Chapitre 5

mySQL et Internet

5.1 Présentation de mySQL

5.1.1 Spécificité de mySQL

mySQL est un SGBDR complet qui offre pas mal de fonctions intéressantes. Il est destiné à la gestion de bases de données acceptant un nombre limité de clients simultanés pour lesquels on ne fait pas trop de mises à jour. La version 5 est conçue pour un usage plus professionnel et elle fait une terrible concurrence à des produits beaucoup plus chers. Les rachats successifs de la société qui commercialise mySQL par Sun puis par Oracle montre à quel point ce produit devient incontournable. Notons quelques-unes des caractéristiques de la version 4.x :

- implémentation complète des jointures de SQL2 ;
- les clés étrangères n'étant pas prises en compte dans le moteur par défaut (la commande `FOREIGN KEY` est admise, vérifiée syntaxiquement, mais n'a pas d'effet), il faut utiliser le moteur InnoDB [1] pour que cela fonctionne ;
- les champs automatiques sont gérés par une sympathique clause `AUTO_INCREMENT` placées dans la définition des clés primaires ;
- sous-requêtes (introduites dans les releases tardifs de la version 4).

Il faut la version 5.x pour les caractéristiques suivantes

- procédures stockées et triggers ;
- vues ;

Il faut noter que PHP peut également fonctionner avec un gestionnaire de bases de données plus léger (sans serveur) nommé SQLite qui présente quelques atouts (transactions, vues, triggers et requêtes imbriquées). Il gère des fichiers en local uniquement et ne peut donc pas jouer le rôle d'un serveur. Pour gérer quelques tables, il est parfaitement utile.

5.1.2 Installation, utilisateurs et sécurité

Les produits intégrés comme WAMP et XAMPP rendent l'installation de mySQL triviale. Il n'en sera donc pas question plus longtemps.

Par défaut le système s'installe avec un seul utilisateur, l'administrateur nommé *root* qui ne peut se connecter que sur le serveur. Il est fréquent de disposer des utilisateurs suivants :

- *l'internaute* (connexion locale uniquement) sera un utilisateur virtuel utilisé par les scripts PHP pour consulter les données, il n'aura aucun droit en écriture.
- les *gestionnaires* (connexion locale uniquement) seront des utilisateurs virtuels grâce auquel les scripts PHP opéreront des modifications de données. On peut évidemment

1. InnoDB a lui aussi été racheté par Oracle.

distinguer plusieurs types de gestionnaires aux droits plus ou moins étendus ou limités à certaines tables.
– l'administrateur se verra doté éventuellement d'une seconde ligne dans la table des utilisateurs pour autoriser des connexions distantes. On veillera à restreindre au maximum les machines d'où il pourra modifier le système, de manière à éviter les risques de piratage.

La gestion des utilisateurs de mySQL autorise curieusement des utilisateurs homonymes qu'on distingue par la famille de machines d'où ils peuvent se connecter.

Cette politique de sécurité est nécessaire au fonctionnement du serveur. On peut y superposer une politique d'authentification gérée par l'application. Il ne semble pas raisonnable d'utiliser le système des utilisateurs intégrés à mySQL pour gérer un grand nombre d'utilisateurs.

Comme beaucoup de SGBDR, mySQL se compose d'un logiciel serveur et d'un logiciel client. Le client fourni fonctionne en ligne de commandes et présente un aspect assez rugueux. La plupart des gestionnaires de site utilisent un client graphique écrit en PHP, nommé *PHP-MyAdmin*.

5.1.3 Sécurisation de l'administrateur

Il est indispensable que l'administrateur dispose d'un mot de passe sérieux. Par défaut, il n'y en a pas. Ce mot de passe doit être changé à deux niveaux : dans la base de données et dans les fichiers de paramétrages de l'interface phpMyAdmin (qui travaille avec des droits d'administrateurs). Rappelons que les services offerts en ligne ne proposent en général qu'une seule base et un seul utilisateur, que le client n'a donc pas le droit de modifier. La modification peut se faire à l'aide d'une programme en ligne de commande ou à l'aide de l'interface phpMyAdmin.

```
mysqladmin -u root password abcd
```

Si un mot de passe existe déjà, il faudra prévoir une option `-p` supplémentaire qui provoquera la demande du mot de passe actuel [2].

Pour l'interface Web, on localisera le fichier `config.inc.php` et on modifiera les lignes suivantes. Notons que le fichier contient le mot de passe en clair. Il convient donc de protéger l'accès à ce fichier.

```
// Authentication method (config, http or cookie based)?
$cfg['Servers'][$i]['auth_type']     = 'config';
// MySQL user
$cfg['Servers'][$i]['user']          = 'root';
// MySQL password (only needed with 'config' auth_type)
$cfg['Servers'][$i]['password']      = 'xxxxxxxx';
```

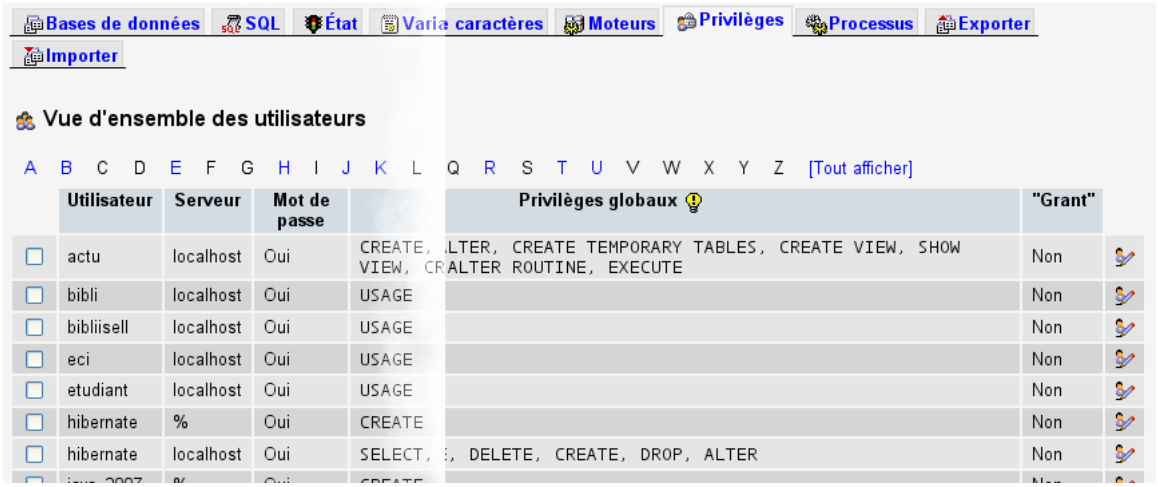
5.1.4 Créer utilisateur et base

Création des utilisateurs

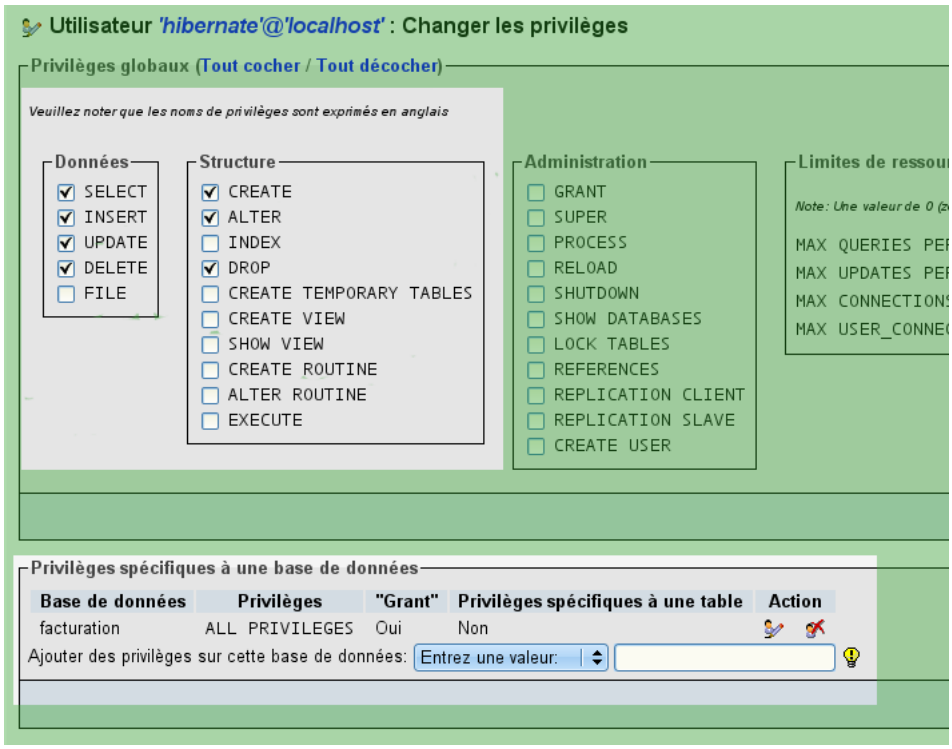
Les utilisateurs et leurs droits sont mémorisés dans la base *mysql* présente sur le serveur. Je déconseille de le faire directement dans les tables. La gestion des utilisateurs en ligne de

2. Si on a perdu son mot de passe administrateur (une mauvaise idée), l'administrateur de la machine peut suivre la procédure indiquée dans la section A.4.1 du manuel disponible sur Internet. Elle décrit la procédure à suivre sous Linux et sous Windows.

commandes ou par le biais de commandes SQL ne se fait pas sans mal. Il est vraiment conseillé d'utiliser l'interface PhpMyAdmin. On y parvient par la section privilège du menu principal. Rappelons que cette option n'est généralement pas disponible chez un hébergeur et que la gestion du mot de passe du seul utilisateur s'y fait généralement par des moyens propres au site.



Chaque utilisateur est défini par son nom et par le serveur d'où il peut appeler. L'illustration montre que l'utilisateur hibernate se différencie selon qu'il appelle depuis la machine qui héberge le serveur ou de n'importe où. Les droits ne sont pas les mêmes. En général, on donne à l'utilisateur des droits proches de ceux d'un administrateur, limité à une base. Par exemple, voici les droits principaux d'un utilisateur utilisé pour gérer un site Web :

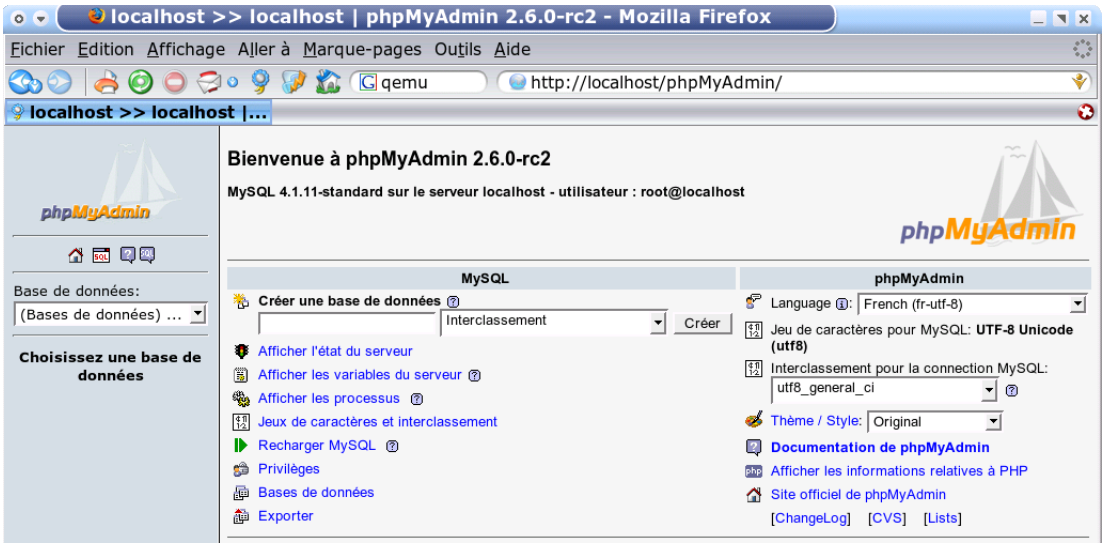


La zone réservée aux privilèges liés à la base de données *facturation* joue un rôle déterminant. On pourrait sans problème supprimer les autres droits.

Remarque importante :

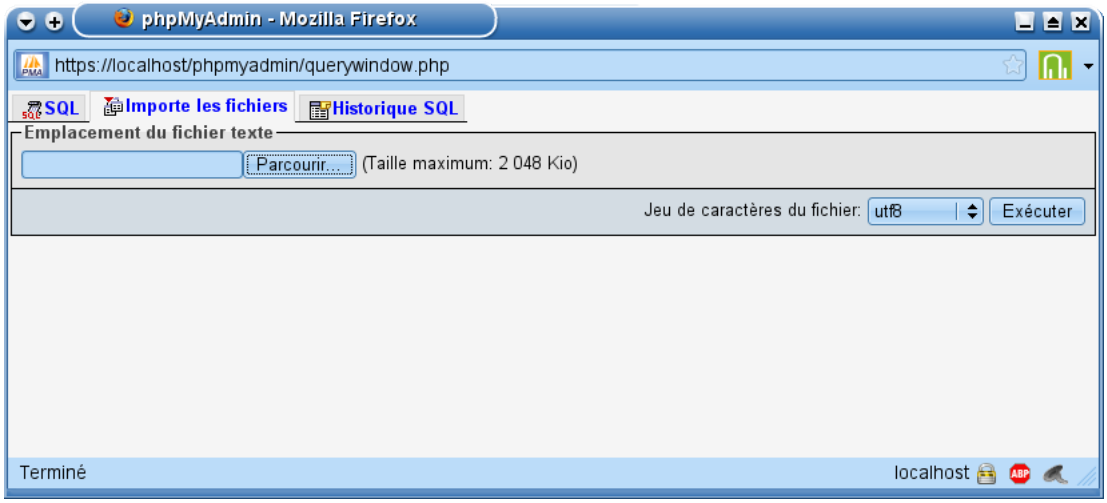
Pour la création d'un serveur Web, il n'est en général pas conseillé de créer des utilisateurs distants pour la base de données. En effet, l'utilisation par un serveur et PHP fonctionne au moyen d'un utilisateur sur localhost (à moins que le serveur Web et le serveur ne soient sur des machines distinctes). Dans le cas d'un hébergement, les droits ne sont généralement pas modifiables : on dispose d'un utilisateur-administrateur unique qui a tous les droits sur les tables, mais aucun autre droit (sauf parfois, celui de changer son mot de passe).

Création d'une base et des tables



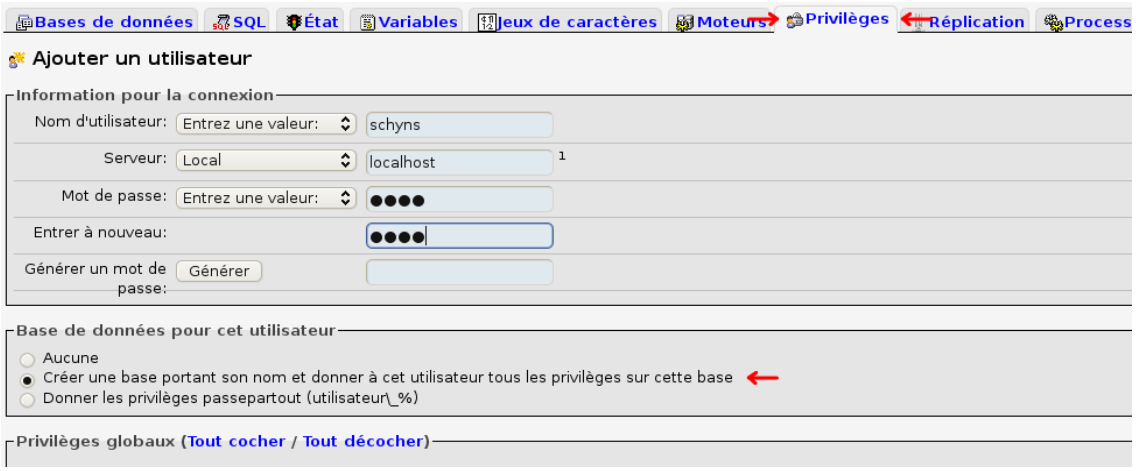
Si on accède à un serveur gratuit sur Internet, on ne dispose en général que d'une seule base de données, créée par le fournisseur. Autrement, l'écran d'accueil propose la création d'une base. La création d'une table se fait sans problème.

La technique la plus efficace pour installer les données sur le serveur consiste à passer par l'écriture d'un script reprenant toutes les commandes SQL. Ce script peut se trouver sur une machine cliente du serveur (on notera que sa taille est limitée à 2048 Ko par défaut, ce qui peut poser problème pour des tables très remplies).



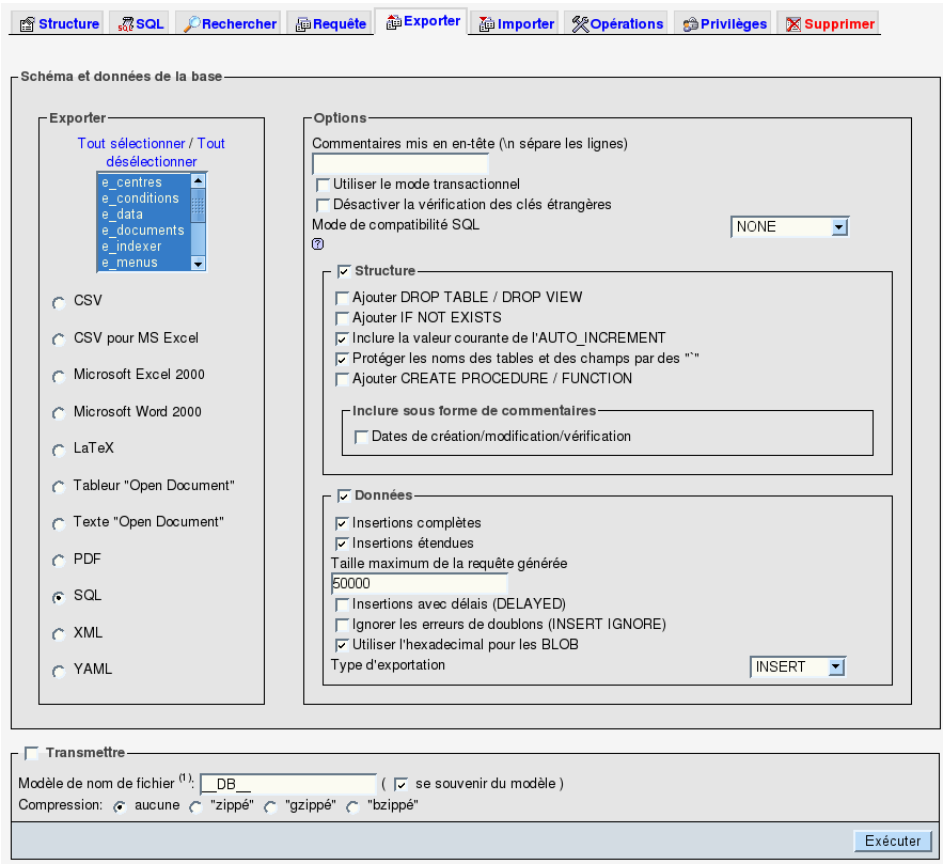
Méthode simplifiée

L'interface de PHPmyAdmin fournit un moyen simple de créer un utilisateur et une base qui lui appartient. Dans l'onglet, *Privilèges*, choisir la fonction *Créer un nouvel utilisateur*. On choisira, pour un site Web, un serveur local. Il suffit ensuite de cocher la case *Créer une base portant son nom...* pour obtenir un utilisateur et une base homonyme.



5.1.5 Déploiement d'une base de données sur un autre serveur

En général, on met au point une base de donnée sur une machine de développement avant d'installer la base sur une machine de production. L'exportation et la réimportation se font très facilement au moyen de PHPmyAdmin. Si l'exportation se fait au départ d'un serveur distant, il vaut la peine de demander des options de compressions.



5.2 Utiliser mySQL dans PHP avec l'extension PDO

La version 5 de PHP offre une nouvelle méthode de gestion des bases de données : l'extension PDO (pour *PHP Data Object*). L'avantage de cette nouvelle classe réside dans son indépendance par rapport à la base de données. On spécifie une unique chaîne de connexion, reprenant le pilote, la base, l'utilisateur et son mot de passe, ce qui permet de migrer les données d'un serveur vers un autre sans devoir modifier autre chose que cette chaîne. Il faut savoir que cette liberté impose la présence d'une couche supplémentaire dans le mécanisme et, par conséquent, une légère dégradation des performances. C'est largement compensé par la possibilité d'utilisation de requêtes précompilées.

5.2.1 Établissement de la connexion

```
try{
$connexion=
new PDO('mysql:host=cesar;port=3306;dbname=facturation;',
   'hibernate','codd');
}
// sur un site ne pas donner autant de détails au pirate potentiel
catch(Exception $e){
        die( 'Erreur : '.$e->getMessage().'<br />'.
                        'Numérp : '.$e->getCode());
}
```

À partir de ce moment, on peut utiliser l'objet `$connexion` pour la suite du programme. On peut remarquer immédiatement que si on remplace *mysql* par un autre type de serveur, la suite ne changera pas.

Remarques importantes :

1. afin de faciliter le portage de l'application (notamment entre le site de développement et le site de production), on conseille de concentrer un seul point la création de la connexion. La technique la plus simple consiste à créer un fichier contenant une fonction réalisation cette tâche. On peut alors facilement créer une connexion dans n'importe quel fichier.

   Contenu du fichier :

```
<?php // fichier include/fonctions.php
function getConnexion(){
        try{
        $connexion=
        new PDO('mysql:host=cesar;port=3306;dbname=facturation;',
           'hibernate','codd');
        }
        catch(Exception $e){
                die( 'Erreur de connexion';
        }
        return $connexion;
}
... autres fonctions si nécessaire
```

   Exemple d'utilisation :

```
require_once "include/fonctions.php";
$connexion = getConnexion();
```

2. l'instruction de création de l'objet `$connexion` comporte en clair tous les paramètres de connexion (y compris le mot de passe). Pour éviter que ces paramètres ne soient lus par un indésirable, il est conseillé de placer le fichier `fonctions.php` dans un sous-répertoire protégé. On peut placer un fichier `index.php` avec une redirection vers la racine du site ou encore mieux, un fichier `.htaccess` avec le contenu suivant :

```
<limit GET>
   order deny,allow
   deny from all
</Limit>
```

5.2.2 Requête SELECT

Il est extrêmement rare de récupérer une donnée isolée depuis une base de données. On travaille généralement avec une ligne complète et même un ensemble de lignes. On parle de *curseur*. Un curseur est une structure complexe (ou, dans le cas de PDO, un objet) qui contient une petite relation, susceptible d'être examinée. Généralement, la relation est en lecture seule et peut être parcourue une seule fois.

Pour récupérer un curseur sur des données, on peut utiliser deux méthodes. La première travaille avec une boucle `while` et la méthode `fetch()`.

```
$resultat = $connexion->query('SELECT * FROM Client ORDER BY 1');
while($ligne=$resultat->fetch(PDO::FETCH_OBJ)){
      echo 'Client'.$ligne->id.' - Nom: '.$ligne->nom.'<br/>';
}
```

La seconde, plus synthétique, utilise `foreach()`. Si on veut préciser le type de données récupérées, on utilise la méthode `setFetchMode()`. Dans l'exemple qui suit, chaque ligne est récupérée sous la forme d'un objet (`PDO::FETCH_OBJECT`).

```
$resultat = $connexion->query('SELECT * FROM Facture ORDER BY 1');
$resultat->setFetchMode(PDO::FETCH_OBJ);
foreach($resultat as $ligne){
      echo 'facture numéro '.$ligne->id.' - Montant: '.
          $ligne->montant.'<br/>';
}
```

L'utilisation d'un objet impose de connaître le nom des champs. Si on veut, on peut travailler avec un tableau associatif (`PDO::FETCH_ASSOC`), qui nous dispense de connaître le nom des champs et leur nombre.

```
$resultat = $connexion->query('SELECT * FROM Facture ORDER BY 1');
$resultat->setFetchMode(PDO::FETCH_ASSOC);
foreach($resultat as $ligne){
      foreach($ligne as $champ=>$valeur)
        echo $champ.':'.$valeur.'<br/>';
      echo '<hr/>';
}
```

Pour connaître le nombre de lignes récupérées, on peut facilement examiner la variable qui reçoit le curseur.

```
$resultat = $connexion->query('SELECT * FROM Facture ORDER BY 1');
$lignes=$resultat->fetchAll();
echo "Il y a ".count($lignes).' lignes dans la table <br/>';
```

5.2.3 Requêtes de modifications

Pour modifiers les données (`INSERT`, `UPDATE` ou `DELETE`), on utilise la fonction `exec()`. La valeur renvoyée représente ici le nombre de lignes affectées.

```
echo "nombre de lignes affectées";
echo $connexion->exec('UPDATE Facture SET Montant=Montant*2');
```

5.2.4 Requêtes préparées

Comme beaucoup d'interfaces modernes, PDO nous offre la possibilité de travailler avec des *statements*. On « prépare » la requête en y laissant des parties variables, qu'il suffit de remplacer au moment d'obtenir les données. La chaîne SQL contient des marqueurs pour indiquer les parties variables. Il existe des marqueurs anonymes et des marqueurs nommés. Ils ne peuvent être mélangés dans une même requête.

Marqueurs anonymes

Dans l'exemple suivant, on exécute deux fois la requête avec des valeurs différentes pour l'identifiant. S'il y a plusieurs paramètres, il suffit de les placer dans l'ordre dans le tableau fourni à la fonction `EXECUTE`. Dans l'exemple, le tableau n'a qu'un seul élément, puisqu'il n'y a qu'un paramètre.

```
$statement = $connexion->prepare('SELECT * FROM Client WHERE id = ?');

$statement->execute(array(1));
$ligne=$statement->fetch(PDO::FETCH_OBJ);
echo $ligne->nom.'<br/>';

$statement->execute(array(2));
$ligne=$statement->fetch(PDO::FETCH_OBJ);
echo $ligne->nom.'<br/>';
```

Marqueurs nommés et tableaux associatifs

On peut utiliser des marqueurs nommés qui peuvent être associés à une valeur par le biais d'un tableau associatif.

```
$statement = $connexion->prepare('SELECT * FROM Client WHERE id = :id');
$statement->execute(array(":id"=>1));
$ligne=$statement->fetch(PDOFETCH_OBJ);
echo $ligne->nom.'<br/>';
```

```
$statement->execute(array(":id"=>2));
$ligne=$statement->fetch(PDO::FETCH_OBJ);
echo $ligne->nom.'<br/>';
```

Une autre technique utilise les méthodes `bindValue()` et `bindParam()` pour spécifier la valeur des marqueurs. La seconde est extrêmement puissante. Elle permet de lier le marqueur à une variable PHP de façon à ce que toute modification de cette dernière soit répercutée sur la requête. Dans l'exemple suivant, on insère deux valeurs en modifiant simplement les variables liées à la requête préparée. On notera que la liaison des variables demande l'utilisation de l'opérateur de référence (`&`).

```
$SQL="INSERT␣INTO␣Client(nom,localite)␣VALUES(:nom,:localite)";
$statement = $connexion->prepare($SQL)  ;
$statement->bindParam(':nom',&$Nom);
$statement->bindParam(':localite',&$Localite);

$Nom="Dubois";
$Localite="Namur";
$statement->execute();
$Nom="Durant";
$Localite="Jambe";
$statement->execute();
```

L'objet PDO possède encore d'autres méthodes, mais il n'est pas possible de les couvrir dans cette petite introduction. On pourra se référer à la documentation de PHP disponible sur la toile (*http ://www.manuelphp.com/*).

5.2.5 Informations diverses concernant la base de données

L'extension PDO n'est pas conçue pour donner des renseignements sur la base de données et le serveur. Il reste néanmoins possible, au prix de la perte de la portabilité, d'interroger la base au moyen de commandes spécifiques passées sous forme de chaînes.

Obtenir la liste des bases d'un serveur

Il suffit de récupérer le résultat de la commande `SHOW DATABASES`.

```
/*
 * fonction renvoyant un tableau avec le nom
 * des bases de données du serveur
 */
function getBaseNames ($pdo) {
        $recordset = $pdo->query("SHOW␣DATABASES");
        $bases = $recordset->fetchAll(PDO::FETCH_ASSOC);
        foreach ($bases as $base) {
                $baseNames[] = $base['Database'];
        }
        return $baseNames;
}
/*
 * Utilisation de la fonction
 */
```

```
$pdo = new PDO('mysql:host=localhost;port=3306;dbname=facturation;',
'hibernate', 'codd');
$baseNames = getBaseNames($pdo);
foreach ($baseNames as $name) {
     echo "$name<br/>\n";
}
```

Obtenir la liste des tables d'une base de données

La liste des tables s'obtient par la requête `SHOW TABLES FROM` *Database*.

```
/*
 * Fonction renvoyant le nom des tables d'une
 * base de données
 */
function getTableNames ($pdo, $baseName) {
        $recordset = $pdo->query("SHOW␣TABLES␣FROM␣$baseName");
        $tables = $recordset->fetchAll(PDO::FETCH_ASSOC);
        foreach ($tables as $table) {
                $tableNames[] = $table['Tables_in_'.$baseName] ;
        }
        return $tableNames;
}
/*
 * Utilisation de la fonction
 */
$pdo=new PDO('mysql:host=localhost;port=3306;dbname=facturation;',
'hibernate', 'codd');
$tableNames = getTableNames($pdo, 'facturation');
foreach ($tableNames as $name) {
        echo "$name<br/>\n";
```

Obtenir la liste des champs d'une table

Pour mySQL, on emploiera la requête `SHOW COLUMNS FROM` *Table*.

```
/*
 * Fonction renvoyant le nom des champs d'une
 * table données
 */
function getFieldNames ($pdo, $tableName) {
        $recordset = $pdo->query("SHOW␣COLUMNS␣FROM␣$tableName");
        $fields = $recordset->fetchAll(PDO::FETCH_ASSOC);
        foreach ($fields as $field) {
                $fieldNames[] = $field['Field'];
        }
        return $fieldNames;
}
/*
 * Utilisation de la fonction
 * /
$pdo=new PDO('mysql:host=localhost;port=3306;dbname=facturation;',
```

```
'hibernate', 'codd');
$fieldNames = getFieldNames($pdo, 'factures');
foreach ($fieldNames as $name) {
        echo "$name<br/>\n";
}
```

La requête permet en fait d'obtenir six champs reprenant les caractéristiques de la table demandée.

Champ	Contenu
`Field`	Le nom du champ
`Type`	Le type
`Null`	NO ou YES selon que les nuls sont admis
`Key`	P indique la clé primaire
`Default`	Donne la valeur par défaut
`Extra`	Ici peut figurer Autoincrement pour les clés automatiques

Si on veut des renseignements plus complet sur la structure d'une table, se référer aux fonctions spécifiques à mySQL (sous-section 5.3.3 page suivante).

5.2.6 Problèmes d'encodage des caractères

J'ai récemment été particulièrement agacé par des problèmes d'encodage de caractères. Ma base de données, mes tables, mon éditeur, mes sources PHP, tout était en UTF-8, et malgré cela, j'affichais des caractères étranges à l'écran. Le problème a été résolu après une recherche sur Internet. Il suffit d'envoyer une requête spécifique qui choisit l'encodage voulu.

```
$dbo->query("SET NAMES 'UTF8'");
```

Ce n'est qu'une toute petite ligne, mais elle soulage bien des maux de têtes.

5.3 Bibliothèque mySQL

PHP est connu pour ses nombreuses bibliothèques, notamment concernant les bases de données. Parmi celles-ci, figure un ensemble de fonctions dédiées à mySQL, dont les noms commencent par `mysql_`. La plupart d'entre elles fournissent des fonctionnalités équivalentes à ce que propose PDO. Je me contenterai de mentionner quelques fonctions permettant d'obtenir des informations sur la base de données. Elles présupposent toutes l'établissement d'une connexion.

5.3.1 Établir une connexion et la terminer (`mysql_connect`)

mySQL demandait trois paramètres pour établir une connexion. Il n'est pas étonnant de les retrouver dans les paramètres de la fonction PHP.

```
$Serveur = mysql_connect($NomServeur,$Utilisateur,$MDP)
  or die("Serveur inconnu");
```

Il n'est pas obligatoire d'utiliser `die`. On peut également tester la valeur de retour (placée ici dans `$Serveur`) et vérifier qu'elle n'est pas nulle. Notons que tout ce qui suit `die` sera ignoré.

Pour terminer une connexion, on réutilise le lien obtenu lors de la création. Il faut noter que la connexion se ferme automatiquement à la fin du script. Il existe une fermeture explicite.

```
mysql_close($Serveur);
```

5.3.2 Choisir une base de données (`mysql_select_db`)

Une fois initialisée la connexion, il faut choisir sa base de données.

```
mysql_select_db($Base) or die("Base␣inconnue");
```

La base ainsi choisie devient celle qu'utilisent toutes les requêtes ultérieures. Une valeur fausse est renvoyée en cas d'échec. On peut également utiliser `die`. Ici encore, on peut utiliser une constante (voir plus haut).

5.3.3 Informations diverses concernant la base de données

Obtenir la liste des bases d'un serveur (`mysql_list_dbs`)

La fonction `mysql_list_dbs` renvoie une table que l'on peut analyser à l'aide de `mysql_fetch_row`. Il est également possible de compter le nombre de bases en utilisant `mysql_num_rows()` appliquée au même objet.

```
echo "Liste␣des␣bases␣de␣données␣disponibles␣:␣<br/>";
$Resultat = mysql_list_dbs($Serveur);
echo "<OL>";
while($base = mysql_fetch_row($Resultat))
   echo"<LI/>".$base[0];
echo "</OL>";
```

Obtenir la liste des tables d'une base de données (`mysql_list_tables`)

Le code suivant affiche la liste des tables de la base *$Base*. On notera l'emploi de `mysql_fetch_row` pour lire chaque ligne du résultat obtenu.

```
echo "Liste␣des␣tables␣dans␣$Base␣:␣<br/>␣<UL>";
$ListeTables = mysql_list_tables($Base,$Serveur);
while($TableSuivante = mysql_fetch_row($ListeTables))
   echo"<LI>".$TableSuivante[0];
echo "</UL>";
```

Obtenir la liste des champs d'une table (`mysql_list_fields`)

Le code suivant permet d'afficher à l'écran les noms, la taille et le type des champs de la table *$Table* d'une base *$Base*.

```
echo "Liste␣des␣champs␣dans␣Revues␣:␣<br>";
$Champs = mysql_list_fields($Base,$Table);
$NbreChamps = mysql_num_fields($Champs);
for($I=0;$I<$NbreChamps;$I++)
     {echo mysql_field_name($Champs,$I);
      echo "␣-␣";
      echo mysql_field_len($Champs,$I);
      echo "␣-␣";
      echo mysql_field_type($Champs,$I);
      echo "␣<br/>␣";
     }
```

Si on veut des renseignements plus complet sur la structure d'une table, on peut utiliser la fonction `mysql_fetch_field`, qui accepte les mêmes arguments que les trois fonctions précédentes. Douze méthodes permettent d'extraire les renseignements voulus : `name`, `table`, `max_length`, `not_null`, `primary_key`, `unique_key`, `multiple_key`, `numeric`, `blob`, `type`, `unsigned`, `zero_fill`.

```
$Objet=mysql_fetch_field($Champs,$I);
echo $Objet->name;
```

est équivalent à

```
echo mysql_field_name($Champs,$I);
```

Cette notation est plus pratique quand on veut manipuler plusieurs caractéristiques d'un champ.