

PHP, une initiation

Dominique Gonzalez
Université Lille3-Charles de Gaulle

PHP, une initiation

par Dominique Gonzalez

Publié mercredi 25 novembre 2009 à 13h58

Copyright © 2008 D.Gonzalez

Ce document est soumis à la licence GNU FDL. Permission vous est donnée de distribuer, modifier des copies de ces pages tant que cette note apparaît clairement.

Table des matières

I. Le cours.....	vii
1. Pourquoi et comment ?.....	1
1.1. Introduction.....	1
1.2. Pourquoi ce document ?	1
1.3. Comment a-t-il été construit ?	1
1.4. Où trouver ce document ?	1
1.5. QBullets	1
2. Généralités sur les langages informatiques.....	3
2.1. Qu'appelle-t-on langage informatique?	3
2.2. Langage interprété.....	3
2.3. Langage compilé	3
2.4. Langages intermédiaires.....	4
3. Des langages particuliers : les langages du web.....	5
3.1. Pourquoi programmer ?.....	5
3.2. Quels langages ?.....	5
3.3. Avantages et inconvénients.....	5
3.4. Choisir	5
4. Une FAQ élémentaire sur PHP	7
4.1. Qu'est ce que PHP ?.....	7
4.2. Quelle différence avec Javascript ?.....	7
4.3. Pourquoi choisir PHP au lieu de Javascript ?	7
4.4. Que se passe-t-il à l'affichage d'une page PHP ?	7
4.5. Comment intégrer du PHP dans du HTML ?.....	7
5. Premiers exemples en PHP	9
5.1. LE premier programme.....	9
5.2. LE deuxième programme	9
5.3. Rappels (?) d'algorithmique.....	9
5.4. Variables et expressions	10
5.5. Fonctions	11
5.6. À vous.....	11
6. Formulaires en HTML	15
6.1. Présentation	15
6.2. Principaux contrôles.....	15
6.3. Exemples	16
7. Traitement d'un formulaire en PHP	19
7.1. Traitement des données d'un formulaire en PHP.....	19
7.2. Exercices	19
8. Chaînes de caractères.....	21
8.1. Qu'est-ce qu'une chaîne de caractères ?	21
8.2. Que choisir ? Guillemets ou apostrophes ?	21
8.3. Principales utilisations du <i>backslash</i>	21
8.4. Principales opérations sur les chaînes	22
8.5. Principales fonctions	22
8.6. Conversions de types	23
8.7. Exercices	23
9. Les tableaux.....	25
9.1. Qu'est-ce qu'un tableau ?	25
9.2. Quelques exemples d'utilisation	25
9.3. Les tableaux associatifs	26
9.4. Parcours d'un tableau associatif	27
9.5. Exercices	27
10. PDO	29
10.1. Qu'est-ce que PDO.....	29
10.2. Quelle utilité ?	29
10.3. Se connecter à la base de données	29
10.4. Exécuter une requête	30
10.5. Nombre de lignes et colonnes d'une requête <i>select</i>	30
10.6. Accéder aux résultats d'une requête <i>select</i>	30
10.7. Gestion des erreurs	31
11. Programmation séparée en PHP	33
11.1. La programmation séparée	33
11.2. Les fonctions <i>require</i> et <i>include</i>	33

12. Identification en PHP	35
12.1. Préliminaires : la fonction <code>header</code>	35
12.2. Contrôle des mots de passe	35
12.3. Amélioration : plusieurs utilisateurs	36
12.4. Séparer les données du traitement	36
12.5. Cacher les mots de passe	37
12.6. Crypter les mots de passe	37
13. Sessions	39
13.1. Préambule	39
13.2. Application : identification des visiteurs	39
II. Hors programme	41
14. Utiliser PHP sur une base de données PostgreSQL	43
14.1. Connexion à la base	43
14.2. Exécution d'une requête	43
14.3. Obtention des résultats d'une requête	43
14.4. Un exemple	44
14.5. Requetes autres que <code>SELECT</code>	44
14.6. Suppression des messages d'erreurs	45
14.7. Gestion des erreurs et des messages	45
15. <code>peardb</code> , une présentation	47
15.1. Qu'est-ce que <code>pear</code>	47
15.2. Quelle utilité ?	47
15.3. Utiliser <code>peardb</code>	47
15.4. Se connecter à la base de données	47
15.5. Gestion des erreurs	48
15.6. Exécuter une requête	49
15.7. Accéder aux résultats d'une requête <code>select</code>	49
15.8. <code>setfetchmode()</code>	49
15.9. <code>peardb</code> , informations sur les requêtes	51
16. <code>dbx</code>	53
16.1. Qu'est-ce que <code>dbx</code>	53
16.2. Quelle utilité ?	53
16.3. Se connecter à la base de données	53
16.4. Exécuter une requête	54
16.5. Nombre de lignes et colonnes d'une requête <code>select</code>	54
16.6. Accéder aux résultats d'une requête <code>select</code>	54
17. Exploration du contenu d'un répertoire	57
17.1. Les notions nécessaires en PHP	57
17.2. Les notions nécessaires en HTML	57
17.3. Exercices	58
18. News	59
18.1. Présentation	59
18.2. Votre travail	59
18.3. Si vous avez le temps	60
19. Les fichiers	61
19.1. Manipulations de base	61
19.2. Un exemple	61
19.3. Exercice	63
III. Corrigés des exercices	65
20. Premiers exercices d'algorithmique, corrigés	67
21. Premiers formulaires en PHP, corrigés	73
22. Chaînes de caractères, corrigé	81
23. Exercices sur les tableaux, corrigés	83
24. Exercices sur les fichiers, corrigés	85
25. Exploration d'un répertoire, corrigés	87
26. News..., corrigé	89
26.1. Accueil	89
26.2. Liste des noms de fichiers de nouvelles	89
26.3. Liste des liens vers les fichiers de nouvelles	89
26.4. Affichage de toutes les nouvelles	90
26.5. Affichage des titres des nouvelles	90
26.6. Affichage des titres faisant lien vers les nouvelles	91
26.7. Affichage de toutes les nouvelles, proprement	91

26.8. Choix du sujet.....	91
IV. Études de cas	93
27. Études de cas.....	95
28. Projet <i>Disques 2009</i>	97
28.1. Présentation	97
28.2. La base de données.....	98
29. Projet <i>Inscriptions</i>	101
29.1. Présentation	101
29.2. La base de données.....	101
30. Projet <i>Tenirraq</i>	105
30.1. Présentation	105
30.2. La base de données.....	106
31. Projet <i>Camping</i>	111
31.1. Présentation	111
31.2. Les informations à conserver	111
31.3. Les traitements	111
31.4. La base de données.....	112
32. Projet <i>Association</i>	115
32.1. Contenu du site web.....	115
32.2. Conditions de travail.....	115
32.3. Base de données	115
32.4. MLD du projet <i>Association</i>	116
32.5. Création des tables de la base <i>Association</i>	116
32.6. Évaluation	117
32.7. Dernier conseil	118
33. Projet <i>Généalogie</i>	119
33.1. Généralités	119
33.2. Les données à conserver	119
33.3. Traitement des données	119
33.4. Optimisations	120
33.5. Import-export	120
33.6. MCD du projet <i>généalogie</i>	120
33.7. Les différentes tables	121
33.8. Les différentes contraintes.....	123
34. Projet <i>Brazil</i>	127
34.1. Description.....	127
34.2. Les tables des entités	127
34.3. Les tables des relations.....	128
34.4. Les contraintes.....	129
35. Projet <i>Services</i>	133
35.1. But de ce projet.....	133
35.2. Fonctionnement	133
35.3. Informations plus techniques.....	134
35.4. Question subsidiaire : comment gérer l'historique de la base ?.....	134
35.5. Vocabulaire utilisé.....	134
35.6. Première ébauche de la structure	135
35.7. Structures des tables.....	136
35.8. Création des tables.....	138
35.9. Remplir les tables de paramètres.....	139
36. Projet <i>Disques</i>	143
36.1. Approche naïve	143
36.2. Analyse.....	143
36.3. Dernières remarques	145
36.4. Une autre analyse du projet <i>Disques</i>	146
36.5. Création des tables.....	148
36.6. Votre travail	149
Index	151

I. Le cours

Chapitre 1. Pourquoi et comment ?

1.1. Introduction

Ces pages sont destinées à des personnes ayant déjà programmé. Il ne s'agit absolument pas d'un cours d'algorithmique. En particulier: les notions de bases de l'algorithmique (séquence, alternative, itération) sont supposées connues et comprises. On ne traitera ici que de leur traduction en PHP.

Les notions abordées seront:

- Les langages informatiques, et en particulier les langages du web, en répondant à la question « pourquoi programmer pour le web ? ».
- Les notions classiques en programmation, abordées sous l'angle PHP : les structures de contrôle, les chaînes de caractères, les tableaux, etc.
- Les particularités de la programmation pour le web: les formulaires et leur traitement, l'identification.
- Nous terminerons par la mise en relation de pages web avec une base de données (PostgreSQL).

1.2. Pourquoi ce document ?

Ces pages ont pour origine un cours destiné aux étudiants de 2^{ème} année de l'IUP IIES de l'université de Lille III-Charles de Gaulle, à Villeneuve d'Ascq, pour les années universitaires 2002-2003 et 2003-2004.

Elles ont ensuite été peu à peu remaniées et augmentées à l'occasion d'un cours destiné aux étudiants de 3^{ème} année de la licence MIASHS.

Ces pages ne sont pas destinées à être un cours autonome : elles ne sont qu'un support de cours, et beaucoup de choses, qui sont transmises à l'oral pendant les cours, ne sont pas écrites.

L'environnement technique du cours est constitué de machines sous `linux`. L'installation de ces logiciels ne sera pas abordée et ne fait pas partie du contenu du cours.

1.3. Comment a-t-il été construit ?

Ce polycopié a été rédigé au format DocBook :

- Le texte source a été écrit au format XML avec `emacs` et `Quanta`, en respectant la DTD de DocBook.
- Le code source a été compilé au format PDF avec `openjade` et au format HTML avec `xsltproc`.
- La version que vous avez devant les yeux a été compilée le mercredi 25 novembre 2009 à 13h58 .

1.4. Où trouver ce document ?

Ce document est disponible sous plusieurs formats sur le web:

- Un seul document HTML : <http://grappa.univ-lille3.fr/polys/php/php.html> (Lourd à charger, mais facile à sauvegarder ou à imprimer)
- Plusieurs pages HTML : <http://grappa.univ-lille3.fr/polys/php/index.html> (Plus faciles à consulter)
- Une version HTML sans feuilles de style : <http://grappa.univ-lille3.fr/polys/php/book1.htm> (Quelle idée ? Mais si vous y tenez...)
- PDF : <http://grappa.univ-lille3.fr/polys/php/php.pdf>

1.5. QBullets

Les petites images animées qui illustrent les liens de la version web de ce document proviennent de QBullets



1. <http://www.matterform.com/>
2

Chapitre 2. Généralités sur les langages informatiques

Note : Ce document issu de [CommentCaMarche.net](http://www.commentcamarche.net)¹ est soumis à la licence GNU FDL. Permission vous est donnée de distribuer, modifier des copies de cette page tant que cette note apparaît clairement.

2.1. Qu'appelle-t-on langage informatique?

On appelle langage informatique un langage destiné à décrire l'ensemble des actions consécutives qu'un ordinateur doit exécuter. Les langages naturels (l'anglais, le français) représentent l'ensemble des façons qu'ont un groupe d'individu de communiquer. Les langages servant aux ordinateurs à communiquer n'ont rien à voir avec des langages informatiques, on parle dans ce cas de protocoles, ce sont deux notions totalement différentes. Un langage informatique est une façon pratique pour nous (humains) de donner des instructions à un ordinateur.

Un langage informatique est rigoureux : à CHAQUE instruction correspond UNE action du processeur.

Le langage utilisé par le processeur, c'est-à-dire les données telles qu'elles lui arrivent, est appelé langage machine. Il s'agit d'une suite de 0 et de 1 (du binaire) mais pour plus de « clarté » il peut être décrit en hexadécimal. Toutefois le langage machine n'est pas compréhensible facilement par l'humain moyen.

Ainsi il est plus pratique de trouver un langage intermédiaire, compréhensible par l'homme, qui sera ensuite transformé en langage machine pour être exploitable par le processeur.

L'assembleur est le premier langage informatique qui ait été utilisé. Celui-ci est encore très proche du langage machine mais il permet déjà d'être plus compréhensible. Toutefois un tel langage est tellement proche du langage machine qu'il dépend étroitement du type de processeur utilisé (chaque type de processeur peut avoir son propre langage machine). Ainsi un programme développé pour une machine ne pourra pas être porté sur un autre type de machine (on désigne par portable un programme qui peut être utilisé sur un grand nombre de machines). Pour pouvoir l'utiliser sur une autre machine il faudra alors parfois réécrire entièrement le programme !

Un langage informatique a donc plusieurs avantages :

- il est plus facilement compréhensible que le langage machine,
- il permet une plus grande portabilité, c'est-à-dire une plus grande facilité d'adaptation sur des machines de types différents.

Les langages informatiques peuvent grossièrement se classer en deux catégories : les langages interprétés et les langages compilés.

2.2. Langage interprété

Un langage informatique est par définition différent du langage machine. Il faut donc le traduire pour le rendre intelligible du point de vue du processeur. Un programme écrit dans un langage interprété a besoin d'un programme auxiliaire (l'interpréteur) pour traduire au fur et à mesure les instructions du programme.

2.3. Langage compilé

Un programme écrit dans un langage dit « compilé » va être traduit une fois pour toutes par un programme annexe (le compilateur) afin de générer un nouveau fichier qui sera autonome, c'est-à-dire qui n'aura plus besoin d'un programme autre que lui pour s'exécuter (on dit d'ailleurs que ce fichier est exécutable).

Un programme écrit dans un langage compilé a comme avantage de ne plus avoir besoin, une fois compilé, de programme annexe pour s'exécuter. De plus, la traduction étant faite une fois pour toute, il est plus rapide à l'exécution. Toutefois il est moins souple que programme écrit avec un langage interprété car à chaque modification du fichier source (fichier intelligible par l'homme : celui qui va être compilé) il faudra recompiler le programme pour que les modifications prennent effet.

D'autre part, un programme compilé a pour avantage de garantir la sécurité du code source. En effet, un langage interprété, étant directement intelligible (lisible), permet à n'importe qui de connaître les secrets de

1. <http://www.commentcamarche.net/>

fabrication d'un programme et donc de copier le code voire de le modifier. Il y a donc risque de non-respect des droits d'auteur. D'autre part, certaines applications sécurisées nécessitent la confidentialité du code pour éviter le piratage (transaction bancaire, paiement en ligne, communications sécurisées, etc.).

2.4. Langages intermédiaires

Certains langages appartiennent en quelque sorte aux deux catégories (LISP, Java, Python, etc.) car le programme écrit avec ces langages peut dans certaines conditions subir une phase de compilation intermédiaire vers un fichier écrit dans un langage qui n'est pas intelligible (donc différent du fichier source) et non exécutable (nécessité d'un interpréteur). Les applets Java, petits programmes insérés parfois dans les pages Web, sont des fichiers qui sont compilés mais que l'on ne peut exécuter qu'à partir d'un navigateur internet (ce sont des fichiers dont l'extension est « class »).

Chapitre 3. Des langages particuliers : les langages du web

3.1. Pourquoi programmer ?

Une réponse générale s'impose : pour obtenir des pages web dynamiques.

Attention, parler de « pages web dynamiques » ne signifie pas qu'on parle nécessairement de pages où tout bouge dans tous les sens. Il s'agit simplement de pages qui n'ont pas d'existence statique, et qui sont construites à la volée, au moment où elles sont demandées au serveur.

Leur utilité se fait sentir quand on doit écrire des pages qui doivent s'adapter (besoin de l'heure, accès à des bases de données, réponse personnalisée, traitement de formulaire, etc.), mais aussi pour faciliter la gestion d'un site (de grande taille par exemple), séparation du contenu (la structure) et du traitement (la présentation).

3.2. Quels langages ?

Beaucoup. Pratiquement tous les langages de programmation peuvent être utilisés, et il existe des langages spécifiques à la programmation web.

On les classe en deux catégories :

- Exécution côté serveur : PHP (syntaxe de type C), ASP (serveur Microsoft), JSP (syntaxe Java), CGI (tout langage, toute plateforme), etc.
- Exécution côté client : Javascript (syntaxe Java), applet Java, Flash, etc.

3.3. Avantages et inconvénients

	Exécution côté serveur	Exécution côté client
<i>Code source</i>	Pas visible : on n'obtient que le résultat de l'exécution	Pour Javascript : code source visible dans le source de la page (Ne jamais confier à javascript le contrôle des mots de passe : ils devraient alors être écrits en clair dans le code source de la page web). Pour Applet : précompilé, donc non lisible.
<i>Rapidité dans l'exécution</i>	Dépend de la vitesse et de la charge du serveur.	Dépend de la vitesse et de la charge du client.
<i>Compatibilité</i>	Totale : le client ne reçoit que du HTML.	Problématique : le client doit posséder le logiciel capable d'exécuter le script (par exemple la bonne version de flash ou de java).
<i>Désactivation par le client</i>	Impossible, le client n'a pas le contrôle du serveur.	Très simple, il suffit de désactiver le logiciel.
<i>Contrôles de sécurité</i>	Parfaits : tout se passe sur le serveur	Inexistants : tout se passe sur le client (Ne jamais confier au seul javascript le contrôle de validité des données d'un formulaire, ou un quelconque contrôle de sécurité).

3.4. Choisir

Lequel choisir ? Bof ! Ça dépend de ce qu'on veut faire. La seule chose certaine c'est que les fonctions importantes (vitales pour l'affichage ou mettant en jeu la sécurité du site) ne doivent pas être confiées à un script

qui s'exécute côté client.

À quoi ça ressemble ? Vous trouverez de quoi tester tout ça dans les pages de Fabien Torre¹.

1. <http://grappa.univ-lille3.fr/~torre/guide.php?id=tpprogweb>

Chapitre 4. Une FAQ élémentaire sur PHP

Note : Ce qui suit est un extrait condensé provenant de deux sources :

- la FAQ PHP, maintenue par Armand Delcros¹
- A brief PHP Tutorial²

4.1. Qu'est ce que PHP ?

PHP est un langage de scripting embarqué dans les pages HTML et traité par le serveur. PHP permet de construire dynamiquement des pages HTML contenant les résultats de calculs ou de requêtes SQL adressées à un système de gestion de bases de données (SGBD).

4.2. Quelle différence avec Javascript ?

Javascript est lui aussi en effet intégré dans les pages HTML mais Javascript est interprété par le client Web alors que PHP est directement interprété par le serveur Web (s'il est inclus comme étant un module du serveur web). Le client web reçoit donc directement du HTML et ne voit jamais apparaître le code PHP.

4.3. Pourquoi choisir PHP au lieu de Javascript ?

Il n'y a pas à choisir entre PHP ou Javascript. En réalité leur utilisation est différente : Javascript est très bien adapté à l'aspect présentation et manipulation du client Web. PHP est lui adapté à la création de page HTML dynamique et rapide. PHP permet de faire des pages HTML dynamiques, dans le sens que deux appels consécutifs à une URL peuvent donner deux pages HTML différentes. Mais chacune de ces pages est statique. On peut dire qu'au niveau du serveur la page est dynamique alors qu'elle apparaît comme étant statique au niveau du client Web. Javascript permet lui de générer une page dynamique pour le client Web, c'est une des principales différences entre ces deux outils de développement.

4.4. Que se passe-t-il à l'affichage d'une page PHP ?

L'interpréteur intégré fonctionne de la manière suivante :

- Lorsque le serveur HTTP reconnaît un fichier comme intégrant du code PHP il le parcourt avant de l'expédier au client.
- S'il rencontre une instruction PHP, il la transmet à l'interpréteur .
- L'interpréteur exécute l'instruction et transmet éventuellement les sorties au serveur.
- Celui ci redirige ces sorties vers le client.

Il importe de signaler que les programmes PHP ne sont pas transmis au client mais exécutés sur le serveur. Le poste client ne fera pas de différence entre les pages statiques et celles générées dynamiquement. Il n'y a donc pas lieu de configurer spécialement les navigateurs pour supporter PHP.

4.5. Comment intégrer du PHP dans du HTML ?

Pour que le serveur HTTP reconnaisse du code PHP il faut que :

- Le nom du document HTML se termine par le suffixe « php » et non « html ».

1. <http://perso.cybercable.fr/adeltros/docs/php/php.html>
2. http://www.linux-france.org/article/dev1/php3/tut/php3_tut.html

- Le code PHP soit délimité par les balises « `<?php` » et « `?>` ».

Bien que les possibilités du langage PHP soient étendues, il suffit d'une connaissance élémentaire du langage pour réaliser les applications Web courantes.

Ce langage dispose d'une bibliothèque de fonctions très étendue. Elle fournit en particulier un interface avec les principaux systèmes de gestion de base de données et avec le système de gestion de fichier du serveur. On se reportera à la documentation du langage³ pour la description de ces fonctions.

3. <http://php.net>

Chapitre 5. Premiers exemples en PHP

Les corrigés des exercices de ce chapitre se trouvent Chapitre 20.

5.1. LE premier programme

```
<html><body>
<?php
    echo "Hello world!";
?>

</body></html>
```

Remarque

« echo¹ » sert à produire un affichage.

Remarque

Pour que « echo » affiche le texte tel quel, ce texte doit être entouré de guillemets « " ». Sinon « echo » essaie de *comprendre*. Par exemple « echo 1+2; » provoque l’affichage de « 3 ». Tandis que « echo "1+2"; » provoque l’affichage de « 1+2 ».

Pour voir le résultat produit : `helloworld.php`²

En allant voir le code source de la page produite (sur votre navigateur) vous ne verrez que cela :

```
<html><body>
Hello world!
</body></html>
```

C’est normal : souvenez-vous que le serveur exécute le programme et n’envoie au client que le résultat, pas le programme lui-même.

5.2. LE deuxième programme

```
<html><body>
<?php
    phpinfo();
?>
</body></html>
```

Remarque

Vous obtenez tous les renseignements possibles sur la version de PHP installée sur votre machine.

Remarque

« `phpinfo`³ » est une *fonction* : une seule instruction qui permet à PHP d’effectuer *beaucoup de choses*. Nous verrons plus tard comment écrire nos propres fonctions. On reconnaît une fonction aux parenthèses qui la suivent.

Pour voir le résultat produit : `phpinfo.php`⁴

2. <http://grappa.univ-lille3.fr/polys/php/exemples/helloworld.php>

4. <http://grappa.univ-lille3.fr/polys/php/exemples/phpinfo.php>

5.3. Rappels (?) d'algorithmique

1. La structure de base d'un programme est la *séquence* : les instructions sont exécutées les unes après les autres, dans l'ordre où elles sont écrites. Les instructions sont *toujours* terminées par un point-virgule (« ; »). *Ce n'est pas un séparateur d'instructions.*
2. Il existe des structures qui permettent de rompre la séquence. Vous trouverez ci-dessous les principales. Il y en a d'autres...
3. L'*alternative*⁵ : un test est exécuté ; selon que son résultat soit OUI ou NON, une partie du programme ou une autre est exécutée.

Sa syntaxe en PHP est :

```
if (test) {  
    instructions à exécuter quand la réponse est OUI  
} else {  
    instructions à exécuter quand la réponse est NON  
}
```

4. L'*itération*⁶ (TANT QUE) : une partie du programme est exécutée tant que le résultat d'un test est OUI (le calcul est fait AVANT chaque entrée dans la boucle).

Sa syntaxe en PHP est :

```
while (test) {  
    instructions à exécuter tant que la réponse est OUI  
}
```

5. La *boucle POUR*⁷ : ce n'est qu'un cas particulier de la boucle TANT QUE.

Sa forme générale est :

```
for (avant d'entrer ; test ; action de fin de boucle) {  
    instructions à exécuter quand la réponse au test est OUI  
}
```

où « avant d'entrer » et « action de fin de boucle » sont des actions quelconques. Cette forme est *totalement équivalente* à celle-ci :

```
avant d'entrer ;  
while (test) {  
    instructions à exécuter quand la réponse au test est OUI ;  
    action de fin de boucle ;  
}
```

Mais on l'utilisera le plus souvent sous cette forme :

```
for (initialiser compteur ; test compteur ; incrémenter compteur) {  
    instructions à exécuter quand la réponse au test est OUI  
}
```

comme dans l'exemple suivant qui provoque l'affichage : « 357911131517 » :

```
for ($i=3 ; $i<19 ; $i=$i+2) {  
    echo $i;  
}
```

5.4. Variables et expressions

1. Les noms des variables sont préfixés par \$. *Il n'est pas nécessaire de déclarer les variables avant de les utiliser.* Le contrôle des types est plutôt souple...
2. L'affectation est notée « = ».
3. Les opérateurs classiques sur les nombres sont disponibles : « + » (addition), « - » (soustraction), « * » (multiplication), « / » (division).
4. Le seul opérateur sur les chaînes est la concaténation, symbolisée par un point (« . »).

5. <http://fr.php.net/manual/fr/language.control-structures.php#control-structures.if>

6. <http://fr.php.net/manual/fr/control-structures.while.php>

7. <http://fr.php.net/manual/fr/control-structures.for.php>

5. Les opérateurs de comparaison sont eux aussi (presque tous) classiques : « < » (inférieur), « <= » (inférieur ou égal), « > » (supérieur), « >= » (supérieur ou égal), « == » (égal), « != » (différent).
6. Les opérateurs logiques de base sont « || » (ou) et « && » (et).

5.5. Fonctions

Une fonction sans paramètres se définit par :

```
function nom_de_la_fonction () {
    corps_de_la_fonction
}
```

Une fonction avec paramètres se définit par :

```
function nom_de_la_fonction (paramètres) {
    corps_de_la_fonction
}
```

Les paramètres sont toujours passés par valeur.

Le résultat de la fonction est la valeur qui se trouve derrière la première instruction `return` rencontrée. Par exemple « `return 5;` ».

Il n’y a pas de procédures : si vous voulez qu’une de vos fonctions se comporte comme une procédure, il suffit de ne lui faire renvoyer aucune valeur, par exemple avec « `return;` ».

5.6. À vous

Dans les exercices qui suivent vous ferez en sorte que les résultats soient présentés proprement (je vous laisse libre choix sur le sens exact de ce mot...).

1. Écrire un programme PHP qui affiche tous les nombres impairs entre 0 et 15000, par ordre croissant : « 1 3 5 7 ... 14995 14997 14999 ».

Exemple : `impair.php`⁸

2. Écrire un programme qui écrit 500 fois « *Je dois faire des sauvegardes régulières de mes fichiers.* »

Exemple : `punition.php`⁹

3. Écrire un programme qui affiche la table de multiplication par 13.

Exemple : `multiplication.php`¹⁰ ou mieux : `multiplicationbis.php`¹¹.

4. Écrire un programme qui calcule 30!.

Exemple : `factorielle.php`¹²

5. Écrire un programme qui affiche :

```
12345678910111213
12345678910111213
12345678910111213
12345678910111213
```

Exemple : `4lignes.php`¹³

6. Écrire un programme qui affiche la table de multiplication totale de {1,...,12} par {1,...,12}.

Exemple : `multiplicationtotale.php`¹⁴

8. <http://grappa.univ-lille3.fr/polys/php/exemples/impair.php>

9. <http://grappa.univ-lille3.fr/polys/php/exemples/punition.php>

10. <http://grappa.univ-lille3.fr/polys/php/exemples/multiplication.php>

11. <http://grappa.univ-lille3.fr/polys/php/exemples/multiplicationbis.php>

12. <http://grappa.univ-lille3.fr/polys/php/exemples/factorielle.php>

13. <http://grappa.univ-lille3.fr/polys/php/exemples/4lignes.php>

14. <http://grappa.univ-lille3.fr/polys/php/exemples/multiplicationtotale.php>

7. Écrire une fonction qui renvoie $n!$. Puis utilisez cette fonction pour construire une table des factorielles.

Exemple : `tablefact.php`¹⁵.

8. Écrire une fonction qui affiche un triangle et qui admet comme paramètre le nombre de lignes du triangle :

```
*
**
***
```

Utilisez cette fonction pour dessiner un *demi-sapin* (de 2 en 2) :

```
*
**
*
**
***
****
*
**
***
****
*****
*****
```

Exemple : `demisapin.php`¹⁶

9. Coefficients du binôme :

- Écrire une fonction qui renvoie $n!$.
- Utiliser la fonction précédente pour écrire une fonction qui renvoie

$$C_n^p$$

- Utiliser les fonctions précédentes dans un programme qui affiche les coefficients du binôme pour toutes les valeurs de n dans $\{0,1,\dots,20\}$:

$$C_n^0 C_n^1 C_n^2 \dots C_n^{n-2} C_n^{n-1} C_n^n$$

Exemple : `binome.php`¹⁷

10. La suite de Fibonacci est définie par les relations suivantes :

- $F_0=0$,
- $F_1=1$,
- $F_n=F_{n-1}+F_{n-2}$ pour tout $n>1$.

Donc $F_0=0, F_1=1, F_2=0+1=1, F_3=1+1=2, F_4=2+1=3, F_5=3+2=5, F_6=5+3=8, F_7=8+5=13\dots$

Écrire un programme qui affiche les 50 premières valeurs de F_n .

Exemple : `fibonacci.php`¹⁸

11. Fibonacci, le retour : afficher les rapports

$$\frac{F_n}{F_{n-1}}$$

ainsi que leurs différences avec le nombre

15. <http://grappa.univ-lille3.fr/polys/php/exemples/tablefact.php>

16. <http://grappa.univ-lille3.fr/polys/php/exemples/demisapin.php>

17. <http://grappa.univ-lille3.fr/polys/php/exemples/binome.php>

18. <http://grappa.univ-lille3.fr/polys/php/exemples/fibonacci.php>

$$\frac{\sqrt{5}+1}{2}$$

Exemple : `fibonacci-retour.php`¹⁹

12. Afficher un cadre 10x10.

Exemple : `cadre-pre.php`²⁰ ou `cadre-table.php`²¹

13. Tableau d'additions : faire réaliser par une page PHP un tableau d'additions à compléter (pour l'entraînement d'un enfant au calcul).

Exemple : `tableauadditionsfixe.php`²² ou `tableauadditionsvariable.php`²³

19. <http://grappa.univ-lille3.fr/polys/php/exemples/fibonacci-retour.php>

20. <http://grappa.univ-lille3.fr/polys/php/exemples/cadre-pre.php>

21. <http://grappa.univ-lille3.fr/polys/php/exemples/cadre-table.php>

22. <http://grappa.univ-lille3.fr/polys/php/exemples/tableauadditionsfixe.php>

23. <http://grappa.univ-lille3.fr/polys/php/exemples/tableauadditionsvariable.php>

Chapitre 6. Formulaire en HTML

6.1. Présentation

Les formulaires de saisie permettent à l'utilisateur de fournir des informations et ainsi d'obtenir une réponse personnalisée. Les informations contenues dans les champs remplis par l'utilisateur sont transmises par le programme client au serveur qui les transmet à son tour à un programme de traitement, soit un CGI (*Common Gateway Interface*), soit un script de type PHP.

La structure d'un formulaire de saisie est simple, elle se compose d'un élément `FORM` contenant essentiellement une suite de contrôles (éléments `input`, `textarea`, `select`, `button`, etc.) mais aussi des éléments de structuration de document afin d'aligner correctement les champs d'entrée.

Globalement un formulaire se présente sous cette forme :

```
<form method="(1)" action="(2)">
  du texte, des boutons, des zones de saisie ...
  <input type="submit" value="(3)" />
</form>
```

où :

(1)

désigne la méthode à utiliser pour envoyer les informations ; c'est « GET » (par défaut ; les données du formulaire sont envoyées dans l'URL) ou « POST » (les données sont envoyées dans le corps du message, elles ne sont donc pas visibles dans l'URL).

(2)

désigne l'URL du programme (CGI, PHP, etc.) qui va traiter les données. (Ce peut être aussi `MAIL`, mais c'est déconseillé... Cela ne marche que si le navigateur de l'utilisateur est BIEN configuré. De toutes façons il faudra traiter les données après, et les stocker dans des boîtes aux lettres ne facilite pas l'automatisation de cette tâche.)

(3)

désigne le texte qui va apparaître dans le bouton d'envoi.

Dans l'affichage de la page, rien ne distingue le formulaire du reste du texte. Il est donc conseillé de le séparer du reste par la balise « `<hr />` » avant et après, ou le mettre dans un tableau aux bordures visibles.

6.2. Principaux contrôles

Les principaux contrôles sont :

- `input` : les zones de saisie, les boutons radios et les cases à cocher sont définis par ce contrôle ; la syntaxe (simplifiée) en est :

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

où :

- *type* peut être :

`text`

pour une zone de saisie au sens habituel du terme (une zone rectangulaire où l'utilisateur écrit des données, sur une seule ligne) ; c'est ce qui sera dans cette zone qui sera envoyé au programme de traitement ;

`password`

pour une zone de saisie de mot de passe (identique à `text`, mais le texte saisi n'apparaît à l'écran que sous la forme d'astérisques, pour éviter d'être lisible par une tierce personne) ;

`checkbox`

pour une case à cocher ;

radio

pour un bouton radio (des boutons radios de même nom sont mutuellement exclusifs) ;

submit

pour un bouton d'envoi ; c'est le clic sur ce bouton qui envoie le contenu du formulaire au programme de traitement ;

hidden

pour une variable cachée ; permet de cacher des valeurs nécessaires au traitement mais qu'on ne veut pas voir affichées à l'écran ; attention, *caché* ne veut pas dire *secret* : ces valeurs sont visibles dans le code source de la page ;

reset

pour remettre les zones de saisie à leurs valeurs par défaut (les valeurs qu'elles ont lors du chargement de la page) ;

- *nom* est le nom de la variable qui sera envoyé au programme ; cela n'a *a priori* pas de sens pour un bouton *submit* ou *reset*, mais c'est absolument nécessaire pour les autres ; les zones doivent en principe avoir des noms différents, sauf pour les boutons radios : les boutons radios de même nom sont mutuellement exclusifs ;
- *value* est la valeur du contrôle :
 - pour *text* et *password* il s'agit d'une valeur qui permet de pré-remplir la zone ;
 - pour *checkbox* et *radio* c'est la valeur qui sera donnée à la variable si la case est sélectionnée ;
 - pour *submit* et *reset* c'est le texte qui sera écrit dans le bouton ;
 - pour *hidden* c'est la valeur qui sera donnée à la variable.
- *select* : pour créer des listes déroulantes ; exemple d'utilisation :

```
<select name="menu">
  <option>premier choix</option>
  <option>deuxième choix</option>
  <option>troisième choix</option>
</select>
```

Cette zone permettra d'envoyer une variable de nom `menu` qui aura pour valeur le choix sélectionné. Pour pré-sélectionner un choix dans la liste il suffit d'ajouter l'attribut `selected="selected"` dans la balise `<option>` correspondante.

On peut avoir besoin d'envoyer une valeur différente de ce qui est affichée (par exemple demander à l'utilisateur de choisir une personne par ses nom et prénom, et envoyer l'identifiant de cette personne dans une table d'une base de données). On utilisera alors l'attribut `value` dans la balise `<option>` comme par exemple :

```
<select name="responsable">
  <option value="1"> marcel durand</option>
  <option value="2"> georges dupont</option>
  <option value="3"> pierre martin</option>
</select>
```

Dans cet exemple si on choisit `georges dupont` la variable `responsable` aura la valeur `2`.

- *textarea* : pour des zones de saisies plus grandes qu'avec *input* ; exemple d'utilisation :

```
<textarea name="nom" rows="4" cols="40">Texte par défaut...</textarea>
```

6.3. Exemples

Vous pourrez trouver deux exemples (presque identiques, sauf pour la méthode d'envoi : `GET` pour l'un, `POST` pour l'autre) aux adresses suivantes :

<http://grappa.univ-lille3.fr/~gonzalez/prog/test01.html>

<http://grappa.univ-lille3.fr/~gonzalez/prog/test02.html>

Vous pouvez tester vos propres formulaires en suivant les instructions de la page :

<http://grappa.univ-lille3.fr/~torre/Enseignement/TPs/Formulaires>

Votre formulaire doit contenir un attribut caché `identifiant` et le champ `action` doit contenir la valeur :

<http://grappa.univ-lille3.fr/~torre/Enseignement/TPs/Formulaires/universel.php>

Chapitre 7. Traitement d'un formulaire en PHP

Les corrigés des exercices de ce chapitre se trouvent Chapitre 21.

7.1. Traitement des données d'un formulaire en PHP

Tout programme PHP peut recevoir des variables.

Si on connaît par avance le nom des variables qui seront passées au programme, il suffit de les appeler par leur nom (avec la syntaxe PHP, c'est-à-dire précédé d'un \$).

Par exemple l'appel de l'URL suivante

```
http://grappa.univ-lille3.fr/~gonzalez/prog/form01.php3?x=5&y=7
```

autorise l'emploi des variables « \$x » (qui vaut 5) et « \$y » (qui vaut 7) dans le programme « form01.php3 ».

Attention !

Il est de plus en plus courant de rencontrer des serveurs configurés en *safe mode*, c'est-à-dire avec des réglages destinés à se protéger du détournement de certaines pages par d'éventuels pirates.

Si c'est votre cas vous aurez accès au paramètre `truc` non pas par `$truc` mais par :

- `$_GET["truc"]` `$HTTP_GET_VARS["truc"]` pour un envoi par la méthode GET :
- `$_POST["truc"]` ou (au choix) par `$HTTP_POST_VARS["truc"]` pour un envoi par la méthode POST.

Dans la suite nous supposons ne pas être en *safe mode*.

La question est donc : *quelles variables (avec quelles valeurs) sont transmises par un formulaire ?* Les noms des variables sont ceux des champs du formulaire.

Pour les *zones de saisies*, le texte tapé est transmis tel quel. Pour les *cases à cocher*, la valeur est « on » si la case a été cochée, une chaîne vide sinon. Pour les *boutons radios* c'est la valeur affectée à « value » pour le bouton sélectionné. Pour les *listes déroulantes*, c'est la valeur affectée à « option » pour le choix sélectionné.

7.2. Exercices

1. Écrire un formulaire qui demande le nom et l'âge de l'utilisateur. Le bouton *submit* de ce formulaire provoquera l'affichage d'une page qui saluera l'utilisateur avec cette phrase : « Bonjour *machin*, vous avez *xx* ans... » (avec les bonnes valeurs, bien entendu).

Exemple : traitement-01.html¹

2. Deux vacanciers ont abandonné à Montpellier leur bébé de 9 mois, qui n'avait pas été sage.

Quelle ne fut pas leur surprise quand 6 mois plus tard, rentrés chez eux à Lille, ils ont vu arriver leur enfant qui avait fait à quatre pattes le trajet Montpellier-Lille par l'autoroute.

Écrire un formulaire PHP permet de saisir la distance parcourue par le bébé, le nombre d'heures où il marchait par jour, et le nombre de jours qu'il a passés sur la route. Le formulaire affichera alors la vitesse du bébé.

Exemple : traitement-02.html²

3. Écrire un formulaire qui demande le nom et le sexe de l'utilisateur (M ou Mme). Ce formulaire appelle une page qui affichera « Bonjour monsieur *Truc* » ou « Bonjour madame *Bidule* » suivant le cas (avec le vrai nom de la personne, bien entendu !) :

Exemple : traitement-03.html³

4. Un permis de chasse à points remplace désormais le permis de chasse traditionnel.

1. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-01.html>

2. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-02.html>

3. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-03.html>

Chaque chasseur possède au départ un capital de 100 points. S'il tue une poule il perd 1 point, 3 points pour un chien, 5 points pour une vache et 10 points s'il tue son meilleur ami.

Le permis coûte 1000 francs.

Écrire un formulaire PHP qui permet de saisir la liste des victimes du chasseur et calcule le prix à payer pour les permis supplémentaires nécessaires.

Exemple : `traitement-04.html`⁴

5. Écrire un formulaire « calculatrice » : 2 cases pour la saisie des opérandes, un groupe de 4 cases à cocher (ou une liste déroulante) pour le choix de l'opération, et affichage du résultat de l'opération.

Exemple : `traitement-05.php`⁵

6. Écrire un formulaire qui demande deux nombres *a* et *b*. Il affiche ensuite la table de multiplication par *a* sur *b* lignes. Le formulaire doit se souvenir des valeurs choisies.

Exemple : `traitement-06.php`⁶

7. Écrire un formulaire qui demande un mot de passe. Deux mots de passe différents sont acceptés. On saluera l'utilisateur différemment en fonction du mot de passe utilisé.

Exemple : `traitement-07.php`⁷

8. Addition à compléter : un formulaire propose une addition incomplète que l'utilisateur doit terminer.

Exemple : `addition.php`⁸

9. Devinette n°1 : vous devez deviner le nombre que le programme a choisi. Vous proposez une solution, et le programme vous répond « trop petit » ou « trop grand » suivant le cas, jusqu'à trouver le bon nombre.

Exemple : `devinette01.php`⁹

10. Devinette n°2 : même chose que l'exercice précédent, mais les rôles sont inversés (c'est le programme qui devine).

Exemple : `devinette02.php`¹⁰

11. Le *pipotron* : l'ordinateur vous aide à écrire n'importe quoi.

Exemple : `pipotron.php`¹¹

4. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-04.html>

5. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-05.php>

6. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-06.php>

7. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-07.php>

8. <http://grappa.univ-lille3.fr/polys/php/exemples/addition.php>

9. <http://grappa.univ-lille3.fr/polys/php/exemples/devinette01.php>

10. <http://grappa.univ-lille3.fr/polys/php/exemples/devinette02.php>

11. <http://grappa.univ-lille3.fr/polys/php/exemples/pipotron.php>

Chapitre 8. Chaînes de caractères

Les corrigés des exercices de ce chapitre se trouvent Chapitre 22.

La création de pages web nécessite la plupart du temps la manipulation de chaînes de caractères.

Nous allons en explorer les principales fonctions disponibles en PHP.

8.1. Qu'est-ce qu'une chaîne de caractères ?

Trois choses à savoir :

1. Le mot anglais qui désigne les chaînes de caractères est « `string` » (utile, car la documentation est souvent en anglais).
2. Ayez toujours à portée de main la documentation de PHP. Et, d'après le point précédent, allez voir le chapitre *String*¹.
3. Une chaîne, c'est n'importe quoi entouré de guillemets (« " ») ou d'apostrophes (« ' »). Voici deux exemples :

```
"Je suis une chaîne de caractères."  
'Je suis aussi une chaîne de caractères.'
```

8.2. Que choisir ? Guillemets ou apostrophes ?

C'est pareil, sauf que :

- Si on commence par l'un, on finit par le même. Voici deux exemples qui *NE* sont *PAS* des chaînes de caractères :

```
"Je ne suis pas une chaîne de caractères.'  
'Je ne suis pas non plus une chaîne de caractères."
```

- Si votre chaîne contient des apostrophes, utilisez des guillemets, si votre chaîne contient des guillemets, utilisez des apostrophes. Par exemple :

```
"J'en rêvais..."  
'Il a dit "Bonjour".'
```

Remarque

Si votre chaîne doit contenir les deux, vous pouvez utiliser le caractère *backslash* (« \ ») :

```
'J\'en rêvais... Il m\'a dit "Bonjour".'  
"J'en rêvais... Il m'a dit \"Bonjour\"."
```

- Les variables sont interprétées dans les guillemets, mais pas dans les apostrophes.

Par exemple si la variable `$a` vaut 5, l'instruction « `echo "$a";` » affichera « 5 », tandis que « `echo '$a';` » affichera « `$a` ».

Il en est de même des caractères précédés du caractère *backslash* (sauf « \' »).

8.3. Principales utilisations du *backslash*

Le caractère *backslash* (« \ ») permet d'introduire dans les chaînes des caractères qui ne pourraient pas y être sinon (par exemple des guillemets dans une chaîne entourée de guillemets, des retours à la ligne, etc...). Comme il est de ce fait lui-même un caractère spécial, il faut une façon particulière de le noter pour qu'il soit inclus dans une chaîne. Ce sera « \\ ».

Voici un tableau récapitulant les principales utilisations du *backslash* :

1. <http://fr.php.net/manual/fr/ref.strings.php>

Tableau 8-1. Principales utilisations du *backslash*

code	signification
<code>\n</code>	retour à la ligne
<code>\t</code>	tabulation
<code>\\</code>	<i>backslash</i> lui-même (« <code>\</code> »)
<code>\"</code>	guillemet (« <code>"</code> ») dans une chaîne entourée de guillemets
<code>\'</code>	apostrophe (« <code>'</code> ») dans une chaîne entourée d'apostrophes
<code>\\$</code>	dollar (« <code>\$</code> »)

8.4. Principales opérations sur les chaînes

Affectation :

Comme toute variable, l'affectation d'une variable chaîne se fait par le symbole « = » :

```
$a="bla bla bla bla...";
```

Affichage :

Comme pour toute valeur, l'affichage se fait par l'instruction « `echo` ».

Concaténation :

L'opérateur de concaténation est le point (« `.` »).

8.5. Principales fonctions

Il ne s'agit là que des fonctions *principales*, et elles ne seront pas expliquées, ce qui signifie qu'il vous *faudra* aller voir dans la documentation de PHP² par vos propres moyens.

Conversion majuscules-minuscules :

```
« strtolower3 ».
```

Conversion minuscules-majuscules :

```
« strtoupper4 ».
```

Découper une chaîne :

```
« explode5 » ou « split6 » (renvoient un tableau). Les fonctions inverses sont « implode7 » et « join8 »
```

Enlever les espaces en début ou fin d'une chaîne :

```
« ltrim9 », « chop10 », « trim11 ».
```

Longueur d'une chaîne :

```
« strlen12 »
```

-
2. <http://fr.php.net/manual/fr/ref.strings.php>
 3. <http://fr.php.net/manual/fr/function strtolower.php>
 4. <http://fr.php.net/manual/fr/function strtoupper.php>
 5. <http://fr.php.net/manual/fr/function explode.php>
 6. <http://fr.php.net/manual/fr/function split.php>
 7. <http://fr.php.net/manual/fr/function implode.php>
 8. <http://fr.php.net/manual/fr/function join.php>
 9. <http://fr.php.net/manual/fr/function ltrim.php>
 10. <http://fr.php.net/manual/fr/function chop.php>
 11. <http://fr.php.net/manual/fr/function trim.php>
 12. <http://fr.php.net/manual/fr/function strlen.php>

Position d'une chaîne dans une autre :

« strpos¹³ »

Remplacer une partie d'une chaîne par une autre :

« str_replace¹⁴ » ou « substr_replace¹⁵ »

Un extrait d'une chaîne :

« substr¹⁶ »

8.6. Conversions de types

PHP est extrêmement tolérant sur les conversions de types et essaiera toujours de faire le mieux et le plus naturellement possible.

N'en profitez quand même pas trop et prévoyez (et comprenez) toujours son comportement, cela vous évitera sans doute des déconvenues...

Un exemple de ce qu'on peut faire (mais évitez d'en abuser...) :

```
<?php
    $truc = "5";
    echo $truc." ".gettype($truc)."<br />\n";
    $truc = $truc."2";
    echo $truc." ".gettype($truc)."<br />\n";
    $truc = $truc*5;
    echo $truc." ".gettype($truc)."<br />\n";
    $truc = $truc."2";
    echo $truc." ".gettype($truc)."<br />\n";
    $truc = $truc/100;
    echo $truc." ".gettype($truc)."<br />\n";
?>
```

Ce programme affichera :

```
5 string
52 string
260 integer
2602 string
260.2 double
```

Remarque

La commande « echo \$truc." ".gettype(\$truc)."
\n"; » provoque l'affichage de la valeur de \$truc, suivie de son type, tout ceci étant terminé par un passage à la ligne.

8.7. Exercices

Écrire un formulaire qui devra vérifier l'identité de l'utilisateur en demandant un nom, un prénom et un mot de passe (ce dernier ne devra pas apparaître à l'écran pendant la frappe).

Tant que l'utilisateur n'est pas reconnu, seul s'affiche le formulaire.

Si l'utilisateur est reconnu, seul s'affiche un message d'accueil.

La casse des valeurs ne devra être prise en compte (c'est-à-dire le fait que ce soit tapé en majuscules ou en minuscules ne devra pas changer le comportement des pages).

Les espaces tapés en début ou en fin de mot de passe, de nom ou de prénom devront être éliminés.

13. <http://fr.php.net/manual/fr/function.strpos.php>

14. <http://fr.php.net/manual/fr/function.str-replace.php>

15. <http://fr.php.net/manual/fr/function.substr-replace.php>

16. <http://fr.php.net/manual/fr/function.substr.php>

Vous trouverez un exemple sur la page `identite.php`¹⁷

17. <http://grappa.univ-lille3.fr/polys/php/exemples/identite.php>

Chapitre 9. Les tableaux

Les corrigés des exercices de ce chapitre se trouvent Chapitre 23.

Le stockage et la manipulation d'un grand nombre de données nécessitent souvent l'utilisation de tableaux.

9.1. Qu'est-ce qu'un tableau ?

Pour simplifier, voici 3 définitions *équivalentes* :

- c'est une variable qui peut stocker plusieurs valeurs à la fois, chacune étant numérotée ;
- OU : c'est un ensemble de variables différentes, regroupées sous le même nom, chacune étant différenciée des autres par un numéro ;
- OU : c'est la représentation informatique d'un vecteur (ou d'une matrice), chacune des composantes étant connue par son indice.

D'une manière plus générale :

- Un tableau est d'abord une variable, donc son nom commence par un *dollar* (« \$ »).
- La syntaxe pour accéder aux cases d'un tableau est la même que dans la plupart des langages : « `$x[5]` » désigne le 5^{ème} élément du tableau « `$x` ».

Remarque

En PHP la numérotation des éléments d'un tableau commence *toujours* à zéro.

9.2. Quelques exemples d'utilisation

9.2.1. Remplir et afficher le contenu un tableau

On peut remplir, par exemple, les cases d'un tableau une par une :

```
$t[0]="bonjour";  
$t[1]="bonsoir";  
$t[2]="bla bla bla";
```

On peut aussi utiliser une boucle `for` pour remplir une série de cases :

```
for ($i=3 ; $i<8 ; $i++) {  
    $t[$i]=$i*5;  
}
```

Pour ce qui de l'affichage, on est obligé de passer aussi par les cases une par une : l'utilisation de « `echo $t;` » ne permet pas d'afficher le contenu du tableau.

```
echo "case numéro 2 : ".$t[2]."<br />\n";  
for ($i=2 ; $i<6 ; $i++) {  
    echo "case numéro $i : ".$t[$i]."<br />\n";  
}
```

9.2.2. Un exemple complet

En regroupant les lignes précédentes (et en les habillant un peu) on obtient le programme suivant (tableau-01.php¹) :

```
<html><body>
<h2>Remplissage du tableau</h2>

(Il ne se passe rien à l'écran dans cette partie : normal,
c'est du travail sur la modification des variables, par sur
leur affichage.)

<?php
    $t[0]="bonjour";
    $t[1]="bonsoir";
    $t[2]="bla bla bla";
    for ($i=3 ; $i<8 ; $i++) {
        $t[$i]=$i*5;
    }
?>

<h2>Affichage de certaines cases particulières</h2>

<?php
    echo "case numéro 2 : ".$t[2]."<br />\n"
        ."case numéro 5 : ".$t[5]."<br />\n";
?>

<h2>Affichage d'une suite de cases</h2>

<?php
    for ($i=2 ; $i<6 ; $i++) {
        echo "case numéro $i : ".$t[$i]."<br />\n";
    }
?>
</body></html>
```

9.3. Les tableaux associatifs

Il s'agit d'une généralisation des tableaux, mais au lieu d'utiliser des nombres pour les indices on peut utiliser n'importe quoi...

Ce programme (tableau-04.php²) permet de choisir un prénom dans une liste déroulante et d'afficher les renseignements correspondants.

```
<html><body>
<form action="tableau-04.php">
    <select name="prenom">
        <option value="Thècle">Thècle</option>
        <option value="Édith">Édith</option>
        <option value="Kelly">Kelly</option>
        <option value="Mélusine">Mélusine</option>
        <option value="Elmer">Elmer</option>
    </select>
    <input type="submit" value="renseignements" />
</form>
<?php
    $nom["Thècle"]="Assicmonpote";
    $nom["Édith"]="Avuleur";
    $nom["Kelly"]="Diocy";
    $nom["Mélusine"]="Enfaillite";
    $nom["Elmer"]="Hitmieux";
    $adresse["Thècle"]="123 rue Alex KUZBIDON";
    $adresse["Édith"]="951 rue Ondine OUCESOIR";
    $adresse["Kelly"]="5 boulevard Rosa REMECITOYENS";
```

1. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-01.php>

2. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-04.php>

```

$adresse["Mélusine"]="7 rue Jean TANLAMER";
$adresse["Elmer"]="99 avenu Dino ZORE";
$ville["Thècle"]="Rennes";
$ville["Édith"]="Douarnenez";
$ville["Kelly"]="Auchel";
$ville["Mélusine"]="Chamonix";
$ville["Elmer"]="Langres";

if (isset($prenom)) {
    echo "Prénom : $prenom<br />\nNom : ".$nom[$prenom]
        . "<br />\nAdresse : ".$adresse[$prenom]
        . "<br />\nVille : ".$ville[$prenom];
}
?>
</body></html>

```

9.4. Parcours d'un tableau associatif

Quand on travaille avec un tableau classique (à indice numérique), il est facile d'en parcourir toutes les cases avec une boucle `for`.

Mais pour parcourir un tableau associatif, c'est un peu moins direct. La méthode classique ne marche pas : la fonction `count`³ ne donne pas un résultat exploitable facilement dans le cas d'un tableau associatif. Il est alors difficile d'imaginer, par exemple pour l'exemple précédent, une boucle qui construirait automatiquement la suite « *Thècle, Édith, Kelly, Mélusine, Elmer* ».

Ces valeurs, il faut aller les chercher dans le tableau lui-même.

Ne nous énervons pas, PHP fournit tout ce qu'il faut. La structure `foreach`⁴ permet de parcourir un tableau associatif, comme dans le programme suivant (`tableau-05bis.php`⁵).

Le programme `tableau-05bis.php`⁶ parcourt le tableau `$nom`, et en affiche le contenu, chaque ligne d'affichage contenant le couple clef-valeur (la clef correspondant à ce qu'on a l'habitude d'appeler indice quand il s'agit d'un nombre) :

```

<html><body><table border="border">
<?php
    $nom["Thècle"]="Assicmonpote";
    $nom["Édith"]="Avaleur";
    $nom["Kelly"]="Diocy";
    $nom["Mélusine"]="Enfaillite";
    $nom["Elmer"]="Hitmieux";
    foreach ($nom as $clef => $valeur) {
        echo "<tr><td>$clef</td><td>$valeur</td></tr>\n";
    }
?>
</table></body></html>

```

9.5. Exercices

1. On dispose du fichier suivant contenant des adresses web (rien en vous empêche d'en choisir d'autres).

Écrire un programme (`tableau-06.php`⁷) qui lit ce fichier pour construire une page web contenant une liste de liens hypertextes (`tableau-06.txt`⁸).

```
http://www.hoaxbuster.com/
```

3. <http://fr.php.net/manual/fr/function.count.php>

4. <http://fr.php.net/foreach>

5. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-05bis.php>

6. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-05bis.php>

7. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-06.php>

8. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-06.txt>

```
http://www.gazel.nu/faqs/virus.htm#e-mail
http://www.electriccafe.org/JBT/
http://www.zetetique.ldh.org/
http://perso.wanadoo.fr/jean.brissonnet/
http://www.thejackytouch.com/
http://www.multimania.com/lepoulpe
```

2. Même exercice (tableau-07.php⁹), mais cette fois chaque ligne comprend aussi une description du site pointé, la séparation étant assuré par la chaîne « "*" » (tableau-07.txt¹⁰) :

```
http://www.hoaxbuster.com/**HoaxBuster, première....
http://www.gazel.nu/faqs/virus.htm#e-mail**Est-il....
http://www.electriccafe.org/JBT/**Nemo Joe Bar Team Spirit
http://www.zetetique.ldh.org/**Le cercle Zététique
http://perso.wanadoo.fr/jean.brissonnet/**Éthique et toc
http://www.thejackytouch.com/**The Jacky Touch
http://www.multimania.com/lepoulpe**Le poulpe sur la toile
```

3. Même exercice (tableau-08.php¹¹), mais cette fois chaque description et l'adresse correspondante sont sur deux lignes consécutives (tableau-08.txt¹²) :

```
HoaxBuster, première ressource francophone sur les hoax
http://www.hoaxbuster.com/
Est-il possible d'attraper un virus en ouvrant un e-mail ?
http://www.gazel.nu/faqs/virus.htm#e-mail
Nemo Joe Bar Team Spirit
http://www.electriccafe.org/JBT/
Le cercle Zététique
http://www.zetetique.ldh.org/
Éthique et toc
http://perso.wanadoo.fr/jean.brissonnet/
The Jacky Touch
http://www.thejackytouch.com/
Le poulpe sur la toile
http://www.multimania.com/lepoulpe
```

4. On donne une liste de personnes (tableau-09.txt¹³) dont voici les premières lignes :

```
19;Thor;Aipaleur;tata
30;Dick;Sionnaire;dsds 35;Debbie;Zoudanlkou;dzd
47;Mélanie;Zaitofrai;mzmz 48;Helmut;Ardelpic;haha
49;Jacques-André;Lejouré-Lanuit;jljl 68;Phil;Alapate;papa
```

Chaque ligne est composée, dans l'ordre, d'un identifiant (un nombre), un prénom, un nom, et un mot de passe. Écrire une page web (tableau-09.php¹⁴) qui donne à sélectionner une des personnes et qui affiche ensuite son mot de passe.

9. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-07.php>

10. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-07.txt>

11. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-08.php>

12. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-08.txt>

13. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-09.txt>

14. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-09.php>

Chapitre 10. PDO

10.1. Qu'est-ce que PDO

Ce texte est extrait de « XXVII. Fonctions PDO¹ », un chapitre du manuel PHP sur sur PHP net².

L'extension PHP Data Objects (PDO) définit une excellente interface pour accéder à une base de données depuis PHP. Chaque pilote de base de données implémenté dans l'interface PDO peut utiliser des fonctionnalités spécifiques de chacune des bases de données en utilisant des extensions de fonctions. Notez que vous ne pouvez exécuter aucune fonction de base de données en utilisant l'extension PDO par elle-même ; vous devez utiliser un driver PDO spécifique à la base de données³ pour accéder au serveur de base de données.

PDO fournit une interface d'abstraction à l'accès de données, ce qui signifie que vous utilisez les mêmes fonctions pour exécuter des requêtes ou récupérer les données quelque soit la base de données utilisée. PDO ne fournit pas une abstraction de base de données : il ne réécrit pas le SQL, n'émule pas des fonctionnalités manquantes. Vous devriez utiliser une interface d'abstraction complète si vous avez besoin de cela.

PDO est fourni avec PHP 5.1 et est disponible en tant qu'extension PECL pour PHP 5.0 ; PDO requiert les nouvelles fonctionnalités OO fournies par PHP 5 et donc, ne fonctionne pas avec les versions antérieures de PHP.

10.2. Quelle utilité ?

L'hétérogénéité des moyens d'accès aux bases de données a été un des problèmes les plus gênants en PHP. Le problème vient du fait que, par exemple, les fonctions permettant l'accès à une base PostgreSQL sont différentes de celles pour MySQL, et elles sont toutes différentes de celles pour SQLite, etc.

Cela a pour effet de rendre plus difficile la migration d'un système vers un autre : vous avez écrit (par exemple en cours) un site utilisant une base PostgreSQL et vous voulez le transférer chez un hébergeur qui n'offre que MySQL (comme c'était par exemple le cas pour Free il y a encore peu de temps) : votre seule solution est de reprendre (et corriger) dans toutes vos pages tous les appels à des fonctions concernant PostgreSQL. Il peut y en avoir beaucoup...

Une couche d'abstraction⁴ permet d'éviter ce problème : tous vos accès à votre base de données passent par elle (avec la même syntaxe, quelle que soit la base de données) et c'est elle qui se débrouille pour s'adapter à la base de données utilisée. Pour changer de base de données (passer de PostgreSQL à MySQL par exemple) vous n'avez en général qu'à modifier une ligne, celle qui donne les paramètres de la base de données utilisée.

10.3. Se connecter à la base de données

Ce texte est extrait de « Fonctions PDO⁵ », un chapitre du manuel PHP sur sur PHP net⁶.

Les connexions sont établies en créant des instances de la classe de base de PDO. Peu importe quel driver vous voulez utiliser ; vous utilisez toujours le nom de la classe PDO. Le constructeur accepte des paramètres pour spécifier la source de la base de données (connue en tant que DSN) et optionnellement, le nom d'utilisateur et le mot de passe (s'il y en a un).

Il suffit pour cela d'exécuter la commande

```
$dbh = new PDO("type_de_base:host=machine_qui_héberge_la_base;dbname=nom_de_la_base",
               "nom_d_utilisateur",
               "mot_de_passe" );
```

Le nom de la variable \$dbh est sans importance. Ce n'est qu'une variable, vous pouvez lui donner le nom qui vous plaît. Il faut seulement garder à l'esprit qu'il faut conserver cette variable (quelque soit son nom), c'est elle qui permettra d'accéder à la base de données avec les fonctions des sections suivants.

1. <http://fr.php.net/manual/fr/ref.pdo.php>
2. <http://fr.php.net/>
3. <http://fr.php.net/manual/fr/ref.pdo.php#pdo.drivers>
4. Il existe d'autres couches d'abstraction : dbx, PHPLib, ADODB, MetaData, Pear-DB, etc.
5. <http://fr.php.net/manual/fr/ref.pdo.php>
6. <http://fr.php.net/>

Le `type_de_base` quant à lui est une valeur parmi :

- `mssql` (FreeTDS/Microsoft SQL Server/Sybase),
- `firebird` (Firebird/Interbase 6),
- `informix` (IBM Informix Dynamic Server),
- `mysql` (MySQL 3.x/4.x/5.x),
- `oci` (Oracle Call Interface),
- `odbc` (ODBC v3 --IBM DB2 unixODBC et win32 ODBC--),
- `pgsql` (PostgreSQL),
- `sqlite` (SQLite 3 et SQLite 2).

On se reportera à la page « Fonctions PDO⁷ » pour la gestion des erreurs de connexion.

Lorsque la connexion à la base de données a réussi, une instance de la classe PDO est retournée à votre script. La connexion est active tant que l'objet PDO l'est. Pour clore la connexion, vous devez détruire l'objet en vous assurant que toutes ses références sont effacées. Vous pouvez faire cela en assignant `NULL` à la variable gérant l'objet. Si vous ne le faites pas explicitement, PHP fermera automatiquement la connexion lorsque le script arrivera à la fin.

10.4. Exécuter une requête

Vous trouverez plus d'information sur la page `PDO->query()`⁸ du manuel PHP.

En ayant auparavant obtenu un identifiant de connexion `$dbh` (instance de la classe de base de PDO) comme précédemment, la syntaxe est simple :

```
$res = $dbh->query(votre_requête) ;
```

La valeur renvoyée (rangée ici dans la variable `$res`) N'EST PAS le résultat de l'exécution de la requête, mais un identifiant (propre au système, sa valeur réelle ne nous intéresse pas⁹) qui nous permettra d'accéder aux résultats grâce aux fonctions décrites ci-dessous.

10.5. Nombre de lignes et colonnes d'une requête `select`

On peut facilement obtenir le nombre de lignes et de colonnes du résultat d'une requête de type `SELECT` :

```
$result = $dbh->query('SELECT id FROM table') ;  
echo $result->rowCount() ; // nombre de lignes  
echo $result->columnCount() ; // nombre de champs
```

Avertissement

Avec certains gestionnaires de bases de données (en particulier `Sqlite`) il arrive que la méthode `rowCount()` ne fonctionne pas. Dans ce cas-là, remplacez `$res->rowCount()` par `count($data)` (où `$data` est égal à `count($result->fetchAll())`, voir Section 10.6), cela produira le même effet.

Plus d'informations sur PHP net pour `rowCount()`¹⁰ et `columnCount()`¹¹.

7. <http://fr.php.net/manual/fr/ref.pdo.php>

8. <http://fr.php.net/manual/fr/function.pdo-query.php>

9. En fait c'est un objet.

10. <http://fr.php.net/manual/fr/function.pdostatement-rowcount.php>

11. <http://fr.php.net/manual/fr/function.pdostatement-columncount.php>

10.6. Accéder aux résultats d'une requête `select`

`$result` étant un résultat valide renvoyé par `$dbh->query()`, `$result->fetchAll()` renvoie un tableau à 2 dimensions qui contient les résultats : le premier indice concerne les lignes, le deuxième concerne les colonnes.

Plus d'informations sur PHP net pour `fetchAll()`¹².

10.6.1. Accès par le nom des colonnes

Exemple :

```
$result = $dbh->query("SELECT * FROM villes");
$data = $result->fetchAll();
foreach ( $data as $row ) {
    echo $row["codepostal"]." - ".$row["ville"]."<br />\n";
}
```

10.6.2. Accès par le numéro des colonnes

Exemple :

```
$result = $dbh->query("SELECT * FROM villes");
$data = $result->fetchAll();
foreach ( $data as $row ) {
    for ($i=0 ; $i<$result->columnCount() ; $i++) {
        echo $row[$i]." ** ";
    }
    echo "<br />\n";
}
```

10.6.3. Accès par le numéro des lignes et le numéro des colonnes

Exemple :

```
$result = $dbh->query("SELECT * FROM villes");
$data = $result->fetchAll();
for ($l=0;$l<$result->rowCount();$l++) {
    for ($i=0;$i<$result->columnCount();$i++) {
        echo $data[$l][$i]." ** ";
    }
    echo "<br />\n";
}
```

10.7. Gestion des erreurs

Quand une instruction PDO échoue, aucun message n'est affiché. On peut alors croire à tort que tout s'est bien passé. Il est donc important de savoir obtenir des informations sur d'éventuelles erreurs. Pour avoir des informations plus précises que ces quelques lignes, allez voir les pages dont elles sont extraites¹³ sur PHP.net¹⁴.

Pour la suite nous continuerons à nous placer dans le cas où on a auparavant obtenu un identifiant de connexion `$dbh` (instance de la classe de base de PDO).

12. <http://fr.php.net/manual/fr/function.pdostatement-fetchall.php>

13. <http://fr.php.net/manual/fr/pdo.error-handling.php>

14. <http://fr.php.net/>

10.7.1. Code d'erreur

La méthode `$db->errorCode()` renvoie le code d'erreur associé avec la dernière opération effectuée sur la base de données. Sa valeur est 0 (zéro) si il n'y a pas eu d'erreur.

10.7.2. Informations associées à l'erreur

La méthode `$db->errorInfo()` renvoie les informations associées à l'erreur survenue lors de la dernière opération sur la base de données. Il s'agit d'un tableau qui contient les champs décrit Tableau 10-1.

Tableau 10-1. Champs de `$db->errorInfo()`

Élément	Information
0	Code erreur SQLSTATE (un identifiant alphanumérique de cinq caractères défini dans le standard ANSI SQL).
1	Code erreur spécifique au driver.
2	Message d'erreur spécifique au driver.

Si le code erreur SQLSTATE n'est pas défini ou s'il n'y a pas d'erreur spécifique du driver, l'élément suivant l'élément 0 sera défini à NULL.

10.7.3. Exemple de traitement d'erreur

Pour gérer les éventuelles erreurs, les méthodes précédentes peuvent être utilisée de cette façon (ici on exécute une requête `$req` définie par ailleurs) :

```
$db->query($req); // exécution de la requête
if ($db->errorCode() != 0) { // il y a une erreur
    echo "<b>Erreur</b> sur la requête <tt>$req</tt><br />\n";
    $t=$db->errorInfo(); // récupération des informations sur l'erreur
    echo "<b>Code erreur SQLSTATE :</b> ".$t[0]."<br />\n";
    echo "<b>Code erreur spécifique au driver :</b> ".$t[1]."<br />\n";
    echo "<b>Message d'erreur :</b><blockquote><p>".$t[2]."</p></blockquote>\n";
    echo "<b>Le programme a été interrompu</b>";
    die();
}
```

Chapitre 11. Programmation séparée en PHP

11.1. La programmation séparée

La programmation séparée c'est écrire un programme en plusieurs petits fichiers au lieu d'un seul gros.

Quelques avantages :

- Des fichiers plus petits sont plus faciles à relire, comprendre, modifier, tester et corriger.
- Cela permet de faire travailler plusieurs personnes sur le même programme en même temps.
- Cela permet de regrouper plus facilement les fonctions qui sont dans le même domaine (chaque fichier a sa propre cohérence).
- Il est plus facile de réutiliser le travail déjà fait.

11.2. Les fonctions `require` et `include`

Les fonctions « `require`¹ » et « `include`² » permettent d'inclure un fichier dans un autre.

Il est dit par exemple :

la commande `require` se remplace elle-même par le contenu du fichier spécifié.

Différentes façons d'utiliser ces fonctions :

1. Pour rendre homogène la présentation d'un site, on écrit une seule fois le début (avec la partie `head`, la couleur de fond, la couleur de texte, la présentation du grand titre, etc.) et la fin de page (avec par exemple une signature ou une adresse mail) dans des fichiers séparés et on se contente d'ajouter dans chaque feuille un `require` au début et un à la fin sur ces fichiers.
2. Si une partie de code est identique dans plusieurs pages, on l'écrit une seule fois dans un fichier séparé et on l'inclut par `require` à chaque fois qu'elle est nécessaire.

Ces façons de faire n'ont pas seulement l'avantage de vous permettre de ne pas taper plusieurs fois la même chose pour vous éviter une fatigue inutile (surtout que le *copier-coller* marche bien).

Leur utilité est surtout de rendre les modifications faciles à répercuter : si vous décidez de changer la présentation globale de votre site, il n'y a qu'un seul fichier à modifier ; si vous décidez de modifier la partie de code commune (une façon de faire un calcul, la façon de se connecter, etc.), il n'y a également qu'un seul fichier à modifier.

1. <http://fr.php.net/manual/fr/function.require.php>

2. <http://fr.php.net/manual/fr/function.include.php>

Chapitre 12. Identification en PHP

Nous n'aborderons ici que l'identification gérée par le navigateur.

12.1. Préliminaires : la fonction `header`

La fonction `header`¹ permet de spécifier une en-tête HTTP lors de l'envoi des fichiers HTML. Reportez-vous à *HTTP 1.1 Specification*² pour plus d'informations sur les en-têtes HTTP.

Remarque

La fonction `header` doit être appelée avant la première balise HTML, et avant n'importe quel envoi de commande PHP.

Nous allons utiliser ici cette fonction pour demander la vérification d'une identité.

Pour cela il suffit de faire commencer votre programme PHP par la commande :

```
header("WWW-Authenticate: Basic realm='private'");
```

qui a pour effet de faire demander (et retenir) un nom et un mot de passe par le navigateur.

Vous pouvez ainsi tester le programme élémentaire suivant (`ident-exple01.php`)³ :

```
<?php
if (!isset($PHP_AUTH_USER) || !isset($PHP_AUTH_PW)) {
    header("WWW-Authenticate: Basic realm='private'");
} else {
    echo "<html><body>";
    echo "les données connues sont $PHP_AUTH_USER et $PHP_AUTH_PW\n";
    echo "</body></html>";
}
?>
```

Vous pouvez remarquer que la demande n'est faite que lors de la première exécution de la page. La seule façon, *avec le programme tel qu'il est écrit*, de devoir répondre à nouveau à la question est de sortir de votre navigateur (fermer toutes les fenêtres du navigateur, oui !) et de le relancer.

Attention !

Si votre serveur est configuré en *safe mode* vous n'aurez pas accès aux variables `$PHP_AUTH_USER` et `$PHP_AUTH_PW` mais à `$_SERVER["PHP_AUTH_USER"]` et à `$_SERVER["PHP_AUTH_PW"]`

Si votre serveur est configuré en *safe mode*, voir aussi Section 7.1.

C'est cette propriété que nous allons utiliser pour gérer l'identification des utilisateurs.

12.2. Contrôle des mots de passe

Il ne manque qu'une seule chose (ou presque !) au programme précédent pour être parfait : tester les valeurs des variables `$PHP_AUTH_USER` (qui contient le nom) et `$PHP_AUTH_PW` (qui contient le mot de passe) pour vérifier qu'on a affaire à la bonne personne.

Une version simple pourrait ressembler à ça (`ident-exple02.php`)⁴ :

```
<?php
if (($PHP_AUTH_USER=="marcel") && ($PHP_AUTH_PW=="bidule")) {
    echo "<html><body>";
    echo "Bravo, identification réussie.\n";
    echo "</body></html>";
} else {
```

1. <http://fr.php.net/manual/fr/function.header.php>

2. <http://www.w3.org/Protocols/rfc2616/rfc2616>

3. <http://grappa.univ-lille3.fr/polys/php/exemples/ident-exple01.php>

4. <http://grappa.univ-lille3.fr/polys/php/exemples/ident-exple02.php>

```
header("WWW-Authenticate: Basic realm='private'");
}
?>
```

12.3. Amélioration : plusieurs utilisateurs

Il est rare, quand l'accès à une page est contrôlée par un mot de passe, que tous les utilisateurs aient les mêmes droits.

Il faut donc pouvoir détecter l'identité de la personne concernée au moment de la saisie du mot de passe. Pour cela on donnera un mot de passe différent à chaque utilisateur. Cela nous permet d'avoir l'exemple suivant ([ident-exple03.php](http://grappa.univ-lille3.fr/polys/php/exemples/ident-exple03.php)⁵) dans lequel les renseignements sont rangés dans un tableau de chaînes de caractères, une chaîne par personne, présentée sous la forme « *vrai nom/nom d'utilisateur/mot de passe* » :

```
<?php
// la liste des informations
$liste=array("Jean Némard/nemard/jjjj",
             "Sophie Fonfec/fonfec/ssss",
             "Yves Adrouille-Toultan/adrouille/yyyy");
// création des tableaux
for ($i=0;$i<count($liste);$i++) {
    $l=explode("/",trim($liste[$i]));
    $nom[$i]=$l[0]; // identité réelle de l'utilisateur n°i
    $user[$i]=$l[1]; // identifiant de l'utilisateur n°i sur le système
    $pass[$i]=$l[2]; // mot de passe de l'utilisateur n°i
}
$nbusers=count($liste);
// contrôle de l'identité
$ok=-1; // on démarre sans connaître l'utilisateur
for ($i=0;$i<$nbusers;$i++) {
    if (($PHP_AUTH_USER==$user[$i]) && ($PHP_AUTH_PW==$pass[$i])) {
        // on a reconnu un utilisateur -> on garde son numéro
        $ok=$i;
    }
}
// si l'identification a raté, $ok contient toujours -1
// on demande le nom et le mot de passe
if ($ok==-1) {
    header("WWW-Authenticate: Basic realm='private'");
    return; // On notera l'utilisation de l'instruction return
            // pour sortir du programme.
}
// si on arrive ici, c'est que l'identification a réussi
// et que $ok contient le numéro de l'utilisateur
echo "<html><body>";
echo "Bravo, identification réussie.<br />\n";
echo "vous êtes <em>nom[$ok]</em>,\n";
echo "votre identifiant est <em>$user[$ok]</em>,\n";
echo "votre mot de passe est <em>$pass[$ok]</em>.<br />\n";
// À vous d'ajouter ici tout ce qui concerne l'utilisateur
// ...
// ...
echo "</body></html>";
?>
```

5. <http://grappa.univ-lille3.fr/polys/php/exemples/ident-exple03.php>

12.4. Séparer les données du traitement

Il est tout à fait raisonnable d'imaginer qu'on va avoir besoin de changer les mots de passe, ainsi qu'ajouter des nouveaux utilisateurs ou en supprimer des anciens.

Il n'est par contre pas raisonnable du tout d'imaginer que celui qui va le faire aura envie d'aller manipuler à chaque fois le fichier du programme. (Ce n'est même d'ailleurs pas souhaitable...)

Il est donc préférable de garder ces informations dans un fichier séparé.

Qu'est-ce que cela change ? Pas grand chose : il suffit par exemple de remplacer les premières lignes :

```
// la liste des informations
$liste=array("Jean N  mar/nemar/jjjj",
            "Sophie Fonfec/fonfec/ssss",
            "Yves Adrouille-Toultan/adrouille/yyyy");
```

par la lecture d'un fichier contenant les m  mes informations :

```
// lire la liste des informations
$liste=file("mettre ici le chemin d'acc  s au fichier");
```

Le reste du programme ne change pas.

Le fichier    lire sera alors un simple fichier texte, qui contiendra dans le cas pr  sent :

```
Jean N  mar/nemar/jjjj
Sophie Fonfec/fonfec/ssss
Yves Adrouille-Toultan/adrouille/yyyy
```

12.5. Cacher les mots de passe

La solution pr  c  dente a un *tr  s gros inconv  nient* si vous placez le fichier de mots de passe n'importe o  ... N'oubliez pas que les pages web (HTML et PHP) sont faites pour   tre vues par tout le monde, depuis n'importe o  . Pour permettre cela les droits d'acc  s de l'arborescence web sont positionn  s    *lisible par tous* (« `chmod a+r*` ») et il ne faut *surtout pas changer cela*.

  a ne posera aucun probl  me, tant que vous n'y entreposez pas de donn  es confidentielles.

Mais les mots de passe,   a c'est confidentiel !

Alors, comment faire ?

Il suffit de ne pas mettre votre fichier de mot de passe dans l'arborescence web, mais de lui choisir un emplacement quelconque dont la seule obligation est d'  tre accessible pour l'utilisateur *apache* (c'est lui qui ex  cute les programmes PHP).

Son chemin d'acc  s dans l'instruction

```
$liste=file("mettre ici le chemin d'acc  s au fichier");
```

devra   tre absolu, c'est-  -dire partir de la racine et commencer par un *slash* (/).

12.6. Crypter les mots de passe

Afin de ne pas rendre la vie trop facile    un   ventuel pirate, il peut aussi   tre int  ressant de crypter les mots de passe qui sont dans le fichier.

Cela se fait avec la fonction `crypt`.

Il n'y a pas d'algorithme de d  cryptage.

Mais il ne faut pas croire que cela puisse   tre une protection suffisante. En effet si les mots de passe, m  me crypt  s, sont accessibles, il suffit de recopier chez soi les versions crypt  es, et de les d  coder tranquillement par la *force brute*.

Vous pourrez trouver un exemple de ce que cela signifie en ex  cutant le programme suivant `decrypt.php`⁶ :

6. <http://grappa.univ-lille3.fr/polys/php/exemples/decrypt.php>

Il décrypte tout mot de passe dont vous fournissez la version cryptée d'une manière totalement inintelligente, voire même plutôt bestiale : il essaie toutes les chaînes de caractères possibles (en commençant par 1 caractère, puis 2, etc.) en les énumérant systématiquement (par exemple pour 2 caractères : aa, ab, ac, ad, ae, ... , az, ba, bc, ...). Pour chaque chaîne il calcule sa version cryptée (avec la fonction `crypt`) et la compare à celle qu'on lui a fournie. Quand elles sont identiques, il sait qu'il vient de trouver le mot de passe.

Idiot, d'accord, mais ça marche.

Remarque

Pour ne pas encombrer inutilement le serveur le temps de calcul a été limité, mais cela permet quand même de trouver à coup sûr (si le serveur n'a pas trop de travail par ailleurs) n'importe quel mot de passe de 4 lettres au plus.

Et tout cela avec un programme qui n'est pas vraiment optimisé, et dans un langage qui n'est pas le plus rapide...

Il faut donc *vraiment* cacher les mots de passe...

Chapitre 13. Sessions

13.1. Préambule

Le texte qui suit est une partie de la page consacrée aux sessions¹ sur le site de <http://fr.php.net> :

Le support des sessions de PHP est un moyen de préserver des données entre plusieurs accès. Cela vous permet de créer des applications personnalisées, et d'augmenter l'attrait de votre site.

Chaque visiteur accédant à votre page web se voit assigner un identifiant unique, appelé « identifiant de session ». Il peut être stocké soit dans un cookie, soit propagé dans l'URL.

Le support des sessions vous permet d'enregistrer un nombre illimité de variables qui doivent être préservées entre les requêtes. Lorsqu'un visiteur accède à votre site, PHP va vérifier automatiquement (si `session.auto_start` est activé) ou sur demande (explicitement avec `session_start()` ou implicitement avec `session_register()`) s'il existe une session du même nom. Si c'est le cas, l'environnement précédemment sauvé sera recréé.

La seule fonction nécessaire pour un usage simple des sessions est `session_start()`³, mais je ne peux que vous encourager à aller voir les autres⁴

13.2. Application : identification des visiteurs

Principe :

- Deux variables de session contiennent le *login* et le mot de passe du visiteur (variables vides si le visiteur n'est pas identifié).
- Si le visiteur n'est pas identifié, le menu propose un choix *Se connecter* qui envoie sur un formulaire de saisie (on peut aussi inclure ce formulaire directement dans le menu). Si le visiteur est identifié, le menu propose un choix *Se déconnecter* qui vide les variables de session qui contiennent le *login* et le mot de passe du visiteur.
- Chaque page commence par quelque chose qui ressemblera à :

```
if (le visiteur n'a pas le bon niveau d'autorisation pour cette page) {  
    message de refus  
    fin de page  
    return ;  
}
```

- L'évaluation du niveau d'autorisation peut se faire de plusieurs manières :
 - (Cette solution est mauvaise, car elle complique les modifications.) Coder en dur le test sur chaque possibilité :

```
if ((nom=="bidule") && (mot de passe=="truc")) {  
    ...  
} elseif ((nom=="chose") && (mot de passe=="n'importe quoi")) {  
    ...  
} else  
    etc.
```

- (Mieux.) Ranger les noms et mots de passe dans un tableau, et parcourir le tableau pour faire la vérification.
- (Parfait.) La même chose, mais créer une table spéciale dans la base de données au lieu d'utiliser un tableau.
- Pour être parfait ce serait bien que ce soit une fonction qui affiche le menu, et que cette fonction décide d'elle-même des choix disponibles en fonction du niveau d'autorisation du visiteur.
- Exemple d'utilisation des sessions : http://grappa.univ-lille3.fr/~gonzalez/session_php.

1. <http://fr.php.net/manual/fr/ref.session.php>

3. <http://fr.php.net/manual/fr/function.session-start.php>

4. <http://fr.php.net/manual/fr/ref.session.php#session.customhandler>

II. Hors programme

Chapitre 14. Utiliser PHP sur une base de données PostgreSQL

Ce chapitre ne figure ici qu'en guise de témoignage : il est préférable d'utiliser `PDO` pour les accès aux bases de données (Chapitre 10).

14.1. Connexion à la base

La fonction à utiliser est « `pg_connect`¹ ».

Syntaxe résumée :

```
$www = pg_connect ("host=xxx dbname=ddd user=yyy password=zzz") ;
```

Avec :

`$www`

une variable qui sera utilisée comme identifiant de connexion à chaque exécution d'une requête sur la base.

`ddd`

la base de données à laquelle vous essayez de vous connecter.

`xxx`

le nom (l'adresse) de la machine sur laquelle se trouve la base de données.

`yyy`

le nom de l'utilisateur.

`zzz`

le mot de passe de l'utilisateur.

14.2. Exécution d'une requête

La fonction à utiliser est « `pg_query`² ».

Syntaxe résumée :

```
$rrr = pg_query ($www, $requete) ;
```

Avec :

`$rrr`

une variable qui sera utilisée comme identifiant si vous voulez obtenir les résultats de l'exécution de cette requête.

`$www`

un identifiant de connexion obtenu par `pg_connect` (voir ci-dessus).

`$requete`

une chaîne de caractère contenant la requête à exécuter (ce peut être une chaîne littérale, ou une variable).

1. <http://fr.php.net/manual/fr/function.pg-connect.php>
2. <http://fr.php.net/manual/fr/function.pg-query.php>

14.3. Obtention des résultats d'une requête

Les fonctions à utiliser sont :

- « `pg_numrows`³ » :

Syntaxe résumée :

```
$nnn = pg_numrows ($rrr) ;
```

Avec :

`$nnn`

une variable qui contiendra le nombre de lignes du résultat.

`$rrr`

un identifiant d'exécution de requête obtenu par `pg_query` (voir ci-dessus).

- « `pg_fetch_array`⁴ » :

Syntaxe résumée :

```
$l111 = pg_fetch_array ($rrr, $iii) ;
```

Avec :

`$rrr`

un identifiant d'exécution de requête obtenu par `pg_query` (voir ci-dessus).

`$iii`

numéro de la ligne que vous voulez obtenir.

`$l111`

une variable qui contiendra une table constituée de tous les champs de la ligne demandée.

Il s'agit d'un tableau associatif dont les clefs sont les titres des colonnes.

14.4. Un exemple

Ce programme

```
<?php
$c=pg_connect ("dbname=**** user=**** password=****");
$r=pg_query ($c , "SELECT nom,prenom FROM emp ORDER BY nom");
for ($i=0; $i<pg_numrows($r); $i++) {
    $l=pg_fetch_array($r,$i);
    echo $l["prenom"]." <em>".$l["nom"]."</em>.\n";
}
?>
```

produit le résultat suivant :

Yves Adrouille-Toultan. Tex Ajerre. Quentin Amartakaldire. Yvon Anchier. Helmut Ardelpic. Terry Blaireur. Thierry Chmonfisse. Mélusine Enfaillite. Odile Eurktumeme. Teddy Fairant. Sophie Fonfec. Olaf Hotdeugou. Xavier Kaécouté. Alex Kuzbidon. Pacôme Odlavieille. Adhémair Patamob. Humphrey Peursconla. Thomas Phototetedemort. Elsa Rivenbusse. Armand Teutmaronne. Samira Traibien. Hélène Vrante. Agathe Zeublouze. Debbie Zoudanlkou.

3. <http://fr.php.net/manual/fr/function.pg-numrows.php>

4. <http://fr.php.net/manual/fr/function.pg-fetch-array.php>

14.5. Requêtes autres que SELECT

Les requêtes de type INSERT, UPDATE, et DELETE seront exécutées de la même manière que les requêtes de type SELECT, avec la fonction « pg_query⁵ ».

Il sera cependant important de contrôler les résultats de cette exécution. C'est ce que nous allons étudier dans ce chapitre.

14.6. Suppression des messages d'erreurs

Vous pouvez avoir envie de gérer vous-même les erreurs vos requêtes (par exemple éviter l'affichage des « Warning » et autres « Query failed »).

Pour cela il suffit de préfixer les fonctions que vous utilisez avec « @ » (opérateur de contrôle d'erreur⁶) : lorsque cet opérateur est ajouté en préfixe d'une expression PHP, les messages d'erreur qui peuvent être générés par cette expression seront ignorés.

Par exemple au lieu d'écrire

```
$res = pg_query ($c,$req) ;
```

on écrira

```
$res = @pg_query ($c,$req) ;
```

Remarque

L'opérateur @ ne fonctionne qu'avec les expressions. La règle générale de fonctionnement est la suivante : si vous pouvez prendre la valeur de quelque chose, vous pouvez le préfixer avec @. Par exemple, vous pouvez ajouter @ aux variables, fonctions, à include, aux constantes, etc. Vous ne pourrez pas le faire avec des éléments de langage tels que les classes, if et foreach, etc.

14.7. Gestion des erreurs et des messages

Il est impératif, quand vous supprimez les avertissements, de gérer vous-même les erreurs qui peuvent survenir.

Dans le cas d'accès à une base PostgreSQL il existe un certain nombre de fonctionnalités pour cela.

14.7.1. Résultat renvoyé par pg_query en cas d'échec

« pg_query⁷ » retourne FALSE en cas d'échec (c'est-à-dire quand aucun résultat réel ne peut exister).

On peut ainsi traiter l'échec éventuel de la requête :

```
if ( $r=@pg_query($c,$req) ) {
    // traitement cas de succès
    . . .
} else {
    // traitement cas d'échec
    . . .
}
```

14.7.2. Obtenir le texte de la dernière erreur

La fonction « pg_last_error⁸ » renvoie un éventuel message d'erreur correspondant à la dernière exécution d'une requête.

5. <http://fr.php.net/manual/fr/function.pg-query.php>

6. <http://fr.php.net/manual/fr/language.operators.errorcontrol.php>

7. http://fr.php.net/manual/fr/function.pg_query.php

8. <http://fr.php.net/manual/fr/function.pg-last-error.php>

14.7.3. Obtenir le statut de la connexion

On peut connaître l'état de la connexion en utilisant la fonction « `pg_result_status`⁹ ». Les valeurs retournées possibles sont :

- `PGSQL_EMPTY_QUERY` (0)
- `PGSQL_COMMAND_OK` (1)
- `PGSQL_TUPLES_OK` (2)
- `PGSQL_COPY_TO` (3)
- `PGSQL_COPY_FROM` (4)
- `PGSQL_BAD_RESPONSE` (5)
- `PGSQL_NONFATAL_ERROR` (6)
- `PGSQL_FATAL_ERROR` (7)

14.7.4. Obtenir le nombre de lignes affectées par une requête

« `pg_affected_rows`¹⁰ » retourne le nombre de lignes affectées par les requêtes de type `INSERT`, `UPDATE`, et `DELETE`, exécutées par la fonction `pg_query`.

Si aucune ligne n'a été affectée, `pg_affected_rows` retourne 0.

9. <http://fr.php.net/manual/fr/function.pg-connection-status.php>

10. <http://fr.php.net/manual/fr/function.pg-affected-rows.php>

Chapitre 15. pear**db**, une présentation

Ce chapitre ne figure ici qu'en guise de témoignage : il est préférable d'utiliser `PDO` pour les accès aux bases de données (Chapitre 10).

15.1. Qu'est-ce que pear

Ce texte est extrait de la Présentation de pear¹, sur JDN² (Journal du Net).

`pear` (*PHP Extension and Application Repository*) est une archive de composants `PHP` réutilisables et un projet majeur de la communauté `PHP`.

C'est une archive stockant de la façon la plus exhaustive possible des programmes, des macros et de la documentation. Son but est de fournir aux développeurs de bibliothèques/modules `PHP` un moyen d'organiser leur code `PHP` ou `C` d'une manière définie et partagée par les autres développeurs, ainsi que d'offrir à la communauté `PHP` une seule source pour ces ressources. En bref, c'est une architecture et un système de distribution pour composants `PHP` réutilisables.

15.2. Quelle utilité ?

`pear` est utile pour tout développeur `PHP`.

Mais nous allons surtout l'utiliser pour son module `DB` (*DataBase*) qui est une *couche d'abstraction* pour les accès aux bases de données.

L'hétérogénéité des moyens d'accès aux bases de données a été un des problèmes les plus gênants en `PHP`³. Le problème vient du fait que, par exemple, les fonctions permettant l'accès à une base `PostgreSQL` sont différentes de celles pour `MySQL`, et elles sont toutes différentes de celles pour `SQLite`, etc.

Cela a pour effet de rendre plus difficile la migration d'un système vers un autre : vous avez écrit (par exemple en cours) un site utilisant une base `PostgreSQL` et vous voulez le transférer chez un hébergeur qui n'offre que `MySQL` (*Free*, par exemple) : votre seule solution est de reprendre (et corriger) dans toutes vos pages tous les appels à des fonctions concernant `PostgreSQL`. Il peut y en avoir beaucoup...

Une *couche d'abstraction*⁴ permet d'éviter ce problème : tous vos accès à votre base de données passent par elle (avec la même syntaxe, quelle que soit la base de données) et c'est elle qui se débrouille pour s'adapter à la base de données utilisée. Pour changer de base de données (passer de `PostgreSQL` à `MySQL` par exemple) vous n'avez en général qu'à modifier une ligne, celle qui donne les paramètres de la base de données utilisée.

Vous trouverez sur le *web* de la documentation en français sur `pear`⁵, et en particulier le module `peardb`⁶.

15.3. Utiliser pear**db**

Pour que les fonctions de `peardb` soient utilisables dans votre programme, il doit commencer par :

```
require_once 'DB.php';
```

15.4. Se connecter à la base de données

Vous trouverez plus de détails sur la page « Introduction - DSN⁷ ».

La première action, avant d'utiliser une base de données, est de s'y connecter.

1. http://developpeur.journaledunet.com/tutoriel/php/021017php_pear1.shtm
2. <http://developpeur.journaledunet.com/>
3. Il semble que `PHP5` ait réglé ce problème, mais nous utiliserons `PHP4`...
4. Il existe d'autres *couches d'abstraction* : `PHPLib`, `ADODB`, `MetatData`, etc.
5. <http://pear.php.net/manual/fr/>
6. <http://pear.php.net/manual/fr/package.database.php>
7. <http://pear.php.net/manual/fr/package.database.db.intro-dsn.php>

Pour cela il faut définir un *DSN* (*Data Source Name*) qui donnera tous les renseignements sur la base de données à utiliser : nom, utilisateur, mot de passe, machine hébergeant le serveur, type de SGBD, etc. Sa syntaxe (légèrement simplifiée) est la suivante :

```
type_de_BD://utilisateur:mot_de_passe@machine/base_de_données?option=valeur
```

Par exemple :

- Pour une base PostgreSQL :

```
pgsql://utilisateur:mot_de_passe@localhost/base
```

- Pour une base SQLite, sous linux :

```
sqlite:///chemin_complet_vers_fichier.db?mode=0666
```

- Pour une base SQLite, sous Windows :

```
sqlite:///c:/chemin_complet_vers_fichier.db?mode=0666
```

Il suffit ensuite d'exécuter la commande

```
$db =& DB::connect (votre_DSN) ;
```

Le nom de la variable `$db` est sans importance. Ce n'est qu'une variable, vous pouvez lui donner le nom qui vous plaît. Il faut seulement garder à l'esprit qu'il faut conserver cette variable (quel que soit son nom), c'est elle qui permettra d'accéder à la base de données avec les fonctions des chapitres suivants.

15.5. Gestion des erreurs

15.5.1. En cas d'erreur...

Après toute utilisation d'une fonction de *peardb*, y compris une tentative de connexion, il peut être utile de vérifier qu'aucune erreur ne s'est produite :

```
if (PEAR::isError($db)) {
    echo "<hr />"
    . "Message Standard      : " . $db->getMessage() . "<hr />\n"
    . "Message DBMS/Utilisateur : " . $db->getUserInfo() . "<hr />\n"
    . "Message DBMS/Débogage   : " . $db->getDebugInfo() . "<hr />\n";
    exit;
}
```

(À vous de choisir les messages qu'il vous paraît intéressant d'afficher...)

15.5.2. Application

On peut utiliser la gestion des erreurs pour écrire un programme plus portable.

Si vous écrivez un programme PHP que vous voulez utiliser à plusieurs endroits où les conditions sont différentes (en salles de TP, chez vous, chez un hébergeur, etc.), vous devrez en principe modifier le programme à chaque fois que vous le transférez d'un site à l'autre. Même si l'utilisation de *peardb* réduit le nombre de modifications à faire, il reste toujours le *DSN* à adapter.

Cependant vous pouvez oublier également cette manipulation en utilisant la gestion des erreurs :

```
$db =& DB::connect (DSN_pour_les_salles_de_TP, $options);
if (PEAR::isError($db)) {
    $db =& DB::connect ($DSN_pour_votre_machine, $options);
    if (PEAR::isError($db)) {
        $db =& DB::connect (DSN_pour_votre_hébergeur, $options);
        if (PEAR::isError($db)) {
```

```

        echo "<hr />Connexion impossible<hr />\n";
        exit;
    }
}

```

15.6. Exécuter une requête

La syntaxe est simple :

```
$res =& $db->query (votre_requête) ;
```

La valeur renvoyée (rangée ici dans la variable `$res`) N'EST PAS le résultat de l'exécution de la requête, mais un identifiant (propre au système, sa valeur réelle ne nous intéresse pas) qui nous permettra d'accéder aux résultats grâce aux fonctions décrites ci-dessous.

15.7. Accéder aux résultats d'une requête `select`

`peardb` fournit deux fonctions pour traiter les lignes résultant d'une requête : `fetchRow()`⁸ et `fetchInto()`⁹.

15.7.1. `fetchRow()`

`fetchRow()` retourne la rangée.

Exemple d'utilisation en supposant que le mode par défaut de récupération est `DB_FETCHMODE_ORDERED` (voir `setfetchmode()`) :

```

while ($row =& $res->fetchRow()) {
    echo $row[0] . "\n";
}

```

15.7.2. `fetchInto()`

`fetchInto()` a besoin d'une variable, à qui on assignera par référence le contenu de la rangée du résultat et retournera `DB_OK`.

Exemple d'utilisation en supposant que le mode par défaut de récupération est `DB_FETCHMODE_ORDERED` (voir `setfetchmode()`) :

```

while ($res->fetchInto($row)) {
    echo $row[0] . "\n";
}

```

15.8. `setfetchmode()`

La commande `setfetchmode()`¹⁰ configure le mode de récupération par défaut utilisé par les méthodes `fetch*()` (et `get*()`).

Nous ne l'utiliserons que deux manières (mais il en existe d'autres, allez voir la page qui lui est consacrée¹¹) :

8. <http://pear.php.net/manual/fr/package.database.db.db-result.fetchrow.php>

9. <http://pear.php.net/manual/fr/package.database.db.db-result.fetchinto.php>

10. <http://pear.php.net/manual/fr/package.database.db.db-common.setfetchmode.php>

11. <http://pear.php.net/manual/fr/package.database.db.db-common.setfetchmode.php>

- `setfetchmode(DB_FETCHMODE_ORDERED)` : les rangées sont mises en tableau ordonné, on y accède par leur numéro.
- `setfetchmode(DB_FETCHMODE_ASSOC)` : les rangées sont mises en tableau associatif, on y accède par le titre de la colonne.

Pour bien comprendre la différence vous pouvez utiliser la fonction PHP « `print_r`¹² » (qui permet d’afficher un tableau en entier).

Ainsi ce programme

```
$db->setfetchmode(DB_FETCHMODE_ORDERED); // sortie en tableau ordonné
$res =& $db->query("SELECT * FROM personne"); // une requête
$res->fetchInto($row); // récupération de la 1ère ligne
echo "<pre>"; // affichage « préformaté »
print_r ($row); // affichage de la ligne en tableau
echo "</pre>"; // fin de l’affichage « préformaté »
```

produira cet affichage

```
Array
(
    [0] => 5
    [1] => Kuzbidon
    [2] => Ginette
    [3] => 263
    [4] => 393
    [5] =>
    [6] =>
    [7] =>
)
```

Tandis que celui-ci

```
$db->setfetchmode(DB_FETCHMODE_ASSOC); // sortie en tableau associatif
$res =& $db->query("SELECT * FROM personne"); // une requête
$res->fetchInto($row); // récupération de la 1ère ligne
echo "<pre>"; // affichage « préformaté »
print_r ($row); // affichage de la ligne en tableau
echo "</pre>"; // fin de l’affichage « préformaté »
```

produira cet affichage

```
Array
(
    [codepersonne] => 5
    [nom] => Kuzbidon
    [prenom] => Ginette
    [numerodanslarue] => 263
    [refrue] => 393
    [tel] =>
    [fax] =>
    [email] =>
)
```

On peut également utiliser les constantes `DB_FETCHMODE_ORDERED` et `DB_FETCHMODE_ASSOC` directement dans les fonctions `fetchRow()` et `fetchInto()` comme par exemple :

```
$row =& $res->fetchRow(DB_FETCHMODE_ASSOC);
```

ou

```
$res->fetchInto($row,DB_FETCHMODE_ORDERED);
```

12. <http://fr.php.net/manual/fr/function.print-r.php>

15.9. *peardb*, informations sur les requêtes

Ce chapitre provient de « PEAR-DB - Résultats¹³ ».

Il y a quatre manières de récupérer des informations intéressantes des jeux de résultats.

15.9.1. `numRows()`

`numRows()`¹⁴ retourne le nombre de lignes disponibles dans le jeu de résultats issu d'une requête `SELECT`.

```
<?php
// Une fois que vous avez un objet DB valide nommé $db
$res =& $db->query('SELECT bla bla');
echo $res->numRows();
?>
```

15.9.2. `numCols()`

`numCols()`¹⁵ retourne le nombre de colonnes disponibles dans le jeu de résultats issu d'une requête `SELECT`.

```
<?php
// Une fois que vous avez un objet DB valide nommé $db
$res =& $db->query('SELECT bla bla');
echo $res->numCols();
?>
```

15.9.3. `affectedRows()`

`affectedRows()`¹⁶ retourne le nombres de lignes affectées par une requête du type `INSERT`, `UPDATE` ou `DELETE`.

```
<?php
// Souvenez-vous que cette requête ne retourne pas d'objet de résultats
$db->query('DELETE bla bla');
echo "J'ai effacé " . $db->affectedRows() . " clients";
?>
```

15.9.4. `tableInfo()`

`tableInfo()`¹⁷ retourne un tableau associatif contenant des informations sur les colonnes issues d'un résultat de requête du type `SELECT`.

```
<?php
// Une fois que vous avez un objet DB valide nommé $db
$res =& $db->query('SELECT bla bla');
print_r($res->tableInfo());
?>
```

13. <http://pear.php.net/manual/fr/package.database.db.intro-fetch.php>

14. <http://pear.php.net/manual/fr/package.database.db.db-result.numrows.php>

15. <http://pear.php.net/manual/fr/package.database.db.db-result.numcols.php>

16. <http://pear.php.net/manual/fr/package.database.db.db-common.affectedrows.php>

17. <http://pear.php.net/manual/fr/package.database.db.db-common.tableinfo.php>

Chapitre 16. dbx

Ce chapitre ne figure ici qu'en guise de témoignage : il est préférable d'utiliser PDO pour les accès aux bases de données (Chapitre 10).

16.1. Qu'est-ce que dbx

Ce texte est extrait de « XXVII. Fonctions dbx¹ », un chapitre du manuel PHP sur sur PHP net².

Le module dbx est un module d'abstraction de base de données (db pour database (base de données) et 'X' pour toutes les bases supportées). Les fonctions dbx vous permettent d'accéder à toutes les bases supportées, avec la même convention. Les fonctions dbx elles-mêmes ne s'interfaçent pas directement avec les bases de données, mais s'interfaçent avec les modules utilisées pour supporter ces bases.

Pour pouvoir utiliser une base de données avec le module dbx, le module doit être soit lié, soit chargé dans PHP et le module de base de données doit être supporté par le module dbx. Actuellement les bases suivantes sont supportées, et d'autres suivront :

- FrontBase (disponible depuis PHP 4.1.0),
- Microsoft SQL Server,
- MySQL,
- ODBC,
- PostgreSQL,
- Sybase-CT (disponible depuis PHP 4.2.0),
- Oracle (disponible depuis PHP 4.3.0),
- SQLite (PHP 5).

16.2. Quelle utilité ?

L'hétérogénéité des moyens d'accès aux bases de données a été un des problèmes les plus gênants en PHP. Le problème vient du fait que, par exemple, les fonctions permettant l'accès à une base PostgreSQL sont différentes de celles pour MySQL, et elles sont toutes différentes de celles pour SQLite, etc.

Cela a pour effet de rendre plus difficile la migration d'un système vers un autre : vous avez écrit (par exemple en cours) un site utilisant une base PostgreSQL et vous voulez le transférer chez un hébergeur qui n'offre que MySQL (*Free*, par exemple) : votre seule solution est de reprendre (et corriger) dans toutes vos pages tous les appels à des fonctions concernant PostgreSQL. Il peut y en avoir beaucoup...

Une *couche d'abstraction*³ permet d'éviter ce problème : tous vos accès à votre base de données passent par elle (avec la même syntaxe, quelle que soit la base de données) et c'est elle qui se débrouille pour s'adapter à la base de données utilisée. Pour changer de base de données (passer de PostgreSQL à MySQL par exemple) vous n'avez en général qu'à modifier une ligne, celle qui donne les paramètres de la base de données utilisée.

16.3. Se connecter à la base de données

Vous trouverez plus de détails sur la page dbx_connect⁴ du manuel PHP⁵.

La première action, avant d'utiliser une base de données, est de s'y connecter.

Il suffit pour cela d'exécuter la commande

```
$db = dbx_connect (type_de_base,  
                  machine_qui_héberge_la_base,  
                  nom_de_la_base,
```

1. <http://fr.php.net/manual/fr/ref.dbx.php>

2. <http://fr.php.net/>

3. Il existe d'autres *couches d'abstraction* : PHPLib, ADODB, MetaData, Pear-DB, etc.

4. <http://fr.php.net/manual/fr/function.dbx-connect.php>

5. <http://fr.php.net/manual/fr/index.php>

```
nom_d_utilisateur,  
mot_de_passe)
```

Le nom de la variable `$db` est sans importance. Ce n'est qu'une variable, vous pouvez lui donner le nom qui vous plaît. Il faut seulement garder à l'esprit qu'il faut conserver cette variable (quel que soit son nom), c'est elle qui permettra d'accéder à la base de données avec les fonctions des chapitres suivants.

Le `type_de_base` quant à lui est une valeur parmi `DBX_MYSQL`, `DBX_ODBC`, `DBX_PGSQL`, `DBX_MSSQL`, `DBX_FBSQL`, `DBX_SYBASECT`, `DBX_OCI8`, `DBX_SQLITE`.

16.4. Exécuter une requête

Vous trouverez plus d'information sur la page `dbx_query`⁶ du manuel PHP.

La syntaxe est simple :

```
$res = dbx_query($db, votre_requête);
```

La valeur renvoyée (rangée ici dans la variable `$res`) N'EST PAS le résultat de l'exécution de la requête, mais un identifiant (propre au système, sa valeur réelle ne nous intéresse pas) qui nous permettra d'accéder aux résultats grâce aux fonctions décrites ci-dessous.

16.5. Nombre de lignes et colonnes d'une requête `select`

On peut facilement obtenir le nombre de lignes et de colonnes du résultat d'une requête de type `SELECT` :

```
$result = dbx_query ($db, 'SELECT id FROM table');  
echo $result->rows; // nombre de lignes  
echo $result->cols; // nombre de champs
```

16.6. Accéder aux résultats d'une requête `select`

`$result` étant un résultat valide renvoyé par `dbx_query()`, `$result->data` est un tableau à 2 dimensions qui contient les résultats : le premier indice concerne les lignes, le deuxième concerne les colonnes.

16.6.1. Accès par le nom des colonnes

Exemple :

```
$result = dbx_query($db, "SELECT * FROM ville");  
foreach ( $result->data as $row ) {  
    echo $row["codepostal"]." - ".$row["ville"]."<br />\n";  
}
```

16.6.2. Accès par le numéro des colonnes

Exemple :

```
$result = dbx_query($db, "SELECT * FROM ville");  
foreach ( $result->data as $row ) {  
    for ($i=0;$i<$result->cols;$i++) {  
        echo $row[$i]." ** ";  
    }  
    echo "<br />\n";  
}
```

6. <http://fr.php.net/manual/fr/function.dbx-query.php>

```
}
```

16.6.3. Accès par le numéro des lignes et le numéro des colonnes

Exemple :

```
$result = dbx_query($db, "SELECT * FROM ville");
for ($l=0;$l<$result->rows;$l++) {
    for ($i=0;$i<$result->cols;$i++) {
        echo $result->data[$l][$i]. " ** ";
    }
    echo "<br />\n";
}
```


Chapitre 17. Exploration du contenu d'un répertoire

Les corrigés des exercices de ce chapitre se trouvent Chapitre 25.

17.1. Les notions nécessaires en PHP

17.1.1. Exploration d'un répertoire

La première fonction à connaître est « `opendir`¹ ». Elle est destinée à être utilisée avec les fonctions « `readdir`² » et « `closedir`³ ».

Le programme suivant (`opendir.php`) affiche le contenu du répertoire `/mnt` :

```
<?php
$dir = opendir("/mnt");
while($file = readdir($dir)) {
    echo "$file\n";
}
closedir($dir);
?>
```

C'est également une excellente application de l'utilisation de la structure `while`.

17.1.2. Extraction d'une sous-chaîne de caractères

Il est souvent nécessaire de tester une partie d'une chaîne de caractères ; il faut donc pouvoir l'isoler.

Ce sera fait avec la fonction « `substr`⁴ », dont voici quelques exemples de résultats

Pour voir le résultat produit : `substr.php`⁵ :

```
<?php
echo substr("abcdef", 1). "<br />"; // retourne "bcdef"
echo substr("abcdef", 1, 3). "<br />"; // retourne "bcd"
echo substr("abcdef", -1). "<br />"; // retourne "f"
echo substr("abcdef", -2). "<br />"; // retourne "ef"
echo substr("abcdef", -3, 1). "<br />"; // retourne "d"
echo substr("abcdef", 1, -1). "<br />"; // retourne "bcde"
?>
```

Le premier paramètre est la chaîne dont on doit extraire une partie.

Le deuxième paramètre indique la position de la lettre où va commencer l'extraction. S'il est négatif, cela indique qu'on compte à partir de la fin de la chaîne.

Si le troisième paramètre n'est pas mentionné, on va jusqu'à la fin de la chaîne. S'il est présent, il indique le nombre de lettres à prendre.

17.2. Les notions nécessaires en HTML

17.2.1. Images cliquables

Comment faire en sorte qu'en cliquant sur une image on obtienne le même résultat qu'en cliquant que un mot ?

Tout simplement en faisant la même chose avec l'image qu'avec le texte...

1. <http://fr.php.net/manual/fr/function.opendir.php>
2. <http://fr.php.net/manual/fr/function.readdir.php>
3. <http://fr.php.net/manual/fr/function.closedir.php>
4. <http://fr.php.net/manual/fr/function.substr.php>
5. <http://grappa.univ-lille3.fr/polys/php/exemples/substr.php>

Pour transformer un mot en lien il suffit de l'inclure entre `` et ``.

On fera la même chose avec une image... Ainsi écrire

```
<a href="simple.php"></a>
```

aura pour effet d'afficher l'image « mini/rpb.jpg » tout en faisant en sorte que si on clique dessus on arrive sur la page.

17.2.2. Pour qu'un lien affiche une image plutôt qu'une page HTML ou PHP

Il suffit de mettre dans le lien l'adresse de l'image au lieu de l'adresse d'une page.

17.3. Exercices

Tous les exercices qui suivent sont à faire dans un répertoire qui contiendra des images (et d'autres choses aussi, pourquoi pas), ainsi qu'un sous-répertoire qui contiendra les mêmes images mais en réduction (en onglets, ou *thumbnails* en anglais).

1. Première version du programme⁶ : on affiche les images trouvées.
2. Deuxième version du programme⁷ : on affiche les onglets, qui font liens vers les vraies images.
3. Troisième version du programme⁸ : on affiche les onglets, qui font liens vers des pages contenant les vraies images dans un cadre.
4. Version *parfaite* du programme⁹ : on affiche les onglets en tableau régulier. Ils font liens vers des pages contenant les vraies images dans un cadre.

6. <http://grappa.univ-lille3.fr/polys/php/exemples/simple.php>

7. <http://grappa.univ-lille3.fr/polys/php/exemples/thumb.php>

8. <http://grappa.univ-lille3.fr/polys/php/exemples/thumbplus.php>

9. <http://grappa.univ-lille3.fr/polys/php/exemples/parfait.php>

Chapitre 18. News...

Les corrigés des exercices de ce chapitre se trouvent Chapitre 26.

Nous allons terminer ce tour d'horizon de PHP avec un mini-projet dont le but est d'afficher des articles de presse, dans différents domaines.

Vous pourrez trouver un exemple sur internet¹.

18.1. Présentation

Ces articles sont rangés chacun dans un fichier.

Le nom des fichiers est composé de trois parties :

1. 3 lettres qui définissent le domaine :

- `inf` pour *informatique*,
- `msc` pour *miscellaneous* (divers, autrement dit ce qui n'entre pas dans les autres catégories),
- `sci` pour *science*.

2. 3 chiffres qui forment en fait un numéro d'ordre du fichier : 001, 002, 003, 004, 005, etc.

3. une extension `jnl`.

On aura par exemple :

```
inf001.jnl, inf002.jnl, inf003.jnl, inf004.jnl, inf005.jnl, msc001.jnl, msc002.jnl, sci001.jnl,
sci002.jnl, sci003.jnl, etc.
```

Chaque fichier contiendra du texte, sans codes HTML ni PHP. La première ligne devra obligatoirement être le titre de l'article.

Un des fichiers (`inf004.jnl`) contient par exemple :

```
Xenux un site simple pour le débutant
Xenux.fr.st est un nouveau site pour les pressés de Linux sans
chichi sans blabla. On sait où on va dès le début de la lecture de
l'article.
Intéressant et ça marche.....
À Bientôt
```

18.2. Votre travail

Le site à réaliser aura les caractéristiques suivantes :

1. Chaque page devra aller chercher elle-même les fichiers nécessaires dans le répertoire : pas de liste toute faite des articles, il suffit de copier le fichier de l'article pour qu'il soit pris en compte.

Inspirez-vous de ce qui a été fait dans le cours sur l'exploration du contenu d'un répertoire (Chapitre 17).

2. Vous devez réaliser une page `listenoms.php`² qui affiche les noms des fichiers.

3. Vous devez réaliser une page identique à la précédente mais telle que les noms des fichiers soient des liens vers l'affichage des fichiers eux-même (`listeliens.php`³).

1. <http://grappa.univ-lille3.fr/polys/php/exemples/news/>

2. <http://grappa.univ-lille3.fr/polys/php/exemples/news/listenoms.php>

3. <http://grappa.univ-lille3.fr/polys/php/exemples/news/listeliens.php>

4. Vous devez réaliser une page qui affiche le texte de TOUS les articles les uns sous les autres (`affiche tout.php`⁴). Vous n'êtes pas obligés de soigner la présentation, c'est pour un des exercices suivants.
5. Vous devez réaliser une page `affiche titre.php`⁵ qui affiche uniquement les titres des articles.
6. Vous devez réaliser une page identique à la précédente, mais dans laquelle les titres affichés sont des liens vers les articles correspondants (`affiche titre lien.php`⁶).
7. Vous devez réaliser une page qui affiche proprement les articles (`affiche tout_br.php`⁷) :
 - les articles sont bien séparés (par une ligne horizontale par exemple),
 - les titres sont bien visibles (en utilisant par exemple la construction `<H2>...</H2>`),
 - les fins de paragraphe du texte original sont respectés.
8. Vous devez réaliser une page qui permet de choisir le sujet qui nous intéresse (informatique, miscalaneous, divers ; voir section *description*) et qui affiche les articles correspondants (`sujet.php`⁸).
9. Une page de menu⁹ qui permette d'aller facilement vers chacune de ces pages, et d'en revenir tout aussi facilement.

18.3. Si vous avez le temps...

Si vous avez le temps vous pourrez également ajouter ces quelques fonctions à votre travail :

1. un formulaire qui permette d'écrire un article (qui sera alors ajouté automatiquement aux autres),
2. un moyen de corriger un article existant,
3. un moyen de supprimer un article existant.

4. <http://grappa.univ-lille3.fr/polys/php/exemples/news/affiche tout.php>

5. <http://grappa.univ-lille3.fr/polys/php/exemples/news/affiche titre.php>

6. <http://grappa.univ-lille3.fr/polys/php/exemples/news/affiche titre lien.php>

7. http://grappa.univ-lille3.fr/polys/php/exemples/news/affiche tout_br.php

8. <http://grappa.univ-lille3.fr/polys/php/exemples/news/sujet.php>

9. <http://grappa.univ-lille3.fr/polys/php/exemples/news/index.php>

Chapitre 19. Les fichiers

Les corrigés des exercices de ce chapitre se trouvent Chapitre 24.

Un fichier (ou *file* en anglais, ou *document* dans le monde Windows) est l'objet qui permet à un système d'exploitation (donc aussi aux programmes qu'il supporte) d'enregistrer des informations sur le disque dur (ou la disquette).

19.1. Manipulations de base

Pour utiliser des fichiers il faut savoir les écrire et les relire.

19.1.1. Ouverture (et fermeture) d'un fichier

Avant tout accès à un fichier il faut l'ouvrir, ce qui se fait par la fonction « `fopen`¹ » dont la syntaxe est :

```
int fopen (string fichier, string mode)
```

où :

- `int` signifie que la fonction renvoie une valeur entière.
- `fichier` est le nom du fichier à ouvrir.
- `mode` décrit la façon dont le fichier sera utilisé. Les valeurs les plus courantes sont :
 - `'r'` : Ouvre en lecture seule, et place le pointeur de fichier au début du fichier.
 - `'w'` : Ouvre en écriture seule ; place le pointeur de fichier au début du fichier et réduit la taille du fichier à 0. Si le fichier n'existe pas, on tente de le créer.
 - `'a'` : Ouvre en écriture seule ; place le pointeur de fichier à la fin du fichier. Si le fichier n'existe pas, on tente de le créer.

Un fichier ouvert par « `fopen` » sera fermé par « `fclose`² ».

19.1.2. Écrire dans un fichier

Nous n'étudierons que les fichiers de texte. Mais vous trouverez plus d'information sur les fichiers dans l'aide de PHP³.

Pour écrire dans un fichier de texte il faut utiliser la fonction « `fwrite`⁴ » (ou la fonction « `fputs`⁵ » qui est en tous points identique).

19.1.3. Lire le contenu d'un fichier

Deux solutions radicalement différentes :

- « `file`⁶ » qu'on a déjà vue dans le chapitre sur les tableaux, et qui lit le fichier en une seule fois en le rangeant dans un tableau.
- « `fgets`⁷ » qui lit le fichier ligne par ligne, et qui impose donc un certain contrôle.

Il existe d'autres façons de lire les informations rangées dans un fichier mais nous ne les aborderons pas. Comme précédemment vous trouverez plus d'information sur les fichiers dans l'aide de PHP⁸.

1. <http://fr.php.net/manual/fr/function.fopen.php>

2. <http://fr.php.net/manual/fr/function.fclose.php>

3. <http://fr.php.net/manual/fr/html/ref.filesystem.html>

4. <http://fr.php.net/manual/fr/function.fwrite.php>

5. <http://fr.php.net/manual/fr/function.fputs.php>

6. <http://fr.php.net/manual/fr/function.file.php>

7. <http://fr.php.net/manual/fr/function.fgets.php>

8. <http://fr.php.net/manual/fr/html/ref.filesystem.html>

19.2. Un exemple

Nous allons réaliser un mini-site qui permet d'ajouter une ligne à la fois dans un fichier, tandis qu'une page permettra d'afficher le contenu du fichier.

1. Formulaire : fichier-01.php⁹

```
<html><body>
<form action="fichier-02.php">
Saisie : <input name="ligne">
<input type="submit">
</form>
</body></html>
```

il s'agit en fait d'une simple page HTML contenant un formulaire.

2. Traitement du formulaire (écriture dans le fichier) : fichier-02.php¹⁰

```
<html><body>
<?php
// ouverture du fichier "fichier-04.txt en mode "a" (append=ajout)
// l'identifiant renvoyé est $fp
// il sera utilisé pour l'écriture et la fermeture
$fp = fopen ("fichier-04.txt","a");
// écriture de la valeur reçue (plus une fin de ligne)
fputs ($fp,$ligne."\n");
// fermeture du fichier
fclose ($fp)
?>
Le fichier a été modifié.
</body></html>
```

Remarque

Pour que ce programme fonctionne il faut qu'il ait les droits en écriture sur le fichier fichier-04.txt.

Remarque

Ce programme ne devrait pas être laissé en accès libre sur le réseau : n'importe qui peut l'utiliser et remplir le disque en écrivant dans le fichier sans contrôle. (C'est pour cela que la version proposée en exemple est désactivée.)

3. Affichage du contenu du fichier :

- Première solution, avec la fonction « file¹¹ » : fichier-03.php¹²:

```
<html><body>
<?php
$f=file("fichier-04.txt");
for ($i=0;$i<count($f);$i++) {
    echo $f[$i]."<br />";
}
?>
</body></html>
```

- Deuxième solution, avec la fonction « fgets¹³ » : fichier-04.php¹⁴

```
<html><body>
<?php
$fp = fopen ("fichier-04.txt","r");
while ($l=fgets($fp,500)) {
    echo $l."<br />";
}
fclose($fp);
```

9. <http://grappa.univ-lille3.fr/polys/php/exemples/fichier-01.php>

10. <http://grappa.univ-lille3.fr/polys/php/exemples/fichier-02.php>

11. <http://fr.php.net/manual/fr/function.file.php>

12. <http://grappa.univ-lille3.fr/polys/php/exemples/fichier-03.php>

13. <http://fr.php.net/manual/fr/function.fgets.php>

14. <http://grappa.univ-lille3.fr/polys/php/exemples/fichier-04.php>

```
?>  
</body></html>
```

19.3. Exercice

Écrire un gestionnaire d'annuaire téléphonique.

Le minimum à atteindre :

1. Une page (formulaire) qui permet de saisir le nom, le prénom et le numéro de téléphone d'une personne.
2. L'appui sur la touche d'envoi enregistre les coordonnées dans un fichier.
3. Une page qui permette de présenter proprement la liste des contacts. Vous avez le choix sur le sens à donner au mot « proprement » : en tableau, en liste, etc. L'important est que ce soit lisible.
4. On doit pouvoir passer facilement d'une page à l'autre : soit des liens entre les pages, soit une page d'accueil.

Des améliorations possibles, qui ne seront prises en compte que ce qui précède est réalisé :

1. La liste affichée pourra être mise dans l'ordre alphabétique (voir la fonction « `sort`¹⁵ »).
2. On pourra offrir la possibilité de supprimer un contact.
3. On pourra offrir la possibilité de modifier un contact.

15. <http://fr.php.net/manual/fr/function.sort.php>

III. Corrigés des exercices

Chapitre 20. Premiers exercices d'algorithmique, corrigés

Vous trouverez les énoncés de ces exercices Chapitre 5.

1. Nombres impairs : impair.php¹

```
<html><body>
<?php
for ($i=1; $i<15000; $i=$i+2) {
    echo "$i ";
}
?>
</body></html>
```

2. Puniton : puniton.php²

```
<html><body>
<?php
for ($i=1; $i<=500; $i++) {
    echo "$i. Je dois faire des sauvegardes régulières...<br />\n";
}
?>
</body></html>
```

3. 30! : factorielle.php³

```
<html><body>
<?php
$f=1;
for ($i=2; $i<=30; $i++) {
    $f=$f*$i;
}
echo "30! = $f\n";
?>
</body></html>
```

4. Multiplication par 13 : multiplication.php⁴

```
<html><body>
<?php
for ($i=0; $i<14; $i++) {
    $r=$i*13;
    echo "$i*13=$r<br />\n";
}
?>
</body></html>
```

Mais aussi multiplicationbis.php⁵

```
<html><body>
<table>
<?php
for ($i=0; $i<14; $i++) {
    $r=$i*13;
    echo "<tr><td align=\"right\">$i</td><td>*</td><td>13</td><td>=</td>";
    echo "<td align=\"right\">$r</td></tr>\n";
}
?>
</table>
</body></html>
```

5. 4 lignes : 4lignes.php⁶

```
<html><body>
<?php
for ($l=1 ; $l<=4 ; $l++) {
    for ($n=1 ; $n<=13 ; $n++) {
        echo $n;
    }
}
```

1. <http://grappa.univ-lille3.fr/polys/php/exemples/impair.php>
2. <http://grappa.univ-lille3.fr/polys/php/exemples/puniton.php>
3. <http://grappa.univ-lille3.fr/polys/php/exemples/factorielle.php>
4. <http://grappa.univ-lille3.fr/polys/php/exemples/multiplication.php>
5. <http://grappa.univ-lille3.fr/polys/php/exemples/multiplicationbis.php>
6. <http://grappa.univ-lille3.fr/polys/php/exemples/4lignes.php>

```

    }
    echo "<br />";
}
?>
</body></html>

```

6. *Table de multiplication totale* : `multiplicationtotale.php`⁷

```

<html><body>
<table border="border">
<?php
// 1ère ligne
echo "<tr><td></td>";
for ($c=0;$c<13;$c++) {
    echo "<th>$c</th>";
}
echo "</tr>\n";
// toutes les lignes
for ($l=0;$l<13;$l++) {
    // 1 ligne
    echo "<tr><th>$l</th>";
    for ($c=0;$c<13;$c++) {
        $r=$c*$l;
        echo "<td align=\"right\">$r</td>";
    }
    echo "</tr>\n";
}
?>
</table>
</body></html>

```

7. *Table des factorielles* : `tablefact.php`⁸

```

<html><body>
<?php
// fonction factorielle
function fact($n) {
    $f=1; // initialisation de la variable d'accumulation
    for ($i=1;$i<=$n;$i++) {
        $f *= $i;
    }
    return $f;
}
for ($x=0;$x<=10;$x++) {
    echo "$x ! = ".fact($x)."<br />\n";
}
?>
</body></html>

```

8. *Demi-sapin* : `demisapin.php`⁹

```

<html><body>
<?php
function triangle($n) {
    for ($l=1 ; $l<=$n ; $l++) {
        for ($i=1 ; $i<=$l ; $i++) {
            echo "*";
        }
        echo "<br />";
    }
}

for ($n=2;$n<=10;$n+=2) triangle($n);
?>
</body></html>;

```

9. *Coefficients du binôme* : `binome.php`¹⁰

```

<html><body>
<?php

```

7. <http://grappa.univ-lille3.fr/polys/php/exemples/multiplicationtotale.php>

8. <http://grappa.univ-lille3.fr/polys/php/exemples/tablefact.php>

9. <http://grappa.univ-lille3.fr/polys/php/exemples/demisapin.php>

10. <http://grappa.univ-lille3.fr/polys/php/exemples/binome.php>

```

function fact($n) {
    $f=1;
    for ($i=1;$i<=$n;$i++) {
        $f*=$i;
    }
    return $f;
}
function cnp($n,$p) {
    return fact($n)/(fact($p)*fact($n-$p));
}
function binome ($n) {
    echo "<tr>";
    for ($i=0;$i<=$n;$i++) {
        echo "<td align=\"right\">".cnp($n,$i). "</td>";
    }
    echo "</tr>\n";
}
echo "<table>\n";
for ($i=0;$i<=20;$i++) {
    binome ($i);
}
echo "</table>\n";
?>
</body></html>

```

10. *Fibonnacci* : fibonnacci.php¹¹

```

<html><body>
<?php
function fibonnacci($n) {
    if ($n<2)
        return $n;
    $fmoins1=1;
    $fmoins2=0;
    for ($i=2;$i<=$n;$i++) {
        $f=$fmoins1+$fmoins2;
        $fmoins2=$fmoins1;
        $fmoins1=$f;
    }
    return $f;
}
for ($i=0;$i<50;$i++) {
    echo "$i : ".fibonnacci($i). "<br />\n";
}
?>
</body></html>

```

On peut aussi une version récursive, plus belle, sans doute plus facile à programmer, mais beaucoup moins efficace (pourquoi ?) : fibonnacci-rec.php¹²

```

<html><body>
<?php
function fibonnacci($n) {
    if ($n<2)
        return $n;
    return (fibonnacci($n-1)+fibonnacci($n-2));
}
for ($i=0;$i<20;$i++) {
    echo "$i : ".fibonnacci($i). "<br />\n";
}
?>
</body></html>

```

11. *Fibonnacci, le retour* : fibonnacci-retour.php¹³

```

<html><body>
<?php
function fibonnacci($n) {
    ... comme l'exercice précédent ...
}

```

11. <http://grappa.univ-lille3.fr/polys/php/exemples/fibonnacci.php>

12. <http://grappa.univ-lille3.fr/polys/php/exemples/fibonnacci-rec.php>

13. <http://grappa.univ-lille3.fr/polys/php/exemples/fibonnacci-retour.php>

```
$or=(sqrt(5)-1)/2;
echo "<table border=\"border\"><tr><th>ordre</th><th>rapport</th>";
echo "<th>différence</th></tr>\n";
for ($i=0;$i<50;$i++) {
    $r=fibonacci($i)/fibonacci($i+1);
    echo "<tr><td>$i</td><td>$r</td><td>".($r-$or). "</td></tr>\n";
}
?>
</body></html>
```

12. Afficher un cadre 10x10.

a. avec <PRE> cadre-pre.php¹⁴

```
<html><body><pre><tt>
<?php
$k="*";
$MAX=10;
for ($i=1;$i<=$MAX;$i++) {
    echo $k;
}
echo "\n";
for ($i=2;$i<$MAX;$i++) {
    echo $k;
    for ($j=2;$j<$MAX;$j++) {
        echo " ";
    }
    echo $k."\n";
}
for ($i=1;$i<=$MAX;$i++) {
    echo $k;
}
echo "\n";
?>
</tt></pre></body></html>
```

b. avec un tableau : cadre-table.php¹⁵

```
<html><body>
<blockquote><table border="border"><tr>
<?php
$k="*";
$max=10;
for ($i=1;$i<=$max;$i++) {
    echo "\t<td>$k</td>\n";
}
echo "</tr>\n";
for ($i=2;$i<$max;$i++) {
    echo "<tr>\n\t<td>$k</td>\n";
    for ($j=2;$j<$max;$j++) {
        echo "\t<td></td>\n";
    }
    echo "\t<td>$k</td>\n</tr>\n";
}
echo "<tr>\n";
for ($i=1;$i<=$max;$i++) {
    echo "\t<td>$k</td>\n";
}
?>
</tr>
</table></blockquote>
</body></html>
```

13. Tableau d'additions à compléter :

a. avec le total fixe : tableauadditionsfixe.php¹⁶

```
<body><html>
<h2>additions à compléter, total fixe</h2>
<table>
```

14. <http://grappa.univ-lille3.fr/polys/php/exemples/cadre-pre.php>

15. <http://grappa.univ-lille3.fr/polys/php/exemples/cadre-table.php>

16. <http://grappa.univ-lille3.fr/polys/php/exemples/tableauadditionsfixe.php>

```

<?php
$nombre=10; // nombre de lignes
$total=100; // total à atteindre
for ($i=0;$i<$nombre;$i++) {
    $connu=rand(1,$total); // le nombre affiché
    echo "\t<tr><th>n°".($i+1)." : </td>";
    echo "<td align=\"right\">$connu</td>";
    echo "<td>+</td>";
    echo "<td align=\"center\"> _____ = </td>";
    echo "<td align=\"right\">$total</td></tr>\n";
}
?>
</table>
</body></html>

```

b. avec le total variable : `tableauadditionsvariable.php`¹⁷

```

<body><html>
<h2>additions à compléter, total variable</h2>
<table>
<?php
$nombre=10; // nombre de lignes
$min=80; // borne inférieure des totaux possibles
$max=120; // borne supérieure des totaux possibles
for ($i=0;$i<$nombre;$i++) {
    $total=rand($min,$max);
    $connu=rand(1,$total);
    echo "\t<tr><th>n°".($i+1)." : </td>";
    echo "<td align=\"right\">$connu</td>";
    echo "<td>+</td>";
    echo "<td align=\"center\"> _____ = </td>";
    echo "<td align=\"right\">$total</td></tr>\n";
}
?>
</table>
</body></html>

```

17. <http://grappa.univ-lille3.fr/polys/php/exemples/tableauadditionsvariable.php>

Chapitre 21. Premiers formulaires en PHP, corrigés

Vous trouverez les énoncés de ces exercices Chapitre 7.

1. Bonjour machin, vous avez xx ans...

Le formulaire : traitement-01.html¹

```
<html><body>
<form method="get" action="traitement-01.php">
<table>
<tr>
  <td align="right">Votre nom</td>
  <td><input name="nom" /></td>
</tr><tr>
  <td align="right">Votre âge</td>
  <td><input name="age" /></td>
</tr><tr>
  <td align="center" colspan="2"><input type="submit" value="Envoyer" />
</tr>
</table>
</form>
</body></html>
```

Son traitement : traitement-01.php²

```
<html><body>
<?php
echo "Bonjour $nom, vous avez $age ans.\n";
?>
</body></html>
```

2. Montpellier.

Le formulaire : traitement-02.html³

```
<html><body><form method="get" action="traitement-02.php">
<table>
<tr>
  <td align="right">Distance</td>
  <td><input name="distance" /></td>
</tr><tr>
  <td align="right">Heures/Jour</td>
  <td><input name="heures" /></td>
</tr><tr>
  <td align="right">Jours</td>
  <td><input name="jours" /></td>
</tr><tr>
  <td align="center" colspan="2"><input type="submit" value="Envoyer" />
</tr>
</table>
</form></body></html>
```

Son traitement : traitement-02.php⁴

```
<html><body>
<?php
echo "Le bébé a parcouru $distance km, "
    ."à raison de $heures heures par jour, "
    ."pendant $jours jours.<br />\n"
    ."Sa vitesse est donc de "
    .($distance/$heures/$jours)." km/h.\n";
?>
</body></html>
```

3. Bonjour M. Truc.

Le formulaire : traitement-03.html⁵

```
<html><body><form method="get" action="traitement-03.php">
<table>
```

-
1. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-01.html>
 2. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-01.php>
 3. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-02.html>
 4. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-02.php>
 5. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-03.html>

```
<tr>
  <td align="right">sexe</td>
  <td>
    <input name="sexe" type="radio" value="M" checked="checked" />masculin
    <input name="sexe" type="radio" value="F" />féminin
  </td>
</tr><tr>
  <td align="right">nom</td>
  <td><input name="nom" /></td>
</tr><tr>
  <td align="center" colspan="2"><input type="submit" value="Envoyer" />
</td>
</tr>
</table>
</form></body></html>
```

Son traitement : traitement-03.php⁶

```
<html><body>
<?php
if ($sexe=="M") {
  echo "Bonjour Monsieur $nom.";
} else {
  echo "Bonjour Madame $nom.";
}
?>
</body></html>
```

4. Permis de chasse :

Le formulaire : traitement-04.html⁷

```
<html><body><form method="get" action="traitement-04.php">
<table><tr>
  <td align="right">poules</td>
  <td><input name="poules" /></td>
</tr><tr>
  <td align="right">chiens</td>
  <td><input name="chiens" /></td>
</tr><tr>
  <td align="right">vaches</td>
  <td><input name="vaches" /></td>
</tr><tr>
  <td align="right">amis</td>
  <td><input name="amis" /></td>
</tr><tr>
  <td align="center" colspan="2"><input type="submit" value="envoyer" />
</td>
</tr></table></form>
</body></html>
```

Son traitement : traitement-04.php⁸

```
<html><body>
<?php
echo "Vous avez tué:\n";
echo "<ul>\n";
echo "  <li>$poules poules,</li>\n";
echo "  <li>$chiens chiens,</li>\n";
echo "  <li>$vaches vaches,</li>\n";
echo "  <li>$amis amis.</li>\n";
echo "</ul>\n";
$perdu=$poules*1+$chiens*3+$vaches*5+$amis*10;
echo "Vous avez donc perdu: "
  ."$poules*1 + $chiens*3 + $vaches*5 + $amis*10=$perdu points."
  ."\n<br />\n";
$permis=floor($perdu/100);
if ($permis<1) {
  echo "<b>Votre permis est encore valide, mais attention!</b>\n";
} else {
  echo "<b>Vous avez épuisé $permis permis, vous devez payer "
    .("$permis*1000)." francs.\n". "</b>";
}
```

6. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-03.php>

7. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-04.html>

8. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-04.php>

```

}
?>
</body></html>

```

5. Calculatrice, le formulaire et son traitement : traitement-05.php⁹

```

<html><body>
avec liste déroulante:<br />
<form method="get" action="traitement-05.php">
<input name="a" size="5" />
<select name="par">
  <option value="+">+</option>
  <option value="-">-</option>
  <option value="*">*</option>
  <option value="/">/</option>
</select>
<input name="b" size="5" />
<br />
<input type="submit" value="envoyer" />
</form><hr />
<!-- ----- -->
avec boutons radios:<br />
<form method="get" action="traitement-05.php">
<table><tr>
  <td>
    <input name="a" size="5" />
  </td><td>
    <input type="radio" name="par" value="+" checked="checked" />+<br />
    <input type="radio" name="par" value="-" />-<br />
    <input type="radio" name="par" value="*" />*<br />
    <input type="radio" name="par" value="/" />/
  </td><td>
    <input name="b" size="5">
  </td>
</tr><tr>
  <td colspan="3" align="center">
    <input type="submit" value="envoyer" />
  </td>
</tr></table>
</form><hr />
<!-- ----- -->
<?php
if (isset($par)) {
  echo "le résultat du calcul précédent est: $a$par$b=";
  if ($par=="+") {
    echo $a+$b;
  } elseif ($par=="-") {
    echo $a-$b;
  } elseif ($par=="*") {
    echo $a*$b;
  } else {
    echo $a/$b;
  }
}
?>
</body></html>

```

6. Table de multiplication à la demande : traitement-06.php¹⁰

```

<html><head>
<style type="text/css">
  td { text-align:right; }
</style>
</head><body>
<form action="traitement-06.php">
choisissez votre table : <select name="table">
<?php
for ($i=1;$i<=20;$i++) {
  echo "<option";
  if ($table==$i) echo " selected=\"selected\"";

```

9. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-05.php>

10. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-06.php>

```

        echo ">$i</option>\n";
    }
    ?>
</select><br />
choisissez le nombre de lignes : <select name="lignes">
<?php
for ($i=10;$i<=20;$i++) {
    echo "<option";
    if ($lignes==$i) echo " selected=\"selected\"";
    echo ">$i</option>\n";
}
?>
</select><br />
<input type="submit" value="afficher la table choisie" />
</form>
<p />
<?php
if (isset($table)) { // si on utilisé le formulaire...
    echo "<table>\n";
    for ($l=0;$l<=$lignes;$l++) { // afficher les lignes de la table
        echo "<tr><td>$l</td><td>x</td><td>$table</td><td>=</td><td>"
            . ($l*$table) . "</td></tr>\n";
    }
    echo "</table>\n";
}
?>
</body></html>

```

7. Mots de passe : traitement-07.php¹¹

```

<html><body>
<form action="exocontr.php">
tapez votre mot de passe :<input type="password" name="pass" />
<input type="submit" /></form>
<?php
if (isset($pass)) {
    if ($pass=="un") { // premier mot de passe autorisé
        echo "<h2>bienvenue. vous êtes l'utilisateur principal.</h2>";
    } elseif ($pass=="deux") { // deuxième mot de passe autorisé
        echo "<h4>bienvenue. vous êtes un utilisateur de base.</h4>";
    } else { // mot de passe non reconnu
        echo "<h1 style=\"color:red;background:yellow;text-align:center;\">"
            . "accès interdit</h1>";
    }
}
?>
</body></html>

```

8. Compléter les additions : addition.php¹²

```

<html><body>
remplis la case avec le bon nombre pour que le calcul soit correct:
<?php
$ok=0;
if (isset($a)) {
    echo "tu viens d'essayer $a+$c=$b.<br />\n";
    if ($a+$c==$b) {
        echo "c'est exact. bravo!";
        $ok=1;
    } else {
        echo "c'est faux, essaie encore.";
    }
} else {
    $b=rand(50,500);
    $a=rand(0,$b);
}
if ($ok==0) {
    ?>
<form method="post">
<?php echo $a;?><input size="4" name="c" />=<?php echo $b;?>

```

11. <http://grappa.univ-lille3.fr/polys/php/exemples/traitement-07.php>

12. <http://grappa.univ-lille3.fr/polys/php/exemples/addition.php>

```

<input type="submit" value="verifier" />
<input type="hidden" name="a" value="<?php echo $a;?>" />
<input type="hidden" name="b" value="<?php echo $b;?>" />
</form>
<?php
}
echo "<form method=\"post\">"
    . "<input type=\"submit\" value=\"recommencer\" /></form>";
?>
</body></html>

```

9. L'ordinateur choisit, vous devinez : devinette01.php¹³

```

<html><body>
<h1>Devinette n°1</h1>
<h2>Vous devez deviner un nombre que le programme a choisi</h2>
<?php
// si le jeu est commencé
if (isset($limite)){
    // si c'est la première fois
    if (!isset($guess)) {
        // choisir le nombre à deviner
        $guess=rand(0,$limite);
        // c'est le premier essai
        $nbessai=1;
    }
    echo "Le nombre que j'ai choisi est entre 0 et $limite.<br />\n";
    // si il y a une proposition à traiter
    if (isset($proposition)){
        $proposition=$proposition*1;
        echo "Vous venez de proposer $proposition: ";
        // si le joueur a trouvé
        if ($proposition==$guess) {
            echo "\n<h3>Bravo! Trouvé en "
                . ($nbessai-1) . " essais</h3>\n";
            echo "<form method=\"post\">\n";
            echo "<input type=\"submit\" value=\"Recommencer\" />\n";
            echo "</form>\n";
        } else { // $proposition!= $guess : pas trouvé
            if ($proposition<$guess) { // proposition trop petite
                echo "Trop petit.<p />\n";
            }
            if ($proposition>$guess) { // proposition trop grande
                echo "Trop grand.<p />\n";
            }
        } // fin de if($proposition==$guess)
    }
    if (isset($guess)&&($proposition!= $guess)) {
        echo "C'est votre ".$nbessai."<sup>". ($nbessai==1?"er":"ème")
            . "</sup> essai.<p />\n";
    }
?>
<table><tr><td>
<form method="post">
    que proposez-vous? <input name="proposition" size="4" />
    <input type="hidden" name="guess" value="<?php echo $guess;?>" />
    <input type="hidden" name="limite" value="<?php echo $limite;?>" />
    <input type="hidden" name="nbessai" value="<?php echo $nbessai+1;?>" />
    <input type="submit" value="Tester votre proposition" />
</form></td><td><form method="post">
    <input type="submit" value="Abandonner" />
</form>
</td></tr></table>
<?php
    } // fin de if(isset($guess)&&($proposition!= $guess))
} else { // on commence un jeu
?>
<form method="post">
    Choisissez la limite (donc la difficulté):
<select name="limite">
    <option selected="selected">10</option>

```

13. <http://grappa.univ-lille3.fr/polys/php/exemples/devinette01.php>

```

        <option>100</option>
        <option>1000</option>
        <option>10000</option>
    </select>
    <input type="submit" value="commencer" />
</form>
<?php
}
?>
</body></html>

```

10. Vous choisissez, l'ordinateur devine : devinette01.php¹⁴

```

<html><body>
<h1>Devinette n°2</h1>
<h2>Le programme va deviner le nombre que vous avez choisi</h2>
<?php
// si c'est la première exécution
if (!isset($_POST["min"])) {
    // demander à sélectionner la limite
    ?>
    Sélectionnez la limite supérieure, puis quand vous avez
    choisi le nombre non nul que je dois deviner (sans me le dire)
    cliquez sur <em>Commencer</em>.
    <p />
    <form method="post">
    Limite supérieure? <select name="max">
    <option selected="selected">10
        <option>100</option>
        <option>1000</option>
        <option>10000</option>
    </select>
    <input type="hidden" name="min" value="0" />
    <input type="hidden" name="nbessai" value="0" />
    <input type="submit" value="Commencer" />
    </form>
    <?php
    } else {
        $nbessai=$_GET["nbessai"]+1;
        if ($_POST["result"]=="=") { // si le nombre est trouvé
            echo "<h3>Gagné en ".$_POST["nbessai"]." essais.</h3>\n";
        } elseif ($_POST["result"]=="+") { // trop grand, on réduit la borne supérieure
            $max=$_POST["test"];
        } elseif ($_POST["result"]=="-") { // trop petit, on réduit la borne inférieure
            $min=$_POST["test"];
        }
        // si on n'a pas trouvé
        if ($_POST["result"]!="=") {
            if ($_POST["min"]==$_POST["max"]) {
                // les bornes sont égales : il y a eu triche
                echo "<h3>Vous avez triché</h3>\n";
            } else { // une nouvelle proposition
                $test=floor(($_POST["min"]+$_POST["max"])/2);
                if ($_POST["max"]-$$_POST["min"]==1) {
                    $test=$_POST["max"];
                }
                echo "<table><tr><td align=\"center\" colspan=\"3\">";
                echo "Mon ".$_POST["nbessai"]."<sup>".($_POST["nbessai"]==1?"er":"ème")."</sup>";
                echo " est le nombre <b>".$_POST["test"]</b></td></tr>\n";
            }
        }
        ?>

        <td><form method="post">
            <input type="hidden" name="result" value="-" />
            <input type="hidden" name="test" value="<?php echo $_POST["test"];?>" />
            <input type="hidden" name="min" value="<?php echo $_POST["min"];?>" />
            <input type="hidden" name="max" value="<?php echo $_POST["max"];?>" />
            <input type="hidden" name="nbessai" value="<?php echo $_POST["nbessai"];?>" />
            <input type="submit" value="Trop petit" />
        </form></td>
        <td><form method="post">

```

14. <http://grappa.univ-lille3.fr/polys/php/exemples/devinette01.php>

```

        <input type="hidden" name="result" value="" />
        <input type="hidden" name="test" value="<?php echo $_POST["test"];?>" />
        <input type="hidden" name="min" value="<?php echo $_POST["min"];?>" />
        <input type="hidden" name="max" value="<?php echo $_POST["max"];?>" />
        <input type="hidden" name="nbessai" value="<?php echo $_POST["nbessai"];?>" />
        <input type="submit" value=" G A G N É ! ! " />
    </form></td>
<td><form method="post">
    <input type="hidden" name="test" value="<?php echo $_POST["test"];?>" />
    <input type="hidden" name="result" value="+" />
    <input type="hidden" name="min" value="<?php echo $_POST["min"];?>" />
    <input type="hidden" name="max" value="<?php echo $_POST["max"];?>" />
    <input type="hidden" name="nbessai" value="<?php echo $_POST["nbessai"];?>" />
    <input type="submit" value="Trop grand" />
</form></td>
</tr></table>
<?php
    }
}
}
if (isset($_POST["min"])){
    ?>
    <form method="post">
    <input type="submit" value="Recommencer" />
    </form>
    <?php
}
?>
</body></html>

```

11. Le pipotron : pipotron.php¹⁵

```

<html><body>
<?php
if (isset($_POST["pipo1"])) {
echo stripslashes("<hr />  " . $_POST["$pipo1"] . " " . $_POST["$pipo2"] . " " . $_POST["$pipo3"]
    . $_POST["$pipo4"] . " $ " . $_POST["$pipo5"] . " " . $_POST["$pipo6"] . " " . $_POST["$pipo7"] . "
    " . "<hr />");
}
?>

<form method="post">
<select name="pipo1">
    <option>Avec</option>
    <option>Considérant</option>
    <option>Où que nous mène</option>
    <option>Eu égard à</option>
    <option>Vu</option>
    <option>En ce qui concerne</option>
    <option>Dans le cas particulier de</option>
    <option>Quelle que soit</option>
    <option>Du fait de</option>
    <option>Tant que durera</option>
</select>
<select name="pipo2">
    <option>la situation</option>
    <option>la conjoncture</option>
    <option>la crise</option>
    <option>l'inertie</option>
    <option>l'impasse</option>
    <option>l'extrémité</option>
    <option>la dégradation des mœurs</option>
    <option>la sinistrose</option>
    <option>la dualité de la situation</option>
    <option>la baisse de confiance</option>
</select>
<select name="pipo3">
    <option>présente</option>
    <option>actuelle</option>
    <option>qui nous occupe</option>

```

15. <http://grappa.univ-lille3.fr/polys/php/exemples/pipotron.php>

```
<option>qui est la nôtre</option>
<option>induite</option>
<option>conjoncturelle</option>
<option>contemporaine</option>
<option>de cette fin de siècle</option>
<option>de la société</option>
<option>de ces derniers temps</option>
</select>
<select name="pipo4">
  <option>il convient de</option>
  <option>il faut</option>
  <option>on se doit de</option>
  <option>il est préférable de</option>
  <option>il serait intéressant de</option>
  <option>il ne faut pas négliger de</option>
  <option>on ne peut se passer de</option>
  <option>il est nécessaire de</option>
  <option>il serait bon de</option>
  <option>il faut de toute urgence</option>
</select>
<select name="pipo5">
  <option>étudier</option>
  <option>examiner</option>
  <option>ne pas négliger</option>
  <option>prendre en considération</option>
  <option>anticiper</option>
  <option>imaginer</option>
  <option>se préoccuper de</option>
  <option>s'intéresser à</option>
  <option>avoir à l'esprit</option>
  <option>se remémorer</option>
</select>
<select name="pipo6">
  <option>toutes les</option>
  <option>chacune des</option>
  <option>la majorité des</option>
  <option>toutes les</option>
  <option>l'ensemble des</option>
  <option>la somme des</option>
  <option>la totalité des</option>
  <option>la globalité des</option>
  <option>toutes les</option>
  <option>certaines</option>
</select>
<select name="pipo7">
  <option>solutions</option>
  <option>issues</option>
  <option>problématiques</option>
  <option>voies</option>
  <option>alternatives</option>
  <option>solutions</option>
  <option>issues</option>
  <option>problématiques</option>
  <option>voies</option>
  <option>alternatives</option>
</select>
<select name="pipo8">
  <option>envisageables</option>
  <option>possibles</option>
  <option>déjà en notre possession</option>
  <option>s'offrant à nous</option>
  <option>de bon sens</option>
  <option>envisageables</option>
  <option>possibles</option>
  <option>déjà en notre possession</option>
  <option>s'offrant à nous</option>
  <option>de bon sens</option>
</select>
<input type="submit" />
</form>
</body></html>
```

Chapitre 22. Chaînes de caractères, corrigé

Vous trouverez l'énoncé de cet exercice Chapitre 8.

identite.php¹:

```
<html><body>
<h2>premiers essais de contrôle d'identité</h2>
<?php
if ((trim(strtolower($nom))==trim(strtolower("Atte"))
    &&(trim(strtolower($prenom))==trim(strtolower("Tom")))
    &&(trim(strtolower($password))==trim(strtolower("Rouge")))) {
    echo "Bienvenue chez vous, Tom Atte...<br />\n";
} else {
    // sortie provisoire du mode php (pour écrire du texte html pur)
?>
    <hr />
    Veuillez vous identifier SVP:<p />
    <form method="post" action="identite.php" />
    Nom: <input name="nom" /><br />
    Prénom: <input name="prenom" /><br />
    Mot de passe: <input type="password" name="password" /><br />
    <input type="submit" /><br />
    </form>
    <hr />
<?php
    // retour dans le mode php (pour fermer le "else")
}
?>
</body></html>
```

1. <http://grappa.univ-lille3.fr/polys/php/exemples/identite.php>

Chapitre 23. Exercices sur les tableaux, corrigés

Vous trouverez les énoncés de ces exercices Chapitre 9.

1. Écrire un programme qui lit ce fichier pour construire une page web contenant une liste de liens hypertextes : `tableau-06.php`¹

```
<html><head>
<title>Utilisation des tableaux-6</title>
</head><body><ul>
<?php
$liens=file("tableau-06.txt");
for ($i=0;$i<count($liens);$i++) {
    echo "<li><a href=\"". $liens[$i]. "\">". $liens[$i]. "</a></li>\n";
}
?>
</ul></body></html>
```

2. Même exercice, mais cette fois chaque ligne comprend aussi une description du site pointé (la séparation étant assurée par la chaîne « `***` ») : `tableau-07.php`²

```
<html><body><ul>
<?php
$liens=file("tableau-07.txt");
for ($i=0;$i<count($liens);$i++) {
    $parties=explode("***", $liens[$i]);
    echo "<li><a href=\"". $parties[0]. "\">". $parties[1]. "</a></li>\n";
}
?>
</ul></body></html>
```

3. Même exercice, mais cette fois chaque description et l'adresse correspondante sont sur deux lignes consécutives : `tableau-08.php`³

```
<html><body><ul>
<?php
$liens=file("tableau-08.txt");
for ($i=0;$i<count($liens);$i=$i+2) {
    echo "<li><a href=\"". $liens[$i+1]. "\">". $liens[$i]. "</a></li>\n";
}
?>
</ul></body></html>
```

4. On donne une liste de personnes dont chaque ligne est composée, dans l'ordre, d'un identifiant (un nombre), un prénom, un nom, et un mot de passe. Écrire une page web qui donne à sélectionner une des personnes et qui affiche ensuite son mot de passe : `tableau-09.php`⁴

```
<html><body>
<?php
$personnes=file("tableau-09.txt");
for ($i=0;$i<count($personnes);$i++) {
    $parties=explode(";", $personnes[$i]);
    $code=$parties[0];
    $prenom[$code]=$parties[1];
    $nom[$code]=$parties[2];
    $password[$code]=$parties[3];
}
?>
<form action="tableau-09.php" method="post">
<select name="qui">
<?php
reset($nom);
while (list($code,$n)=each($nom)) {
    echo "<option value=\"".$code.">". $prenom[$code]. " ".strtoupper($n). "</option>\n";
}
?>
</select>
```

1. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-06.php>
2. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-07.php>
3. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-08.php>
4. <http://grappa.univ-lille3.fr/polys/php/exemples/tableau-09.php>

```
<input type="submit" value="Quel mot de passe ?" />
</form>
<?php
if (isset($qui)) {
    echo "<br /><hr />Le mot de passe de ".$prenom[$qui]." "
        .strtoupper($nom[$qui])." est ".$password[$qui];
}
?>
</body></html>
```

Chapitre 24. Exercices sur les fichiers, corrigés

Vous trouverez les énoncés de ces exercices Chapitre 19.

1. `index.php`¹ : page d'accueil. (Il s'agit en fait d'une page en HTML pur, sans PHP.)

```
<html><<body>>
<h1>Premier projet en Licence AES</h1>
<h2>Accueil</h2>
<ul>
<li><a href="liste.php">Liste</a> des contacts
    (ou une autre <a href="liste2.php">présentation</a>
    des contacts)</li>
<li>Saisir un <a href="saisie.php">nouveau contact</a></li>
</ul>
</body>></HTML>
```

2. `liste.php`² : liste des contacts.

```
<html><<body>>
<h1>Premier projet en Licence AES</h1>
<h2>Liste des contacts</h2>
<ul>
<?php
// lecture du fichier
$f=file("liste.txt");
// pour trier sur le nom
sort($f);
// parcours de la liste
for ($i=0;$i<count($f);$i++) {
    // découpage d'1 ligne suivant les ";"
    $l=explode(";", $f[$i]);
    // affichage après découpage
    echo "<li>Nom: $l[0]<br />Prénom: $l[1]<br />Téléphone: $l[2]\n";
}
?>
</ul>
<a href="index.php">accueil</a>
</body>></html>
```

3. `liste2.php`³ : une autre liste des contacts

```
<html><<body>>
<h1>Premier projet en Licence AES</h1>
<h2>Liste des contacts</h2>
<table border="border">
<tr><th>Nom</th><th>Prénom</th><th>Téléphone</th></tr>
<?php
// lecture du fichier
$f=file("liste.txt");
// pour trier sur le nom
sort($f);
// parcours de la liste
for ($i=0;$i<count($f);$i++) {
    // découpage d'1 ligne suivant les ";"
    $l=explode(";", $f[$i]);
    // affichage après découpage
    echo "<tr><td>$l[0]</td><td>$l[1]</td><td>$l[2]</td></tr>\n";
}
?>
</table>
<a href="index.php">accueil</a>
</body>></html>
```

4. `saisie.php`⁴ : formulaire de saisie. (Il s'agit en fait d'une page en HTML pur, sans PHP.)

```
<html><<body>>
<h1>Premier projet en Licence AES</h1>
<h2>Saisie d'un nouveau contact</h2>
```

1. <http://grappa.univ-lille3.fr/polys/php/exemples/index.php>
2. <http://grappa.univ-lille3.fr/polys/php/exemples/liste.php>
3. <http://grappa.univ-lille3.fr/polys/php/exemples/liste2.php>
4. <http://grappa.univ-lille3.fr/polys/php/exemples/saisie.php>

```
<form action="enregistrer.php">
  Nom : <input name="nom" /><br />
  Prénom : <input name="prenom" /><br />
  Téléphone : <input name="telephone" /><br />
  <input type="submit" />
</form>
<a href="index.php">accueil</a>
</<body>></html>
```

5. enregistrer.php⁵ : Enregistrer un nouveau contact

```
<html><<body>>
<h1>Premier projet en Licence AES</h1>
<h2>Enregistrement d'un nouveau contact</h2>
<?php
// ouverture du fichier en ajout
$fp=fopen("liste.txt", "a");
// écriture de la nouvelle ligne
fputs($fp, "$nom; $prenom; $telephone\n");
// fermeture du fichier
fclose ($fp);
// Affichage pour vérification
echo "Le nouveau contact a été enregistré :\n"
  . "<ul></li>\n"
  . "<li>Nom: $nom</li>\n"
  . "<li>Prénom: $prenom</li>\n"
  . "<li>Téléphone: $telephone</li>\n"
  . "</ul>\n";
?>
<a href="index.php">accueil</a>
</<body>></html>
```

5. <http://grappa.univ-lille3.fr/polys/php/exemples/enregistrer.php>

Chapitre 25. Exploration d'un répertoire, corrigés

Vous trouverez les énoncés de ces exercices Chapitre 17.

1. simple.php¹:

```
<html><body>
<h2>On affiche les images trouvées</h2>
<?php
$dir = opendir(".");
while($file = readdir($dir)) {
    if (substr($file,-4)==".jpg") {
        echo "<img src=\""$file\"" />\n";
    }
}
closedir($dir);
?>
</body></html>
```

2. thumb.php²:

```
<html><body>
<h2>On affiche les onglets, qui font liens vers les vraies images</h2>
<?php
$dir = opendir("mini");
while($file = readdir($dir)) {
    if (substr($file,-4)==".jpg") {
        echo "<a href=\""$file\""><img src=\"mini/"$file\"" /></a>\n";
    }
}
closedir($dir);
?>
</body></html>
```

3. thumbplus.php³:

```
<html><body>
<h2>On affiche les onglets, qui font liens vers
des pages contenant les vraies images dans un cadre</h2>
<?php
$dir = opendir("mini");
while($file = readdir($dir)) {
    if (substr($file,-4)==".jpg") {
        echo "<a href=\"image.php?image=$file\"">
            . "<img src=\"mini/"$file\"" /></a>\n";
    }
}
closedir($dir);
?>
</body></html>
```

4. parfait.php⁴:

```
<html><body>
<h2>On affiche les onglets en tableau régulier.
Ils font liens vers des pages contenant les vraies images
dans un cadre</h2>
<table border="border">
<?php
$nb=4;
$i=0;
$dir = opendir("mini");
while($file = readdir($dir)) {
    if (substr($file,-4)==".jpg") {
        if ($i==0) {
            echo "<TR>";
        }
        echo "<td><a href=\"image.php?image=$file\"">
            . "<img src=\"mini/"$file\"" /></a></td>\n";
    }
}
```

1. <http://grappa.univ-lille3.fr/polys/php/exemples/simple.php>
2. <http://grappa.univ-lille3.fr/polys/php/exemples/thumb.php>
3. <http://grappa.univ-lille3.fr/polys/php/exemples/thumbplus.php>
4. <http://grappa.univ-lille3.fr/polys/php/exemples/parfait.php>

```
        $i++;  
        if ($i>=$nb) {  
            echo "</tr>\n";  
            $i=0;  
        }  
    }  
}  
closedir($dir);  
?>  
</table>  
</body></html>
```

Chapitre 26. News..., corrigé

Vous trouverez les énoncés de ces exercices Chapitre 18.

26.1. Accueil

index.php¹:

```
<html><body>
<h1>News...</h1>
<h2>Menu</h2>
<ul>
<li>liste des <a href="listenoms.php">noms</a> de fichiers de
nouvelles</li>
<li>liste des <a href="listeliens.php">liens</a> vers les fichiers de
nouvelles</li>
<li>affichage de <a href="affichetout.php">toutes les nouvelles</a></li>
<li>affichage des <a href="affichetitre.php">titres</a> des nouvelles</li>
<li>affichage des <a href="affichetitrelien.php">titres faisant</a>
vers les nouvelles</li>
<li>affichage de <a href="affichetout_br.php">toutes les nouvelles
proprement</a></li>
<li><a href="sujet.php">choix du sujet</a></li>
</ul>
</body></html>
```

26.2. Liste des noms de fichiers de nouvelles

listenoms.php²:

```
<html><body>
<h1>News...</h1>
<h2>Liste des noms de fichiers de nouvelles</h2>
<ul>
<?php
$dir = opendir(".");
while($file = readdir($dir)) {
    if (substr($file,-4)==".jnl") {
        echo "<li>$file;</li>\n";
    }
}
closedir($dir);
?>
</ul>
</body></html>
```

26.3. Liste des liens vers les fichiers de nouvelles

listeliens.php³:

```
<html><body>
<h1>News...</h1>
<h2>Liste des liens vers les fichiers de nouvelles</h2>
<ul>
<?php
$dir = opendir(".");
while($file = readdir($dir)) {
```

1. <http://grappa.univ-lille3.fr/polys/php/exemples/news/index.php>
2. <http://grappa.univ-lille3.fr/polys/php/exemples/news/listenoms.php>
3. <http://grappa.univ-lille3.fr/polys/php/exemples/news/listeliens.php>

```
        if (substr($file,-4)==".jnl") {
            echo "<li><a href=\"unenouvelle.php?f=$file\">$file</a></li>\n";
        }
    }
    closedir($dir);
?>
</ul>
</body></html>
```

Cette page nécessite l'existence d'une page `unenouvelle.php` qui permet d'afficher le contenu d'une nouvelle:

```
<html><body>
<?php require($f);?>
</body></html>
```

26.4. Affichage de toutes les nouvelles

`affichetout.php`⁴:

```
<html><body>
<h1>News...</h1>
<h2>Affichage de toutes les nouvelles</h2>
<hr />
<?php
$dir = opendir(".");
while($file = readdir($dir)) {
    if (substr($file,-4)==".jnl") {
        $f=file($file);
        for ($i=0;$i<count($f);$i++) {
            echo $f[$i];
        }
        echo "<hr />";
    }
}
closedir($dir);
?>
</ul>
</body></html>
```

26.5. Affichage des titres des nouvelles

`affichetitre.php`⁵:

```
<html><body>
<h1>News...</h1>
<h2>Affichage des titres des nouvelles</h2>
<ul>
<?php
$dir = opendir(".");
while($file = readdir($dir)) {
    if (substr($file,-4)==".jnl") {
        $f=file($file);
        echo "<li>$f[0]</li>\n";
    }
}
closedir($dir);
?>
</ul>
</body></html>
```

4. <http://grappa.univ-lille3.fr/polys/php/exemples/news/affichetout.php>

5. <http://grappa.univ-lille3.fr/polys/php/exemples/news/affichetitre.php>

26.6. Affichage des titres faisant lien vers les nouvelles

affichetitrelien.php⁶:

```
<html><body>
<h1>News...</h1>
<h2>Affichage des titres faisant lien vers les nouvelles</h2>
<ul>
<?php
$dir = opendir(".");
while($file = readdir($dir)) {
    if (substr($file,-4)==".jnl") {
        $f=file($file);
        echo "<li><a href=\"\$file\">$f[0]</a></li>\n";
    }
}
closedir($dir);
?>
</ul>
</body></html>
```

Cette page nécessite l'existence de unenouvelle.php.

26.7. Affichage de toutes les nouvelles, proprement

affichetout_br.php⁷:

```
<html><body>
<h1>News...</h1>
<h2>Affichage de toutes les nouvelles, proprement</h2>
<hr />
<?php
$dir = opendir(".");
while($file = readdir($dir)) {
    if (substr($file,-4)==".jnl") {
        $f=file($file);
        echo "<h3>$f[0]</h3>\n";
        for ($i=1;$i<count($f);$i++) {
            echo $f[$i];
            if (trim($f[$i])=="") {
                echo "<br />";
            }
        }
        echo "<hr />";
    }
}
closedir($dir);
?>
</body></html>
```

6. <http://grappa.univ-lille3.fr/polys/php/exemples/news/affichetitrelien.php>

7. http://grappa.univ-lille3.fr/polys/php/exemples/news/affichetout_br.php

26.8. Choix du sujet

sujet.php⁸:

```
<html><body>
<h1>News...</h1>
<h2>Choix du sujet</h2>
<form action="sujet2.php">
  <select name="sujet">
    <option value="sci">Sciences</option>
    <option value="inf">informatique</option>
    <option value="msc">Divers</option>
  </select>
  <input type="submit">
</form>
</body></html>
```

8. <http://grappa.univ-lille3.fr/polys/php/exemples/news/sujet.php>

IV. Études de cas

Chapitre 27. Études de cas

Cette partie contient quelques études de cas à peu près réalistes, si ce n'est réels...

Ce sont des projets qui ont été demandés aux étudiants de maîtrise AES-TEG, 2^{ème} année d'IUP IIES et de 3^{ème} année de licence MIASHS.

Ils sont pour la plupart basés sur des cas réels, mais ont été le plus souvent modifiés dans un but pédagogique (pour simplifier une situation vraiment trop complexe, ou au contraire pour complexifier une situation un peu trop simpliste pour un exercice d'application du cours).

Aucun corrigé de ces projets n'est ni ne sera disponible. Il est donc inutile d'en réclamer...

Chapitre 28. Projet *Disques 2009*

Licence MIA SHS 3^{ème} année, 2009-2010. Vous pouvez consulter les pages originales¹ sur le web.

28.1. Présentation

Vous allez devoir créer un site web qui gère le contenu d'une discothèque. Vous manipulerez des informations comme le nom des disques, leurs années de parution, les artistes, les textes des chansons, etc. Ces informations seront accessibles à d'autres personnes, vous devrez aussi gérer les droits d'accès à votre site web.

28.1.1. Fonctionnalités générales

- Liste des disques.
- Liste des chansons pour chaque disque.
- Un artiste/groupe pour chaque disque, mais possibilité de préciser des artistes/groupes particuliers pour chaque chanson (compilations, duos, etc.).
- Genres musicaux, langue(s) pour chaque artiste/groupe, langue(s) pour chaque chanson.
- Auteur(s) des chansons.
- Paroles des chansons.
- Appartenance(s) d'un artiste à un groupe.

28.1.2. Fonctionnalités de consultation

- Obtenir une liste des disques (avec les caractéristiques) et la liste des chansons.
- Obtenir une liste des artistes.
- Rechercher une chanson.
- Avoir accès à quelques statistiques de base.

28.1.3. Modification des données, accès aux données sensibles

Il est indispensable que les fonctionnalités suivantes soient protégées par une authentification de l'utilisateur. En plus des fonctionnalités de consultation (voir ci-dessus) qui peuvent être protégées par un mot de passe si vous le désirez (le site serait alors d'accès exclusivement privé, mais ce n'est pas obligatoire), on distinguera deux niveaux d'autorisation, qu'on appellera fort et faible dans la suite.

28.1.3.1. Niveau faible

C'est le niveau où sont accessibles des données sensibles, où sont modifiées des informations non vitales. La plupart des membres de confiance auront accès à ce niveau.

- Consultations des paroles des chansons (accès privé nécessaire pour des raisons de respect du droit d'auteur).
- Insertion d'information dans la base (créer un artiste, un genre musical, un disque, une chanson, etc.).

1. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/2009-2010/bd/projetdisques/>

28.1.3.2. Niveau fort

C'est le niveau où sont modifiées les informations vitales pour le fonctionnement du site. Seules quelques personnes, responsables du site, auront accès à ce niveau.

- Gérer les utilisateurs et leurs droits (créer, modifier, supprimer).
- Modifier ou supprimer des données existantes (paroles de chanson, orthographe des noms, etc.).

28.2. La base de données

L'analyse ne sera pas présentée ici. Nous nous contenterons de donner la structure de la base.

28.2.1. Les tables

- La table artiste :

```
CREATE TABLE artiste (  
    art_num INTEGER NOT NULL PRIMARY KEY,  
    art_prenom CHARACTER VARYING,  
    art_nom CHARACTER VARYING,  
    art_groupe CHARACTER VARYING,  
    art_modif TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP);
```

- La table etat :

```
CREATE TABLE etat (  
    eta_num INTEGER NOT NULL PRIMARY KEY,  
    eta_nom CHARACTER VARYING);
```

- La table genre :

```
CREATE TABLE genre (  
    gen_num INTEGER NOT NULL PRIMARY KEY,  
    gen_nom CHARACTER VARYING,  
    gen_modif TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP);
```

- La table droits :

```
CREATE TABLE droits (  
    dro_num INTEGER NOT NULL PRIMARY KEY,  
    dro_nom CHARACTER VARYING,  
    dro_niveau INTEGER);
```

- La table langue :

```
CREATE TABLE langue (  
    lan_num CHARACTER VARYING NOT NULL PRIMARY KEY,  
    lan_nom CHARACTER VARYING);
```

- La table langueartiste :

```
CREATE TABLE langueartiste (  
    laa_langue CHARACTER VARYING NOT NULL REFERENCES langue(lan_num),  
    laa_artiste INTEGER NOT NULL REFERENCES artiste(art_num),  
    laa_maternelle BOOLEAN DEFAULT true,  
    PRIMARY KEY (laa_langue, laa_artiste));
```

- La table autorisations :

```
CREATE TABLE autorisations (  
    aut_login CHARACTER VARYING NOT NULL PRIMARY KEY,  
    aut_motdepasse CHARACTER VARYING,  
    aut_nom CHARACTER VARYING,  
    aut_prenom CHARACTER VARYING,  
    aut_droits INTEGER REFERENCES droits(dro_num));
```

- La table chanson :

```
CREATE TABLE chanson (  
    cha_num INTEGER NOT NULL PRIMARY KEY,  
    cha_titre CHARACTER VARYING,
```

```

cha_genre INTEGER REFERENCES genre(gen_num),
cha_texte CHARACTER VARYING,
cha_libre BOOLEAN,
cha_modif TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP);

```

- La table languechanson :

```

CREATE TABLE languechanson (
    lac_langue CHARACTER VARYING NOT NULL REFERENCES langue(lan_num),
    lac_chanson INTEGER NOT NULL REFERENCES chanson(cha_num),
    lac_principale BOOLEAN DEFAULT true NOT NULL,
    PRIMARY KEY (lac_langue, lac_chanson, lac_principale));

```

- La table membre :

```

CREATE TABLE membre (
    mem_membre INTEGER NOT NULL REFERENCES artiste(art_num),
    mem_groupe INTEGER NOT NULL REFERENCES artiste(art_num),
    mem_debut CHARACTER VARYING NOT NULL,
    mem_fin CHARACTER VARYING NOT NULL,
    PRIMARY KEY (mem_membre, mem_groupe, mem_debut, mem_fin));

```

- La table disque :

```

CREATE TABLE disque (
    dis_num INTEGER NOT NULL PRIMARY KEY,
    dis_titre CHARACTER VARYING,
    dis_artiste INTEGER REFERENCES artiste(art_num),
    dis_annee CHARACTER VARYING,
    dis_cddb CHARACTER VARYING,
    dis_anneeachat CHARACTER VARYING,
    dis_prixachat NUMERIC(8,2),
    dis_etat INTEGER REFERENCES etat(eta_num),
    dis_perdu BOOLEAN DEFAULT false,
    dis_modif TIMESTAMP WITHOUT TIME ZONE DEFAULT CURRENT_TIMESTAMP);

```

- La table interprete :

```

CREATE TABLE interprete (
    int_artiste INTEGER NOT NULL (int_artiste) REFERENCES artiste(art_num),
    int_chanson INTEGER NOT NULL REFERENCES chanson(cha_num),
    int_disque INTEGER NOT NULL REFERENCES disque(dis_num),
    int_numero INTEGER NOT NULL,
    PRIMARY KEY (int_artiste, int_chanson, int_disque, int_numero));

```

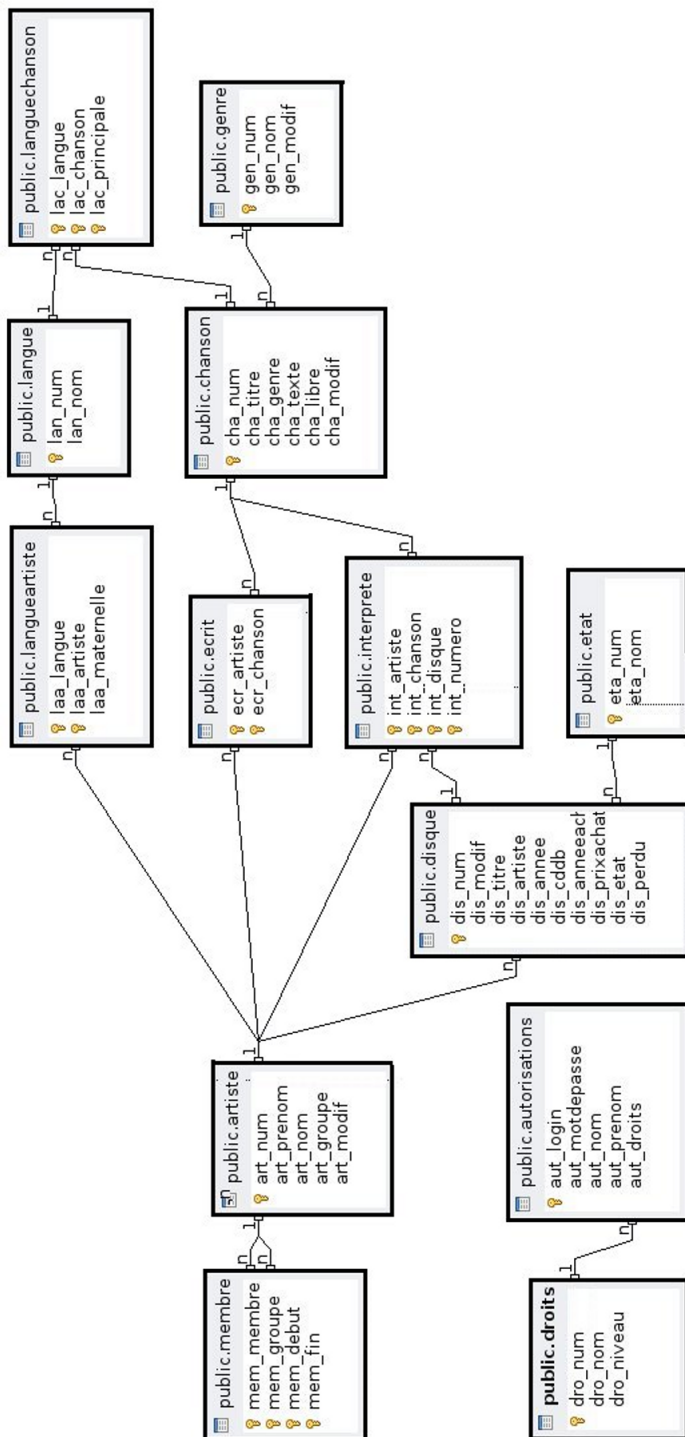
- La table ecrit :

```

CREATE TABLE ecrit (
    ecr_artiste INTEGER NOT NULL REFERENCES artiste(art_num),
    ecr_chanson INTEGER NOT NULL REFERENCES chanson(cha_num),
    PRIMARY KEY (ecr_artiste, ecr_chanson)
);

```

28.2.2. MLD



Chapitre 29. Projet *Inscriptions*

Licence MIA SHS 3^{ème} année, 2008-2009. Vous pouvez consulter les pages originales¹ sur le web.

29.1. Présentation

Vous allez travailler pour Lille 3...

Lille 3 Numérique² organise des stages de familiarisation des étudiants avec leur ENT³. Il faut une application qui permette l'inscription des étudiants Lille 3 sur des créneaux horaires précis.

Vous allez leur venir en aide : c'est vous qui allez réaliser cette application.

29.1.1. Fonctionnalités générales

- Un calendrier avec des créneaux horaires sur lesquels on peut cliquer.
- Possibilité de s'inscrire sur le créneau de son choix mais sans authentification nécessaire (nom, prénom, UFR et formation, numéro étudiant).
- Pas de possibilité de suppression d'une inscription par l'étudiant. Un étudiant ne peut s'inscrire qu'à un seul créneau horaire. En cas d'erreur il devra s'adresser au gestionnaire dont l'adresse figurera sur le site. (Vérification par le numéro d'étudiant qu'un étudiant n'est pas déjà inscrit sur un créneau pour éviter les inscriptions multiples.)
- Notification sur le calendrier qu'un créneau est complet.

29.1.2. Fonctionnalités du calendrier

- Étendue du calendrier paramétrée automatiquement en fonction des séances existantes.
- Un créneau se ferme automatiquement dès que le seuil est atteint (seuil paramétrable par le gestionnaire).

29.1.3. Gestion de la base

- Authentification indispensable.
- Créer, modifier ou supprimer une séance.
- Lister tous les créneaux et/ou les imprimer.
- Lister tous les inscriptions et/ou les imprimer.
- Pouvoir fermer un créneau (éventuellement en ouvrir un nouveau) : la mention « groupe complet » apparaîtra sur le calendrier.
- Modifier ou supprimer une inscription d'un étudiant à un créneau.
- Pouvoir facilement vider complètement un créneau, voire tous les créneaux pour utilisation ultérieure de la base.
- Visualiser la liste de tous les créneaux avec tri par : créneau / nom / numéro étudiant / année / UFR.
- Possibilité de faire un export sur tableur.

29.2. La base de données

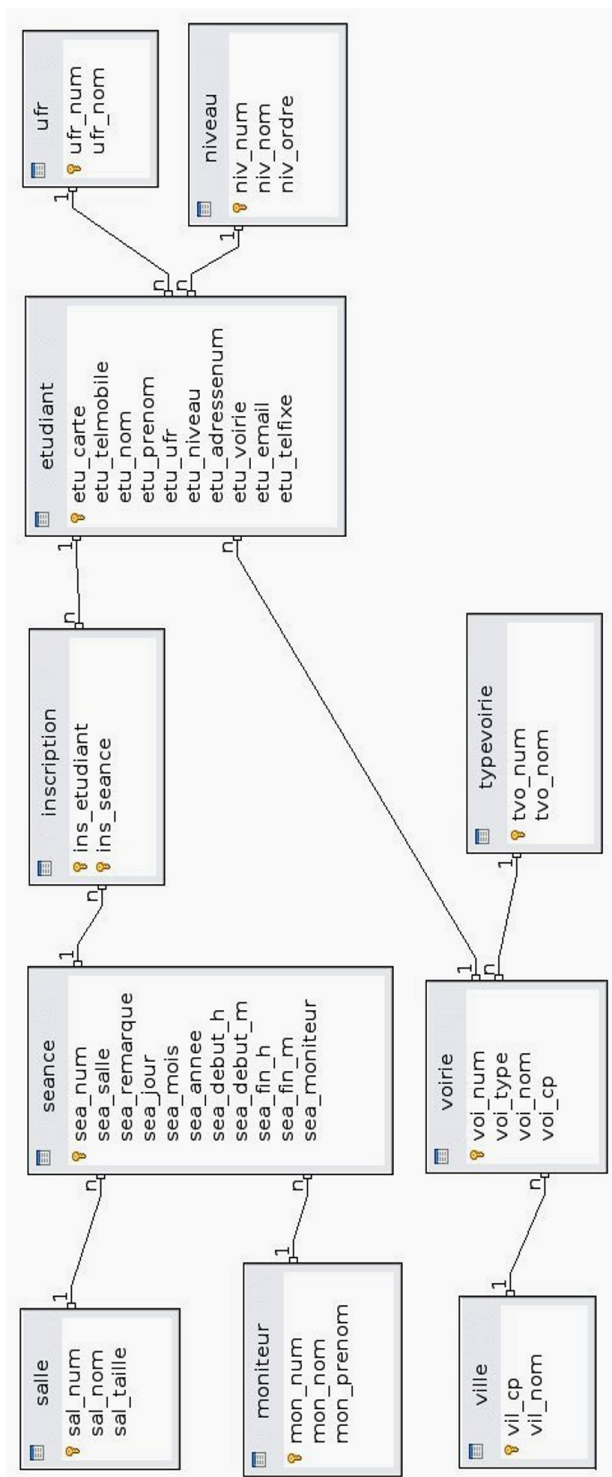
L'analyse ne sera pas présentée ici. Nous nous contenterons de donner la structure de la base.

1. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/2008-2009/technoweb/projetinscription/>

2. <http://www.univ-lille3.fr/fr/lille3-numerique/>

3. <http://www.univ-lille3.fr/fr/lille3-numerique/ent/>

29.2.1. MLD



29.2.2. Les tables

La table `ufr` contient les renseignements sur les UFR :

```

CREATE TABLE ufr (
    ufr_num INTEGER PRIMARY KEY,
    ufr_nom VARCHAR(40) NOT NULL
);
    
```

La table `moniteur` contient les renseignements sur les moniteurs qui encadrent les séances :

```
CREATE TABLE moniteur (
    mon_num INTEGER PRIMARY KEY,
    mon_nom VARCHAR(40) NOT NULL,
    mon_prenom VARCHAR(40)
);
```

La table `salle` contient les renseignements sur les salles où se déroulent les séances :

```
CREATE TABLE salle (
    sal_num INTEGER PRIMARY KEY,
    sal_nom VARCHAR(40) NOT NULL,
    sal_taille INTEGER
);
```

La table `niveau` contient les renseignements sur les niveaux d'études des étudiants (licence 1ère année, etc.) :

```
CREATE TABLE niveau (
    niv_num INTEGER PRIMARY KEY,
    niv_nom VARCHAR(40) NOT NULL,
    niv_ordre INTEGER NOT NULL
);
```

La table `seance` contient les renseignements sur les séances :

```
CREATE TABLE seance (
    sea_num INTEGER PRIMARY KEY,
    sea_jour INTEGER NOT NULL,
    sea_mois INTEGER NOT NULL,
    sea_annee INTEGER NOT NULL,
    sea_debut_h INTEGER NOT NULL,
    sea_debut_m INTEGER NOT NULL,
    sea_fin_h INTEGER NOT NULL,
    sea_fin_m INTEGER NOT NULL,
    sea_moniteur INTEGER REFERENCES moniteur(mon_num),
    sea_salle INTEGER REFERENCES salle(sal_num),
    sea_remarque VARCHAR(200) DEFAULT ""
);
```

La table `ville` contient les renseignements sur les villes :

```
CREATE TABLE ville (
    vil_cp VARCHAR(10) PRIMARY KEY,
    vil_nom VARCHAR(40) NOT NULL
);
```

La table `typevoirie` contient les renseignements sur les types de voiries (rue, boulevard, avenue, etc.) :

```
CREATE TABLE typevoirie (
    tvo_num INTEGER PRIMARY KEY,
    tvo_nom VARCHAR(40) NOT NULL
);
```

La table `voirie` contient les renseignements sur les voiries :

```
CREATE TABLE voirie (
    voi_num INTEGER PRIMARY KEY,
    voi_type INTEGER NOT NULL REFERENCES typevoirie(tvo_num),
    voi_nom VARCHAR(100) NOT NULL,
    voi_cp VARCHAR(6) NOT NULL REFERENCES ville(vil_cp)
);
```

La table `etudiant` contient les renseignements sur les étudiants :

```
CREATE TABLE etudiant (
    etu_carte VARCHAR PRIMARY KEY,
    etu_nom VARCHAR(40) NOT NULL,
    etu_prenom VARCHAR(40),
    etu_ufr INTEGER REFERENCES ufr(ufr_num),
    etu_niveau INTEGER REFERENCES niveau(niv_num),
    etu_adressenum VARCHAR(10),
```

```
        etu_voirie INTEGER NOT NULL REFERENCES voirie(voi_num),
        etu_email VARCHAR(40),
        etu_telfixe VARCHAR(20),
        etu_telmoblie VARCHAR(20)
    );
```

La table `inscription` contient les renseignements sur les inscriptions des étudiants aux séances :

```
CREATE TABLE inscription (
    ins_etudiant VARCHAR(20) REFERENCES etudiant(etu_carte),
    ins_seance INTEGER REFERENCES seance(sea_num),
    PRIMARY KEY(ins_etudiant,ins_seance)
);
```

La table `parametre` contient les renseignements sur les différents paramètres de l'application (mots de passe, etc.) :

```
CREATE TABLE parametre (
    par_num INTEGER PRIMARY KEY,
    par_nom VARCHAR(20),
    par_texte VARCHAR(100),
    par_valeur VARCHAR(200),
    par_ordre INTEGER
);
```

Chapitre 30. Projet *Teniraq*

Licence MIA SHS 3^{ème} année, 2007-2008. Vous pouvez consulter les pages originales¹ sur le web.

30.1. Présentation

Le club de tennis *TENIRAQ* (RAQ comme raquette) utilise une base de données pour administrer son fonctionnement : adhésions, entrées dans les locaux, réservations sur les 3 courts (deux courts couverts A et B et un court extérieur C), matchs défis (matchs internes au club qui permettent de classer les joueurs).

Chaque adhérent reçoit dès qu'il est inscrit (c'est-à-dire dès qu'il a payé sa cotisation annuelle) un badge possédant un numéro (appelé numéro d'adhérent) et un mot de passe lui permettant de se connecter sur la base de données du club par internet.

30.1.1. Adhésions

Les adhésions en cours sont mémorisées dans une table *badges* contenant pour chaque personne le numéro de badge, le mot de passe, le nom, le prénom, etc. Lorsqu'un adhérent régularise sa situation, la date limite de validité est augmentée d'une année, jour pour jour, pour un badge en cours de validité. Pour les autres, la date de validité est la date d'inscription augmentée d'une année, jour pour jour.

En plus des joueurs, plusieurs autres personnes reçoivent un badge de catégorie *logistique* afin de pouvoir entrer dans le club. Il s'agit de l'adjoint aux sports de la commune, du responsable des associations ainsi que du chef de travaux. Il s'agit également des personnes qui assurent l'entretien du matériel, le nettoyage de la salle. Ce nettoyage a lieu tous les mardis matins entre 9h et 12h. Durant ce créneau, aucune réservation de terrain n'est autorisée pour les joueurs. Ces personnes sont considérées comme des adhérents à vie. Au cas où un personnel de mairie autorisé serait aussi joueur, il aurait deux badges : un *badge logistique* et un *badge joueur*.

Les badges donnent des droits différents à leur propriétaire, selon la catégorie :

- *Logistique* pour les non joueurs : droit d'entrer dans le club, pas le droit de réserver.
- *Jeune* pour les joueurs ayant moins de 15 ans : pas le droit d'entrée (ils doivent être accompagnés d'un adhérent plus âgé pour entrer, pour des problèmes de responsabilité civile).
- *Ado* pour les joueurs dont l'âge est compris entre 15 et 18 ans. Ils ont le droit d'entrer dans le club et de réserver excepté les soirs de la semaine après 17h (afin de permettre aux travailleurs de réserver le soir).
- *Senior* pour les joueurs de plus de 18 ans. Ils ont le droit d'entrer et de réserver à tout moment.

30.1.2. Entrées dans les salles

Chaque adhérent doit saisir, sur un clavier situé près de la porte d'entrée, son numéro d'adhérent et son mot de passe pour ouvrir la porte. Ce faisant, l'évènement est enregistré par l'ajout d'une ligne dans la table *acces*. Cette table constitue donc un historique des entrées dans le club. Elle permet, en cas de vol ou de dégradation de retrouver qui se trouvait dans le club à ce moment là.

Un adhérent qui n'a pas renouvelé sa cotisation annuelle a encore le droit de rentrer pendant un mois, mais ne peut plus réserver de terrain.

30.1.3. Réservations de courts

En se connectant sur la base de données, chaque adhérent joueur a la possibilité :

- soit de valider sa réservation pour prouver qu'il est venu jouer à ce moment,
- soit d'annuler une réservation qu'il a faite,
- soit de réserver un court (A ou B ou C) pour une certaine date, à une certaine heure (par exemple le court B jeudi prochain à 17h), avec un partenaire.

1. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/2007-2008/technoweb/projetteniraq/>

Une réservation peut être effectuée pour les 4 semaines à venir, jour pour jour. Une nouvelle ligne est alors ajoutée à la table *reservations*, qui mémorise l'ensemble des réservations (numéro du court, date, heure). Le joueur réservant enregistre également la réservation de son partenaire de jeu.

Une annulation ne peut être faite qu'au moins 6 heures avant de jouer, pour permettre à d'autres joueurs de prendre la place libre.

Une validation n'est possible que dans le créneau [réservation-15 mn, réservation+15 min]. Hors de ce créneau, on considère que le joueur n'a pas honoré sa réservation, et n'a pas occupé un créneau qui aurait pu être occupé avec plus de profit par un autre joueur. Il est alors automatiquement mis en prison pour 4 jours, c'est-à-dire qu'il ne peut plus effectuer de réservation pendant cette période. La date d'entrée en prison est enregistrée dans la table *badge*.

Une seule réservation (à venir) par badge joueur.

Un *badge logistique* ne donne pas le droit de réserver un terrain.

30.1.4. Matches défis

Les joueurs ont 0 point à chaque début de saison (début octobre). Chaque dimanche, des matchs défis se déroulent sur un créneau d'une heure. A la fin de l'heure, on compte le nombre de jeux gagnés par chacun des deux joueurs. En cas d'égalité, le joueur le plus jeune est déclaré vainqueur. Chaque joueur reçoit un point de participation. Le vainqueur reçoit en plus la différence entre les nombres de jeux gagnés. Par exemple, un joueur ayant gagné 8/3 reçoit $1 + (8-3) = 5$ points Défis.

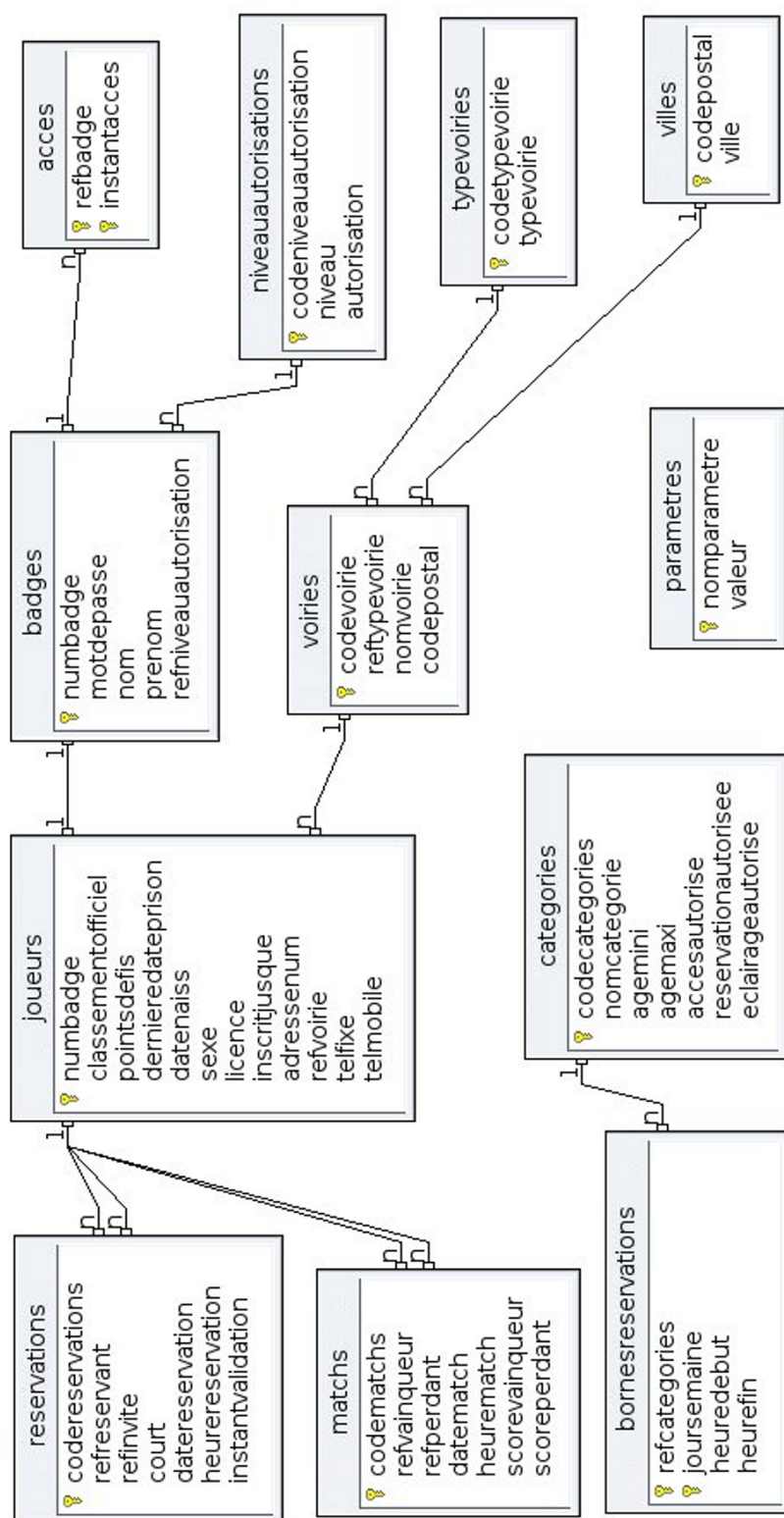
30.1.5. Paramétrage

Certaines valeurs sont modifiables : délai d'annulation d'une réservation (6 heures), délai pour valider sa réservation (15 min), nombre de jours de prison pour non validation (4 jours), nombre de semaines de réservation (4 semaines). Ces valeurs seront enregistrées dans une table *parametres* qui ne sera modifiable que par les possesseurs des badges dont le niveau d'autorisation est suffisant.

30.2. La base de données

L'analyse ne sera pas présentée ici. Nous nous contenterons de donner la structure de la base.

30.2.1. MLD



30.2.2. Les tables

La table `villes` contient les renseignements sur les villes :

```
CREATE TABLE villes (
    codepostal VARCHAR(6) PRIMARY KEY,
    ville VARCHAR(50)
```

);

La table `typevoiries` contient les renseignements sur les types de voiries (rue, boulevard, avenue, etc.) :

```
CREATE TABLE typevoiries (  
    codetypevoirie INTEGER PRIMARY KEY,  
    typevoirie VARCHAR(20)  
);
```

La table `voiries` contient les renseignements sur les voiries :

```
CREATE TABLE voiries (  
    codevoirie INTEGER PRIMARY KEY,  
    reftypevoirie INTEGER NOT NULL,  
    nomvoirie VARCHAR(100),  
    codepostal VARCHAR(6) NOT NULL,  
    FOREIGN KEY (reftypevoirie) REFERENCES typevoiries(codetypevoirie),  
    FOREIGN KEY (codepostal) REFERENCES villes(codepostal)  
);
```

La table `niveauautorisations` contient les renseignements sur les niveaux d'autorisation :

```
CREATE TABLE niveauautorisations (  
    codeniveauautorisation INTEGER PRIMARY KEY,  
    niveau INTEGER,  
    autorisation VARCHAR(20)  
);
```

La table `badges` contient les renseignements sur les personnes possédant un badge (joueurs et logistiques) :

```
CREATE TABLE badges (  
    numbadge INTEGER PRIMARY KEY,  
    motdepasse VARCHAR(20),  
    nom VARCHAR(50),  
    prenom VARCHAR(50),  
    refniveauautorisation INTEGER NOT NULL,  
    FOREIGN KEY (refniveauautorisation) REFERENCES niveauautorisations(codeniveauautorisation)  
);
```

La table `joueurs` contient les renseignements sur les joueurs :

```
CREATE TABLE joueurs (  
    numbadge INTEGER PRIMARY KEY,  
    datenaiss DATE,  
    sexe VARCHAR(1),  
    licence varchar(10),  
    inscritjusque DATE,  
    adressenum VARCHAR(10),  
    refvoirie INTEGER NOT NULL,  
    telfixe VARCHAR(20),  
    telmobile VARCHAR(20),  
    classementofficiel VARCHAR(20),  
    pointsdefis INTEGER,  
    dernieredateprison DATE,  
    FOREIGN KEY (refvoirie) REFERENCES voiries(codevoirie),  
    FOREIGN KEY (numbadge) REFERENCES badges(numbadge)  
);
```

La table `acces` contient les renseignements sur les accès aux salles :

```
CREATE TABLE acces (  
    refbadge INTEGER,  
    instantacces TIMESTAMP,  
    primary key (refbadge, instantacces),  
    FOREIGN KEY (refbadge) REFERENCES badges(numbadge)  
);
```

La table `reservations` contient les renseignements sur les réservations de courts :

```
CREATE TABLE reservations (  

```

```

codereservations INTEGER PRIMARY KEY,
refreservant INTEGER NOT NULL,
refinvite INTEGER NOT NULL,
court VARCHAR(4),
datereservation DATE,
heurereservation INTEGER,
instantvalidation TIME,
FOREIGN KEY (refreservant) REFERENCES joueurs(numbadge),
FOREIGN KEY (refinvite) REFERENCES joueurs(numbadge)
);

```

La table `matches` contient les renseignements sur les matchs :

```

CREATE TABLE matches (
    codematches INTEGER PRIMARY KEY,
    refvainqueur INTEGER NOT NULL,
    refperdant INTEGER NOT NULL,
    datematch DATE,
    heurematch INTEGER,
    scorevainqueur INTEGER,
    scoreperdant INTEGER,
    FOREIGN KEY (refvainqueur) REFERENCES joueurs(numbadge),
    FOREIGN KEY (refperdant) REFERENCES joueurs(numbadge)
);

```

La table `categories` contient les renseignements sur les catégories de joueurs :

```

CREATE TABLE categories (
    codecategories VARCHAR(1) PRIMARY KEY,
    nomcategorie VARCHAR(30),
    agemini INTEGER,
    agemaxi INTEGER,
    accesautorise BOOLEAN,
    reservationautorisee BOOLEAN,
    eclairageautorise BOOLEAN
);

```

La table `bornesreservations` contient les renseignements sur les heures de réservations possibles pour chaque jour :

```

CREATE TABLE bornesreservations (
    refcategories VARCHAR(1) NOT NULL,
    joursemaine VARCHAR(15),
    heuredebut INTEGER,
    heurefin INTEGER,
    PRIMARY KEY (refcategories, joursemaine),
    FOREIGN KEY (refcategories) REFERENCES categories(codecategories)
);

```

La table `parametres` contient les renseignements sur les paramètres de l'application :

```

CREATE TABLE parametres (
    nomparametre VARCHAR(40) PRIMARY KEY,
    valeur VARCHAR(50)
);

```


Chapitre 31. Projet *Camping*

Licence MIA SHS 3^{ème} année, 2006-2007. Vous pouvez consulter les pages originales¹ sur le web.

31.1. Présentation

Vous allez devoir gérer un camping-gîte...

Il s'agit d'un établissement qui offre à ses clients les services suivants :

- des emplacements pour des tentes de toutes tailles,
- des emplacements pour des caravanes ou des camping-cars (différents des précédents par la présence de branchements électriques),
- des *mobillhomes*,
- des chambres à deux lits (où il est possible d'ajouter un ou deux lits pliants pour des enfants),
- des lits en dortoirs.

Les tarifs dépendent de la durée de location : à la nuit, à la semaine, au mois ou à l'année, ainsi que du nombre de personnes. Une redevance sera également perçue sur les véhicules.

31.2. Les informations à conserver

Les informations sur les clients et les locations sont conservées d'une année sur l'autre, afin de pouvoir procéder à des envois de courriers publicitaires.

On devra dans ce cas garder les informations sur toutes les personnes, et pas seulement sur celui qui a réglé la facture.

On aura les informations suivantes sur les emplacements : Est-ce pour tente et/ou caravane et/ou camping-car ? Est-ce ombragé ? Est-ce herbeux ?

On aura les informations suivantes sur les *mobillhomes* : équipé d'un réfrigérateur ? équipé d'un four ? etc.

On aura les informations suivantes sur les chambres et dortoirs : nombre de lits, présence de toilettes, lavabo, douche.

Pour toutes les prestations (chambres, dortoir, emplacement pour tente ou caravane, etc.) on aura les informations suivantes : prix à la nuit, à la semaine, au mois ou à l'année, localisation géographique (une référence sur un plan du camping).

On devra conserver les informations suivantes sur les clients :

- leur état civil le plus complet possible (mais en restant raisonnable quand même dans le respect de la vie privée...) : nom, prénom, adresse, téléphone, adresse électronique
- l'historique de toutes leurs présences au camping : date, avec quelles personnes, sur quel type d'emplacement (tente, caravane, etc.).

31.3. Les traitements

C'est bien beau d'avoir toutes ces informations, encore faut pouvoir les utiliser...

On devra pouvoir :

- bien entendu établir une facture pour tout séjour,
- mais aussi faire des réservations,
- ajouter des clients *potentiels* dans la base (c'est-à-dire des personnes qui ne sont pas réellement clientes, mais qu'on espère pouvoir démarcher),
- faire quelques statistiques nécessaires à la gestion du camping : recettes par année, recettes par mois, fréquentation par année, fréquentation par mois, par type d'hébergement, etc.

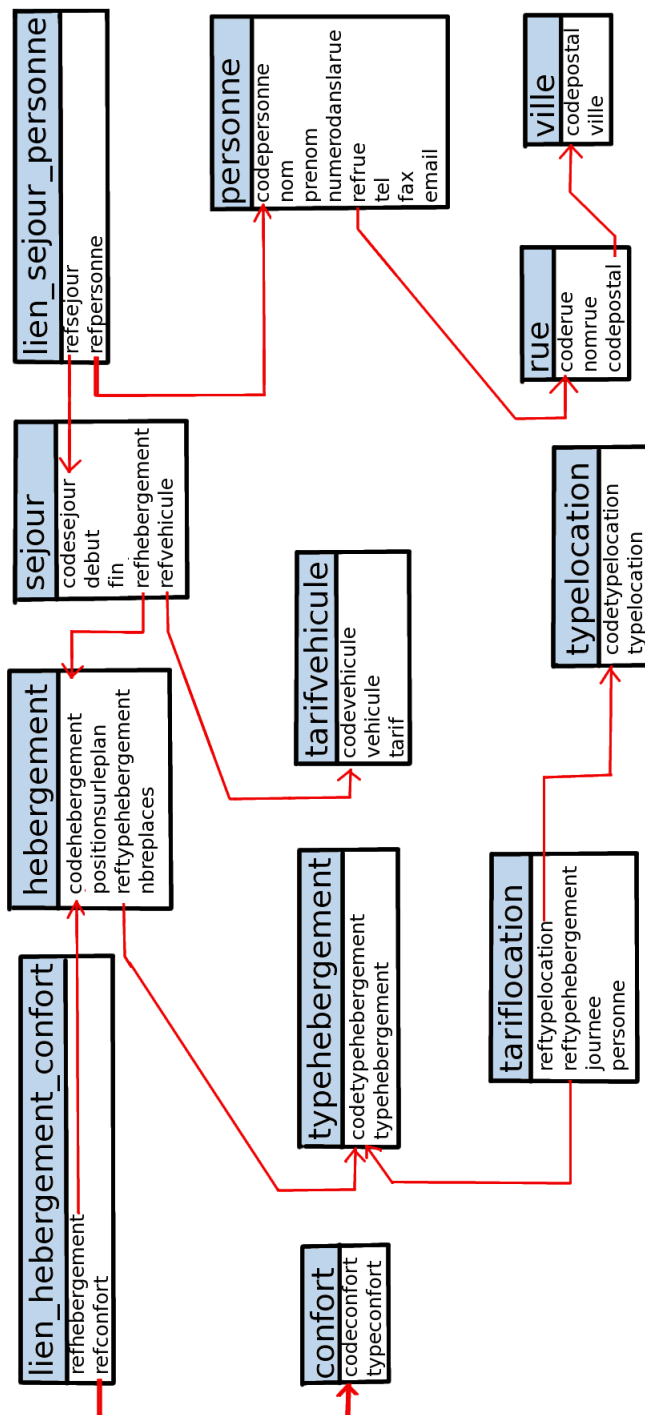
1. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/2006-2007/technoweb/projetcamping/>

- etc.

31.4. La base de données

L'analyse ne sera pas présentée ici. Nous nous contenterons de donner la structure de la base.

31.4.1. MLD



31.4.2. Les tables

La table `ville` contient les renseignements sur les villes :

```
CREATE TABLE ville (
    codepostal VARCHAR PRIMARY KEY,
    ville VARCHAR
);
```

La table `rue` contient les informations sur les rues, elle se réfère à la table `ville` :

```
CREATE TABLE rue (
    coderue INT PRIMARY KEY,
    nomrue VARCHAR,
    codepostal VARCHAR,
    CONSTRAINT rue_codepostal_fk
        FOREIGN KEY (codepostal)
            REFERENCES ville(codepostal)
);
```

La table `personne` rassemble les informations sur les personnes, clientes ou non :

```
CREATE TABLE personne (
    codepersonne INT PRIMARY KEY,
    nom VARCHAR NOT NULL,
    prenom VARCHAR,
    numerodanslarue VARCHAR,
    refrue INT,
    tel VARCHAR,
    fax VARCHAR,
    email VARCHAR,
    CONSTRAINT personne_refrue_fk
        FOREIGN KEY (refrue)
            REFERENCES rue(coderue)
);
```

La table `confort` liste les éléments de confort (douche, four, etc.) qui peuvent être disponibles dans les différents hébergements ; c'est la table `lien_hebergement_confort` qui permet de faire le lien entre les hébergements et les éléments de confort :

```
CREATE TABLE confort (
    codeconfort INT PRIMARY KEY,
    typeconfort VARCHAR NOT NULL
);
```

La table `typehebergement` rassemble les différents types d'hébergement (camping, dortoir, etc.) :

```
CREATE TABLE typehebergement (
    codetypehebergement INT PRIMARY KEY,
    typehebergement VARCHAR NOT NULL
);
```

La table `typelocation` rassemble les différents types de location (à la journée, à la semaine, etc.) :

```
CREATE TABLE typelocation (
    codetypelocation INT PRIMARY KEY,
    typelocation VARCHAR NOT NULL
);
```

La table `tariflocation` rassemble les différents tarifs de location en fonction de la durée, du type d'hébergement, etc.) :

```
CREATE TABLE tariflocation (
    reftypelocation INT,
    reftypehebergement INT,
    journee NUMERIC(5,2),
    personne NUMERIC(5,2),
    PRIMARY KEY (reftypelocation, reftypehebergement),
    CONSTRAINT tariflocation_reftypehebergement_fk
        FOREIGN KEY (reftypehebergement)
            REFERENCES typehebergement(codetypehebergement)
);
```

```
REFERENCES typehebergement (codetypehebergement),
CONSTRAINT tariflocation_reftypepelocation_fk
    FOREIGN KEY (reftypepelocation)
        REFERENCES typelocation (codetypelocation)
);
```

La table `tarifvehicule` définit le montant des redevances en fonction des véhicules :

```
CREATE TABLE tarifvehicule (
    codevehicule INT PRIMARY KEY,
    vehicule VARCHAR,
    tarif NUMERIC(5,2) NOT NULL
);
```

La table `hebergement` contient la liste de tous les hébergements disponibles :

```
CREATE TABLE hebergement (
    codehebergement INT PRIMARY KEY,
    positionsurleplan VARCHAR NOT NULL,
    reftypehebergement INT NOT NULL,
    nbreplaces INT NOT NULL,
    CONSTRAINT hebergement_reftypehebergement_fk
        FOREIGN KEY (reftypehebergement)
            REFERENCES typehebergement (codetypehebergement)
);
```

La table `lien_hebergement_confort` fait le lien entre les hébergements et les éléments de confort disponibles :

```
CREATE TABLE lien_hebergement_confort (
    refhebergement INT,
    refconfort INT,
    PRIMARY KEY (refhebergement, refconfort),
    CONSTRAINT lienhebconfort_refhebergement_fk
        FOREIGN KEY (refhebergement)
            REFERENCES hebergement (codehebergement),
    CONSTRAINT lienhebconfort_confort_fk
        FOREIGN KEY (refconfort)
            REFERENCES confort (codeconfort)
);
```

La table `sejour` rassemble tous les renseignements sur les séjours effectués ou réservés par les clients :

```
CREATE TABLE sejour (
    codesejour INT PRIMARY KEY,
    debut DATE,
    fin DATE,
    refhebergement INT,
    refvehicule INT,
    CONSTRAINT sejour_refvehicule_fk
        FOREIGN KEY (refvehicule)
            REFERENCES tarifvehicule (codevehicule),
    CONSTRAINT sejour_refhebergement_fk
        FOREIGN KEY (refhebergement)
            REFERENCES hebergement (codehebergement)
);
```

La table `lien_sejour_personne` fait le lien entre les séjours (effectués ou réservés) et les personnes (clients) :

```
CREATE TABLE lien_sejour_personne (
    refsejour INT,
    refpersonne INT,
    PRIMARY KEY (refsejour, refpersonne),
    CONSTRAINT liensejourpersonne_refsejour_fk
        FOREIGN KEY (refsejour)
            REFERENCES sejour (codesejour),
    CONSTRAINT liensejourpersonne_refpersonne_fk
        FOREIGN KEY (refpersonne)
            REFERENCES personne (codepersonne)
);
```

Chapitre 32. Projet *Association*

Licence MIA SHS 3^{ème} année, 2004-2005. Vous pouvez consulter les pages originales¹ sur le web.

Vous allez devoir construire un site web permettant aux membres d'une association de gérer et consulter différents renseignements sur les adhésions.

32.1. Contenu du site web

Votre site devra permettre d'effectuer les opérations nécessaires au bon fonctionnement de la base, c'est-à-dire :

- Il devra y avoir (au moins) trois niveaux d'utilisateurs (identifiés par *login* et mot de passe) :

le niveau *simple utilisateur*

seule la consultation est autorisée (cette consultation peut même être restreinte à certaines données) ; c'est le niveau par exemple accessible aux simples adhérents de l'association ;

le niveau *administrateur*

on a alors accès à des fonctions importantes (par exemple de modification des données ou d'ajout dans la base), mais pas aux fonctions vitales ; ce sera par exemple le niveau réservé au trésorier ou au président de l'association ;

le niveau *gourou*

on a alors accès à tout ; c'est le niveau réservé exclusivement au responsable de la base de données ; c'est par exemple seulement à ce niveau de sécurité qu'on pourra supprimer des enregistrements dans la base.

- Le site web permettra toutes les actions dans la base :
 - consultation,
 - modification,
 - ajout,
 - suppression.

32.2. Conditions de travail

Vous pouvez travailler par groupe de deux (c'est même plutôt conseillé), mais pas plus.

C'est un travail libre, dans les conditions du réel (ou presque). Je ne juge pas pendant tout le semestre, je ne suis là que pour répondre aux questions.

32.3. Base de données

La base de données contenant les renseignements est celle vue au premier semestre : la base Association².

Quelques rappels sur ses buts et son contenu :

Rappel : Une association veut gérer ses adhérents par l'intermédiaire d'une base de données. Voici les principales informations sur le sujet :

1. On doit garder les informations sur les adhérents, même sur les anciens.
2. On doit connaître, pour chaque personne de la base :
 - a. son nom ;

1. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/2004-2005/technoweb/projet.php>
2. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/2004-2005/bd/association>

- b. son prénom ;
 - c. son adresse ;
 - d. s'il est membre du C.A. ;
 - e. s'il est jeune (pour le tarif réduit) ;
 - f. s'il est adhérent à titre gratuit ;
 - g. s'il est adhérent, quel est le montant de sa cotisation, le nombre de personnes concernées par cette adhésion, et cela pour chaque année.
3. On doit connaître le référent de chaque personne. Il s'agit de l'adhérent qui sera chargé de lui faire parvenir les différents courriers envoyés par l'association (comptes-rendus, relance, convocations, etc.). Il est également possible que ces courriers soient transmis par la poste.

32.4. MLD du projet *Association*

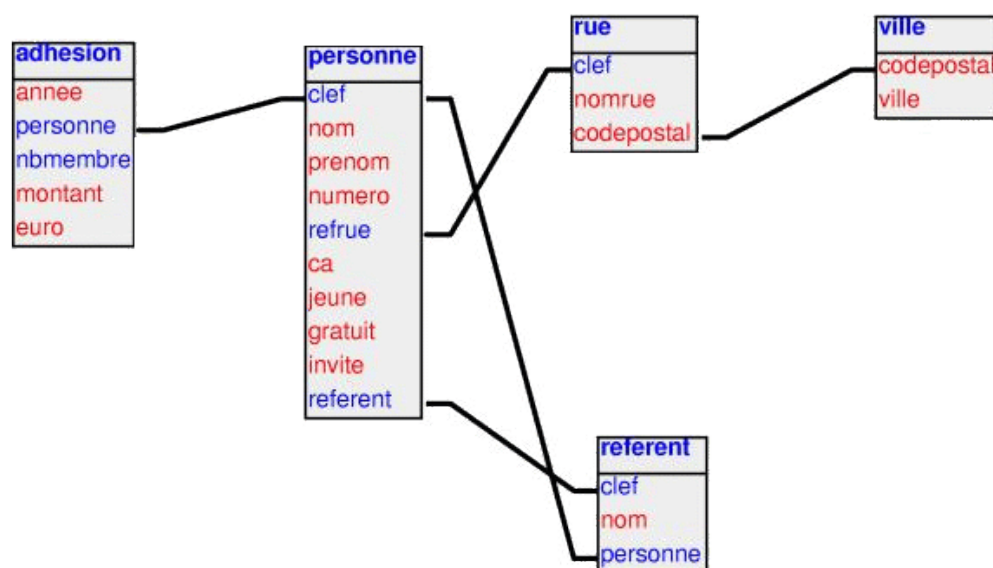


Figure 32-1. MCD du projet *Association*

32.5. Création des tables de la base *Association*

32.5.1. Création des tables

Ce fichier est téléchargeable sur le web³.

```

create table PERSONNE (
  clef int primary key ,
  nom varchar,
  prenom varchar,
  numero varchar,
  refrue integer,
  ca boolean,
  jeune boolean,
  gratuit boolean,

```

3. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/2004-2005/bd/association/creation/create.sql>

```

        invite boolean,
        referent int
    );

create table RUE (
    clef int primary key,
    nomrue varchar,
    codepostal varchar
);

create table VILLE (
    codepostal varchar primary key,
    ville varchar
);

create table REFERENT (
    clef int primary key,
    nom varchar,
    personne int
);

create table ADHESION (
    annee varchar,
    personne int,
    nbmembre int,
    montant numeric(8,2),
    euro boolean default 'T',
    primary key (annee,personne)
);

```

32.5.2. Contraintes

Ce fichier est téléchargeable sur le web⁴.

```

ALTER TABLE personne
    ADD CONSTRAINT persons_referent_fk
        FOREIGN KEY (referent) REFERENCES referent(clef);
ALTER TABLE personne
    ADD CONSTRAINT persons_rue_fk
        FOREIGN KEY (refrue) REFERENCES rue(clef);
ALTER TABLE rue
    ADD CONSTRAINT rue_ville_fk
        FOREIGN KEY (codepostal) REFERENCES ville(codepostal);
ALTER TABLE referent
    ADD CONSTRAINT referent_personne_fk
        FOREIGN KEY (personne) REFERENCES personne(clef);
ALTER TABLE adhesion
    ADD CONSTRAINT adhesions_person_fk
        FOREIGN KEY (personne) REFERENCES personne(clef);

```

32.6. Évaluation

Les critères sur lesquels vous serez notés :

- Il faut que ça marche, c'est la moindre des choses...
- La facilité d'utilisation du site : il faut que la navigation soit simple, intuitive et efficace ; il faut que les formulaires soient clairs.
- La conformité avec les normes du W3C. (L'utilisation de feuilles de style n'est absolument pas obligatoire, mais je crois que ça vous faciliterait quand même le travail.) Aucune restriction *a priori* n'est faite sur HTML : il suffit que ça passe les contrôles du W3C. Si vous êtes un fanatique de *javascript*, pourquoi pas,

4. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/2004-2005/bd/association/creation/contraintes.sql>

mais rien de vital ne doit être confié à *javascript*, et ça doit pouvoir fonctionner sur un navigateur sans *javascript*.

- La lisibilité de votre code source. Un code illisible, mal présenté, non commenté vous sera très préjudiciable.

Par contre un découpage en petites fonctions, faciles à comprendre, voire même le découpage du code en *bibliothèques thématiques*, vous avantageront.

32.7. Dernier conseil

Aphorisme : Les premiers 95% d'un projet prennent 95% du temps. Les 5% restant prennent aussi 95% du temps.

Conclusion : travaillez beaucoup le plus tôt possible, pendant que vous n'avez pas encore de partiels à réviser.

Chapitre 33. Projet *Généalogie*

IUP IIES 2^{ème} année et Maîtrise AES-TEG, 2003-2004. Vous pouvez consulter les pages originales¹ sur le web.

33.1. Généralités

Vous allez devoir réaliser un site de généalogie. Votre réalisation devra être pleinement fonctionnelle : elle devra, entre autres, être capable d'échanger des données avec d'autres logiciels de généalogie existant (logiciels commerciaux et logiciels libres).

33.2. Les données à conserver

Vous trouverez dans cette partie une description des informations que manipulent les généalogistes.

33.2.1. Les données de base

On s'occupe des personnes, de leur naissance et de leur décès, de leur ascendance et de leur descendance. On doit pouvoir en partant de n'importe quelle personne produire la liste des ses ascendants, ainsi que celle de ses descendants. On doit pouvoir aussi connaître les conjoints de toute personne ; dans le cas où une personne aurait eu plusieurs conjoints, on doit au moins pouvoir connaître l'ordre dans lequel ces couples ont existé. On doit pouvoir si possible avoir les dates de début et de fin des ces éventuelles unions. Il faut noter que le modèle européen n'est pas le seul qui existe au monde : il est possible que certaines personnes aient plusieurs conjoints en même temps. De plus les enfants peuvent provenir de liaisons non officialisées qu'il faut pouvoir aussi représenter.

On devra bien entendu conserver les dates et lieux de naissance et décès des individus.

Bien entendu, comme toute ou partie de ces informations est quelques fois difficile à obtenir, votre site devra pouvoir fonctionner en l'absence de certaines.

33.2.2. Données supplémentaires

Avec l'introduction de l'outil informatique on peut ajouter d'autres types d'information, par exemple :

- multimédia : photo, vidéo, enregistrement audio, etc. ;
- copie de document officiel : acte de naissance, de mariage, de décès, etc.

Ces informations peuvent être associées à une ou plusieurs personnes (ou à un ou plusieurs événements, ou à une ou plusieurs familles).

Il serait bon que ces documents soient datés et qu'on conserve le nom de leur propriétaires.

33.3. Traitement des données

Les données rassemblées n'ont de sens que si on peut les utiliser...

Voici les fonctionnalités minimales que devra avoir votre site web :

- ajouter, supprimer, modifier un individu ;
- ajouter, supprimer, modifier une union (mariage ou autre) ;
- lier un individu à ses parents (ou inversement un couple à ses enfants) ;
- associer un fichier multimédia à un individu, une famille, un événement ;
- connaissant un individu, on doit pouvoir obtenir facilement son ascendance et sa descendance proche ; ce serait cependant parfait d'avoir toute son ascendance et toute sa descendance ;

1. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/commun/genealogie/>

- pour tout individu on doit pouvoir avoir un album de toutes les photos qui lui sont associées (ce serait parfait d'avoir la même chose pour tous les types de documents).

33.4. Optimisations

On évitera de dupliquer les informations multimédia, même si elles sont liées à plusieurs personnes.

On pourra avoir le choix d'effacer ou pas les informations multimédia qui ne sont plus associées à rien.

La façon dont seront représentées les relations de descendance et d'ascendance à l'écran est laissée à votre libre choix, mais il faut noter que la plus classique est arborescente. C'est celle qui colle au plus près de la structure réelle.

33.5. Import-export

Votre site devra être capable d'exporter ses données au format GEDCOM.

Qu'est-ce qu'un fichier GEDCOM ? En tapant le mot dans n'importe quel moteur de recherche vous obtiendrez des dizaines (voire des centaines) de pages qui vous expliqueront ce que c'est.

En voici un résumé :

Défini par les mormons (ou sous leur nom complet : *Église de Jésus-Christ des Saints des Derniers Jours*), le standard GEDCOM a été créé pour permettre l'échange de données entre les différents logiciels de généalogie existants. Ainsi la plupart des logiciels de généalogie permettent d'exporter le fruit de votre travail dans ce format.

Quelques pages sur internet :

- version originale (en anglais)²
- une explication en français³
- une autre explication en français⁴

Mais vous pourrez facilement en trouver d'autres.

Attention

Comprendre cette norme par vous-même fait partie du travail qui vous est demandé.

Une partie de votre note portera sur la capacité de votre site à exporter vos données vers un fichier GEDCOM. La correction sera facile : je ferai lire le fichier produit par un logiciel de généalogie, et je regarderai ce qui aura vraiment été exporté.

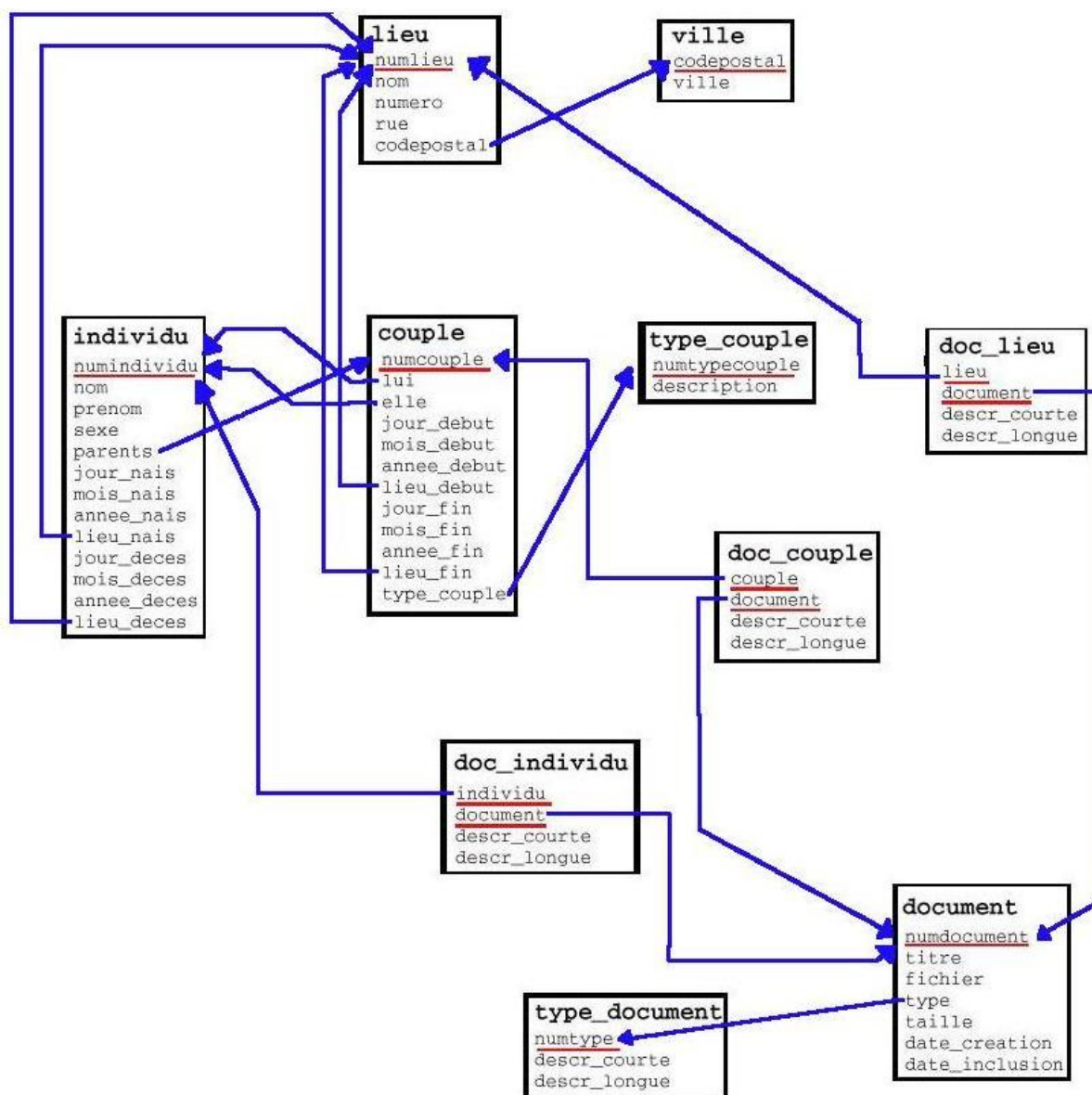
L'importation de données depuis un fichier GEDCOM ne vous est pas demandée (en principe c'est moins facile), mais si vous voulez le faire, il en sera bien entendu tenu compte.

2. <http://www.gendex.com/gedcom55/55gctoc.htm>

3. <http://www.amicale-genealogie.org/Informatique/InfoGed.htm>

4. <http://www.weinland.nom.fr/gedc.php>

33.6. MCD du projet *généalogie*

Figure 33-1. MCD du projet *Généalogie*

33.7. Les différentes tables

33.7.1. Les tables des entités

33.7.1.1. La table des individus

```

CREATE TABLE individu (
    numindividu INTEGER NOT NULL PRIMARY KEY,
    nom VARCHAR,
    prenom VARCHAR,
    sexe CHAR(1),
    parents INTEGER,
    jour_nais INTEGER,
    mois_nais INTEGER,
    annee_nais INTEGER,

```

```
    lieu_nais INTEGER,  
    jour_deces INTEGER,  
    mois_deces INTEGER,  
    annee_deces INTEGER,  
    lieu_deces INTEGER,  
    CONSTRAINT individu_sexe_ck  
        CHECK (sexe IN ('F','M','f','m'))  
    ) ;
```

33.7.1.2. La table des villes

```
CREATE TABLE ville (  
    codepostal VARCHAR NOT NULL PRIMARY KEY,  
    ville VARCHAR NOT NULL  
    ) ;
```

33.7.1.3. La table des lieux

```
CREATE TABLE lieu (  
    numlieu INTEGER NOT NULL PRIMARY KEY,  
    nom VARCHAR,  
    numero VARCHAR,  
    rue VARCHAR,  
    codepostal VARCHAR  
    ) ;
```

33.7.1.4. La table des couples

```
CREATE TABLE couple (  
    numcouple INTEGER NOT NULL PRIMARY KEY,  
    lui INTEGER NOT NULL,  
    elle INTEGER NOT NULL,  
    jour_debut INTEGER,  
    mois_debut INTEGER,  
    annee_debut INTEGER,  
    lieu_debut INTEGER,  
    jour_fin INTEGER,  
    mois_fin INTEGER,  
    annee_fin INTEGER,  
    lieu_fin INTEGER,  
    type_couple INTEGER  
    ) ;
```

33.7.1.5. La table des types de couple

```
CREATE TABLE type_couple (  
    numtypecouple INTEGER NOT NULL PRIMARY KEY,  
    description VARCHAR  
    ) ;
```

33.7.1.6. La table des types de document

```
CREATE TABLE type_document (  
    numtype INTEGER NOT NULL PRIMARY KEY,  
    descr_courte VARCHAR NOT NULL,  
    descr_longue VARCHAR  
    ) ;
```

33.7.1.7. La table des documents

```
CREATE TABLE document (
    numdocument INTEGER NOT NULL PRIMARY KEY,
    titre VARCHAR,
    fichier VARCHAR NOT NULL,
    type INTEGER NOT NULL,
    taille INTEGER,
    date_creation DATE,
    date_inclusion DATE
) ;
```

33.7.2. Les tables des relations

33.7.2.1. La table qui lie documents et couples

```
CREATE TABLE doc_couple (
    couple INTEGER NOT NULL,
    document INTEGER NOT NULL,
    descr_courte VARCHAR,
    descr_longue VARCHAR,
    CONSTRAINT docCouple_pk
        PRIMARY KEY (couple,document)
) ;
```

33.7.2.2. La table qui lie documents et lieux

```
CREATE TABLE doc_lieu (
    lieu INTEGER NOT NULL,
    document INTEGER NOT NULL,
    descr_courte VARCHAR,
    descr_longue VARCHAR,
    CONSTRAINT docLieu_pk
        PRIMARY KEY (lieu,document)
) ;
```

33.7.2.3. La table qui lie documents et individus

```
CREATE TABLE doc_individu (
    individu INTEGER NOT NULL,
    document INTEGER NOT NULL,
    descr_courte VARCHAR,
    descr_longue VARCHAR,
    CONSTRAINT docIndividu_pk
        PRIMARY KEY (individu,document)
) ;
```

33.8. Les différentes contraintes

33.8.1. Sur la table individu

```
ALTER TABLE individu
    ADD CONSTRAINT individu_lieuNais_fk
        FOREIGN KEY (lieu_nais) REFERENCES lieu(numlieu) ;
ALTER TABLE individu
    ADD CONSTRAINT individu_lieuDeces_fk
```

```
        FOREIGN KEY (lieu_deces) REFERENCES lieu(numlieu) ;
ALTER TABLE individu
    ADD CONSTRAINT individu_parents_fk
        FOREIGN KEY (parents) REFERENCES couple(numcouple) ;
```

33.8.2. Sur la table `ville`

Pas de contraintes...

33.8.3. Sur la table `lieu`

```
ALTER TABLE lieu
    ADD CONSTRAINT lieu_codepostal_fk
        FOREIGN KEY (codepostal) REFERENCES ville(codepostal);
```

33.8.4. Sur la table `couple`

```
ALTER TABLE couple
    ADD CONSTRAINT couple_lui_fk
        FOREIGN KEY (lui) REFERENCES individu(numindividu) ;
ALTER TABLE couple
    ADD CONSTRAINT couple_elle_fk
        FOREIGN KEY (elle) REFERENCES individu(numindividu) ;
ALTER TABLE couple
    ADD CONSTRAINT individu_lieuDebut_fk
        FOREIGN KEY (lieu_debut) REFERENCES lieu(numlieu) ;
ALTER TABLE couple
    ADD CONSTRAINT individu_lieuFin_fk
        FOREIGN KEY (lieu_fin) REFERENCES lieu(numlieu) ;
```

33.8.5. Sur la table `type_couple`

Pas de contraintes...

33.8.6. Sur la table `type_document`

Pas de contraintes...

33.8.7. Sur la table `document`

```
ALTER TABLE document
    ADD CONSTRAINT document_type_fk
        FOREIGN KEY (type) REFERENCES type_document(numtype) ;
```

33.8.8. Sur la table `doc_couple`

```
ALTER TABLE doc_couple
    ADD CONSTRAINT docCouple_couple_fk
        FOREIGN KEY (couple) REFERENCES couple(numcouple);
ALTER TABLE doc_couple
    ADD CONSTRAINT docCouple_document_fk
        FOREIGN KEY (document) REFERENCES document(numdocument) ;
```

33.8.9. Sur la table `doc_lieu`

```
ALTER TABLE doc_lieu
  ADD CONSTRAINT docLieu_lieu_fk
    FOREIGN KEY (lieu) REFERENCES lieu(numlieu) ;
ALTER TABLE doc_lieu
  ADD CONSTRAINT docLieu_document_fk
    FOREIGN KEY (document) REFERENCES document(numdocument) ;
```


Chapitre 34. Projet *Brazil*

IUP IIES 2^{ème} année et Maîtrise AES-TEG, 2002-2003. Vous pouvez consulter les pages originales¹ sur le web.

34.1. Description

Le monde de Brazil est organisé de façon à rendre ses citoyens les plus heureux possible. Pour veiller à cela, tous les renseignements utiles sont stockés dans la base de données centrale du centre de régulation du bonheur.

Vous êtes chargés de la gestion de cette base de données.

Chaque citoyen y est représenté par son numéro d'enregistrement universel, son nom, son prénom, son sexe, sa date de naissance, sa taille, l'historique de ses maladies. On mémorise également ses différentes adresses ainsi que les dates d'emménagement correspondantes.

Chaque citoyen travaille dans un service, contribuant ainsi au bonheur de tous. Tous les services sont regroupés, pour la plus grande efficacité, en différents ministères du bonheur. Ainsi un groupe de plusieurs services forme un ministère, où l'on travaille grâce aux échantillons de bonheur produits par d'autres services de divers ministères, et sous le contrôle administratif d'un troisième ensemble de services de divers ministères aussi. Ces rôles sont régulièrement modifiés par le centre de régulation du bonheur, selon un savant et parfait calcul de maîtrise de l'aléatoire.

Malheureusement, comme dans toute organisation parfaite, les défauts sont toujours dûs aux composants, i.e. les citoyens. Ainsi, on distingue trois types de citoyens :

1. D'une part, il y a ceux qui contribuent à la survie du système, par la production de futurs citoyens, et dont les mérites sont donc proportionnels au nombre de leurs enfants. Le bonheur produit est distillé aux enfants des citoyens. On tient donc particulièrement à conserver l'historique des ancêtres de chacun.
2. Tout citoyen (donc adulte) n'ayant pas d'enfant, fait partie de la seconde catégorie : les suspects. Cela se confirme par le fait qu'un citoyen n'ayant pas d'enfant n'est pas en mesure de répondre au formulaire E707. La suspicion qui pèse sur un citoyen est proportionnelle au nombre de formulaires E707 qu'il n'a pas pu remplir.
3. Tout citoyen dont les parents sont inconnus sera également suspect.

La seule exception, qui confirme la règle de perfection du système de Brazil, est le fait que seul un citoyen n'ayant pas d'enfants dispose d'assez de temps pour être le dirigeant d'un ministère.

Tout citoyen décédé, ne pouvant plus contribuer au bonheur de la société, se verra immédiatement supprimé de la base de données, ainsi que toute référence à son existence (pour ses fonctions ou sa descendance, ce qui rendra bien entendu immédiatement suspects ses éventuels enfants).

34.2. Les tables des entités

34.2.1. La table *citoyen*

```
CREATE TABLE citoyen (  
    numuniv NUMERIC (6) NOT NULL,  
    nom VARCHAR(50),  
    prenom VARCHAR(50),  
    sexe CHAR,  
    naissance DATE,  
    taille NUMERIC (3),  
    pere NUMERIC (6),  
    mere NUMERIC (6)) ;
```

1. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/commun/brazil/>

34.2.2. La table ville

```
CREATE TABLE ville (  
    codepostal VARCHAR (5) NOT NULL,  
    ville VARCHAR (25)) ;
```

34.2.3. La table rue

```
CREATE TABLE rue (  
    numrue NUMERIC (6) NOT NULL,  
    rue VARCHAR(50),  
    codepostal VARCHAR (5)) ;
```

34.2.4. La table maladie

```
CREATE TABLE maladie (  
    nummaladie NUMERIC (6) NOT NULL,  
    nommaladie VARCHAR (25),  
    gravite numeric (2) NOT NULL) ;
```

34.2.5. La table service

```
CREATE TABLE service (  
    numservice NUMERIC (6) NOT NULL,  
    nomservice VARCHAR (50)) ;
```

34.2.6. La table ministere

```
CREATE TABLE ministere (  
    numministere NUMERIC (6) NOT NULL,  
    nomministere VARCHAR (25)) ;
```

34.3. Les tables des relations

34.3.1. Associer une maladie à un citoyen : listemaladie

```
CREATE TABLE listemaladie (  
    citoyen NUMERIC (6) NOT NULL,  
    maladie NUMERIC (6) NOT NULL,  
    debut DATE NOT NULL,  
    fin DATE) ;
```

34.3.2. Associer une adresse à un citoyen : habite

```
CREATE TABLE habite (  
    citoyen NUMERIC (6) NOT NULL,  
    numero NUMERIC (4) NOT NULL,  
    rue NUMERIC (6) NOT NULL,  
    debut DATE NOT NULL) ;
```

34.3.3. Associer un citoyen au service pour lequel il travaille : travaillepour

```
CREATE TABLE travaillepour (
    citoyen NUMERIC (6) NOT NULL,
    service NUMERIC (6) NOT NULL,
    debut DATE NOT NULL,
    fin DATE) ;
```

34.3.4. Associer un service au ministère pour lequel il produit : serviceproduit

```
CREATE TABLE serviceproduit (
    service NUMERIC (6) NOT NULL,
    ministere NUMERIC (6) NOT NULL,
    debut DATE NOT NULL,
    fin DATE) ;
```

34.3.5. Associer un service au ministère auquel il appartient;: serviceappartient

```
CREATE TABLE serviceappartient (
    service NUMERIC (6) NOT NULL,
    ministere NUMERIC (6) NOT NULL,
    debut DATE NOT NULL,
    fin DATE) ;
```

34.3.6. Associer un service au ministère qu'il contrôle : servicecontrole

```
CREATE TABLE servicecontrole (
    service NUMERIC (6) NOT NULL,
    ministere NUMERIC (6) NOT NULL,
    debut DATE NOT NULL,
    fin DATE) ;
```

34.3.7. Associer un citoyen au ministère qu'il dirige : tabledirige

```
CREATE TABLE dirige (
    citoyen NUMERIC (6) NOT NULL,
    ministere NUMERIC (6) NOT NULL,
    debut DATE NOT NULL,
    fin DATE) ;
```

34.4. Les contraintes**34.4.1. Sur la table ville**

```
ALTER TABLE ville ADD CONSTRAINT ville_codepostal_pk
    PRIMARY KEY (codepostal) ;
```

34.4.2. Sur la table rue

```
ALTER TABLE rue ADD CONSTRAINT adresse_numrue_pk
    PRIMARY KEY (numrue) ;
ALTER TABLE rue ADD CONSTRAINT adresse_codepostal_fk
    FOREIGN KEY (codepostal)
    REFERENCES ville (codepostal) ;
```

34.4.3. Sur la table `citoyen`

```
ALTER TABLE citoyen ADD CONSTRAINT citoyen_numuniv_pk
    PRIMARY KEY (numuniv) ;
ALTER TABLE citoyen ADD CONSTRAINT citoyen_sexe_ck
    CHECK (sexe in ('M','F')) ;
ALTER TABLE citoyen ADD CONSTRAINT citoyen_pere_fk
    FOREIGN KEY (pere)
    REFERENCES citoyen (numuniv) ;
ALTER TABLE citoyen ADD CONSTRAINT citoyen_mere_fk
    FOREIGN KEY (mere)
    REFERENCES citoyen (numuniv) ;
```

34.4.4. Sur la table `habite`

```
ALTER TABLE habite ADD CONSTRAINT habite_citoyen_fk
    FOREIGN KEY (citoyen)
    REFERENCES citoyen(numuniv) ;
ALTER TABLE habite ADD CONSTRAINT habite_rue_fk
    FOREIGN KEY (rue)
    REFERENCES rue (numrue) ;
ALTER TABLE habite ADD CONSTRAINT habite_serv_pk
    PRIMARY KEY (citoyen,numero,rue) ;
```

34.4.5. Sur la table `maladie`

```
ALTER TABLE maladie ADD CONSTRAINT maladie_nummaladie_pk
    PRIMARY KEY (nummaladie) ;
ALTER TABLE maladie ADD CONSTRAINT maladie_gravite_ck
    CHECK (gravite in (0,1,2,3,4,5,6,7,8,9,10)) ;
```

34.4.6. Sur la table `listemaladie`

```
ALTER TABLE listemaladie ADD CONSTRAINT listemaladie_cit_moy_deb_pk
    PRIMARY KEY (citoyen,maladie,debut) ;
ALTER TABLE listemaladie ADD CONSTRAINT listemaladie_citoyen_fk
    FOREIGN KEY (citoyen)
    REFERENCES citoyen (numuniv) ;
ALTER TABLE listemaladie ADD CONSTRAINT listemaladie_maladie_fk
    FOREIGN KEY (maladie)
    REFERENCES maladie(nummaladie) ;
```

34.4.7. Sur la table `service`

```
ALTER TABLE service ADD CONSTRAINT service_numservice_pk
    PRIMARY KEY (numservice) ;
```

34.4.8. Sur la table `ministere`

```
ALTER TABLE ministere ADD CONSTRAINT ministere_numministere_pk
    PRIMARY KEY (numministere) ;
```

34.4.9. Sur la table travailleur

```

ALTER TABLE travailleur ADD CONSTRAINT travailleur_cit_serv_pk
    PRIMARY KEY (citoyen, service, debut) ;
ALTER TABLE travailleur ADD CONSTRAINT travailleur_citoyen_fk
    FOREIGN KEY (citoyen)
    REFERENCES citoyen(numuniv) ;
ALTER TABLE travailleur ADD CONSTRAINT travailleur_service_fk
    FOREIGN KEY (service)
    REFERENCES service (numservice) ;

```

34.4.10. Sur la table serviceproduit

```

ALTER TABLE serviceproduit ADD CONSTRAINT serviceproduit_serv_mini_pk
    PRIMARY KEY (service, ministere, debut) ;
ALTER TABLE serviceproduit ADD CONSTRAINT serviceproduit_service_fk
    FOREIGN KEY (service)
    REFERENCES service (numservice) ;
ALTER TABLE serviceproduit ADD CONSTRAINT serviceproduit_ministere_fk
    FOREIGN KEY (ministere)
    REFERENCES ministere (numministere) ;

```

34.4.11. Sur la table serviceappartient

```

ALTER TABLE serviceappartient ADD CONSTRAINT serviceappartient_serv_mini_pk
    PRIMARY KEY (service, ministere, debut) ;
ALTER TABLE serviceappartient ADD CONSTRAINT serviceappartient_service_fk
    FOREIGN KEY (service)
    REFERENCES service (numservice) ;
ALTER TABLE serviceappartient ADD CONSTRAINT serviceappartient_ministere_fk
    FOREIGN KEY (ministere)
    REFERENCES ministere (numministere) ;

```

34.4.12. Sur la table servicecontrole

```

ALTER TABLE servicecontrole ADD CONSTRAINT servicecontrole_serv_mini_pk
    PRIMARY KEY (service, ministere, debut) ;
ALTER TABLE servicecontrole ADD CONSTRAINT servicecontrole_service_fk
    FOREIGN KEY (service)
    REFERENCES service(numservice) ;
ALTER TABLE servicecontrole ADD CONSTRAINT servicecontrole_ministere_fk
    FOREIGN KEY (ministere)
    REFERENCES ministere(numministere) ;

```

34.4.13. Sur la table dirige

```

ALTER TABLE dirige ADD CONSTRAINT dirige_cit_serv_pk
    PRIMARY KEY (citoyen, ministere, debut) ;
ALTER TABLE dirige ADD CONSTRAINT dirige_citoyen_fk
    FOREIGN KEY (citoyen)
    REFERENCES citoyen(numuniv) ;
ALTER TABLE dirige ADD CONSTRAINT dirige_ministere_fk
    FOREIGN KEY (ministere)
    REFERENCES ministere(numministere) ;

```


Chapitre 35. Projet Services

IUP IIES 2^{ème} année et Maîtrise AES-TEG, 2001-2002. Vous pouvez consulter les pages originales¹ sur le web.

Vous allez devoir réaliser un site web dynamique qui permettra de gérer les services des enseignants de l'UFR.

Ce document va vous permettre de définir les caractéristiques de votre site.

35.1. But de ce projet

Le but du site que vous allez réaliser est de permettre au (à la) responsable administratif(ve) d'une UFR de gérer les services des enseignants. Il devra être possible de les enregistrer, les modifier, d'en tirer des récapitulatifs, des synthèses, etc.

Votre projet ne doit pas tenter de gérer les emplois du temps, ni la répartition des cours par semaine.

Les heures d'enseignements ne sont pas toutes comptées de la même manière (elles ne sont pas payées au même prix, suivant que ce sont des TD, des CMTD ou des CM).

Les affectations des enseignants à un cours sont définies au niveau des groupes (s'il y a 5 groupes dans une formation, on doit pouvoir les affecter à 5 enseignants). Mais il n'est pas nécessaire de gérer l'affectation individuelle des groupes (pas besoin de savoir que tel enseignant est en charge du groupe n°1, tel autre du groupe n°2, etc. Il suffit de savoir que tel enseignant a 2 groupes, tel autre en a 1, etc.).

Les enseignants ne doivent pas tous le même service : leur statut donne une base, mais elle peut être modifiée : temps partiel, décharges, contrats particuliers, etc.

35.2. Fonctionnement

35.2.1. Les utilisateurs n'ont pas les mêmes droits suivant leur identité

Les pages que vous allez construire ne seront pas en libre accès : suivant les droits qui lui auront été conférés, chaque utilisateur ne pourra avoir accès qu'à certaines informations et certains traitements.

Il faudra donc gérer un système de mots de passe pour contrôler l'accès au site.

1. *L'utilisateur de base* n'a que le droit de consulter le contenu de la base (et seulement par des requêtes prédéfinies).
2. Le (la) *responsable administratif(ve)* a le droit de modifier le contenu des tables (mais certaines données sensibles peuvent lui être interdites : il(elle) n'est qu'un utilisateur privilégié, avec des droits plus importants que l'utilisateur de base, mais limités quand même).

Les modifications qui lui sont autorisées devront passer, comme pour l'utilisateur de base, par des requêtes prédéfinies.

3. *L'administrateur de la base* a le droit de définir les niveaux de droits des utilisateurs, modifier la structure de la base (il a tous les droits).

35.2.2. Fonctionnalités requises pour la consultation

Consultations accessibles à tous les utilisateurs identifiés.

1. Liste des enseignements (et des enseignants) d'une formation.
2. Liste des enseignements (et des enseignants) d'une année.
3. Liste des enseignements (et des enseignants) d'une UFR.
4. Liste des enseignements (et des enseignants) d'une matière.
5. Liste des enseignements d'un enseignant.

1. <http://grappa.univ-lille3.fr/~gonzalez/enseignement/2001-2002/iies2/projet/>

6. Pour chaque enseignant, calculer son service, et afficher (quand il y en a) le nombre d'heures supplémentaires, ou le nombre d'heures en sous-service.
7. Pouvoir avoir un récapitulatif global des heures d'enseignement, regrouper par section CNU et/ou par formation et/ou par statut.
8. Il faut pouvoir distinguer entre les enseignants permanents et les autres dans les différentes statistiques : quand cela aura un sens il faudra avoir trois versions de chaque calcul : tous les enseignants, les permanents, et les vacataires.

35.2.3. Fonctionnalités requises pour la mise à jour

Pour les utilisateurs autorisés seulement : responsable administratif et administrateur de la base.

1. Créer/modifier/supprimer tout objet intervenant dans la gestion des affectations (UFR, formations, filière, cours, enseignant, etc.).
2. Affecter un enseignement à un enseignant. Supprimer cette affectation.

35.3. Informations plus techniques

1. Un message d'alerte doit être produit quand on a distribué plus de groupes que disponibles.
2. Informations nécessaires pour chaque enseignement : désignation du cours, nombre d'heures, public concerné, type d'enseignement (TD, CM, etc.), nombre de groupes disponibles, section CNU concernée (bien que l'enseignant qui sera chargé du cours ne soit pas obligatoirement rattaché à cette section).
3. Informations nécessaires pour chaque enseignant : nom, prénom, statut administratif, UFR de rattachement, service dû (nombre d'heures), type de contrat (permanent ou vacataire), section CNU.
4. Il serait bon qu'il y ait un historique des commandes exécutées :
 - afin de réparer toute erreur grave,
 - et pour connaître le type d'utilisation de la base.

35.4. Question subsidiaire : comment gérer l'historique de la base ?

Le problème n'est pas de conserver les requêtes (pour cela voir Section 35.3) ou les modifications.

La question est : « Comment garder d'une année à l'autre les renseignements disponibles et exploitables ? »

35.5. Vocabulaire utilisé

Année

Par exemple 2ème année de DEUG AES, 3ème année d'TUP IIES, ...

Cours

Unité d'enseignement. Cela correspond à un certain nombre d'heures de présence devant les étudiants, sur un certain sujet, donnés à un certain nombre de groupes d'étudiants. Plusieurs enseignants peuvent être concernés par le même cours, dans des groupes différents.

Enseignants permanents

Pour une UFR il y a deux catégories d'enseignants : les *permanents* et les *vacataires*. Les premiers dépendent directement de l'UFR (ils y sont nommés à titre *définitif*), mais sont payés par l'université. Les seconds sont recrutés au cas par cas, pour une année (ou un semestre) et sont payés sur les ressources propres de l'UFR (ils peuvent être permanents dans d'autres UFR, ou être complètement extérieurs à l'université).

Formation

Par exemple AES, IUP IIES, MASS, ...

Heures supplémentaires

Heures d'enseignements faites en plus du service dû.

Section CNU

CNU=Conseil National des Universités. C'est l'organisme chargé des affectations des universitaires. Chaque universitaire est qualifié dans une section du CNU (elle dépend de son domaine de recherche : informatique, économie, histoire, etc.), il ne peut être recruté que sur un poste de cette section. On pourra traduire pour ce projet l'expression « section CNU » par « matière enseignée », bien que ce soit en réalité réducteur.

Service

Le service d'un enseignant est le nombre d'heures d'enseignements qu'il assure.

Service dû

Nombre d'heures d'enseignement que doit assurer un enseignant. Ce nombre dépend directement de son statut, mais peut subir des modifications (par exemple un maître de conférences doit en principe 198 heures eqTD, mais une décharge pour services administratifs peut lui permettre d'en faire moins, tandis qu'un contrat pédagogique peut lui demander d'en faire plus).

Sous-service

Se dit quand un enseignant assure moins d'heures que son service dû.

Statut d'un enseignant

Les principaux statuts existants (pour les enseignants titulaires, et accompagnés de leurs services) sont : *professeur* (192 heures), *maître de conférences* (192 heures), *assistant* (192 heures), *PRCE* (384 heures), *PRAG* (384 heures).

TD, CM, CTD

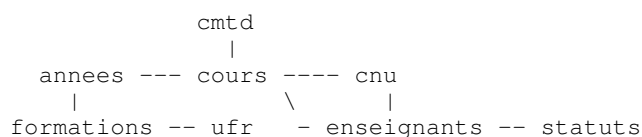
Dans les décomptes des heures d'enseignement on distingue les TD (Travaux Dirigés), les CM (Cours Magistraux), et les CMTD, intermédiaires entre les deux précédents. L'unité de base est l'*heure équivalent TD* (ou eqTD). En principe on a les correspondances suivantes :

- 1 TD=1 eqTD (heureusement !),
- 1 CM=1,5 eqTD,
- 1 CMTD=7/6 eqTD.

Mais cela peut varier.

35.6. Première ébauche de la structure

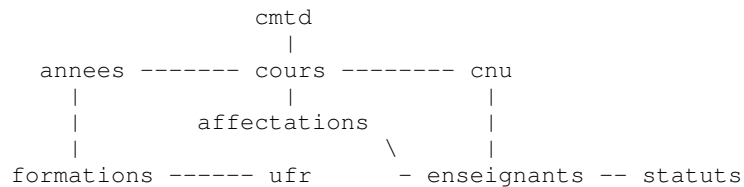
Les entités sont :



Toutes les liaisons, sauf celle entre `cours` et `enseignants`, sont de type 1-n (par exemple une année fait partie d'une seule formation, mais une formation peut correspondre à plusieurs années) : ces liaisons peuvent être réalisées avec une référence externe (par exemple un champ dans la table `annee` contient LE code de LA formation dont cette année dépend, on parle alors de *clef étrangère*).

Mais dans le cas de la liaison entre `cours` et `enseignants` il s'agit d'une relation n-n : chaque cours peut être associé à plusieurs enseignants, chaque enseignant peut avoir à assurer plusieurs cours.

Pour réaliser ce type de liaison on est obligé de faire intervenir une table supplémentaire qui assurera la liaison par deux liaisons 1-n (il s'agit de la table `affectations` dans le schéma suivant) :



35.7. Structures des tables

On trouvera pour chaque table la liste des champs qui la composent.

Les clefs primaires sont suivies d'une étoile (« * »). Les clefs étrangères sont représentées *ainsi*.

35.7.1. `cmt d`

Cette table contient les descriptions des types de cours (TD, CM, CMTD).

- `numcmt d (*)` : identifiant
- `intitule` : intitulé en toutes lettres d'un type de cours (TD, CM, CMTD)
- `coefficient` : valeur en eqTD (équivalent TD) d'une heure d'enseignement

35.7.2. `ufr`

Cette table contient les descriptions des UFR.

- `numufr (*)` : identifiant
- `intitule` : intitulé en toutes lettres de l'UFR (MSES, Philosophie, etc.)

35.7.3. `formations`

Cette table contient les descriptions des formations (AES, IUP IIES, etc.).

- `numformation (*)` : identifiant
- `intitule` : intitulé en toutes lettres de la formation (AES, IUP IIES, etc.)
- `ufr` : code de l'UFR dont dépend cette formation

35.7.4. annees

Cette table contient les descriptions des années (IUP IIES 2^{ème} année, Maîtrise AES, etc.).

- `numannee (*)` : identifiant
- `intitule` : intitulé en toutes lettres d'une année (IUP IIES 2^{ème} année, Maîtrise AES, etc.)
- `niveau` : équivalence en nombre d'années après le bac (3 pour une licence, 1 pour une première année de DEUG, etc.)
- `formation` : code de la formation dont dépend cette année

35.7.5. cnu

Cette table contient les descriptions des différentes section CNU.

- `numcnu (*)` : identifiant
- `intitulé` : intitulé en toutes lettres de la section CNU correspondante

35.7.6. cours

Cette table contient les descriptions des cours (Bases de données partagées, Système et réseaux, Macro-économie, etc.).

- `numcours (*)` : identifiant
- `intitule` : intitulé en toutes lettres du cours (Bases de données partagées, Système et réseaux, Macro-économie, etc.)
- `numannée` : code de la formation dont dépend ce cours
- `numcmt d` : code du type de cours (CMTD, TD, etc.)
- `numcnu` : code de la section CNU à laquelle est rattaché le cours
- `heures` : nombre d'heures d'enseignement par groupe pour ce cours
- `groupes` : nombre de groupes qui sont concernés par ce cours

35.7.7. statut

Cette table contient les descriptions des statuts (Maître de conférences, Professeur, etc.).

- `numstatut (*)` : identifiant
- `intitulé` : intitulé en toutes lettres du statut (Maître de conférences, Professeur, etc.)
- `service` : service dû statutairement (peut ensuite être modifié pour chaque enseignant)

35.7.8. enseignants

Cette table contient les descriptions des enseignants.

- `numenseignant (*)` : identifiant
- `nom` : nom de l'enseignant

- `prenom` : prénom de l'enseignant
- `service` : service dû (exprimé en heures)
- `numstatut` : code du statut de l'enseignant
- `numufr` : code de l'UFR dont dépend l'enseignant

35.7.9. affectations

Cette table sert à représenter la liaison d'affectation existant entre les cours et les enseignants.

- `numcours (*)` : code du cours concerné par cette affectation
- `numenseignant (*)` : code de l'enseignant concerné par cette affectation
- `nbgroupe` : nombre de groupes concernés par cette affectation

35.8. Création des tables

35.8.1. cmt_d

```
CREATE TABLE cmt_d (  
    numcmt_d NUMERIC(7) NOT NULL PRIMARY KEY,  
    intitule TEXT,  
    valeur NUMERIC(6, 4) ) ;
```

35.8.2. ufr

```
CREATE TABLE ufr (  
    numufr NUMERIC(7) NOT NULL PRIMARY KEY,  
    intitule TEXT) ;
```

35.8.3. formations

```
CREATE TABLE formations (  
    numformation NUMERIC(7) NOT NULL PRIMARY KEY,  
    intitule TEXT,  
    numufr NUMERIC(7) NOT NULL,  
    FOREIGN KEY (numufr) REFERENCES ufr (numufr)) ;
```

35.8.4. annees

```
CREATE TABLE annees (  
    numannee NUMERIC(7) NOT NULL PRIMARY KEY,  
    intitule TEXT,  
    niveau INT4,  
    numformation NUMERIC(7) NOT NULL,  
    FOREIGN KEY (numformation) REFERENCES formations (numformation)) ;
```

35.8.5. cnu

```
CREATE TABLE cnu (
    numcnu NUMERIC(7) NOT NULL PRIMARY KEY,
    intitulé TEXT) ;
```

35.8.6. cours

```
CREATE TABLE cours (
    numcours NUMERIC(7) NOT NULL PRIMARY KEY,
    intitule TEXT,
    numannee NUMERIC(7) NOT NULL,
    numcmtid NUMERIC(7) NOT NULL,
    numcnu NUMERIC(7) NOT NULL,
    heures NUMERIC (4,2),
    groupes NUMERIC (4,2),
    FOREIGN KEY (numcmtid) REFERENCES cmtid (cnumcmtid),
    FOREIGN KEY (numcnu) REFERENCES cnu (cnumcnu),
    FOREIGN KEY (numannee) REFERENCES anneess (numannee)) ;
```

35.8.7. statut

```
CREATE TABLE statuts (
    numstatut NUMERIC(7) NOT NULL PRIMARY KEY,
    intitulé TEXT,
    service NUMERIC (6,2)) ;
```

35.8.8. enseignants

```
CREATE TABLE enseignants (
    numenseignant NUMERIC(7) NOT NULL PRIMARY KEY,
    nom TEXT,
    prenom TEXT,
    service NUMERIC (6,2),
    numstatut NUMERIC(7) NOT NULL,
    numufr NUMERIC(7) NOT NULL,
    FOREIGN KEY (numufr) REFERENCES ufr (cnumufr),
    FOREIGN KEY (numstatut) REFERENCES statuts (numstatut)) ;
```

35.8.9. affectations

```
CREATE TABLE affectations (
    numcours NUMERIC(7) NOT NULL,
    numenseignant NUMERIC(7) NOT NULL,
    groupes INT4,
    PRIMARY KEY (numcours,numenseignant),
    FOREIGN KEY (numcours) REFERENCES cours (cnumcours),
    FOREIGN KEY (numenseignant) REFERENCES enseignants (numenseignant)) ;
```

35.9. Remplir les tables de paramètres**35.9.1. statut**

```
INSERT INTO statuts VALUES (1,'MdC',192.0);
INSERT INTO statuts VALUES (2,'Professeur',192);
INSERT INTO statuts VALUES (3,'PRAG',384);
INSERT INTO statuts VALUES (4,'ATER',192);
```

```
INSERT INTO statuts VALUES (5,'Chargé de cours',0);
INSERT INTO statuts VALUES (6,'PAST',96);
```

35.9.2. cnu

```
INSERT INTO cnu VALUES (1,
    'Droit privé et sciences criminelles');
INSERT INTO cnu VALUES (2,
    'Droit public');
INSERT INTO cnu VALUES (3,
    'Histoire du droit et des institutions');
INSERT INTO cnu VALUES (4,
    'Science politique');
INSERT INTO cnu VALUES (5,
    'Sciences économiques');
INSERT INTO cnu VALUES (6,
    'Sciences de gestion');
INSERT INTO cnu VALUES (7,
    'Sciences du langage : linguistique et phonétique '
    ||'générales');
INSERT INTO cnu VALUES (8,
    'Langues et littératures anciennes');
INSERT INTO cnu VALUES (9,
    'Langue et littérature françaises');
INSERT INTO cnu VALUES (10,
    'Littératures comparées');
INSERT INTO cnu VALUES (11,
    'Langues et littératures anglaises et anglo-saxonnes');
INSERT INTO cnu VALUES (12,
    'Langues et littératures germaniques et scandinaves');
INSERT INTO cnu VALUES (13,
    'Langues et littératures slaves');
INSERT INTO cnu VALUES (14,
    'Langues et littératures romanes : espagnol, italien, '
    ||'portugais, autres langues romanes');
INSERT INTO cnu VALUES (15,
    'Langues et littératures arabes, chinoises, japonaises, '
    ||'hébraïques, d\'autres domaines linguistiques');
INSERT INTO cnu VALUES (16,
    'Psychologie, psychologie clinique, psychologie sociale');
INSERT INTO cnu VALUES (17,
    'Philosophie');
INSERT INTO cnu VALUES (18,
    'Arts : plastiques, du spectacle, musique, musicologie, '
    ||'esthétique, sciences de l\'art');
INSERT INTO cnu VALUES (19,
    'Sociologie, démographie');
INSERT INTO cnu VALUES (20,
    'Anthropologie, ethnologie, préhistoire');
INSERT INTO cnu VALUES (21,
    'Histoire et civilisations : histoire et archéologie des '
    ||'mondes anciens et des mondes médiévaux; de l\'art');
INSERT INTO cnu VALUES (22,
    'Histoire et civilisations : histoire des mondes modernes; '
    ||'histoire du monde contemporain; de l\'art; de la musique');
INSERT INTO cnu VALUES (23,
    'Géographie physique, humaine, économique et régionale');
INSERT INTO cnu VALUES (24,
    'Aménagement de l\'espace, urbanisme');
INSERT INTO cnu VALUES (25,
    'Mathématiques');
INSERT INTO cnu VALUES (26,
    'Mathématiques appliquées et applications des mathématiques');
INSERT INTO cnu VALUES (27,
    'Informatique');
INSERT INTO cnu VALUES (28,
    'Milieux denses et matériaux');
INSERT INTO cnu VALUES (29,
    'Constituants élémentaires');
```

```

INSERT INTO cnu VALUES (30,
    'Milieux dilués et optique');
INSERT INTO cnu VALUES (31,
    'Chimie théorique, physique, analytique');
INSERT INTO cnu VALUES (32,
    'Chimie organique, minérale, industrielle');
INSERT INTO cnu VALUES (33,
    'Chimie des matériaux');
INSERT INTO cnu VALUES (34,
    'Astronomie, astrophysique');
INSERT INTO cnu VALUES (35,
    'Structure et évolution de la terre et des autres planètes');
INSERT INTO cnu VALUES (36,
    'Terre solide : géodynamique des enveloppes supérieures, '
    || 'paléobiosphère');
INSERT INTO cnu VALUES (37,
    'Météorologie, océanographie physique et physique de '
    || 'l\'environnement');
INSERT INTO cnu VALUES (39,
    'Sciences physico-chimiques et technologies pharmaceutiques');
INSERT INTO cnu VALUES (40,
    'Sciences du médicament');
INSERT INTO cnu VALUES (41,
    'Sciences biologiques');
INSERT INTO cnu VALUES (60,
    'Mécanique, génie mécanique, génie civil');
INSERT INTO cnu VALUES (61,
    'Génie informatique, automatique et traitement du signal');
INSERT INTO cnu VALUES (62,
    'Energétique, génie des procédés');
INSERT INTO cnu VALUES (63,
    'Electronique, optronique et systèmes');
INSERT INTO cnu VALUES (64,
    'Biochimie et biologie moléculaire');
INSERT INTO cnu VALUES (65,
    'Biologie cellulaire');
INSERT INTO cnu VALUES (66,
    'Physiologie');
INSERT INTO cnu VALUES (67,
    'Biologie des populations et écologie');
INSERT INTO cnu VALUES (68,
    'Biologie des organismes');
INSERT INTO cnu VALUES (69,
    'Neurosciences');
INSERT INTO cnu VALUES (70,
    'Sciences de l\'éducation');
INSERT INTO cnu VALUES (71,
    'Sciences de l\'information et de la communication');
INSERT INTO cnu VALUES (72,
    'Epistémologie, histoire des sciences et des techniques');
INSERT INTO cnu VALUES (73,
    'Cultures et langues régionales');
INSERT INTO cnu VALUES (74,
    'Sciences et techniques des activités physiques et sportives');
INSERT INTO cnu VALUES (7501,
    'Théologie catholique');
INSERT INTO cnu VALUES (7502,
    'Théologie protestante');

```

35.9.3. cmd

```

INSERT INTO cmd VALUES (1, 'CM', 1.5);
INSERT INTO cmd VALUES (2, 'TD', 1);

```


Chapitre 36. Projet *Disques*

Maîtrise AES-TEG, 2000-2002. Vous pouvez consulter les pages originales¹ sur le web.

36.1. Approche naïve

On souhaite mémoriser le contenu d'une discothèque dont le but est le prêt de disques (CD ou vinyl) à ses adhérents.

Il faut connaître les œuvres, les exemplaires de chaque œuvre, les artistes correspondants, les renseignements sur les adhérents.

Pour ce faire la première solution consisterait en un (gros) tableau permettant de stocker toutes les informations dont on dispose sur un album : *nom de l'album*, *interprète*, *morceaux* (plusieurs), le *genre de musique*, le *type de support* (vinyl, CD), etc... ainsi que celles dont on dispose sur les adhérents.

La représentation de ces informations « à plat » sous la forme d'un seul tableau montre d'emblée ses limites :

- En effet, on ne va pas reprendre systématiquement toutes les informations liées à un album du fait que la colonne des titres contient plusieurs titres pour un même album.
- Par ailleurs, cette disposition privilégie l'approche de ces informations album par album. Et par conséquent, elle est moins commode pour une approche par interprète, par emprunteur.

Avec cette solution on a donc des difficultés à percevoir séparément des éléments (entités) qui sont pourtant bien distinctes (albums, interprètes, adhérents).

On a par la même occasion, des difficultés à regrouper certaines informations : les albums d'un même interprète, les albums d'un même genre, les albums empruntés par un même adhérent, etc. On dispose bien dans un tableau de la possibilité de trier un tableau suivant telle ou telle colonne mais là on se heurte à un problème de choix (il n'y a pas de raison de privilégier un ordre par rapport à un autre).

Par ailleurs, dès lors qu'un tableau est trié, il s'agit de respecter cet ordre lors de toute mise-à-jour du tableau, ce qui complique la procédure de saisie de nouveaux albums.

Même si le tableau devait être trié, la recherche d'une information précise demeurerait assez fastidieuse, cette recherche devient encore moins aisée s'il s'agit de retrouver des croisements d'information (X a-t-il fait une reprise d'un titre déjà interprété par Y). La difficulté se corse encore lorsqu'il ne s'agit plus d'accéder à une information mais d'extraire une série d'informations (les albums de *Jean Ferrat* qui contiennent des textes d'*Aragon*).

On remarque également que, dans une telle représentation, de nombreuses informations figurent plusieurs fois (artiste : nom et prénom ; emprunteur : adresse ...). Ces informations étant saisies individuellement, outre la perte d'efficacité, il existe des risques d'incohérence (une même donnée saisie différemment à deux endroits distincts).

La répétition d'une même donnée peut être également pénalisante si celle-ci vient à changer, en effet au lieu d'avoir à procéder à une seule mise-à-jour, on est obligé de reporter la même modification partout où cette donnée figure (là encore perte d'efficacité et risque d'incohérence). Par exemple que faire lorsqu'un chanteur décide de changer de nom² ?

36.2. Analyse

Quelles entités peut-on identifier ? Elles sont au nombre de 5 :

Les *Disques*

L'entité centrale de notre base de données. Il s'agit de l'œuvre « abstraite », comme on parle de *l'album blanc des Beatles*, sans se référer à un exemplaire précis, mais seulement à l'œuvre elle-même. Les propriétés d'un disque seront :

- son intitulé,

1. http://grappa.univ-lille3.fr/~gonzalez/enseignement/2000-2001/mait_aes/projet

2. Idée bizarre, je sais... Mais c'est quand même ce qu'a fait *Prince*. (Ce qui n'enlève d'ailleurs rien à la bizarrerie du fait...)

- son année de parution,
- les morceaux qui la composent.
- son genre de musique (rock, classique, etc...)

Les Artistes

Qui a enregistré tel disque ? Les propriétés d'un artiste seront :

- son nom,
- son prénom,
- éventuellement le groupe qui l'accompagne,

Les Exemplaires

L'entité *Exemplaires* est totalement différente de l'entité *Disques*. Il s'agit de l'objet physique, l'exemplaire physique d'un disque, par exemple l'exemplaire en vinyl de *l'album blanc des Beatles* qui se trouve dans la discothèque de votre grand-mère. Il peut y avoir plusieurs exemplaires d'un même disque. Les propriétés d'un exemplaire seront :

- son type de support (CD ou vinyl),
- son prix (pour remboursement en cas de perte ou de détérioration).

Les Adhérents

Il s'agit des personnes inscrites au club. Les propriétés d'un adhérent seront :

- son nom,
- son prénom
- son adresse³,
- la date du début de son adhésion,
- le nombre de personnes qui auront accès aux disques empruntés par cet adhérent⁴ pour pouvoir faire quelques statistiques sur la pénétration des disques en stock dans le public.

Les Prêts

Tel adhérent emprunte tel exemplaire⁵. Les propriétés d'un prêt seront :

- sa date de début (date de sortie de l'exemplaire),
- sa date de fin (date de rentrée de l'exemplaire).

Les relations existant entre ces entités sont décrites dans la figure ci-dessous :

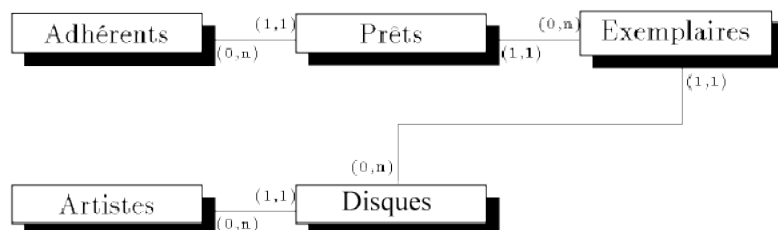


Figure 36-1. Relations dans la base *Disques*, première version

On remarque assez vite que ces choix vont amener des difficultés :

- Si on fait figurer les villes dans chaque fiche d'adhérent, on crée une *redondance* des données et un risque d'*inconsistance*. On va donc créer une nouvelle entité : *Villes*. Les propriétés d'une ville seront :

3. Par un pur souci de simplification de la base, nous nous sommes un peu éloignés de la réalité en décidant de ne conserver que la ville de résidence pour l'adresse d'un adhérent. Ce choix serait tout à fait mauvais dans la réalité.

4. Ce sera souvent le nombre de personnes vivant au même domicile que lui.

5. Une autre analyse aurait pu amener à considérer *Prêts* comme une relation entre *Exemplaires* et *Adhérents*, et non comme une entité propre. La discussion peut être très longue sur ce sujet. Mais de toutes façons, le résultat aurait été identique, ou presque...

- son nom,
 - son code postal.
- Même remarque pour le genre musical des disques. On va donc créer une nouvelle entité *Genres*. La seule propriété d'un genre sera son nom.
 - Garder les titres des morceaux composant un disque avec les renseignements concernant le disque lui-même pose un autre problème : quelle place réserver ? Si on réserve peu de place, on sera vite débordé. Si on réserve beaucoup de place (par exemple de quoi ranger 40 titres) on perdra une place énorme qui sera réservée pour tous les disques, y compris ceux qui ne contiennent qu'une dizaine de morceaux (ce qui est le cas pour la plupart des disques) ; et il pourra quand même exister des disques qui aient plus de morceaux que ce que vous aviez réservé⁶. Un autre problème résultant du choix d'une telle structure : elle rend presque impossible la recherche d'un morceau particulier, dont on ne connaît que le titre, mais ni l'interprète, ni le disque.

Nous allons donc créer une nouvelle entité pour « éclater » les données : *chansons*. Les propriétés d'une chanson seront :

- son titre,
- son numéro d'ordre sur le disque.

Les relations existant entre ces entités sont décrites dans la figure ci-dessous :

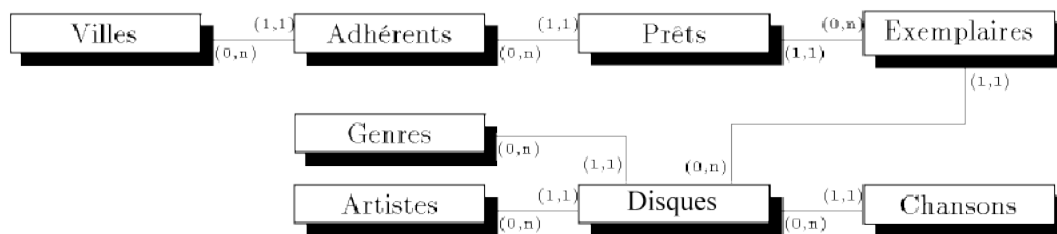


Figure 36-2. Relations dans la base *Disques*, deuxième version

Il n'est plus question maintenant, par exemple, de reporter sur la ligne correspondant à un exemplaire toutes les informations relatives au disque. On placera donc, sur la ligne correspondante, une information particulière à partir de laquelle on pourra retrouver, dans le tableau correspondant, ce qui concerne le disque (ce type d'information est appelé *clef étrangère*).

Mais, a priori on ne dispose pas de clef pour les disques. Par contre, on dispose de ce type d'information pour les villes, plus exactement les bureaux distributeurs : le code postal. Ainsi, on pourra se contenter au niveau de l'adresse d'une personne du bureau distributeur dès lors que l'on dispose d'un tableau (*codePostal*, *bureauDistributeur*).

Reprenons cette idée du code (du numéro, du matricule...) pour nos albums, nos artistes... Cela fait partie de notre vie : vous avez un numéro INSEE, un numéro de carte d'étudiant, un numéro de dossier pour votre organisme logeur, un numéro de compte en banque, les livres ont un numéro ISBN et une cote s'ils appartiennent à une bibliothèque, les voitures ont un numéro minéralogique, les produits du commerce des « codes à barres », etc.

Pour les entités qui n'ont pas de clef « naturelle », nous devons en créer une de toutes pièces.

36.3. Dernières remarques

On a fait certaines approximations dans l'analyse de cette base.

On n'a par exemple pas pris en compte le fait qu'un artiste peut enregistrer un album seul ou à l'intérieur d'un groupe. Cette information n'est pas gérée dans la base. Il est, par exemple, impossible de trouver dans

6. Tout le monde sait que, d'après les axiomes de \mathbb{N} (ensemble des entiers naturels), quelque soit l'entier choisi, il en existera toujours un plus grand. Ainsi vos connaissances musicales (étendues, j'en suis sûr) vont vous laisser supposer que 40 titres, comme proposé, c'est déjà beaucoup. Eh non! *The commercial album* des *Residents* contient 50 morceaux.

la base le moindre lien entre les disques des *Rolling Stones* et ceux de *Mick Jagger* ou de *Keith Richard*. Même remarque pour les *Beatles* et *John Lennon*.

On a également fait de grosses approximations (si on se réfère au réel) sur les adresses : il a déjà été mentionné que l'on ne garde que la ville; on n'a pas non plus tenu compte que certaines villes n'ont pas de bureau distributeur propre; ce cas a été volontairement *oublié* pour conserver au champ `CodePostal` dans la table `Villes` sa fonction de clef.

Toutes ces approximations ont été faites en général dans un but pédagogique, afin de ne pas surcharger la base et pour en rendre la compréhension plus aisée.

Il faut bien se souvenir que l'analyse d'un système informatique n'est pas un travail objectif mais qu'elle dépend surtout de la vision (plus ou moins claire, souvent) qu'en ont ses futurs utilisateurs. Quand un *informaticien* réalise une analyse pour le compte de quelqu'un d'autre, c'est à lui de détecter les problèmes éventuels et de les signaler au *client*. Cependant ce dernier peut parfaitement faire, en toute connaissance de cause, des choix qui paraissent réducteurs, pour peu que les *oublis* ainsi commis ne concernent que des informations qui ne l'intéressent pas.

Le gros problème dans un tel cas est un éventuel changement d'avis dans le futur. Il serait alors sans doute impossible d'adapter la solution choisie.

Mieux vaut donc une sur-abondance raisonnable⁷ d'informations qu'on n'utilise pas pour l'instant, plutôt qu'un *oubli* d'informations dont on pourrait avoir besoin plus tard.

36.4. Une autre analyse du projet *Disques*

Vous trouverez dans les lignes qui suivent une approche fondamentalement différente de l'analyse faite dans le chapitre précédent.

Cette analyse ne sera pas prise en compte dans le reste du projet. Elle vous est seulement présentée pour vous montrer que le même problème peut amener à deux structures complètement différentes suivant l'approche qu'on en a.

36.4.1. Description

Un disque est composé de plages musicales. Pour avoir une base performante, on désire distinguer les *interprétations* différentes d'un même titre.

Par exemple il faut différencier

- *String Quartet N°12* de *D.Schostakowitsch* sur le disque *STRING QUARTETS Nos 1, 9 & 12* enregistré le 27/09/81 par le *Borodin String Quartet*
- et *String Quartet N°12* de *D. Schostakowitsch* sur le disque *GIDON KREMER Ed.Lockenhaus Vol 4/5* enregistré le 02/07/86 par *G. Kremer, Y. Horigome, K. Kashkashian, D. Geringas*

Les informations suivantes sont à mémoriser : le titre du disque, ses références d'édition, son code barre, les titres qui le composent ; puis pour chaque titre, le compositeur, la date d'enregistrement, le lieu d'enregistrement, le ou les interprètes et la durée de l'interprétation. Pour les compositeurs et les interprètes, on ne mémorise que le nom.

On adoptera les choix suivants :

- Les références d'édition sont celles que donne l'éditeur au disque (exemple : *ECM New Serie 736943*). On ne s'occupe pas de l'éditeur.
- Le compositeur d'un titre est considéré comme étant unique. S'il y avait plusieurs compositeurs (ce qui est relativement rare), ils seraient notés comme un nom unique et considérés comme un groupe.

Exemple 36-1. Un groupe

String Quartet N°12 composé par *D. SCHOSTAKOWITSCH* ou *The Black Angel's Death Song* composé par le groupe *REED/CALE*.

Le compositeur peut donc être soit un individu, soit un groupe.

7. Le problème étant de bien définir le mot *raisonnable*.

- Les interprètes varient selon les interprétations. Les interprètes peuvent être : soit le compositeur (qui est alors un interprète comme un autre), soit un groupe, soit un regroupement d'interprètes, soit un groupe et des interprètes invités... Dans tous les cas, seuls le ou les noms (du groupe, des interprètes s'il s'agit d'un regroupement d'interprètes) nous intéressent. Puis, dans une recherche plus approfondie, les membres du groupe pourront être demandés.

Vous pouvez donc considérer un groupe comme un interprète particulier.

Exemple 36-2. Un groupe avec interprète invité

The Black Angel's Death Song, par *THE VELVET UNDERGROUND* et *NICO*, *THE VELVET UNDERGROUND* étant un groupe (à ce niveau, peu importe qui composait le groupe) et *NICO* étant une interprète invitée.

Exemple 36-3. Interprète et compositeur

Riding The Westerleys composé et interprété par *TERRY RILEY*.

- Une même interprétation peut apparaître sur plusieurs disques.
- On ne tiendra pas compte des différents mouvements à l'intérieur d'une composition (classique).
- La probabilité pour qu'un même titre soit enregistré par des interprètes différents, le même jour, sera considérée comme nulle. L'égalité possible des lieux et des dates d'enregistrements ne sera pas considérée comme une redondance.

Il sera intéressant de savoir quels sont les membres d'un groupe. Pour cela, on mémorisera à partir de quelle date un interprète appartient à un groupe et à quelle date on suppose qu'il le quitte. On supposera que le programme gère la validité de ces dates.

36.4.2. Résultat

On obtient le schéma conceptuel de la figure suivante :

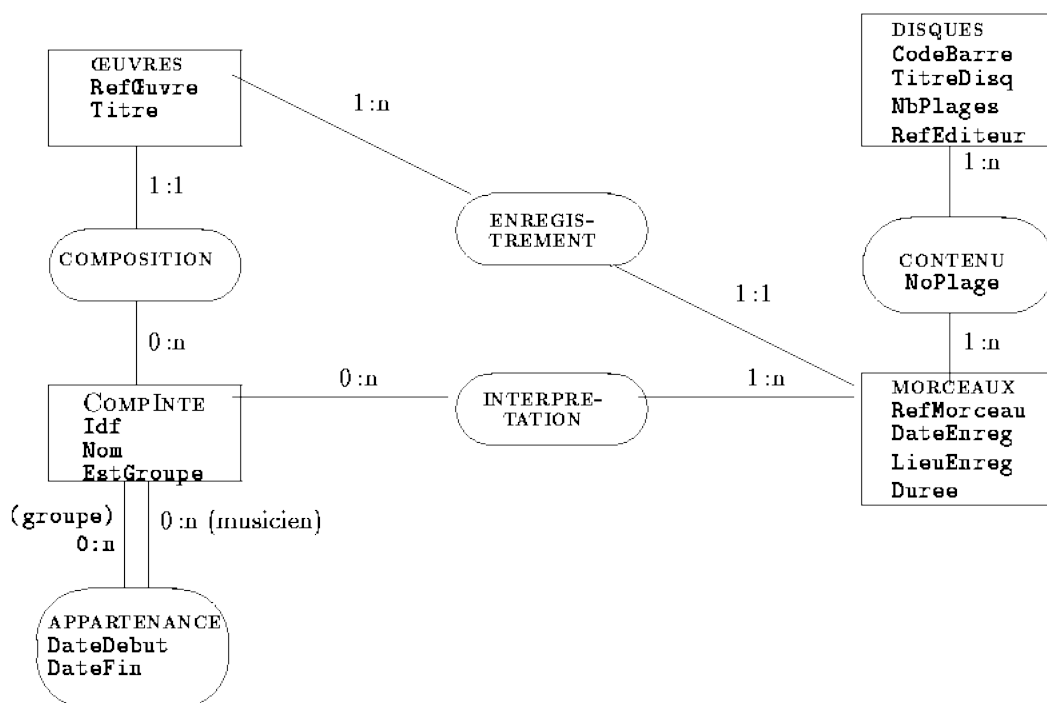


Figure 36-3. Un autre MCD pour la base *Disques*

Les tables sont :

- La table `Disques` qui contient le code barre, le titre du disque, le nombre de plages et les références de l'éditeur.
- La table `Morceaux` qui correspond aux interprétations. Elle contient les informations associées à l'enregistrement et la durée. Une même interprétation peut apparaître sur plusieurs disques.
- La table `Ouvres` qui correspond à la création du compositeur.
- La table `CompInte` (nom donné pour `Compositeurs` et `Interprètes`) contient les compositeurs et les interprètes. En effet, un compositeur pouvant être un interprète et les informations mémorisées dans les deux cas étant les mêmes, on peut regrouper les deux entités en une seule, évitant ainsi de dupliquer les informations pour tous les compositeurs-interprètes.

De plus cette table contient des fiches correspondant aussi bien à des groupes qu'à des individus.

Quand une œuvre est interprétée par un groupe, on trouvera la fiche du groupe et, le plus souvent possible, les fiches des interprètes qui appartiennent au groupe. La fiche d'une interprétation, si elle est interprétée par un groupe, sera de préférence reliée au groupe plutôt qu'aux membres du groupe. On trouvera les membres du groupe par la relation `appartenance`.

Cela permettra de traiter tous les cas possibles ; en particulier, il n'y a aucun problème pour une interprétation par un groupe invitant un interprète. Il n'y a aucune difficulté pour enregistrer un groupe sans connaître les membres, pour ajouter un membre à un groupe existant, pour définir une interprétation effectuée par un seul membre du groupe...

On ajoute un champ logique, appelé *EstGroupe*, qui permettra de différencier les individus et les groupes. Il contiendra la valeur `vrai` si la fiche est celle d'un groupe, la valeur `faux` s'il s'agit d'un individu.

`Appartenance` est une relation entre la table `CompInte`, vue comme contenant les fiches des groupes, et la table `CompInte`, vue comme contenant les fiches des individus. A cette relation sont associées des informations : les dates de début et de fin d'appartenance à un groupe. Un élément de cette relation indique qu'un individu, interprète dont la fiche est dans la table `CompInte`, a appartenu à un groupe, dont la fiche est aussi dans la table `CompInte`. Les dates seront gérées de telle façon que la base reste cohérente. Un interprète référencé sur une fiche de la table `Membres` est supposé avoir participé à tous les enregistrements effectués par le groupe référencé, entre la date de début et la date de fin d'appartenance à ce groupe.

36.5. Création des tables

Pour la garnir de données vous allez pouvoir laisser faire un fichier qui fera tout à votre place. Pour cela déconnectez-vous de PostgreSQL (\q) et reconnectez-vous par la commande suivante :

```
psql -f /home/enseign/aesteg/creerdisques.sql
```

qui va faire exécuter par PostgreSQL le fichier `creerdisques.sql` contenant les commandes et les données nécessaires au remplissage de la base.

Si vous avez quelques messages d'erreur, ne vous en faites pas trop à condition que le programme arrive à son terme.

Les tables ont été créées par les commandes suivantes :

- La table `artistes` :

```
CREATE TABLE artistes (  
    codeartiste INT4 PRIMARY KEY,  
    prenom VARCHAR,  
    nom VARCHAR,  
    groupe VARCHAR) ;
```

- La table `disques` : l'œuvre logique, comme on parle de *l'album blanc des Beatles*, sans se référer à un exemplaire précis, mais seulement à l'œuvre elle-même. La plupart des renseignements se trouvent dans d'autres tables auxquelles on aura accès grâce aux clefs étrangères

```
CREATE TABLE disques (  
    codedisque INT4 PRIMARY KEY,  
    artiste INT4,  
    annee VARCHAR,  
    intitule VARCHAR,
```

```
genre INT4) ;
```

- la table `chansons` : une fiche correspond à une chanson dans un disque. On connaît le disque par son code.

```
CREATE TABLE chansons (
    numero INT4,
    disque INT4,
    titre VARCHAR,
    PRIMARY KEY (numero, disque)) ;
```

- la table `genres` :

```
CREATE TABLE genres (
    codegenre INT4 PRIMARY KEY,
    genre VARCHAR) ;
```

- la table `prets` : une fiche correspond à un emprunt d'un exemplaire.

```
CREATE TABLE prets (
    codepret INT4 PRIMARY KEY,
    exemplaire INT4,
    adherent INT4,
    sortie DATE,
    retour DATE) ;
```

- la table `villes` :

```
CREATE TABLE villes (
    codepostal VARCHAR PRIMARY KEY,
    ville VARCHAR) ;
```

- la table `adherents` :

```
CREATE TABLE adherents (
    codeadherent INT4 PRIMARY KEY,
    nom VARCHAR,
    prenom VARCHAR,
    dateadhesion date,
    codepostal VARCHAR,
    nbrepersonnes INT4) ;
```

- La table `exemplaires` : un exemplaire physique d'une œuvre, par exemple l'exemplaire en vinyl de l'*album blanc des Beatles* qui se trouve dans la discothèque de votre grand-mère. Chaque exemplaire peut être sur vinyl ou CD. Son prix est indiqué pour pouvoir établir une facture en cas de perte.

```
CREATE TABLE exemplaires (
    codeexemplaire INT4 PRIMARY KEY,
    disque INT4,
    CDouVinyl VARCHAR,
    prix FLOAT4) ;
```

36.6. Votre travail

Vous allez devoir réaliser le site web de cette discothèque.

Vous êtes totalement libre pour le choix du contenu de ce site.

Les seules choses importantes sont :

- Votre site devra bien entendu utiliser `PHP` et `PostgreSQL` (c'est quand même LE projet du cours !). Un ou deux formulaires ne seraient pas trop mal vus...
- Vous serez noté sur l'état de votre site à la fin du dernier cours. Mais il y aura un contrôle oral sur son contenu (en d'autres mots : je vérifierai que vous êtes bien l'auteur de ce que vous me présenterez...).
- Faites le mieux de ce que vous pouvez. Il n'y a pas de minimum à atteindre.
- L'ergonomie et la facilité d'utilisation du site seront prises en compte dans la notation.
- Vous pouvez travailler par groupe de deux.

Vous pourrez trouver un exemple⁸ de ce à quoi cela peut ressembler sur le web.

8. http://grappa.univ-lille3.fr/~gonzalez/enseignement/2000-2001/mait_aes/disques

Index

action, 17
ADODB, 29, 47, 53
affectation, 10, 22
affectedRows(), 51
algorithmique, 1, 10
alternative, 1, 10
analyse, 146
apostrophe, 21
archive, 47
assembleur, 3
backslash, 21
base de données, 43
boucle POUR, 10
bouton
 d'envoi, 15
 radio, 15
 reset, 16
 submit, 16
case à cocher, 15
champs, 15
chaînes de caractères, 21
 opération sur les -, 22
checkbox, 15
chop, 22
clef étrangère, 145
closedir, 57
concaténation, 10, 22
connexion à la base de données, 47
contrôle des types, 10
conversion
 de types, 23
 majuscules-minuscules, 22
 minuscules-majuscules, 22
couche d'abstraction, 47
count, 27
DBX, 29, 53
DB_FETCHMODE_ASSOC, 50
DB_FETCHMODE_ORDERED, 49, 50
DELETE, 45, 46
DocBook, 1
droits d'accès, 37
DSN, 47
echo, 9
emacs, 1
en-tête HTTP, 35
entité, 143
erreur, 48
explode, 22
FAQ, 7
fclose, 61
fermeture d'un fichier, 61
fetchInto(), 49
fetchRow(), 49
fgets, 61, 62
fichier, 61
 fermeture, 61
 ouverture, 61
file, 61, 62
fonction, 11
 chop, 22
 closedir, 57
 echo, 9
 explode, 22
 fclose, 61
 fgets, 61, 62
 file, 61, 62
 fopen, 61
 fputs, 61
 fwrite, 61
 gettype, 23
 header, 35
 implode, 22
 include, 33
 join, 22
 ltrim, 22
 opendir, 57
 pg_affected_rows, 46
 pg_connect, 43
 pg_fetch_array, 44
 pg_numrows, 44
 pg_query, 43, 45, 45
 pg_result_status, 46
 phpinfo, 9
 readdir, 57
 require, 33
 sort, 63
 split, 22
 strlen, 22
 strpos, 23
 strtolower, 22
 strtoupper, 22
 str_replace, 23
 substr, 23, 57
 substr_replace, 23
 trim, 22
fopen, 61
for, 10
foreach, 27
formulaire, 15, 19
fputs, 61
fwrite, 61
GEDCOM, 120
gestion des erreurs, 48
GET, 15, 16, 19
gettype, 23
\$_GET, 19
guillemet, 21
header, 35
hidden, 16
HTML, 1, 1
\$HTTP_GET_VARS, 19
\$HTTP_POST_VARS, 19
hébergeur, 29, 47, 53
identification, 35, 39
if, 10
images cliquables, 57
implode, 22
include, 33
informations, 51
input, 15
INSERT, 45, 46
interpréteur, 3, 7
itération, 1, 10
JDN, 47
join, 22
Journal du Net, 47

- langage
 - compilé, 3
 - de scripting, 7
 - du web, 1
 - informatique, 1, 3
 - intermédiaire, 4
 - interprété, 3
 - machine, 3
 - naturel, 3
- licence GNU FDL, 3
- longueur d'une chaîne, 22
- ltrim, 22
- MCD, 121
- messages d'erreurs, 45
- MetaData, 29, 53
- MetatData, 47
- MLD, 100, 102, 107, 112, 116
- MySQL, 29, 47, 53
- numCols(), 51
- numRows(), 51
- opendir, 57
- openjade, 1
- opérateur, 10
 - de comparaison, 11
 - de concaténation, 10, 22
 - logique, 11
- opérations sur les chaînes, 22
- ouverture d'un fichier, 61
- page web dynamique, 5
- password, 15
- PDF, 1, 1
- PDO, 29, 29
- pear, 47
- Pear-DB, 29, 53
- peardb, 47
- pg_affected_rows, 46
- pg_connect, 43
- pg_fetch_array, 44
- pg_last_error, 45
- pg_numrows, 44
- pg_query, 43, 45
- pg_result_status, 46
- phpinfo, 9
- PHPLib, 29, 47, 53
- \$PHP_AUTH_PW, 35
- \$PHP_AUTH_USER, 35
- \$PHP_AUTH_USER, 35
- point-virgule, 10
- portable, 3
- POST, 15, 16, 19
- PostgreSQL, 29, 43, 45, 47, 53
- \$_POST, 19
- procédure, 11
- programmation séparée, 33
- programmer, 1
- projet
 - Association, 115
 - Brazil, 127
 - Camping, 105, 111
 - Disques, 143
 - Disques 2009, 97
 - Généalogie, 119
 - Inscriptions, 101
 - Services, 133
- protocole, 3
- Quanta, 1
- radio, 16
- readdir, 57
- require, 33
- require_once, 47
- requête, 49
- reset, 16
- return, 11
- répertoire, 57
- safe mode, 19, 35
- saisie de mot de passe, 15
- select, 16
- \$_SERVER, 35
- sessions, 39
- setfetchmode, 49
- SGBD, 7
- sort, 63
- split, 22
- SQL, 7
- SQLite, 29, 47, 53
- string, 21
- strlen, 22
- strpos, 23
- strtolower, 22
- strtoupper, 22
- structures de contrôle, 1, 10
- str_replace, 23
- submit, 16
- substr, 23, 57
- substr_replace, 23
- séparation du contenu et du traitement, 5
- séquence, 1, 10
- tableau, 25
 - associatif, 26
- tableInfo(), 51
- TANT QUE, 10
- text, 15
- textarea, 16
- trim, 22
- UPDATE, 45, 46
- variable, 10, 21
- version, 1
- W3C, 117
- while, 10
- XML, 1
- xsltproc, 1
- zone de saisie, 15