
MySQL

pour booster votre site web PHP

Hugo Etiévant

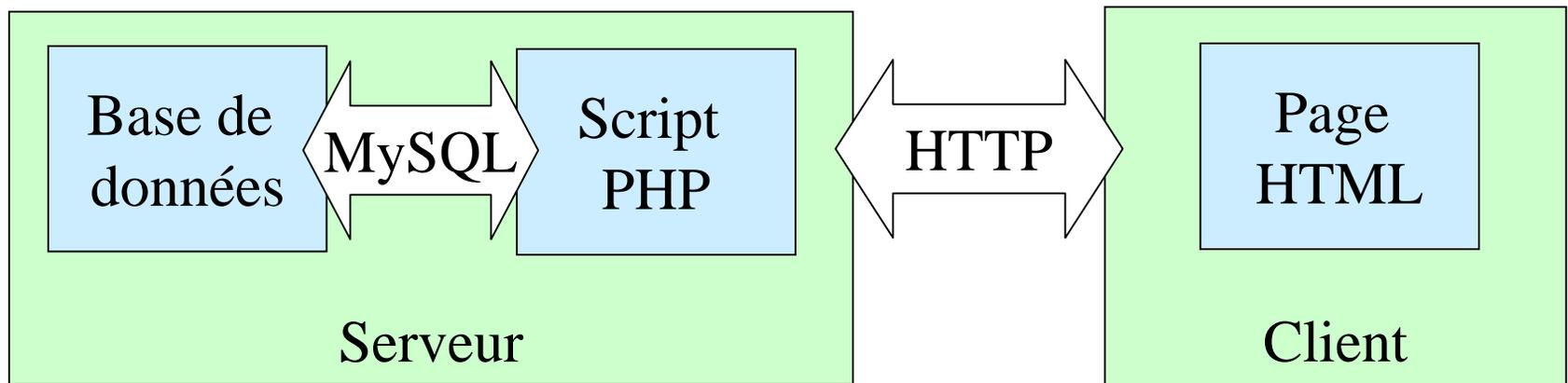
Dernière mise à jour : 20 juillet 2003

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

1

Théorie des bases de données

Concepts du modèle relationnel

Avant d'attaquer le vif du sujet, un petit glossaire du jargon des bases de données :

Domaine : ensemble des valeurs d'un attribut.

Relation : sous ensemble du produit cartésien d'une liste de domaines. C'est en fait un tableau à deux dimensions dont les colonnes correspondent aux domaines et dont les lignes contiennent des tuples. On associe un nom à chaque colonne.

Attribut : une colonne d'une relation, caractérisé par un nom.

Tuple : liste des valeurs d'une ligne d'une relation.

Une relation est un peu une classe (programmation orientée objet) qui ne posséderait que des attributs et donc chaque instance représenterait un tuple.

Les relations

Une *relation* est une *table* comportant des *colonnes* (appelées aussi *attributs*) dont le nom et le *type* caractérisent le contenu qui sera inséré dans la table.

Imaginons que l'on veuille stocker dans notre base de données notre carnet d'adresses. On va donc créer la relation **Personne** qui aura pour attributs : *nom*, *prénom*, *adresse*, *téléphone*. Autrement dit, c'est une table nommée **Personne** possédant les colonnes : *nom*, *prénom*, *adresse*, *téléphone*.

Les *lignes* que contiendra cette table seront appelées *enregistrements* ou *tuples*.

Personnes

<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
Dupond	Marc	8 rue de l'octet	0123456789

Algèbre relationnelle

L'algèbre relationnelle regroupe toutes les opérations possibles sur les relations. Voici la liste des opérations possibles :

Projection : on ne sélectionne qu'un ou plusieurs attributs d'une relation (on ignore les autres). Par exemple n'afficher que les colonnes *nom* et *prénom* de la table **Personnes**.

Jointure : on fabrique une nouvelle relation à partir de 2 ou plusieurs autres en prenant comme pivot 1 ou plusieurs attributs. Par exemple, on concatène la table du carnet d'adresse et celle des inscrits à la bibliothèque en fonction du nom de famille (c'est typiquement du recoupement de fichiers).

Sélection : on sélectionne tous les tuples ou bien seulement une partie en fonction de critères de sélection qui portent sur les valeurs des attributs. Par exemple n'afficher que les lignes de la table **Personnes** qui vérifient la condition suivante : le nom ne commence pas par la lettre 'C'.

Cette algèbre est facilement possible avec les commandes de MySQL (SELECT... FROM... WHERE...).

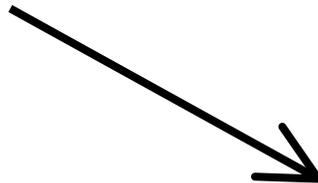
Projection

Personnes

<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
Martin	Pierre	7 allée des vers	0258941236
Dupond	Jean	32 allé Poivrot	0526389152
Dupond	Marc	8 rue de l'octet	0123456789



```
SELECT nom, prénom  
FROM Personnes
```



<i>nom</i>	<i>prénom</i>
Martin	Pierre
Dupond	Jean
Dupond	Marc

On projette la table ***Personnes*** sur les colonnes *nom* et *prénom*.

Jointure

Personnes

<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
Martin	Pierre	7 allée des vers	0258941236
Dupond	Jean	32 allé Poivrot	0526389152

Bibliothèque

<i>nom</i>	<i>Dernierlivre</i>
Dupond	Robinson
Jospin	Faust
Martin	Misère

SELECT *Personnes.prénom, dernierlivre*
FROM *Personnes, Bibliothèque*
WHERE *Personnes.nom = Bibliothèque.nom*

<i>prénom</i>	<i>Dernierlivre</i>
Jean	Robinson
Pierre	Misère

On joint les deux tables, grâce à la colonne *nom*.

Et on combine cette jointure à une projection sur les attributs *nom* et *dernierlivre*.

Attention à lever toute ambiguïté sur les noms d'attribut dans le cas où deux tables possèdent des colonnes de même nom.

Sélection

Personnes

<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
Martin	Pierre	7 allée des vers	0258941236
Dupond	Jean	32 allé Poivrot	0526389152
Dupond	Marc	8 rue de l'octet	0123456789



```
SELECT *  
FROM Personnes  
WHERE nom = "Dupond"
```



<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
Dupond	Jean	32 allé Poivrot	0526389152
Dupond	Marc	8 rue de l'octet	0123456789

On ne sélectionne que les tuples dont l'attribut *nom* est égale à 'Dupond'.

2

Syntaxe de MySQL

Types des attributs (I)

Les propriétés de vos objets peuvent être de types très différents :

- Nombre entier signé ou non (température, quantité commandée, âge)
- Nombre à virgule (prix, taille)
- Chaîne de caractères (nom, adresse, article de presse)
- Date et heure (date de naissance, heure de parution)
- Énumération (une couleur parmi une liste prédéfinie)
- Ensemble (une ou des monnaies parmi une liste prédéfinie)

Il s'agit de choisir le plus adapté à vos besoins.

Ces types requièrent une plus ou moins grande quantité de données à stocker. Par exemple, ne pas choisir un LONGTEXT pour stocker un prénom mais plutôt un VARCHAR(40) !

Types des attributs (II) – entiers

nom	borne inférieure	borne supérieure
TINYINT	-128	127
TINYINT UNSIGNED	0	255
SMALLINT	-32768	32767
SMALLINT UNSIGNED	0	65535
MEDIUMINT	-8388608	8388607
MEDIUMINT UNSIGNED	0	16777215
INT*	-2147483648	2147483647
INT* UNSIGNED	0	4294967295
BIGINT	-9223372036854775808	9223372036854775807
BIGINT UNSIGNED	0	18446744073709551615

(*) : **INTEGER** est un synonyme de **INT**.

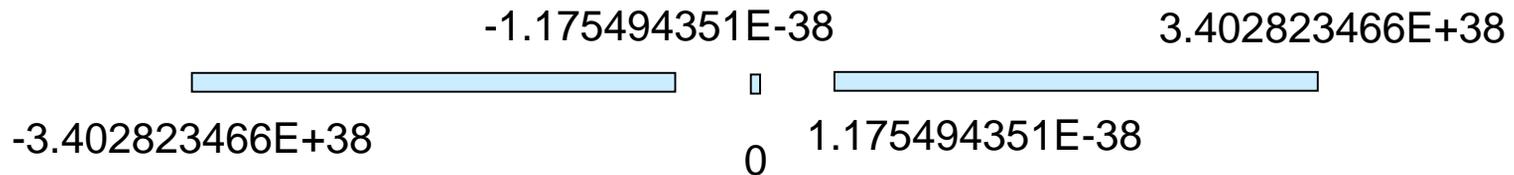
UNSIGNED permet d'avoir un type non signé.

ZEROFILL : remplissage des zéros non significatifs.

Types des attributs (III) – flottants

Les flottants – dits aussi nombres réels – sont des nombres à virgule. Contrairement aux entiers, leur domaine n'est pas continu du fait de l'impossibilité de les représenter avec une précision absolue.

Exemple du type **FLOAT** :



nom	domaine négatif : borne inférieure borne supérieure	Domaine positif : borne inférieure borne supérieure
FLOAT	-3.402823466E+38 -1.175494351E-38	1.175494351E-38 3.402823466E+38
DOUBLE*	-1.7976931348623157E+308 -2.2250738585072014E-308	2.2250738585072014E-308 1.7976931348623157E+308

(*) : **REAL** est un synonyme de **DOUBLE**.

Types des attributs (IV) – chaînes

nom	longueur
CHAR(M)	Chaîne de taille fixée à M, où $1 < M < 255$, complétée avec des espaces si nécessaire.
CHAR(M) BINARY	Idem, mais insensible à la casse lors des tris et recherches.
VARCHAR(M)	Chaîne de taille variable, de taille maximum M, où $1 < M < 255$, complété avec des espaces si nécessaire.
VARCHAR(M) BINARY	Idem, mais insensible à la casse lors des tris et recherches.
TINYTEXT	Longueur maximale de 255 caractères.
TEXT	Longueur maximale de 65535 caractères.
MEDIUMTEXT	Longueur maximale de 16777215 caractères.
LONGTEXT	Longueur maximale de 4294967295 caractères.
DECIMAL(M,D)*	Simule un nombre flottant de D chiffres après la virgule et de M chiffres au maximum. Chaque chiffre ainsi que la virgule et le signe moins (pas le plus) occupe un caractère.

(*) : **NUMERIC** est un synonyme de **DECIMAL**.

Types des attributs (V) – chaînes

Les types TINYTEXT, TEXT, MEDIUMTEXT et LONGTEXT peuvent être judicieusement remplacés respectivement par TINYBLOB, BLOB, MEDIUMBLOB et LONGBLOB.

Ils ne diffèrent que par la sensibilité à la casse qui caractérise la famille des BLOB. Alors que la famille des TEXT sont insensibles à la casse lors des tris et recherches.

Les BLOB peuvent être utilisés pour stocker des données binaires.

Les VARCHAR, TEXT et BLOB sont de taille variable. Alors que les CHAR et DECIMAL sont de taille fixe.

Types des attributs (VI) – dates et heures

nom	description
DATE	Date au format anglophone AAAA-MM-JJ.
DATETIME	Date et heure au format anglophone AAAA-MM-JJ HH:MM:SS.
TIMESTAMP	Affiche la date et l'heure sans séparateur : AAAAMMJJHHMMSS.
TIMESTAMP(M)	Idem mais M vaut un entier pair entre 2 et 14. Affiche les M premiers caractères de TIMESTAMP .
TIME	Heure au format HH:MM:SS.
YEAR	Année au format AAAA.

nom	description
TIMESTAMP(2)	AA
TIMESTAMP(4)	AAMM
TIMESTAMP(6)	AAMMJJ
TIMESTAMP(8)	AAAAMMJJ
TIMESTAMP(10)	AAMMJJHHMM
TIMESTAMP(12)	AAMMJJHHMMSS
TIMESTAMP(14)	AAAAMMJJHHMMSS

En cas d'insertion d'un enregistrement en laissant vide un attribut de type **TIMESTAMP**, celui-ci prendra automatiquement la date et heure de l'insertion. Contrairement à Unix (où le timestamp est le nombre de secondes écoulées depuis le 1er janvier 1970), en MySQL, il est une chaîne de format comme indiqué ci-contre.

Types des attributs (VII) – dates et heures

nom	description
DATE	Date au format anglophone AAAA-MM-JJ.
DATETIME	Date et heure au format anglophone AAAA-MM-JJ HH:MM:SS.
TIMESTAMP	Affiche la date et l'heure sans séparateur : AAAAMMJJHHMMSS.
TIMESTAMP(M)	Idem mais M vaut un entier pair entre 2 et 14. Affiche les M premiers caractères de TIMESTAMP .
TIME	Heure au format HH:MM:SS.
YEAR	Année au format AAAA.

Le format de date AAAA-MM-JJ signifie : année sur 4 chiffre, mois sur 2 chiffres et jour sur 2 chiffres avec pour séparateur le tiret. Le format d'heure HH:MM:SS signifie : heure, minute et seconde chacune sur 2 chiffres, avec pour séparateur les deux points.

Dans le format étendu qui comprend la date et l'heure, des deux dernières sont séparées par un espace. Les formats de date sont assez permissifs car des variantes sont tolérées. Il est possible de mettre n'importe quel caractère qui ne soit pas un chiffre en guise de séparateur, il est aussi possible de ne pas en mettre, comme cela est affiché par **TIMESTAMP**.

Types des attributs (VIII) – énumérations

Un attribut de type **ENUM** peut prendre une valeur parmi celles définies lors de la création de la table plus la chaîne vide ainsi que **NULL** si la définition le permet. Ces valeurs sont exclusivement des chaînes de caractères. Une énumération peut contenir 65535 éléments au maximum.

Définition d'un tel attribut :

nom_attribut ENUM("valeur 1", "valeur 2" ...)

nom_attribut ENUM("valeur 1", "valeur 2" ...) NULL

A chaque valeur est associée un index allant de 0 à N si N valeurs ont été définies. L'index 0 est associé à la chaîne nulle, l'index 1 à la première valeur... L'index **NULL** est associé à la valeur **NULL**.

Si une sélection (**SELECT** ou **WHERE**) est faite dans un contexte numérique, l'index est renvoyé. Sinon, c'est la valeur qui est retournée.

Il peut être défini jusqu'à 65535 valeurs distinctes insensibles à la casse.

Types des attributs (IX) – ensembles

Un attribut de type **SET** peut prendre pour valeur la chaîne vide, **NULL** ou une chaîne contenant une liste de valeurs qui doivent être déclarées lors de la définition de l'attribut lors de la création de la relation.

Par exemple, un attribut déclaré comme ci :

SET("voiture", "moto", "vélo") NOT NULL

peut prendre les valeurs suivantes :

"" (chaîne vide)

"voiture,moto"

"vélo,voiture,moto"

et autres combinaisons de listes des trois valeurs définie plus haut.

Un attribut déclaré comme suit :

SET("voiture", "moto", "vélo") NULL

peut prendre, en plus ce celles précédentes, la valeur **NULL**.

Il ne peut être défini que 64 éléments maximum.

Identificateurs

Les noms des bases, relations, attributs, index et alias sont constitués de caractères alphanumériques et des caractères `_` et `$`.

Un nom comporte au maximum 64 caractères.

Comme les bases de données et les relations sont codées directement dans le système de fichiers, la sensibilité à la casse de MySQL dépend de celle du système d'exploitation sur lequel il repose. Sous Windows, la casse n'a pas d'importance ; alors que sous Unix, elle en a !

Le point `.` est un caractère réservé utilisé comme séparateur entre le nom d'une base et celui d'une relation, entre le nom d'une relation et celui d'un attribut.

Exemple :

```
SELECT base1.table25.attribut5  
FROM base1.table25
```

Exemple (I)

Imaginons que l'on veuille construire la version web d'un journal papier. Nous devons 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
```

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).

Clé primaire (I)

Pour des raisons pratiques, nous souhaitons pouvoir associer à chacun des enregistrements de la relation un identifiant numérique unique qui puisse être passé en paramètre à nos scripts PHP.

Pour cela on rajoute un nouvelle attribut de type entier. Pour nous faciliter la tâche, cet entier ne devra pas être signé mais être suffisamment grand pour identifier tous nos enregistrements car destiné à un décompte (donc débute forcément à 1 et pas à -127 par exemple).

Dans notre exemple, le carnet d'adresse ne devrait pas excéder plusieurs centaines de personnes. Ainsi un attribut de type **SMALLINT UNSIGNED** devrait faire l'affaire. Nous le nommerons par la suite : *id*.

Cet attribut devra ne jamais être vide, il faut donc préciser l'option **NOT NULL** pour le forcer à prendre une valeur de son domaine (entre 0 et 65535).

Il devra aussi être unique, c'est-à-dire que deux enregistrements ne pourront pas avoir une valeur identique de *id*. Il faut alors faire la déclaration suivante : **UNIQUE (id)** à la suite de la liste des attributs.

Pour simplifier, on utilisera l'option **PRIMARY KEY** qui regroupe **NOT NULL** et **UNIQUE** en remplacement des deux dernières déclarations.

Et pour finir, il faut signifier que cette valeur doit s'incrémenter automatiquement à chaque insertion d'un enregistrement grâce à l'option **AUTO_INCREMENT**.

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

<i>Id</i>	<i>nom</i>	<i>prénom</i>	<i>adresse</i>	<i>téléphone</i>
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')  
)
```

Attribut non nul

Considérons que l'on souhaite que certains attributs aient obligatoirement une valeur. On utilisera l'option **NOT NULL**.

Dans ce cas, si malgré tout, aucune valeur n'est fournie, la valeur par défaut – si elle est déclarée à la création de la relation – sera automatiquement affectée à cet attribut dans l'enregistrement.

Si aucune valeur par défaut n'est déclarée :

- la chaîne vide "" sera affectée à l'attribut s'il est de type chaîne de caractères
- la valeur zéro 0 s'il est de type nombre
- la date nulle 0000-00-00 et/ou l'heure nulle 00:00:00 s'il est de type date, heure ou date et heure.

Exemple :

adresse **TINYTEXT NOT NULL**

Au contraire, on utilisera l'option **NULL** si on autorise l'absence de valeur.

Valeur par défaut

Pour donner une valeur par défaut à un attribut, on utilise l'option **DEFAULT**. Lors de l'ajout d'un enregistrement cette valeur sera affectée à l'attribut si aucune valeur n'est donnée.

Exemple :

'téléphone' **DECIMAL(10,0) DEFAULT '0123456789'**

Les attributs de type chaîne de caractères de la famille TEXT et BLOB ne peuvent pas avoir de valeur par défaut.

Attribut sans doublon (I)

Pour interdire l'apparition de doublon pour un attribut, on utilise l'option **UNIQUE**.

Syntaxe :

UNIQUE [*nomde la contrainte*](*liste des attributs*)

Exemple, pour interdire tout doublon de l'attribut *nom* :

UNIQUE(*nom*)

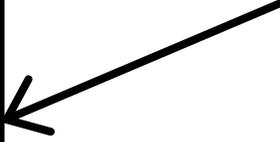
Pour interdire les doublons sur l'attribut *nom* mais les interdire aussi sur '*prénom*', tout en les laissant indépendants :

UNIQUE(*nom*)

UNIQUE('prénom')

<i>nom</i>	<i>prénom</i>
Dupond	Marc
Dupont	Pierre
Martin	Marc

enregistrement interdit
car 'Marc' est un doublon
dans la colonne 'prénom'



Attribut sans doublon (II)

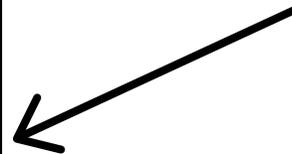
Pour interdire tout doublon à un ensemble d'attributs (tuple), on passe en paramètre à **UNIQUE** la liste des attributs concernés.

Pour interdire tout doublon du couple (*nom*, '*prénom*') :

UNIQUE(*nom*, '*prénom*')

<i>nom</i>	<i>prénom</i>
Dupond	Marc
Dupont	Pierre
Martin	Marc
Martin	Pierre
Martin	Marc

enregistrement interdit car le couple ('Martin', 'Marc') est un doublon du couple (*nom*, '*prénom*')



Index (I)

Lors de la recherche d'informations dans une relation, MySQL parcourt la table correspondante dans n'importe quel ordre. Dans le cas d'un grand nombre de lignes, cette recherche est très très longue du fait du parcours de TOUTE la table.

Pour y remédier, une optimisation possible et **FORTEMENT** recommandée, est d'utiliser des indexs.

La création d'un index associé à un attribut ou à un ensemble ordonné d'attributs va créer une liste ordonnée des valeurs de ces attributs et de l'adresse de la ligne associée. C'est sur les valeurs de cette liste que se fera les recherches et les tris. Les algorithmes de recherche et de tri sur des ensembles ordonnées sont énormément plus rapides !

Ainsi, d'une recherche à coût prohibitif, on passe à une recherche sur un ensemble déjà trié. On gagne donc énormément en temps d'accès aux informations. Bien que cela ralentisse les mises à jour (insertion, suppression, modification de clé).

On choisira de créer des indexs sur les attributs qui seront les plus sollicités par les recherches ou utilisés comme critère de jointure. Par contre, on épargnera les attributs qui contiennent peu de valeurs différentes les unes des autres et ceux dont les valeurs sont très fréquemment modifiées.

Index (II)

Syntaxe :

INDEX *index* (*liste des attributs*)

Exemple, pour créer un index sur les 3 premiers caractères seulement de l'attribut *nom* :

INDEX *idx_nom* (*nom*(3))

Exemple, pour créer un index sur le couple (*nom*, '*prénom*') :

INDEX *idx_nom_prenom* (*nom*, '*prénom*')

Un index peut porter sur 15 colonnes maximum.

Une table peut posséder au maximum 16 indexs.

Un index peut avoir une taille d'au maximum 256 octets et ne doit porter que sur des attributs NOT NULL.

Il suffit de suffixer l'attribut (CHAR, VARCHAR) pour dire de ne prendre que les M premiers caractères pour l'indexation.

Supprimer une relation

La commande **DROP TABLE** prend en paramètre le nom de la table à supprimer. Toutes les données qu'elle contient sont supprimées et sa définition aussi.

Syntaxe :

DROP TABLE *relation*

Exemple :

DROP TABLE *Personnes*

Si un beau jour on s'aperçoit qu'une relation a été mal définie au départ, plutôt que de la supprimer et de la reconstruire bien comme il faut, on peut la modifier très simplement. Cela évite de perdre les données qu'elle contient.

Modifier une relation

La création d'une relation par **CREATE TABLE** n'en rend pas définitives les spécifications. Il est possible d'en modifier la définition par la suite, à tout moment par la commande **ALTER TABLE**.

Voici ce qu'il est possible de réaliser :

- ajouter/supprimer un attribut
- créer/supprimer une clé primaire
- ajouter une contrainte d'unicité (interdire les doublons)
- changer la valeur par défaut d'un attribut
- changer totalement la définition d'un attribut
- changer le nom de la relation
- ajouter/supprimer un index

Ajouter un attribut

Syntaxe :

```
ALTER TABLE relation ADD definition [ FIRST | AFTER attribut]
```

Ajoutons l'attribut *fax* qui est une chaîne représentant un nombre de 10 chiffres:

```
ALTER TABLE Personnes ADD fax DECIMAL(10,0)
```

Nous aurions pu forcer la place où doit apparaître cet attribut. Pour le mettre en tête de la liste des attributs de la relation, il faut ajouter l'option **FIRST** en fin de commande. Pour le mettre après l'attribut '*téléphone*', il aurait fallu ajouter **AFTER** '*téléphone*'.

Note : il ne doit pas déjà avoir dans la relation un attribut du même nom !

Supprimer un attribut (I)

Attention, supprimer un attribut implique la suppression des valeurs qui se trouvent dans la colonne qui correspond à cet attribut, sauf à utiliser l'option IGNORE.

Syntaxe :

```
ALTER TABLE relation DROP attribut
```

Exemple :

```
ALTER TABLE Personnes DROP 'prénom'
```

Supprimer un attribut (II)

La suppression d'un attribut peut incidemment provoquer des erreurs sur les contraintes clé primaire (**PRIMARY KEY**) et unique (**UNIQUE**).

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

```
ALTER TABLE Personnes DROP 'prénom'
```

<i>nom</i>	<i>prénom</i>
Dupond	Marc
Martin	Marc
Martin	Pierre

<i>nom</i>
Dupond
Martin
Martin

Refus d'opérer la suppression, car cela contredirait la contrainte d'unicité qui resterait sur l'attribut *nom*.

Créer une clé primaire

La création d'une clé primaire n'est possible qu'en l'absence de clé primaire dans la relation.

Syntaxe :

```
ALTER TABLE relation ADD PRIMARY KEY (attribut)
```

Exemple :

```
ALTER TABLE Personnes ADD PRIMARY KEY (nom, 'prénom')
```

Supprimer une clé primaire

Comme une clé primaire est unique, il n'y a aucune ambiguïté lors de la suppression.

Syntaxe :

ALTER TABLE *relation* DROP PRIMARY KEY

Exemple :

ALTER TABLE *Personnes* ADD PRIMARY KEY

S'il n'y a aucune clé primaire lorsque cette commande est exécutée, aucun message d'erreur ne sera généré, la commande sera simplement ignorée.

Ajout d'une contrainte d'unicité

Il est possible (facultatif) de donner un nom à la contrainte.

Cette contrainte peut s'appliquer à plusieurs attributs.

Si les valeurs déjà présentes dans la relation sont en contradiction avec cette nouvelle contrainte, alors cette dernière ne sera pas appliquée et une erreur sera générée.

Syntaxe :

```
ALTER TABLE relation ADD UNIQUE [contrainte] (attributs)
```

Exemple pour interdire tout doublon sur l'attribut *fax* de la relation ***Personnes*** :

```
ALTER TABLE Personnes ADD UNIQUE u_fax (fax)
```

Autre exemple fictif :

```
ALTER TABLE Moto ADD UNIQUE u_coul_vitre (couleur, vitre)
```

Changer la valeur par défaut d'un attribut

Pour changer ou supprimer la valeur par défaut d'un attribut.

Attention aux types qui n'acceptent pas de valeur par défaut (les familles **BLOB** et **TEXT**).

Syntaxe :

```
ALTER TABLE relation ALTER attribut { SET DEFAULT valeur |  
DROP DEFAULT }
```

Changer sa valeur par défaut :

```
ALTER TABLE Personnes ALTER 'téléphone' SET DEFAULT  
'9999999999'
```

Supprimer sa valeur par défaut :

```
ALTER TABLE Personnes ALTER 'téléphone' DROP DEFAULT
```

Le changement ou la suppression n'affecte en rien les enregistrements qui ont eu recours à cette valeur lors de leur insertion.

Changer la définition d'un attribut

Pour changer la définition de l'attribut sans le renommer :

ALTER TABLE *relation* MODIFY *attribut* *definition_relative*

Exemple 1 :

ALTER TABLE *Personnes* MODIFY *fax* VARCHAR(14)

Pour changer sa définition en le renommant :

ALTER TABLE *relation* CHANGE *attribut* *definition_absolue*

Exemple 2 :

ALTER TABLE *Personnes* CHANGE *fax* *num_fax* VARCHAR(14)

Attention, si le nouveau type appliqué à l'attribut est incompatible avec les valeurs des enregistrements déjà présents dans la relation, alors elles risquent d'être modifiées ou remises à zéro !

Changer le nom de la relation

Syntaxe :

```
ALTER TABLE relation RENAME nouveau_nom
```

Exemple :

```
ALTER TABLE Personnes RENAME Carnet
```

Cela consiste à renommer la table, et donc le fichier qui la stocke.

Ajouter un index

Une table ne peut comporter que 32 indexs.

Et un index ne peut porter que sur 16 attributs maximum à la fois.

Syntaxe :

```
ALTER TABLE relation ADD INDEX index (attributs)
```

Exemple :

```
ALTER TABLE Personnes ADD INDEX nom_complet  
(nom,prénom)
```

Dans cet exemple, on a ajouté à la relation ***Personnes*** un index que l'on nomme *nom_complet* et qui s'applique aux deux attributs *nom* et '*prénom*'. Ainsi, les recherches et les tris sur les attributs *nom* et '*prénom*' seront grandement améliorés. Car un index apporte les changements sous-jacents permettant d'optimiser les performances du serveur de base de données.

Supprimer un index

Syntaxe :

```
ALTER TABLE relation DROP INDEX index
```

Exemple :

```
ALTER TABLE Personnes DROP INDEX nom_complet
```

Cette exemple permet de supprimer l'index nommé *nom_complet* de la relation ***Personnes***.

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 définies lors 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, 0000000000000000 pour un timestamp (si le champ porte 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 son 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).

Ajouter un enregistrement (II) insertion standard

Une syntaxe plus courte mais plus ambiguë permet d'insérer un enregistrement dans une table. Elle consiste à omettre la liste des noms d'attribut à la suite du nom de la relation. Cela impose que la liste des valeurs suivant le mot clé VALUES soit exactement celle définie dans la table et qu'elles soient dans l'ordre défini dans la définition de la table ; sinon des erreurs se produiront.

Syntaxe d'une « insertion standard » :

INSERT INTO *relation* VALUES(*liste exhaustive et ordonnée des valeurs*)

Exemple :

```
CREATE TABLE Ballon (  
    taille INT NOT NULL,  
    couleur VARCHAR(40)  
)
```

INSERT INTO *Ballon* VALUES(20, 'rouge') ok

INSERT INTO *Ballon* VALUES('rouge', 20) faux

INSERT INTO *Ballon* VALUES('rouge') faux

Ajouter un enregistrement (III) insertion complète

Dans le cas où l'on souhaite procéder à l'insertion de plusieurs enregistrements les uns à la suite des autres, il y a deux méthodes :

- faire une boucle qui envoie autant d'INSERT que nécessaire au serveur
- faire une insertion dite « complète »

Syntaxe d'une « insertion complète » :

INSERT INTO *relation* VALUES (liste des valeurs), (liste d'autres valeurs), (liste d'encore d'autres valeurs), ...

Exemple :

INSERT INTO *Ballon* VALUES (20, 'rouge'), (35, 'vert fluo'), (17, 'orange'), (28, 'céruleen')

Cet exemple est équivalent aux requêtes suivantes :

INSERT INTO *Ballon* VALUES(20, 'rouge')

INSERT INTO *Ballon* VALUES(35, 'vert fluo')

INSERT INTO *Ballon* VALUES(17, 'orange')

INSERT INTO *Ballon* VALUES(28, 'céruleen')

Ajouter un enregistrement (IV) insertion complète

L'insertion complète permet d'insérer plusieurs enregistrements dans une même table. Une insertion complète ne permet pas d'insérer des données dans plusieurs tables différentes.

L'insertion complète et l'insertion étendue peuvent être associées dans une même requête :

```
INSERT INTO Ballon(taille, couleur) VALUES (20, 'rouge'), (35, 'vert fluo'), (17, 'orange'), (28, 'céruleen')
```

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_PRIORITY ] 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_PRIORITY 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.

Modifier un enregistrement (II)

Il est possible de modifier les valeurs d'autant d'attributs que la relation en contient.

Exemple pour modifier plusieurs attributs :

```
UPDATE Personnes SET téléphone='0156281469',  
fax='0156281812' WHERE id = 102
```

Pour appliquer la modification à tous les enregistrements de la relation, il suffit de ne pas mettre de clause **WHERE**.

LIMIT a permet de n'appliquer la commande qu'aux **a** premiers enregistrements satisfaisant la condition définie par **WHERE**.

Autre exemple :

```
UPDATE Enfants SET age=age+1
```

Il est donc possible de modifier la valeur d'un attribut relativement à sa valeur déjà existante.

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
```

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
```

Sélectionner des enregistrements (IV)

Relation de départ :

SELECT * FROM Gens

Gens		
Nom	Prenom	Age
Dupond	Pierre	24
Martin	Marc	48
Dupont	Jean	51
Martin	Paul	36
Dupond	Lionel	68
Chirac	Jacques	70

1

SELECT Nom FROM Gens

Gens
Nom
Dupond
Martin
Dupont
Martin
Dupond
Chirac

2

3

Gens
Nom
Dupond
Martin
Dupont
Chirac

SELECT DISTINCT Nom FROM Gens

Sélectionner des enregistrements (V)

<i>Gens</i>
<i>Nom</i>
Chirac
Dupond
Dupont
Martin

```
SELECT DISTINCT Nom
FROM Gens
ORDER BY Nom ASC
```

4

5

<i>Gens</i>
<i>Nom</i>
Chirac
Dupond

```
SELECT DISTINCT Nom
FROM Gens
ORDER BY Nom ASC
LIMIT 2
```

6

<i>Gens</i>
<i>Nom</i>
Dupond

```
SELECT DISTINCT Nom
FROM Gens
WHERE Nom <> 'Chirac'
ORDER BY Nom ASC
LIMIT 2
```

Optimisation

Après la suppression de grandes parties d'une table contenant des index, les index des tuples supprimés sont conservés, rallongeant d'autant les sélections. Pour supprimer ces index obsolètes et vider les « trous », il faut l'optimiser.

Syntaxe :

OPTIMIZE TABLE *Relation*

Exemple :

OPTIMIZE TABLE *Personnes*

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 concaténées (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		Bibliothèque		Résultat de la jointure	
Nom	Prénom	Nom	Nblivres	Nom	Nblivres
Martin	Jean	Martine	5	Martin	NULL
Tartan	Pion	Tartan	10	Tartan	10
Dupond	Jacques	Dupond	3	Dupond	3

3

Fonctions de MySQL

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 être 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 dont 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 produits 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 : $SIGN(ca) = -1 \Leftrightarrow ca < 0$

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 de dates et heures

Fonction	Description
NOW()	Retourne la date et heure du jour.
TO_DAYS(x)	Conversion de la date X en nombre de jours depuis le 1er janvier 1970.
DAYOFWEEK(x)	Retourne le jour de la semaine de la date x sous la forme d'un index qui commence à 1 (1=dimanche, 2=lundi...)
DAYOFMONTH(x)	Retourne le jour du mois (entre 1 et 31).
DAYOFYEAR(x)	Retourne le jour de l'année (entre 1 et 366).
SECOND(x), MINUTE(x), HOUR(x), MONTH(x), YEAR(x), WEEK(x)	Retournent respectivement les secondes, minutes, heures, mois, année et semaine de la date.

```
SELECT titre  
FROM article  
WHERE (TO_DAYS(NOW())) – TO_DAYS(parution)) < 30
```

Cet exemple affiche le titre des articles parus il y a moins de 30 jours.

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 modèles 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.

4

Interface avec PHP

Connexion (I)

Pour se connecter à une base depuis un script php, il faut spécifier un nom de serveur, un nom d'utilisateur, un mot de passe et un nom de base.

Aucune connexion n'est possible sans authentification auprès du serveur de base de données.

Les actions possibles de l'utilisateur sur la base à laquelle il se connecte dépendent des droits qui lui auront été fournis par l'administrateur de la base de données.

mysql_connect(\$server,\$user,\$password) : permet de se connecter au serveur **\$server** en tant qu'utilisateur **\$user** avec le mot de passe **\$password**, retourne l'identifiant de connexion si succès, FALSE sinon. Si ces arguments manquent, les valeurs par défaut du fichier de configuration *php.ini* seront utilisées.

mysql_select_db(\$base[\$id]) : permet de choisir la base **\$base**, peut prendre un identifiant **\$id** de connexion ; retourne TRUE en cas de succès, sinon FALSE. Les identifiants de connexion ne sont pas nécessaires si on ne se connecte qu'à un seul serveur à la fois, ils permettent seulement de lever toute ambiguïté en cas de connexions multiples (vers plusieurs serveurs dans le même script).

Connexion (II)

mysql_close([\$id]) : permet de fermer la connexion à un serveur de bases de données, l'argument optionnel **\$id** est l'identifiant de session retourné à l'ouverture de la connexion.

A noté que toutes les connexions aux serveurs de bases de données sont automatiquement fermées à la fin de l'exécution du script qui les aura ouvertes.

Dans le cas où le visiteur du site doit naviguer à travers différents script PHP qui se connectent tous au même serveur, il est préférable d'avoir recours aux « connexions persistantes ». Une connexion persistante est ouverte avec la fonction **mysql_pconnect()** qui est en tout point comparable à **mysql_connect()** à la seule différence que la connexion n'est pas fermée à la fin du script qui a ouvert la connexion. Ainsi, les scripts suivants peuvent continuer à lancer des requêtes à la base de données sans à avoir à rouvrir de connexion en direction du serveur.

Une connexion persistante ne peut pas être fermée avec la fonction **mysql_close()**. Au delà d'un certain temps d'inactivité, la ou les connexions persistantes ouvertes sont automatiquement fermées.

Connexion (III)

Exemple 1 :

```
if( $id = mysql_connect("localhost","foobar","0478") ) {
    if(mysql_select_db("gigabase") ) {
        echo "Succès de connexion.";
        /* code du script ... */
    } else {
        die("Echec de connexion à la base.");
    }
    mysql_close($id);
} else {
    die("Echec de connexion au serveur de base de données.");
}
```

Connexion (IV)

Exemple 2 :

```
@mysql_connect("localhost","foobar","0478") or die("Echec de connexion au serveur.");
```

```
@mysql_select_db("gigabase") or die("Echec de sélection de la base.");
```

Cet exemple est équivalent au précédent mais plus court à écrire. Le symbole **@** (arobase) permet d'éviter le renvoi de valeur par la fonction qu'il précède.

On pourra avantageusement intégrer ce code dans un fichier que l'on pourra joindre par **include()**. C'est aussi un moyen de sécuriser le mot de passe de connexion.

Une connexion persistante évite d'avoir à rouvrir une connexion dans chaque script. Les connexions sont automatiquement fermées au bout d'un certain temps en cas d'absence de toute activité...

Interrogation

Pour envoyer une requête à une base de donnée, il existe la fonction : **mysql_query(\$str)** qui prend pour paramètre une chaîne de caractères qui contient la requête écrite en SQL et retourne un identificateur de résultat ou FALSE si échec.

Exemple :

```
$result = mysql_query("SELECT téléphone FROM Personnes WHERE nom=\''$name\'");
```

Cet exemple recherche le téléphone d'une personne portant pour nom la valeur de la chaîne **\$name**. L'identificateur de résultat **\$result** permettra à d'autres fonctions d'extraire ligne par ligne les données retournées par le serveur. Chaque appel à cette fonction retournera un tuple du résultat. C'est pourquoi cette instruction pourra être utilisée au sein d'une boucle **while** qui s'arrêtera lorsque **mysql_query()** renverra FALSE.

Extraction des données (I) – tableau

mysql_fetch_row(\$result) : retourne une ligne de résultat (un tuple) sous la forme d'un tableau. Les éléments du tableau étant les valeurs des attributs de la ligne. Retourne FALSE s'il n'y a plus aucune ligne.

Exemple 1 :

```
$requet = "SELECT * FROM users";  
if($result = mysql_query($requet)) {  
    while($ligne = mysql_fetch_row($result)) {  
        $id = $ligne[0];  
        $name = $ligne[1];  
        $address = $ligne[2];  
        echo "$id - $name, $address <br />";  
    }  
} else {  
    echo "Erreur de requête de base de données.";  
}
```

Ici, on accède aux valeurs de la ligne par leur indice dans le tableau.

Extraction des données (II) – associatif

`mysql_fetch_array($result)` et `mysql_fetch_assoc($result)` : retournent un tableau associatif. Les clés étant les noms des attributs et leurs valeurs associées leurs valeurs respectives. Retourne FALSE s'il n'y a plus aucune ligne.

Exemple 2 :

```
$requet = "SELECT * FROM users";  
if($result = mysql_query($requet)) {  
    while($ligne = mysql_fetch_array($result)) {  
        $id = $ligne["id"];  
        $name = $ligne["name"];  
        $address = $ligne["address"];  
        echo "$id - $name, $address <br />";  
    }  
} else {  
    echo "Erreur de requête de base de données.";  
}
```

Ici, on accède aux valeurs de la ligne par l'attribut dans le tableau associatif.

Extraction des données (III) – objet

mysql_fetch_object(\$result) : retourne un objet. Les attributs de l'objet correspondent à ceux de la ligne de résultat. Et les valeurs des attributs de l'objet correspondent à ceux de la ligne de résultat. Retourne FALSE s'il n'y a plus aucune ligne.

Exemple 3 :

```
$requet = "SELECT * FROM users";  
if($result = mysql_query($requet)) {  
    while($ligne = mysql_fetch_object($result)) {  
        $id = $ligne->id;  
        $name = $ligne->name;  
        $address = $ligne->address;  
        echo "$id - $name, $address <br />";  
    }  
} else {  
    echo "Erreur de requête de base de données.";  
}
```

Ici, on accède aux valeurs par leur attribut dans l'objet.

Statistiques sur une requête

mysql_affected_rows([*\$id*]) : retourne le nombre de lignes modifiées par la dernière requête INSERT, UPDATE ou DELETE effectuée sur le serveur identifiée par *\$id* (les DELETE sans clause WHERE retourneront 0 lignes, car la table sera recrée au lieu de supprimer les lignes une à une).

```
$requet = "DELETE FROM users WHERE name LIKE \'Martin%\'" ;  
$result = mysql_query($requet) or die("Erreur de base de données.");  
$num = mysql_affected_rows();
```

mysql_num_rows(*\$result*) : retourne le nombre de lignes retournées par la dernière requête SELECT dont on connaît l'identifiant de résultat *\$result*.

```
$requet = "SELECT name FROM users WHERE birth > \'1980-05-10\'" ;  
$result = mysql_query($requet) or die("Erreur de base de données.");  
$num = mysql_num_rows();
```

mysql_num_fields(*\$result*) : retourne le nombre d'attributs des tuples du résultat d'une requête.

```
$requet = "SELECT * FROM users" ;  
$result = mysql_query($requet) or die("Erreur de base de données.");  
$num = mysql_num_fields();
```

Informations sur les attributs (I)

Les fonctions suivantes s'appliquent au **\$field** ème attribut retourné par la dernière requête identifiée par **\$result** :

mysql_field_name(\$result, \$field) : retourne le nom

mysql_field_len(\$result, \$field) : retourne la taille

mysql_field_type(\$result, \$field) : retourne le type

mysql_field_flags(\$result, \$field) : retourne les drapeaux

mysql_field_table(\$result, \$field) : retourne le nom de la table

mysql_fetch_field(\$result [, \$field]) : retourne un objet contenant des informations sur l'attribut **\$field**. Ses attributs sont **name** (nom), **table** (nom de la table), **max_length** (taille), **type** (type) et les booléens suivants : **not_null**, **primary_key**, **unique_key**, **multiple_key**, **numeric**, **blob**, **unsigned**, **zerofill**.

mysql_field_seek(\$result, \$field) : prépositionne l'index **\$field** afin de ne pas le passer en paramètre à **mysql_fetch_field()**.

L'index commence à zéro.

Elles ne peuvent être utilisées qu'après un appel à la fonction **mysql_query()** retournant le pointeur de résultat **\$result**.

Informations sur les attributs (II)

mysql_list_dbs([\$id]) : retourne un pointeur de résultat simulant la requête suivante : "SHOW DATABASES". Liste des bases de données.

mysql_list_tables(\$base [, \$id]) : retourne un pointeur de résultat simulant la requête suivante : "SHOW TABLES FROM \$base". Liste des relations de la base de données **\$base**.

mysql_list_fields (\$base, \$table [, \$id]) : retourne un pointeur de résultat simulant la requête suivante : "SHOW COLUMNS FROM \$table FROM \$base". Ce pointeur peut être utilisé par les fonctions **mysql_field_*** afin d'avoir des informations sur une table **\$table** de la base **\$base**. L'identifiant de connexion **\$id** est optionnel.

mysql_fetch_lengths(\$result) : retourne un tableau contenant la taille de chaque attribut de la ligne du dernier tuple résultat de la requête **\$result**, ou FALSE sinon. Ne peut être utilisée qu'après l'une des fonctions d'extraction.

Informations sur les attributs (III)

```
if($result = mysql_query("SELECT * FROM forum" ) {  
  for($i=1; $i<= mysql_num_fields($result); $i++ ) {  
    echo mysql_field_name($result, $i-1), ", ",  
    mysql_field_len($result, $i-1), ", ",  
    mysql_field_type($result, $i-1), ", ",  
    mysql_field_flags($result, $i-1), ",",  
    mysql_field_table($result, $i-1), "<br />";  
  }  
} else die("Erreur de base de données.");
```

Définition de la table :

```
CREATE TABLE forum (  
  id bigint(20) unsigned auto_increment,  
  title tinytext NOT NULL,  
  mesg text NOT NULL,  
  hits mediumint(8) unsigned NOT NULL,  
  author_idx bigint(20) unsigned NOT NULL,  
  date datetime NOT NULL,  
  PRIMARY KEY(id)  
)
```

Résultats :

nom	taille	type	drapeaux	table
id	20	int	not_null primary_key unsigned auto_increment	forum
title	255	blob	not_null blob	forum
mesg	65535	blob	not_null blob	forum
hits	8	int	not_null unsigned	forum
author_idx	20	int	not_null unsigned	forum
date	19	datetime	not_null	forum

Informations sur les attributs (IV)

```
if($result = mysql_query("SELECT * FROM forum" ) ) {  
    $infos = mysql_fetch_field($result, 0);  
    print_r($infos);  
}
```

Cet exemple affiche les informations sur le premier attribut des résultats de la requête. On voit qu'il s'appelle 'id', qu'il appartient à la table 'forum', que sa taille maximum est de 2 digits, qu'il porte les contraintes suivantes : NOT NULL et PRIMARY KEY, qu'il est de type numérique non signé : INT, UNSIGNED.

```
Résultat :  
stdClass Object  
(  
    [name] => id  
    [table] => forum  
    [def] =>  
    [max_length] => 2  
    [not_null] => 1  
    [primary_key] => 1  
    [multiple_key] => 0  
    [unique_key] => 0  
    [numeric] => 1  
    [blob] => 0  
    [type] => int  
    [unsigned] => 1  
    [zerofill] => 0  
)
```

Fonctions sur le serveur

mysql_create_db(\$base [, \$id]) : création de la base **\$base**.

mysql_db_name(\$result, \$row [, \$field]) : Lit les noms des bases de données. **\$result** est l'identifiant de résultat issu de **mysql_list_dbs()**. **\$row** est l'index dans le résultat. Retourne FALSE si échec.

mysql_db_query(\$base, \$query [, \$id]) : exécution de la requête **\$query** sur la base **\$base**. Retourne un identifiant de résultat si succès ou FALSE si échec.

mysql_query(\$query [, \$id]) : exécution de la requête sur la base ouverte. Retourne un identifiant de résultat si succès ou FALSE si échec.

mysql_drop_db(\$base [, \$id]) : supprime la base de données **\$base**. Retourne TRUE si succès ou FALSE si échec.

mysql_select_db(\$base [, \$id]) : sélectionne la base de données **\$base** sur le serveur sur lequel on est connecté et dont **\$id** est l'identifiant de connexion. Retourne TRUE si succès ou FALSE si échec.

```
$result = mysql_list_dbs();
```

```
$num = mysql_num_rows($result);
```

```
for ($i=0; $i<$num; $i++)
```

```
    echo mysql_db_name($result, $i)."  
</pre>
```

Cet exemple affiche la liste des bases de données du serveur.

Gestion des erreurs

Il est recommandé de tester systématiquement les valeurs retournées par les fonction de traitement sur une base de données afin d'éviter la pollution de la page web par des *Warning*.

mysql_errno([*\$id*]) : retourne le numéro d'erreur de la dernière opération MySQL effectuée sur la connexion courante ou celle d'identifiant ***\$id***.

mysql_error([*\$id*]) : retourne le message d'erreur de la dernière opération MySQL effectuée sur la connexion courante ou celle d'identifiant ***\$id***.

Exemple :

```
$requet = "DELETE FROM users WHERE name LIKE \'Martin%\''";  
if($result = mysql_query($requet)) {  
    ...  
} else {  
    echo "Erreur de base de données n°".mysql_errno().": ".mysql_error();  
}
```

Fonctions additionnelles

Quelques fonctions supplémentaires très utiles :

mysql_free_result(\$result) : efface de la mémoire du serveur les lignes de résultat de la requête identifiées par **\$requet**. Très utile pour améliorer les performances du serveur. A n'utiliser que si votre script utilise vraiment beaucoup de mémoire.

mysql_insert_id([\$id]) : retourne l'identifiant d'un attribut clé primaire AUTO_INCREMENT de la dernière insertion.

mysql_data_seek(\$result, \$row) : Permet de prépositionner le pointeur interne de résultat **\$result** à la ligne **\$row**. Le prochain appel à une fonction d'extraction de tuple du résultat ira directement à cette ligne. Retourne TRUE si succès et FALSE sinon.

Penser à bien tester la valeur de retour des fonctions (**mysql_query** et les autres) afin de détecter toute erreur et d'éviter de polluer votre page avec des *Warnings*.

Directives de configuration du php.ini

Ces informations sont utilisées si elles sont omises lors d'une connexion :

mysql.default_host chaîne de caractères

Adresse par défaut du serveur de bases de données.

mysql.default_user chaîne de caractères

Utilisateur par défaut.

mysql.default_password chaîne de caractères

Mot de passe par défaut.

Connexions persistantes :

mysql.allow_persistent booléen

Active ou désactive les connexions persistantes.

mysql.max_persistent entier

Nombre maximum de connexions persistantes par processus.

Connexions :

mysql.max_links entier

Nombre de connexion simultanées maximum, par processus, incluant les connexions persistantes

5

Administration avec l'outil web phpMyAdmin

Présentation

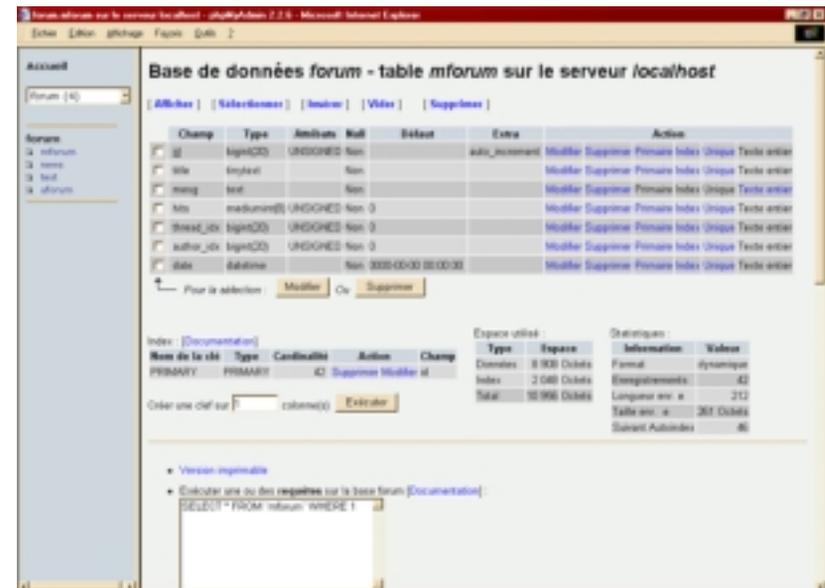
L'outil phpMyAdmin est développé en PHP et offre une interface intuitive pour l'administration des bases de données du serveur.

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

Cet outil permet de :

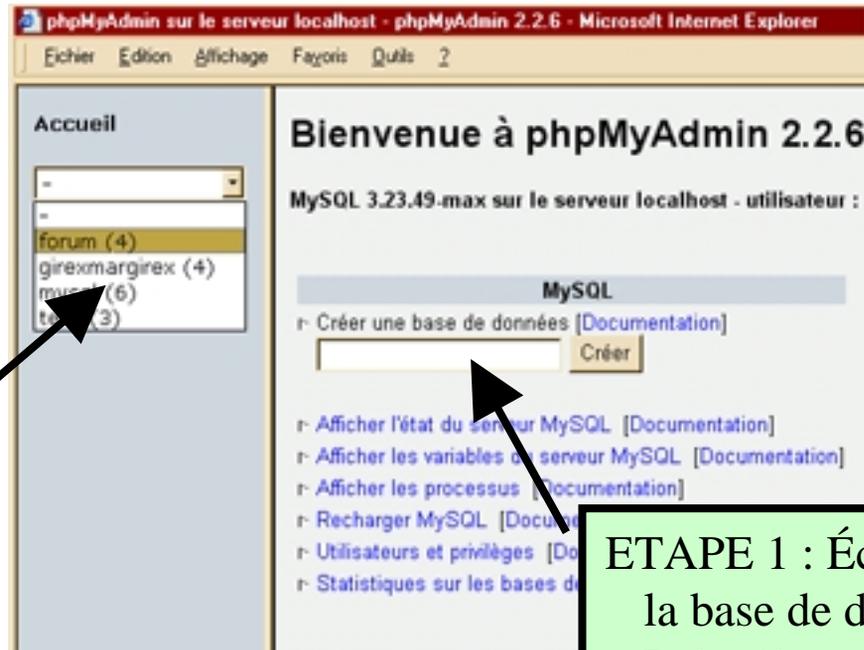
- créer de nouvelles bases
- créer/modifier/supprimer des tables
- afficher/ajouter/modifier/supprimer des tipes dans des tables
- effectuer des sauvegarde de la structure et/ou des données
- effectuer n'importe quelle requête
- gérer les privilèges des utilisateurs

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



Création/sélection d'une base de données

Avant de manipuler des données, il faut créer une ou des bases de données.



ETAPE 2 : sélectionnez le nom de la base à manipuler (le nombre de tables de la base apparaît entre parenthèses)

ETAPE 1 : Écrivez le nom de la base de donnée à créer. Puis cliquez sur « Créer »

Et aussi.. choix de la langue de l'interface de phpMyAdmin



Gestion de la base de données (I)

forum sur le serveur localhost - phpMyAdmin 2.2.6 - Microsoft Internet Explorer

Accueil

forum (4)

forum

- mforum
- news
- test
- uforum

Table	Action	Enregistrements	Type	Taille
<input type="checkbox"/> mforum	Afficher Sélectionner Insérer Propriétés Supprimer Vider	42	MyISAM	10,7 Ko
<input type="checkbox"/> news	Afficher Sélectionner Insérer Propriétés Supprimer Vider	4	MyISAM	2,6 Ko
<input type="checkbox"/> test	Afficher Sélectionner Insérer Propriétés Supprimer Vider	7	MyISAM	2,1 Ko
<input type="checkbox"/> uforum	Afficher Sélectionner Insérer Propriétés Supprimer Vider	5	MyISAM	2,7 Ko
4 table(s)	Somme	58	--	18,1 Ko

Tout cocher / Tout décocher Pour la sélection :

- Version imprimable
- Exécuter une ou des requêtes sur la base forum [Documentation] :

Réafficher la requête après exécution

Où Emplacement du fichier texte :

Parcourir...

Exécuter

Choix d'une table à gérer en particulier

Actions sur les tables : afficher leur contenu intégral, faire une sélection sur critères, ajouter des données, gérer ses propriétés intrinsèques, supprimer, vider.

Écrire une requête MySQL à exécuter

Exécuter une requête MySQL contenue dans un fichier

Gestion de la base de données (II)

- [Requête par un exemple](#)
- Afficher le schéma de la base

mforum
news
test
uforum

- Structure seule
 Structure et données
 Données seulement

[Tout sélectionner](#) / [Tout désélectionner](#)

- Ajouter des énoncés "drop table"
 Insertions complètes
 Insertions étendues
 Protéger les noms des tables et des champs par des ""
 Transmettre ("bzippé")

Exécuter

- Créer une nouvelle table sur la base forum :

Nom :

Champs :

Exécuter

- [Supprimer la base forum](#) [[Documentation](#)]

Accès à un formulaire détaillé pour effectuer une requête

Option permettant de transmettre le schéma sous la forme d'un fichier

Affichage du schéma (structure et/ou données) des tables sélectionnées de la base

Accès à un formulaire détaillé d'ajout d'une table dans la base

Supprimer la base

Affichage d'une table

Base de données *forum* - table *mforum* sur le serveur *localhost*

Affichage des enregistrements 3 - 5 (42 total)

requête SQL : [Modifier]
SELECT * FROM 'mforum' LIMIT 3, 3

<< < Afficher : 3 lignes à partir de 5 en mode horizontal et répéter les en-têtes à chaque groupe de 100 > >>

	id	title	mesg	hits	thread_idx	author_idx	date
Modifier Effacer	4	Concours de logo AML 4	Vous êtes tous invités à participer au concours de...	2	0	1	2002-09-18 11:13:40
Modifier Effacer	5	Concours de logo AML 5	Vous êtes tous invités à participer au concours de...	2	0	1	2002-09-18 11:13:40
Modifier Effacer	6	Concours de logo AML 6	Vous êtes tous invités à participer au concours de...	0	0	1	2002-09-18 11:13:40

<< < Afficher : 3 lignes à partir de 5 en mode horizontal et répéter les en-têtes à chaque groupe de 100 > >>

Insérer un nouvel enregistrement

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

Permet de naviguer dans les pages de résultats

Afficher par page de X lignes

Supprimer un enregistrement

Accès au formulaire de modification d'un enregistrement

Insertion d'un nouvel enregistrement

Insertion/modification d'un enregistrement

Champ	Type	Fonction	Null	Valeur
id	bigint(20) unsigned			
title	tinytext			
mesg	text	SOUNDEX LCASE UCASE NOW PASSWORD MD5 ENCRYPT RAND LAST_INSERT_ID COUNT AVG		hello CyberZoïde
hits	mediumint(8) unsigned		0	
thread_idx	bigint(20) unsigned		0	
author_idx	bigint(20) unsigned		0	
date	datetime			2003-01-11 18:10:02

Insérer en tant que nouvel enregistrement -- et --

Retourner à la page précédente
Ou
 Insérer un nouvel enregistrement

Exécuter

Les champs et leurs types sont définis lors de la création de la table : tous les champs sont pas forcément obligatoires...

Les formulaires d'insertion et de modification sont similaires.

Gestion d'une table (I)

Propriétés des attributs de la table

Quelques actions rapides : affichage, sélection, insertion d'un enregistrement, vidage, suppression

The screenshot shows a database management interface with the following elements:

- Buttons at the top: [Afficher] | [Sélectionner] | [Insérer] | [Vider] | [Supprimer]
- Table of attributes:

Champ	Type	Attributs	Null	Défaut	Extra	Action					
<input type="checkbox"/> id	int(20)	UNSIGNED	Non		auto_increment	Modifier	Supprimer	Primaire	Index	Unique	Texte entier
<input type="checkbox"/> title	varchar(40)		Non			Modifier	Supprimer	Primaire	Index	Unique	Texte entier
<input type="checkbox"/> text	text		Non			Modifier	Supprimer	Primaire	Index	Unique	Texte entier
<input type="checkbox"/> date	date		Non	0000-00-00		Modifier	Supprimer	Primaire	Index	Unique	Texte entier

Below the table, there are buttons: "Pour la sélection : [Modifier] Ou [Supprimer]"

Index section: Index : [Documentation]

Nom de la clé	Type	Cardinalité	Action	Champ
PRIMARY	PRIMARY	4	Supprimer Modifier	

Buttons: "Créer une clef sur [1] colonne(s) [Exécuter]"

Space used section: Espace utilisé :

Type	Espace
Données	612 Octets
Index	2 048 Octets
Total	2 660 Octets

Statistics section: Statistiques :

Information	Valeur
Format	dynamique
Enregistrements	4
Longueur enr. e	153
Taille enr. e	665 Octets
Suivant Autoindex	

Quelques actions sur les attributs : modifier, supprimer ; plus les contraintes : clé primaire et unique ; et y mettre un index

Créer une nouvelle clé sur X attributs

Modifier ou supprimer plusieurs attributs en même temps

Quelques statistiques et infos

Gestion d'une table (II)

Affichage de la version imprimable de la page

The screenshot shows a web-based database management interface. At the top left, there is a link for 'Version imprimable'. Below it, a section titled 'Exécuter une ou des requêtes sur la base forum [Documentation]:' contains a text area with the SQL query 'SELECT * FROM `news` WHERE 1'. There is a checkbox for 'Réafficher la requête après exécution' which is checked. Below the checkbox is a field for 'Ou Emplacement du fichier texte:' with a 'Parcourir...' button. An 'Exécuter' button is located below the text area. Further down, there are three more sections: 'Ajouter un champ:' with a text input containing '1', a dropdown menu set to 'En fin de Table', and an 'Exécuter' button; 'Ordonner la table par:' with a dropdown menu set to 'id' and an 'Exécuter' button; and 'Insérer des données provenant d'un fichier texte dans la table'.

Écrire une requête à exécuter ou spécifier un fichier contenant une ou plusieurs

Ajouter X champs dans la table à une position particulière

Réordonner les données de table en fonction d'un attribut

Accès au formulaire d'insertion de données dans la table à partir d'un fichier

Gestion d'une table (III)

Permet d'afficher le schéma (structure et/ou données) de la table.

Le schéma d'une table est l'ensemble de la structure et des données d'une table.

La structure est composée de la définition des propriétés des attributs et des clés.

The screenshot shows a web interface titled "Afficher le schéma de la table". It features several radio buttons for selecting the output format: "Structure seule" (selected), "Structure et données", "Données seulement", "Données CSV pour Ms Excel", and "Données CSV :". Below these are input fields for "Champs terminés par", "Champs entourés par", "Caractère spécial", and "Lignes terminées par". To the right, there are checkboxes for "Ajouter des énoncés 'drop table'", "Insertions complètes", "Insertions étendues", "Protéger les noms des tables et des champs par des ''", and "Transmettre ([] 'bzippé')". At the bottom, there are input fields for "Premier enregistrement" (set to 0) and "nb. d'enregistrements" (set to 4), along with an "Exécuter" button.

Le résultat de sortie « structure et/ou données » est constitué des requêtes MySQL de création de la table et d'insertion des enregistrements.

Le format CSV est un fichier texte dont chaque ligne représente un enregistrement.

Le résultat peut être transmis sous la forme d'un fichier (qui lui-même peut être compressé).

Gestion d'une table (IV)

The screenshot shows a database management interface with several sections and callouts:

- Changer le nom de la table pour :** A text input field containing 'news' and an 'Exécuter' button. Callout: "Changer le nom de la table".
- Déplacer la table vers (base.table) :** A dropdown menu and a text input field containing 'news', with an 'Exécuter' button. Callout: "Déplacer la table vers une autre base avec changement de nom possible dans la foulée".
- Copier la table vers (base.table) :** A dropdown menu containing 'forum', a text input field, and an 'Exécuter' button. Callout: "La copier (avec ou sans les données) vers une autre base avec changement de nom possible".
- Maintenance de la table :** A section with links for "Vérifier la table [Documentation]", "Analyser la table [Documentation]", "Réparer la table [Documentation]", and "Optimiser la table [Documentation]". Callout: "Quelques opérations de maintenance : vérification, analyse, réparation, optimisation".
- Commentaires sur la table :** A text input field and an 'Exécuter' button. Callout: "Lui associer un commentaire".
- Table en format :** A dropdown menu containing 'MyISAM' and an 'Exécuter' button. Callout: "Changer le format de la table".
- Recharger la table ("FLUSH")** and **Supprimer la table**: Two blue links at the bottom. Callouts: "Recharger la table en mémoire du serveur" and "La supprimer".

Insertion des données dans une table

Emplacement du fichier texte	<input type="text"/> <input type="button" value="Parcourir..."/>	
Remplacer les données de la table avec le fichier	<input type="checkbox"/> Remplacer	Le contenu du fichier remplacera le contenu de la table pour les enregistrements ayant une valeur de clé primaire ou unique identique.
Champs terminés par	<input type="text"/>	Le caractère qui sépare chacun des champs.
Champs entourés par	<input type="text"/> <input type="checkbox"/> OPTIONNEL	Souvent des guillemets. OPTIONNEL signifie que seuls les champs de type char et varchar sont entourés par ce caractère.
Caractère spécial	<input type="text"/>	Optionnel. Indique le caractère qui permet d'enlever l'effet des caractères spéciaux.
Lignes terminées par	<input type="text"/>	Retour de chariot : \r Saut de ligne : \n
Nom des colonnes	<input type="text"/>	Si vous désirez ne charger que certaines colonnes, indiquez leurs noms, séparés par des virgules.
[Documentation]		
<input type="button" value="Exécuter"/> <input type="button" value="Réinitialiser les valeurs"/>		

On accède à cet écran via le lien « *Insérer des données provenant d'un fichier texte dans la table* » de la page de gestion de la table.

Permet d'insérer des enregistrements dans une table à partir d'un fichier de données au format CSV.

On peut changer les valeurs par défaut des séparateurs standards du CSV.

Création d'une clé

----- Créer un nouvelle clef -----

Nom de la clef : ("PRIMARY" doit et ne peut être que le nom d'une clef primaire !)

Type de clef : [Documentation]

Champ	Taille
<input type="text" value="-- Ignorer --"/>	<input type="text"/>
<input type="text" value="-- Ignorer --"/>	<input type="text"/>
<input type="text" value="-- Ignorer --"/>	<input type="text"/>

Ajouter à la clef colonne(s)

On accède à cet écran var le lien « *Créer une clef sur X colonne(s)* » de la page de gestion de la table.

Permet de créer une clé sur une ou plusieurs colonnes.

Il faut nommer la clé, en spécifier le type, et les attributs sur lesquels elle s'applique.

On peut rajouter une autre colonne à cette clé avant de valider l'ajout de la clé.

Liens

La référence PHP (anglais & français) :

<http://www.php.net>

La référence MySQL (anglais) :

<http://www.mysql.com>

Le manuel MySQL traduit en français ici :

<http://dev.nexen.net/docs/>

Des cours et articles intéressants :

<http://www.developpez.com>

dont la FAQ PHP & MySQL : <http://php.developpez.com/faq/>

L'outil phpMyAdmin :

<http://phpmyadmin.sourceforge.net>

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/>