FACULTE POLYDISCIPLINAIRE TAROUDANT

Cours: MySQL

Syntaxes et fonctions de base de MySql avec php

Licence Professionnelle: Génie Informatique
Semestre 4
2021/2022

Pr. J. ANTARI

1

Sommaire

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

2

Introduction

MySQL

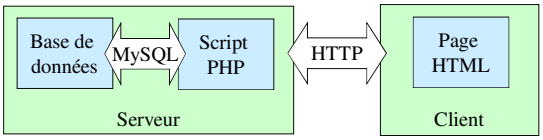
- Gestionnaire de bases de données
- PHP peut interagir avec MySQL: La création et l'interrogation d'une base de données
- Envoi de requêtes SQL
- Récupération du résultat
- Ajouter de données …

3

Introduction

MySQL est basé sur une architecture client/serveur.

C'est-à-dire que les clients doivent s'adresser au serveur qui gère, contrôle et arbitre les accès aux données.



4

Théorie des bases de données

5

Concepts du modèle relationnel

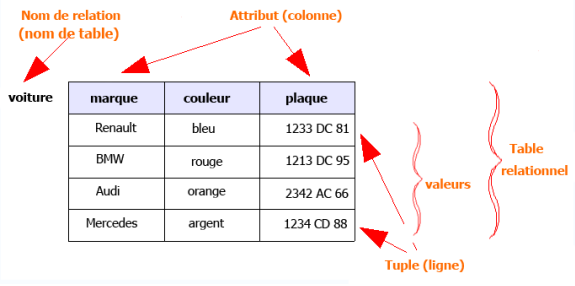
Domaine : Ensemble des valeurs d'un attribut.

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

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.

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

6



7

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**.

Exemple:

Relation: **Personnes**

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

nom	prénom	adresse	téléphone
Dupond	Marc	8 rue	00212666

8

Algèbre relationnelle

L'algèbre relationnelle regroupe toutes les opérations possibles sur les relations.
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'es 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…**).

9

Projection

Personnes

nom	prénom	adresse	téléphone
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***

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

nom	prénom
Martin	Pierre
Dupond	Jean
Dupond	Marc

10

Jointure

Personnes

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

Bibliothèque

nom	Dernierlivre
Dupond	Robinson
Jospin	Faust
Martin	Misère

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

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

prénom	Dernierlivre
Jean	Robinson
Pierre	Misère

11

Sélection

Personnes

nom	prénom	adresse	téléphone
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"**

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

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

12

Syntaxe de MySQL

13

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 VACHAR(40) !

14

Types des attributs (II) – entiers

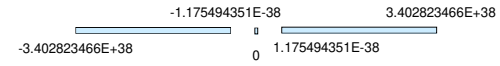
nom	borne inférieure	borne supérieure
TINYINT	-128	127
TINYINT UNSIGNED	0	255
SMALLINT	-32768	32767
SMALLINT UNSIGNED	0	65535
MEDIUMINT	-8388608	8388607
MEDIUMINT UNSIGNED	0	16777215
INT*	-2147483648	2147483647
INT* UNSIGNED	0	4294967295
BIGINT	-9223372036854775808	9223372036854775807
BIGINT UNSIGNED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

(*) : **INTEGER** est un synonyme de **INT**.
UNSIGNED permet d'avoir un type non signé.
ZEROFILL : remplissage des zéros non significatifs.

```
CREATE TABLE yourtable (x INT(8) ZEROFILL NOT NULL, y INT(8) NOT NULL); INSERT INTO yourtable (x,y)
VALUES (1, 1), (12, 12), (123, 123), (123456789, 123456789); SELECT x, y FROM yourtable;
      x          y
00000001          1
00000012         12
00000123        123
123456789  123456789
```

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**.

16

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**.

17

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.

BLOB : Objet binaire de grande taille qui peut contenir une quantité variable de données.
Les quatre types BLOB sont TINYBLOB, BLOB, MEDIUMBLOB, et LONGBLOB. Ceux-ci ne diffèrent que par la longueur maximale des données qu'ils peuvent stocker.

18

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.

19

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.

20

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.

21

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*

22

Commandes MySQL

mysql **–u root –p**

mysql> create database GI;

1. *Lister les bases de données existantes*

mysql>show databases;

2. *Lister les tables de la base de données, exemple BD GI*

mysql>use GI;

mysql>show tables;

3. *Afficher les lignes de la table, exemple tabletest*

mysql>select * from *tabletes*;

4. CREATE [TEMPORARY] TABLE nom_relation [IF NOT EXISTS] (
nom_attribut TYPE_ATTRIBUT [OPTIONS]
…);

23

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 MEDIUMINT UNSIGNED PRIMARY KEY,
  titre VARCHAR(80),
  texte TEXT,
  parution DATE,
  auteur VARCHAR(80),
  rubrique ENUM('économie','sports','international','politique','culture') );
```

24

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 indexes de la table.

25

Créer une relation (II)

Exemple: carnet d'adresse
CREATE TABLE *Personne* (
nom VARCHAR(40),
prenom VARCHAR(40),
adresse TINYTEXT,
telephone DECIMAL
);

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), *prenom* (idem), *adresse* (texte de longueur variable mais inférieure à 255 caractères) et *telephone* (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.

26

Clé primaire (I)

Pour des raisons pratiques, nous souhaitons pouvoir associer à chacun des enregistrements de la relation un identifiant numérique unique qui puise être passé en paramètre à nos scripts PHP.
Pour cela on rajoute un nouvel attribut de type entier. Pour nous facilité 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**.

27

Clé primaire (II)

Notre exemple devient :

CREATE TABLE *Personne* (
id **SMALLINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,**
nom **VARCHAR(40),**
prenom **VARCHAR(40),**
adresse **TINYTEXT,**
telephone **DECIMAL**
);

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	prenom	adresse	telephone
1	Dupond	Marc	8 rue de l'octet	0123456789

28

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.

29

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.

30

Attribut sans doublon (I)

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

Syntaxe :

UNIQUE [*nomdelacontrainte***](***liste des attributs***)**

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

UNIQUE(*nom*)

CREATE TABLE *Personne*(
***nom* VARCHAR(40),**
***prenom* VARCHAR(40),**
unique(nom));

CREATE TABLE Personne(
nom **VARCHAR(40),**
prenom **VARCHAR(40),**
unique(nom),
unique(prenom)) ;

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

UNIQUE(*nom*)
UNIQUE(*prenom*)

nom	*prenom*
Dupond	**Marc**
Dupont	Pierre
Martin	**Marc**

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

31

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 interdit tout doublon du couple (*nom*, *prenom*) :

UNIQUE(*nom*,*prenom*)

nom	*prenom*
Dupond	Marc
Dupont	Pierre
Martin	**Marc**
Martin	Pierre
Martin	**Marc**

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

32

Index (I)

Lors de la recherche d'informations dans une relation, MySQL parcours la table correspondante dans n'importe quel ordre. Dans le cas d'un grand nombre de lignes, cette recherche est 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 !
On choisira de créer des indexes 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.

33

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'*)

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*

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

35

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)

ALTER TABLE *Personnes* ADD *fax* DECIMAL(10,0) AFTER 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 *prenom*

36

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 ***Personnes*** (
id **SMALLINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,**
nom **VARCHAR(40),**
prenom **VARCHAR(40),**
adresse **TINYTEXT,**
telephone **DECIMAL(10,0),**
UNIQUE(*nom***,prenom));**

ALTER TABLE ***Personnes*** **DROP** *prenom*

nom	*prénom*
Dupond	Marc
Martin	Marc
Martin	Pierre

nom
Dupond
Martin
Martin

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

37

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*** **DROP PRIMARY KEY**

38

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.

39

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.

40

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 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')

42

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

43

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, *'céruléen'*)

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, *'céruléen'*)

44

Modifier un enregistrement

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 *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.

Exemple pour modifier plusieurs attributs :
UPDATE Personnes SET *téléphone*=**'0156281469',** *fax*=**'0156281812' WHERE** *id* **= 102**

Exemple :
UPDATE *Enfants* SET *age*=*age*+**1**
Il est donc possible de modifier la valeur d'un attribut relativement à sa valeur déjà existante.

45

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*

46

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%**'**

47

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

48

Sélectionner des enregistrements (III)

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**
DESC:
SELECT DISTINCT *attribut* **FROM *relation* ORDER BY** *attribut* **DESC**
Pour se limiter aux **num** premiers résultats :
LIMIT num
SELECT DISTINCT *attribut* **FROM *relation* WHERE condition ORDER BY** *attribut* **ASC LIMIT num**

49

Sélectionner des enregistrements (IV)

Relation de départ **Gens***:*
SELECT * FROM *Gens* **(Pour sélectionner tous les enregistrements d'une relation)**

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

SELECT *Nom* **FROM *Gens***

2

Gens
Nom
Dupond
Martin
Dupont
Martin
Dupond

3

Gens
Nom
Dupond
Martin
Dupont

SELECT DISTINCT *Nom* **FROM *Gens***
Permet d'ignorer les doublons de ligne

50

Sélectionner des enregistrements (V)

Gens
Nom
Dupond
Dupont
Martin

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

4

5

Gens
Nom
Dupond
Dupont

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

6

Gens
Nom
Dupond

SELECT DISTINCT *Nom*
FROM *Gens*
WHERE *Nom* **<> 'Dupont'**
ORDER BY *Nom* **ASC**
LIMIT 2

51

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***

52

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*.

53

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

54

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

55

Fonctions de MySQL: 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.

56

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%'
```

57

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

58

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'.

59

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.

60

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
```

61

Interface avec PHP

Connexion (I)

Pour se connecter à une base de données 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 BDD.
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).

62

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.

63

Connexion (III)

```
<?php
if($id = mysql_connect("localhost","root","") ) {
   if(mysql_select_db("fpt") ) {
      echo "Succès de connexion.";
      echo '<br/>';
      echo "Salut db fpt.";
      /* code du script … */
   } else {
      die("Echec de connexion à la base.");
   }
   mysql_close($id);
} else {   die("Echec de connexion au serveur de base de données.");
}
?>
```

La fonction die() est une fonction intégrée en PHP qui est utilisé pour afficher le message et quitter le script PHP actuel. C'est équivalent à la fonction exit() en PHP.

64

SHOW TABLES

```
<?php
$dbname = 'fpt';
if (!mysql_connect('localhost', 'root', '')) {
   echo 'Could not connect to mysql';
   exit;
}
$sql = "SHOW TABLES FROM $dbname";
$result = mysql_query($sql);

if (!$result) {
   echo "DB Error, could not list tables\n";
   echo 'MySQL Error: ' . mysql_error();
   exit;
}
while ($row = mysql_fetch_row($result)) {
   echo "Table: {$row[0]}\n";
}
// list databases
$result = mysql_list_dbs();
$num = mysql_num_rows($result);
for ($i=0;  $i<$num; $i++)
   echo mysql_db_name($result, $i)."<br />";
?>
```

CREATE DATABASE

```
<?php
$link = mysql_connect('localhost', 'root', '');
if (!$link) {
   die('Could not connect: ' . mysql_error());
}
$sql = 'CREATE DATABASE fptGI';
if (mysql_query($sql, $link)) {
   echo "Database created successfully\n";
} else {
   echo 'Error creating database: ' .
mysql_error() . "\n";
}
?>
```

65

CREATE DATABASE: atelierdb.php

```
<?php
$link = mysqli_connect("localhost","root","");
if (!$link) {
   echo "Erreur de connexion ";
}
$sql = "CREATE DATABASE GII";
if (mysqli_query($link,$sql)) {
   echo "Database created successfully\n";
} else {
  echo "Erreur de requete";
}
?>
```

66

SHOW TABLE: atelier.php

```
<html><head><link rel="stylesheet" type="text/css" href="style1.css"></head>
<body>
<?php
        $cnx=mysqli_connect("localhost","root","");
        mysqli_select_db($cnx,"GI");
        $requete="SELECT*FROM test";
        $resultat=mysqli_query($cnx,$requete);
        if (!$resultat) {
                # code...
                echo "Erreur de requete ";
        }else
        {
                while ($ligne=mysqli_fetch_array($resultat)) {
                        # code..commentaire .
                        echo $ligne['nom'].' '.$ligne['prenom'];
echo "<br><br>" ;
                }
        }
?>
</body>
</html>
```

67

Connexion (V)

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…

68

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.

69

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.

70

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.

71

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.

72

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).

```
$request = ''DELETE FROM users WHERE name LIKE \"Martin%\"";
$result = mysql_query($request) 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**.

```
$request = ''SELECT name FROM users WHERE birth > \"1980-05-10\"";
$result = mysql_query($request) 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.

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

73

Informations sur les attributs

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, $field)** : retourne le type
- **mysql_field_table($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 utilisée qu'après un appel à la fonction **mysql_query()** retournant le pointeur de résultat **$result**.

74

mysql_list_fields

```
<?php
$link = mysql_connect('localhost', 'root', '');
$fields = mysql_list_fields("fpt", "client",
$link);
$columns = mysql_num_fields($fields);
for ($i = 0; $i < $columns; $i++) {
 echo mysql_field_name($fields, $i) . "\n";
}       ?>
```

75

```php
<?php
/* Supposons que la table utilisée contienne trois champs :
*   user_id
*   username
*   password.
 */
$link = @mysql_connect('localhost', 'mysql_user', 'mysql_password');
if (!$link) {
   die('Impossible de se connecter au serveur MySQL : ' . mysql_error());
}
$dbname = 'mydb';
$db_selected = mysql_select_db($dbname, $link);
if (!$db_selected) {
   die("Impossible de se connecter à la base $dbname: " . mysql_error());
}
$res = mysql_query('select * from users', $link);
echo mysql_field_name($res, 0) . "\n";
echo mysql_field_name($res, 2);
?>
```

Résultat d'affichage:
user_id
password

76

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 FASE 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 FASE si échec.

```php
$result = mysql_list_dbs();
$num = mysql_num_rows($result);
for ($i=0;  $i<$num; $i++)
   echo mysql_db_name($result, $i)."<br />";
```

Exemple affiche la liste des bases de données.

77

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 :

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

78

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.

79

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

80

Administration avec l'outil web phpMyAdmin

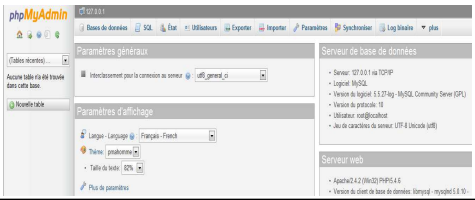
81

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

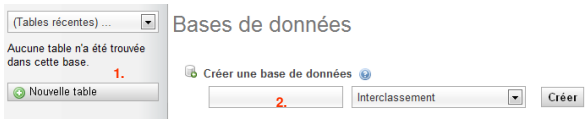
Cet outil permet de :

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



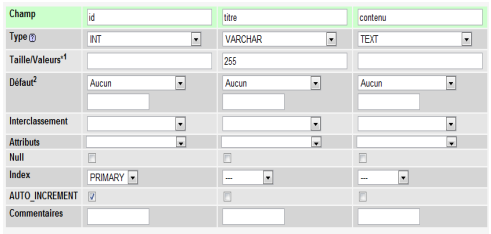
82

Créer une base de donnée



1. Liste des BDD
2. Créer une BDD

83



Champ : permet de définir le nom du champ
Type : le type de données que va stocker le champ
Taille/Valeurs : permet d'indiquer la taille maximale du champ, utile pour le type VARCHAR notamment, afin de limiter le nombre de caractères autorisés ;
AUTO_INCREMENT : permet au champ de s'incrémenter tout seul à chaque nouvelle entrée.
On l'utilise fréquemment sur les champs de type id.

84

Table commerce a été créé(e).

requête SQL:

```
CREATE TABLE `commerce` (
  `id` INT NOT NULL AUTO_INCREMENT ,
  `nom` VARCHAR( 40 ) NOT NULL ,
  `prenom` VARCHAR( 40 ) NOT NULL ,
  `adresse` VARCHAR( 40 ) NOT NULL ,
  PRIMARY KEY ( `id` )
);
```

[Modifier] [Créer source PHP]

	Champ	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
☐	id	int(11)			Non		auto_increment	
☐	nom	varchar(40)	latin1_swedish_ci		Non			
☐	prenom	varchar(40)	latin1_swedish_ci		Non			
☐	adresse	varchar(40)	latin1_swedish_ci		Non			

Tout cocher / Tout décocher *Pour la sélection :*

85

Serveur: localhost ▸ Base de données: igefpt

Structure | SQL | Exporter | Rechercher | Requête | Opérations | Supprimer

Aucune table n'a été trouvée dans cette base.

Créer une nouvelle table sur la base igefpt:

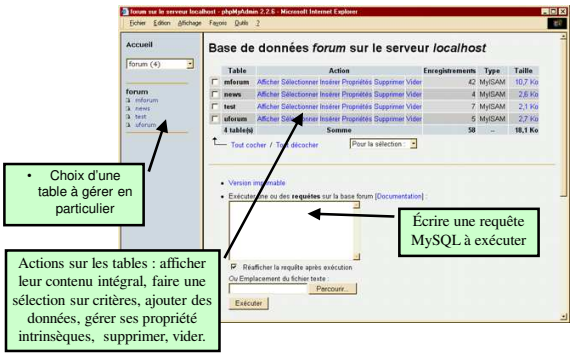
Nom:

Champs: Exécuter

- Changer le nom de la table
- Déplacer la table vers ….
- Copier la table

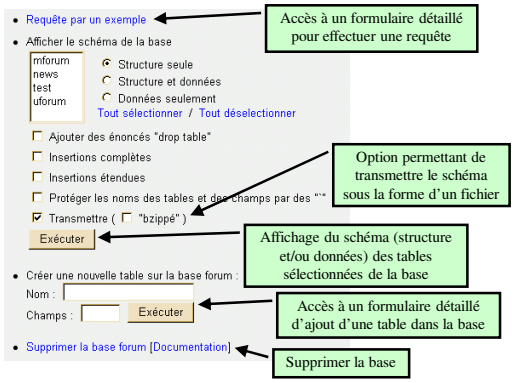
86

Gestion de la base de données (I)



87

Gestion de la base de données (II)



Accès à un formulaire détaillé pour effectuer une requête

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

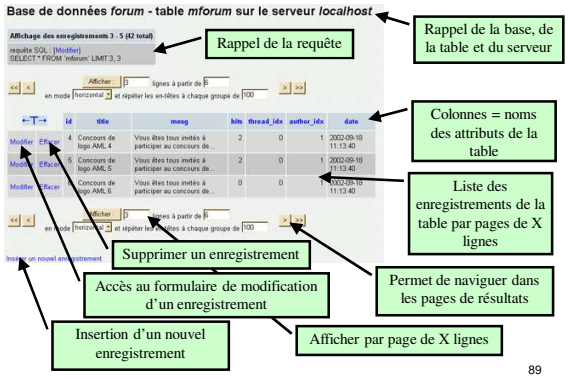
Affichage du schéma (structure et/ou données) des tables sélectionnées de la base

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

Supprimer la base

88

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

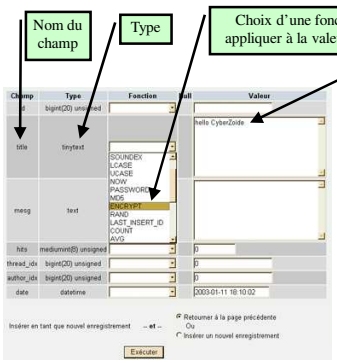
Permet de naviguer dans les pages de résultats

Insertion d'un nouvel enregistrement

Afficher par page de X lignes

89

Insertion/modification d'un enregistrement



Nom du champ

Type

Choix d'une fonction à appliquer à la valeur saisie

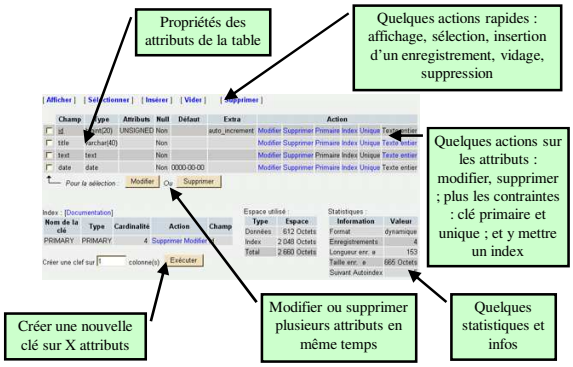
Valeur à saisir

- 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.

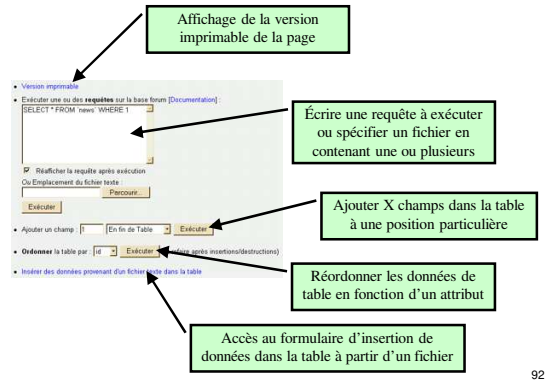
90

Gestion d'une table (I)



91

Gestion d'une table (II)



92

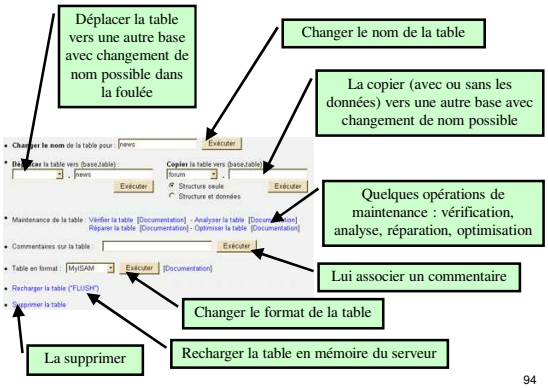
Gestion d'une table (III)

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.

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).

93

Gestion d'une table (IV)



94

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

Hugo Etiévant: MySQL pour booster votre site web PHP, 20 juillet 2003
https://fr.slideshare.net/webzan2008/php-mysql-cours

95