

Introduction aux bases de données

Qu'est ce qu'une base de données ? Pour faire simple, une base de données permet de stocker et de récupérer facilement des informations. Ces informations sont structurées de manière à refléter la réalité tout en respectant quelques règles (unicité de l'information, chaque élément doit pouvoir être identifié de manière unique...).

Sur l'auteur

JULIEN HERBIN

Julien Herbin est ingénieur chez Pimentech, une Société de services en logiciels libres parisienne. Il travaille sur des projets s'appuyant de manière intensive sur des bases de données, principalement pour de grandes enseignes de l'immobilier.

SYLVESTRE LEDRU

Sylvestre Ledru est ingénieur à l'INRIA, responsable des versions GNU/Linux et Unix de logiciel de calcul opensource Scilab. Il a travaillé sur de nombreux sites internet mêlant de nombreuses technologies comme les sites sportsregions.fr et Kikooboo.com.



Niveau de difficulté

A l'heure actuelle, le modèle de base de données (BD) qui domine est le modèle relationnel. Les systèmes qui implémentent ce système s'appellent les systèmes de gestion de bases de données relationnelles (SGBDR, souvent appelés SGBD).

Malgré des avantages indéniables, le modèle concurrent, le modèle objet n'a toujours pas percé.

Les différents logiciels

Le marché du SGBDR est principalement concentré entre 4 logiciels :

- MySQL, logiciel libre proposé par la quasi totalité des hébergeurs grands publics. Rapide, facile d'accès et efficace, il rattrape son manque criant de fonctionnalités dans ses dernières versions. Il fonctionne efficacement sous GNU/Linux, quelques Unix et Windows
- PostgreSQL, logiciel libre malheureusement trop méconnu qui propose toutes les fonctionnalités (et même plus) que l'on attend d'un SGBDR moderne. Les développements actuels sur ce SGBD visent à optimiser les calculs et traitements
- Oracle, le leader en terme de base de données professionnelles
- Microsoft SQL Server, un des rares produits Microsoft de qualité même si ses failles de sécurité assez sensibles ont égratigné sa réputation

Outre ces quatre logiciels, il existe de nombreux autres SGBD comme Firebird, Microsoft Access, DB2 qui ont leurs avantages propres et désavantages (non, on vise à peine Access).

MySQL

MySQL possède de nombreuses API (interface de programmation) pour accéder à une base de données MySQL à partir de langages hétéroclites comme C, C++, PHP, Python, Ruby... Le formalisme des fonctions est autant que possible le même d'un langage à l'autre. Ces API permettent d'interagir avec la base de données et ainsi exécuter des requêtes sur la base de données, obtenir le résultat de l'opération (succès, erreurs, avertissements) ou récupérer un jeu de résultats dans les structures de données du langage en question.

Il ne faut pas perdre de vue qu'une base de données n'est qu'une manière (certes très efficace) de stocker des données et d'y accéder. Une application conçue au dessus d'une base de données pourrait avoir ses données

conservées dans une série de fichiers à plats. Cependant, la plupart des traitements deviendraient rapidement d'une complexité rare avec des performances fortement dégradées. Par exemple, essayez de pondre un algorithme permettant de récupérer dans un fichier plat client et un fichier de commandes tous les utilisateurs de plus de 25 ans ayant effectué plus de trois commandes.

La disparité de fonctionnalités et, dans un moindre mesure, de syntaxe d'un SGBD à l'autre pose de gros problèmes de portabilité d'une application. Le choix du système de gestion de base de données doit donc être un choix mûrement réfléchi de la part de l'équipe technique, une migration étant une opération très coûteuse, parfois longue et risquée.

Cependant, de nouvelles technologies (en particulier des API de programmation) permettent une abstraction du SGBD en déléguant la gestion des différences syntaxiques et fonctionnelles. Hibernate en Java est un bon exemple de technologie réussie.

Listing 1. Création d'une table

```
CREATE TABLE <nom de la table> (
  <nom du champs> <type> [default <value>] [extra options],
  [...]
  PRIMARY KEY <champs>,
  [ FOREIGN KEY <champs> REFERENCES TO <table> (<champs> ) ]
);
```

Tableau 1. Le tableau des clés primaires

id_message	date_ajout	date_modif	titre	texte	en_ligne
2	2006-08-10 23:26:34	2006-08-10 23:26:34	titre du message	mon premier test	1
3	2006-08-10 23:28:44	2006-08-10 23:42:42	second message	j'adore vraiment les sgbd	0

On peut accéder à la base de données via une interface en ligne de commande simple (ou via une application plus évoluée comme le fameux *PHPMyAdmin* ou *MysqlQuickAdmin*). Cette interface permet d'interagir totalement et directement avec les informations conservées par la base de données. De cette manière, il est possible de tester ses requêtes avant de les intégrer dans l'application.

Table

A la manière d'un fichier CSV, une table contient une catégorie d'information définie (une table qui sert à contenir la liste des messages d'un blog ne pourra et ne devra pas être utilisée pour contenir la liste des utilisateurs par exemple). Une table contient un nombre indéfini d'éléments (de 0 jusqu'à X) appelé Enregistrements. Un enregistrement est constitué de champs. Un champ est d'un type d'information défini (entier, caractère, chaîne, date...).

Idéalement, chaque enregistrement de la table doit pouvoir être identifié de manière unique à l'aide d'un ou de plusieurs champs.

Dans l'exemple concret présenté en deuxième partie de cet article, les tables sont *message* et *commentaire*. Un enregistrement sera un des messages du blog et un champ sera par exemple le titre du message.

Clés primaires

Une clé primaire est un ensemble de champ (même si, généralement, un seul champ est utilisé) qui permet d'identifier de manière unique les enregistrements de la table. Il existe deux manières de gérer les clés primaires :

- on délègue la génération d'un numéro unique au SGBD via une auto-incrémentation (généralement un bête `++`) ou via les séquences (numéro de facture mélangeant lettre et chiffre par exemple)..
- on utilise un champ ou plusieurs que l'on sait unique (numéro de sécurité sociale dans une table utilisateur par exemple).

La première méthode est préférée car les champs réellement uniques sont relativement rares (un numéro de sécurité sociale n'est fourni que rarement par un utilisateur et est limité à la France, un utilisateur peut ne pas avoir d'adresse email ou une adresse email peut être partagée entre plusieurs personnes, etc). A l'inverse, il arrive souvent que le numéro unique généré par le SGBD devienne

un numéro (ou une partie d'un numéro) de référence comme un numéro de facture...

Clés étrangères

Le principe des clés étrangères est le fondement de l'aspect relationnel dans les bases de données. Il permet de lier deux tables entre elles et ainsi de souligner un rapport sémantique entre ces deux entités. La clé étrangère dans une table doit être la clé primaire de l'autre table liée. Grâce à cette clé, une relation va être créée entre les deux tables qui permettra de récupérer toutes ces données lors de l'interrogation de la base.

Comme exemple, l'application blog développée dans la seconde partie utilise deux tables. La table message qui contient les messages des

blogs et la table commentaire qui, elle, stocke les commentaires envoyés sur un message. La nécessité d'une liaison entre un message et tous ses commentaires coule donc de source. Concrètement, la table commentaire va avoir un champ lui indiquant à quel message il se réfère (en l'occurrence *id_message*).

Création d'une table

Les types de données utilisables lors de la création varient dans leur nature d'un SGBD à l'autre. Une chaîne de caractères sous Oracle s'appellera *VARCHAR2* et *VARCHAR* sous MySQL.

Sous MySQL, les types classiques sont :

- *int*, entier
- *float*, nombre décimal (avec virgule)
- *char*, nombre de caractère fixé
- *varchar*, chaîne de caractère
- *text*, chaîne de caractère non limité
- *date*, comme son nom l'indique : date
- *timestamp*, permet de représenter un jour et une heure (*AAAA-MM-DD HH:MM:SS*)

Des variantes d'un SGBD telles que *TINYINT* existent.

Listing 2. La syntaxe pour la création de la table message

```
CREATE TABLE message (
  id_message INT AUTO_INCREMENT,
  date_ajout TIMESTAMP,
  date_modif TIMESTAMP NULL,
  titre VARCHAR(200),
  texte TEXT,
  en_ligne TINYINT(1) UNSIGNED,
  PRIMARY KEY (id_message)
```

Listing 3. Listing du fichier de création de la base de données structure_bd.sql

```
DROP TABLE IF EXISTS message;
CREATE TABLE message (
  id_message INT AUTO_INCREMENT,
  date_ajout TIMESTAMP,
  date_modif TIMESTAMP NULL,
  titre VARCHAR(200),
  texte TEXT,
  en_ligne TINYINT(1) UNSIGNED,
  PRIMARY KEY (id_message)
);
DROP TABLE IF EXISTS commentaire;
CREATE TABLE commentaire(
  id_commentaire INT AUTO_INCREMENT,
  date_ajout TIMESTAMP,
  pseudo VARCHAR(20),
  email VARCHAR(100),
  url VARCHAR(100),
  commentaire TEXT,
  id_message INT,
  PRIMARY KEY (id_commentaire)
);
```

Listing 4. Listing du fichier de configuration config.inc.php

```
<?php
/* Personnalisation de l'application */
define('TITRE_SITE', 'Mon super Blog'); // Titre du Blog
define('AUTEUR_BLOG', 'Moi'); // Nom de l'auteur du Blog
define('LONGUEUR_RESUME', 400); // Longueur maxi d'un message sur la page d'accueil
/* Paramètres de connexion à la base de données */
define('BD_HOTE', 'localhost'); // Nom d'hôte ou adresse IP du serveur de BD
define('BD_PORT', '3306'); // Numéro de port sur lequel tourne le service MySQL sur le
    serveur de BD
define('BD_NOM', 'monblog'); // Nom de votre base de données
define('BD_UTILISATEUR', 'moi'); // Nom d'utilisateur de votre base de données
define('BD_PASS', 'mon_pass'); // Mot de passe associé à votre utilisateur
?>
```

Listing 5. Listing du fichier de connexion à la base de données connexion_bd.inc.php

```
<?php
// on s'assure que le fichier de configuration est toujours inclus
require_once('config.inc.php');
// si la connexion au serveur de BD n'aboutit pas, un message d'erreur est affiché et
    l'exécution du script se termine
$conn_bd_blog = mysql_connect(BD_HOTE . ':' . BD_PORT, BD_UTILISATEUR, BD_PASS) or
    die("Connexion à la base de données impossible");
// si la sélection de la BD n'aboutit pas, un message d'erreur est affiché et
    l'exécution du script se termine
mysql_select_db(DB_NOM, $conn_bd_blog) or die("Impossible de sélectionner la base de
    données");
/*
//Si on arrive à ce niveau, la connexion au serveur est effective et la base de
    données est sélectionnée on //peut commencer à travailler sur
    la base de données !
*/
?>
```

Lors de l'insertion ou de la modification d'enregistrement, sous MySQL avec le comportement par défaut, si le champ ne correspond pas au type, MySQL va essayer de le *caster* (c'est-à-dire le convertir vers un autre type de données) vers le format de destination. Par exemple, une chaîne de 15 caractères insérée dans un champ de type varchar de 10 sera tronquée à la taille maximale. Ce comportement est différent selon le SGBD (et ses options), ainsi, PostgreSQL va refuser l'insertion en générant une erreur si la taille dépasse la taille maximale (de même si le type fournit ne correspond pas à celui spécifié).

Comme pour la plupart des opérations SQL, la syntaxe est relativement triviale.

Le choix des noms des champs est libre. Cependant, il est conseillé d'adopter un formalisme pour les noms en vue de faciliter le développement et la maintenance.

Par exemple, la syntaxe pour créer la table *message* est présentée par le Listing 2.

Modification

Sous MySQL, il est possible de modifier la structure de la table (nom des champs, type

de données, valeur par défaut...) en ajoutant/modifiant/supprimant des champs.

Pour rajouter un champ, la syntaxe sera :

```
ALTER TABLE <nom de la table>
ADD
    <nom du champs>
    <type du champs>
```

Pour la modification d'un champ, on devra utiliser la syntaxe suivante :

```
ALTER TABLE <nom de la table>
CHANGE
    <nom du champs d'origine>
    <nom du champs de
        destination>
    <type du champs>;
```

Suppression

La syntaxe de suppression d'une table est triviale. Aucune confirmation n'étant demandé, il est évident qu'il faut utiliser cette commande avec parcimonie et avec du doigté.

```
DROP TABLE <nom de la table>
```

Enregistrements

- Commun

A savoir : tous les champs impliquant une string/chaîne de caractères doivent être entourés par des simples quotes '' ou des doubles quotes

- Insertion

L'ajout d'un élément dans une table a plusieurs syntaxes.

- La syntaxe "décrite" :

```
INSERT INTO <nom de la table>
    (<champs 1>, <champs 2>, ...)
VALUES (<valeur1>, <valeur2>, ...)
```

On utilise cette syntaxe quand on ne veut pas spécifier tous les champs (quand ils sont inconnus ou lorsque l'on veut laisser les valeurs par défaut). L'ordre des champs n'a pas d'importance.

- La syntaxe *courte* :

```
INSERT INTO <nom de la table>
VALUES (<valeur 1>, <valeur 2>, ...)
```

Cette syntaxe nécessite que tous les champs de l'enregistrement aient une valeur et qu'ils respectent l'ordre de déclaration de la table.

- La syntaxe à *la UPDATE* :

```
INSERT INTO <nom de la table>
SET <champs 1>=<valeur 1>, <champs
    2>=<valeur 2>
```

Cette syntaxe se rapproche fortement de celle employée dans les requêtes de mise à jour (UPDATE). Par conséquent, elle est appréciée des développeurs fainéants (donc par la grande majorité) car elle est facile à dupliquer ou à factoriser.

Cette syntaxe est un exemple des différences de syntaxe d'un SGBD à l'autre. Elle est disponible sous MySQL à la différence de PostgreSQL qui ne la fournit pas.

Il est convenable donc de limiter son utilisation en particulier si le SGBD risque de changer dans l'avenir ou si le logiciel est prévu pour tourner sur différentes architectures.

Toujours dans le cadre de notre création d'un blog... La syntaxe SQL pour rajouter un message dans la table est :

```
INSERT INTO message
    (date_ajout, titre, texte, en_ligne)
```

```
VALUES
    (now(), 'Mon premier message',
    "Alors, facile le SQL ? C'est le premier
    message sur mon blog !", 1);
```

Cet exemple nous permet de souligner qu'il est possible de faire appel à des fonctions internes à MySQL (traitement sur les chaînes de caractères, dates...).

Mise à jour

La syntaxe pour mettre à jour un enregistrement n'est pas complexe. Elle s'écrit de la manière suivante :

```
UPDATE <nom de la table>
SET
    <nom du champs 1>=<nouvelle valeur>,
    <nom du champs 2>=<nouvelle valeur>
WHERE <condition>
```

La condition d'exécution de la requête est généralement faite sur la clé primaire mais peut être plus générale. Par exemple, on peut imaginer une requête de mise à jour qui ne changerait que les messages publiés du blog. Exemple :

```
UPDATE message SET
    <changement voulu>
WHERE en_ligne=1
```

Une fois exécutée, cette requête retourne le nombre d'éléments impactés par cette modification pour vérification ou réutilisation.

Suppression

De la même manière que pour la mise à jour, une requête de suppression se fait sur une condition. Condition qui n'est pas imposé au développeur. Généralement, étant donné le risque intrinsèque de cette requête, elle est effectuée sur la clé primaire mais rien n'oblige l'utilisateur à le faire.

```
DELETE FROM
    <nom de la table>
WHERE <condition>
```

Comme pour la mise à jour, la suppression retourne le nombre d'éléments impactés.

Requêtes

Toutes les possibilités offertes pour la récupération des données pourraient faire l'objet d'un article dédié. Nous allons aborder les plus utilisées et accessibles.

Requête de base

Les deux requêtes les plus courantes lorsque l'on débute dans les SGBD sont :

- récupérer tous les enregistrements d'une table
- récupérer un enregistrement spécifique

Récupérer tous les enregistrements est une procédure basique. La syntaxe de base est :

```
SELECT * FROM
    <nom de la table>
```

Listing 6. Listing du fichier de modèle HTML base_page.inc.php

```
<?php
require_once("lib_blog.inc.php");
?>
<html>
<head>
<title><?php echo const_ou_defaut('TITRE_PAGE', TITRE_SITE);?></title>
</head>
<body>
<h1><?php echo TITRE_SITE;?></h1>
<?php echo const_ou_defaut('CORPS_HTML', "<b>Attention :</b> la constante CORPS_HTML
    n'est pas définie");?>
</body>
</html>
```

Listing 7. Listing du fichier de librairie PHP lib_blog.inc.php

```
<?php
function const_ou_defaut($nom_const, $val_defaut) {
    if (defined($nom_const)) {
        eval("\$val = \$nom_const;");
        return $val;
    } else
        return $val_defaut;
}
?>
```

Contrairement à ce que l'on pourrait imaginer, l'étoile (*) ne sert pas à récupérer tous les enregistrements mais elle permet d'obtenir tous les champs des enregistrements.

Généralement, la recherche est faite sur la clé primaire.

```
SELECT * FROM message
WHERE id_message=42
```

Pour récupérer un enregistrement spécifique, il faut rajouter une condition à la requête. Pour cela, il faut rajouter une (ou plusieurs) clause(s) :

```
SELECT * FROM message
WHERE titre="Intro aux BDS";
```

De cette manière, on va récupérer tous les messages dont le titre est "Intro aux BDS".

Pour des questions de performance et de clarté dans le code, il est conseillé de spécifier tous les champs que l'on veut récupérer. Par exemple :

```
SELECT id_message, titre,
texte FROM message;
```

Astuce : Il est possible de trier les résultats. L'utilisation du mot clé ORDER BY sur un champs donné. Par exemple, SELECT id_message, titre, texte FROM message ORDER BY titre; triera par ordre alphabétique (a->z) tous les messages en fonction de leur titre.

Requêtes avancées

Il est intéressant de pouvoir récupérer un élément spécifique mais il serait possible de réaliser cela d'une manière relativement simple en utilisant un autre moyen de stockage. Un des atouts des SGBDs vient justement du principe de relations détaillé au - dessus.

Ainsi, une jointure permet de *relier* deux tables entre elles lors d'une interrogation de la base de données. Ceci permet de grouper et filtrer des informations selon des critères multiples et hétérogènes.

La jointure

La jointure s'effectue sur la clé primaire d'une table et sur la clé secondaire de l'autre.

Il existe plusieurs manières de réaliser des jointures. Celle présentée ici retourne les enregistrements qui existent dans les deux tables et dont les clés correspondent.

La syntaxe de jointure de deux tables est :

```
SELECT * FROM
    <nom de la table 1>,
    <nom de la table 2>
WHERE <nom de la table 1>.
    <clé primaire> =
        <nom de la table 2>
        .<clé secondaire>
```

Par exemple, cas pratique pas si rare dans le monde réel, imaginons que l'on veut récupérer tous les commentaires des messages en ligne. L'information qui va discriminer les

Listing 8. Listing de la page d'accueil du blog index.php

```

<?php
require_once("connexion_bd.inc.php");
$sout_html = "";
$resultat = mysql_query("SELECT UNIX_TIMESTAMP(date_ajout) AS date_ajout, UNIX_
                        TIMESTAMP(date_modif) AS date_modif, id_message, titre, texte
                        FROM message WHERE en_ligne = 1 ORDER BY date_ajout DESC");
while ($message = mysql_fetch_array($resultat)) {
    $sout_html .= '<h2>' . $message["titre"] . "</h2>\n";
    $sout_html .= '<p>' . substr($message["texte"], 0, LONGUEUR_RESUME) . ' <a
href="detail.php?id_message=' . $message["id_message"] . '">Lire
la suite</a></p>'. "\n";
    $sout_html .= '<p>Le ' . date("d/m/Y H:i", $message["date_ajout"]) . ' par ' .
AUTEUR_BLOG . ($message["date_modif"] ? ' (modifié le ' .
date("d/m/Y H:i", $message["date_modif"]) . ') : ' . ' . "</p>\n";
}
define('CORPS_HTML', $sout_html);
require_once("base_page.inc.php");
?>

```

commentaires sur le fait qu'un message soit en ligne ou pas (champs *tinyint en_ligne*).

La requête sera :

```

SELECT * FROM message,
commentaire WHERE message.
id_message=commentaire.id_message
AND message.en_ligne=1

```

Plutôt que d'exécuter deux requêtes et d'effectuer en plus une partie du traitement dans un langage au - dessus, cette requête permet de récupérer d'un coup tous les enregistrements qui correspondent à notre besoin.

Autres fonctionnalités

Nous n'avons fait qu'aborder les fonctionnalités de base d'un SGBD. De nombreuses possibilités sont offertes par les SGBDs. Si les bases de données vous passionnent (ce dont je ne doute pas), vous découvrirez l'intérêt des subselects, l'indexation des champs, les contraintes d'intégrité fonctionnelles, les autres types de jointures, les vues, les triggers, les procédures stockées, l'héritage de tables, le mapping objet/relationnel, la réplication...

Écriture d'un Blog en PHP

Maintenant que vous connaissez les concepts fondamentaux des bases de données, passons à la pratique en écrivant une application s'appuyant sur une base de données pour assurer le stockage des données persistantes. En écrivant une version basique d'un système de blog, nous allons apprendre à manipuler une base de données avec un langage de programmation.

La compréhension de cette petite application, écrite en PHP, ne nécessite pas la maîtrise de

ce langage, mais il est important d'avoir tout de même quelques notions de programmation (constantes, variables, fonctions). PHP est en langage de script objet open source dont la syntaxe est assez proche du C, très répandu et principalement employé pour développer des sites Internet. Il est doté de librairies très riches et d'une communauté d'utilisateurs très vaste et dynamique.

À ce niveau, nous supposons que vous disposez d'un environnement de développement PHP/MySQL, c'est - à - dire d'un serveur Web accompagné du module PHP (avec le support MySQL) et d'un serveur MySQL. Bien entendu, le serveur de base de données ne doit pas nécessairement tourner sur la même machine que le serveur Web. Vous aurez aussi besoin d'un client MySQL sur votre machine ou sur une machine sur laquelle vous disposez d'un compte pour effectuer quelques manipulations sur la base de données.

Création de la base de données

La première étape de notre travail consiste à créer la base de données. Connectez vous avec

le compte root au serveur MySQL et tapez la commande suivante :

```

mysql>
CREATE DATABASE monblog

```

Puis créez un utilisateur dédié pour l'application de Blog et attribuez lui les droits lui permettant d'effectuer un certain nombre d'opérations à l'aide de la commande GRANT de MySQL :

```

mysql> GRANT ALL PRIVILEGES
ON monblog.* TO 'moi'@'localhost'
IDENTIFIED BY 'mon_pass'
WITH GRANT OPTION;

```

Cette commande permet aussi de définir les machines depuis lesquelles notre utilisateur aura l'autorisation de se connecter, ainsi qu'un mot de passe lui permettant de s'authentifier.

À l'issue de l'exécution de ces deux instructions, nous disposons d'une base de données monblog vide et d'un utilisateur moi ayant tous les droits sur cette base. Il n'est plus nécessaire de travailler sous la session root, vous pouvez donc la quitter et tenter de vous connecter à la base monblog en utilisant le compte nouvellement créé :

```
$ mysql -u moi monblog -p
```

Après avoir saisi votre mot de passe, si l'authentification est réussie, vous obtiendrez un prompt MySQL (`mysql>`) vous permettant d'exécuter des instructions SQL sur la base monblog.

Modèle des données

L'application que nous allons écrire n'étant pas très complexe, le modèle des données associé est très simple : il comporte seulement deux tables. La première va nous servir à stocker les messages postés sur le blog, la seconde à enregistrer les commentaires des internautes.

Créons un fichier *structure_bd.sql* contenant les instructions SQL permettant de créer ces deux tables présentées par le Listing 3.



Figure 1. Capture d'écran de la page d'accueil

Listing 9. Listing de la page de détail d'un message detail.php

```

<?php
assert(isset($_GET["id_message"]) && is_numeric($_GET["id_message"]));
require_once("connexion_bd.inc.php");
Sout_html = '<< <a href="index.php">Accueil</a>' . "\n";
// si un nouveau commentaire a été posté, on l'ajoute à notre base de données
if (isset($_POST["action"]) && $_POST["action"] == "Envoyer") {
    if (mysql_query("INSERT INTO commentaire (date_ajout, pseudo, email, url,
        commentaire, id_message) VALUES (now(), '" . $_POST["pseudo"]
        . "', '" . $_POST["email"] . "', '" . $_POST["url"] . "', '" .
        $_POST["commentaire"] . "', '" . $_GET["id_message"] . "')")) {
        Sout_html .= '<h4>Votre commentaire a bien été ajouté !</h4>' . "\n";
    }
}
// affichage du message
$resultat = mysql_query("SELECT UNIX_TIMESTAMP(date_ajout) AS date_ajout, UNIX_
    TIMESTAMP(date_modif) AS date_modif, id_message, titre, texte
    FROM message WHERE id_message = " . $_GET["id_message"] . " AND
    en_ligne = 1");
if ($message = mysql_fetch_array($resultat)) {
    Sout_html .= '<h2>' . $message["titre"] . "</h2>\n";
    Sout_html .= '<p>' . $message["texte"] . '</p>.\n';
    Sout_html .= '<p><i>Le ' . date("d/m/Y H:i", $message["date_ajout"]) . ' par
        ' . AUTEUR_BLOG . ($message["date_modif"] ? ' (modifié le ' .
        date("d/m/Y H:i", $message["date_modif"]) . ') : ' . $message["pseudo"] .
        '</i></p>\n';
}
// affichage des commentaires
Sout_html .= '<h3>Commentaires</h3>' . "\n";
$resultat_commentaires = mysql_query("SELECT UNIX_TIMESTAMP(date_ajout) AS date_
    ajout, id_commentaire, commentaire, pseudo, email, url FROM
    commentaire WHERE id_message = " . $_GET["id_message"]);
if (mysql_numrows($resultat_commentaires) == 0) Sout_html .= '<p>Aucun commentaire</
    p>' . "\n";
while ($commentaire = mysql_fetch_array($resultat_commentaires)) {
    Sout_html .= '<p>' . $commentaire["commentaire"] . '</p>.\n';
    out_html .= '<p><i>Le ' . date("d/m/Y H:i", $message["date_ajout"]) . '
        par <a href="mailto:' . $commentaire["email"] . '>' .
        $commentaire["pseudo"] . "</a>". ($commentaire["url"] ? " (" .
        $commentaire["url"] . ")") . "</i></p>\n";
}
// affichage du formulaire pour poster un commentaire
Sout_html .= '<h3>Ajoutez votre commentaire</h3>.\n';
Sout_html .= '<form method="post">' . "\n";
Sout_html .= 'Pseudo : <input type="text" name="pseudo" /> <br />' . "\n";
Sout_html .= 'Email : <input type="text" name="email" /> <br />' . "\n";
Sout_html .= 'Url : <input type="text" name="url" /> <br />' . "\n";
Sout_html .= 'Commentaire : <br /><textarea name="commentaire"></textarea> <br />' .
    "\n";
Sout_html .= '<input type="submit" name="action" value="Envoyer" /> <br />' . "\n";
Sout_html .= '</form>';
define('CORPS_HTML', $out_html);
require_once("base_page.inc.php");
?>

```

Les lignes débutant par DROP TABLE IF EXISTS suivies du nom d'une table permettent de supprimer une table si elle existe déjà dans la base de données. En effet, la création d'une table portant le même nom qu'une ta-

ble déjà existante est impossible (et n'aurait pas de sens). Ce genre d'instruction permet de rejouer plusieurs fois le fichier SQL (dans le cas d'un changement de modèle) sans générer d'erreur. Attention tout de me mêm-

me, supprimer une table aura pour effet de supprimer l'intégralité des enregistrements qu'elle contient.

La table *message* contient six champs :

- *id_message* : un nombre entier, identifiant unique attribué automatiquement à chaque nouvel enregistrement inséré dans la table message. Ce champ est la clé primaire de notre table, il est fondamental, car il identifie chaque enregistrement.
- *date_ajout* : un champ de type TIMESTAMP correspondant à la date précise de l'ajout de notre message.
- *date_modif* : un second champ de type TIMESTAMP qui va nous servir à stocker la date de modification du message. Notons que le mot clé NULL autorise à ne pas renseigner ce champ, qui n'a pas de sens tant qu'un message n'a pas été modifié.
- *titre* : un champ de type VARCHAR contenant le titre de notre message, avec une limite à 200 caractères.
- *texte* : le champ qui va contenir le corps du message. Comme il est de type TEXT, il peut contenir un nombre illimité de caractères.
- *en_ligne* : dernier champ de notre table, de type TINYINT(1), il déterminera si le message doit être affiché sur le Blog ou non.

Notre seconde table, *commentaire* est aussi constituée de six champs :

- *id_commentaire* : à l'image du champ *id_message* de la table message, ce champ sera l'identifiant entier unique attribué automatiquement de la table commentaire. Il constitue la clé primaire de cette table.
- *date_ajout* : champ de type TIMESTAMP contenant la date d'ajout du commentaire
- *pseudo* : chaîne de caractère de longueur 20 au maximum contenant le pseudonyme de l'internaute qui a écrit le commentaire
- *email* : chaîne de caractère de longueur 100 au maximum contenant l'adresse e-mail de l'internaute
- *url* : autre chaîne de caractère de longueur 100 au maximum contenant cette fois l'adresse du site web de l'internaute
- *id_message* : nous retrouvons ici un champ portant le même nom que la clé primaire de la table message, et c'est justement parce qu'il y fait référence. En effet, ce champ contiendra l'identifiant du message auquel se réfère le commentaire. C'est une clé étrangère de la table commentaire.



Figure 2. Capture d'écran du détail avec commentaires

Afin d'alimenter notre Blog, nous allons ajouter un premier message à la fin du fichier, à l'aide d'une requête `INSERT` :

```
INSERT INTO message (date_ajout, titre,
texte, en_ligne) VALUES (now(),
'Mon premier message', "Alors, facile
le SQL ? C'est le premier message sur mon
blog !", 1)
```

Le fichier d'instructions SQL complet, vous disposez de deux solutions pour jouer son contenu sous MySQL :

- copier / coller le code SQL dans une console où vous êtes connecté à MySQL (ou dans un front-end Mysql, comme phpMyAdmin)
- jouer directement le fichier en ligne de commande :

```
mysql -u nom_utilisateur -p
nom_de_la_base < structure_bd.sql
```

La base de données est maintenant opérationnelle !

Fichier de configuration de l'application

Une bonne habitude à prendre en programmation est de créer un ou plusieurs fichiers de configuration pour donner un peu plus de souplesse à l'application. Ne dérogeons pas à cette règle et écrivons un fichier `config.inc.php` qui va nous servir à définir différentes constantes, comme les paramètres de connexion à la base de données, utilisées à différentes reprises par notre application. En PHP, ON définit une constante à l'aide de la fonction `define`, qui prend en paramètre le nom de la constante et sa valeur (voir le Listing 4).

Connexion à la base de données

Créons maintenant un fichier `connexion_bd.inc.php` qui va se charger d'initialiser une connexion à la base de données. La fonction

`require_once` nous permet d'inclure le code du fichier `config.inc.php` au début du fichier de connexion à la base. Une fois cette ligne exécutée, les constantes définies dans le fichier de configuration deviennent accessibles. Cette situation est présentée par le Listing 5.

La connexion à la base de données se fait en deux étapes : d'abord on s'authentifie auprès du serveur MySQL, puis on *sélectionne* la base de données sur laquelle on désire travailler. Si une de ces deux étapes se traduit par un échec, l'exécution du script est stoppée et un message d'erreur est affiché. Il est possible de configurer plusieurs connexion MySQL en parallèle ; dans ce cas, il faut travailler avec des références vers les connexions pour indiquer aux fonctions PHP sur quelle connexion jouer les requêtes.

Il n'est pas nécessaire de fermer une connexion MySQL en PHP car, dans le cas d'une connexion non persistante, celle-ci est automatiquement fermée à la fin de l'exécution du script. Mais il reste possible de fermer explicitement une connexion en invoquant de fonction `mysql_close`.

Squelette HTML et librairie blog

Une des difficultés avec PHP est que le code peut très rapidement se retrouver *noyé* dans du code HTML, ce qui nuit considérablement à la lisibilité et à la reprise du code. Généralement, afin de séparer le calcul (PHP) de la présentation (HTML + CSS), on s'oriente vers un moteur de templates (comme Smarty pour PHP). Nous nous contenterons ici d'écrire un petit système très simple permettant de factoriser le code HTML commun à toutes nos pages. Nous n'aurons ainsi qu'à générer le code HTML spécifique à une page donnée dans le fichier PHP qui effectue le calcul.

Le fichier `base_page.inc.php` met en place le squelette d'une page HTML et s'attend à ce que deux constantes soient définies avant d'être inclus à la fin d'un script PHP :

- `TITRE_PAGE` : le titre de la page, affiché dans la barre de titre de la fenêtre du navigateur et en haut de notre page HTML. Si cette constante n'est pas définie, le titre du site, défini comme constante (`TITRE_SITE`) dans le fichier de configuration s'affichera à la place.
- `CORPS_HTML` : la partie de code HTML spécifique à une page. Si la constante n'est pas définie, un message d'erreur s'affichera.

Vous aurez probablement remarqué l'inclusion du fichier `lib_blog.inc.php` au début du fichier, ainsi que l'utilisation de la fonction `const_ou_defaut` invoquée à trois reprises dans le fichier `base_page.inc.php`. Cette fonction n'est pas fournie par PHP, mais elle est définie par nos soins dans un fichier de *librairie* `lib_blog.inc.php` dans lequel nous placerons les fonctions spécifiques à notre application. La fonction `const_ou_defaut` prend en paramètre le nom d'une constante, dont le contenu sera affiché si cette

constante est définie, et une valeur par défaut qui sera affichée dans le cas contraire.

Page d'accueil du Blog

Après tout ce travail de préparation, nous allons enfin écrire la première page visible du site, et c'est logiquement à l'écriture de la page d'accueil que nous allons nous atteler. On trouvera sur cette page un résumé de chaque message publié sur notre Blog, accompagné d'un lien pour accéder au message dans son intégralité et aux commentaires des internautes. Il va donc falloir se connecter à la base de données, exécuter une requête de type SELECT nous renvoyant la liste des messages et traiter le résultat pour l'affichage.

Le simple fait d'inclure le code du fichier *connexion_bd.inc.php* nous assure que la connexion à la base de données est effective. Comme nous l'avons vu plus haut, dans le cas contraire, le script n'ira pas plus loin et affichera un message d'erreur.

Deux appels de fonctions sont particulièrement importants dans ce script. D'abord, l'appel à *mysql_query* qui prend en paramètre une requête SQL sous la forme d'une chaîne de caractères et renvoie une ressource contenant le jeu de résultat de la requête. La requête que nous jouons ici retourne tous les messages en ligne par ordre inverse de création. Notez que les champs de type *TIMESTAMP* sont transformés en timestamps au format Unix à l'aide de la fonction *MySQL_UNIX_TIMESTAMP* car la fonction de formatage de timestamp date de PHP fonctionne avec ce type de paramètre.

Le second appel de fonction qu'il faut bien comprendre est l'appel à *mysql_fetch_array* qui prend une ressource MySQL correspondant au résultat d'une requête en paramètre et renvoie le prochain enregistrement du résultat sous forme de tableau indexé par le nom des champs, ou le booléen false si le dernier enregistrement est atteint. Ainsi, il devient trivial, à l'aide de l'instruction *while* de PHP, d'itérer sur les enregistrements renvoyés par une requête. Les instructions situées dans la boucle *while* effectuent simplement une concaténation des données présentes dans l'enregistrement en cours à la chaîne de caractère *\$out_html*, précédemment initialisée.

À la fin du script, le contenu de la variable *\$out_html* est placé dans la constante *CORPS_HTML* et l'inclusion de notre squelette HTML provoque l'affichage de notre page d'accueil.

Remarquez l'identifiant *id_message* passé à la page *detail.php* au niveau du lien *Lire la suite*. Cet identifiant va nous permettre de connaître le message pour lequel on désire afficher le détail et les commentaires lié à ce message.

Page de détail d'un message et des commentaires

La dernière étape de notre travail sera d'écrire une page qui affiche le détail d'un message posté sur notre blog, suivi des commentaires envoyés par les internautes et d'un formulaire leur permettant de les saisir. Même si le code

de cette page est plus long, la technique reste identique : on se connecte à la base en premier lieu, on effectue une série de requêtes, puis on réalise un traitement sur le jeu de résultat.

La première instruction sert à vérifier que l'on a bien reçu l'identifiant en paramètre HTTP GET du message pour lequel on souhaite obtenir le détail. Cet identifiant étant la clé primaire de la table message, un seul message peut lui correspondre.

Suite à cette première vérification, la connexion à la base de données est initialisée, puis une première requête est éventuellement jouée, si un commentaire vient d'être posté. Cette requête insère un enregistrement dans la table commentaire à partir des informations saisies dans le formulaire, en prenant bien soin de positionner la clé étrangère *id_message* afin de connaître le message auquel correspond le commentaire. Notons que l'on emploie aussi la fonction PHP *mysql_query* pour jouer une requête de type *INSERT*.

La partie concernant l'affichage du message original sur cette page est très similaire à celle de la page d'accueil, à la différence que la requête de sélection applique un filtre sur la clé primaire pour ne recevoir que l'enregistrement correspondant au message désiré. D'autre part, le message n'est cette fois pas tronqué s'il est trop long, comme c'est le cas sur la page *index.php* (cf : appel à la fonction PHP *substr*).

Une dernière requête est finalement exécutée pour sélectionner l'ensemble des commentaires se rapportant au message, qui sont présentés d'une manière très similaire à celle du message. Au cas où la requête ne renvoie pas de résultat, ce que l'on vérifie en invoquant la fonction PHP *mysql_numrows* (qui donne le nombre d'enregistrements présents dans le jeu de résultat de la requête), le message *Aucun commentaire* est affiché à la place de la liste des commentaires.

Pour finir, un formulaire HTML permettant de saisir un commentaire est affiché à la fin de la page.

À vous de jouer...

Pour réaliser le gain de temps et le confort que nous apporte la base de données, même dans le cadre d'un projet très simple comme celui-ci, il suffit d'imaginer le temps et le code supplémentaire nécessaire pour arriver au même résultat en utilisant des fichiers plats. Même si cet exemple est très pratique, il reflète un schéma classique d'interaction entre un langage de programmation et une base de données *Connexion > Requête > Itération sur les résultats > Présentation du résultat*, valable quelque soit le serveur de base de données ou le langage retenu.

Vous devriez maintenant pouvoir laisser libre court à votre imagination et démarrer votre propre projet en PHP / MySQL ou pourquoi pas étoffer ce blog en y rajoutant des fonctionnalités (pages d'administration, navigation par pages, flux RSS...). 🐘



www.linuxdevjournal.com