MySQL

pour booster votre site web PHP

Hugo Etiévant

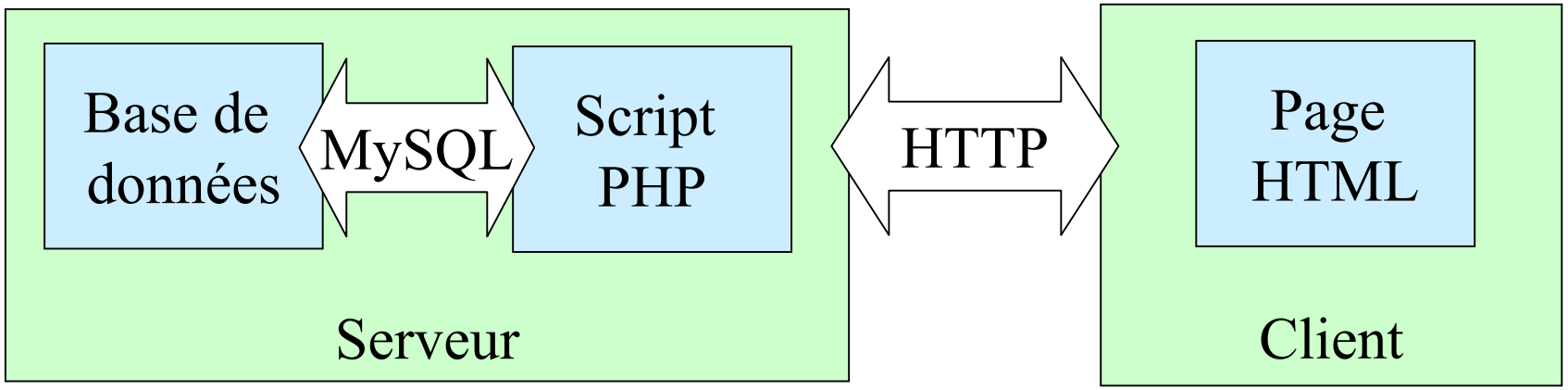
Dernière mise à jour : 20 juillet 2003

Introduction

MySQL dérive directement de SQL (Structured Query Language) qui est un langage de requête vers les bases de données exploitant le modèle relationnel.

Il en reprend la syntaxe mais n'en conserve pas toute la puissance puisque de nombreuses fonctionnalités de SQL n'apparaissent pas dans MySQL (sélections imbriquées, clés étrangères…)

Le serveur de base de données MySQL est très souvent utilisé avec le langage de création de pages web dynamiques : PHP. Il sera discuté ici des commandes MySQL utilisables via PHP dans les conditions typiques d'utilisation dans le cadre de la gestion d'un site personnel hébergé gratuitement (par exemple sur Free.fr).



Sommaire

- Théorie des bases de données relationnelles
- Syntaxe de MySQL
- Fonctions de MySQL
- Interface avec PHP
- Administration avec l'outil phpMyAdmin

Sélectionner des enregistrements (I)

Pour extraire de votre base de données des informations, comme la liste des personnes de votre carnet d'adresse qui vivent à Paris.

Syntaxe générale :

SELECT [DISTINCT] *attributs*
[INTO OUTFILE fichier]
[FROM *relation*]
[WHERE condition]
[GROUP BY *attributs* **[ASC | DESC]]**
[HAVING condition]
[ORDER BY *attributs* **]**
[LIMIT [a,] b]

Exemple :

SELECT *nom*,*prénom* **FROM *Personnes* WHERE** *adresse* **LIKE** '%paris%'

Sélectionner des enregistrements (II)

Nom	Description
SELECT	Spécifie les attributs dont on souhaite connaître les valeurs.
DISTINCT	Permet d'ignorer les doublons de ligne de résultat.
INTO OUTFILE	Spécifie le fichier sur lequel effectuer la sélection.
FROM	Spécifie le ou les relations sur lesquelles effectuer la sélection.
WHERE	Définie le ou les critères de sélection sur des attributs.
GROUP BY	Permet de grouper les lignes de résultats selon un ou des attributs.
HAVING	Définie un ou des critères de sélection sur des ensembles de valeurs d'attributs après groupement.
ORDER BY	Permet de définir l'ordre (**ASC**endant par défaut ou **DESC**endant) dans l'envoi des résultats.
LIMIT	Permet de limiter le nombre de lignes du résultats

Sélectionner des enregistrements (III)

Procédons par étapes :

Pour sélectionner tous les enregistrements d'une relation :

SELECT * FROM *relation*

Pour sélectionner toutes les valeurs d'un seul attribut :

SELECT *attribut* **FROM *relation***

Pour éliminer les doublons :

SELECT DISTINCT *attribut* **FROM *relation***

Pour trier les valeurs en ordre croissant :

SELECT DISTINCT *attribut* **FROM *relation* ORDER BY** *attribut* **ASC**

Pour se limiter aux **num** premiers résultats :

SELECT DISTINCT *attribut* **FROM *relation* ORDER BY** *attribut* **ASC LIMIT num**

Pour ne sélectionner que ceux qui satisfont à une condition :

SELECT DISTINCT *attribut* **FROM *relation* WHERE condition ORDER BY** *attribut* **ASC LIMIT num**

Jointure évoluée (I)

En début de ce document, on a vu la jointure suivante :

SELECT ***Personnes***.*nom*, *nblivres*
FROM ***Personnes, Bibliothèque***
WHERE ***Personnes***.*nom* **=** ***Bibliothèque***.*nom*

qui permet de concaténer deux relation en prenant un attribut comme pivot.

Il est possible de concaténer deux relation sur plusieurs attributs à la fois, ou même de concaténer X relation sur Y attributs.

Les requêtes utilisant très souvent les jointures, il a été créé une syntaxe spéciale plus rapide : JOIN que la méthode vue plus haut : avec la clause WHERE.

Ainsi la jointure précédente peut s'écrire aussi :

SELECT ***Personnes***.*nom*, *nblivres*
FROM ***Personnes*** **INNER JOIN** ***Bibliothèque***
USING (*nom*)

ce qui signifie que les deux relations ***Personnes*** et ***Bibliothèque*** sont concaténée (INNER JOIN) en utilisant (USING) l'attribut *nom*.

Jointure évoluée (II)

La syntaxe USING permet de lister les attributs servant de pivot. Ces attributs doivent porter le même nom dans chacune des tables devant être concaténées.

Si les attributs pivots ne portent pas le même nom, il faut utiliser la syntaxe ON.

Ainsi la jointure précédente peut s'écrire aussi :

```
SELECT Personnes.nom, nblivres
FROM Personnes INNER JOIN Bibliothèque
ON Personnes.nom = Bibliothèque.nom
```

La méthode **INNER JOIN** n'inclus les enregistrements de la première table que s'ils ont une correspondance dans la seconde table.

Personnes

Nom	Prénom
Martin	Jean
Tartan	Pion
Dupond	Jacques

Bibliothèque

Nom	Nblivres
Martine	5
Tartan	10
Dupond	3

Résultat de la jointure

Nom	Nblivres
Tartan	10
Dupond	3

Jointure évoluée (III)

Pour remédier aux limites de **INNER JOIN**, il existe la syntaxe **LEFT JOIN** qui inclus **tous** les enregistrements de la première table même s'ils n'ont pas de correspondance dans la seconde table. Dans ce cas précis, l'attribut non renseigné prendra la valeur NULL.

Là encore, le **ON** peut avantageusement être remplacé par le **USING**.

La jointure devient :

SELECT ***Personnes.****nom, nblivres*
FROM ***Personnes*** **LEFT JOIN** ***Bibliothèque***
ON ***Personnes.****nom* **=** ***Bibliothèque.****nom*

Personnes

Nom	Prénom
Martin	Jean
Tartan	Pion
Dupond	Jacques

Bibliothèque

Nom	Nblivres
Martine	5
Tartan	10
Dupond	3

Résultat de la jointure

Nom	Nblivres
Martin	NULL
Tartan	10
Dupond	3

Les fonctions

Bien que ces fonctions appartiennent typiquement à MySQL, la création d'un chapitre à part se justifie par le fait que je vais me contenter ici d'énumérer les fonctions les plus courantes.

Reportez-vous au manuel MySQL pour la liste détaillée de toutes les fonctions disponibles dans votre version du serveur MySQL.

Ces fonctions sont à ajouter à vos requêtes dans un SELECT, WHERE, GROUP BY ou encore HAVING.

D'abord sachez que vous avez à votre disposition :

- les parenthèses **( )**,
- les opérateurs arithmétiques (**+**, **-**, *****, **/**, **%**),
- les opérateurs binaires (**<**, **<<** , **>**, **>>**, **|**, **&**),
- les opérateurs logiques qui retournent **0** (faux) ou **1** (vrai) (**AND**, **OR**, **NOT**, **BETWEEN**, **IN**),
- les opérateurs relationnels (**<**, **<=**, **=**, **>**, **>=**, **<>**).

Les opérateurs et les fonctions peuvent êtres composés entre eux pour donner des expressions très complexes.

Quelques exemples

```
SELECT nom
FROM produits
WHERE prix <= 100.5
```

Liste du nom des produits dont le prix est inférieur ou égale à 100.5 EUR.

```
SELECT nom,prénom
FROM élèves
WHERE age BETWEEN 12 AND 16
```

Liste des nom et prénom des élèves dont l'âge est compris entre 12 et 16 ans.

```
SELECT modèle
FROM voitures
WHERE couleur IN ('rouge', 'blanc', 'noir')
```

Liste des modèles de voiture dont la couleur est dans la liste : rouge, blanc, noir.

```
SELECT modèle
FROM voitures
WHERE couleur NOT IN ('rose', 'violet')
```

Liste des modèles de voiture dont la couleur n'est pas dans la liste : rose, violet.

Fonctions de comparaison de chaînes

Le mot clé **LIKE** permet de comparer deux chaînes.

Le caractère '**%**' est spécial et signifie : 0 ou plusieurs caractères.

Le caractère '**_**' est spécial et signifie : 1 seul caractère, n'importe lequel.

L'exemple suivant permet de rechercher tous les clients sont le prénom commence par 'Jean', cela peut être 'Jean-Pierre', etc… :

```
SELECT nom
FROM clients
WHERE prénom LIKE 'Jean%'
```

Pour utiliser les caractères spéciaux ci-dessus en leur enlevant leur fonction spéciale, il faut les faire précéder de l'antislash : '****'.

Exemple, pour lister les produit dont le code commence par la chaîne '_XE' :

```
SELECT *
FROM produit
WHERE code LIKE '\_XE%'
```

Fonctions mathématiques

Fonction	Description
ABS(x)	Valeur absolue de X.
SIGN(x)	Signe de X, retourne -1, 0 ou 1.
FLOOR(x)	Arrondi à l'entier inférieur.
CEILING(x)	Arrondi à l'entier supérieur.
ROUND(x)	Arrondi à l'entier le plus proche.
EXP(x), LOG(x), SIN(x), COS(x), TAN(x), PI()	Bon, là c'est les fonctions de maths de base…
POW(x,y)	Retourne X à la puissance Y.
RAND(), RAND(x)	Retourne un nombre aléatoire entre 0 et 1.0 Si x est spécifié, entre 0 et X
TRUNCATE(x,y)	Tronque le nombre X à la Yème décimale.

```
SELECT nom
FROM filiales
WHERE SIGN(ca) = -1
ORDER BY RAND()
```

Cet exemple affiche dans un ordre aléatoire le nom des filiales dont le chiffre d'affaire est négatif.
A noter que : SIGN(ca) = -1 ⇔ ca < 0

Fonctions de chaînes

Fonction	Description
TRIM(x)	Supprime les espaces de début et de fin de chaîne.
LOWER(x)	Converti en minuscules.
UPPER(x)	Converti en majuscules.
LONGUEUR(x)	Retourne la taille de la chaîne.
LOCATE(x,y)	Retourne la position de la dernière occurrence de x dans y. Retourne 0 si x n'est pas trouvé dans y.
CONCAT(x,y,…)	Concatène ses arguments.
SUBSTRING(s,i,n)	Retourne les n derniers caractères de s en commençant à partir de la position i.
SOUNDEX(x)	Retourne une représentation phonétique de x.

```
SELECT UPPER(nom)
FROM clients
WHERE SOUNDEX(nom) = SOUNDEX('Dupond')
```

On affiche en majuscules le nom de tous les clients dont le nom ressemble à 'Dupond'.

Fonctions à utiliser dans les GROUP BY

Fonction	Description
COUNT([DISTINCT]x,y,…)	Décompte des tuples du résultat par projection sur le ou les attributs spécifiés (ou tous avec '*'). L'option DISTINCT élimine les doublons.
MIN(x), MAX(x), AVG(x), SUM(x)	Calculent respectivement le minimum, le maximum, la moyenne et la somme des valeurs de l'attribut X.

```
SELECT DISTINCT model
FROM voiture
GROUP BY model
HAVING COUNT(couleur) > 10
```

Ici on affiche le palmarès des models de voitures qui proposent un choix de plus de 10 couleurs.

```
SELECT COUNT(*)
FROM client
```

Affichage de tous les clients.

```
SELECT DISTINCT produit.nom, SUM(vente.qt * produit.prix) AS total
FROM produit, vente
WHERE produit.id = vente.produit_idx
GROUP BY produit.nom
ORDER BY total
```

Classement des produits par la valeur totale vendue.

Créer une relation (I)

La création d'une relation utilise la commande **CREATE TABLE** selon la syntaxe suivante :

CREATE [TEMPORARY] TABLE *nom_relation* [IF NOT EXISTS] (
nom_attribut TYPE_ATTRIBUT [OPTIONS]
…
)

TEMPORARY donne pour durée de vie à la table : le temps de la connexion de l'utilisateur au serveur, après, elle sera détruite. En l'absence de cette option, la table sera permanente à moins d'être détruite par la commande DROP TABLE.

L'option IF NOT EXIST permet de ne créer cette table que si une table de même nom n'existe pas encore.

A l'intérieur des parenthèses, il sera listé tous les attributs, clés et indexs de la table.

Créer une relation (II)

Le type de l'attribut doit être d'un type vu précédemment.

Les options seront vues au fur et à mesure du cours.

Exemple du carnet d'adresse :

CREATE TABLE *Personne* (

nom **VARCHAR(40),**

'*prénom'* **VARCHAR(40),**

adresse **TINYTEXT,**

'*téléphone'* **DECIMAL(10,0)**

)

Notre carnet d'adresse est stocké dans un tableau (appelé ***Relation***) de nom *Personne* qui comporte les colonnes (dites aussi ***attributs***) suivantes : *nom* (chaîne de 40 caractères maximum), *prénom* (idem), *adresse* (texte de longueur variable mais inférieure à 255 caractères) et *téléphone* (chaîne de 10 caractères). Chacune des personnes à ajouter au carnet d'adresse occupera une ligne de cette table. Une ligne est dite ***enregistrement*** dans le jargon des bases de données.

Créer une relation (III)

A sa création, la table peut être remplie par une requête SELECT (qui sera vue en détail plus tard). Par défaut, une table est vide à sa création.

Exemple :

CREATE TABLE *Personne* (
nom **VARCHAR(40),**
'*prénom*' **VARCHAR(40),**
adresse **TINYTEXT,**
'*téléphone*' **DECIMAL(10,0)**
) SELECT *firstname*, *name*, *town*, **tel FROM *Users***

Les options IGNORE et REPLACE à placer entre la parenthèse fermante et le SELECT permettent respectivement d'ignorer les doublons et de remplacer les doublons par la dernière valeur trouvée. Ces options ne sont prises en compte que pour gérer le problème des contraintes d'unicité (UNIQUE, PRIMARY KEY).

Exemple (I)

Imaginons que l'on veuille construire la version web d'un journal papier. Nous devrons créer une table pour stocker les articles de presse. Les informations relatives à un article sont les suivantes : titre, texte, date de parution, auteur, rubrique.

Un titre ayant une longueur raisonnable, il sera de type VARCHAR(80), le texte pourra être très grand : TEXT (65535 caractères !), la date sera au format DATE (YYYY:MM:JJ). L'auteur pourra être codé sur un VARCHAR(80). Et la rubrique pourrait être un ENUM.

```
CREATE TABLE article (
   id MEDIUM INT UNSIGNED PRIMARY KEY,
   titre VARCHAR(80),
   texte TEXT,
   parution DATE,
   auteur VARCHAR(80),
   rubrique ENUM('économie','sports','international','politique','culture')
)
```

Exemple (II)

Cette définition apporte certaines limitations : le nombre et le nom des rubriques sont fixés à la création de la relation (CREATE TABLE). Bien sûr, il sera toujours possible de modifier la définition de la table (ALTER TABLE) pour modifier ou ajouter une rubrique ; mais ce ne sera pas pratique du tout.

On peut imaginer une interface web qui permette à un administrateur d'ajouter, de renommer ou de supprimer des rubriques. Pour cela on va créer une nouvelle relation : la table rubrique.

La relation article contiendra non plus le nom de la rubrique mais une référence vers le nom de cette rubrique. Ainsi, lors d'une sélection, il faudra faire une jointure entre les deux tables 'article' et 'rubrique' pour connaître le nom de la rubrique associée à un article.

```
CREATE TABLE article (
  …
  rubrique_idx TINYINT
)
```

```
CREATE TABLE rubrique (
  id TINYINT UNSIGNED PRIMARY KEY,
  label VARCHAR(40)
)
```

```
SELECT *
FROM article,rubrique
WHERE article.rubrique_idx=rubrique.id
```

Clé primaire (II)

Notre exemple devient :

```
CREATE TABLE Personne (
        id SMALLINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
        nom VARCHAR(40),
        'prénom' VARCHAR(40),
        adresse TINYTEXT,
        'téléphone' DECIMAL(10,0)
)
```

Cet identifiant numérique unique auto-incrémental, s'appelle une « *clé primaire* ».

La numérotation des clés primaires, débute à 1 et pas à 0.

Personnes

Id	*nom*	*prénom*	*adresse*	*téléphone*
1	Dupond	Marc	8 rue de l'octet	0123456789

Clé primaire (III)

Notre clé primaire peut être associée simultanément à plusieurs attributs mais selon une syntaxe différente.

Si au lieu de créer un identifiant numérique unique, on souhaite simplement interdire d'avoir des doublon sur le couple (*nom*,'*prénom*') et d'en interdire la nullité, on va créer une clé primaire sur ce couple.

La connaissance des seuls nom et prénom suffit à identifier sans ambiguïté un et un seul enregistrement.

Mauvaise syntaxe :

```
CREATE TABLE Personne (
        nom VARCHAR(40) PRIMARY KEY,
        'prénom' VARCHAR(40) PRIMARY KEY,
        adresse TINYTEXT,
        'téléphone' DECIMAL(10,0)
)
```

Bonne syntaxe :

```
CREATE TABLE Personne (
        nom VARCHAR(40),
        'prénom' VARCHAR(40),
        adresse TINYTEXT,
        'téléphone' DECIMAL(10,0),
        PRIMARY KEY (nom,'prénom')
)
```

Ajouter un enregistrement (I) *insertion étendue*

Ajouter un enregistrement à une relation revient à ajouter une ligne à la table. Pour cela, pour chacun des attributs, il faudra en préciser la valeur. Si certaines valeurs sont omises, alors les valeurs par défauts définie les de la création de la relation seront utilisées. Si on ne dispose pas non plus de ces valeurs par défaut, alors MySQL mettra 0 pour un nombre, ‘’’ pour une chaîne, 0000-00-00 pour une date, 00:00:00 pour une heure, 00000000000000 pour un timestamp (si le champs poste la contrainte NOT NULL). Dans le cas où l’attribut porte la contrainte NULL (par défaut) alors la valeur par défaut de l’attribut – quel soit sont type – sera la suivante : NULL.

Syntaxe d’une « insertion étendue » :

INSERT INTO *relation*(*liste des attributs***) VALUES(***liste des valeurs***)**

Exemple :

INSERT INTO *Personnes*(*nom*,*prénom*) VALUES(‘Martin’,’Jean’)

REPLACE est un synonyme de INSERT, mais sans doublon. Pratique pour respecter les contraintes d’unicité (UNIQUE, PRIMARY KEY).

Modifier un enregistrement (I)

Pour modifier un ou des enregistrement(s) d'une relation, il faut préciser un critère de sélection des enregistrement à modifier (clause **WHERE**), il faut aussi dire quels sont les attributs dont on va modifier la valeur et quelles sont ces nouvelles valeurs (clause **SET**).

Syntaxe :

UPDATE [LOW_PRORITY] ***relation*** **SET** *attribut*=valeur**, … [WHERE condition] [LIMIT a]**

Exemple :

UPDATE ***Personnes*** **SET** *téléphone*=**'0156281469' WHERE** *nom*=**'Martin' AND** *prénom* **= 'Pierre'**

Cet exemple modifie le numéro de téléphone de Martin Pierre.

LOW_PRORITY est une option un peu spéciale qui permet de n'appliquer la ou les modification(s) qu'une fois que plus personne n'est en train de lire dans la relation.

Supprimer un enregistrement

Attention, la suppression est définitive !

Syntaxe :

DELETE [LOW_PRIORITY] FROM *relation* [WHERE condition] [LIMIT a]

Exemple :

DELETE FROM *Personnes* WHERE *nom*='Martin' AND *prénom*='Marc'

Pour vider une table de tous ces éléments, ne pas mettre de clause WHERE. Cela efface et recrée la table, au lieu de supprimer un à un chacun des tuples de la table (ce qui serait très long).

Exemple :

DELETE FROM *Personnes*

5 Administration avec l'outil web phpMyAdmin

Présentation

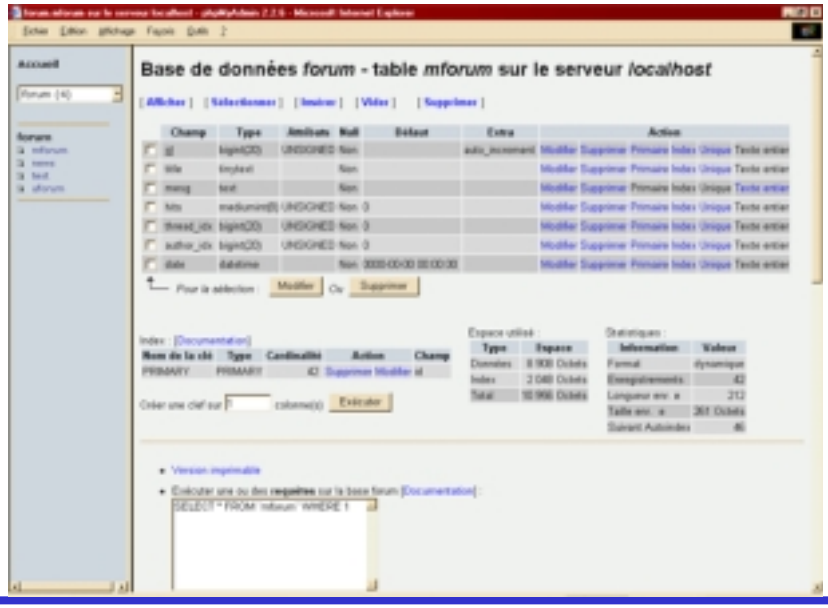
L'outil phpMyAdmin est développé en PHP et offre une interface intuitive pour l'administration des base de données du serveur.

Il est téléchargeable ici : http://phpmyadmin.sourceforge.net

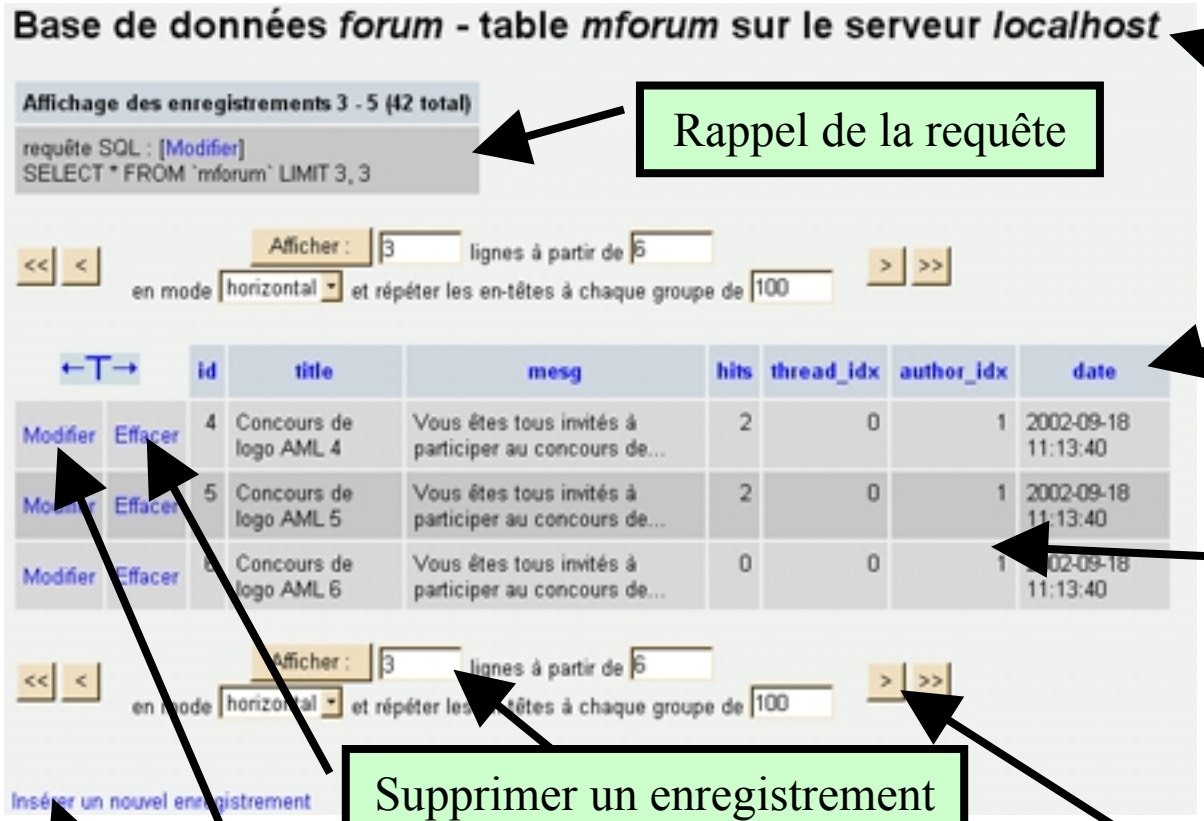
Cet outil permet de :

- créer de nouvelles bases
- créer/modifier/supprimer des tables
- afficher/ajouter/modifier/supprimer des tupes dans des tables
- effectuer des sauvegarde de la structure et/ou des données
- effectuer n'importe quelle requête
- gérer les privilèges des utilisateurs

Les exemples qui vont suivrent s'appliquent à la version 2.2.6



Affichage d'une table



Rappel de la base, de la table et du serveur

Rappel de la requête

Colonnes = noms des attributs de la table

Liste des enregistrements de la table par pages de X lignes

Supprimer un enregistrement

Accès au formulaire de modification d'un enregistrement

Permet de naviguer dans les pages de résultats

Insertion d'un nouvel enregistrement

Afficher par page de X lignes

Historique

- **20 juillet 2003** : insertions complètes et étendues ; jointures (105 diapos)
- **9 mars 2003** : corrections, aide phpMyAdmin (101 diapos)
- **17 décembre 2002** : plus d'exemples, api php, jointures (90 diapos)
- **9 décembre 2002** : première publication (73 diapos)
- **5 décembre 2002** : création du document par Hugo Etiévant (54 diapos)

Remerciement à tous ceux qui – part leurs suggestions et critiques – font progresser la qualité de ce document.

Ce cours s'inspire des ressources suivantes :

- cours de SQL de M. PICHAT (UCBL)
- transparents de Mohand-Saïd HACID (LISI, UCBL)
- manuel MySQL (Nexen)

Hugo Etiévant
cyberzoide@yahoo.fr
http://cyberzoide.developpez.com/