PC Portable Sony Z1-XMP	2399.00	informatique

**Figure 14-14**  
*La table articles après insertion de données*

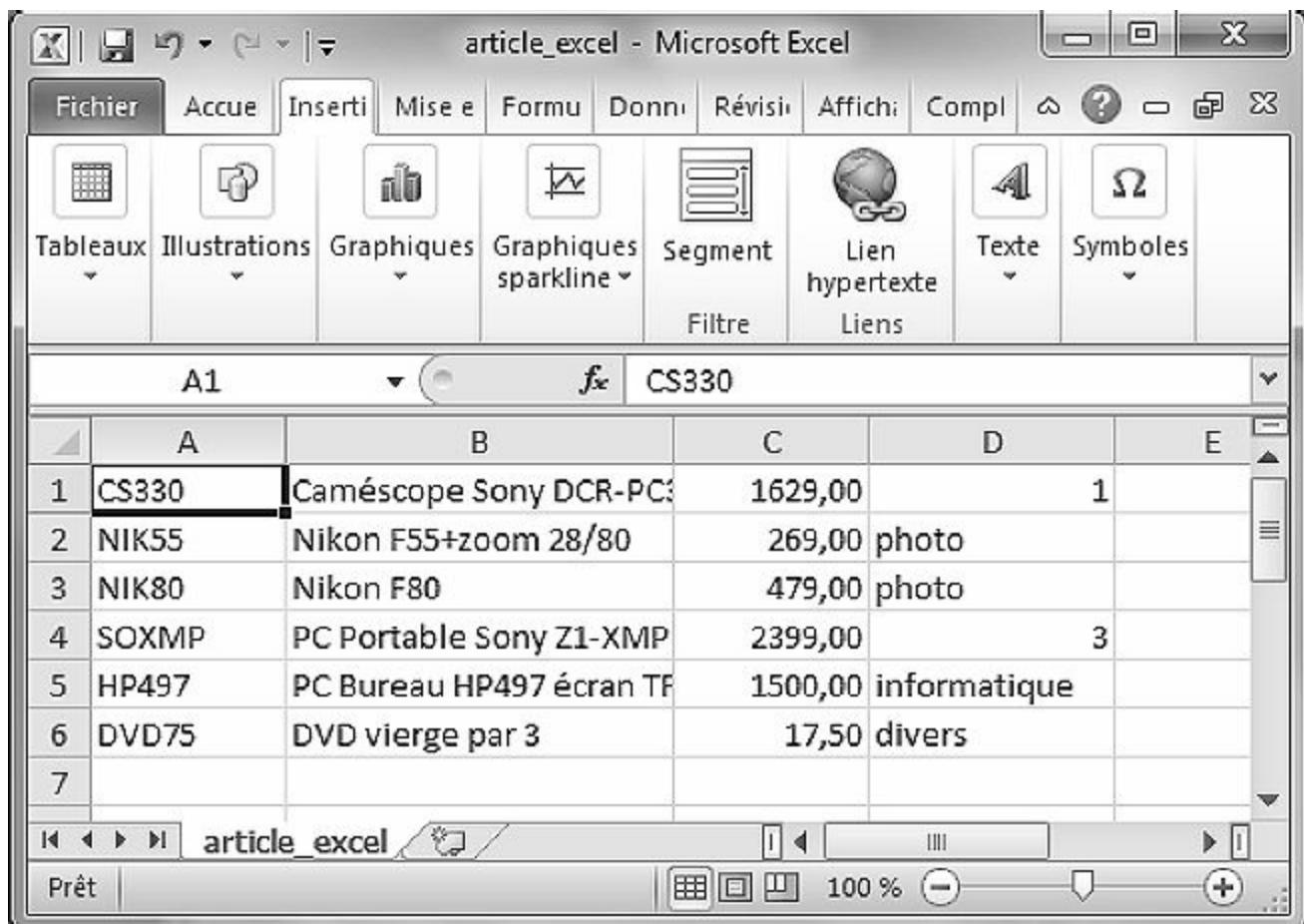
Si vous souhaitez insérer des données dans une table dont la clé primaire est un entier ayant l'option `AUTO_INCREMENT` sélectionnée, le fichier texte doit contenir à la place de cette clé la valeur `NULL` représentée par la chaîne "`\N`". Pour insérer une liste de clients dans la table `client`, par exemple, vous utiliseriez un fichier texte ayant la structure suivante :

```
"\N";"Marti";"Jean";"36";"5 av Einstein";"Orléans";marti@marti.com
"\N";"Rapp";"Paul";"44";"8 rue du Port";"Paris";rapp@libert.com
```

## ***Insertion à partir d'un fichier Excel***

La figure 14-15 représente une feuille de tableur Excel contenant un ensemble de données présentées selon l'ordre des colonnes de la table `article`. Pour les insérer dans la table, vous devez enregistrer la feuille au format CSV dans Excel, puis choisir le type de fichier CSV ( séparateur : point-virgule). Vous obtenez un fichier avec l'extension `.csv`, dont la structure est similaire à celle d'un fichier texte, le séparateur choisi entre chaque champ étant `,`.

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



The screenshot shows a Microsoft Excel window titled "article\_excel - Microsoft Excel". The ribbon menu is visible at the top, with the "Insertion" tab selected. Below the ribbon, there are several icons for inserting various types of content: Tableaux, Illustrations, Graphiques, Graphiques sparkline, Segment, Lien hypertexte, Texte, and Symboles. The main area of the window displays a table with data in columns A through E. Column A contains row numbers (1 to 7). Column B contains article codes (CS330, NIK55, NIK80, SOXMP, HP497, DVD75, and an empty cell). Column C contains descriptions and prices (Caméscope Sony DCR-PC3, Nikon F55+zoom 28/80, Nikon F80, PC Portable Sony Z1-XMP, PC Bureau HP497 écran TFT, DVD vierge par 3). Column D contains unit prices (1629,00, 269,00, 479,00, 2399,00, 1500,00, 17,50). Column E contains quantities (1, photo, photo, 3, informatique, divers). The status bar at the bottom shows the file name "article\_excel" and zoom level "100 %".

	A	B	C	D	E
1	CS330	Caméscope Sony DCR-PC3	1629,00	1	
2	NIK55	Nikon F55+zoom 28/80	269,00	photo	
3	NIK80	Nikon F80	479,00	photo	
4	SOXMP	PC Portable Sony Z1-XMP	2399,00		3
5	HP497	PC Bureau HP497 écran TFT	1500,00	informatique	
6	DVD75	DVD vierge par 3	17,50	divers	
7					

**Figure 14-15**  
*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`**

<b>id_client</b>	<b>nom</b>	<b>prenom</b>	<b>age</b>	<b>adresse</b>	<b>ville</b>	<b>mail</b>
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 14-5 – Données de la table *article***

<b>id_article</b>	<b>designation</b>	<b>prix</b>	<b>categorie</b>
CS330	Caméscope Sony DCR-PC330	1629.00	vidéo
NIK55	Nikon F55+zoom 28/80	269.00	photo
NIK80	Nikon F80	479.00	photo
SOXMP	PC portable Sony Z1-XMP	2399.00	informatique
HP497	PC bureau HP497 écran TFT	1500.00	informatique
DVD75	DVD vierge par 3	17.50	divers
CAS07	Cassette DV60 par 5	26.90	divers
DEL30	Portable Dell X300	1715.00	informatique
CP100	Caméscope Panasonic SV-AV100	1490.00	vidéo
SAX15	Portable Samsung X15 XVM	1999.00	informatique
CA300	Canon EOS 3000V zoom 28/80	329.00	photo

**Tableau 14-6 – Données de la table *commande***

<b>id_comm</b>	<b>id_client</b>	<b>date</b>
1	5	2012-06-11
2	9	2012-06-25
3	1	2012-07-12
4	3	2012-07-14
5	9	2012-07-31
6	10	2012-08-08
7	2	2012-08-25
8	7	2012-09-04
9	11	2012-10-15

10	4	2012-11-23
11	8	2013-01-21
12	5	2013-02-01
13	9	2013-03-03

**Tableau 14-7 – Données de la table *ligne***

<b>id_comm</b>	<b>id_article</b>	<b>quantite</b>	<b>prixunit</b>
1	CS330	1	1629.00
1	CAS07	3	26.90
2	HP497	2	1500.00
3	NIK80	5	479.00
4	SOXMP	3	2399.00
4	DVD75	2	17.50
5	CA300	1	329.00
6	CAS07	3	26.90
6	CP100	1	1490.00
7	SAX15	5	1999.00
8	SOXMP	1	2399.00
8	CP100	1	1490.00
9	NIK55	1	269.00
10	DEL30	2	1715.00
10	SAX15	1	1999.00
11	DVD75	10	17.50
12	CS330	3	1629.00
12	CAS07	4	26.90
13	SAX15	2	1999.00

## Sélection des données

L'opération la plus importante pour créer une page dynamique en réponse à une demande formulée par un visiteur est la sélection des données qui vont composer cette page. Il est pour cela nécessaire de bien maîtriser les commandes SQL de sélection des données et de s'assurer que l'opération de sélection retenue fournit bien les résultats attendus et uniquement ceux-ci.

La sélection des données peut porter sur une table (sélection simple) ou sur deux ou plusieurs tables simultanément (il s'agit alors d'une jointure). Pour illustrer l'importance du choix des commandes SQL de sélection, vous n'utiliserez pas les possibilités d'automatisation du code SQL offertes par phpMyAdmin mais écrirez à la

main chacune des requêtes à exécuter.

Commencez par choisir la base `magasin`, puis la table à interroger et cliquez sur l'onglet SQL. Vous obtenez la page illustrée à la figure 14-16. Vous saisissez vos requêtes dans la zone de saisie multiligne (repère 1). La liste des attributs de la table apparaît dans une liste déroulante à droite (repère 2) afin d'éviter les erreurs dans la saisie des attributs. Il faut ensuite lancer l'exécution de la requête (repère 3).



**Figure 14-16**  
*La page de saisie des requêtes*

## Sélection dans une table

La commande SQL essentielle pour opérer des sélections de lignes dans une table est `SELECT`. Elle comporte de nombreuses options en fonction du résultat souhaité. Les sections qui suivent détaillent les plus importantes d'entre elles.

### Sélection de toutes les lignes

Pour opérer une sélection de toutes les lignes et de toutes les colonnes d'une table, écrivez :

```
| SELECT * FROM client
```

Le seul intérêt de cette requête est de visualiser l'ensemble des données. Elle a le même effet que le bouton Afficher de phpMyAdmin.

Pour restreindre l'affichage à certaines colonnes, vous pouvez les énumérer.

La commande suivante :

```
| SELECT nom, prenom, ville FROM client
```

n'affiche que les trois colonnes indiquées avec leurs valeurs pour toutes les lignes de la

table client.

## Alias

Dans les requêtes de sélection, vous pouvez définir des alias pour les noms des bases, des tables ou des colonnes. Lors de l'affichage, les noms affichés sont ceux des alias et non ceux des colonnes de la table.

Ces exemples affichent toutes les lignes de la table, y compris si cette dernière contient des doublons. Pour ne pas afficher plusieurs fois les mêmes données, vous pouvez utiliser la clause DISTINCT.

L'exemple suivant :

```
| SELECT DISTINCT ville FROM client
```

affiche la liste des villes sans doublon, comme ci-dessous :

ville	Orléans	Paris	Lille	Chartres	Versailles	Lyon	Nantes
-------	---------	-------	-------	----------	------------	------	--------

Vous pouvez trier les champs affichés par ordre croissant ou décroissant en utilisant la clause ORDER BY du nom du champ et des options ASC (croissant) ou DESC.

La requête suivante :

```
| SELECT DISTINCT ville FROM client ORDER BY ville ASC
```

affiche la liste ordonnée des villes, comme ci-dessous :

ville	Chartres	Lille	Lyon	Nantes	Orléans	Paris	Versailles
-------	----------	-------	------	--------	---------	-------	------------

La requête suivante :

```
| SELECT DISTINCT ville FROM client ORDER BY ville DESC
```

affiche la même liste en ordre inverse, comme ci-dessous :

ville	Versailles	Paris	Orléans	Nantes	Lyon	Lille	Chartres
-------	------------	-------	---------	--------	------	-------	----------

## Restriction du nombre de lignes

La possibilité de restreindre le nombre de lignes illustre tout l'intérêt des opérations de sélection. Elle permet aux données auxquelles est appliquée la restriction de répondre à une ou plusieurs conditions particulières. Pour appliquer une restriction, il suffit d'ajouter la clause WHERE à la commande SELECT.

La syntaxe générale de la commande SELECT devient :

```
| SELECT col1,col2,... FROM table WHERE condition
```

La condition peut porter sur n'importe quelle colonne. Elle doit être rédigée à l'aide

d'opérateurs et de fonctions, à la manière des expressions conditionnelles des instructions `if`.

Par exemple, pour sélectionner les clients qui habitent Paris, vous écrivez :

```
SELECT nom, prenom, adresse, mail FROM client WHERE ville='Paris'
```

Vous obtenez la liste illustrée à la figure 14-17.

nom	prenom	adresse	mail
Rapp	Paul	32 Av Foch	rapp@libert.com
Marti	Pierre	4 Av Henri	martin7@tiscali.fr
Darc	Jeanne	9 Av d'Orléans	NULL

**Figure 14-17**  
Sélection avec une clause WHERE

## Les opérateurs de comparaison

Les opérateurs de comparaison sont applicables à tous les types de données. Ils donnent pour résultats les valeurs `TRUE`, `FALSE` ou `NULL`. Les opérandes sont convertis en nombre ou en chaîne selon le contexte.

Pour les chaînes, la comparaison est insensible à la casse et ignore les espaces, les tabulations (`\t`) et les sauts de ligne (`\n`).

**Tableau 14-8 – Les opérateurs de comparaison**

Opérateur	Description
=	Égalité des nombres ou des chaînes de caractères
<> ou !=	Différent de
<=	Inférieur ou égal à
<	Inférieur à
>=	Supérieur ou égal à
>	Supérieur à
IS NULL	Retourne <code>TRUE</code> si la valeur est <code>NULL</code> . Par exemple : <code>SELECT nom, prenom FROM auteurs WHERE prenom IS NULL</code>
IS NOT NULL	Retourne <code>TRUE</code> si la valeur n'est pas <code>NULL</code> . Par exemple : <code>SELECT nom, prenom FROM auteurs WHERE prenom IS NOT NULL</code>
express BETWEEN min AND max	Retourne <code>TRUE</code> si la valeur est comprise entre <code>min</code> et <code>max</code> (limites comprises). Par exemple, le code suivant sélectionne tous les noms dont l'initiale est comprise entre A et E : <code>SELECT nom FROM client WHERE nom BETWEEN 'A' and 'E'</code>
express NOT BETWEEN min AND max	Retourne <code>TRUE</code> si la valeur n'est pas comprise entre <code>min</code> et <code>max</code> (limites comprises). Par exemple, le code suivant sélectionne tous les noms dont l'initiale n'est pas comprise entre A et E : <code>SELECT nom FROM client WHERE nom NOT BETWEEN 'A' and 'E'</code>

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.

`express LIKE 'motif'`

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`

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
```

## Exemples de requêtes avec des opérateurs

- Pour sélectionner les clients dont l'âge est supérieur à 40 ans :

```
SELECT nom, prenom, age, adresse, ville  
FROM client WHERE age > 40
```

nom	prenom	age	adresse	ville
Rapp	Paul	44	32 Av Foch	Paris
Hachette	Jeanne	45	60 rue d'Amiens	Versailles
Mac Neal	John	52	89 rue Diana	Lyon
Gaté	Bill	45	9 Bd des Bugs	Lyon

- Pour sélectionner les clients qui n'ont pas d'adresse e-mail :

```
SELECT nom, prenom, age, adresse, ville
FROM client WHERE mail IS NULL
```

nom	prenom	age	adresse	ville
Hachette	Jeanne	45	60 rue d'Amiens	Versailles
Darc	Jeanne	19	9 Av d'Orléans	Paris

Pour sélectionner les clients dont les noms commencent par une lettre entre A et H, la lettre H n'étant pas prise en compte :

```
SELECT nom, prenom, age, ville
FROM client WHERE nom BETWEEN 'A' AND 'H'
```

nom	prenom	age	ville
Devos	Marie	18	Lille
Grave	Nuyen	18	Lille
Basile	Did	37	Nantes
Darc	Jeanne	19	Paris
Gaté	Bill	45	Lyon

Pour sélectionner les articles dont le prix est compris entre 1 000 et 2 000 euros :

```
SELECT id_article, designation, prix FROM article WHERE prix BETWEEN 1500 AND 2000
```

id_article	designation	prix
CS330	Caméscope Sony DCR-PC330	1629.00
HP497	PC Bureau HP497 écran TFT	1500.00
DEL30	Portable Dell X300	1715.00
SAX15	Portable Samsung X15 XVM	1999.00

Pour sélectionner les clients dont l'âge figure dans la liste (18, 19, 20) :

```
SELECT nom, prenom, age, ville FROM client WHERE age IN ( 18, 19, 20 )
```

nom	prenom	age	ville
Devos	Marie	18	Lille
Grave	Nuyen	18	Lille
Darc	Jeanne	19	Paris

Pour sélectionner les clients qui habitent une avenue, et dont l'adresse contient donc la chaîne "Av" (sensible à la casse) :

```
| SELECT nom, prenom, age, adresse, ville FROM client WHERE adresse LIKE '%Av%'
```

nom	prenom	age	adresse	ville
Marti	Jean	36	5 Av Einstein	Orléans
Rapp	Paul	44	32 Av Foch	Paris
Marti	Pierre	25	4 Av Henri	Paris
Darc	Jeanne	19	9 Av d'Orléans	Paris

Pour sélectionner les articles de la marque Sony, dont la désignation contient donc la chaîne sony :

```
| SELECT id_article, designation, prix FROM article WHERE designation LIKE '%Sony%'
```

id_article	designation	prix
CS330	Caméscope Sony DCR-PC330	1629.00
SOXMP	PC Portable Sony Z1-XMP	2399.00

Pour sélectionner les clients dont les noms commencent par Ma à l'aide d'une expression régulière :

```
| SELECT nom, prenom, age, adresse, ville
  FROM client WHERE nom REGEXP 'Ma.*'
```

nom	prenom	age	adresse	ville
Marti	Jean	36	5 Av Einstein	Orléans
Marti	Pierre	25	4 Av Henri	Paris
Mac Neal	John	52	89 rue Diana	Lyon

## Les opérateurs logiques

Il est évidemment possible d'écrire des conditions complexes dans la clause WHERE en reliant plusieurs conditions par des opérateurs logiques. La liste et la description de ces opérateurs est donnée au tableau 14-9.

Une expression contenant un ou deux opérandes reliés par un opérateur logique est évaluée à TRUE (1), FALSE (0) ou NULL.

**Tableau 14-9 – Les opérateurs logiques**

Opérateur	Description
NOT express ou !express	Le NON logique renvoie FALSE si express vaut TRUE et réciproquement ou NULL si express vaut NULL. Par exemple, le code suivant retourne tous les noms différents de 'dupont' : <code>SELECT nom FROM client WHERE NOT (nom='dupont')</code>
express1 AND express2	Le ET logique renvoie TRUE si les deux opérandes sont différents de FALSE et de NULL et renvoie FALSE s'ils sont tous deux FALSE. Dans les autres cas, l'évaluation est NULL. Par exemple, le code suivant retourne tous les âges des clients nommés 'pierre dupont' : <code>SELECT age FROM client WHERE nom='dupont' AND prenom='pierre'</code>
express1 OR express2	Le OU logique renvoie TRUE si au moins un des opérandes vaut TRUE, NULL si l'un des deux est NULL et FALSE dans les autres cas. Par exemple, le code suivant retourne tous les âges des clients dont le nom est 'dupont' ou le prénom 'pierre' : <code>SELECT age FROM client WHERE nom='dupont' OR prenom='pierre'</code>
express1 XOR express	Le OU exclusif logique renvoie NULL si l'un des opérandes vaut NULL, TRUE si un seul des deux vaut TRUE et FALSE sinon. Par exemple, le code suivant retourne les âges des clients dont le nom est 'dupont' et dont le prénom n'est pas 'pierre' ainsi que de ceux dont le prénom est 'pierre' et dont le nom est différent de 'dupont' : <code>SELECT age FROM client WHERE nom='dupont' XOR prenom='pierre'</code>

## Exemples utilisant les opérateurs logiques

Pour sélectionner les clients qui ont moins de 30 ans et qui habitent Paris :

```
SELECT nom, prenom, age, adresse, ville
FROM client WHERE age <30 AND ville = 'Paris'
```

nom	prenom	age	adresse	ville
Marti	Pierre	25	4 Av Henri	Paris
Darc	Jeanne	19	9 Av d'Orléans	Paris

Pour sélectionner les clients qui habitent Lyon ou dont le nom commence par la lettre H :

```
SELECT nom, prenom, age, adresse, ville FROM client
WHERE ville='Lyon' OR nom LIKE 'H%'
```

nom	prenom	age	adresse	ville
Hochon	Paul	22	33 rue Tsétsé	Chartres
Hachette	Jeanne	45	60 rue d'Amiens	Versailles
Mac Neal	John	52	89 rue Diana	Lyon
Gaté	Bill	45	9 Bd des Bugs	Lyon

Pour sélectionner les clients qui n'habitent ni à Paris ni à Lyon :

```
SELECT nom, prenom, age, adresse, ville
FROM client
WHERE NOT(ville='Paris') AND NOT(ville='Lyon')
```

## Les opérateurs arithmétiques

MySQL reconnaît les quatre opérations fondamentales (+, -, \*, /) plus l'opérateur modulo (%). Excepté pour la division, si les opérandes sont entiers, le résultat est de type `BIGINT` 64 bits. Si un seul des opérandes comporte l'option `UNSIGNED`, le résultat a aussi cette propriété `UNSIGNED`. En cas de dépassement de la capacité du type `BIGINT`, le résultat est 0. Le résultat de la division par zéro a la valeur `NULL`, et l'opération ne provoque pas d'erreur visible.

nom	prenom	age	adresse	ville
Marti	Jean	36	5 Av Einstein	Orléans
Devos	Marie	18	75 Bd Hochimin	Lille
Hochon	Paul	22	33 rue Tsétsé	Chartres
Grave	Nuyen	18	75 Bd Hochimin	Lille
Hachette	Jeanne	45	60 rue d'Amiens	Versailles
Basile	Did	37	26 rue Gallas	Nantes

## Les fonctions mathématiques

MySQL gère un grand nombre de fonctions mathématiques classiques, telles que celles que vous avez définies dans la partie spécifique de PHP (voir chapitre 2).

Le tableau 14-10 récapitule les fonctions mathématiques utilisables dans des requêtes.

**Tableau 14-10 – Fonctions mathématiques**

Fonctions	Description
-X	Opposé de X
ABS (X)	Valeur absolue de X
ACOS (X)	Angle en radian dont le cosinus est X.

ASIN (X)	Angle en radian dont le sinus est X.
ATAN (X)	Angle en radian dont la tangente est X.
ATAN2 (X, Y)	Angle en radian dont la tangente est la valeur de X/Y.
CEILING (X)	Arrondi à l'entier supérieur à X
COS (X)	Cosinus de X en radian
COT (X)	Cotangente de X en radian
DEGREES (X)	Convertit X de radian en degré.
EXP (X)	Exponentielle de base e de X
FLOOR (X)	Arrondi à l'entier inférieur à X
GREATEST (X, Y, ...)	Retourne la plus grande valeur de la liste. Utilisable avec des nombres et des lettres.
LEAST (X, Y, ...)	Retourne la plus petite valeur de la liste. Utilisable avec des nombres et des lettres.
LOG (X)	Logarithme népérien de X
LOG10 (X)	Logarithme de base 10 de X
MOD (X, N)	Modulo : reste de la division de X par N (équivalent de $X \% N$ )
PI ()	Valeur de pi
POW <sub>X</sub> , Y) ou POWER (X, Y)	Nombre X à la puissance Y (X et Y peuvent être décimaux).
RADIANS (X)	Convertit X de degré en radian.
RAND () ou RAND (N)	Retourne un nombre aléatoire compris entre 0 et 1. Si le paramètre N est précisé, ce nombre est utilisé pour initialiser le générateur aléatoire. À chaque valeur de N correspond un nombre unique toujours identique.
ROUND (X)	Arrondit à l'entier le plus proche.
SIGN (X)	Retourne -1, 0 ou 1 selon que X est négatif, nul ou positif.
SIN (X)	Sinus de X exprimé en radian
SQRT (X)	Racine carrée de X, qui doit être positive.
TAN (X)	Tangente de X exprimée en radian
TRUNCATE (X, N)	Arrondit X avec N décimales.

## 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 (*)	Retourne le nombre de lignes dont la valeur n'est pas <code>NULL</code> dans la

COUNT(colonne)	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.

Pour déterminer l'âge du client le plus jeune :

```
| SELECT MIN(age) FROM client
```

Vous obtenez la valeur 18.

Pour calculer le nombre total d'articles commandés :

```
| SELECT SUM(quantite) FROM ligne
```

Vous obtenez la valeur 51.

## Le regroupement

La clause GROUP BY utilisée dans une commande SELECT permet de regrouper des lignes ayant une caractéristique commune. Vous obtenez de la sorte des sous-ensembles de

lignes de la table auxquels vous pouvez appliquer des opérations telles que les fonctions statistiques détaillées précédemment. Les calculs sont effectués sur ces sous-ensembles.

Vous pouvez, par exemple, regrouper les clients par ville en écrivant la clause 'GROUP BY ville'. Au lieu de calculer l'âge moyen de la totalité des clients, vous pouvez ainsi calculer l'âge moyen des clients par ville.

La requête suivante :

```
| SELECT AVG( age ) AS 'age moyen', ville FROM client GROUP BY ville
```

affiche le résultat ci-dessous, dans lequel l'utilisation d'un alias pour la colonne AVG(age) donne une information plus lisible en remplaçant le nom de la colonne age par le texte age moyen.

age moyen	ville
22.0000	Chartres
18.0000	Lille
48.5000	Lyon
37.0000	Nantes
36.0000	Orléans
29.3333	Paris
45.0000	Versailles

Les exemples précédents d'utilisation de fonctions statistiques n'utilisaient pas la clause GROUP BY. Il n'était donc pas possible de sélectionner plusieurs colonnes dans une même requête. Avec cette clause, vous pouvez regrouper les lignes par ville et afficher dans un même tableau l'âge minimal, l'âge moyen et l'âge maximal des clients.

Par exemple, la requête suivante :

```
| SELECT MIN(age) AS 'Age minimum', AVG(age) AS 'Age moyen', MAX(age) AS 'Age maximum',  
| ville  
| FROM client  
| GROUP BY ville
```

retourne le tableau illustré à la figure 14-18.

Age minimum	Age moyen	Age maximum	ville
22	22.0000	22	Chartres
18	18.0000	18	Lille
45	48.5000	52	Lyon
37	37.0000	37	Nantes
36	36.0000	36	Orléans
19	29.3333	44	Paris
45	45.0000	45	Versailles

**Figure 14-18**  
*Calculs statistiques sur les âges*

### Exemple

Pour afficher le montant total et le nombre d'articles de chaque commande, vous allez utiliser la colonne `id_comm` comme critère de regroupement. En appliquant la fonction `SUM()`, vous pouvez calculer le prix de chaque ligne puis le total :

```
SELECT id_comm AS 'Numéro de commande',
SUM( prix_unit * quantite ) AS 'Prix Total', SUM(quantite) AS 'Nombre d\'articles'
FROM ligne
GROUP BY id_comm
```

La requête affiche le résultat illustré à la figure 14-19.

Numéro de commande	Prix Total	Nombre d'articles
1	1709.70	4
2	3000.00	2
3	2395.00	5
4	7232.00	5
5	329.00	1
6	1570.70	4
7	9995.00	5
8	3889.00	2
9	269.00	1
10	5429.00	3
11	175.00	10
12	4994.60	7
13	3998.00	2

**Figure 14-19**

## *Calcul du prix total par commande*

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 suivant :

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.

La requête suivante :

```
SELECT commande.id_comm,nom,prenom,ville  
FROM commande,client  
WHERE client.id_client = commande.id_client ORDER BY nom
```

affiche le résultat illustré à la figure 14-20 (la clause `ORDER BY` trie les noms par ordre alphabétique).

La condition `WHERE` peut comporter d'autres composantes permettant d'opérer une sélection plus précise.

<code>id_comm</code>	<code>nom</code>	<code>prenom</code>	<code>ville</code>
2	Basile	Did	Nantes
5	Basile	Did	Nantes
13	Basile	Did	Nantes
6	Darc	Jeanne	Paris
4	Devos	Marie	Lille
9	Gaté	Bill	Lyon
12	Grave	Nuyen	Lille
1	Grave	Nuyen	Lille
10	Hochon	Paul	Chartres
11	Mac Neal	John	Lyon
3	Marti	Jean	Orléans
8	Marti	Pierre	Paris
7	Rapp	Paul	Paris

**Figure 14-20**

*Jointure sur les tables client et commande*

Par exemple, pour sélectionner les commandes du client dont l'identifiant est 5, vous ajoutez la condition suivante à la clause `WHERE` :

```
SELECT commande.id_comm, client.id_client, commande.date  
FROM commande,client  
WHERE client.id_client = commande.id_client AND client.id_client =5
```

Cette requête affiche le résultat ci-dessous :

<b>id_comm</b>	<b>id_client</b>	<b>date</b>
1	5	2012-06-11
12	5	2013-02-01

Il est possible d'utiliser la clause `GROUP BY` dans une jointure pour effectuer des calculs statistiques sur un groupe de données.

Dans l'exemple suivant, vous cherchez à établir la liste des articles les plus vendus sur le site et à afficher leur code, leur désignation, leur prix et le nombre total d'articles. Vous effectuez pour cela une jointure sur les tables `article` et `ligne` en utilisant comme critère de jointure leur colonne commune `id_article`. Vous appliquez ensuite à la colonne `id_article` la clause `GROUP BY` pour obtenir la somme des ventes de chaque article et la clause `ORDER BY` pour afficher les résultats en ordre décroissant :

```
SELECT article.id_article, designation, prix, SUM( quantite ) AS 'Total'
FROM article, ligne
WHERE article.id_article = ligne.id_article
GROUP BY id_article
ORDER BY Total DESC
```

La requête affiche le résultat illustré à la figure 14-21.

<b>id_article</b>	<b>designation</b>	<b>prix</b>	<b>Total</b>
DVD75	DVD vierge par 3	17.00	12
CAS07	Cassette DV60 par 5	26.90	10
SAX15	Portable Samsung X15 XVM	1999.00	8
NIK80	Nikon F80	479.00	5
CS330	Caméscope Sony DCR-PC330	1629.00	4
SOXMP	PC Portable Sony Z1-XMP	2399.00	4
HP497	PC Bureau HP497 écran TFT	1500.00	2
DEL30	Portable Dell X300	1715.00	2
CP100	Cam&eacute;scope Panasonic SV-AV 100	1490.00	2
CA300	Canon EOS 3000V zoom 28/80	329.00	1
NIK55	Nikon F55+zoom 28/80	269.00	1

**Figure 14-21**  
*Jointure et groupement*

## ***Jointure de plus de deux tables***

Les jointures peuvent porter sur plus de deux tables, à condition que les tables jointes contiennent un même type de donnée dans une de leurs colonnes.

L'exemple suivant est d'une forme plus complexe que les précédents car il opère successivement une jointure sur les tables `commande` et `client` ayant en commun la colonne `id_client` et une jointure sur les tables `commande` et `ligne` ayant en commun la colonne `id_comm`. Ces jointures permettent tout à la fois de sélectionner les numéros des commandes, les nom, prénom et ville du client associé à la commande et le montant total de la commande obtenu et de regrouper les résultats par numéro de commande à l'aide de la clause `GROUP BY`.

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 * prix_unit ) 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 14-22.

<b>id_comm</b>	<b>nom</b>	<b>prenom</b>	<b>ville</b>	<b>Prix total</b>
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	269.00
12	Grave	Nuyen	Lille	4994.60
1	Grave	Nuyen	Lille	1709.70
10	Hochon	Paul	Chartres	5429.00
11	Mac Neal	John	Lyon	175.00
3	Marti	Jean	Orléans	2395.00
8	Marti	Pierre	Paris	3889.00
7	Rapp	Paul	Paris	9995.00

**Figure 14-22**  
*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.

## Exercice 3

Supprimez toutes les tables de la base `voitures`.

## Exercice 4

Recréez les tables de la base `voitures` en utilisant les fichiers SQL précédents.

## Exercice 5

Insérez des données dans la table `proprietaire` de la base `voitures`, puis vérifiez-en la bonne insertion.

## Exercice 6

Créez un fichier texte contenant une liste de modèles de voitures avec autant de données par ligne que de colonnes dans la table `modele` de la base `voitures`. Insérez ces données dans la base.

## Exercice 7

Créez un fichier Excel ou OpenOffice contenant une liste de modèles de voitures avec autant de données par ligne que de colonnes dans la table `modele`. Enregistrez-le au format CSV, et insérez les données dans la base.

## Exercice 8

Insérez des données dans les autres tables de la base `voitures`. Effectuez des mises à jour en modifiant certaines valeurs.

## Exercice 9

Dans la base `magasin`, sélectionnez les articles dont le prix est inférieur à 1 500 euros.

## Exercice 10

Dans la base `magasin`, sélectionnez les articles dont le prix est compris entre 100 et 500 euros.

## **Exercice 11**

Dans la base `magasin`, sélectionnez tous les articles de marque Nikon (dont la désignation contient ce mot).

## **Exercice 12**

Dans la base `magasin`, sélectionnez tous les caméscopes, leur prix et leur référence.

## **Exercice 13**

Dans la base `magasin`, sélectionnez tous les produits de la catégorie `informatique`, et affichez leur code, leur désignation et leur prix par ordre décroissant de prix.

## **Exercice 14**

Dans la base `magasin`, sélectionnez tous les clients de moins de 40 ans, et ordonnez les résultats par ville en ordre alphabétique.

## **Exercice 15**

Dans la base `magasin`, calculez le prix moyen de tous les articles.

## **Exercice 16**

Dans la base `magasin`, calculez le nombre d'e-mails non `NULL` et distincts l'un de l'autre.

## **Exercice 17**

Dans la base `magasin`, affichez les coordonnées des clients ayant la même adresse (même adresse et même ville).

## **Exercice 18**

Dans la base `magasin`, sélectionnez tous les articles commandés par chaque client.

## **Exercice 19**

Dans la base `magasin`, sélectionnez tous les clients dont le montant d'une commande dépasse 1 500 euros.

## **Exercice 20**

Dans la base `magasin`, sélectionnez tous les clients dont le montant total de toutes les commandes dépasse 5 000 euros.

## **Exercice 21**

Dans la base `voitures`, sélectionnez tous les véhicules d'une personne donnée.

## **Exercice 22**

Dans la base `voitures`, sélectionnez toutes les personnes ayant le même modèle de voiture.

### **Exercice 23**

Dans la base voitures, sélectionnez tous les véhicules ayant plusieurs copropriétaires.

## Accès objet à MySQL avec PHP

Une tendance évidente qui a prévalu dans le développement de PHP est la programmation objet ; l'accès à MySQL n'y a pas échappé, et cela n'a fait que s'accentuer dans les versions successives de PHP, si bien que l'accès procédural a été supprimé dans PHP 7 et que son utilisation entraîne un avertissement. Cette possibilité est liée à l'utilisation de l'extension `mysqli`, dite « mysql améliorée », qui comprend les trois classes suivantes :

- La classe `mysqli` qui permet de créer des objets de type `mysqli_object`. Cette dernière ne possède pas moins de 33 méthodes et 15 propriétés pouvant permettre la connexion à la base, l'envoi de requêtes, les transactions ou les requêtes préparées.
- La classe `mysqli_result`, permettant de gérer les résultats d'une requête SQL effectuée par l'objet précédent.
- La classe `mysql_stmt` qui représente une requête préparée. Il s'agit là d'une nouveauté par rapport à l'extension `mysql`.

## Connexion au serveur MySQL

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

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

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

Par mesure de sécurité, vous avez tout intérêt à placer les valeurs des paramètres de connexion dans un fichier séparé, en prenant soin d'y définir ces paramètres sous forme de constantes. Vous pouvez ensuite l'inclure dans les scripts de création de connexions utilisant les noms de ces constantes. Le fichier `myparam.inc.php` de l'exemple 15-1 réalise cette opération avec ici les paramètres du serveur local. Vous l'utiliserez

systématiquement par la suite.

## Exemple 15-1. Le fichier myparam.inc.php

```
<?php  
define("HOST","localhost");  
define("USER","root");  
define("PASS","root");  
define("PORT",8889);  
?>
```

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

En cas d'erreur de connexion, nous pouvons récupérer le code d'erreur généré et le message d'erreur associé grâce aux propriétés `connect_errno` et `connect_error`.

La connexion prend fin quand l'exécution du script PHP est terminée, mais on peut y mettre fin explicitement pour observer le serveur MySQL. Si les résultats d'une requête sont entièrement récupérés, la connexion peut en effet être coupée avec la méthode `close()` selon la syntaxe suivante :

```
boolean $idcom->close()
```

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

```
boolean $idcom->select_db (string $base)
```

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

### Sélection de la base via SQL

Vous pouvez aussi sélectionner une base en envoyant la requête "USE nom\_base" au serveur à l'aide de la méthode `query()` détaillée dans les sections suivantes.

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

```
boolean $idcom->ping()
```

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

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

```
<?php  
//Inclusion des paramètres de connexion
```

```

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

```

Comme nous l'avons fait pour l'accès procédural à MySQL, nous avons intérêt à créer une fonction de connexion au serveur que nous réutiliserons systématiquement dans tous les exemples qui suivent. C'est l'objet de l'exemple 15-2 qui crée la fonction `connexobjet()` dont les paramètres sont le nom de la base dans la variable `$base` et le nom du fichier `.inc.php` contenant les paramètres de connexion dans la variable `$param`.

La fonction inclut d'abord les paramètres de connexion (repère 1) puis crée un objet `mysqli` (repère 2) ; elle vérifie ensuite que la connexion est bien réalisée (repère 3), affiche un message d'alerte JavaScript et sort de la fonction en cas de problème (repère 4). Si la connexion est bien réalisée elle retourne l'objet `$idcom` (repère 5).

## Exemple 15-2. Fonction de connexion au serveur

```

<?php
function connexobjet($base,$param)
{
    include_once($param.".inc.php"); ← 1
    $idcom = new mysqli(HOST,USER,PASS,$base, PORT); ← 2
    if (!$idcom) ← 3
    {
        echo "<script type=text/javascript>";
        echo "alert('Connexion impossible à la base')</script>";
        exit(); ← 4
    }
    return $idcom; ← 5
}
?>

```

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

```

include("exemple15.2.php");
$idcom = connexobjet ("nom_base", "myparam");

```

L'opération de connexion est donc gérée en deux lignes, en indiquant simplement le nom de la base.

# Envoi de requêtes SQL au serveur

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

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

```
| divers $idcom->query (string $requete [,int mode ])
```

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

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

## Exemple 15-3. Envoi de requête type

```
<?php
include_once("exemple15.2.php"); ←①
$idcom=connexobjet("magasin","myparam"); ←②
$requete="SELECT * FROM article ORDER BY categorie"; ←③
$result=$idcom->query($requete); ←④
if(!$result)
{
    echo "Lecture impossible"; ←⑤
}
else
{
    // Lecture des résultats ←⑥
    while ($row = $result->fetch_array(MYSQLI_NUM) )
    {
        foreach($row as $d)
        {
            echo $d," ";
        }
        echo "<hr />";
    }
    // Destruction de l'objet $result
    $result->free(); ←⑦
}
// Fermeture de la connexion
$idcom->close(); ←⑧
?>
```

Ce script effectue successivement l'inclusion du fichier `exemple15.2.php` (repère ①), la

connexion au serveur (repère ②), l'écriture de la requête SQL dans la variable `$requete` (repère ③), l'envoi de la requête et la récupération du résultat (repère ④), l'affichage d'un éventuel message d'erreur (repère ⑤) ou bien des résultats (repère ⑥) – procédure que nous détaillerons dans le paragraphe suivant –, et enfin, la libération de la mémoire occupée par l'objet `mysqli_result` (repère ⑦) et la fermeture de la connexion (repère ⑧).

## Lecture du résultat d'une requête

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

Par contre, lorsqu'il s'agit de lire le résultat d'une requête contenant la commande `SELECT`, la méthode `query()` retourne un objet de type `mysqli_result`, identifié dans nos exemples par la variable `$result`. La classe `mysqli_result` offre une grande variété de méthodes permettant de récupérer des données sous des formes diverses, la plus courante étant un tableau. Chacune de ces méthodes ne récupérant qu'une ligne du tableau à la fois, il faut recourir à une ou plusieurs boucles pour lire l'ensemble des données.

### *Lecture à l'aide d'un tableau*

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

```
| array $result->fetch_array (int type)
```

Elle retourne un tableau qui peut être indicé (si la constante `type` vaut `MYSQLI_NUM`), associatif (si `type` vaut `MYSQLI_ASSOC`), dont les clés sont les noms des colonnes ou les alias de la table interrogée, ou encore mixte, contenant à la fois les indices et les clés (si `type` vaut `MYSQLI_BOTH`). Pour lire toutes les lignes du résultat, il faut écrire une boucle (`while` par exemple) qui effectue un nouvel appel de la méthode `fetch_array()` pour chaque ligne. Cette boucle teste s'il existe encore des lignes à lire, la méthode `fetch_array()` retournant la valeur `NULL` quand il n'y en a plus. Pour lire et afficher chaque ligne nous utilisons ensuite une boucle `foreach`. Notez encore une fois que si le tableau retourné est indicé, l'indice 0 correspond au premier attribut écrit dans la requête et ainsi de suite.

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

```
| array $result->fetch_assoc(void)
```

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

```
| array $result->fetch_row(void)
```

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

L'exemple 15-4 met cette méthode en pratique dans le but d'afficher le contenu de la table `article` de la base `magasin` dans un tableau HTML. Après l'inclusion de la fonction de connexion (repère 1) puis son appel sur la base `magasin` (repère 2) nous écrivons la requête SQL de sélection (repère 3). L'envoi de la requête avec la méthode `query()` permet de récupérer un objet `$result` ou `FALSE` en cas d'échec (repère 4). Un test sur la variable `$result` (repère 5) permet d'afficher un message d'erreur (repère 6) ou le contenu de la table (repère 7). Nous récupérons d'abord le nombre d'articles dans la table grâce à la propriété `num_rows` de l'objet `mysqli_result` (repère 8) et affichons ce nombre dans un titre `<h4>` (repère 9). Une boucle `while` permet de lire une ligne à la fois dans un tableau indicé (repère 10), puis une boucle `foreach` permet d'afficher chacune des valeurs du tableau dans un tableau HTML (repère 11). L'objet `$result` est alors supprimé (repère 12) et la connexion fermée (repère 13). La figure 15-1 montre l'affichage obtenu dans un navigateur.

Si l'on veut récupérer toutes les lignes de résultats dans un seul tableau en une seule fois sans effectuer de boucle `while` comme au repère 10 de l'exemple, il est possible d'utiliser la méthode `fetch_all()` dont la syntaxe est :

```
| array $result->fetch_all(inttype)
```

Le paramètre `type` prend comme valeur les mêmes constantes que celles de la méthode `fetch_array()`. S'il y a plusieurs lignes de résultats, le tableau sera donc multidimensionnel et lisible avec deux boucles `foreach` imbriquées. Cette possibilité sera exploitée dans l'exemple 15-5.

## → Exemple 15-4. Lecture de la table article

```
<!DOCTYPE html>
<html 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 ("exemple15.2.php"); ← 1
$idcom=connexobjet ("magasin", "myparam"); ← 2
$requete="SELECT * FROM article ORDER BY categorie"; ← 3
$result=$idcom->query ($requete); ← 4
if (!$result) ← 5
{
    echo "Lecture impossible"; ← 6
}
else ← 7
{
    $nbart=$result->num_rows; ← 8
```

```
echo "<h3>Tous nos articles par catégorie</h3>";
echo "<h4>Il y a $nbart articles en magasin</h4>"; ←⑨
echo "<table border=\"1\">";
    echo "<tr><th>Code      article</th>      <th>Description</th>      <th>Prix</th>
<th>Catégorie</th></tr>";
while($ligne=$result->fetch_array(MYSQLI_NUM)) ←⑩
{
    echo "<tr>";
    foreach($ligne as $valeur) ←⑪
    {
        echo "<td> $valeur </td>";
    }
    echo "</tr>";
}
echo "</table>";
}
$result->free(); ←⑫
$idcom->close(); ←⑬
?>
</body>
</html>
```

The screenshot shows a Firefox browser window with the title "Lecture de la table article - Mozilla Firefox". The address bar displays "http://localhost/chap16/mysqllex4.php" and "php 5.3". The page content is titled "Tous nos articles par catégorie" and states "Il y a 11 articles en magasin". Below this, a table lists 11 articles with columns: Code article, Description, Prix, and Catégorie. The table data is as follows:

Code article	Description	Prix	Catégorie
CA300	Canon EOS 3000V zoom 28/80	329.00	photo
NIK80	Nikon F80	479.00	photo
NIK55	Nikon F55+zoom 28/80	269.00	photo
CP100	Caméscope Panasonic SV-AV 100	1490.00	vidéo
CS330	Caméscope Sony DCR-PC330	1629.00	vidéo
SAX15	Portable Samsung X15 XVM	1999.00	informatique
HP497	PC Bureau HP497 écran TFT	1500.00	informatique
SOXMP	PC Portable Sony Z1-XMP	2399.00	informatique
DEL30	Portable Dell X300	1715.00	informatique
CAS07	Cassette DV60 par 5	26.90	divers
DVD75	DVD vierge par 3	17.50	divers

Terminé

**Figure 15-1**  
*Lecture de la table article*

## Lecture des noms de colonnes

Dans l'exemple précédent, les titres des colonnes du tableau HTML étant écrits à l'avance dans le script, nous pouvons automatiser cette opération en récupérant les noms des colonnes de la table interrogée ou les alias figurant dans les requêtes SQL. La méthode `fetch_fields()` d'un objet `mysqli_result` nous fournit ces informations et bien d'autres concernant la table. Sa syntaxe est :

```
| array $result->fetch_fields(void)
```

Le tableau retourné contient autant d'objets qu'il existe de colonnes dans la requête SQL. Ces derniers ont tous les mêmes propriétés, détaillées dans le tableau 15-1, permettant d'obtenir des informations sur les attributs de la table.

**Tableau 15-1 Définition des propriétés**

Propriété	Définition
name	Nom de la colonne
orgname	Nom initial de la colonne si un alias a été créé
+table	Nom de la table à laquelle la colonne appartient (s'il n'a pas été obtenu dynamiquement par concaténation, par exemple)
orgtable	Nom initial de la table si un alias a été créé
def	Valeur par défaut de l'attribut, donnée dans une chaîne de caractères
max_length	Longueur maximale du champ pour le jeu de résultats
length	Longueur du champ dans la définition de table
charsetnr	Jeu de caractères pour cet attribut
flags	Entier représentant le bit-flags pour cet attribut
type	Type de donnée utilisée pour l'attribut
decimals	Nombre de décimales utilisées (pour les attributs de type entier)

Ces informations peuvent nous permettre de reconstituer la structure de la table à laquelle nous avons accès sans en connaître les détails.

Nous allons utiliser une de ces propriétés pour créer automatiquement les en-têtes d'un tableau HTML à partir des noms des colonnes ou des alias éventuels définis dans la requête. L'exemple 15-5 illustre cette possibilité à partir d'une requête SQL sélectionnant les articles de la table `article` dont la désignation contient le mot « Sony » en définissant des alias pour les noms des colonnes (repère ①). Après l'envoi de la requête par la méthode `query()`, nous récupérons le résultat (repère ②) puis son nombre de lignes (repère ③) et le tableau d'objets `$titres` contenant les informations présentées dans le tableau 15-1 (repère ④). Les noms des colonnes ou des alias employés ici sont contenus dans les propriétés `$titres->name` et lus un par un à l'aide d'une boucle `foreach` (repère ⑤), puis affichés dans les en-têtes par des éléments `<th>` du tableau HTML (repère ⑥). L'ensemble des lignes du résultat de la requête est obtenu dans le tableau `$tabresult`, dont chaque élément est lui-même un tableau, grâce à la méthode `fetch_all()` (repère ⑦). Les résultats sont affichés à l'aide de deux boucles `foreach` imbriquées (repères ⑧ et ⑨). Le résultat et l'objet connexion sont ensuite supprimés.

## Exemple 15-5. Lecture des noms des colonnes

```

<!DOCTYPE html>
<html 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("exemple15.2.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"; ←①
//*****
$result=$idcom->query($requete); ←②
//*****
if(!$result)
{
    echo "Lecture impossible";
}
else
{
    $nbart=$result->num_rows; ←③
    $titres=$result->fetch_fields(); ←④
    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 $colonne) ←⑤
    {
        echo "<th>", htmlentities($colonne->name), "</th>"; ←⑥
    }
    echo "</tr>";
    // Lecture de TOUTES les lignes du résultat
    $tabresult=$result->fetch_all(); ←⑦
    foreach($tabresult as $ligne) ←⑧
    {
        echo "<tr>";
        foreach($ligne as $valeur) ←⑨
        {
            echo "<td> $valeur </td>";
        }
        echo "</tr>";
    }
    echo "</table>";
}
$result->free();
$idcom->close();
?>
</body>
</html>

```

La figure 15-2 illustre les résultats obtenus.

The screenshot shows a Firefox browser window with the title "Lecture de la table article - Mozilla Firefox". The address bar displays "http://localhost/chap16/mysqlie5.php". The page content includes the heading "Tous nos articles de la marque Sony" and a message "Il y a 2 articles en magasin". Below this is a table with four columns: "Code article", "Désignation", "Prix Unitaire", and "Catégorie". The table contains two rows of data:

Code article	Désignation	Prix Unitaire	Catégorie
CS330	Caméscope Sony DCR-PC330	1629.00	vidéo
SOXMP	PC Portable Sony Z1-XMP	2399.00	informatique

At the bottom of the page, there is a status bar with the text "Terminé".

**Figure 15-2**  
*Lecture des noms des colonnes*

## Récupération des valeurs dans un objet

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

```
| object $result->fetch_object()
```

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

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

```
| $ligne=$result->fetch_object()
```

La valeur d'un attribut `nom` est lue avec la ligne de code :

```
| $ligne->nom
```

L'exemple 15-6 réalise, en employant cette méthode, la recherche et l'affichage dans un tableau HTML 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 ①). Pour les lire, nous appelons une première fois la méthode `fetch_object()` pour l'objet `$result` (repère ②) et récupérons un objet `$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 ③). Nous intégrons alors ces valeurs dans des éléments `<th>` (repère ④). 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 5), 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 6) et afficher les données. Remarquez que, comme nous l'avons déjà signalé, les noms des propriétés de l'objet `$ligne` sont ici les alias définis dans la requête SQL (repère 7). Le tableau HTML obtenu est représenté à la figure 15-3.

## Exemple 15-6. Lecture des données dans un objet

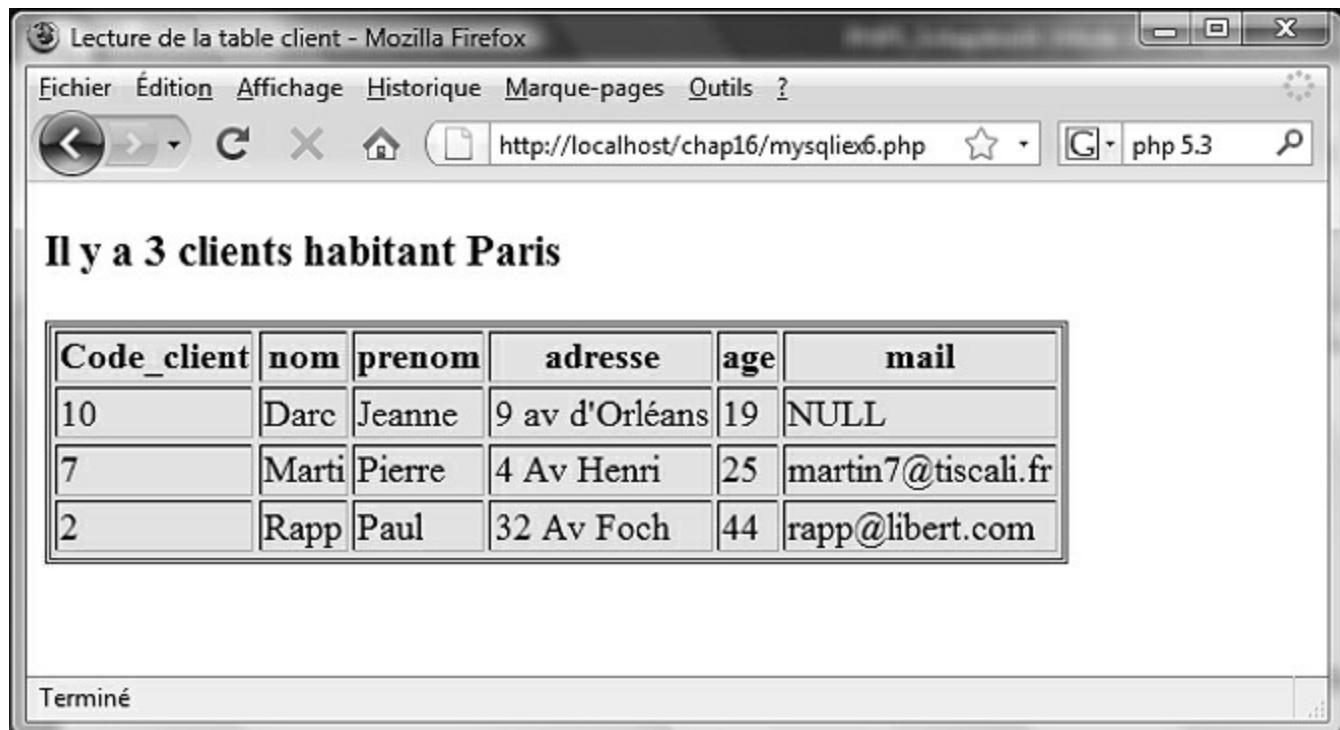
```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Lecture de la table client</title>
<style type="text/css" >
table {border-style: double; border-width: 3px; border-color: red; background-color: yellow; }
</style>
</head>
<body>
<?php
include("exemple15.2.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"; ← 1
$result=$idcom->query($requete);
if(!$result)
{
    echo "Lecture impossible";
}
else
{
    $nbart=$result->num_rows;
    echo "<h3> Il y a $nbart clients habitant Paris</h3>";
    // Affichage des titres du tableau
    $titres=$result->fetch_object(); ← 2
    echo "<table border=\"1\"> <tr>";
    foreach($titres as $colonne=>$val) ← 3
    {
        echo "<th>", $colonne , "</th>"; ← 4
    }
    echo "</tr>";
    // Affichage des valeurs du tableau
    echo "<tr>";
    $result->data_seek(0); ← 5
    while ($ligne = $result->fetch_object()) ← 6
    {
        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>"; ← 7
}
echo "</table>";
$result->free();
$idcom->close();
}
?>
</body>
</html>

```



**Figure 15-3**  
*Lecture des noms des colonnes*

## Insertion de données dans la base

Dans un site interactif, il faut pouvoir enregistrer dans la base de données les informations saisies par les visiteurs dans un formulaire HTML en vue d'une réutilisation ultérieure, comme dans le cas des coordonnées complètes d'un client.

En vous situant, dans la perspective d'un site de e-commerce, vous allez réaliser la saisie puis l'insertion des coordonnées d'un client dans la table `client` de la base `magasin`. Dans un second temps, nous lui permettrons de mettre à jour ces informations.

## Insertion des données

Le formulaire HTML est l'outil privilégié pour saisir de données et les envoyer vers le serveur PHP/MySQL. Nous disposons désormais de la fonction `connexobjet()` pour effectuer la connexion et de la méthode `query()` pour l'envoi des requêtes. Seule la commande SQL `INSERT` distingue cette opération de celle de lecture de données. Le

script de l'exemple 15-7 réalise ce type d'insertion en récupérant les données saisies par le client dans un formulaire lors d'une commande. Nous commençons par vérifier l'existence des saisies obligatoires correspondant aux variables `$_POST['nom']`, `$_POST['adresse']` et `$_POST['ville']` (repère 1). Quand une requête est formée en utilisant les saisies faites par l'utilisateur, il est préférable d'utiliser le caractère d'échappement pour les caractères spéciaux des chaînes récupérées dans le tableau `$_POST`, en particulier les guillemets, qui peuvent poser problème dans la requête. Nous disposons pour cela de la méthode `real_escape_string()` des objets `mysqli` dont la syntaxe est :

```
| string $idcom->real_escape_string(string $chaine)
```

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

Le script récupère toutes les saisies et les protège (repères 3 à 8). La colonne `id_client` de la table `client` ayant été déclarée avec l'option `AUTO_INCREMENT`, il faut y insérer la valeur `NULL` en lui donnant la valeur `"\N"` (repère 2). Le résultat de la requête est ici un booléen permettant de vérifier la bonne insertion des données dans la table (repère 9). Le script doit communiquer son identifiant au client pour qu'il puisse modifier éventuellement ses données. La valeur de la colonne `id_client` est récupérable à l'aide de la propriété `insert_id` de l'objet `mysqli`. Ce nombre entier est affiché dans une boîte d'alerte JavaScript (repère 10).

## Exemple 15-7. Insertion de données

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Saisissez vos coordonnées</title>
</head>
<body>
<form action= "<?php echo $_SERVER['PHP_SELF'];?>" method="post"
enctype="application/x-www-form-urlencoded">
<fieldset>
<legend><b>Vos coordonnées</b></legend>
<table>
<tr><td>Nom : </td><td><input type="text" name="nom" size="40" maxlength="30"/></td>
</tr>
<tr><td>Prénom : </td><td><input type="text" name="prenom" size="40" maxlength="30"/>
</td></tr>
<tr><td>Âge : </td><td><input type="text" name="age" size="40" maxlength="2"/></td>
</tr>
<tr><td>Adresse : </td><td><input type="text" name="adresse" size="40" maxlength="60"/></td></tr>
<tr><td>Ville : </td><td><input type="text" name="ville" size="40" maxlength="40"/>
</td></tr>
<tr><td>E-mail : </td><td><input type="text" name="mail" size="40" maxlength="50"/>
</td></tr>
<tr>
<td><input type="reset" value="Effacer"></td>
<td><input type="submit" value="Envoyer"></td>
</tr>
</table>
```

```

</fieldset>
</form>
<?php
include("exemple15.2.php");
$idcom=connexobjet('magasin', 'myparam');
if(!empty($_POST['nom']) && !empty($_POST['adresse']) && !empty($_POST['ville'])) ← ①
{
    $id_client="\N"; ← ②
    $nom=$idcom->escape_string($_POST['nom']); ← ③
    $prenom=$idcom->escape_string($_POST['prenom']); ← ④
    $age=$idcom->escape_string($_POST['age']); ← ⑤
    $adresse=$idcom->escape_string($_POST['adresse']); ← ⑥
    $ville=$idcom->escape_string($_POST['ville']); ← ⑦
    $mail=$idcom->escape_string($_POST['mail']); ← ⑧
    // Requête SQL
    $requete="INSERT INTO client VALUES('$id_client', '$nom', '$prenom', '$age',
    '$adresse', '$ville', '$mail')";
    $result=$idcom->query($requete); ← ⑨
    if(!$result)
    {
        echo $idcom->errno;
        echo $idcom->error;
        echo "<script type=\"text/javascript\">
        alert('Erreur : ".$idcom->error."')</script>";
    }
    else
    {
        echo "<script type=\"text/javascript\">
        alert('Vous êtes enregistré. Votre numéro de client est :
        ".$idcom->insert_id.")</script>"; ← ⑩
        $idcom->close();
    }
}
else {echo "<h3>Formulaire à compléter !</h3>";}
?>
</body>
</html>

```

La figure 15-4 illustre la page de saisie et la figure 15-5 la boîte d'alerte JavaScript qui donne son identifiant au client.

Saisissez vos coordonnées - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

http://localhost/chap16/mysqlie7.php

G+ php 5.3

**Vos coordonnées**

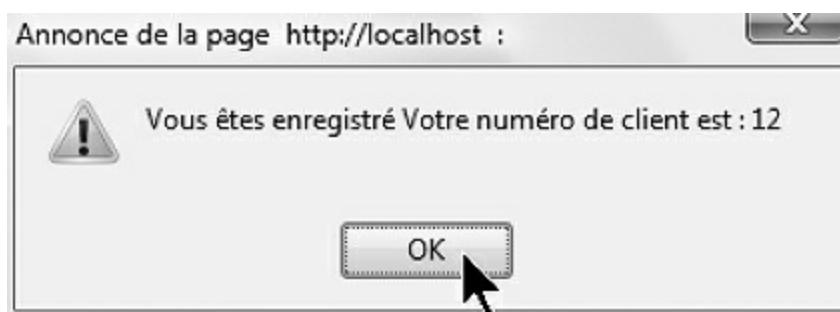
Nom :	Engels
Prénom :	Jean
Age :	55
Adresse :	Rue compoint
Ville :	Paris
Mail :	php5@funhtml.com

**Effacer** **Envoyer**

**Formulaire à compléter!**

Terminé

**Figure 15-4**  
*Formulaire d'insertion de données*



**Figure 15-5**  
*Boîte d'alerte JavaScript donnant le numéro de client*

## Mise à jour d'une table

Un client doit pouvoir modifier ses coordonnées, comme son adresse de livraison ou son e-mail. L'exemple 15-8 crée une page contenant un formulaire qui permet la saisie du code client dans une zone de texte HTML (repère ①). L'attribut `action` de l'élément `<form>` renvoie le traitement de la saisie au fichier `mysqlie9.php` de l'exemple 15-9. La page créée est conforme à la figure 15-6.

### Exemple 15-8. Page de saisies des modifications

```
<!DOCTYPE html>
```

```

<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Modifiez vos coordonnées</title>
</head>
<body>
<form action= "exemple15.9.php" method="post" enctype="application/x-www-form-urlencoded">
    <fieldset>
        <legend><b>Saisissez votre code client pour modifier vos coordonnées</b></legend>
        <table><tbody>
            <tr>
                <td>Code client : </td>
                <td><input type="text" name="code" size="20" maxlength="10"/></td>←①
            </tr>
            <tr>
                <td>Modifier : </td>
                <td><input type="submit" value="Modifier"/></td>
            </tr>
        </tbody></table>
    </fieldset>
</form>
</body>
</html>

```



**Figure 15-6**  
*Page de saisie du code client*

La mise à jour des coordonnées du client est réalisée par le script de l'exemple 15-9.

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 ①). 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 et la seconde enregistre ces données dans la base.

Lors du premier appel du fichier de l'exemple 15-9, la condition de l'instruction `if`

(repère 2) est nécessairement vérifiée car la variable `$_POST['modif']` ne contient rien. Elle correspond à la valeur associée au bouton `submit` du formulaire qui n'est pas encore créé. Le script crée une connexion au serveur MySQL pour y lire les coordonnées actuelles du client, dont le code est contenu dans la variable `$code` issue de la page de saisie de l'exemple 15-8 (repère 3).

La requête SQL sélectionne alors 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 `$code` (repère 4), dans le but de compléter le formulaire avec les données actuelles. Cela permet de ne saisir que les modifications éventuelles de coordonnées du client, sans devoir ressaisir l'ensemble. Ces coordonnées sont lues à l'aide de la méthode `fetch_row()` de l'objet `$result` de type `mysqli_result` (repère 5), puisque le résultat de la requête `SELECT` ne comporte qu'une seule ligne. Elles sont alors contenues dans la variable `$coord` de type `array`. Pour afficher les coordonnées dans le formulaire, vous devez attribuer les valeurs de ses éléments aux attributs `value` des différents champs `<input />` (repère 6).

La figure 15-7 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 7).

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 8). Seules les colonnes `nom`, `adresse`, `ville` et `mail` peuvent être mises à jour à l'aide de la requête suivante (repère 9) le reste étant forcément inchangé :

```
UPDATE client SET nom='$nom', adresse='$adresse', ville='$ville',
  mail='$mail'
 WHERE id_client='$code'
```

La vérification du résultat de la requête (repère 10) permet d'afficher une boîte d'alerte JavaScript contenant soit un message d'erreur, soit la confirmation de l'enregistrement (repère 11). Le visiteur est automatiquement redirigé vers la page `exemple15.8.html` de saisie d'un code (repère 12).

## Exemple 15-9. Mise à jour de données

```
<?php
  if(empty($_POST['code'])) {header("Location:exemple15.8.php");} ← 1
?
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Modifiez vos coordonnées</title>
</head>
<body>
<?php
  include('exemple15.2.php');
```

```

$idcom=connexobjet('magasin','myparam');

if(!isset($_POST['modif'])) ←②
{
    $code=$idcom->escape_string($_POST['code']); ←③
    // Requête SQL
    $requete="SELECT * FROM client WHERE id_client='".$code' "; ←④
    $result=$idcom->query($requete);
    $coord=$result->fetch_row(); ←⑤
    // Création du formulaire complété avec les données existantes ←⑥
    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="$coord[2]"/></td></tr>";
    echo "<tr><td>Âge : </td><td><input type=\"text\" name=\"age\" size=\"40\""
        maxlength="2" value="$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>E-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\"/>"; ←⑦
    echo "</form>";
    $result->free();
    $idcom->close();
}

elseif(isset($_POST['nom']) && isset($_POST['adresse']) && isset($_POST['ville'])) ←⑧
{
    // ENREGISTREMENT
    $nom=$idcom->real_escape_string($_POST['nom']);
    $adresse=$idcom->real_escape_string($_POST['adresse']);
    $ville=$idcom->real_escape_string($_POST['ville']);
    $mail=$idcom->real_escape_string($_POST['mail']);
    $age=(integer)$_POST['age'];
    $code=$idcom->real_escape_string($_POST['code']);
    // Requête SQL
    $requete="UPDATE client SET nom='$nom', adresse="
        . '$adresse', ville='$ville', mail='$mail', age=$age WHERE id_client='".$code'"; ←⑨
    $result=$idcom->query($requete);

    if(!$result) ←⑩
    {
        echo "<script type=\"text/javascript\"> alert('Erreur : ".$result->error."')"
            . "</script>"; ←⑪
    }
}

```

```

}
else
{
    echo "<script type=\"text/javascript\"> alert('Vos modifications sont
    ↵enregistrées');window.location='exemple15.8.php';</script>";←⑫
}
$result->free();
$idcom->close();
}
else
{
    echo "Modifiez vos coordonnées !";
}
?>
</body>
</html>

```

## Vérification de l'insertion

La propriété `affected_rows` de l'objet `mysqli` contient le nombre de lignes affectées par une mise à jour. Elle peut donc nous permettre de savoir si la modification est bien réalisée, car elle doit ici être égale à 1. Au repère ⑩ de l'exemple 15-9, vous pourriez donc écrire :

```

if($idcom->affected_rows() !=1)
{ // Affichage de l'erreur}
else
{ // Message de confirmation}

```

The screenshot shows a Mozilla Firefox browser window with the title "Modifiez vos coordonnées - Mozilla Firefox". The address bar displays the URL `http://localhost/chap16/mysqlie9.php`. The page content is a form titled "Modifiez vos coordonnées" with the following fields:

- Nom :
- Prénom :
- Age :
- Adresse :
- Ville :
- Mail :

At the bottom of the form are two buttons: "Effacer" and "Enregistrer". A mouse cursor is hovering over the "Enregistrer" button. Below the form, there is a status message "Terminé".

**Figure 15-7**  
*Formulaire de saisie des coordonnées créé dynamiquement*

# Recherche dans la base

Un site de commerce en ligne doit permettre à ses visiteurs et futurs clients d'effectuer des recherches dans la base de données afin d'accéder plus rapidement à l'information sur le produit recherché. Il doit en outre permettre d'effectuer des statistiques marketing à l'usage du propriétaire du site. Ces recherches concernent aussi bien les sites de commerce en ligne que les annuaires et moteurs de recherche des sites de contenu.

L'exemple 15-10 crée un formulaire classique permettant de saisir un mot-clé et d'effectuer des choix de tri des résultats. Les critères de tri selon le prix, la catégorie ou l'identifiant d'article sont affichés sous forme de liste déroulante. Le choix de l'ordre croissant ou décroissant s'effectue au moyen de deux boutons radio ayant le même attribut `name`, ce qui les rend exclusifs l'un de l'autre.

Le script contrôle d'abord que le visiteur a saisi un mot-clé dans le formulaire en vérifiant que la variable `$_POST['motcle']` n'est pas vide (repère 1). Il récupère ensuite le motclé, la catégorie, le critère de tri et l'ordre d'affichage, respectivement dans les variables `$motcle`, `$categorie`, `$ordre` et `$tri` (repères 2 à 5).

Si la catégorie choisie est "tous", la partie de la commande `WHERE` concernant cette catégorie est vide. Pour les autres choix, elle est égale à "`AND categorie=$categorie`" (repère 6). La requête de sélection suivante (repère 7) est alors créée par le code :

```
"SELECT id_article AS 'Code article',
designation AS 'Description',
prix,categorie AS 'Catégorie'
FROM article WHERE lower(designation) LIKE '%$motcle%'".$reqcategorie.
"ORDER BY $tri $ordre";
```

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` (repère 8). La lecture des résultats et l'affichage de toutes les lignes retournées sont réalisés avec la méthode `fetch_row()` (repère 9). La figure 15-8 illustre la page créée après la recherche du mot-clé `portable`.

## Exemple 15-10. Page de recherche d'articles

```
<!DOCTYPE html>
<html 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-clé : </td>
<td><input type="text" name="motcle" size="40" maxlength="40" /></td>
</tr>
<tr>
<td>Dans la catégorie : </td>
<td>
<select name="categorie">
<option value="tous">Tous</option>
<option value="vidéo">Vidéo</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">Catégorie</option>
<option value="id_article">Code</option>
</select>
</td>
</tr>
<tr><td>En ordre : </td>
<td>Croissant<input type="radio" name="ordre" value="ASC" checked="checked"/>
 Décroissant</input type="radio" name="ordre" value="DESC" />
</td> </tr>
<tr><td>Envoyer</td><td><input type="submit" name="" value="OK"/></td> </tr>
</tbody>
</table>
</fieldset>
</form>
<?php
if(!empty($_POST['motcle'])) ← ①
{
    include('exemple15.2.php');
    $motcle=strtolower($_POST['motcle']); ← ②
    $categorie=$_POST['categorie']; ← ③
    $ordre=$_POST['ordre']; ← ④
    $tri=$_POST['tri']; ← ⑤

    // Requête SQL
    $reqcategorie=($_POST['categorie']=="tous")?"":"AND categorie='".$categorie."'";
    $requete="SELECT id_article AS 'Code article',designation AS 'Description',
 prix,categorie AS 'Catégorie' FROM article WHERE lower(designation) LIKE
 lower('%$motcle%').".$reqcategorie."ORDER BY $tri $ordre"; ← ⑦
    $idcom=connexobjet('magasin','myparam');
    $result=$idcom->query($requete); ← ⑧
    if(!$result) ← ⑨
    {
        echo "Lecture impossible";
    }
}

```

```
else
{
$nbcol=$result->field_count;
$nbart=$result->num_rows;
$titres=$result->fetch_fields();
echo "<h3>Il y a $nbart articles correspondant au mot-clé : $motcle</h3>";
    // Affichage des titres du tableau
    echo "<table border=\"1\"> <tr>";
foreach($titres as $nomcol=>$val)
{
    echo "<th>, $titres[$nomcol]->name ,</th>";
}
echo "</tr>";
    // Affichage des valeurs du tableau
for($i=0;$i<$nbart;$i++)
{
    $ligne=$result->fetch_row();
    echo "<tr>";
    for($j=0;$j<$nbcol;$j++)
    {
        echo "<td>",$ligne[$j], "</td>";
    }
    echo "</tr>";
}
echo "</table>";
$result->free();
$idcom->close();
}
}
?>
</body>
</html>
```

Rechercher un article dans le magasin - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

http://localhost/chap16/mysqllex10.php

php 5.3

**Rechercher un article en magasin**

Mot-clé:	portable
Dans la catégorie :	Informatique
Trier par :	Prix
En ordre:	Croissant <input checked="" type="radio"/> Décroissant <input type="radio"/>
Envoyer	<b>OK</b>

**Il y a 3 articles correspondant au mot-clé : portable**

Code article	Description	prix	Catégorie
DEL30	Portable Dell X300	1715.00	informatique
SAX15	Portable Samsung X15 XVM	1999.00	informatique
SOXMP	PC Portable Sony Z1-XMP	2399.00	informatique

Terminé

**Figure 15-8**  
*Formulaire de recherche et résultats obtenus*

## Les requêtes préparées

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

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

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

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

- 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 15-2.

**Tableau 15-2. Signification des caractères de la chaîne \$types**

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

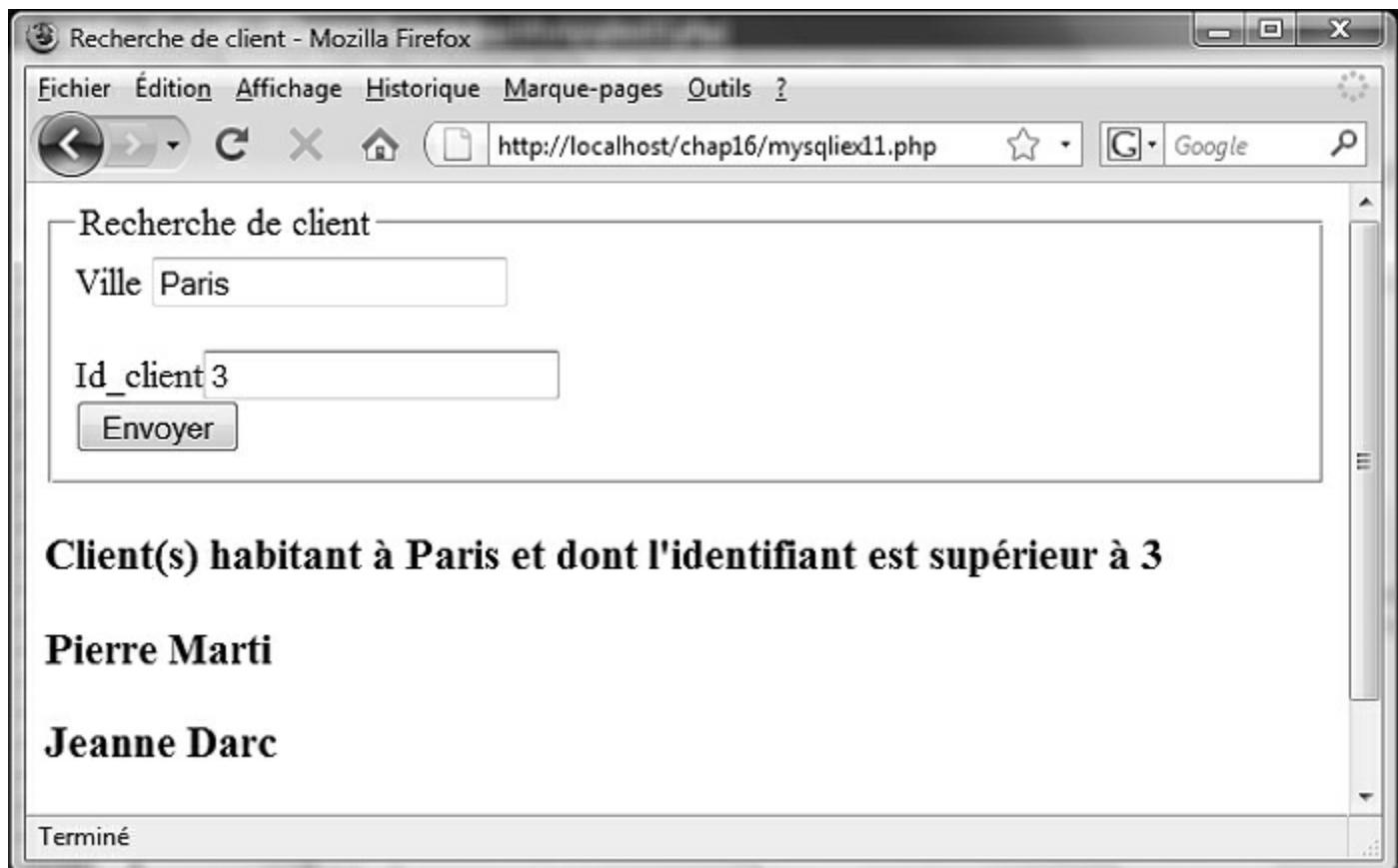
3. Pour trois paramètres qui seraient, dans l'ordre d'apparition dans la requête, un décimal, une chaîne et un entier, la chaîne `$types` serait par exemple "dsi".
4. Exécuter la requête en appelant la méthode `execute()` de l'objet `mysqli_stmt`.
5. Pour les requêtes préparées qui retournent des résultats, comme `SELECT`, il faut ensuite lier les résultats à des variables PHP, avec la méthode `bind_result()` selon la syntaxe : `mysqli_stmt->bind_result($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 15-11 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 ① et ②) 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 ③ et ④). Après la connexion à la base, nous écrivons la requête préparée avec la méthode `prepare()` (repère ⑤), nous lions les paramètres de type `string` et `integer` aux variables `$ville` et `$id_client` (repère ⑥), puis nous exécutons la requête ainsi complétée en appelant la méthode `execute()` (repère ⑦). Comme il s'agit d'une requête `SELECT`, elle retourne une ou plusieurs lignes contenant chacune deux valeurs que nous lions aux variables `$prenom` et `$nom` (repère ⑧). Les différentes lignes sont alors lues et affichées avec une boucle `while` en appelant la méthode `fetch()` (repère ⑨), les valeurs cherchées étant contenues dans les variables `$prenom` et `$nom`. La mémoire est ensuite libérée (repère ⑩) et l'objet `mysqli` supprimé (repère ⑪).

## Exemple 15-11. Utilisation des requêtes préparées

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Recherche de client</title>
<style type="text/css" >
div{font-size: 16px; }
</style>
</head>
<body>
<form method="post" action="exemple15.11.php">
<fieldset>
<legend>Recherche de client</legend>
<label>Ville </label>
<input type="text" name="ville" /><br /> ← 1
<label>Id_client</label>
<input type="text" name="id_client" /> ← 2
<input type="submit" value="Envoyer" />
</fieldset>
</form>
</body>
</html>
<?php
if(isset($_POST['ville']) && isset($_POST['id_client']))
{
    $ville=strtolower($_POST['ville']); ← 3
    $id_client=$_POST['id_client']; ← 4
    include('exemple15.2.php');
    $idcom=connexobjet('magasin','myparam');
    $reqprep=$idcom->prepare("SELECT prenom,nom FROM client WHERE lower(ville)
    ←=? AND id_client=?"); ← 5
    $reqprep->bind_param("si",$ville,$id_client); ← 6
    $reqprep->execute(); ← 7
    $reqprep->bind_result($prenom,$nom); ← 8
    echo "<div><h3>Client(s) habitant à ", ucfirst($ville), " et dont l'identifiant est
    ← supérieur ou égal à $id_client</h3>";
    // Affichage des résultats
    while ($reqprep->fetch()) ← 9
    {
        echo "<h3> $prenom $nom</h3>";
    }
    echo "</div>";
    $reqprep->free_result(); ← 10
    $idcom->close(); ← 11
}
?>
```

Avec notre base de données magasin et les paramètres « Paris » et « 3 », nous obtenons par exemple l'affichage présenté à la figure 15-9.



**Figure 15-9**  
*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.

Dans l'exemple 15-12, nous illustrons la procédure à suivre pour effectuer deux requêtes `INSERT` dans la table `article`. Par défaut, le mode `autocommit` de MySQL est activé, ce qui signifie que chaque requête est automatiquement validée. Il nous faut donc le désactiver au moyen de la méthode `autocommit()` de l'objet `mysqli` avec pour paramètre la valeur `FALSE` (repère ①). À partir de cet instant, les deux requêtes

d'insertion que nous voulons réaliser (repères ② et ③) ne seront validées que si nous effectuons explicitement la validation au moyen de la méthode `commit()` (repère ⑥), ou bien l'ensemble sera annulé en appelant la méthode `rollback()` de l'objet `mysqli` (repère ⑦). Pour choisir de valider ou annuler les insertions, nous comptons le nombre total de lignes insérées dans la base en lisant la propriété `affected_rows` (repères ④ et ⑤). S'il est bien égal à 2, la validation est effectuée (repère ⑥), sinon tout est annulé et un message d'information s'affiche (repère ⑦).

Pour tester l'efficacité du mécanisme, il suffit de créer une erreur en écrivant par exemple dans la variable `$requete2` le nom `articles`, au lieu de `article`, comme nom de table inexistante. L'affichage de la table `article` avec phpMyAdmin permet de vérifier que même la première requête, qui était correcte, n'a pas été exécutée.

## Exemple 15-12. Insertions avec transaction

```
<?php
include('exemple15.2.php');
$idcom=connexobjet('magasin','myparam');
$idcom->autocommit(FALSE); ←①
$requete1="INSERT INTO article VALUES ('AZERT', 'Lecteur MP3', 59.50,'divers');"; ←②
$requete2="INSERT INTO articles VALUES ('QSDFG', 'Bridge Samsung 10 Mo',
358.90,'photo');"; ←③
// Pour empêcher la validation, écrire "articles" au lieu de "article" comme nom de
table
//*****$idcom->query($requete1);
$nb=$idcom->affected_rows; ←④
echo "LIGNES INSEREEES",$nb,"<hr />";
$idcom->query($requete2);
$nb+=$idcom->affected_rows; ←⑤
if($nb==2)
{
    $idcom->commit(); ←⑥
    echo $nb, " lignes insérées";
}
else
{
    $idcom->rollback(); ←⑦
    echo "transaction annulée";
}
?>
```

## Mémo des méthodes et propriétés

### *Classe mysqli : méthodes*

```
mysqli ( [string $host [, string $username [, string $passwd [, string $base [, int $port [, string $socket]]]]] ] )
```

Crée un objet mysqli.

---

```
boolean autocommit ( boolean $mode )
```

Active (`$mode =TRUE`) ou désactive (`$mode=FALSE`) le mode autocommit.

---

```
boolean change_user ( string $user, string $password, string $base )
```

Change l'utilisateur de la connexion et retourne `TRUE` en cas de réussite et `FALSE` sinon.

---

```
boolean close (void)
```

Ferme la connexion et retourne `TRUE` en cas de réussite et `FALSE` sinon.

---

```
boolean commit ( void )
```

Valide la transaction courante et retourne `TRUE` en cas de réussite et `FALSE` sinon.

---

```
string real_escape_string ( string $chaine )
```

Crée une chaîne SQL valide qui pourra être utilisée dans une requête SQL. La chaîne de caractères `$chaine` est encodée en une chaîne SQL échappée, en tenant compte du jeu de caractères courant de la connexion.

---

```
boolean kill ( int $processid )
```

Demande au serveur de terminer un thread MySQL identifié par son identifiant.

---

```
boolean multi_query ( string $query )
```

Exécute une ou plusieurs requêtes, rassemblées dans le paramètre `query` par des points-virgules.

---

```
boolean ping ( void )
```

Teste la connexion pour s'assurer que le serveur est bien en fonctionnement. S'il ne fonctionne pas et que l'option globale `mysqli.reconnect` est activée, une connexion automatique sera tentée avec les derniers paramètres utilisés.

---

```
divers query ( string $query [, int $mode] )
```

Exécute une requête sur la base de données. `$mode` est une constante qui vaut `MYSQLI_USE_RESULT` ou `MYSQLI_STORE_RESULT` (par défaut), suivant le comportement désiré. Retourne `TRUE` en cas de succès, `FALSE` en cas d'échec. Pour `SELECT`, `SHOW`, `DESCRIBE` ou `EXPLAIN` retourne un objet `mysqli_result`.

---

```
boolean rollback ( void )
```

Annule la transaction courante.

---

```
boolean select_db ( string $base )
```

Sélectionne une base de données.

---

```
string stat ( void )
```

Retourne une chaîne de caractères contenant des informations sur la connexion ouverte : le temps de fonctionnement, exprimé en secondes, le nombre de threads courant, le nombre de commandes, les tables rechargées et ouvertes.

---

```
objet mysqli_result store_result ( void )
```

Stocke un ensemble de résultats dans un objet `mysqli_result` à partir de la dernière requête.

## ***Classe mysqli : propriétés***

`integer affected_rows`

Contient le nombre de lignes affectées par la dernière requête `INSERT`, `UPDATE`, `REPLACE` ou `DELETE`.

---

```
integer errno
```

Contient un code d'erreur pour la dernière commande SQL.

---

```
integer error
```

Contient une chaîne décrivant la dernière erreur.

---

```
integer field_count
```

Contient le nombre de colonnes concernées dans la dernière requête.

---

```
integer insert_id
```

Contient l'identifiant automatiquement généré pour un attribut déclaré `AUTO_INCREMENT` ou `0` sinon.

---

```
thread_id
```

Contient l'identifiant du thread pour la connexion en cours.

---

```
integer warning_count
```

Contient le nombre d'avertissements générés par la dernière requête.

## ***Classe mysqli\_result : méthodes***

```
boolean close ( void )
```

Supprime l'objet résultat.

---

```
boolean data_seek(integer N)
```

Déplace le pointeur interne de résultat à la position `N`.

---

```
divers fetch_array ( [integer $type] )
```

Retourne un tableau qui correspond à la ligne récupérée, ou `NULL` s'il n'y a plus de ligne dans le résultat. Le paramètre `$type` détermine le type du tableau ; il vaut `MYSQLI_ASSOC`, `MYSQLI_NUM` ou `MYSQLI_BOTH` respectivement pour obtenir un tableau associatif, indicé ou mixte.

---

```
array fetch_assoc ( void )
```

Retourne une ligne de résultat sous forme de tableau associatif.

---

```
object fetch_field ( void )
```

Retourne un objet qui contient les caractéristiques d'un attribut de table.

---

```
array fetch_fields ( void )
```

Retourne un tableau d'objets qui contient les caractéristiques de tous les attributs de table présents dans une requête.

---

```
object fetch_object (void)
```

Retourne un objet dont les propriétés sont les noms des colonnes utilisées dans la requête ou `NULL` s'il n'y a plus de ligne dans le résultat.

---

```
divers fetch_row ( void )
```

Retourne un tableau indicé contenant une ligne de résultat ou `NULL` s'il n'y a plus de ligne de résultat.

---

```
boolean field_seek ( int N )
```

Place le pointeur de résultat sur le champ `N`.

---

```
void free ( void ) ou void close (void)
```

Libère la mémoire associée à l'objet résultat.

## ***Classe mysqli\_result : propriétés***

```
integer field_count
```

Contient le nombre de colonnes pour la dernière requête.

```
int num_rows
```

Retourne le nombre de lignes d'un résultat.

## ***Classe mysqli\_stmt : méthodes***

```
boolean bind_param ( string $types, divers $var1 [...divers $varN] )
```

Lie des variables à une requête SQL préparée avec la méthode `prepare()`.

```
boolean bind_result (divers $var1 [,...divers $varN] )
```

Associe des variables à un résultat de requête préparée et retourne `TRUE` en cas de réussite ou `FALSE` sinon.

```
boolean close ( void )
```

Termine une requête préparée et retourne `TRUE` en cas de réussite ou `FALSE` sinon.

```
void data_seek ( integer N)
```

Déplace le pointeur de résultat à la position `N`.

```
boolean execute ( void )
```

Exécute une requête préparée et retourne `TRUE` en cas de réussite ou `FALSE` sinon.

```
boolean fetch ( void )
```

Lit des résultats depuis une requête MySQL préparée dans les variables liées.

```
void free_result ( void )
```

Libère le résultat de la mémoire.

```
divers prepare ( string $requete )
```

Prépare une requête SQL et retourne `TRUE` en cas de succès, `FALSE` en cas d'échec.

```
boolean reset ( void )
```

Annule une requête préparée et retourne `TRUE` en cas de succès, `FALSE` en cas d'échec.

## ***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 HTML. 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 HTML.

## Exercice 5

Créez un formulaire de recherche permettant de retrouver tous les véhicules possédés par une personne donnée. Affichez les résultats sous forme de tableau HTML.

## Exercice 6

Réécrivez entièrement le code de l'exercice 5 en récupérant tous les résultats dans des objets et en manipulant leurs propriétés.

## Exercice 7

Refaire l'exercice 4 en utilisant une requête préparée.

## Exercice 8

Refaire l'exercice 3 en utilisant une transaction pour s'assurer que les données sont bien insérées dans les différentes tables.

## PDO et MySQL

PDO (*PHP Data Objects*) est une extension qui offre une couche d'abstraction de données introduite dans PHP 5, ce qui signifie qu'elle n'est pas liée à MySQL comme l'extension `mysqli` que nous avons abordée dans le chapitre précédent, mais indépendante de la base de données utilisée. Grâce à elle, le code devient portable très facilement, c'est-à-dire qu'elle modifie uniquement la ligne de connexion, d'une base de données MySQL à SQLite, PostgreSQL, Oracle et d'autres encore par exemple. PDO offre donc aussi bien l'avantage de la portabilité, de la facilité d'utilisation que de la rapidité.

L'extension PDO comprend les trois classes suivantes :

- La classe `PDO`, qui permet de créer des objets représentant la connexion à la base et qui dispose des méthodes permettant de réaliser les fonctions essentielles, à savoir l'envoi de requête, la création de requêtes préparées et la gestion des transactions.
- La classe `PDOStatement`, qui représente, par exemple, une requête préparée ou un résultat de requête `SELECT`. Ses méthodes permettent de gérer les requêtes préparées et de lire les résultats des requêtes.
- La classe `PDOException` qui permet de gérer et d'afficher des informations sur les erreurs à l'aide d'objets.

Ce chapitre reprend exactement la même démarche que le précédent en utilisant exclusivement PDO ; nous allons y reprendre les différents exemples du chapitre 15 et voir comment obtenir les mêmes résultats via un accès objet `PDO`. Il peut donc être abordé indépendamment des deux chapitres précédents.

### Connexion au serveur MySQL

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

- Pour MySQL :

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

- PDO est aussi utilisable avec les bases de données SQLite.

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

Nous pouvons également réutiliser le fichier `myparam.inc.php` créé au chapitre 15, contenant les paramètres de connexion (les paramètres suivants correspondent à une utilisation en local avec WampServer) :

### Exemple 16-1. Le fichier `myparam.inc.php`

```
<?php
define("HOST","localhost");
define("USER","root");
define("PASS","");
define("PORT",8889);
?>
```

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

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

#### Sélection de la base via SQL

Comme nous l'avons vu au chapitre 15, vous pouvez aussi sélectionner une base en envoyant la requête `"USE nom_base"` au serveur à l'aide de la méthode `query()` détaillée dans les sections suivantes.

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

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

```
//Fermeture de la connexion
$Idcom=NULL;
?>
```

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

La fonction inclut d'abord les paramètres de connexion (repère 1), définit la chaîne DSN (*Data Source Name*) pour MySQL (repère 2), puis crée un objet `PDO` dans un bloc `try` (repère 3). En cas d'échec, un bloc `catch` crée un objet `PDOException` (repère 4) qui permet d'afficher un message d'erreur en appelant la méthode `getMessage()` (repère 5). Si la connexion est bien réalisée elle retourne l'objet `$idcom` ou `FALSE` dans le cas contraire.

## Exemple 16-2. Fonction de connexion au serveur

```
<?php
function connexpdo ($base, $param)
{
    include_once ($param.".inc.php"); ← 1
    $dsn="mysql:host=".HOST.";dbname=".$base; ← 2
    $user=USER;
    $pass=PASS;
    try
    {
        $idcom = new PDO($dsn,$user,$pass); ← 3
        return $idcom;
    }
    catch(PDOException $except) ← 4
    {
        echo"Échec de la connexion",$except->getMessage(); ← 5
        return FALSE;
        exit();
    }
}
?>
```

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

```
include("connexpdo.inc.php")
$idcom = connexpdo ("nom_base", "myparam") ;
```

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

## Envoi de requêtes SQL au serveur

Les différentes opérations à réaliser sur la base MySQL impliquent l'envoi de requêtes

## SQL au serveur.

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

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

```
| integer $idcom->exec(string requete)
```

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

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

```
| object $idcom->query(string $requete)
```

Elle retourne `FALSE` en cas d'erreur ou, sinon, un objet de la classe `PDOSTatement` représentant l'ensemble des lignes de résultats – pour lequel il faut ensuite appeler les méthodes spécialisées pour afficher les valeurs. En résumé, un script d'envoi de requêtes se présente sous la forme suivante :

### Exemple 16-3. Envoi type de requête

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Insertion et lecture de la table client</title>
</head>
<body>
<?php
include_once("exemple16.2.php"); ←①
$idcom=connexpdo("magasin","myparam"); ←②
// Requête sans résultats
$requete1="UPDATE client SET age=43 WHERE id_client=7"; ←③
$nb=$idcom->exec($requete1); ←④
echo "<p>$nb ligne(s) modifiée(s)<hr /></p>"; ←⑤
// Requête avec résultats
$requete2="SELECT * FROM client ORDER BY nom"; ←⑥
$result=$idcom->query($requete2); ←⑦
if(!$result) ←⑧
{
    $mes_erreur=$idcom->errorInfo();
    echo "Lecture impossible, code", $idcom->errorCode(),$mes_erreur[2];
}
else ←⑨
{
    while ($row = $result->fetch(PDO::FETCH_NUM))
    {
        foreach($row as $d)
        {
            echo $d," ";
        }
    }
}
```

```

    echo "<hr />";
}
$result->closeCursor(); ← 10
}
$idcom=null;
?>
</body>
</html>

```

Il effectue successivement l'inclusion du fichier `exemple16.2.php` (repère 1), la connexion au serveur (repère 2), l'écriture d'une première requête SQL dans la variable `$requete1` (repère 3) et son envoi à l'aide de la méthode `exec()`, qui retourne le nombre de lignes affectées dans la variable `$nb` (repère 4) et son affichage (repère 5). Une seconde requête contenant la commande `SELECT` (repère 6) est envoyée via la méthode `query()` qui retourne un résultat dans la variable `$result` – qui est un objet de type `PDOStatement` (repère 7). Un test permet de vérifier la bonne exécution de la requête et l'affichage des résultats au moyen de méthodes que nous développerons dans les sections suivantes (repères 8 et 9). Enfin, pour libérer la mémoire occupée par le résultat obtenu, nous utilisons la méthode `closeCursor()` de l'objet `PDOStatement`, surtout utile avant d'envoyer une nouvelle requête `SELECT` au serveur (repère 10).

## Lecture du résultat d'une requête

Pour les opérations d'insertion, de suppression ou de mise à jour des données dans une base, il est utile de vérifier si la requête a bien été exécutée ou, comme dans l'exemple 16-3, de vérifier le nombre de lignes affectées.

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

## *Lecture à l'aide d'un tableau*

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

```
| array $result->fetch(integer type)
```

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

l'indice 0 correspond au premier attribut écrit dans la requête, et ainsi de suite.

Nous pouvons également récupérer toutes les lignes de résultats dans un seul tableau multidimensionnel pour lequel le premier niveau est indicé de 0 à N-1 pour lire N lignes, chaque élément étant lui même un tableau qui peut être indicé, associatif ou mixte selon la valeur du paramètre type qui prend les mêmes valeurs que pour la méthode `fetch()`. Il s'agit de la méthode `fetchAll()` qui sera mise en œuvre dans l'exemple 16-5 et dont la syntaxe est :

```
| array $result->fetchAll(integer type)
```

L'exemple 16-4 met cette méthode en pratique dans le but d'afficher le contenu de la table `article` de la base `magasin` dans un tableau HTML. Après l'inclusion de la fonction de connexion (repère 1), puis son appel sur la base `magasin` (repère 2), nous écrivons la requête SQL de sélection (requête 3). L'envoi de la requête avec la méthode `query()` permet de récupérer un objet `$result` ou `FALSE` en cas d'échec (repère 4). Un test sur la variable `$result` (repère 5) permet d'afficher un message d'erreur (repère 6) ou le contenu de la table (repère 7). Nous récupérons d'abord le nombre d'articles dans la table grâce à la méthode `rowCount()` de l'objet `PDOStatement` (repère 8) et affichons ce nombre dans un titre `<h4>` (repère 9). Une boucle `while` permet de lire une ligne à la fois dans un tableau indicé (repère 10), puis une boucle `foreach` permet d'afficher chacune des valeurs du tableau dans un tableau HTML (repère 11). L'objet `$result` est alors supprimé (repère 12) et la connexion fermée (repère 13). La figure 16-1 illustre l'affichage obtenu dans un navigateur.

## Exemple 16-4. Lecture de la table article

```
<!DOCTYPE html>
<html 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 ("exemple16.2.php"); ← 1
if($idcom=connexpdo("magasin", "myparam")) ← 2
{
  $requete="SELECT * FROM article ORDER BY categorie"; ← 3
  $result=$idcom->query($requete); ← 4
  if(!$result) ← 5
  {
    $mes_erreur=$idcom->errorInfo();
    echo "Lecture impossible, code", $idcom->errorCode(),$mes_erreur[2]; ← 6
  }
  else ← 7
{
```

```

$nbart=$result->rowCount(); ← 8
echo "<h3>Tous nos articles par catégorie</h3>";
echo "<h4>Il y a $nbart articles en magasin</h4>"; ← 9
echo "<table border=\"1\">";
echo "<tr><th>Code article</th> <th>Description</th> <th>Prix</th>
    <th>Catégorie</th></tr>";
while($ligne=$result->fetch(PDO::FETCH_NUM)) ← 10
{
    echo "<tr>";
    foreach($ligne as $valeur) ← 11
    {
        echo "<td> $valeur </td>";
    }
    echo "</tr>";
}
echo "</table>";
}
$result->closeCursor(); ← 12
$idcom=null; ← 13
}
?>
</body>
</html>

```

## Lecture des noms de colonnes

Dans l'exemple précédent, les titres des colonnes du tableau HTML sont écrits à l'avance dans le script. Nous pouvons automatiser cette opération en récupérant les noms des colonnes de la table interrogée, ou les alias figurant dans les requêtes SQL, ce qui peut permettre la réalisation d'un affichage dynamique.

Pour cela, nous allons lire les résultats dans un tableau associatif et récupérer les noms des colonnes de la table ou les alias éventuels, en récupérant les clés de ce tableau dans un autre tableau et en lui appliquant la fonction PHP `array_keys()` (voir le chapitre 5).

Lecture de la table article - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

http://localhost/chap17/mysqlpdoex4.php

Google

## Tous nos articles par catégorie

Il y a 13 articles en magasin

Code article	Description	Prix	Catégorie
NIK80	Nikon F80	479.00	photo
CA300	Canon EOS 3000V zoom 28/80	329.00	photo
NIK55	Nikon F55+zoom 28/80	269.00	photo
QSDFG	Bridge numérique Samsung	358.90	photo
CP100	Caméscope Panasonic SV-AV 100	1490.00	vidéo
CS330	Caméscope Sony DCR-PC330	1629.00	vidéo
HP497	PC Bureau HP497 écran TFT	1500.00	informatique
DEL30	Portable Dell X300	1715.00	informatique
SAX15	Portable Samsung X15 XVM	1999.00	informatique
SOXMP	PC Portable Sony Z1-XMP	2399.00	informatique
DVD75	DVD vierge par 3	17.50	divers
CAS07	Cassette DV60 par 5	26.90	divers
AZERT	Lecteur MP3	59.50	divers

http://validator.w3.org/check?uri=referer

**Figure 16-1**  
*Lecture de la table article*

L'exemple 16-5 illustre cette possibilité à partir d'une requête SQL sélectionnant les articles de la table `article` dont la désignation contient le mot « Sony », en définissant des alias pour les noms des colonnes (repère ❶). Après l'envoi de la requête par la méthode `query()`, nous récupérons le résultat (repère ❷) puis son nombre de lignes (repère ❸). Nous utilisons ici la méthode `fetchAll()` pour récupérer toutes les lignes dans un unique tableau indiqué `$tabresult` (repère ❹). Chacun des éléments de ce tableau est lui-même un tableau associatif car nous avons passé la constante `PDO::FETCH_ASSOC` en paramètre à la méthode `fetchAll()`. Par exemple, le tableau `$tabresult[0]` contient les données de la première ligne du résultat ; il est associatif donc ses clés correspondent aux noms des colonnes ou alias précisés dans la requête SQL. Le tableau `$titres`, retourné par la fonction `array_keys()` (repère ❺), ne contient donc que ces clés que nous lisons dans une boucle `foreach` pour afficher les en-têtes du tableau HTML (repère ❻).

Deux boucles, `for` puis `foreach` (repères 7 et 8), permettent la lecture du tableau multidimensionnel `$tabresult` pour l'affichage des données. Les objets connexion et résultat sont ensuite détruits (repères 9 et 10).

## Exemple 16-5. Lecture automatique des noms des colonnes

```
<!DOCTYPE html>
<html 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("exemple16.2.php");
    $idcom=connexpdo ("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"; ← 1
    $result=$idcom->query($requete); ← 2
    if (!$result)
    {
        $mes_erreur=$idcom->errorInfo();
        echo "Lecture impossible, code : ", $idcom->errorCode(),"<br />", $mes_erreur[2];
    }
    else
    {
        $nbart=$result->rowCount(); ← 3
        $tabresult=$result->fetchAll(PDO::FETCH_ASSOC); ← 4
        // Récupération des noms des colonnes ou des alias
        $titres=array_keys($tabresult[0]); ← 5
        // Affichage des titres de la page
        echo "<h3> Tous nos articles de la marque Sony</h3>";
        echo "<h4> Il y a $nbart articles en magasin </h4>";
        echo "<table border=\"1\"> <tr>";
        // Affichage des titres du tableau
        foreach($titres as $nomcol) ← 6
        {
            echo "<th>", htmlentities($nomcol) , "</th>";
        }
        echo "</tr>";
        // Affichage des lignes de données
        for($i=0;$i<$nbart;$i++) ← 7
        {
            echo "<tr>";
            foreach($tabresult[$i] as $valeur) ← 8
            {
                echo "<td> $valeur </td>";
            }
            echo "</tr>";
        }
    }
}
```

```

    echo "</table>";
}
$result->closeCursor(); ← 9
$idcom=null; ← 10
?>
</body>s
</html>

```

La figure 16-2 illustre les résultats obtenus.

Code article	Désignation	Prix Unitaire	Catégorie
CS330	Caméscope Sony DCR-PC330	1629.00	vidéo
SOXMP	PC Portable Sony Z1-XMP	2399.00	informatique

**Figure 16-2**  
*Lecture des noms des colonnes*

## Récupération des valeurs dans un objet

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

```
| object $result->fetchObject()
```

À chaque appel de cette méthode, l'objet retourné représente une ligne de résultat et possède autant de propriétés qu'il existe d'attributs dans la requête SQL `SELECT` ; les noms de celles ci sont d'ailleurs ceux des colonnes de la table ou des alias éventuels. Nous pouvons également récupérer les lignes du résultat dans un objet en utilisant la méthode `fetch()` avec pour paramètre la constante `PDO::FETCH_OBJ`.

Si nous récupérons une ligne de résultat dans la variable :

```
| $ligne=$result->fetchObject()
```

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

```
| $ligne->nom
```

L'exemple 16-6 réalise la recherche et l'affichage dans un tableau HTML de tous les

clients qui habitent Paris en utilisant cette méthode. En revanche, pour afficher les titres du tableau, nous n'allons pas utiliser la méthode développée dans l'exemple précédent. Dans la requête SQL, nous définissons des alias qui vont correspondre aux en-têtes (repère ①). Pour les lire, nous appelons la méthode `fetchObject()` pour l'objet `$result` (repère ②) et récupérons la première ligne de résultat dans la variable `$ligne`. Cette variable est un objet de type `stdClass` (la classe de base de PHP 5) dont les propriétés ont pour nom ceux des colonnes ou des alias de la requête. Comme il s'agit d'un objet, nous pouvons le parcourir au moyen d'une boucle `foreach` pour écrire les en-têtes du tableau HTML (repère ③). Les valeurs associées ne nous intéressent pas encore ici. Pour récupérer le tableau des données nous utilisons une boucle `do...while` (repère ④), 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 ⑤).

## Exemple 16-6. Lecture des données dans un objet

```

<!DOCTYPE html>
<html 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("exemple16.2.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"; ←①
$result=$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>

```

The screenshot shows a Firefox browser window with the title "Lecture de la table client - Mozilla Firefox". The address bar displays the URL "http://localhost/chap17/mysqlpdoex6.php". The main content area contains the following text:

**Il y a 4 clients habitant Paris**

Code_client	nom	prenom	adresse	age	mail
10	Darc	Jeanne	9 av d'Orléans	19	NULL
12	Engels	Jean	Rue compoint	56	php5@funhtml.com
7	Marti	Pierre	4 Av Henri	43	martin7@tiscali.fr
2	Rapp	Paul	32 Av Foch	44	rapp@libert.com

In the status bar at the bottom, it says "Terminé".

**Figure 16-3**  
*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)
```

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

```

while($ligne=$result->fetch(PDO::FETCH_OBJ))
{
    foreach($ligne as $donnee)
    {
        echo $donnee;
    }
    echo "<br />";
}

```

# Insertion de données dans la base

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

Comme dans le chapitre 15, placez-vous dans la perspective d'un site de e-commerce. Vous allez ainsi envisager la réalisation de la saisie puis de l'insertion des coordonnées d'un client dans la table `client` de la base magasin. Dans un second temps, vous lui permettrez de mettre à jour les informations enregistrées.

## Insertion des données

Le formulaire HTML est l'outil privilégié pour saisir des données et les envoyer vers le serveur PHP/MySQL. Nous disposons désormais de la fonction `connexpdo()` pour effectuer la connexion et des méthodes `exec()` et `query()` pour l'envoi des requêtes. Seule la commande SQL `INSERT` distingue cette opération de celle de lecture de données. Le script de l'exemple 16-7 réalise ce type d'insertion en récupérant les données saisies par le client dans un formulaire lors d'une commande. Nous commençons par vérifier l'existence des saisies obligatoires correspondant aux variables `$_POST['nom']`, `$_POST['adresse']` et `$_POST['ville']` (repère 1). Quand une requête est formée en utilisant les saisies faites par l'utilisateur, il est préférable d'utiliser le caractère d'échappement pour les caractères spéciaux des chaînes récupérées dans le tableau `$_POST`, en particulier les guillemets, qui peuvent poser problème dans la requête. Nous disposons pour cela de la méthode `quote()` des objets `PDO` dont la syntaxe est :

```
| string $idcom->quote(string $chaine)
```

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

Le script récupère toutes les saisies et les protège (repères 3 à 8). La colonne `id_client` de la table `client` ayant été déclarée avec l'option `AUTO_INCREMENT`, il faut y insérer la valeur `NULL` en lui donnant la valeur "`\N`" (repère 2). L'envoi de la requête se faisant avec la méthode `exec()`, la valeur renournée est le nombre de lignes insérées (ici « 1 ») (repère 9), nous pouvons contrôler la bonne fin de l'insertion (repère 10). Le script doit communiquer son identifiant au client pour qu'il puisse modifier éventuellement ses données. La valeur de la colonne `id_client`, qui a été déclarée `AUTO_INCREMENT` dans la table `client`, est récupérable à l'aide de la méthode `lastInsertId()` de l'objet `PDO`. Ce nombre entier est affiché dans une boîte d'alerte JavaScript (repère 11).

### Exemple 16-7. Insertion de données

```
<!DOCTYPE html>
<html lang="fr">
<head>
```

```

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Saisissez vos coordonnées</title>
</head>
<body>
<form action= "<?php echo $_SERVER['PHP_SELF'];?>" method="post"
enctype="application/x-www-form-urlencoded">
<fieldset>
<legend><b>Vos coordonnées</b></legend>
<table>
<tr><td>Nom : </td><td><input type="text" name="nom" size="40" maxlength="30"/>
</td></tr>
<tr><td>Prénom : </td><td><input type="text" name="prenom" size="40" maxlength="30">
</td></tr>
<tr><td>Âge : </td><td><input type="text" name="age" size="40" maxlength="2"/>
</td></tr>
<tr><td>Adresse : </td><td><input type="text" name="adresse" size="40" maxlength="60"/></td></tr>
<tr><td>Ville : </td><td><input type="text" name="ville" size="40" maxlength="40"/>
</td></tr>
<tr><td>E-mail : </td><td><input type="text" name="mail" size="40" maxlength="50"/>
</td></tr>
<tr>
<td><input type="reset" value="Effacer"></td>
<td><input type="submit" value="Envoyer"></td>
</tr>
</table>
</fieldset>
</form>
<?php
include("exemple16.2.php");
$idcom=connexpdo ('magasin','myparam');

if(!empty($_POST['nom']) && !empty($_POST['adresse']) && !empty($_POST['ville'])) ←❶
{
    $id_client="\N"; ←❷
    $nom=$idcom->quote($_POST['nom']); ←❸
    $prenom=$idcom->quote($_POST['prenom']); ←❹
    $age=$idcom->quote($_POST['age']); ←❺
    $adresse=$idcom->quote($_POST['adresse']); ←❻
    $ville=$idcom->quote($_POST['ville']); ←❾
    $mail=$idcom->quote($_POST['mail']); ←❿
    // Requête SQL
    $requete="INSERT INTO client
VALUES($id_client,$nom,$prenom,$age,$adresse,$ville,$mail)"; // pas de guillemets
    si on applique la méthode quote aux variables
    $nblignes=$idcom->exec($requete); ←❽
    if($nblignes!=1) ←❾
    {
        $mess_erreur=$idcom->errorInfo();
        echo "Insertion impossible, code", $idcom->errorCode(),$mess_erreur[2];
        echo "<script type=\"text/javascript\">
        alert('Erreur : ".$idcom->errorCode()."')</script>";
    }
    else
    {
        echo "<script type=\"text/javascript\">

```

```

        alert('Vous êtes enregistré. Votre numéro de client est :  

        ↵". $idcom->lastInsertId()."')</script>"; ←⑪  

$idcom=null;  

}  

}  

else {echo "<h3>Formulaire à compléter !</h3>";}  

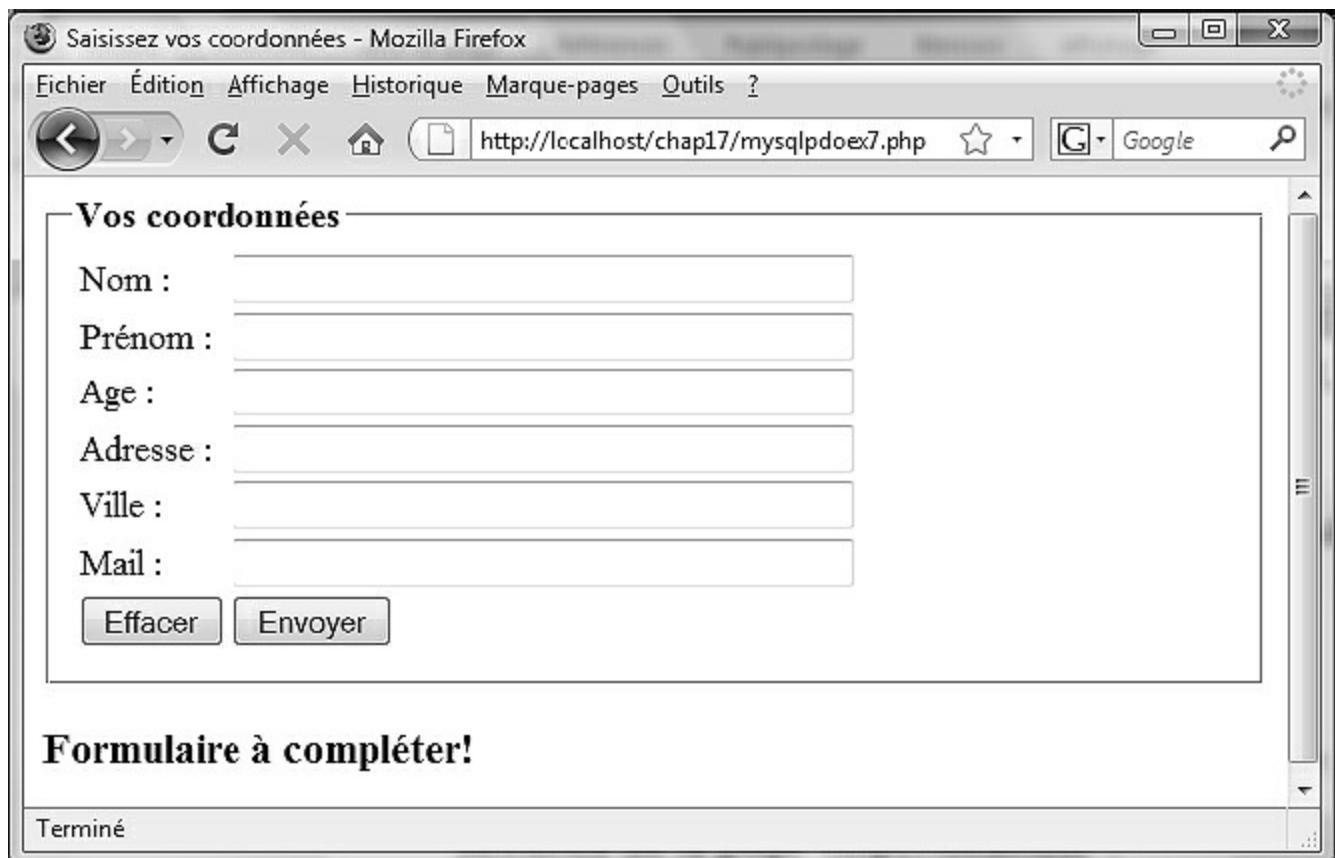
?>  

</body>  

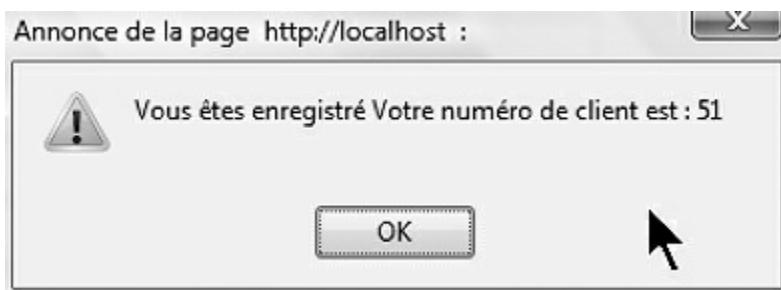
</html>

```

La figure 16-4 illustre la page de saisie et la figure 16-5 la boîte d'alerte JavaScript qui donne son identifiant au client.



**Figure 16-4**  
*Formulaire d'insertion de données*



**Figure 16-5**  
*Boîte d'alerte JavaScript donnant le numéro de client*

## Mise à jour d'une table

Un client doit pouvoir modifier ses coordonnées, son adresse de livraison ou son adresse mail par exemple. L'exemple 16-8 crée une page contenant un formulaire qui permet la saisie du code client dans une zone de texte HTML (repère 1). L'attribut `action` de l'élément `<form>` renvoie le traitement de la saisie au fichier `mysqlpdoex9.php` de l'exemple 16-9. La page créée est conforme à la figure 16-6.

## Exemple 16-8. Page de saisie des modifications

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Modifiez vos coordonnées</title>
</head>
<body>
    <form action= "exemple16.9.php" method="post" enctype=
        "application/x-www-form-urlencoded">
        <fieldset>
            <legend><b>Saisissez votre code client pour modifier vos coordonnées</b></legend>
            <table><tbody>
                <tr>
                    <td>Code client : </td>
                    <td><input type="text" name="code" size="20" maxlength="10"/></td> ← 1
                </tr>
                <tr>
                    <td>Modifier : </td>
                    <td><input type="submit" value="Modifier"/></td>
                </tr>
            </tbody></table>
        </fieldset>
    </form>
</body>
</html>
```



**Figure 16-6**  
*Page de saisie du code client*

La mise à jour des coordonnées du client est réalisée par le script de l'exemple 16-9. 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 ①). 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, quant à la seconde, elle enregistre ces mêmes données dans la base.

Lors du premier appel du fichier de l'exemple 16-9, la condition de l'instruction `if` (repère ②) est nécessairement vérifiée car la variable `$_POST['modif']` n'existe pas encore. Elle correspond à la valeur associée au bouton Modifier du formulaire qui n'est pas encore créé. Le script génère une connexion au serveur MySQL pour y lire les coordonnées actuelles du client, dont le code est contenu dans la variable `$code` issue de la page de saisie de l'exemple 16-8 (repère ③).

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 `$code` (repère ④). Cela permet de ne saisir que les modifications éventuelles de coordonnées du client, sans devoir ressaisir l'ensemble. Ces coordonnées sont lues à l'aide de la méthode `fetch()` de l'objet `$result` de type `PDOStatement` (repère ⑤). 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 ⑥).

La figure 16-7 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 ⑦).

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 ⑧). Seules les colonnes `nom`, `adresse`, `ville` et `mail` peuvent être mises à jour à l'aide de la requête suivante (repère ⑨), le reste étant forcément inchangé :

```
UPDATE client SET nom='$nom',adresse='$adresse',ville='$ville',mail='$mail'  
WHERE id_client='$code'
```

La vérification du résultat de la requête (repère ⑩) permet d'afficher une boîte d'alerte JavaScript contenant soit un message d'erreur, soit la confirmation de l'enregistrement. La page ne devant pas être une impasse, le visiteur est redirigé d'office vers la page d'accueil du site `index.html` (repères ⑪ et ⑫).

## Exemple 16-9. Mise à jour des données

```
<?php  
if(empty($_POST['code'])) {header("Location:exemple16.8.php");} ←①  
?>
```

```

<!DOCTYPE html>
<html lang="fr">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Modifiez vos coordonnées</title>
</head>
<body>
<?php
include('exemple16.2.php');
$idcom=connexpdo('magasin','myparam');
if(!isset($_POST['modif'])) ← ②
{
  $code=(integer) $_POST['code']; ← ③
  // Requête SQL
  $requete="SELECT * FROM client WHERE id_client='".$code' " ; ← ④
  $result=$idcom->query($requete);
  $coord=$result->fetch(PDO::FETCH_NUM); ← ⑤
  // Création du formulaire complété avec les données existantes ← ⑥
  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=\"$coord[2]\"/></td></tr>";
  echo "<tr><td>Âge : </td><td><input type=\"text\" name=\"age\" size=\"40\" 
  maxlength=\"2\" value=\"$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>E-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\"/>"; ← ⑦
  echo "</form>";
$result->closeCursor();
$idcom=null;

}

elseif(isset($_POST['nom']) && isset($_POST['adresse']) && isset($_POST['ville'])) ← ⑧
{
  // ENREGISTREMENT
  $nom=$idcom->quote($_POST['nom']);
  $adresse=$idcom->quote($_POST['adresse']);
  $ville=$idcom->quote($_POST['ville']);
  $mail=$idcom->quote($_POST['mail']);
  $age=(integer) $_POST['age'];
  $code=(integer) $_POST['code'];
  // Requête SQL
  $requete="UPDATE client SET nom=$nom,adresse=$adresse,ville=$ville,mail=

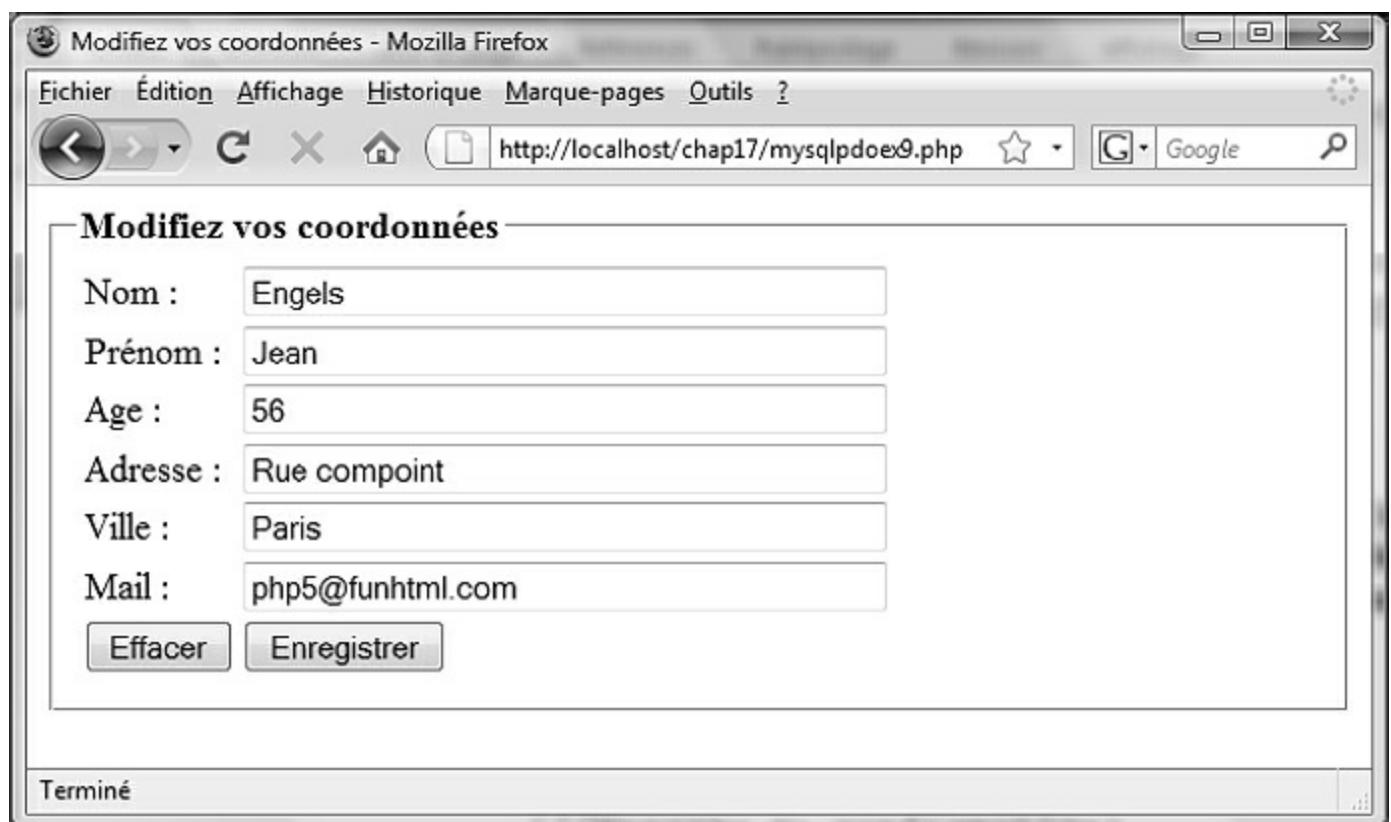
```

```

$mail,age=$age WHERE id_client=$code"; ← 9
$result=$idcom->exec($requete);

if($result!=1) ← 10
{
    echo "<script type=\"text/javascript\">
        alert('Erreur : ".$idcom->errorCode()."')</script>"; ← 11
}
else
{
    echo "<script type=\"text/javascript\"> alert('Vos modifications sont
        enregistrées');window.location='exemple16.8.php';</script>"; ← 12
}
$idcom=null;
}
else
{
    echo "Modifiez vos coordonnées !";
}
?>
</body>
</html>

```



**Figure 16-7**  
*Formulaire de saisie des coordonnées créé dynamiquement*

## Recherche dans la base

Un site de commerce en ligne doit permettre à ses visiteurs et futurs clients d'effectuer des recherches dans la base de données afin d'accéder plus rapidement à l'information correspondant au produit recherché. Il doit en outre permettre d'effectuer des statistiques marketing à l'usage du propriétaire du site. Ces éléments concernent aussi bien les sites de commerce en ligne que les annuaires et moteurs de recherche des sites de contenu.

L'exemple 16-10 crée un formulaire classique permettant de saisir un mot-clé et de choisir le type de tri des résultats. Les critères de tri selon le prix, la catégorie ou l'identifiant d'article sont affichés sous forme de liste déroulante. Le choix de l'ordre croissant ou décroissant s'effectue au moyen de deux boutons radio possédant le même attribut `name`, ce qui les rend exclusifs l'un de l'autre.

Le script contrôle d'abord que le visiteur a saisi un mot-clé dans le formulaire en vérifiant que la variable `$_POST['motcle']` n'est pas vide (repère 1). Il récupère ensuite le mot-clé, la catégorie, le critère de tri et l'ordre d'affichage respectivement dans les variables `$motcle`, `$categorie`, `$ordre` et `$tri` (repères 2 à 5).

Si la catégorie choisie est "tous", la partie de la commande `WHERE` concernant cette catégorie est vide. Pour les autres choix, elle est égale à "`AND categorie=$categorie`" (repère 6). La requête de sélection suivante (repère 7) est alors créée par le code :

```
"SELECT id_article AS 'Code article',
designation AS 'Description',
prix, categorie AS 'Catégorie'
FROM article WHERE lower(designation) LIKE '%$motcle%'".$reqcategorie.
"ORDER BY $tri $ordre";
```

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` (repère 8). La lecture des résultats et l'affichage de toutes les lignes retournées sont réalisés avec la méthode `fetch()` de la même manière que dans les exemples précédents (repère 9). La figure 16-8 montre la page créée après la recherche du mot-clé `portable`.

## Exemple 16-10. Page de recherche d'articles

```
<!DOCTYPE html>
<html 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-clé : </td>
<td><input type="text" name="motcle" size="40" maxlength="40" /></td>
</tr>
<tr>
<td>Dans la catégorie : </td>
<td>
<select name="categorie">
<option value="tous">Tous</option>
<option value="vidéo">Vidéo</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">Catégorie</option>
<option value="id_article">Code</option>
</select>
</td>
</tr>
<tr><td>En ordre : </td>
<td>Croissant<input type="radio" name="ordre" value="ASC" checked="checked"/>
 Décroissant<input type="radio" name="ordre" value="DESC" />
</td> </tr>
<tr><td>Envoyer</td><td><input type="submit" name="" value="OK"/></td> </tr>
</tbody>
</table>
</fieldset>
</form>
<?php
if(!empty($_POST ['motcle'])) ←❶
{
    include("exemple16.2.php");
    $motcle=strtolower($_POST ['motcle']); ←❷
    $categorie=($_POST ['categorie']); ←❸
    $ordre=($_POST ['ordre']); ←❹
    $tri=($_POST ['tri']); ←❺
    // Requête SQL
    $reqcategorie=($_POST['categorie']=="tous")?"":"AND categorie='".$categorie."'";
    $requete="SELECT id_article AS 'Code article',designation AS 'Description',
 prix,categorie AS 'Catégorie' FROM article WHERE lower(designation)
 LIKE '%$motcle%'.$reqcategorie."ORDER BY $tri $ordre"; ←❻
    $idcom=connexpdo('magasin','myparam');
    $result=$idcom->query($requete); ←❼
    if(!$result) ←❽
    {
        echo "Lecture impossible";
    }
    else

```

```

{
    $nbcoll=$result->columnCount();
    $nbart=$result->rowCount();
    if($nbart==0)
    {
        echo "<h3>Il n'y a aucun article correspondant au mot-clé : $motcle</h3>";
        exit;
    }
    $ligne=$result->fetch(PDO::FETCH_ASSOC); // Tableau associatif
    $titres=array_keys($ligne);
    $ligne=array_values($ligne);
    // print_r($titres);
    echo "<h3>Il y a $nbart articles correspondant au mot-clé : $motcle</h3>";
    // Affichage des titres du tableau
    echo "<table border=\"1\"> <tr>";
    foreach($titres as $val)
    {
        echo "<th>", htmlentities($val), "</th>";
    }
    echo "</tr>";
    // Affichage des valeurs du tableau
    do
    {
        echo "<tr>";
        foreach($ligne as $donnees)
        {
            echo "<td>", $donnees, "</td>";
        }
        echo "</tr>";
    }
    while($ligne=$result->fetch(PDO::FETCH_NUM));
}
echo "</table>";
$result->closeCursor();
$idcom=null;
}
?>
</body>
</html>

```

Rechercher un article dans le magasin - Mozilla Firefox

Fichier Édition Affichage Historique Marque-pages Outils ?

Mot-clé: portable

Dans la catégorie : Tous

Trier par : Prix

En ordre: Croissant  Décroissant

Envoyer

Il y a 3 articles correspondant au mot-clé : portable

Code article	Description	prix	Catégorie
DEL30	Portable Dell X300	1715.00	informatique
SAX15	Portable Samsung X15 XVM	1999.00	informatique
SOXMP	PC Portable Sony Z1-XMP	2399.00	informatique

Terminé

**Figure 16-8**  
*Formulaire de recherche et résultats obtenus*

## 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 WHERE 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 `PDOSatement` qui représente la requête préparée et qui est noté `$reqprep` 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 `PDOSatement` ou encore la méthode `bindValue()` 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 `PDOSatement` 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 :

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

ou encore :

```
$reqprep->bindColumn(prenom,$prenom)
```

Dans ce cas, la valeur de la colonne 1 désignée dans la requête sera contenue dans la variable `$prenom`, et ainsi de suite.
5. Les lignes de résultats sont obtenues en appliquant une des méthodes vues précédemment, comme `fetch()` ou `fetchAll()`, à l'objet `PDOSatement` représentant la requête préparée, désignée ici par `$reqprep`. Une ou plusieurs boucles, selon les cas, permet d'afficher les résultats en utilisant les variables créées à l'étape 4.
6. Libérer la mémoire occupée par l'objet `PDOSatement` avec la méthode `closeCursor()`.

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 ① et ②) dans le but de trouver tous les clients habitant cette ville et dont l'identifiant est supérieur ou égal à la valeur saisie. Ces informations sont tout d'abord récupérées dans les variables `$ville` et `$id_client` (repères ③ et ④). Après la connexion à la base, nous écrivons la requête préparée avec la méthode `prepare()` (repère ⑤). La liaison des paramètres aux variables `$ville` et `$id_client` est effectuée en appelant les

méthodes `bindParam()` et `bindValue()` avec les arguments qui précisent le type de variable (repères 6 et 7). La requête est ensuite exécutée avec la méthode `execute()` sans paramètre (repère 8). Les résultats des colonnes `prenom` et `nom` de la table `client` sont ensuite liés aux variables `$prenom` et `$nom` (repères 9 et 10). Une boucle `while` parcourt l'ensemble des résultats et permet l'affichage de toutes les lignes (repère 11).

Notez que, comme nous le venons de le voir en exposant la méthode générale, pour obtenir un résultat identique, les trois lignes de liaison des paramètres pourraient être remplacées par les lignes suivantes :

```
//*****Liaison des paramètres
$reqprep->execute(array(':ville' => $ville, ':id_client' => $id_client));
```

## Exemple 16-11. Utilisation des requêtes préparées

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Recherche de client</title>
<style type="text/css" >
    div{font-size: 20px; }
</style>
</head>
<body>
<form method="post" action="php echo $_SERVER['PHP_SELF'];?&gt;"&gt;

    &lt;fieldset&gt;
        &lt;legend&gt;Recherche de client&lt;/legend&gt;
        &lt;label&gt;Ville &lt;/label&gt;&lt;input type="text" name="ville" /&gt;&lt;br /&gt;&lt;br /&gt; ← 1
        &lt;label&gt;Id_client&lt;/label&gt;&lt;input type="number" step="1" name="id_client" /&gt;&lt;br /&gt; ←
    2
        &lt;input type="submit" value="Envoyer" /&gt;
    &lt;/fieldset&gt;
&lt;/form&gt;
&lt;/body&gt;
&lt;/html&gt;
&lt;?php
if(isset($_POST['ville']) &amp;&amp; isset($_POST['id_client']))
{
    $ville=strtolower($_POST['ville']); ← 3
    $id_client=$_POST['id_client']; ← 4
    include("exemple16.2.php");
    $idcom=connexpdo('magasin','myparam');
    $reqprep=$idcom-&gt;prepare("SELECT prenom,nom FROM client WHERE lower(ville)=
    ← 5
    :ville AND id_client=&gt; :id_client ");
    //*****Liaison des paramètres
    $reqprep-&gt;bindValue(':ville',$ville,PDO::PARAM_STR); ← 6
    $reqprep-&gt;bindParam(':id_client',$id_client,PDO::PARAM_INT); ← 7
    $reqprep-&gt;execute(); ← 8
    //*****Liaison des résultats à des variables
    $reqprep-&gt;bindColumn('prenom', $prenom); ← 9
    $reqprep-&gt;bindColumn('nom', $nom); ← 10</pre

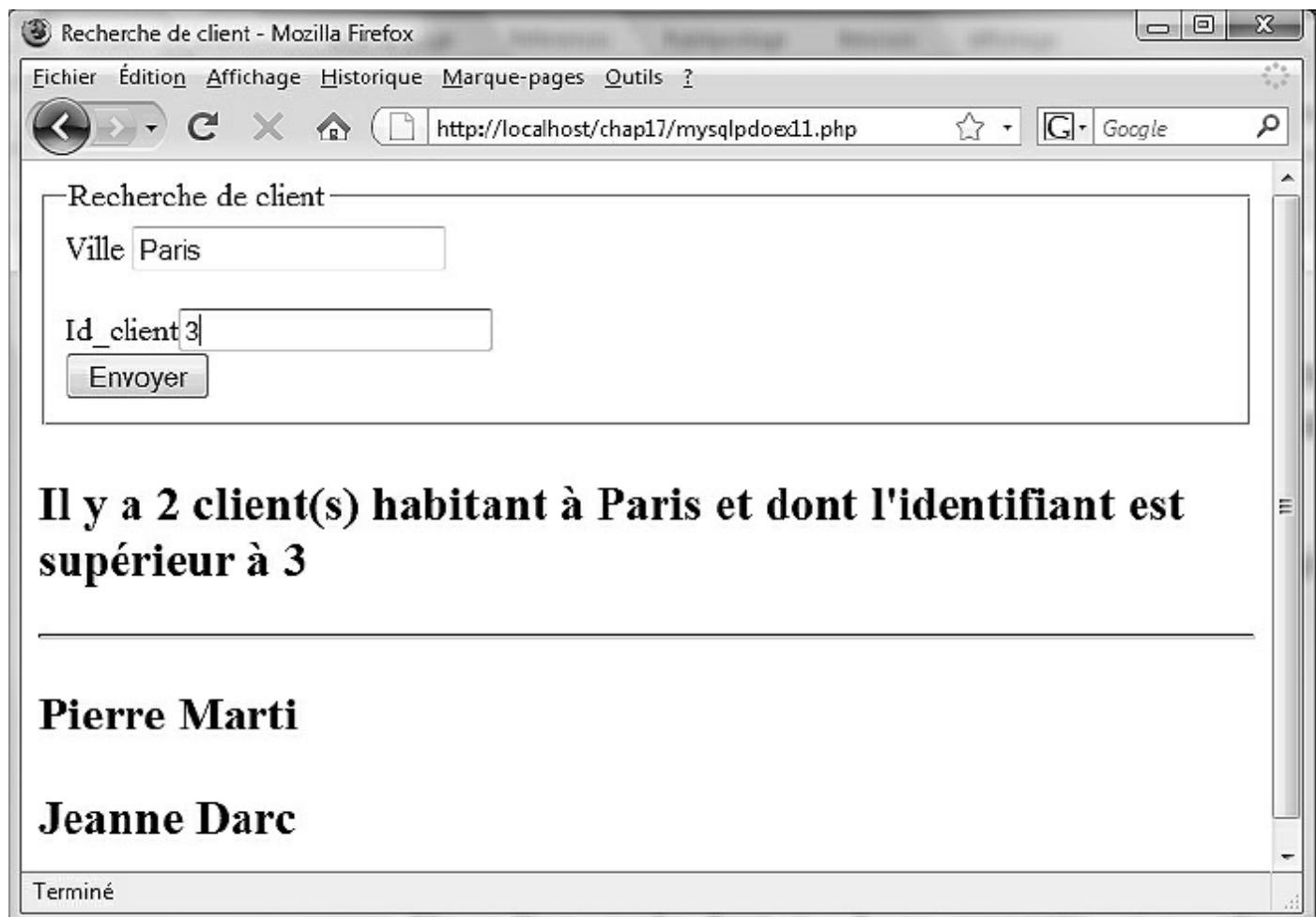
```

```

//*****Affichage
echo "<div><h3>Il y a ", $reqprep->rowCount() , " client(s) habitant à
",ucfirst($ville),
" et dont l'identifiant est supérieur ou égal à $id_client</h3><hr />";
while($result=$reqprep->fetch(PDO::FETCH_BOUND)) ←⑪
{
    echo "<h3> $prenom $nom</h3>";
}
echo "</div>";
$reqprep->closeCursor();
$idcom=null;
}
?>

```

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 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 insertions 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()`. Nous pouvons vérifier que nous sommes en mode transaction en appelant la méthode `inTransaction()` qui retourne `TRUE` ou `FALSE` selon que la condition est vérifiée ou non.

Dans l'exemple 16-12, nous illustrons la procédure à suivre pour effectuer deux requêtes `INSERT` sur la table `article` (repères ③ et ④) qui doivent être impérativement réalisées toutes les deux. Après le déclenchement de la transaction avec la méthode `beginTransaction()` (repère ①), nous vérifions que ce mode est bien actif (repère ②). Nous envoyons ensuite ces requêtes au serveur avec la méthode `exec()` qui retourne le nombre de lignes affectées par chaque requête. La variable `$verif` cumule donc le nombre total d'insertions (repères ⑤ et ⑥). Si ce nombre vaut 2 (repère ⑦), nous pouvons valider la transaction avec la méthode `commit()` (repère ⑧), sinon elle est annulée avec la méthode `rollBack()` (repère ⑨), un message d'erreur est affiché (repères ⑩ et ⑪) et le tableau indicé retourné par la méthode `errorInfo()` contenant le code d'erreur pour l'indice 0 et le message d'erreur en clair pour l'indice 2. Pour tester l'efficacité du script, il suffit d'introduire une erreur dans la seconde requête en écrivant, par exemple, le nom de table inexistant "clients" à la place de "client", pour constater avec phpMyAdmin qu'aucune insertion n'est effectuée alors que la première est valable.

## Exemple 16-12. Insertions avec transaction

```
<?php
include('exemple16.2.php');
$idcom=connexpdo('magasin','myparam');
$idcom->beginTransaction(); ←①
if($idcom->inTransaction()){echo "Début transaction";;} ←②
echo "<hr />";
```

```

$requete1="INSERT INTO client(id_client,nom,prenom,age,adresse,ville,mail) VALUES
(NULL , 'Spencer', 'Marc', '32', 'rue du blues', 'New Orleans',
'marc@spencer.be');"; ← 3
echo $requete1,"<hr />";
$requete2="INSERT INTO client(id_client,nom,prenom,age,adresse,ville,mail) VALUES
(NULL ,
'Spancer', 'Diss', '89', 'Metad Street', 'New York', 'diss@metad.fr');"; ← 4
echo $requete2,"<hr />";
// Insertions des données
$verif= $idcom->exec($requete1); ← 5
$verif+= $idcom->exec($requete2); ← 6
if($verif==2 ) ← 7
{
    $idcom->commit(); ← 8
    echo "Insertions réussies de $verif lignes<br />";
}
else
{
    $idcom->rollBack();← 9
    $tab_erreur=$idcom->errorInfo();
    echo "Insertions annulées. Erreur n° :",$tab_erreur[0],"<br />";← 10
    echo "Info : ",$tab_erreur[2];← 11
}
?>

```

## Mémo des méthodes

### *Classe PDO*

---

boolean beginTransaction ( void )

Commence une transaction et retourne `TRUE` en cas de réussite ou `FALSE` sinon.

---

boolean commit ( void )

Valide une transaction.

---

objetPDO \_\_construct ( string \$dsn [, string \$username [, string \$password ]] )

Constructeur de la classe PDO. Crée un objet représentant la connexion.

---

string errorCode ( void )

Retourne un code d'erreur.

---

array errorInfo ( void )

Retourne un tableau contenant un code d'erreur (indice 0) et un message d'erreur (indice 2).

---

integer exec ( string \$requete )

Exécute une requête qui ne retourne pas de résultat et retourne le nombre de lignes affectées.

---

boolean inTransaction ()

Retourne `TRUE` si le mode transaction est activé et `FALSE` sinon

---

string lastInsertId ()

Retourne l'identifiant de la colonne dont l'attribut est déclaré `AUTO_INCREMENT`.

---

objet PDOStatement prepare ( string \$requete )

Crée une requête préparée et retourne un objet `PDOStatement`.

---

```
objet PDOStatement query ( string $requete )
```

Envoie une requête au serveur et retourne un objet résultat.

---

```
string quote( string $chaine )
```

Protège les caractères spéciaux d'une chaîne.

---

```
boolean rollBack ( void )
```

Annule la transaction courante.

## ***Classe PDOStatement***

---

```
boolean bindColumn ( divers $colonne, divers $var [, integer type] )
```

Lie une colonne désignée par son numéro ou son nom à une variable ; le type peut être `PDO::PARAM_STR` pour une chaîne ou `PDO::PARAM_INT` pour un nombre.

---

```
boolean bindParam (divers $parametre, divers $variable [, int type] )
```

Lie un paramètre nommé ou interrogatif à une variable. La constante type prend les mêmes valeurs que ci-dessus.

---

```
integer columnCount ( void )
```

Retourne le nombre de lignes d'un résultat.

---

```
string errorCode ( void )
```

Retourne un code d'erreur.

---

```
array errorInfo ( void )
```

Retourne un tableau contenant les informations sur l'erreur en cours.

---

```
bool execute ( [array $parametre] )
```

Exécute une requête préparée ; le paramètre peut contenir les liaisons des paramètres nommés ou interrogatifs.

---

```
divers fetch ( [integer $style] )
```

Retourne une ligne de résultat sous forme de tableau indicé ou associatif. `style` est une constante qui vaut `FETCH_NUM` (indicé), `FETCH_ASSOC` (associatif), `FETCH_OBJ` (objet), `FETCH_BOTH` (indicé et associatif) ou `FETCH_LAZY` (indicé, associatif et objet).

---

```
array fetchAll ( [integer $style] )
```

Retourne l'ensemble des lignes de résultat sous forme de tableau multidimensionnel indicé ou associatif. Idem que ci-dessus pour le paramètre.

---

```
object fetchObject ()
```

Retourne la ligne suivante de résultat dans un objet dont les propriétés ont pour nom ceux des champs et contiennent les valeurs de ceux-ci.

---

```
int rowCount ( void )
```

Retourne le nombre de lignes affectées par la dernière requête.

## ***Classe PDOException***

---

```
objet PDOException __construct()
```

Crée un objet exception.

---

```
string getCode()
```

Retourne le code d'erreur.

---

```
string getFile()
```

Retourne le nom du fichier dans lequel s'est produite l'erreur.

---

```
integer getLine()
```

Retourne le numéro de ligne du script où s'est produite l'erreur.

---

```
string getMessage()
```

Retourne le message d'erreur.

## 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 PDO.

### Exercice 1

Créez un script permettant d'afficher le contenu de la table `modele` dans un tableau HTML. 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 HTML.

### Exercice 5

Créez un formulaire de recherche permettant de retrouver tous les véhicules possédés par une personne donnée. Affichez les résultats sous forme de tableau HTML.

### Exercice 6

Réécrivez entièrement le code de l'exercice 5 en récupérant tous les résultats dans des objets et en manipulant leurs propriétés.

### Exercice 7

Refaire l'exercice 4 en utilisant une requête préparée.

## **Exercice 8**

Refaire l'exercice 3 en utilisant une transaction pour s'assurer que les données sont bien insérées dans les différentes tables.

## La base SQLite

---

SQLite était l'une des grandes nouveautés de PHP 5. Il s'agit d'une base de données relativement puissante. Contrairement à MySQL ou PostgreSQL, elle ne nécessite pas l'installation d'un serveur de base de données en plus d'un serveur PHP car elle est incorporée à PHP comme l'est le module standard.

D'après ses concepteurs et les tests réalisés, SQLite se révèle beaucoup plus rapide que MySQL et PostgreSQL pour les opérations de lecture. Les opérations d'écriture perdent un peu de cet avantage car dans ce cas la base est verrouillée dans son intégralité, même si l'opération ne concerne qu'une table.

Dans l'état actuel de son développement, SQLite n'est intéressant que pour les applications qui ne nécessitent pas un grand nombre d'accès concurrents en écriture à la base. Il y a peu de chance, par exemple, qu'un site comme Google l'adopte, mais elle pourrait suffire à des entreprises de taille moyenne et particulièrement pour des applications installées sur des terminaux mobiles.

### Caractéristiques générales

Contrairement à de nombreux autres SGBD, comme MySQL, SQLite crée la base directement sur le disque dur de votre serveur. À chaque base correspond donc un seul fichier, quel que soit le nombre de tables qu'elle contient. Dans ce fichier sont à la fois enregistrées la structure des tables et les données (voir figure 17-1). Comme vous le voyez, ce fichier est inutilisable tel quel. Pour cette raison, la modification de la structure d'une table est impossible en utilisant la commande `ALTER TABLE`. Vous devez d'abord sauvegarder les données, effacer la table puis la recréer avec une nouvelle structure.

```

Fichier Edition Format Affichage ?
SQLITE format 3 @ .
-â%ç - 3 4É,æ 3 æ
    0|• ,3tablepratiquepratique-CREATE TABLE pratique (
    id_personne INTEGER NOT NULL,
    id_sport INTEGER NOT NULL ,
    niveau TINYINT,
    PRIMARY KEY (id_personne,id_sport))/--C
indexsqlite_autoindex_pratique_1pratique•,4|H ictablesportsportCREATE TABLE sport(
    id_sport INTEGER PRIMARY KEY AUTOINCREMENT,
    design VARCHAR( 30 ) UNIQUE NOT NULL)•-i= indexsqlite_autoindex_sport_1sport|  JP,--+
+ Ytablessqlite_sequencessqlite_sequenceCREATE TABLE sqlite_sequence
(name,seq) • fitablepersonnepersonne.CREATE TABLE personne (
    id_personne INTEGER PRIMARY KEY AUTOINCREMENT ,
    nom VARCHAR( 20 ) NOT NULL ,
    prenom VARCHAR( 20 ) ,
    depart INTEGER( 2 ) NOT NULL,
    mail VARCHAR(50) NOT NULL)• à JibJibÄLlf

```

- + 'EngelsJeanKaazert@dfg.fr' - H )  
AbbassAlitali@abbass.org à + 'speakJanjan@azerty.fr' - + 'spencerMarcKmarc@spen.org' -  
à à

**Figure 17-1**  
*Structure d'un fichier de base SQLite*

SQLite supporte l'essentiel des fonctionnalités du langage SQL.

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 concurrents à la base.

Dans sa version actuelle, la version 3, 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.

Dans la gestion de la base, SQLite ne gère que les types TEXT, INTEGER, REAL, NUMERIC, BLOB, NONE et NULL.

Si vous créez des tables à partir d'un code SQL standard, par exemple à partir d'un fichier .sql récupéré d'une table MySQL avec PhpMyAdmin, SQLite va gérer les déclarations comme indiqué dans le tableau 17-1.

**Tableau 17-1 – Gestion des types déclarés par SQLite**

Type déclaré dans une requête CREATE TABLE	Type créé dans SQLite3
--	------------------------

INT	INTEGER
CHARACTER (20)	
VARCHAR (255)	
VARYING CHARACTER (255)	
NCHAR (55)	
NATIVE CHARACTER (70)	TEXT
NVARCHAR (100)	
TEXT	
CLOB	
BLOB	NONE
REAL	
DOUBLE	
DOUBLE PRECISION	REAL
FLOAT	
NUMERIC	
DECIMAL (10, 5)	
BOOLEAN	
DATE	
DATETIME	
NULL	NULL

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 suit :

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

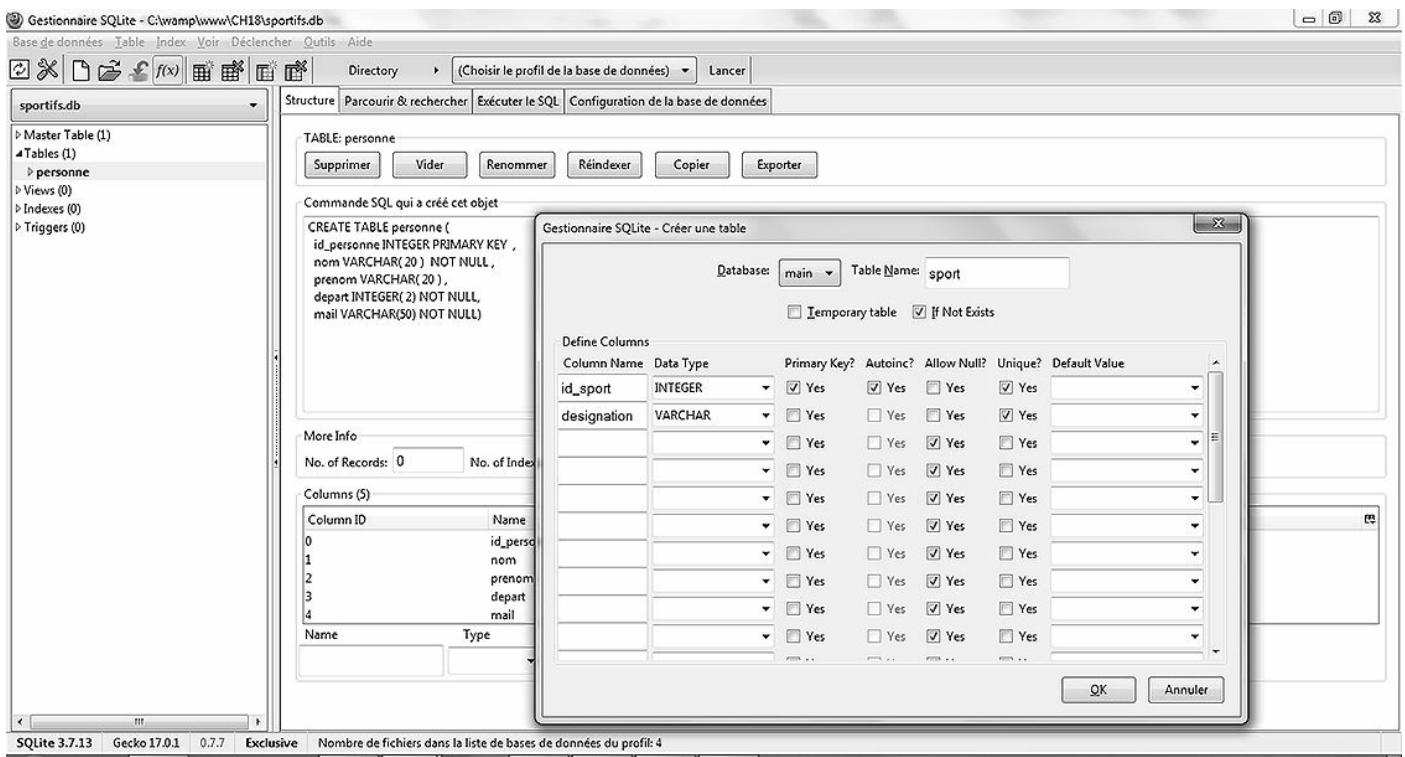
L'insertion de la valeur `NULL` dans la colonne `id` permet cette incrémentation automatique.

## *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.

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>). En local, vous pouvez utiliser SQLiteManager en installant une extension de Firefox téléchargeable à partir du menu Outils>Modules complémentaires.

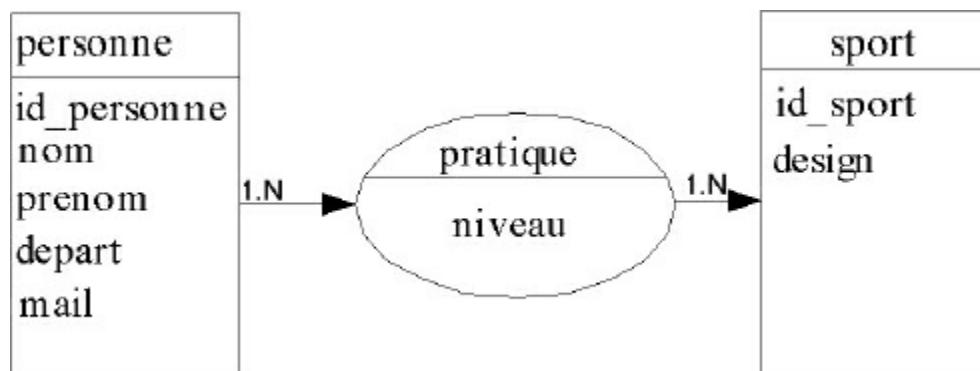
La figure 17-2 montre la page d'accueil de SQLiteManager dans Firefox, à partir de laquelle vous pouvez créer une base et les tables qui la composent. L'application sur laquelle reposent les exemples de ce chapitre a pour but de créer un site de contact entre sportifs partageant la même passion et leur permettant de trouver des partenaires éventuels. Ce modèle pourrait être facilement transposé à d'autres applications pratiques. Vous aurez à développer entièrement cette application au chapitre 19.



**Figure 17-2**  
*La page d'accueil de SQLiteManager*

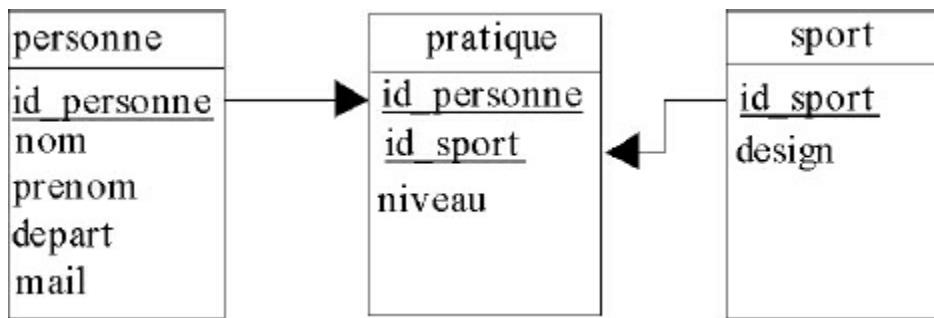
Le modèle conceptuel de données (MCD) de la base nommée `sportifs` est illustré à la figure 17-3. Il comporte les contraintes suivantes :

- L'entité `personne` regroupe les données personnelles d'un visiteur inscrit.
- L'entité `sport` contient les caractéristiques de chaque sport.
- L'association `pratique` relie ces deux entités.
- Une personne pratique un ou plusieurs sports. La cardinalité du côté de l'entité `personne` est donc 1:N.
- Un sport est pratiqué par une ou plusieurs personnes. La cardinalité du côté de l'entité `sport` est donc 1:N.



**Figure 17-3**  
*Le MCD de la base sportifs*

Le modèle logique de données (MLD) correspondant, en application des règles vues au chapitre 13, est représenté à la figure 17-4. Chaque personne peut pratiquer de 1 à  $N$  sports, et un sport peut être pratiqué par 1 à  $N$  personnes.



**Figure 17-4**  
*Le MLD de la base sportifs*

La base `sportifs` comprend donc les trois tables suivantes :

- La table `personne`, qui contient les coordonnées de chaque personne inscrite sur le site (nom, prénom, département, adresse e-mail). L'identifiant auto-incrémenté `id_personne` est la clé primaire de la table.
- La table `sport`, qui contient le nom de chaque sport pratiqué contenu dans l'attribut `design`. Elle sera enrichie par les visiteurs du site en fonction de leurs besoins. Chaque sport a un identifiant unique auto-incrémenté `id_sport`, qui est la clé primaire de la table.
- La table `pratique` est la représentation de l'association entre les tables `personne` et `sport`. Sa clé primaire est constituée des clés primaires des tables `personne` et `sport`. On obtient ainsi l'association entre une personne et un sport. L'attribut `niveau` contient le niveau de pratique de chaque utilisateur pour un sport donné.

Pour créer les tables, vous utilisez SQLiteManager. Complétez les zones de saisie conformément à la figure 17-2, puis cliquez sur Enregistrer. Un fichier nommé `sportifs` est alors créé dans le répertoire choisi à partir du menu Directory. La création des tables se déroule ensuite selon la même démarche qu'avec phpMyAdmin. Il faut d'abord saisir le nom de la table et son nombre de colonnes, puis une nouvelle page permet la définition des caractéristiques de chaque colonne.

## Méthodes d'accès à SQLite

L'extension `sqlite` de PHP est devenue obsolète ; par conséquent un accès procédural n'est désormais plus possible. L'accès objet à une base SQLite3 peut s'effectuer de deux manières.

- En utilisant les méthodes de l'extension `php_sqlite3` qui doit être installée sur le serveur.
- En utilisant PDO, comme nous l'avons vu au chapitre 16 avec MySQL.

# La méthode SQLite3

L'extension `sqlite3` offre trois classes qui permettent de créer des objets dont les méthodes gèrent différentes situations.

- La classe `SQLite3` permet la connexion à la base et l'envoi des requêtes.
- La classe `SQLite3stmt` permet d'utiliser des requêtes préparées.
- La classe `SQLite3Result` permet la lecture des résultats des requêtes de sélection.

Pour utiliser une base SQLite, la démarche est identique à celle que nous avons suivie avec MySQL. Elle consiste à effectuer les étapes suivantes.

1. Ouverture de la base, qui s'apparente plutôt à une ouverture de fichier en créant un objet `SQLite3`.
2. Envoi des requêtes SQL à effectuer sur la base.
3. Récupération de la valeur de retour de la fonction de requête, qui est soit une valeur booléenne pour contrôler que la requête est bien exécutée – par exemple, `CREATE TABLE`, `INSERT`, `UPDATE` –, soit un objet `SQLite3Result` quand la requête SQL retourne des données – par exemple, `SELECT`.
4. Utilisation le cas échéant des fonctions spécialisées de récupération des résultats, et création d'un affichage dynamique approprié avec PHP et HTML.
5. Suppression des résultats et fermeture de la base.

## Connexion à la base

Pour se connecter à une base SQLite, vous devez créer un objet instance de la classe `SQLite3` en appelant son constructeur selon la syntaxe suivante :

```
$objdb= new SQLite3('base', ['drapeau']);
```

dans laquelle le paramètre `'base'` correspond au nom de la base à ouvrir ou à créer, par exemple `"mabase.db"` ou `"mabase.sqlite"`. Si l'on indique le mot-clé ":memory:" en premier paramètre, la base est créée en mémoire vive et donc effacée quand elle est fermée. Le second paramètre est facultatif, il s'agit d'une constante qui prend les valeurs suivantes :

- `SQLITE3_OPEN_READONLY` qui ouvre en lecture seule ;
- `SQLITE3_OPEN_READWRITE` qui ouvre en lecture/écriture ;
- `SQLITE3_OPEN_CREATE` qui crée, puis ouvre la base si elle n'existe pas encore.

Si l'objet `SQLite3` existe toujours mais que la connexion a été fermée, il est possible avec cet objet d'appeler la méthode `open()` avec les mêmes paramètres que ceux du constructeur de l'objet pour rétablir la connexion. Cette méthode retourne un booléen (`TRUE` ou `FALSE`) afin de s'assurer que l'opération est réussie.

## *Envoi des requêtes*

L'étape suivante consiste à envoyer des requêtes à la base à l'aide de commandes SQL standards. Les connaissances que vous avez acquises au chapitre 14 sont mises en pratique avec SQLite, aussi bien qu'avec MySQL. Pour envoyer une requête à la base ouverte, il faut appeler le plus souvent la méthode `query()` dont la syntaxe est la suivante :

```
| $result = $objdb->query($requete);
```

La requête est une chaîne de caractères qui doit être délimitée par des guillemets simples ou doubles, ou être contenue dans une variable.

Avec cette méthode, la valeur renournée est un objet de type `SQLite3Result` dont les méthodes vont nous permettre de lire les valeurs des champs. Nous utiliserons typiquement cette méthode pour des requêtes `SELECT`.

Pour les requêtes qui ne retournent pas de résultat comme `INSERT` ou `UPDATE`, nous pouvons appeler la méthode `exec()` dont la syntaxe est la suivante :

```
| $test=$objdb->exec($requete);
```

Elle retourne simplement un booléen qui permet de vérifier la bonne exécution.

En dernier lieu, si l'on ne veut obtenir qu'un seul résultat, il suffit d'appeler la méthode `querySingle()` de l'objet `SQLite3` avec la syntaxe :

```
| $result=$objdb->querySingle($requete,TRUE | FALSE);
```

Elle va retourner la valeur de la première colonne du résultat si le second paramètre vaut `FALSE` (valeur par défaut) ou toute la première ligne dans un tableau associatif dont les clés sont les noms de champs de la table.

Si une requête échoue, nous pouvons récupérer le code d'erreur généré avec la méthode `lastErrorCode()` et le message d'erreur correspondant avec `lastErrorMsg()` de l'objet `SQLite3` afin de les afficher.

Après avoir envoyé une requête à la base et récupéré éventuellement un résultat, nous pouvons fermer la connexion avec la méthode `close()` de l'objet `SQLite3` selon le modèle suivant :

```
| $objdb->close();
```

Dans ce cas, l'objet existe toujours mais il n'est plus possible d'effectuer des opérations sur la base. Pour y accéder de nouveau, il faut appeler la méthode `open()` étudiée précédemment.

L'exemple 17-1 donne un modèle de script de connexion à une base SQLite3.

### **Exemple 17-1. Connexion et envoi de requête SQL**

```
| <?php  
| $objdb=new SQLite3("sportifs.db");  
| $requete = "SELECT * FROM personne";
```

```

if($result=$objdb->query($requete))
{
    echo "La requête est réalisée";
    // Lecture des résultats
    var_dump($result);
}
else
{
    echo "Erreur n° :", $objdb->lastErrorCode(), "...", $objdb->lastErrorMsg();
}
$objdb->close();
?>

```

Si l'on ne dispose pas de l'interface SQLiteManager, il est possible de créer les tables de la base à l'aide d'un script. C'est le but de l'exemple 17-2 qui envoie les requêtes SQL de création des trois tables en utilisant la méthode `exec()` (repères ①, ③ et ⑤). Si une requête n'est pas exécutée, nous affichons un message d'erreur (repères ②, ④ et ⑥).

## Exemple 17-2. Création des tables avec un script

```

<?php
if ($objdb = new SQLite3('sportifs.db'))
{
    echo "La base sportifs est ouverte <br />";
    //*****
    // Création de la table personne
    $req_personne="CREATE TABLE IF NOT EXISTS personne (
        id_personne INTEGER PRIMARY KEY AUTOINCREMENT,
        nom VARCHAR(20) NOT NULL,
        prenom VARCHAR(20),
        depart INTEGER(2) NOT NULL,
        mail VARCHAR(50) NOT NULL)";
    if( $objdb->exec($req_personne)) echo "La table personne est créée<br />"; ←①
    else{echo $objdb->lastErrorMsg();} ←②
    //*****
    // Création de la table sport

    $req_sport="CREATE TABLE IF NOT EXISTS sport(
        id_sport INTEGER PRIMARY KEY AUTOINCREMENT,
        design VARCHAR(30) UNIQUE NOT NULL)";
    if($objdb->exec($req_sport)) echo "La table sport est créée<br />"; ←③
    else{echo $objdb->lastErrorMsg();} ←④
    //*****
    // Création de la table pratique
    $req_pratique="CREATE TABLE IF NOT EXISTS pratique (
        id_personne INTEGER NOT NULL,
        id_sport INTEGER NOT NULL,
        niveau TINYINT,
        PRIMARY KEY (id_personne,id_sport))";
    if($objdb->exec($req_pratique)) echo "La table pratique est créée<br />"; ←⑤
    else{echo $objdb->lastErrorMsg();} ←⑥
    $objdb->close() ;
}
else
{

```

```
| echo "ERREUR de connexion";
| }
| ?>
```

## Insertion de données

L'insertion de données a déjà été abordée au chapitre 15 avec MySQL. Elle ne présente pas de difficultés particulières dans SQLite. Vous pouvez donc utiliser de la même façon la commande SQL `INSERT` pour procéder à une insertion. Les données insérées dans la base proviennent des saisies effectuées par les visiteurs lors de leur inscription sur le site. Vous retrouvez ici le code de création d'un formulaire qui vous est familier. La récupération des valeurs côté serveur s'effectue également à l'aide de la variable `$_POST`, qui est un tableau superglobal accessible dans tous les scripts.

Dans les tables qui comportent une clé primaire auto-incrémentée, il est possible, après l'envoi de la requête d'insertion, de récupérer la valeur de la clé en utilisant la méthode `lastInsertRowID()` de la classe `SQLite3`, dont la syntaxe est la suivante :

```
| $num=$objdb->lastInsertRowID();
```

Elle retourne un entier, la valeur de la clé primaire qui vient d'être insérée.

Les chaînes saisies dans le formulaire peuvent contenir des caractères spéciaux susceptibles de poser problème lors de leur incorporation aux chaînes de requête SQL. Pour protéger ces caractères, il faut appeler la méthode `escapeString()`, selon la syntaxe suivante :

```
| $chaine=$objdb->escapeString(string $chaine)
```

Elle retourne une chaîne dans laquelle les caractères spéciaux sont précédés du caractère d'échappement antislash (\).

L'exemple 17-3 réalise l'insertion des coordonnées d'un visiteur saisies dans un formulaire similaire à celui utilisé dans l'exemple 15-7 avec MySQL objet. La gestion des données saisies est attribuée au script lui-même dans l'attribut `action` de l'élément `<form>` (repère ①). Lors de son inscription, le visiteur doit saisir son nom, son prénom, son département et son adresse e-mail dans des zones de saisie nommées respectivement `nom`, `prenom`, `depart` et `mail`. Le script récupère les valeurs saisies dans les variables `$nom`, `$prenom`, `$depart` et `$mail` à partir des variables `$_POST['nom']`, `$_POST['prenom']`, `$_POST['depart']` et `$_POST['mail']` (repères ② à ⑤).

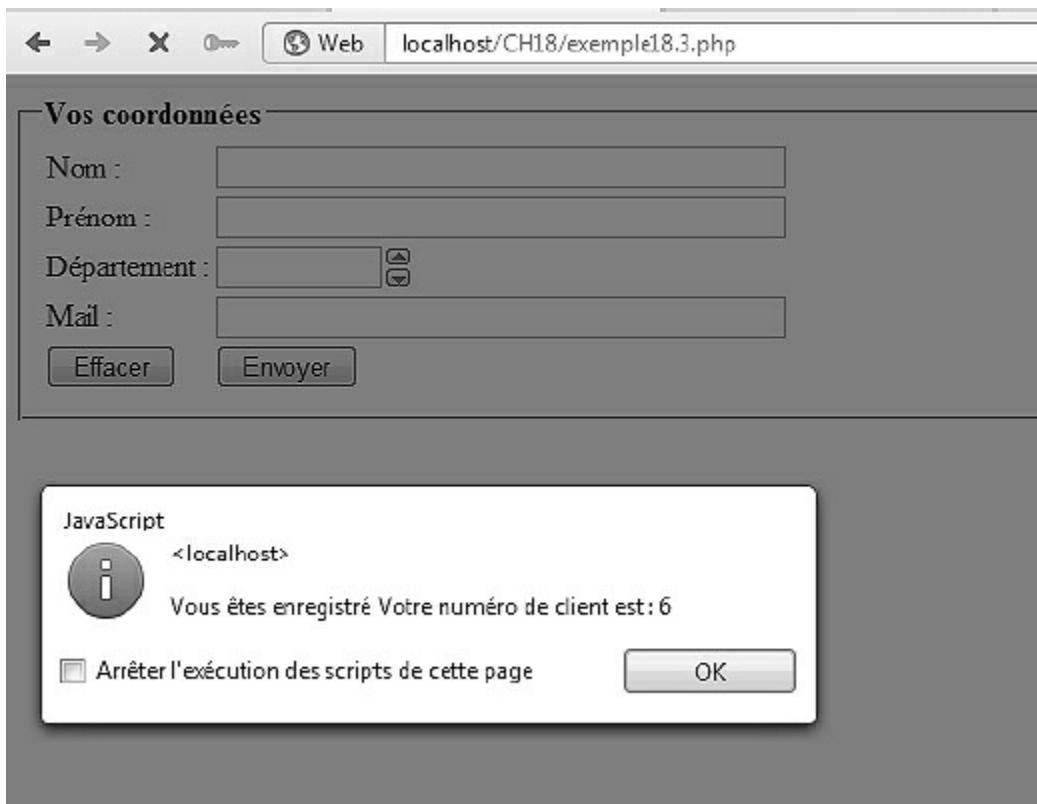
Cette création de nouvelles variables sert uniquement à améliorer la lisibilité du code, en particulier celle de la requête SQL d'insertion (repère ⑥) envoyée par la méthode `exec()` (repère ⑦). Si la requête n'est pas exécutée, vous affichez le code d'erreur retourné par la fonction `lastErrorCode()` (repère ⑧). Dans le cas contraire, vous informez le visiteur que l'enregistrement est réalisé en lui communiquant son numéro d'enregistrement retourné par la méthode `lastInsertRowID()` et en affichant une boîte d'alerte JavaScript (repère ⑨). Dans tous les cas, la base est ensuite fermée (repère ⑩).

## Exemple 17-3. Insertion de données

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Saisissez vos coordonnées</title>
</head>
<body>
<form action= "<?php echo $_SERVER['PHP_SELF'];?>" method="post"
enctype="application/x-www-form-urlencoded"> ←①
<fieldset>
<legend><b>Vos coordonnées</b></legend>
<table>
<tr><td>Nom : </td><td><input type="text" name="nom" size="40" maxlength="30"/>
    </td></tr>
<tr><td>Prénom : </td><td><input type="text" name="prenom" size="40" maxlength="30"/>
    </td></tr>
<tr><td>Département : </td><td><input type="number" step="1" name="depart" /></td>
    </tr>
<tr><td>E-mail : </td><td><input type="email" name="mail" size="40" maxlength="50"/>
    </td></tr>
<tr>
    <td><input type="reset" value="Effacer"></td>
    <td><input type="submit" value="Envoyer"></td>
</tr>
</table>
</fieldset>
</form>
<?php
if(!empty($_POST['nom']) && !empty($_POST['depart']) && !empty($_POST['mail']))
{
    $objdb=new SQLite3("sportifs.db");
    $nom= $objdb->escapeString($_POST['nom']); ←②
    $prenom= $objdb->escapeString($_POST['prenom']); ←③
    $depart= $objdb->escapeString($_POST['depart']); ←④
    $mail= $objdb->escapeString($_POST['mail']); ←⑤
    // Requête SQL
    $requete="INSERT INTO personne VALUES(NULL, '$nom', '$prenom', '$depart', '$mail')"; ←⑥
    $result=$objdb->exec($requete); ←⑦
    if(!$result)
    {
        echo "<h2>Erreur d'insertion \n n°", $objdb->lastErrorCode(), "</h2>"; ←⑧
    }
    else
    {
        echo "<script type=\"text/javascript\">
            alert('Vous êtes enregistré. Votre numéro de client est :
                ". $objdb->lastInsertRowId()."')</script>"; ←⑨
    }
    $objdb->close(); ; ←⑩
}
else {echo "Formulaire à compléter !";}
?>
</body>
```

```
| </html>
```

La figure 17-5 illustre la page de saisie et la boîte d'alerte qui indique l'identifiant du client après l'insertion.



**Figure 17-5**  
*La page d'insertion et l'affichage du code*

## ***Les transactions***

Dans le cas de la base `sportifs`, que vous aurez à développer au chapitre 19, quand un utilisateur s'inscrit sur le site, il saisit ses coordonnées, le sport qu'il pratique et son niveau. Lorsqu'il valide son formulaire, il faut que le script réalise les opérations suivantes :

1. Récupération des coordonnées du visiteur, du sport et du niveau indiqué dans des variables PHP.
2. Création de la requête d'écriture des coordonnées dans la table `personne`.
3. Récupération de l'identifiant et du niveau du visiteur, qui a été créé dans la table `personne`, dans la colonne `id_personne`.
4. Récupération de l'identifiant du sport choisi par le visiteur.
5. Requête d'écriture de l'identifiant du visiteur, de l'identifiant du sport et du niveau dans la table `pratique` qui représente l'association entre une personne et un sport.

Si les deux requêtes SQL sont exécutées sans problème, il n'y a pas à se poser de question. Mais que se passe-t-il si, après l'envoi de la première requête, la connexion à

la base est interrompue pour une raison quelconque (panne du serveur, erreur dans la deuxième requête, etc.) ? Vous vous retrouvez dans la situation où une personne est enregistrée dans la base mais ne correspond à aucun sport, ce qui crée une incohérence dans la base.

Pour éviter cela, le langage SQL propose le mécanisme des transactions, qui consiste à délimiter un ensemble de requêtes qui doivent être exécutées en bloc ou pas du tout. En cas de problème avant l'exécution de la deuxième requête SQL, l'effet de la première dans la table `personne` est annulé. La base conserve ainsi son intégrité.

Pour supprimer ce danger, il faut utiliser les transactions en délimitant l'ensemble des commandes SQL à exécuter en « tout ou rien » par la commande `BEGIN TRANSACTION` avant toute autre commande et par la commande `COMMIT TRANSACTION` après la dernière. Tant que cette commande n'est pas rencontrée, aucune insertion n'a lieu. Après son apparition, l'ensemble des insertions est réalisé en bloc.

L'exemple 17-4 a pour but d'insérer deux des informations dans deux tables. Il nous faut donc envoyer d'abord la commande SQL `BEGIN TRANSACTION` en donnant le nom `trans` qui permettra d'identifier la transaction (repère 1), puis l'envoi de deux requêtes d'insertion dans les tables `personne` (repère 2) et `pratiques` (repère 4) avec une erreur volontaire sur le nom de la table). Si les deux requêtes sont bien exécutées, ce qui est contrôlé dans les variables `$test1` et `$test2` (repères 3,5 et 6), la transaction est validée avec l'envoi de la commande `COMMIT TRANSACTION trans` (repère 7) qui évite le danger de l'insertion de lignes orphelines. Dans le cas contraire, l'ensemble des insertions (sauf les éventuelles commandes `SELECT`) est supprimé à l'aide de la commande `ROLLBACK TRANSACTION trans` (repère 8).

## Exemple 17-4. Utilisation d'une transaction

```
<?php
if ($objdb=new SQLite3("sportifs.db"))
{
    $requete1="BEGIN TRANSACTION trans"; ← 1
    $objdb->exec($requete1);
    $requete2="INSERT INTO personnes(id_personne,nom,prenom,depart,mail) VALUES
        (NULL,'Spencer','Marc','75','marc@spen.org');"; ← 2
    $test1= $objdb->exec($requete2); ← 3
    $requete3="INSERT INTO pratique(id_personne,id_sport,niveau)
        VALUES(last_insert_rowid(),2,3);"; ← 4
    $test2= $objdb->exec($requete3); ← 5
    if($test1 and $test2 ) ← 6
    {
        $requete4="COMMIT TRANSACTION trans"; ← 7
        $objdb->exec($requete4);
        echo "<br />Les requêtes ont été exécutées<hr />";
    }
    else
    {
        $requete5="ROLLBACK TRANSACTION trans"; ← 8
    }
}
```

```

    $objdb->exec($requete5);
    echo "<br />Les requêtes n'ont pas été exécutées<br />";
}
$objdb->close();
}
else
{
    echo "ERREUR !";
}
?>

```

L'erreur sur le nom de la table `pratiques` étant maintenue, le script affiche un message d'erreur et un avis, mais vous êtes assuré qu'aucune donnée parasite ne figure dans la base. La commande `COMMIT` peut aussi être envoyée séparément des autres et après avoir envoyé d'autres requêtes, comme c'est le cas ici. Ce n'est qu'après cet envoi que les commandes précédentes ont un effet dans la base s'il n'y a pas d'erreur. En corrigeant le nom de la table de "`personnes`" en "`personne`", qui est la table existante, la condition (repère ⑥) est vérifiée et on peut constater dans SQLiteManager, par exemple, que l'insertion a bien eu lieu.

## *Lecture des résultats d'une requête*

Contrairement à PDO qui permet de récupérer un résultat de différentes manières, l'objet `SQLite3Result` retourné par la méthode `query()` ne possède que la méthode `fetchArray()` pour lire les valeurs obtenues par une requête. Sa syntaxe est la suivante :

```
| $tab=$result->fetchArray(int tabtype)
```

La constante `tab_type` détermine le type du tableau retourné. Elle vaut `SQLITE3_NUM` pour un tableau indicé, `SQLITE3_ASSOC` pour un tableau associatif ou `SQLITE3_BOTH` (valeur par défaut) pour un tableau possédant à la fois des indices et des clés.

Lors de son premier appel, elle retourne un tableau contenant les données de la première ligne du résultat `$result` retourné par `query()`. À chaque nouvel appel, ce tableau contient la ligne suivante. Pour lire l'ensemble des données par une requête de sélection, il faut réaliser une boucle `while`, par exemple.

Pour libérer la mémoire après avoir lu les résultats, il faut appeler la méthode `finalize()` de l'objet `SQLite3Result` selon le modèle :

```
| $result->finalize();
```

Signalons enfin les méthodes suivantes de l'objet résultat qui donnent des informations sur les champs de la table.

- La méthode `columnName(N)` retourne le nom de la colonne placée au rang `N` dans le résultat d'une requête `SELECT` (la première colonne ayant le rang 0).
- La méthode `columnType(N)` retourne le type de la colonne placée au rang `N` dans le résultat d'une requête `SELECT` (la première colonne ayant le rang 0).
- La méthode `numColumns()` retourne le nombre de colonnes du résultat et peut être utile

pour indexer une boucle.

Pour relire les résultats en vue d'une autre utilisation, il est possible de recommencer la lecture au début en appliquant la méthode `reset()` à l'objet `SQLiteResult`.

L'exemple 17-5 utilise la méthode `fetchArray()` pour lire et afficher toutes les données de la table "personne" dans un tableau HTML (requête `SELECT`, repère 1). La requête est envoyée et le résultat contenu dans la variable `$result` (repère 2). Nous récupérons également le nombre de colonnes (repère 3) pour écrire une boucle (repère 4) qui va afficher l'en-tête du tableau en lisant les noms des colonnes de la table (repère 5). Une boucle `while` permet d'effectuer une lecture de l'ensemble en récupérant chaque ligne dans le tableau associatif `$ligne` (repère 6). Ses éléments permettent de créer le corps du tableau HTML (repères 7 et 8). Le résultat est ensuite fermé pour libérer la mémoire (repère 9).

## Exemple 17-5. Lecture des résultats d'une requête

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Lecture des personnes enregistrées</title>
<style>
table,td,th {border: 2px solid red; font-size: larger; }
</style>
</head>
<body>

<?php
if ($objdb=new SQLite3("sportifs.db",SQLITE3_OPEN_READONLY))
{
    $requete = "SELECT * FROM personne"; ← 1
    if ($result = $objdb->query ($requete)) ← 2
    {
        echo "<h3>Liste des personnes enregistrées</h3>";
        echo "<table><thead><tr>";
        $nbcoll=$result->numColumns (); ← 3
        for($i=0;$i<$nbcoll;$i++) ← 4
        {
            echo "<th>",$result->columnName ($i), "</th>"; ← 5
        }
        echo "</tr></thead><tbody>";
        while($ligne=$result->fetchArray(SQLITE3_NUM) ) ← 6
        {
            echo "<tr>";
            for($i=0;$i<$nbcoll;$i++) ← 7
            {
                echo "<td>{$ligne[$i]}&ampnbsp</td>"; ← 8
            }
            echo "</tr>";
        }
        echo "</tbody></table>";
```

```

    $result->finalize(); ←❾
}
else echo " La requête n'a pas aboutie";
$objdb->close();
}
else echo $objdb->lastErrorCode();
?>

</body>
</html>

```

Ce script affiche la liste des personnes inscrites dans la table `personne` comme le montre la figure 17-6.



<b>id_personne</b>	<b>nom</b>	<b>prenom</b>	<b>depart</b>	<b>mail</b>
1	Engels	Jean	75	aazert@dfg.fr
2	Spencer	Marc	75	marc@spen.org
3	Speak	Jan	13	jan@azerty.fr
4	Abbass	Ali	84	ali@abbass.org
5	MACA	PAUL	66	MAC@DD

**Figure 17-6**  
*Affichage des personnes enregistrées*

## *Création de fonctions SQL personnalisées*

Le langage SQL offre un grand nombre de fonctions utilisables dans les requêtes. Nous en avons donné une liste dans les chapitres précédents. Au cas où vous auriez besoin d'une fonction qui n'existe pas, SQLite vous permet de créer vos propres fonctions et de les enregistrer. L'avantage de cette technique est qu'une fonction écrite dans une requête est appliquée automatiquement à toutes les lignes du résultat pour les colonnes choisies dans la requête SQL. Cela évite de devoir effectuer un traitement répétitif *a posteriori* sur les données après lecture des lignes.

Pour parvenir à ce résultat, il suffit de suivre les deux étapes suivantes :

1. Créez une fonction personnalisée à l'aide de code PHP selon la méthode habituelle (voir le chapitre 7).
2. Déclarez cette fonction comme utilisable par SQL à l'aide de la méthode `createFunction()` de l'objet `SQLite3`, selon la syntaxe suivante :

```
boolean $objdb->createFunction(string nom_sql, string nom_php [,integer nb_param])
```

Le paramètre `nom_sql` est le nom qui servira à appeler la fonction personnalisée dans le code SQL. Il doit être différent d'un nom de fonction SQL existant. Le paramètre `nom_php` est le nom de la fonction personnalisée qui a été créée dans le code PHP à l'étape précédente. Le paramètre `nb_param` précise le nombre de paramètres nécessaires à cette fonction. Bien que facultatif, il est préférable de l'utiliser pour les fonctions ayant un nombre fixe de paramètres.

L'exemple 17-6 crée une fonction personnalisée nommée `present`, qui retourne une chaîne dont l'initiale seule est en majuscule (repère 1). Cette fonction est enregistrée par SQLite sous le nom `initialie` (repère 2). Elle est ensuite utilisée sous ce nom dans une requête SQL (repère 3) pour normaliser l'affichage des données. Dans la même requête, vous utilisez une fonction native du langage SQL, la fonction `lower()` (repère 3). L'affichage des résultats est ensuite réalisé en utilisant les indices du tableau `$ligne` (repère 4) et pourrait aussi l'être avec les clés de ce même tableau (repère 5).

### Attention

Pour utiliser les clés du tableau `$lignes`, vous devez impérativement définir des alias. Ces derniers doivent être les clés du tableau pour les colonnes auxquelles vous appliquez des fonctions personnalisées ou PHP, faute de quoi vous n'obtenez aucun résultat. Cela n'est pas nécessaire si vous utilisez seulement les indices pour l'affichage des données.

## Exemple 17-6. Création d'une fonction SQL

```
<?php
function presentation($mot) ← 1
{
    return ucfirst(strtolower($mot));
}

if ($objdb=new SQLite3("sportifs.db",SQLITE3_OPEN_READONLY))
{
    $objdb->createFunction('initialie','presentation'); ← 2
    $requete = "SELECT initialie(nom) AS nom, initialie(prenom) AS prenom, lower(mail)
               AS mail FROM personne"; ← 3
    if ($result = $objdb->query($requete))
    {
        echo "<h3>Liste des personnes enregistrées</h3>";
        while($ligne=$result->fetchArray(SQLITE3_BOTH))
        {

            echo $ligne[0]," &ampnbsp &ampnbsp ", $ligne[1] , " &ampnbsp : &ampnbsp &ampnbsp &ampnbsp ";
            $ligne[2], "<br />"; ← 4
            //ou encore : echo $ligne['nom'] , " &ampnbsp &ampnbsp ", $ligne['prenom'] , " &ampnbsp
            : &ampnbsp&ampnbsp&ampnbsp, $ligne['mail'] , "<br />"; ← 5
        }
    }
    else echo "La requête n'a pas aboutie";
    $objdb->close();
}
else echo $erreur;
```

| ?>

Le résultat obtenu a la forme suivante :

---

```
Liste des personnes enregistrées
Engels Jean : aazert@dfg.fr
Spencer Marc : marc@spen.org
Speak Jan : jan@azerty.fr
Abbass Ali : ali@abbass.org
Maca Paul : mac@dd.....
```

---

## ***Les requêtes préparées***

Au chapitre 16, nous avons réalisé des requêtes préparées à l'aide de PDO. SQLite3 permet également cette création à partir d'informations saisies par l'utilisateur. Les requêtes préparées permettent de créer des requêtes SQL non immédiatement opérationnelles mais qui contiennent des paramètres auxquels on peut donner des valeurs différentes en fonction des besoins, par exemple pour des appels répétés avec des valeurs différentes. Une requête préparée se présente sous la forme suivante, qui utilise des paramètres nommés :

```
| SELECT prenom, nom FROM personne WHERE ville=:ville AND id_personne=:id
```

Ici les paramètres nommés `:ville` et `:id` sont des noms arbitraires. La démarche à suivre pour utiliser une requête préparée est la suivante.

1. Écrivez la chaîne de la requête SQL conformément à la syntaxe précédente.
2. Utilisez cette chaîne comme paramètre de la méthode `prepare()` de l'objet SQLite3. Cette méthode retourne un objet SQLite3Stmt.
3. Liez les paramètres nommés créés précédemment à des valeurs ou à des variables à l'aide des méthodes `bindValue()` ou `bindParam()` selon les syntaxes suivantes :

```
| $objetStmt->bindValue (':param', valeur, type);
```

Le paramètre `valeur` est une valeur scalaire nombre ou chaîne, par exemple, dont le type est précisé éventuellement par une des constantes `SQLITE3_INTEGER`, `SQLITE3_FLOAT`, `SQLITE3_TEXT`, `SQLITE3_BLOB` ou `SQLITE3_NULL`.

```
| $objetStmt->bindParam (':param", $variable, type);
```

La différence entre ces deux méthodes est qu'avec `bindParam()`, on doit passer une variable comme second argument et non plus une valeur (sinon il se produit une erreur). Le type prend les mêmes valeurs des constantes.

3. Exécutez la requête préparée avec la méthode `execute()` de l'objet SQLiteStmt comme on le fait habituellement avec `query()`. Cette méthode retourne un objet SQLiteResult que nous pouvons alors lire comme nous savons déjà le faire. On aura donc le code :

```
| $result=$objetStmt->execute();
```

L'exemple 17-7 présente une application de requête préparée dans laquelle une saisie

d'un utilisateur détermine la valeur du paramètre et donc la requête finale. Un formulaire classique demande la saisie d'un numéro de département (repère 1) dans le but de trouver tous les clients qui y habitent. Cette information est tout d'abord récupérée dans la variable \$dep (repère 2). Après la connexion à la base (repère 3), nous écrivons la requête préparée avec la méthode `prepare()` (repère 4). La liaison du paramètre à la variable \$dep est effectuée en appelant la méthode `bindParam()` avec les arguments qui précisent le type de la variable (repère 5). La requête est ensuite exécutée avec la méthode `execute()` sans paramètre (repère 6). Les résultats des colonnes `id_personne`, `nom` et `prenom` de la table `personne` sont ensuite affichés avec une boucle `while` qui parcourt l'ensemble des résultats et permet l'affichage de toutes les lignes (repère 7). L'objet `SQLiteResult` est ensuite fermé (repère 8) et la connexion arrêtée en appelant la méthode `close()` (repère 9).

## Exemple 17-7. Requêtes préparées

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Recherche de client</title>
<style type="text/css" >
    div{font-size: 20px; }
</style>
</head>
<body>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">

    <fieldset>
        <legend>Recherche de client par département</legend>
        <label>Département</label><input type="number" name="departement" /><br />    <!--
- 1 -->
        <input type="submit" value="Envoyer" /> ← 1
    </fieldset>
</form>
</body>
</html>
<?php
if( isset($_POST['departement']) )
{
    $dep=$_POST['departement']; ← 2
    $objdb = new SQLite3('sportifs.db'); ← 3
    $objstmt=$objdb->prepare("SELECT id_personne,nom,prenom FROM personne
        WHERE depart= :dep "); ← 4
    //*****Liaison du paramètre
    $objstmt->bindParam(':dep',$dep,SQLITE3_INTEGER); ← 5
    $result=$objstmt->execute(); ← 6
    //*****Affichage
    echo "<div><h3>Client(s) dans le département $dep</h3><hr />";
    while($ligne=$result->fetchArray()) ← 7
    {
        echo "<h3> ID: ", $ligne[0],"   ",$ligne[1],"   ",$ligne[2],"</h3>";
    }
}
```

```

$result->finalize(); ← 8
$objdb->close(); ← 9
echo "</div>";
}
?>

```

## Accès à SQLite avec PDO

Il est également possible d'accéder à une base SQLite au moyen de PDO, comme nous l'avons fait au chapitre 16 pour MySQL. Pour cela, il faut réaliser la connexion en créant un objet PDO au moyen de son constructeur selon la syntaxe :

```
$db=new PDO("sqlite:sportifs.db")
```

Cette ligne de code permet d'accéder à notre base `sportifs` créée précédemment. Même pour une base SQLite3, il faudra écrire `sqlite` en début de chaîne de connexion sans préciser le numéro de version, comme c'était le cas pour la version 2. L'objet `$db` de la classe PDO que nous obtenons représente la connexion, de la même façon que dans les sections précédentes. Toutes les méthodes des objets PDO étudiées au chapitre 16 sont également utilisables pour gérer la base SQLite et nous ne les détaillerons pas ici. L'exemple 17-8 illustre une connexion réalisée avec PDO (repère 1), suivie de l'envoi au serveur d'une requête de sélection au moyen de la méthode `query()` (repère 2), puis de la lecture des lignes de résultat à l'aide de la méthode `fetch()` (repère 3) et d'une boucle `foreach` (repère 4). Hormis la chaîne de connexion à la base, écrite dans le constructeur de l'objet PDO, vous pouvez constater qu'il n'y a rien de neuf ici. C'est là tout l'intérêt de PDO, le même script pouvant être réutilisé pour des bases de type très différent au prix d'une modification mineure. Après avoir lu le chapitre 16, vous pouvez remarquer que PDO possède beaucoup plus de méthodes que les objets SQLite3 et il appartiendra donc à chacun de faire le choix de l'extension qu'il préfère utiliser.

### Exemple 17-8. Connexion avec PDO

```

<?php
if ($db=new PDO("sqlite:sportifs.db",SQLITE3_OPEN_READONLY)) ← 1
{
    $requete = "SELECT id_personne as 'Numéro', nom as 'Nom', prenom as 'Prénom',
    ↴ mail FROM personne";
    if ($result=$db->query($requete)) ← 2
    {
        unset($db);
        // Lecture des résultats
        echo "<h3>Personnes enregistrées</h3>";
        while($tab=$result->fetch(PDO::FETCH_ASSOC)) ← 3
        {
            foreach($tab as $cle=>$valeur) ← 4
            {
                echo $cle," :  &ampnbsp",$valeur,"  ";
            }
            echo "<br />";
        }
    }
}

```

```
    }
    else echo "La requête n'a pas aboutie";
}
else echo "ERREUR", $db->ErrorInfo();
?>
```

## Mémo des méthodes des objets

Dans la syntaxe des méthodes suivantes, la variable `$db` désigne un objet `SQLite3`, la variable `$result` un objet `SQLite3Result` et la variable `$stmt` un objet `SQLite3Stmt`.

### *Classe SQLite3*

`integer $db->changes()`

Retourne le nombre de lignes insérées, supprimées ou mises à jour par la requête SQL.

`boolean $db->close()`

Ferme la connexion à la base mais ne détruit pas l'objet `SQLite3`.

`integer $db->createFunction(string $nom_sql, string $nom_php [, int nb_param])`

Crée une fonction SQL personnalisée à partir d'une fonction PHP.

`boolean $db->createFunction(string nom_sql, string nom_php [, integer N])`

Enregistre la fonction personnalisée `nom_php` sous le nom `nom_sql` pour pouvoir l'utiliser dans des requêtes SQL. `N` désigne le nombre de paramètres de la fonction `nom_php`.

`string $db->escapeString(string $ch)`

Retourne la chaîne `$ch` dans laquelle les caractères spéciaux sont échappés.

`boolean $db->exec($requete)`

Exécute une requête qui ne retourne pas de résultat.

`integer $db->lastErrorCode()`

Retourne le dernier code d'erreur.

`string $db->lastErrorMsg()`

Retourne le dernier message d'erreur.

`integer $db->lastInsertRowID()`

Retourne la valeur du dernier identifiant auto incrémenté qui a été inséré.

`void $db->open(string $nom_base [, int SQLITE3_OPEN_READWRITE | SQLITE3_OPEN_CREATE])`

Ouvre la base si elle a été fermée : lecture/écriture | ouverte et création.

`object SQLite3Stmt $db->prepare(string $requete)`

Prépare une requête SQL.

`object SQLiteResult $db->query($requete)`

Exécute une requête et retourne un objet résultat.

`divers $db->querySingle($requete, TRUE | FALSE)`

Envoie la requête et ne retourne que le premier résultat : soit le premier champ (`FALSE`), soit la première ligne dans un tableau (`TRUE`).

### *Classe SQLite3Result*

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

Retourne le nom de la énième colonne du résultat.

```
integer $result->columnType(integer N)
```

Retourne le type de la énième colonne du résultat.

```
array $result->fetchArray(int type)
```

Retourne la ligne suivante du résultat dans un tableau. La constante type vaut `SQLITE3_NUM`, `SQLITE3_ASSOC` ou `SQLITE3_BOTH` (par défaut) pour obtenir un tableau indicé, associatif ou les deux à la fois.

```
boolean $result->finalize()
```

Rend le résultat inaccessible et libère la mémoire.

```
integer $result->numColumns()
```

Retourne le nombre de colonnes du résultat.

```
boolean $result->reset()
```

Replace le pointeur du résultat au début.

## ***Classe SQLite3Stmt***

```
boolean $stmt->bindParam(string :param , divers $param [, int type])
```

Lie un paramètre nommé avec une variable dans une requête. La constante type prend les valeurs `SQLITE3_INTEGER`, `SQLITE3_FLOAT`, `SQLITE3_TEXT`, `SQLITE3_BLOB` ou `SQLITE3_NULL` selon le type de la variable.

```
boolean $stmt->bindValue(string :param , divers valeur [, int type])
```

Lie un paramètre nommé avec une valeur scalaire. Le paramètre type prend les mêmes valeurs que précédemment.

```
boolean $stmt->clear()
```

Supprime les paramètres liés.

```
boolean $stmt->close()
```

Ferme la requête préparée avec la méthode `prepare()` de l'objet SQLite3.

```
object SQLiteResult $stmt->execute()
```

Envoie la requête préparée à la base et retourne un objet comme résultat.

```
integer $stmt->paramCount()
```

Retourne le nombre de paramètres de la requête.

```
boolean $stmt->reset()
```

Réinitialise la requête préparée ce qui permet de modifier les paramètres.

Pour les méthodes des objets PDO, référez-vous au mémo du chapitre 17.

## **Exercices**

Le but de ces exercices est de réaliser avec SQLite la même application que celle réalisée avec MySQL au cours des exercices des chapitres 13 et 14.

## **Exercice 1**

Créez une base nommée `voitures` à l'aide de SQLiteManager. Créez ensuite les tables de la base `voitures` selon le modèle logique défini dans les exercices du chapitre 13 (en cas de problème, voir le corrigé des exercices de ce chapitre). Vérifiez la structure de chaque table.

## **Exercice 2**

Créez un formulaire permettant l'insertion des coordonnées d'une personne dans la table `proprietaire` en utilisant la méthode objet.

## **Exercice 3**

Insérez des données dans la table `modele` en utilisant SQLiteManager, puis créez un script qui affiche la liste de tous les modèles de voiture dans un tableau HTML en effectuant un tri par marque. Utilisez la méthode procédurale.

## **Exercice 4**

Créez dynamiquement un formulaire contenant une liste de sélection HTML (avec les éléments `<select>` et `<option>`) qui donne la liste de tous les modèles présents dans la table `modele`. Ajoutez manuellement à l'aide de SQLiteManager un ou plusieurs modèles à la table, et vérifiez que la liste de sélection prend bien en compte ces ajouts.

## **Exercice 5**

Utilisez le mécanisme des transactions pour insérer simultanément et en toute sécurité des données dans les tables `proprietaire` et `cartegrise`. Utilisez la méthode `SQLite3`, puis la méthode `PDO`.

## **Exercice 6**

Créez un formulaire de recherche permettant de trouver toutes les personnes propriétaires d'un modèle de véhicule donné. Affichez les résultats sous forme de tableau HTML. Utilisez l'exercice 4 pour créer la liste de sélection des modèles.

## **Exercice 7**

Reprendre les exercices précédents et les réécrire en utilisant systématiquement des objets `PDO`.

## PHP et SimpleXML

---

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

À la différence de DOMXML, SimpleXML apporte, comme son nom l'indique, un accès facile à toutes sortes de documents XML, même s'ils ont une structure complexe. Cette extension est dotée de peu de méthodes pour le moment, mais elle ne tardera pas à s'enrichir au fur et à mesure des besoins manifestés par ses utilisateurs.

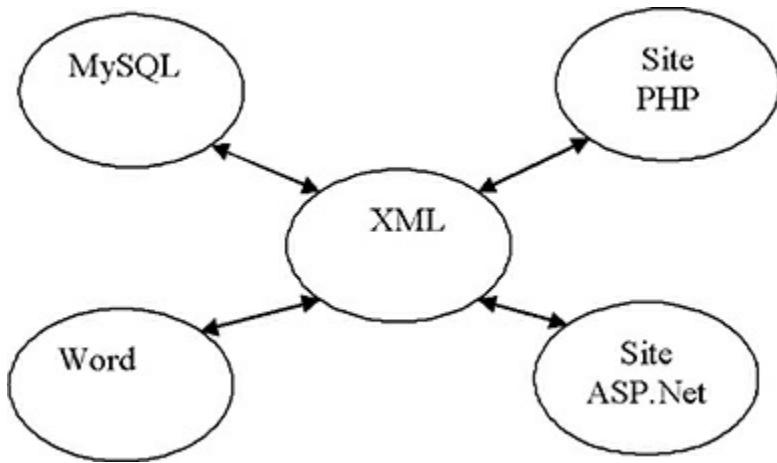
Les fonctions de SimpleXML retournent des objets qui possèdent des méthodes intéressantes d'analyse des fichiers. Les fonctions et méthodes disponibles permettent de lire et de modifier le contenu des éléments ou des attributs d'un fichier XML, de sauvegarder ces modifications et d'effectuer des recherches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

### Notions de XML

Le langage XML (*eXtensible Markup Language*) est un vecteur privilégié d'échange de données entre applications différentes. Comme l'illustre la figure 18-1, XML peut être le point commun d'applications qui ne pourraient pas communiquer entre elles sans son intermédiaire.

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

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



**Figure 18-1**

*XML est un format d'échange entre applications*

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

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

| <balise attribut = "valeur"/>

En HTML 5, il existe moins d'une centaine d'éléments prédéterminés. Les navigateurs implémentent ces éléments et associent à chacun d'eux un type de présentation de leur contenu.

Par exemple, les lignes suivantes :

| <h1>PHP 5 MySQL</h1>

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

### DTD (Document Type Definition)

Une DTD contient la description des éléments admis dans un document HTML ou XML, comme le type d'un contenu et ses attributs. Pour être conforme, le document doit respecter ces définitions. Par exemple, l'élément `<html>` ne peut avoir comme éléments fils que les éléments `<head>` et `<body>` et comme attributs que `xmlns`, `lang` et `dir`.

L'objectif de XML est différent. Il consiste à décrire un grand nombre de types d'informations différents, allant de la description des coordonnées d'un client jusqu'à celle d'une base de données complète, ce que ne permet pas le HTML.

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

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

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

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

Les principales règles syntaxiques de XML sont les suivantes :

- Chaque document commence par l'élément suivant :

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

dans lequel l'attribut `version` définit la version de XML utilisée. La dernière version publiée par le W3C est la 1.1, mais elle n'est actuellement pas bien reconnue par tous les navigateurs. L'attribut `encoding` contient le jeu de caractères utilisé dans le document. L'attribut `standalone` indique si le document est indépendant (valeur `"yes"`) ou s'il doit faire appel à un autre document externe, comme une DTD, pour pouvoir être utilisé (valeur `"no"`).

- Chaque document doit avoir un élément racine qui englobe tous les autres. C'est l'équivalent de l'élément `<html> </html>` dans un document HTML.
- Les noms des éléments sont écrits le plus souvent en minuscules. Les majuscules sont cependant autorisées, à condition que la balise d'ouverture ait la même casse que celle de fermeture.
- Chaque élément ayant un contenu doit avoir une balise d'ouverture et une balise de fermeture, comme en HTML selon la forme :

```
<element>contenu</element>
```

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

```
<element />
```

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

```
<element>
  <sous élément>
    contenu du sous-élément
  </sous élément>
</element>
```

Vous obtenez une hiérarchie père-fils selon le modèle présenté à la figure 18-2.

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

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

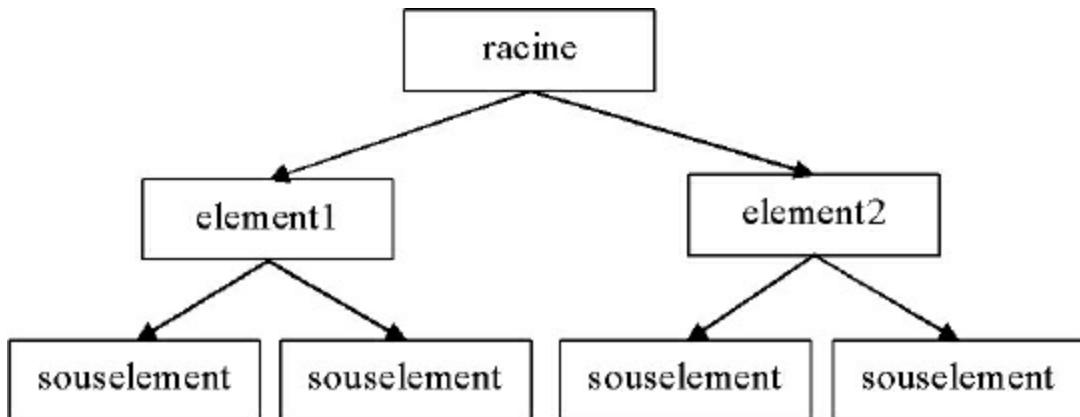
- Les caractères spéciaux suivants présents dans le contenu d'un élément doivent être remplacés par des entités prédéfinies :

```
'<' par '&lt;'  
'>' par '&gt;'  
'&' par '&amp;'  
" par '&quot;'  
' par '&apos;'
```

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

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<racine>  
  <element1 attribut="valeur">  
    <souselementA attribut="valeur">contenu A</souselementA>  
    <souselementB attribut="valeur">contenu B</souselementB>  
  </element1>  
  <element2 attribut="valeur">  
    <souselementC attribut="valeur">contenu C</souselementC>  
    <souselementD attribut="valeur">contenu D</souselementD>  
  </element2>  
</racine>
```

En conséquence de la règle d'emboîtement des éléments, un document a une structure arborescente, comme illustré à la figure 18-2.

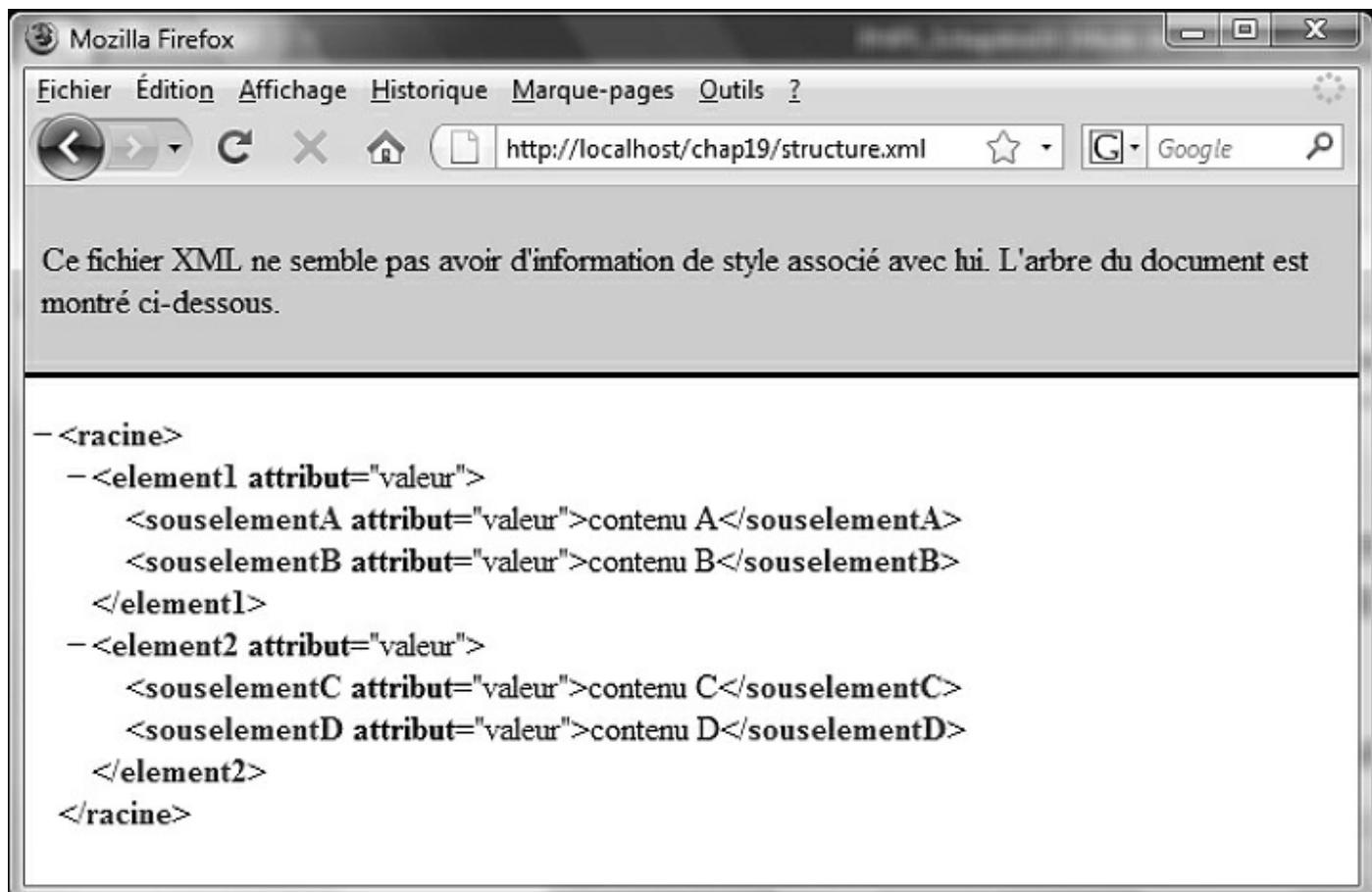


**Figure 18-2**  
*Structure arborescente d'un fichier XML*

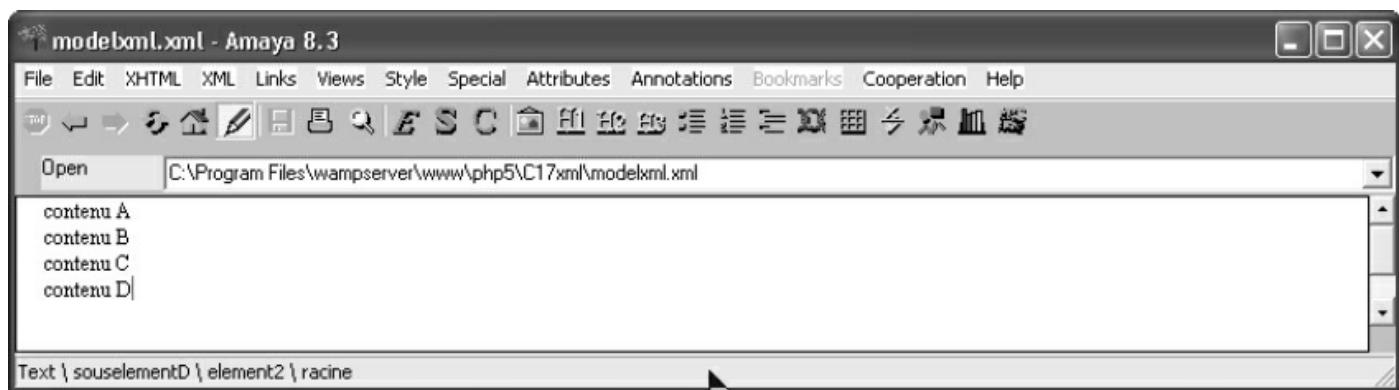
Si vous ouvrez ce fichier dans un navigateur Firefox, vous obtenez un affichage sans grand intérêt pratique, comme le montre la figure 18-3.

Dans le navigateur Amaya créé par le W3C, vous obtenez l'affichage du contenu des éléments sans les balises qui les délimitent ni leurs attributs, comme l'illustre la figure 18-4.

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



**Figure 18-3**  
*Visualisation du fichier dans Firefox*



**Figure 18-4**  
*Visualisation du fichier dans Amaya*

## Lecture d'un fichier XML

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

## Accéder au contenu d'un fichier XML

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

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

La syntaxe de `simplexml_element` est la suivante :

```
| object simplexml_load_file (string nom_fichier)
```

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

```
| object simplexml_load_string (string $code)
```

Cette fonction retourne le même type d'objet.

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

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

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

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

```
| echo $xml->livre;
```

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

### Le fichier `biblio1.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<biblio>
    <titre>L'empire de la honte</titre>
    <auteur>Jean Ziegler</auteur>
    <date>2005</date>
</biblio>
```

Le script de l'exemple 18-1 permet de lire le contenu du fichier `biblio1.xml`. Après le chargement du contenu du fichier dans l'objet `$xml` (repère 1), le script affiche le contenu de chacun des éléments en utilisant les propriétés `titre`, `auteur` et `date` de l'objet `$xml` (repères 2, 3 et 4).

### Exemple 18-1. Lecture des éléments

```
<?php
$xml=simplexml_load_file("biblio1.xml"); ← 1
echo "Titre :",$xml->titre,"<br />" ← 2;
echo "Auteur :",$xml->auteur,"<br />" ← 3;
```

```
echo "Date :" , $xml->date , "<br />" ← ④ ;  
?>
```

L'exemple retourne le résultat suivant :

---

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

---

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

Le fichier `biblio2.xml` contient encore une bibliographie, mais il est maintenant capable de contenir un nombre quelconque de livres. Pour cela, sa structure a été modifiée. L'élément racine `<biblio>` contient autant d'éléments `<livre>` que nécessaire, chaque livre étant encore caractérisé par les sous-éléments `<titre>`, `<auteur>` et `<date>`.

## Le fichier `biblio2.xml`

```
<?xml version="1.0" encoding="UTF-8"?>  
<biblio>  
  <livre>  
    <titre>L'empire de la honte</titre>  
    <auteur>Jean Ziegler</auteur>  
    <date>2005</date>  
  </livre>  
  <livre>  
    <titre>Ritournelle de la faim </titre>  
    <auteur>J.M.G Le Clézio</auteur>  
    <date>2008</date>  
  </livre>  
  <livre>  
    <titre>Singue Sabour : La pierre de patience</titre>  
    <auteur>Atiq Rahimi</auteur>  
    <date>2008</date>  
  </livre>  
</biblio>
```

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

Vous accédez aux informations utiles en écrivant le code suivant :

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

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

Le script de l'exemple 18-2 commence par charger le fichier `biblio2.xml` (repère ①). Il lit ensuite l'ensemble du fichier avec le minimum de code en effectuant une boucle

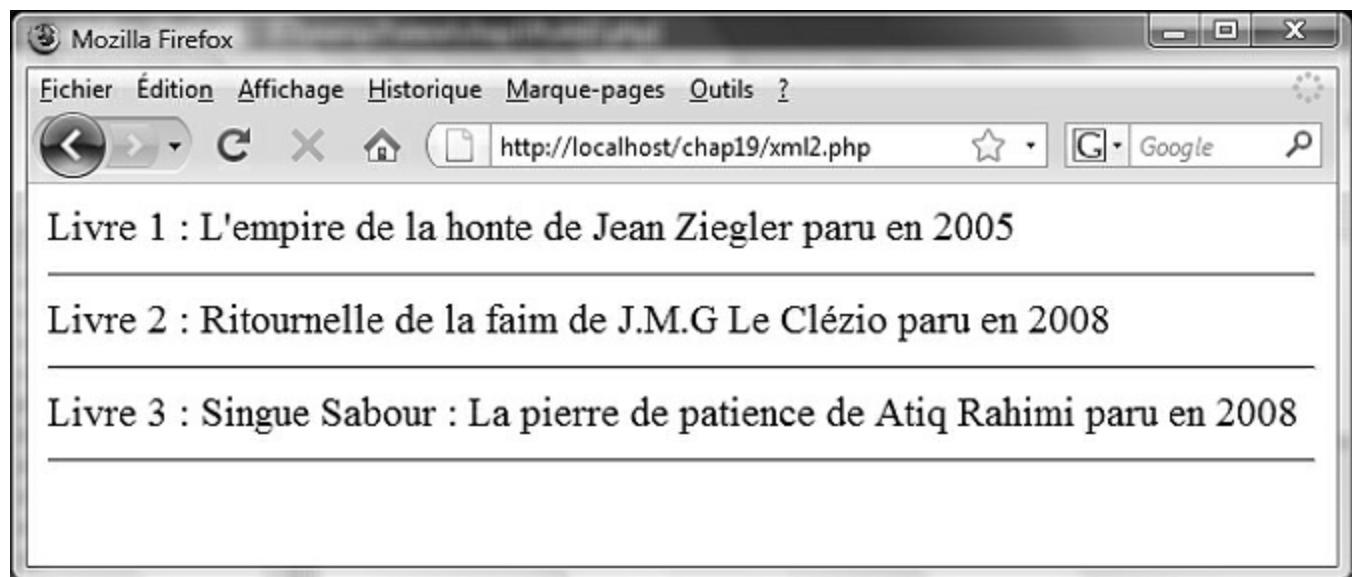
`foreach` sur l'objet `$xml->livre` (repère ②). La variable `$cle` contient la même valeur `livre` à chaque itération. La variable `$val` est aussi un objet de type `simplexml_element`, dont les propriétés sont `titre`, `auteur` et `date`. Vous y accédez avec la même syntaxe qu'à l'exemple 18-1 (repère ④). L'utilisation du mot-clé `static` pour la variable compteur `$i` et son incrémentation permet d'afficher le numéro de chaque livre (repères ③ et ⑤). La figure 18-5 illustre le résultat obtenu.

### La fonction `count()`

La fonction `get_object_vars($xml)` retourne un tableau de toutes les propriétés de l'objet `$xml`. En appliquant la fonction `count()` à ce tableau, vous obtenez le nombre de propriétés de l'objet.

## Exemple 18-2. Lecture d'un ensemble d'éléments

```
<?php
$xml=simplexml_load_file("biblio2.xml"); ←①
// Lecture du contenu des éléments
foreach($xml->livre as $cle=>$val) ←②
{
    static $i=1; ←③
    echo ucfirst($cle), " $i : $val->titre de $val->auteur paru en $val->date<br />"; ←④
    $i++; ←⑤
}
?>
```



**Figure 18-5**  
*Affichage de tous les éléments du fichier biblio2.xml*

## Lecture des attributs d'un élément

Chaque élément du document XML peut avoir des attributs. Les attributs constituent un

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

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

## Le fichier `biblio3.xml`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<biblio>
    <livre editeur="FAYARD" prix="20.00">
        <titre>L'empire de la honte</titre>
        <auteur>Jean Ziegler</auteur>
        <date>2005</date>
    </livre>
    <livre editeur="GALLIMARD" prix="18.00">
        <titre>Ritournelle de la faim </titre>
        <auteur>J.M.G Le Clézio</auteur>
        <date>2008</date>
    </livre>
    <livre editeur="POL" prix="15.00">
        <titre>Singue Sabour : La pierre de patience</titre>
        <auteur>Atiq Rahimi</auteur>
        <date>2008</date>
    </livre>
</biblio>
```

Les méthodes précédentes ne permettent pas la lecture de ces attributs. Pour parvenir à lire ces valeurs, vous disposez des deux possibilités suivantes :

- Si vous connaissez le nom des attributs, vous pouvez lire leur valeur à l'aide de la syntaxe :

```
$xml->livre["editeur"];
$xml->livre["prix"];
```

- Si vous ne connaissez pas le nom des attributs ou si ces noms sont appelés à changer, vous avez intérêt à abstraire la lecture de leur nom et de la valeur associée en utilisant une méthode spécifique. Chaque objet de type `simplexml_element` possède en effet une méthode nommée `attributes()`, qui retourne elle aussi un objet de même type. Comme précédemment, une boucle `foreach` sur cet objet permet de récupérer le nom et la valeur de chaque attribut des différents éléments `<livre>`.

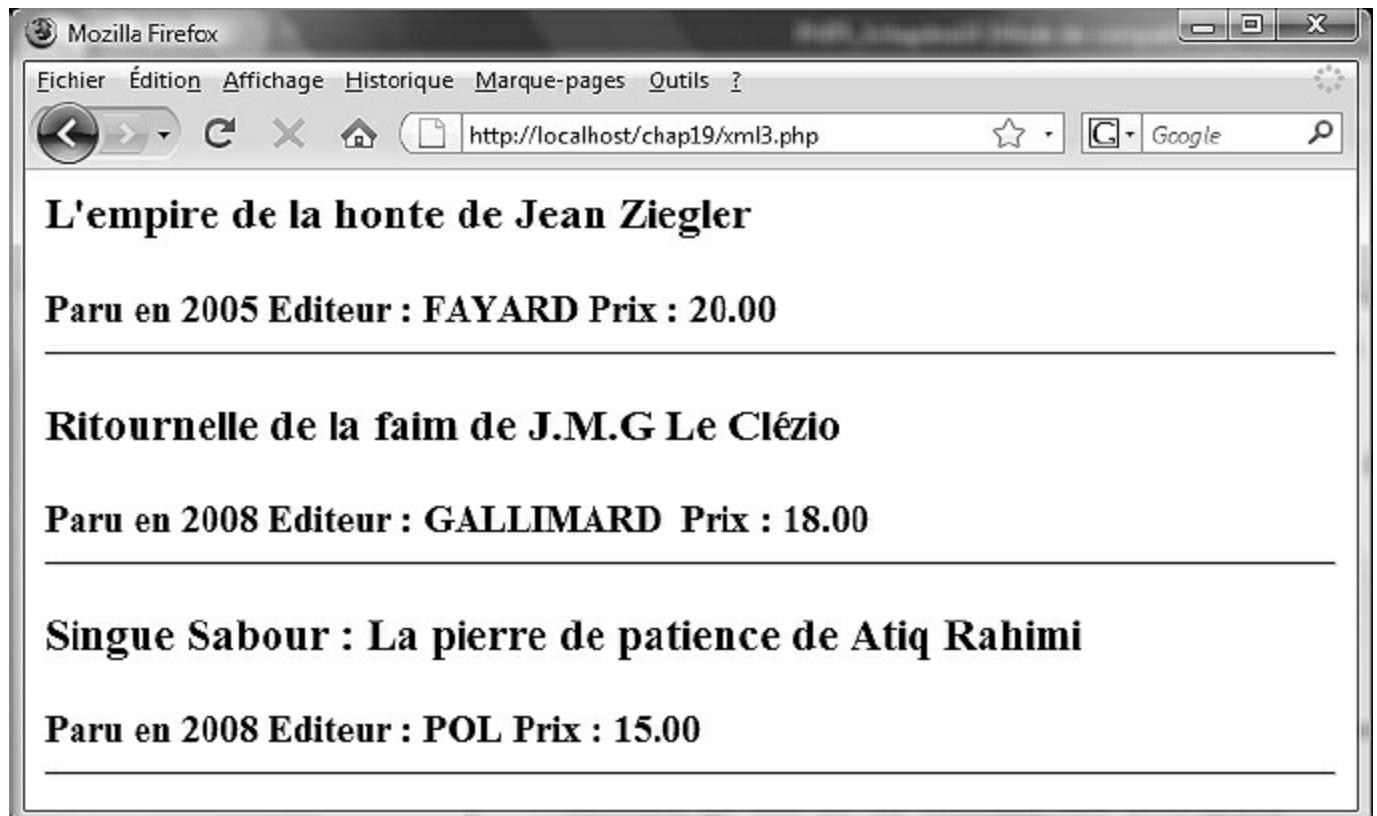
Le script de l'exemple 18-3 réalise à la fois la lecture des éléments et celle des attributs. Après l'habituel chargement du fichier XML (repère ①), une première boucle `foreach`, semblable à celle de l'exemple précédent, lit le contenu des éléments à l'aide des propriétés de l'objet `$val` (repère ②) et les affiche (repère ③). Une seconde boucle `foreach` imbriquée dans la première (repère ④) lit et affiche (repère ⑤) les propriétés de l'objet `$val` en récupérant le nom des attributs dans la variable `$att` et leur valeur

dans la variable \$valatt.

### Exemple 18-3. Lecture des éléments et des attributs

```
<?php  
$xml=simplexml_load_file("biblio3.xml"); ←①  
//Lecture du contenu des éléments  
foreach($xml->livre as $val) ←②  
{  
    echo "<h3>$val->titre de $val->auteur</h3><b> Paru en $val->date </b> "; ←③  
    //Lecture du nom et du contenu des attributs  
    foreach($val->attributes() as $att=>$valatt) ←④  
    {  
        echo "<b> $att : $valatt </b>"; ←⑤  
    }  
    echo "<hr />";  
}  
?>
```

La mise en forme du contenu obtenue à l'aide d'éléments HTML réalise l'affichage illustré à la figure 18-6.



**Figure 18-6**

*Affichage des éléments et des attributs*

### ***Lecture d'un fichier à structure complexe***

Les fichiers XML précédents ont tous une structure homogène et ne contiennent que la hiérarchie suivante répétée autant de fois que la bibliographie contient d'ouvrages :

```

<livre>
    <titre> </titre>
    <auteur> </auteur>
    <date> </date>
</livre>

```

Si le fichier contient divers éléments fils de l’élément racine, ces exemples ne permettent plus une lecture aisée du contenu des éléments. Le fichier `biblio4.xml` présente un type de structure complexe contenant un élément `<ouvrage>` (repère ❶) qui contient des éléments `<livre>` comme précédemment. En plus, un élément `<musique>` (repère ❷) contient des éléments `<disque>`, qui contiennent à leur tour des éléments `<titre>`, `<auteur>` et `<date>`. Chaque grande catégorie (ouvrage et musique) est donc constituée de trois niveaux d’éléments imbriqués.

## Le fichier `biblio4.xml`

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<biblio>
    <ouvrage> ← ❶
        <livre editeur="FAYARD" prix="20.00">
            <titre>L'empire de la honte</titre>
            <auteur>Jean Ziegler</auteur>
            <date>2005</date>
        </livre>
        <livre editeur="GALLIMARD " prix="18.00">
            <titre>Ritournelle de la faim </titre>
            <auteur>J.M.G Le Clézio</auteur>
            <date>2008</date>
        </livre>
        <livre editeur="POL" prix="15.00">
            <titre>Singue Sabour : La pierre de patience</titre>
            <auteur>Atiq Rahimi</auteur>
            <date>2008</date>
        </livre>
    </ouvrage>
    <musique> ← ❷
        <disque editeur="Deutsche Grammophon" Prix="14.00">
            <titre>5 eme Symphonie </titre>
            <auteur>Ludwig van BEETHOVEN</auteur>
            <date>1808</date>
        </disque>
        <disque editeur="EMI" Prix="14.00">
            <titre>Suites pour violoncelle </titre>
            <auteur>Johann Sebastian BACH</auteur>
            <date>1725</date>
        </disque>
    </musique>
</biblio>

```

Pour lire le contenu de ces deux catégories, vous disposez de la méthode `children()` de l’objet `$xml` de type `simplexml_element` retourné par la fonction `simplexml_load_file()` de chargement du fichier XML. Chaque fois qu’elle est appliquée, elle retourne un objet de type `simplexml_element` contenant tous les sous-éléments de même niveau situés dans le document, et donc tous les enfants du nœud précédent.

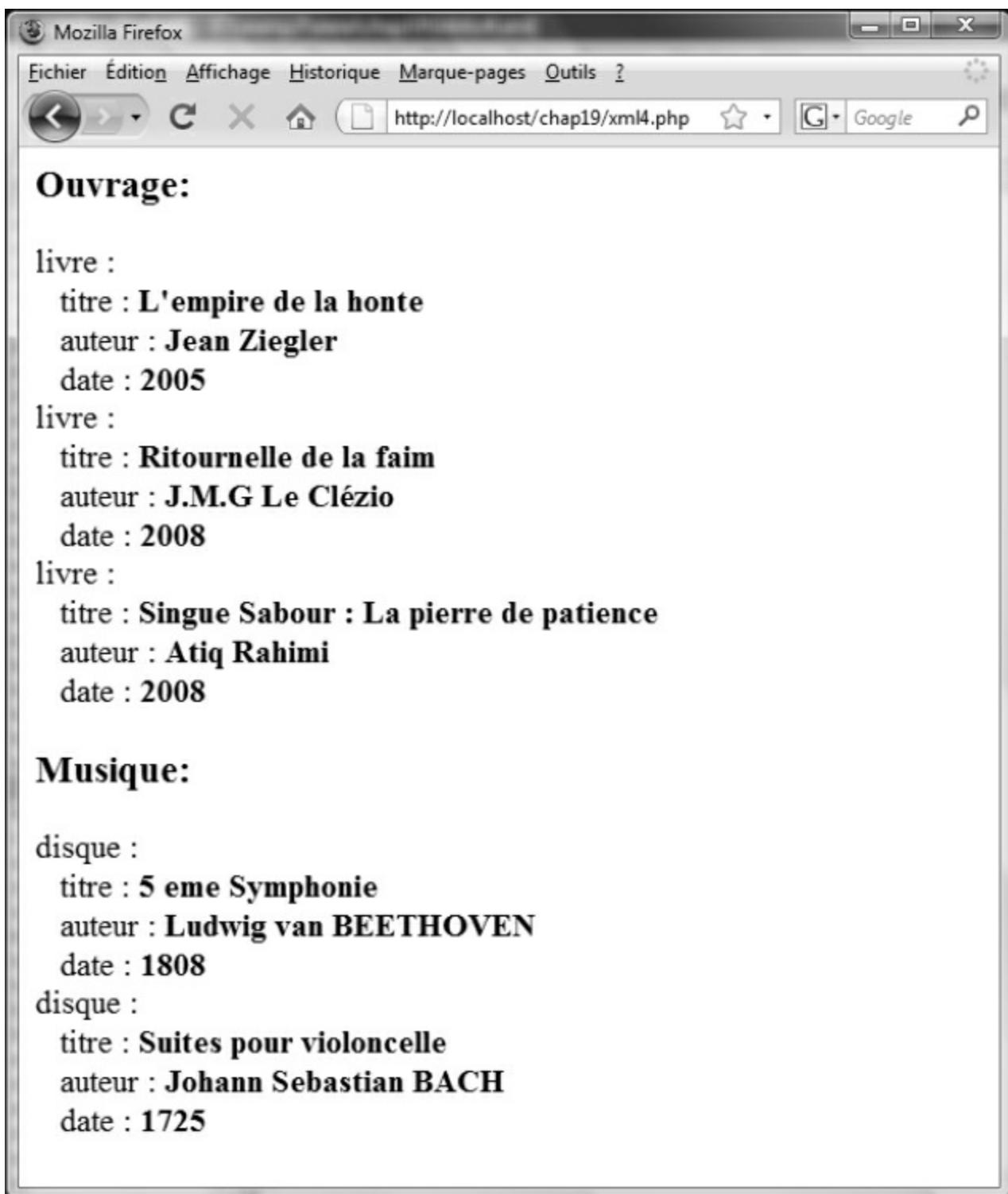
Un premier appel (repère ①) retourne les éléments `<ouvrage>` et `<musique>`, donc tous les enfants de l'élément racine `<biblio>`. Un deuxième appel appliqué à ces éléments retourne ceux qu'ils contiennent, et ainsi de suite (repères ② et ③).

Sur ce principe, l'exemple 18-4 permet la lecture des trois niveaux de chaque catégorie à l'aide de trois boucles `foreach` imbriquées l'une dans l'autre. Le code des deux boucles les plus internes est identique. En imbriquant une quatrième boucle, vous pourriez lire quatre niveaux d'éléments imbriqués.

## Exemple 18-4. Lecture d'une hiérarchie d'éléments complexe

```
<?php
$xml=simplexml_load_file("biblio4.xml");
foreach($xml->children() as $element=>$val) ←①
{
    echo "<h3>", ucfirst($element) ,": $val</h3>";
    foreach($val->children() as $element=>$val) ←②
    {
        echo "$element : <b>$val</b><br />";
        foreach($val->children() as $element=>$val) ←③
        {
            echo " &ampnbsp $element : <b>$val</b><br />";
        }
    }
}
?>
```

La figure 18-7 illustre le résultat du script de lecture du fichier.



**Figure 18-7**  
*Lecture du fichier à structure complexe*

### ***Modification des valeurs des éléments et des attributs***

En utilisant les propriétés de l'objet de type `simplexml_element` retourné par la fonction `simplexml_load_file()`, un script peut modifier la valeur du contenu d'un élément ou d'un attribut. Il suffit pour cela d'utiliser la notation objet qui permet de lire une propriété et de lui affecter une nouvelle valeur. L'exemple 18-5 réalise ces modifications et enregistre le fichier modifié sur le serveur.

Pour modifier, par exemple, les caractéristiques du premier livre du fichier `biblio3.xml`, vous pouvez écrire le code de l'exemple 18-5. Il commence par charger le fichier dans l'objet `$xml` (repère 1) puis change le titre et la date du premier élément `<livre>` (repères 2 et 3).

À ce stade, les données sont censées être modifiées dans l'objet `$xml`. Si vous le vérifiez en effectuant une boucle de lecture (repère 4), vous constatez que le fichier XML n'est pas modifié. La raison à cela est que les modifications n'ont pas été enregistrées. Pour enregistrer les modifications, vous appelez la méthode `asxml()` des objets `simplexml_element`, dont la syntaxe est la suivante :

```
| string $xml->asxml([string nom_fichier])
```

Cette méthode retourne le contenu de l'objet `$xml` dans une chaîne de caractères `$chxml` (ou `FALSE` en cas d'erreur). Utilisée avec comme paramètre un nom de fichier (ici `biblio3a.xml` afin de ne pas écraser l'ancien fichier), la méthode enregistre également le fichier sur le serveur (repère 5). Un contrôle sur la valeur de la variable `$chxml` permet d'afficher un message de confirmation (repère 6).

## Exemple 18-5. Modification et enregistrement des données

```
<?php
$xml=simplexml_load_file("biblio3.xml"); ← 1
// Modification d'un élément et d'un attribut
$xml->livre[0]->titre="La haine de l'Occident"; ← 2
$xml->livre[0]->date="2008"; ← 3
// Affichage des données du fichier
foreach($xml->livre as $cle=>$val) ← 4
{
    static $i=1;
    echo ucfirst($cle)," $i : $val->titre de $val->auteur paru en $val->date<hr />";
    $i++;
}
// Enregistrement des modifications
$chxml= $xml->asxml("biblio3a.xml"); ← 5
if($chxml) echo "Enregistrement réalisé"; ← 6
?>
```

Après l'exécution du script de l'exemple 18-5, 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.
- Pour trouver tous les titres de disque :

```
| $xml->xpath("/biblio/musique/disque/titre");
| $xml->xpath("//musique/disque/titre");
| $xml->xpath("//disque/titre");
```

- De même que précédemment, pour trouver tous les compositeurs, vous remplacez `titre` par `auteur`. Faites de même pour les dates.
- Pour trouver tous les titres ou les auteurs, qu'ils soient dans la catégorie `ouvrage` ou `musique`, vous écrivez :

```
| $xml->xpath("//titre");
| $xml->xpath("//auteur");
```

- Pour trouver la valeur d'un attribut, vous écrivez le nom de l'attribut à la place de celui de l'élément, en le faisant précéder du caractère `@`. Par exemple, pour trouver tous les attributs `éditeur` des éléments `<livre>`, vous écrivez :

```
| $xml->xpath("/biblio/ouvrage/livre/@éditeur");
| $xml->xpath("//ouvrage/livre/@éditeur");
| $xml->xpath("//livre/@éditeur");
```

- Pour trouver toutes les valeurs des attributs `éditeur`, qu'ils soient ceux des éléments `<livre>` ou `<disque>`, vous écrivez :

```
| $xml->xpath("//@éditeur");
```

## XPath

XPath est un langage puissant permettant la création de requêtes. Nous ne nous étendons pas ici sur la composition des requêtes XPath, mais vous pouvez consulter à ce sujet la recommandation du W3C sur le site <http://www.w3.org/TR/xpath>.

Les résultats de ces recherches étant retournés dans un tableau indicé, vous pouvez

utiliser une boucle `for` ou `foreach` pour les lire.

L'exemple 18-6 propose une illustration du mécanisme de recherche utilisant la méthode `xpath()`. Le fichier est composé d'une partie HTML, qui crée le formulaire comprenant deux listes de sélection nommées `choix` (repère 1) et `cat` (repère 2), avec lesquelles l'utilisateur peut choisir respectivement le contenu qu'il désire (titre, auteur ou éditeur) et la catégorie (ouvrage, musique ou les deux à la fois). La valeur associée à chaque option servira à composer la chaîne de la requête XPath. Ces valeurs sont récupérées par le script dans les variables `$_POST['choix']` et `$_POST['cat']` (repères 3 et 4).

Après le chargement du fichier `biblio4.xml` (repère 5), l'objet `$xml` appelle la méthode `xpath()`, dont la chaîne de requête est la concaténation de deux variables, `$cat` et `$choix`, les résultats étant retournés dans le tableau `$result` (repère 6). L'élimination des doublons s'effectue en appliquant la fonction `array_unique()` au tableau `$result` (repère 7). L'utilisation d'une boucle `foreach` permet l'affichage de tous les résultats sous forme de liste ordonnée (repère 8).

La figure 18-8 illustre les résultats obtenus après une recherche de tous les titres de la catégorie ouvrage, la chaîne de requête étant alors la concaténation des chaînes `//ouvrage/livre/titre`.

## Exemple 18-6. Recherche dans un fichier XML

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Bibliographie XML</title>
</head>
<body>
<form action= "<?php echo $_SERVER['PHP_SELF'] ?>" method="post"
      enctype="application/x-www-form-urlencoded">
<fieldset>
<legend><b>Bibliographie</b></legend>
<table><tbody>
<tr>
<td>Rechercher tous les : </td>
<td>
<select name="choix"> ← 1
<option value="titre">Titre</option>
<option value="auteur">Auteur</option>
<option value="@editeur">Éditeur</option>
</select>
</td>
<td>Dans les catégories
<select name="cat"> ← 2
<option value="//ouvrage/livre/"> Ouvrages </option>
<option value="//musique/disque/"> Musique </option>
<option value="//">Toutes</option>
</select>
<input type="submit" name="envoi" value="OK"/>
</td>
</tr>
</tbody></table>
```

```

</fieldset>
</form>

<?php
if(isset($_POST['envoi']))
{
    $choix= $_POST['choix']; ←③
    $cat= $_POST['cat']; ←④
    $xml=simplexml_load_file("biblio4.xml"); ←⑤
    $result= $xml->xpath($cat.$choix); ←⑥
    // Éliminer les doublons
    $result=array_unique($result); ←⑦
    echo "<h3>Résultats de la recherche</h3>";
    // Affichage sous forme de liste
    echo "<div><ol>";
    foreach($result as $valeur) ←⑧
    {
        echo "<li><big>$valeur </big></li>";
    }
    echo "</ol></div>";
}
?>
</body>
</html>

```



**Figure 18-8**  
*Résultats de la recherche des titres dans le fichier XML*

## Création d'un fichier XML à partir d'un formulaire

Il est possible de créer un fichier XML à partir des données saisies dans un formulaire, comme vous l'avez déjà réalisé dans un fichier texte ou dans une base de données MySQL. Vous verrez dans la section suivante qu'il est aussi possible de transférer des données d'une table de base de données dans un fichier XML et même de réaliser l'opération inverse.

L'exemple 18-7 crée une bibliographie en ligne, qui sera enregistrée dans un fichier XML à partir des données saisies dans un formulaire. Ce formulaire contient trois champs de type texte, dans lesquels un visiteur peut indiquer le titre, l'auteur et la date de parution de l'ouvrage qu'il souhaite ajouter à la bibliographie. Celle-ci peut s'enrichir des choix de chaque visiteur. Les scripts utilisés précédemment permettent ensuite l'affichage de l'ensemble des livres.

Le script de traitement des données est contenu dans le fichier `trait.php`. La partie de code PHP qui se situe dans le même fichier que le code HTML vérifie d'abord l'envoi du formulaire et si les champs texte sont bien complétés (repère ①). Les valeurs de ces champs sont ensuite récupérées dans les variables `$titre`, `$auteur` et `$date`. Les caractères spéciaux XML sont transformés en entités de caractères en utilisant la fonction `htmlspecialchars()` (repères ②, ③ et ④).

Si le fichier `biblio6.xml` qui va contenir les données n'existe pas encore, vous l'écrivez dans la variable `$chxml`. Ici, nous réutilisons le fichier `biblio3.xml` (repère ⑤) pour y ajouter un élément. L'élément racine se nomme `<biblio>` et contient un seul élément `<livre>` (repère ⑥). S'il existe, son contenu est récupéré dans la variable objet `$xml` (repère ⑦) et transféré dans la variable chaîne de caractères `$chxml` en appelant la méthode `asxml()` (repère ⑧).

Il vous faut à ce stade supprimer la balise de fermeture `</biblio>` en utilisant la fonction `str_replace()`, qui la remplace par une chaîne vide dans la variable `$chxml` (repère ⑨). Vous pouvez concaténer les nouvelles données avec les anciennes et ajouter la balise `</biblio>` afin que le document XML soit bien formé (repère ⑩). Il ne reste plus qu'à enregistrer le contenu de la chaîne `$chxml` dans le fichier `biblio6.xml` à l'aide de la fonction `file_put_contents()` (repère ⑪).

## Exemple 18-7. Enregistrement des données d'un formulaire au format XML

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Enregistrement en XML</title>
</head>
<body>
<form action= "trait.php" method="post" enctype=
 "application/x-www-form-urlencoded">
<fieldset>
<legend><b>Saisie de données</b></legend>
<table><tbody>
<tr>
```

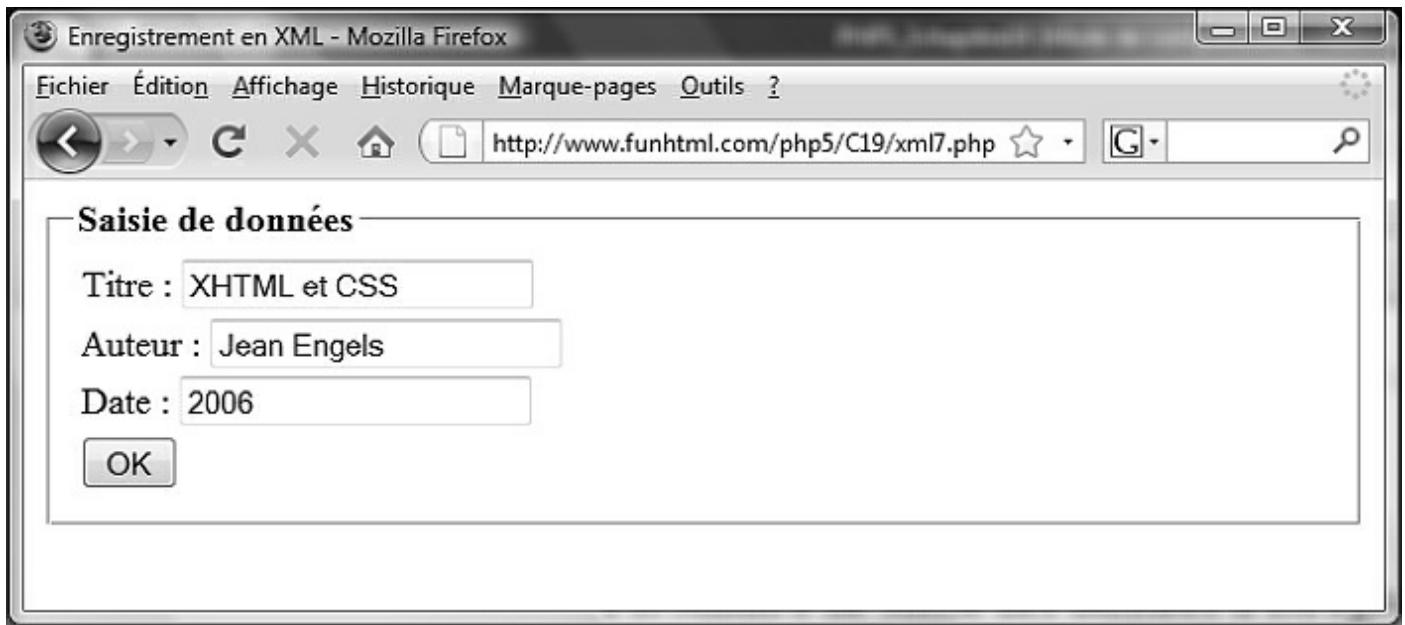
```

<td>Titre :
<input type="text" name="titre" />
</td>
</tr>
<tr>
<td>Auteur :
<input type="text" name="auteur" />
</td>
</tr>
<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>
```

Voici le script de traitement des données trait.php.

```

<?php
if(isset($_POST['envoi'])&& !empty($_POST['titre'])&& !empty($_POST['auteur'])
&& !empty($_POST['date'])) ← 1
{
    $titre= htmlspecialchars($_POST['titre']); ← 2
    $auteur= htmlspecialchars($_POST['auteur']); ← 3
    $date= htmlspecialchars($_POST['date']); ← 4
    if(!file_exists("biblio3.xml")) ← 5
    {
        $chxml= "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n<biblio>\n <livre>
        <titre>$titre</titre>\n <auteur>$auteur</auteur>\n <date>$date</date>\n
        </livre>\n</biblio>"; ← 6
    }
    else
    {
        $xml=simplexml_load_file("biblio3.xml"); ← 7
        $chxml = $xml->asXML(); ← 8
        $chxml = str_replace("</biblio>", "", $chxml); ← 9
        $chxml.= "<livre>\n <titre>$titre</titre>\n <auteur>$auteur</auteur>
        \n <date>$date</date>\n</livre>\n</biblio>"; ← 10
    }
    $verif=file_put_contents("biblio6.xml", $chxml); ← 11
}
?>
```



**Figure 18-9**

*Formulaire de saisie des données à enregistrer au format XML*

## 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.

### ***Création d'un fichier XML à partir d'une table MySQL***

Différentes méthodes permettent de transférer les données d'une table MySQL dans un fichier XML. Ce dernier est alors utilisable par toutes sortes d'applications, incluant PHP, JSP et ASP.Net.

### **Utilisation de phpMyAdmin**

La méthode la plus simple et la plus rapide pour créer un fichier XML à partir d'une base de données consiste à utiliser phpMyAdmin. Cette interface est dotée d'un script, qui réalise cette opération automatiquement. La structure du fichier XML créé a une structure bien déterminée, qui ne correspond pas nécessairement aux désirs du programmeur.

Les règles de transformation appliquées par ce script sont les suivantes :

- Le nom de la base devient l'élément racine du document XML.
- Le nom de la ou des tables devient un élément de premier niveau. Il y a donc autant

d'éléments de ce type que de lignes dans la table MySQL.

- Le nom de chaque colonne de la table devient celui d'un élément imbriqué dans le précédent, dont le contenu est la valeur du champ de la table.
- Aucun attribut n'est créé.

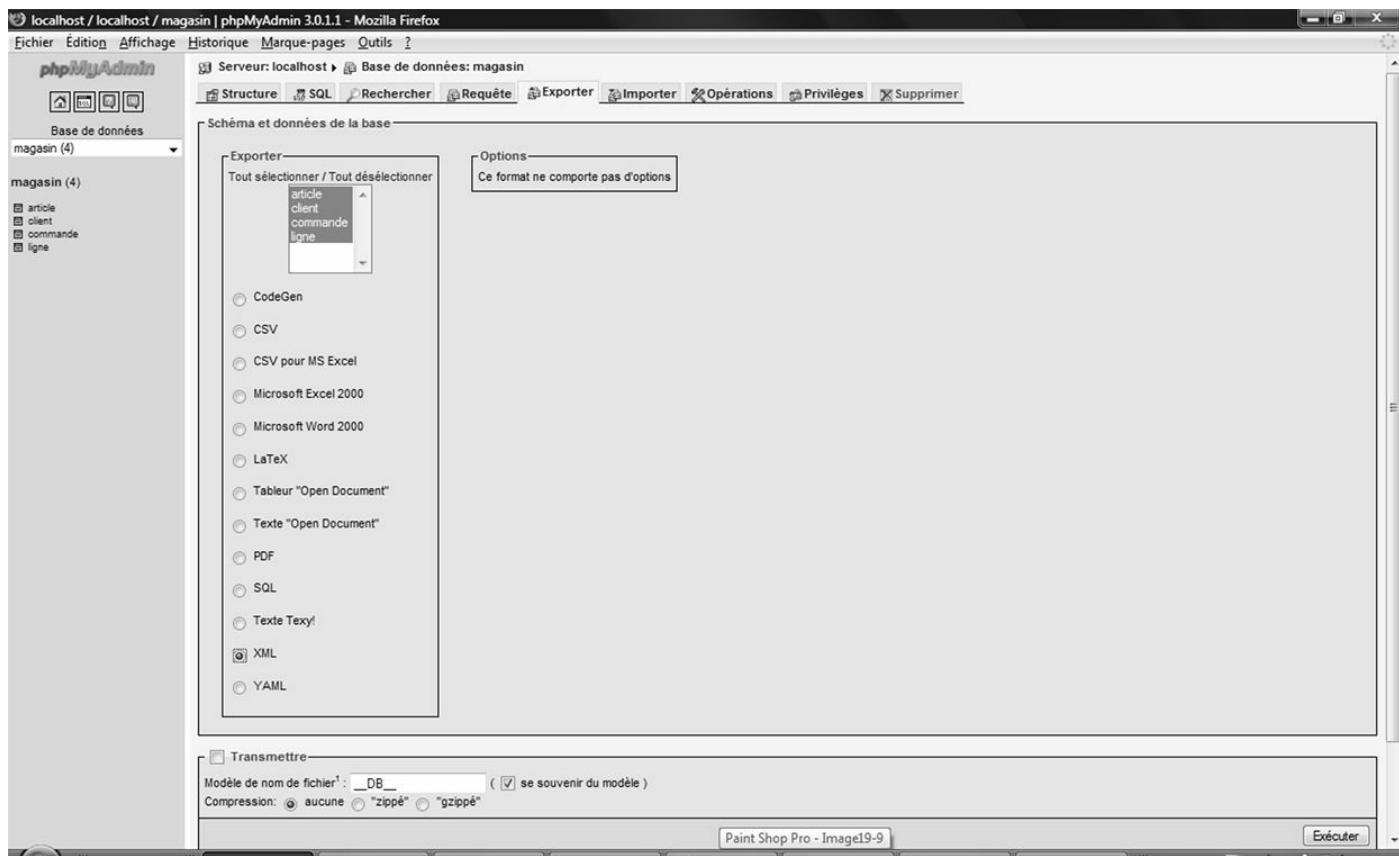
Plus généralement, si vous exportez une base entière comportant plusieurs tables, la structure du fichier XML résultant est de la forme de l'exemple 18-8. Ce dernier est le résultat de l'exportation de la base `magasin` utilisée au chapitre 14.

Les étapes pour exporter cette base en utilisant l'interface phpMyAdmin sont les suivantes :

1. Choisissez la base dans la liste de sélection.
2. Cliquez sur le bouton Exporter.
3. Sélectionnez la ou les tables désirées.
4. Cliquez sur le bouton radio XML.
5. Cochez la case Exécuter.

En principe, une boîte de dialogue apparaît, vous invitant à enregistrer le fichier sur le poste. Cela ne fonctionne toutefois que si vous choisissez la compression zippé ou gzippé. Si vous ne choisissez aucune compression, le fichier XML s'affiche dans son intégralité. Vous pouvez le sauvegarder avec l'extension `.xml` en cliquant sur la fenêtre qui contient le code et en sélectionnant Fichier, Enregistrer sous, Nom de fichier et Type `.xml`.

La figure 18-10 présente la page de l'utilitaire phpMyAdmin qui permet l'exportation des données au format XML.



**Figure 18-10**  
*Exportation de la table article au format XML avec phpMyAdmin*

### Exemple 18-8. Le fichier magasin.xml

```
<!-- - phpMyAdmin XML Dump -- version 3.5.1 -- http://www.phpmyadmin.net -- Client :  

localhost  

-- Généré le : Sam 12 Janvier 2013 à 18:18 -- Version du serveur : 5.5.16-log --  

Version de  

-- PHP : 5.4.3 -->  

<!-- Base de données : 'magasin' -->  

<pma_xml_export>  

<database name="magasin">  

<!-- Table article -->  

<table name="article">  

<column name="id_article">CA300</column>  

<column name="designation">Canon EOS 3000V zoom 28/80</column>  

<column name="prix">329.00</column>  

<column name="categorie">photo</column>  

</table>  

<!--Suite de la table article -->  

<!-- Table client -->  

<table name="client">  

<column name="id_client">  

1  

</column>  

<column name="nom">  

Marti  

</column>  

<column name="prenom">  

Jean
```

```
</column>
<column name="age">
36
</column>
<column name="adresse">
5 Av Einstein
</column>
<column name="ville">
Orléans
</column>
<column name="mail">
mart@marti.com
</column>
</table>
<!--Suite de la table client -->
```

```
<!-- Table commande -->
<table name="commande">
<column name="id_comm">
1
</column>
<column name="id_client">
5
</column>
<column name="date">
2012-06-11
</column>
</table>
<table name="commande">
<column name="id_comm">
2
</column>
<column name="id_client">
9
</column>
<column name="date">
2012-06-25
</column>
</table>
<!--Suite de la table commande -->
```

```
<!-- Table ligne -->
<table name="ligne">
<column name="id_comm">
1
</column>
<column name="id_article">
CAS07
</column>
<column name="quantite">
3
</column>
<column name="prixunit">
26.90
</column>
</table>
<table name="ligne">
<column name="id_comm">
1
```

```

</column>
<column name="id_article">
CS330
</column>
<column name="quantite">
1
</column>
<column name="prixunit">
1629.00
</column>
</database>
<!--Suite de la table ligne -->

</pma_xml_export>

```

## Utilisation d'un script PHP

L'inconvénient de la méthode précédente est que la structure du fichier XML créé par le script de phpMyAdmin est déterminée par avance et qu'elle ne définit aucun attribut. Si, pour une raison quelconque d'organisation, vous souhaitez que certaines colonnes d'une table deviennent des attributs ou toute autre structure particulière, il vous faut écrire un script personnalisé qui crée par un fichier XML approprié.

L'exemple 18-9 en donne une illustration en créant un fichier XML nommé `article.xml`, contenant les données de la table `article` de la base `magasin` créée et utilisée à plusieurs reprises dans l'ouvrage.

La structure choisie pour l'exportation des données est la suivante :

```

<article id="CS330" categorie="vidéo">
  <designation>Caméscope Sony DCR-PC330 </designation>
  <prix>1629.00 </prix>
</article>

```

L'élément `<article>` contient les informations sur les quatre colonnes de la table `article` mais dans une organisation différente. La clé primaire `id_article` de la table devient l'identifiant unique de l'élément `<article>`, et la colonne `categorie` un attribut de cet élément. Les sous-éléments `<designation>` et `<prix>` correspondent aux colonnes de même nom dans la table.

Comme vous devez accéder à la base `magasin`, le script commence par inclure la fonction de connexion (repère ①). Il établit ensuite la connexion au serveur MySQL et récupère l'identifiant de connexion dans la variable `$idcom` (repère ②). La requête SQL est très simple puisque vous souhaitez lire toutes les colonnes de la table (repère ③). Après envoi de la requête (repère ④), l'ensemble des résultats est accessible au moyen de la variable `$result`. Vous créez alors une variable `$chxml` contenant la déclaration XML et l'élément racine `<magasin>` (repère ⑤).

Une boucle `while` de lecture des résultats récupère chaque ligne dans un tableau associatif (repère ⑥) et concatène les différents éléments à la chaîne `$chxml` (repères ⑦, ⑧, ⑨ et ⑩). La balise de fermeture de l'élément racine `</magasin>` est finalement ajoutée à la chaîne du code XML (repère ⑪), qui est enregistrée dans le fichier

article.xml (repère 12).

### Exemple 18-9. Transfert de données d'une table dans un fichier XML

```
<?php
include ("exemple15.2.php"); ← 1
$idcom=connex ("magasin", "myparam"); ← 2
$requete="SELECT * FROM article"; ← 3
$result=mysql_query ($requete,$idcom); ← 4
$chxml=<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>\n<magasin>; ←
5
while ($ligne=mysql_fetch_array ($result,MYSQL_ASSOC)) ← 6
{
    $chxml.= "<article id=\"{$ligne['id_article']}\" categorie=\"{$ligne['categorie']}\""
    ← 7
    $chxml.= " <designation>{$ligne['designation']} </designation>\n"; ← 8
    $chxml.= " <prix>{$ligne['prix']} </prix>\n"; ← 9
    $chxml.= "</article>\n"; ← 10
}
$chxml.= "</magasin>"; ← 11
$verif=file_put_contents ("article2.xml", $chxml); ← 12
?>
```

La figure 18-11 présente une partie du fichier XML créé par ce script.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes" ?>
- <magasin>
-   <article id="CS330" categorie="vidéo">
    <designation>Caméscope Sony DCR-PC330</designation>
    <prix>1629.00</prix>
  </article>
-   <article id="NIK55" categorie="photo">
    <designation>Nikon F55+zoom 28/80</designation>
    <prix>269.00</prix>
  </article>
-   <article id="NIK80" categorie="photo">
    <designation>Nikon F80</designation>
    <prix>479.00</prix>
  </article>
-   <article id="SOXMP" categorie="informatique">
    <designation>PC Portable Sony Z1-XMP</designation>
    <prix>2399.00</prix>
  </article>
-   <article id="HP497" categorie="informatique">
    <designation>PC Bureau HP497 écran TFT</designation>
    <prix>1500.00</prix>
  </article>
-   <article id="DVD75" categorie="divers">
    <designation>DVD vierge par 3</designation>
    <prix>17.50</prix>
  </article>
-   <article id="CAS07" categorie="divers">
    <designation>Cassette DV60 par 5</designation>
    <prix>26.90</prix>
  </article>
-   <article id="DEL30" categorie="informatique">
    <designation>Portable Dell X300</designation>
    <prix>1715.00</prix>
  </article>
-   <article id="CP100" categorie="vidéo">
    <designation>Caméscope Panasonic SV-AV 100</designation>
    <prix>1490.00</prix>
  </article>
-   <article id="SAX15" categorie="informatique">
    <designation>Portable Samsung X15 XVM</designation>
    <prix>1999.00</prix>
  </article>
```

**Figure 18-11**  
*Le fichier article.xml visualisé dans Internet Explorer*

## ***Création d'une table MySQL à partir d'un fichier XML***

Vous pouvez maintenant réaliser l'opération inverse en insérant dans une table d'une base MySQL les données d'un fichier XML en provenance d'une source quelconque. La structure de ce fichier doit être connue pour permettre de créer à l'avance la table destinée à recevoir les données.

Dans l'exemple 18-11, vous récupérez les données d'un fichier XML ayant une structure identique à celle du fichier biblio3.xml utilisé précédemment. L'élément racine se nommait `<biblio>` et son contenu était de la forme suivante :

```
<livre editeur="FAYARD" prix="20.00">
  <titre>L'empire de la honte</titre>
  <auteur>Jean Ziegler</auteur>
```

```
<date>2005</date>
</livre>
```

Les étapes nécessaires sont les suivantes :

- Crédation d'une base nommée `biblio` si le serveur admet plusieurs bases. Dans le cas contraire, l'étape suivante suffit.
- Crédation d'une table nommée `livre` contenant six colonnes, cinq correspondant aux valeurs contenues dans un élément `<livre>` plus un indispensable identifiant numérique entier auto-incrémenté. Ce dernier sera la clé primaire de la table de façon à ne pas déroger aux règles énoncées au chapitre 13.

Pour les détails de création d'une base et d'une table, reportez-vous au chapitre 14. Le code de l'exemple 18-10 donne la commande SQL de création de la table `livre`. La figure 18-12 illustre la structure affichée par phpMyAdmin.

### Exemple 18-10. Code SQL de création de la table `livre`

```
CREATE TABLE livre (
    idlivre smallint(5) unsigned NOT NULL auto_increment,
    editeur varchar(20) NOT NULL default '',
    prix decimal(4,2) unsigned NOT NULL default '0.00',
    titre varchar(40) NOT NULL default '',
    auteur varchar(30) NOT NULL default '',
    date year(4) NOT NULL default '0000',
    PRIMARY KEY (idlivre)
)
```

Afin d'opérer ce transfert de données, vous commencez par récupérer le contenu du fichier `biblio3.xml` dans la variable objet `$xml` (repère ①). La connexion au serveur MySQL est réalisée comme aux chapitres précédents par la fonction `connex()`, dont vous devez inclure le code (repère ②) avant d'effectuer la connexion (repère ③). Vous lisez avec `count()` le nombre d'éléments `<livre>` présents dans le document XML en comptant le nombre d'éléments du tableau retourné par `$xml->livre` (repère ④). Ce nombre est utilisé par un boucle `for` (repère ⑤), qui lit l'ensemble des attributs `editeur` et `prix` des éléments `<livre>` (repères ⑥ et ⑦) et le contenu des sous-éléments `<titre>`, `<auteur>` et `<date>` (repères ⑧, ⑨ et ⑩).

Champ	Type	Attributs	Null	Défaut	Extra
<code>idlivre</code>	<code>smallint(5)</code>	<code>UNSIGNED</code>	Non		<code>auto_increment</code>
<code>editeur</code>	<code>varchar(20)</code>		Non		
<code>prix</code>	<code>decimal(4,2)</code>	<code>UNSIGNED</code>	Non	0.00	
<code>titre</code>	<code>varchar(40)</code>		Non		
<code>auteur</code>	<code>varchar(30)</code>		Non		
<code>date</code>	<code>year(4)</code>		Non	0000	

## Figure 18-12

### Structure de la table livre

La requête d'insertion des données dans la table `livre` est écrite dans la variable `$requete` (repère ⑪) puis soumise au serveur (repère ⑫). Une vérification est opérée pour s'assurer que les données sont insérées dans la table (repère ⑬).

#### Exemple 18-11. Transfert de données XML dans une table

```
<?php
$xml=simplexml_load_file("biblio3.xml"); ←①
// Connexion à la base
include_once("exemple15.2.php"); ←②
$idcom= connexobjet("biblio","mtparam"); ←③
// Lecture du contenu des éléments
$nblivre= count($xml->livre); ←④
echo "Nombre de livres : ",$nblivre,"<br />";
for($i=0;$i<$nblivre;$i++) ←⑤
{
    echo "NB = ",$i;
    $editeur=$xml->livre[$i] [@editeur]; ←⑥
    $prix=$xml->livre[$i] [@prix]; ←⑦
    $titre=htmlentities($xml->livre[$i]->titre); ←⑧
    $auteur=htmlentities($xml->livre[$i]->auteur); ←⑨
    $date=$xml->livre[$i]->date; ←⑩
    // Requête SQL
    $requete= "INSERT INTO livre(idlivre,editeur,prix,titre,auteur,date)
VALUES ('\N','$editeur','$prix','$titre','$auteur','$date')"; ←⑪
    //Envoi de la requête d'insertion
    $verif=$idcom->query($requete); ←⑫
    if($verif) echo "<br /> $titre de $auteur a été inséré dans la base<br />"; ←⑬
}
?>
```

## Mémo des fonctions et méthodes

object simplexml\_import\_dom(nodedom node)

Crée un objet `simplexml_element` à partir d'un objet DOM.

object simplexml\_load\_file(string nom\_fichier)

Transfère le contenu d'un fichier XML dans un objet de type `simplexml_element`.

object simplexml\_load\_string(string chaine\_xml)

Transfère un contenu XML d'une chaîne de caractères dans un objet de type `simplexml_element`.

string asxml([string nom\_fichier])

Transfère le contenu d'un objet de type `simplexml_element` dans une chaîne. Si vous fournissez un paramètre, la méthode enregistre le fichier sur le serveur.

object attributes()

Retourne un objet dont les propriétés sont les attributs de l'objet `simplexml_element` auquel vous appliquez la méthode.

---

```
object children()
```

Retourne un objet dont les propriétés sont les éléments enfants de l'élément auquel vous appliquez la méthode.

---

```
array xpath(string requete_path)
```

Effectue une requête XPath sur les données de l'objet auquel vous appliquez la méthode.

## Exercices

### Exercice 1

Créez un fichier XML nommé `iut.xml`, dont l'élément racine est `<iut>`. Les éléments principaux nommés `<etudiant>` ont comme attributs `id` (numéro d'inscription) et `nom`. Chaque élément `<etudiant>` peut contenir autant d'éléments `<uv>` que désiré. Chaque UV doit avoir un nom, une durée et une note enregistrés dans des sous-éléments. Visualisez ce fichier dans un navigateur pour vérifier qu'il est bien formé.

### Exercice 2

Lisez les éléments et les attributs du fichier `iut.xml`, et affichez-les dans un tableau HTML.

### Exercice 3

Créez un formulaire permettant d'insérer des données dans le fichier `iut.xml`. Le script doit permettre la visualisation éventuelle du fichier après l'insertion.

### Exercice 4

Créez un formulaire de recherche permettant d'afficher à la demande les noms des étudiants par ordre alphabétique, ainsi que la liste des UV et leur nom.

### Exercice 5

Transférez toutes les données de la base `voitures` créée aux chapitres 13 et 14 dans un fichier XML d'abord en utilisant phpMyAdmin puis en écrivant un script PHP. La répartition des données dans des attributs ou comme contenu des éléments est libre.

### Exercice 6

Transférez les données du fichier `iut.xml` de l'exercice 3 dans une table MySQL. Créez la table auparavant en lui donnant comme clé primaire la valeur de l'attribut `id` de l'étudiant.

# 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 de fin d’études sur PHP. Chacun d’eux consiste à créer un site de complexité croissante. Des 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/php7>.

## 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.
6. Effectuer des tests approfondis sur chaque page en essayant d’envisager les divers comportements des utilisateurs, leurs erreurs éventuelles et les effets de celles-ci sur votre code.
7. Mettre en place l’ensemble des pages et tester leurs interactions dans les mêmes conditions.

8. Si possible, faire tester le site par des personnes n'ayant aucune connaissance particulière du domaine et noter leur comportement vis-à-vis de l'interface proposée. Noter en particulier leurs hésitations et les questions qu'elles se posent pour accéder au service rendu par le site. Ce qui est évident pour le concepteur ne l'est pas forcément pour les utilisateurs.

## TP n° 1. Un site de rencontres

Ce premier thème, déjà ébauché au chapitre 16, est le plus simple.

Son cahier des charges est le suivant :

- Le site offre aux internautes sportifs la possibilité d'entrer en contact avec d'autres personnes pratiquants ou supporters d'un même sport dans un département donné.
- Le site est réalisé avec PHP et une base SQLite.
- Chaque page affiche un en-tête commun.
- Pour avoir accès aux informations du site, chaque visiteur s'enregistre au préalable en tant que pratiquant ou supporter d'au moins un sport.
- L'identification se fait par le biais de l'e-mail du visiteur. Une fois identifié, le visiteur a accès à une page de recherche affichant les coordonnées des personnes qui répondent aux critères qu'il a définis. L'autorisation d'accès et l'e-mail sont stockés dans un cookie.
- Un visiteur non enregistré souhaitant accéder à la page de recherche est redirigé automatiquement vers la page d'inscription.
- Si un visiteur déjà identifié veut s'inscrire pour un autre sport que celui de sa première inscription, le formulaire affiche ses coordonnées automatiquement dans le formulaire afin de lui faciliter la saisie.

### *L'interface*

L'interface comprend trois pages, la page d'accueil, la page d'inscription et la page de recherche, chacune dotée de fonctionnalités spécifiques.

#### **La page d'accueil**

Nommée `index.php`, la page d'accueil contient les éléments suivants :

- En-tête commun.
- Liste des sports existants dans la base.
- Zone de saisie de l'e-mail pour identifier le visiteur. Si l'e-mail figure déjà dans la base, un message de bienvenue s'affiche avec le nom du visiteur, et les données personnelles du visiteur sont enregistrées dans un cookie. Deux nouveaux liens sont

créés dynamiquement, un vers la page de recherche et un vers la page d'inscription, permettant de s'enregistrer pour un nouveau sport. Si l'e-mail ne figure pas dans la base, le visiteur est redirigé automatiquement vers la page d'inscription.

- Lien vers la page d'inscription pour les personnes non encore enregistrées.

## La page d'inscription

Nommée `ajout.php`, la page d'inscription contient les éléments suivants :

- En-tête commun.
- Formulaire HTML d'enregistrement comprenant trois zones principales :
  - La première comporte les zones de saisie de texte pour le nom, le prénom, le département et l'e-mail.
  - La deuxième contient une zone de sélection proposant le choix des sports existant dans la table `sport`. Cette liste est construite dynamiquement à partir des sélections des sports existants dans la base par les utilisateurs. Une seconde liste de sélection permet à l'utilisateur de choisir son niveau. Les choix possibles sont « débutant », « confirmé », « pro » ou « supporter ». Une zone de saisie de texte et un bouton d'envoi particulier permettent au visiteur d'ajouter un nouveau sport dans la table si l'il n'est pas proposé dans la liste. Après l'enregistrement du nouveau sport, le visiteur est dirigé vers la page d'inscription mise à jour avec ce nouveau sport.
  - Le formulaire se termine par les habituels boutons d'envoi et de réinitialisation.
- Lien vers la page d'accueil.
- Script vérifiant l'existence de saisies dans les zones de texte et les listes de sélection, enregistrant les données dans la base `sportifs` et affichant l'identifiant généré.

La figure 19-1 illustre ce que pourrait être le formulaire de la page d'inscription.

## La page de recherche

Nommée `recherche.php`, la page de recherche contient les éléments suivants :

- En-tête commun.
- Formulaire de saisie contenant trois listes de sélection : – La première contient la liste des sports existants. Elle est construite dynamiquement à partir des données de la table `sport`, comme dans la page d'inscription.
  - La deuxième indique le niveau des pratiquants et comporte les mêmes valeurs que dans le formulaire d'inscription.
  - La troisième contient la liste des départements dans lesquels il existe des personnes inscrites. Elle est construite dynamiquement à partir des données de la

table personne.

The screenshot shows a Mozilla Firefox browser window with the title bar "Insertion des données - Mozilla". The menu bar includes File, Edit, View, Go, Bookmarks, Tools, Window, and Help. The toolbar has icons for Home, Bookmarks, mozilla.org, mozillaZine, and mozdev.org. The main content area displays a registration form:

**Vos coordonnées**

Nom :   
Prénom :   
Département :   
Mail :

**Vos pratiques sportives**

Sport pratiqué :  OU : Ajouter un sport à la liste    
Niveau :

At the bottom, there are standard browser navigation buttons (Back, Forward, Stop, Home, etc.) and a "Done" button.

**Figure 19-1**

*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 HTML. 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 19-2.

Recherche de partenaires - Mozilla

File Edit View Go Bookmarks Tools Window Help

Home Bookmarks mozilla.org mozillaZine mozdev.org

**Le site des rencontres sportives**

Rechercher des partenaires

Sport pratiqué : tennis

Niveau : Débutant

Département : Choisissez!

Recherche

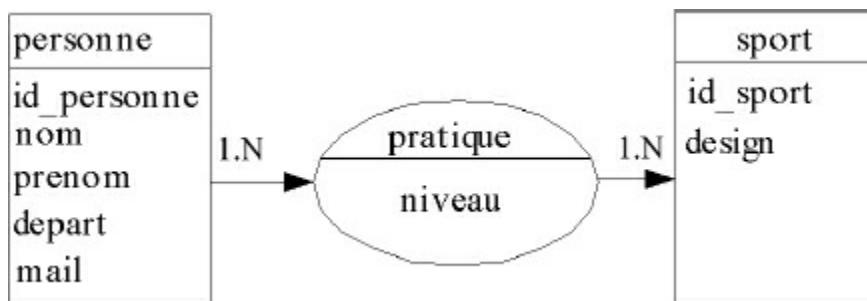
**Figure 19-2**  
*Le formulaire de recherche*

## ***La base de données SQLite***

La définition des besoins pour la conception de la base de données sportifs a déjà été abordée au chapitre 15. Les contraintes à respecter sont les suivantes :

- Les entités en présence sont une personne et un sport.
- Une personne peut pratiquer un ou plusieurs sports. La cardinalité du côté de l'entité personne est donc 1.N.
- Un sport peut être pratiqué par une ou plusieurs personnes. La cardinalité du côté de l'entité sport est donc également 1.N.
- Ces entités sont reliées par l'association pratique, qui possède l'attribut niveau.

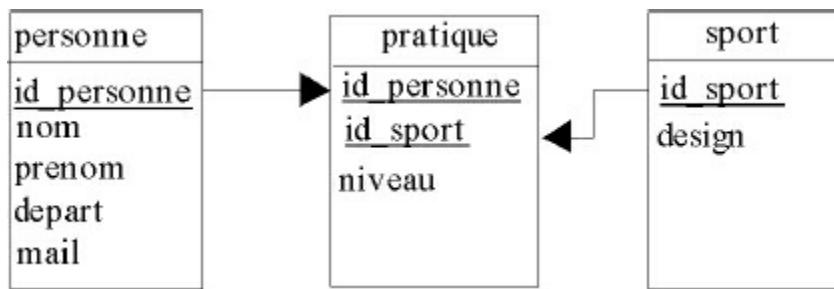
Le modèle conceptuel de données, ou MCD, est illustré à la figure 19-3.



**Figure 19-3**  
*Le modèle conceptuel de données de la base*

En appliquant les règles de normalisation présentées au chapitre 13, vous obtenez le

modèle logique de données illustré à la figure 19-4. La table `pratique` représente l'association entre les tables `personne` et `sport`. Sa clé primaire est la concaténation des clés primaires des tables qu'elle associe. Elle a de plus un attribut `niveau`.



**Figure 19-4**  
*Le modèle logique de données (MLD) de la base*

## TP n° 2. Dictionnaire de citations interactif

Ce TP est une première mise en œuvre simple d'une base de données MySQL. Le projet consiste à créer un dictionnaire de citations littéraires interactif en ligne. Il ne s'agit pas donc d'une banque de données statique mise en consultation. Chaque visiteur peut en enrichir le contenu avec ses citations préférées, qui sont ensuite rendues accessibles à tous. Le concept du site se rapproche de celui d'un forum puisque les données ne sont pas figées.

### *L'interface*

Pour créer une unité dans le site, chaque page doit incorporer les mêmes en-tête et pied de page. L'interface comprend trois pages, la page d'accueil, la page d'affichage des résultats et la page d'insertion.

### La page d'accueil

Nommée `index.php`, la page d'accueil comporte, outre les éléments décoratifs laissés à votre libre choix, les éléments suivants :

- Bandeau contenant la citation du jour tirée au sort dans la base et affichée lors de chaque connexion.
- Formulaire de recherche contenant une zone de saisie de texte dans laquelle le visiteur saisit un mot-clé de recherche d'une citation. Il peut aussi préciser sa recherche en choisissant dans une liste de sélection parmi les auteurs présents dans la base. Cette liste est construite dynamiquement en interrogeant la base. Un dernier critère de sélection est constitué d'une seconde liste de sélection permettant de choisir le siècle des citations, du XVI<sup>e</sup> au XXI<sup>e</sup>. Le critère de tri des résultats se fait par auteur ou par siècle en fonction du choix effectué à l'aide de boutons radio. Aucun de ces critères n'étant obligatoire, chaque choix doit posséder une valeur par

défaut consistant à afficher l'ensemble des citations. Le script de traitement de ce formulaire se trouve sur la page d'affichage des résultats.

- Lien vers la page d'insertion de nouvelles citations.

## La page d'affichage des résultats

Nommée `affichecit.php`, la page d'affichage des résultats contient les éléments suivants :

- Script gérant les saisies du formulaire. Ce script construit la requête SQL dynamiquement en fonction des choix opérés par le visiteur dans la page de recherche et gère l'absence de mot-clé et de choix dans les listes de sélection afin de ne pas créer de blocage du fait d'une requête mal construite.
- Résultats de la recherche effectuée par un visiteur. Chaque citation est présentée dans une cellule de tableau HTML et est suivie du nom de l'auteur et de son siècle. Le tri des citations se fait par siècle ou par nom d'auteur selon le choix fait par le visiteur.
- Lien vers la page d'accueil.
- Lien vers la page d'insertion.

## La page d'insertion

Nommée `saisiecit.php`, la page d'insertion comprend les éléments suivants :

- Formulaire contenant deux zones de saisie de texte pour le nom et le prénom de l'auteur, une liste de sélection du siècle, une zone de saisie multiligne pour le texte de la citation, ainsi que les habituels boutons d'effacement et d'envoi.
- Script de traitement des données situé dans le fichier lui-même, devant vérifier si l'auteur existe déjà dans la base puis insérer les données et afficher un avis d'insertion pour le visiteur.
- Lien vers la page d'accueil.

La figure 19-5 donne une idée de ce que pourrait être le formulaire d'insertion (ici pour les rencontres sportives du TP n° 1).

Insertion des données - Mozilla

File Edit View Go Bookmarks Tools Window Help

Home Bookmarks mozilla.org mozillaZine mozdev.org

### Le site des rencontres sportives

Vos coordonnées

Nom :

Prénom :

Département :

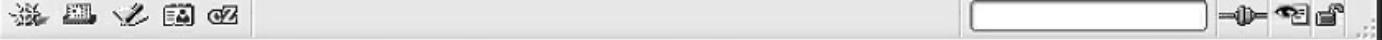
Mail :

Vos pratiques sportives

Sport pratiqué : Choisissez!  OU : Ajouter un sport à la liste  Ajouter

Niveau : Débutant

Rechercher des partenaires

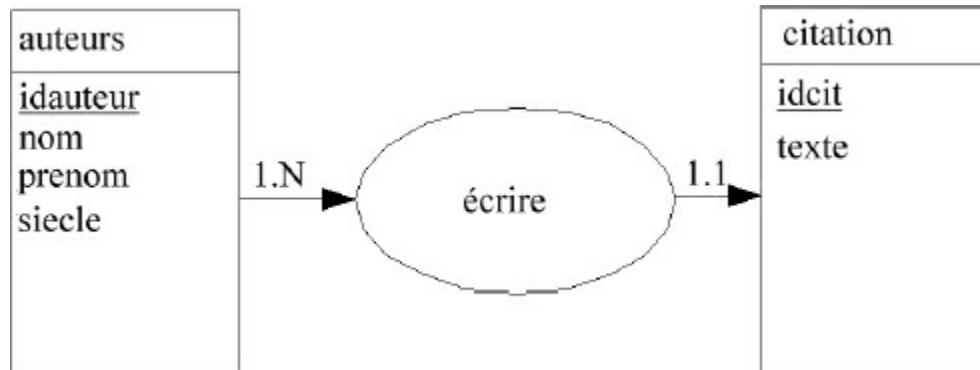


**Figure 19-5**  
*Le formulaire d'insertion*

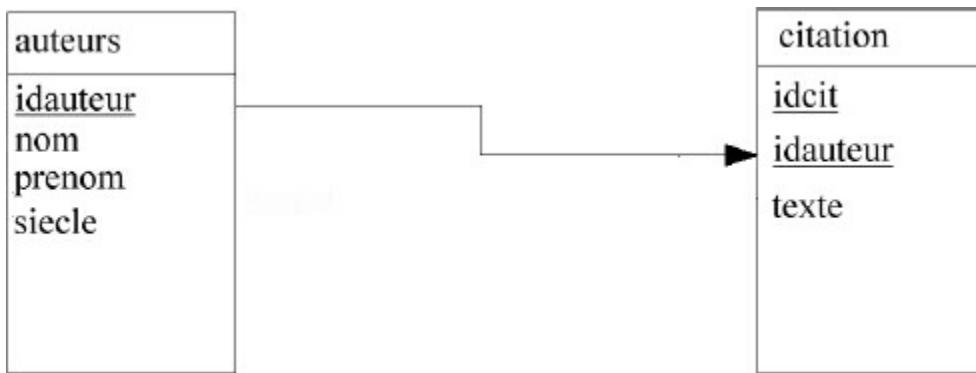
## *La base de données MySQL*

Nommée `dico`, la base de données doit répondre au modèle conceptuel de données représenté à la figure 19-6. Les contraintes sont les suivantes :

- Un auteur peut avoir écrit plusieurs citations.
- Une citation donnée ne peut être l'œuvre que d'un seul auteur.



**Figure 19-6**  
*Le modèle conceptuel de données de la base dico*



**Figure 19-7**

*Le modèle logique de données de la base dico*

Le modèle logique de données réalisé en appliquant les règles du chapitre 13 est illustré à la figure 19-7. Il montre que la base `dico` ne contient que deux tables.

## TP n° 3. Commerce en ligne

Ce troisième thème a pour but de vous placer dans une situation professionnelle.

Vous y jouez le rôle d'un concepteur indépendant vis-à-vis d'un client qui n'est pas un professionnel du Web mais un commerçant. Il ne peut donc exprimer ses besoins dans un langage technique propre à Internet et à la programmation PHP, dont il ignore tout.

### *Les besoins du client*

Le client dirige une PME qui vend des produits informatiques et vidéo. Vous devez lui créer un site de commerce en ligne pour vendre ses produits par correspondance. Son budget ne lui permettant pas de vous rémunérer en permanence pour maintenir le site, vous lui créez une interface Internet à accès privé lui permettant d'ajouter ou d'enlever des produits dans la base de données sans avoir à passer par un logiciel FTP.

### *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 15)

ou, encore mieux, pour les courageux, réalisez une version mysqli puis une version PDO (voir le chapitre 16).

## Fonctionnement du site

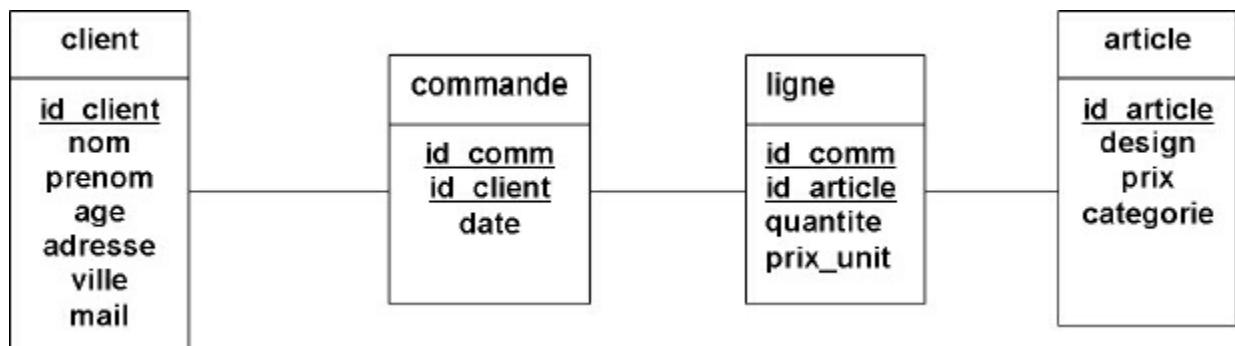
Le site répond aux conditions suivantes :

- 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 un algorithme de vérification du numéro. Vous trouverez cet algorithme sur Internet en faisant une recherche à partir du mot-clé clé de Luhn.
- Si le paiement est bien réalisé, vous finalisez la transaction en enregistrant l'ensemble des informations dans les tables client, commande et ligne puis affichez le numéro de commande à destination du client.
- Pour finir, vous envoyez un e-mail de confirmation au client et au dépôt du magasin chargé de la livraison.

## La base de données MySQL

La base de données étant susceptible d'être soumise à de nombreux accès concurrents, le choix de MySQL s'impose par rapport à SQLite. La base magasin a déjà été présentée en détail au chapitre 13. La figure 19-8 présente le modèle conceptuel de données de

cette base.



**Figure 19-8**

*Le modèle conceptuel de données de la base magasin*

Ce modèle est imposé afin que chaque lecteur travaille sur la même base. N'ajoutez aucun attribut aux entités représentées dans le modèle. Il vous appartient de recréer le MLD correspondant avant de créer la base avec phpMyAdmin.

## Conseils

Les quelques conseils suivants vous permettront de réaliser plus facilement ce TP :

- Établissez le schéma de fonctionnement du site avant de commencer le moindre codage.
- Reportez-vous au chapitre 13 pour établir précisément le MLD de la base.
- Reportez-vous au chapitre 14 pour utiliser MySQL et revoir la partie concernant les jointures entre tables.
- Reportez-vous au chapitre 12 pour implémenter les sessions, les cookies et les e-mails.
- La création de fonctions de traitement est recommandée.
- Ne vous précipitez pas sur le corrigé en cas de problème. Laissez d'abord décanter vos idées.