Introduction à MySQL

Code: mysql-intro

Originaux

url: http://tecfa.unige.ch/guides/tie/html/mysql-intro/mysql-intro.html

url: http://tecfa.unige.ch/guides/tie/pdf/files/mysql-intro.pdf

Auteurs et version

- Daniel K. Schneider - Patrick Jermann- Olivier Clavel- Vivian Synteta

- Version: 1.0 (modifié le 1/10/01 par VS)

Prérequis

- aucun (mais matière difficile)

Modules

Module technique suppl.: php-mysql

Module technique suppl.: java-mysql

© TECFA 10/1/01

Objectifs

- SQL basics :)
- A compléter: design de bases de données et grants, ...

© TECFA 10/1/01

1. Table des matières détaillée

© TECFA 10/1/01

© TECFA 10/1/01

2. Notions de base de données relationnelles

2.1 Tables et relations

Une base de donnée relationnelle est

- composée d'une ou de plusieurs **tables**, reliées d'une façon
- qui contiennent des **enregistrements**, qui ont une relation.
- Ces enregistrements sont composés de **champs** (Fields), aussi appelées colonnes.

Les champs peuvent contenir différents types de données.

- Par exemple, des nombres entiers (int) ou des chaînes de caractères d'une certaine longueur (char).

Accès à l'information dans une simple table

- Par combinaison du nom de la table et des noms de colonnes.
  ..... à suivre

© TECFA 10/1/01

2.2 Le langage SQL

La plupart des bases de données relationnelles 'parlent' un langage qui s'appelle SQL (Structured Query Language).

- **SQL permet de manipuler** les informations stockées dans la base de données. Une commande SQL est souvent appelée une **requête**.

En particulier, SQL permet de:

1. interroger (SELECT)
2. manipuler des entrées (UPDATE, INSERT, DELETE)
3. définir des données (CREATE, ALTER, DROP)
4. contrôler les accès (GRANT, REVOKE)

Syntaxe SQL:

url: http://tecfa.unige.ch/guides/mysql/fr-man/manuel_toc.html

Exemple 2-1: Structure d'une table (demo1) et des champs qui la définissent:

url: http://tecfa.unige.ch/guides/php/examples/mysql-demo/main.html

```
Database: demo  Table: demo1  Rows: 1
+----------+-----------+------+-----+---------+----------------+
| Field    | Type      | Null | Key | Default | Extra          |
+----------+-----------+------+-----+---------+----------------+
| id       | int(10)   |      | PRI | 0       | auto_increment |
| login    | char(10)  |      | MUL |         |                |
| password | char(100) | YES  |     |         |                |
| fullname | char(40)  |      |     |         |                |
| url      | char(60)  |      |     |         |                |
| food     | int(11)   |      |     | 0       |                |
| work     | int(11)   |      |     | 0       |                |
| love     | int(11)   |      |     | 0       |                |
| leisure  | int(11)   |      |     | 0       |                |
| sports   | int(11)   |      |     | 0       |                |
+----------+-----------+------+-----+---------+----------------+
```

- ignorez les détails (Type, Null, Key, Extra) pour le moment !
- voir: 4. "Définition de données (tables)" [15]

3. Interrogation (selection)

- SELECT permet de récupérer des enregistrements d'une table

Voici la syntaxe (sans les détails):

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO OUTFILE 'file_name' export_options]
    [FROM table_references
        [WHERE where_definition]
        [GROUP BY col_name,...]
        [HAVING where_definition]
        [ORDER BY {unsigned_integer | col_name} [ASC | DESC] ,...]
        [LIMIT [offset,] rows]
        [PROCEDURE procedure_name] ]
```

Nous allons ici travailler surtout avec la forme suivante

```
SELECT select_expression[1]
  FROM table_references WHERE where_definition[2] ORDER BY col_name

[1]select_expression:
  une liste de colonnes à chercher
[2]where_definition:
  voir 3.2 "Sélection conditionnelle (SELECT .... WHERE)" [10]
```

3.1 SELECT simple

Syntaxe: **SELECT** *champs1,champs2,...* **FROM** *table* OU

Syntaxe: **SELECT * FROM** *table*

Exemple 3-1: Simple selections

- Dans l'exemple ci-dessous, nous obtenons les valeurs des 5 champs (id,login,fullname,love,sports) pour tous les enregistrements contenus dans la table demo1.

```
SELECT id,login,fullname,love,sports FROM demo1
+----+-----------+-------------------+------+--------+
| id | login     | fullname          | love | sports |
+----+-----------+-------------------+------+--------+
|  1 | test      | Tester Test       |    3 |      3 |
| 34 | colin2    | Patrick Jermann2 |    1 |      4 |
....
```

- Dans l'exemple ci-dessous, nous obtenons les valeurs des tous les champs pour tous les enregistrements contenus dans la table demo1.

```
SELECT * FROM demo1
+----+-----------+----------------------------------+-----------------+-----------------------+------+--.....
| id | login     | password                         | fullname        | url                   | food | w.....
+----+-----------+----------------------------------+-----------------+-----------------------+------+--.....
|  1 | test      | 098f6bcd4621d373cade4e832627b4f6 | Tester Test     | http://tecfa.unige.ch |    3 |    ...
| 34 | colin2    | b9hhhfa9347all893u483            | Patrick Jermann2 | http://tecfa.unige.ch/ |    1 |    ...
```

3.2 Sélection conditionnelle (SELECT .... WHERE)

Syntaxe: **SELECT .... FROM** *table* **WHERE** *condition*

A. Quelques opérateurs pour les conditions:

Opérateur	Explication
opérateurs de comparaison simple	
=	égal
<> or !=	pas égal
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
opérateurs de combinaison	
AND	ET (les 2 propositions doivent être vraie)
OR	OU (une au moins des propositions est vraie)
opérateurs spéciaux	
expr IN (..., ...)	DANS (une liste)
expr NOT IN (..., ..., ...)	PAS dans ....
expr BETWEEN min AND max	Entre ... ET ...
expr NOT BETWEEN ...	(le contraire)
opérateurs de comparaison pour strings seulement	

© TECFA 10/1/01

Opérateur	Explication
expr1 LIKE expr2	comme .... wildcards: %=plusieurs chars, _=1 char
expr NOT LIKE expr2	pas comme ...
expr REGEXP pattern	comme regexp
expr NOT REGEXP pattern	pas comme regexp
STRCMP(exp1, exp2)	strcmp comme dans php, C, etc.
Flux de contrôle	
IF (expr1, expr2, expr3)	Si expr1 est vraie, retourne expr2, sinon expr3
IfNull (expr1, expr2)	Si expr1 est vraie, retourne expr1, sinon expr2
Fonctions mathématiques	
voir le manuel ...	

Note:

- Priorités: Opérateurs de comparaison, AND, OR
  en cas de doute: mettre des parenthèses (...)
- Les strings doivent être mises entre '...' ou "..." MAIS:
  Utilisez des quotes de préférence ('...'), pas des guillemets mal supportés par certaines bases de données.

Exemple 3-2: Simple Select ... where

- Si nous désirons savoir pour qui l'amour est très important (une valeur supérieure à 4) nous ajoutons une contrainte à la requête à l'aide d'un WHERE.

```
SELECT id,login,fullname,love,sports FROM demo1 WHERE love>4
+----+----------+-------------------+------+--------+
| id | login    | fullname          | love | sports |
+----+----------+-------------------+------+--------+
|  3 | colin    | Patrick Jermann   |    6 |      4 |
|  4 | schneide | Daniel Schneider  |    6 |      6 |
+----+----------+-------------------+------+--------+
```

Exemple 3-3: Select ... where

```
SELECT * from demo1 WHERE login = 'colin' AND food < 6
```

Exemple 3-4: Select ... where ... IN

- Pour obtenir juste les noms complets de tous les logins égal à 'colin' ou 'blurp'

```
SELECT fullname from demo1 WHERE login in ('colin', 'blurp')
```

Exemple 3-5: Select ... where ... BETWEEN

```
SELECT * from demo1 WHERE food BETWEEN 3 AND 5
SELECT fullname from demo1 WHERE food BETWEEN 3 AND 5 AND love > 2
```


© TECFA 10/1/01

Exemple 3-6: Select ... where ... LIKE

- Pour obtenir des informations sur les personnes dont le nom contient 'Patrick' nous utilisons l'opérateur LIKE. Les % jouent le rôle de joker, ils remplacent zéro ou plusieurs caractères.

```
SELECT id,login,fullname,love,sports FROM demo1
        WHERE fullname LIKE '%Patrick%';
+----+----------+------------------+------+--------+
| id | login    | fullname         | love | sports |
+----+----------+------------------+------+--------+
|  3 | colin    | Patrick Jermann  |    6 |      4 |
| 93 | michelon | Michelon Patrick |    6 |      6 |
+----+----------+------------------+------+--------+
```

Exemple 3-7: Select ... where ... REGEXP

```
SELECT * from demo1 WHERE fullname REGEXP 'P.*J.*'
SELECT login,fullname from demo1 WHERE fullname REGEXP 'P.*J.*';
+--------+------------------+
| login  | fullname         |
+--------+------------------+
| colin2 | Patrick Jermann2 |
| blurp  | Patrick Jermann2 |
```

3.3 Tri des lignes (SELECT ... ORDER)

- Selectionner toutes les lignes, mais triées selon id

```
SELECT * from demo1 ORDER by id
```

- La même chose mais DESC = trier dans l'ordre inverse

```
SELECT * from demo1 ORDER by id DESC
```

3.4 Compter des lignes

- Compter toutes les lignes (not null)

```
SELECT COUNT(*) FROM demo1
```

- Compter toutes les lignes qui ont une entrée de colonne (login) pareille

```
SELECT login, COUNT(*) FROM demo1 GROUP BY login;
```

3.5 Utilisation de plus d'une table

- Les colonnes sont identifiées par: `nom_table.nom_colonne`

Exemple 3-8: Select dans 2 tables, voir surtout 4.6 "Tables relationnelles" [23]

```
SELECT demo1.fullname FROM demo1, test WHERE demo1.login = test.login
+-------------+
| fullname    |
+-------------+
| Tester Test |
```

4. Définition de données (tables)

La creation d'une table implique:

- *lui donner un nom* (dans une base de données)
- *définir des colonnes:* type, taille, valeurs par défaut, ...
- *rajouter des contraîntes pour les colonnes*
- *donner éventuellement des permissions (grants)*

Voici la syntaxe (voir plus loin pour les explications):

```
Syntaxe: CREATE TABLE [IF NOT EXISTS] tbl_name
   (create_definition[1],...) [table_options] [select_statement]

[1]create_definition:
    col_name  type  [NOT  NULL  |  NULL]  [DEFAULT  default_value]
 [AUTO_INCREMENT]
           [PRIMARY KEY] [reference_definition]
 or    PRIMARY KEY (index_col_name,...)
 or    KEY [index_name] KEY(index_col_name,...)
 or    INDEX [index_name] (index_col_name,...)
 or    UNIQUE [INDEX] [index_name] (index_col_name,...)
    or           [CONSTRAINT  symbol]  FOREIGN  KEY  index_name
 (index_col_name,...)
           [reference_definition]
 or    CHECK (expr)
```

4.1 Les identificateurs MySQL

A. Règles générales

- utilisés pour: noms de db, tables, colonnes, etc.
- Taille: mots de 30 caractères max
- Caractères autorisés: lettres, chiffres, #, $, _ (le premier char doit être une lettre)
- pas de distinction entre minuscules et majuscules
  (sauf pour les bases de données et tables dans MySQL sous Unix !!)
- PAS d'accents !
- PAS de mots clefs (SELECT, WHERE, ...)

B. Tables et colonnes (champs)

- On peut utiliser le même nom de colonne dans plusieurs tables
- Les données dans une colonne doivent être toutes du même type
- Nom complet d'une colonne:

  Syntaxe: base_de_données.table.colonne
  ex: demo.demo1.login

© TECFA 10/1/01

4.2 Types de données

MySQL implémente la plupart des données SQL (mais lire le manuel pour des types exotiques !!)

Strings:

- délimitateurs: ’....’ ou " ....."
- Il faut quoter les caractères spéciaux avec le \:
  \n (newline), \r (CR), \t = (tab), \’, \", \ \, \%, _
- On peut inclure: ’"hello"’, "’hello’" (voir le manuel pour les détails)

Attributs à option (dans le tableau suivant)

- UNSIGNED (pour les entiers): nombres positives seulement
- ZEROFILL (tous les nombres): retourne des 0, par ex. 0004)

Paramètres à option (dans le tableau suivant)

- M : taille de display maximal
- D (nombres flottants): chiffres après la virgule

La valeur NULL

- Une colonne vide contient NULL (qui signifie "vide", et PAS zéro ou "" !!)

© TECFA 10/1/01

Tableau récapitulatif des types

Type	explication	range	exemple
NOMBRES			
TinyInt[(M)][UNSIGNED] [ZEROFILL]	entier minuscule	-128 à 127 (0 à 255)	TinyInt(2) 9
SmallInT[(M)]...	petit entier	-32768 à 32767 (0 à 64K)	20001
MediumINT[(M)]...	entier moyent	-8388608 to 8388607	-234567
INT[(M)] ...	entier	-2147483648 to 2147483647	
BigINT[(M)]...	gros entier	63bits	
FLOAT(precision)	nombre flottant		
FLOAT[(M,D)]...	nombre flottant	-3.402823466E+38 to -1.175494351E-38	
DOUBLE[(M,D)]...	grand nombre flottant		
DATES			
DATE	date	YYYY-MM-DD	3000-12-31
DateTime		YYYY-MM-DD HH:MM:SS	
TimeStamp[(M)]			
TIME			
YEAR			
Chaînes de caractères (strings)			
Char(M) [binary]	String de longeur fixe	M = 1 à 225 chars case insensitif (sauf binary)	char(4) 'ab '

© TECFA 10/1/01

Type	explication	range	exemple
VarChar(M)[binary]	String variable	M = 1 à 225 chars	login(8)[binary] schneiDe
Texte (blobs)			
TINYBLOB TINYTEXT	petit texts	255 chars	
BLOB TEXT		65535 chars	
MEDIUMBLOB MEDIUMTEXT		16777215 chars	
LONGBLOB LONGTEXT	grand text	4294967295 chars	
Enumération			
Enum('val1', 'val2',...)	un string parmi la liste ou NULL	65535 distinct values	'toto'
Set('val1', 'val2', ...)	zéro ou plusieurs strings parmi la liste	64 members	('toto', 'blurp')

Exemple 4-1: Création d'une table simple (CREATE)

Voici un exemple simple:

```
CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20), species
VARCHAR(20), sex CHAR(1), birth DATE);
```

4.3 Les clés

A. Index simple d'une colonne (KEY)

- Les indexes améliorent la performance des opérations SELECT
- Chaque table peut avoir 16 indexes (max.)
- On peut indexer toutes les colonnes (SAUF blob et text), mais les colonnes doivent être déclarées non NULL !!
- On peut limiter l'indexage des CHAR et VARCHAR à qq caractères

```
Syntaxe: KEY index_name (col_name)
Syntaxe: KEY index_name (char_col_name(M))
INDEX est un synonyme de KEY
```

B. Clé primaire (primary KEY)

- Une clé primaire doit être unique
- on utilise des entiers en règle générale
- Ils sont générés automatiquement en règle générale

```
Syntaxe: PRIMARY KEY (index_col_name, index_col_name)
id int(10) DEFAULT '0' NOT NULL auto_increment,
PRIMARY KEY (id),
```


© TECFA 10/1/01

4.4 Définition de colonnes

Note: L'exemple complet se trouve just après (section 4.5, p. 22)

Exemple 4-2: Colonnes dans l'exemple demo1

```
id int(10) DEFAULT '0' NOT NULL auto_increment,
login char(10) DEFAULT '' NOT NULL,
password char(100),
url char(60) DEFAULT '' NOT NULL,
food int(11) DEFAULT '0' NOT NULL,
```

Définition minimaliste d'une colonne:

```
Syntaxe: nom type
Ex: id int
```

Définition habituelle d'une colonne:

```
Syntaxe: nom type (taille) DEFAULT 'valuer_défaut' NOT NULL,
Ex: login char(10) DEFAULT '' NOT NULL,
```

Définition d'une clé primaire:

```
Syntaxe: nom int (taille) DEFAULT '0' NOT NULL auto_increment,
Ex: login char(10) DEFAULT '' NOT NULL,
```

- On rajoute (en règle générale) la définition des clefs après la définition de colonnes

```
PRIMARY KEY (id),
KEY login (login)
```

4.5 Création de tables (CREATE)

```
Syntaxe: CREATE TABLE table (colonne1  spec1,
                                                    colonne2
   spec2,
                                                    clés, )
```

Exemple 4-3: La table demo1

```
CREATE TABLE demo1 (
  id int(10) DEFAULT '0' NOT NULL auto_increment,
  login char(10) DEFAULT '' NOT NULL,
  password char(100),
  fullname char(40) DEFAULT '' NOT NULL,
  url char(60) DEFAULT '' NOT NULL,
  food int(11) DEFAULT '0' NOT NULL,
  work int(11) DEFAULT '0' NOT NULL,
  love int(11) DEFAULT '0' NOT NULL,
  leisure int(11) DEFAULT '0' NOT NULL,
  sports int(11) DEFAULT '0' NOT NULL,
  PRIMARY KEY (id),
  KEY login (login)
);
```

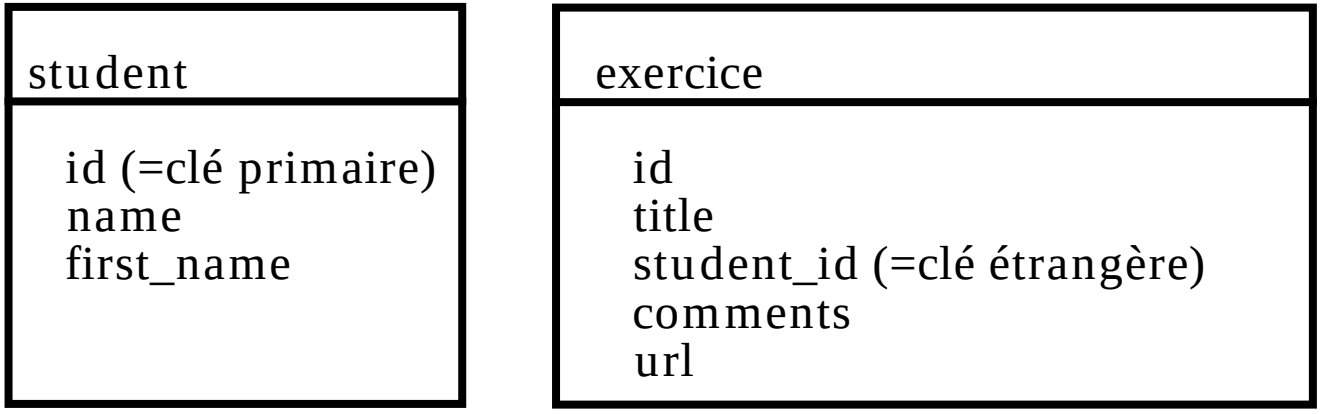
Voir 7. "Utilisation de MySQL" [30] si vous voulez faire vous-même

4.6 Tables relationnelles

- Chapitre difficile, dans cette version du document on fait trop peu !
- Une base de données contient normalement plusieurs tables reliées entre elles
- Chaque table représente une ***entité***, et les colonnes représentent des attributs
- Les relations les plus fréquentes sont du type '1-vers-N', dans ce cas:
  - on met la clé primaire du côté "1"
  - et on insère cette clé primaire du côté "N" comme clé étrangère.

Exemple simple:

- Un simple dispositif pour enregistrer des exercices d'étudiants
- 2 tables: Une pour enregistrer les étudiants et une autre pour les exercices
- Chaque étudiant peut rendre plusieurs exercices
- exercice.student_id correspond a student.id (votre application doit vérifier cela)



© TECFA 10/1/01

Création des tables avec qq données:

Le fichier student_exercice.mysql:

```
DROP TABLE IF EXISTS student;
DROP TABLE IF EXISTS exercice;
CREATE TABLE student (
  id int(10) DEFAULT '0' NOT NULL auto_increment,
  name char(40) DEFAULT '' NOT NULL,
  first_name char(40) DEFAULT '' NOT NULL,
  PRIMARY KEY (id)
);

INSERT INTO student VALUES (NULL,'Testeur','Bill');
INSERT INTO student VALUES (NULL,'Testeur','Joe');
INSERT INTO student VALUES (NULL,'Testeuse','Sophie');

CREATE TABLE exercice (
  id int(10) DEFAULT '0' NOT NULL auto_increment,
  title char(40) DEFAULT '' NOT NULL,
  student_id int(10) NOT NULL,
  comments varchar(128),
  url char(60) DEFAULT '' NOT NULL,
  PRIMARY KEY (id),
  KEY student_id (student_id)
);

INSERT INTO exercice VALUES (NULL,"Exercice 1",'1',"pas de commentaire",'http://tecfa.unige.ch/
');
INSERT INTO exercice VALUES (NULL,"Exercice 2",'1',"pas de commentaire",'http://tecfa.unige.ch/
');
```

Chargement (voir 7.2 “Traitement en "batch"” [32])

```
mysql -h tecfa -u schneide -p demo < student_exercice.mysql
```

Quelques requêtes:

- Lister les travaux d'un étudiant

```
select * FROM student,exercice WHERE student.id = exercice.student_id;
```

- Lister juste quelques colonnes de la même requête

```
select student.name, student.first_name, exercice.title, exercice.url FROM
student,exercice WHERE student.id = exercice.student_id;
+---------+------------+------------+-------------------------+
| name    | first_name | title      | url                     |
+---------+------------+------------+-------------------------+
| Testeur | Bill       | Exercice 1 | http://tecfa.unige.ch/ |
| Testeur | Bill       | Exercice 2 | http://tecfa.unige.ch/ |
+---------+------------+------------+-------------------------+
```


© TECFA 10/1/01

5. Insertion et updates

5.1 Insertion de lignes dans une table

- INSERT permet d'insérer un nouvel enregistrement dans une table.
- Notez que les champs de type char sont entre apostrophes.

INSERTION d'une ligne complète:

```
INSERT INTO demo1 VALUES (NULL,'colin', 'b9hhhfa9347all893u483', 'Patrick
Jermann','http://tecfa.unige.ch/',1,2,1,3,4)
INSERT INTO demo1 VALUES
(5,'user12','098f6bcd4621d373cade4e832627b4f6','Testuser','www.mysql.com',1,4
,5,2,1);
```

INSERTION d'une ligne en spécifiant juste qq valeurs.

```
INSERT INTO demo1 (login, fullname, food) VALUES ('test2', 'Patrick
Test',4)
```

- Attention: marche uniquement si les colonnes manquantes sont définies soit:
  - 'non null' avec des valeurs par défaut, dans ce cas la valeur par défaut est insérée

    ```
    food int(11) DEFAULT '0' NOT NULL,
    ```

  - sans déclaration 'non null' (à éviter pour des raisons de performance), dans ce cas NULL va être inséré.

    ```
    fun int(11), ... fun2 int (11) DEFAULT '1'
    ```


© TECFA 10/1/01

5.2 Mise à jour du contenu d'une table

- UPDATE permet de mettre à jour un ou plusieurs champs dans un ou plusieurs enregistrements.

  Syntaxe: `UPDATE [LOW_PRIORITY] tbl_name SET col_name1=expr1,col_name2=expr2,... [WHERE where_definition]`

Mise à jour d'un champ ('sports') pour un utilisateur ('michelon'):

```
UPDATE demo1 SET sports=3 WHERE login='michelon';
```

Edition de valeurs de deux champs ('love' et 'leisure') en même temps:

```
UPDATE demo1 SET love=5, leisure=4 WHERE login='michelon';
```

Edition avec calcul (rajouter 3 à sports)

```
UPDATE demo1 SET sports=sports+3 WHERE login='test2'
```

Pour l'édition d'une ligne bien précise:

- Toujours prendre la colonne avec le "primary key" !!
- dans l'exemple ci-dessous on met à jour tous les michelon dans la table

© TECFA 10/1/01

5.3 Effacement de lignes d'une table

- Pour effacer tous les enregistrements d'une table

```
DELETE FROM people;
```

- Pour n'effacer que un enregistrement:

```
DELETE FROM people WHERE Id=1;
```


© TECFA 10/1/01

6. Modification/destruction d'une table

6.1 Destruction d'une table

- Attention, réflechissez avant, ou faites un dump par exemple

Tuer une table:

```
DROP TABLE [IF EXISTS] table
ex: DROP TABLE demo2
```

6.2 Modifications de la structure d'une table

- Voir le manuel pour les détails .....

```
Syntaxe: ALTER TABLE table ......
```

Pour rajouter une colonne:

```
Syntaxe: ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
ex: ALTER TABLE demo2 ADD COLUMN fun int(11) DEFAULT '0' NOT NULL
AFTER love;
```

Pour tuer une colonne:

```
Syntaxe: DROP [COLUMN] column_name
ex: ALTER TABLE demo2 DROP fun;
```


© TECFA 10/1/01

7. Utilisation de MySQL

7.1 L'interface SQL "ligne de commande"

A. MySQL possède une simple interface "ligne de commande"

- les commandes SQL doivent être séparées par un "**;**" (!!!)
- Les exemples suivants supposent que vous avez accès à une base de données

B. Connexion à un serveur MySQL (depuis un terminal unix / telnet)

```
Syntaxe: mysql -h machine -u utilisateur -p [base_de_données]
   -h: machine hôte
   -u: utilisateur MySQL (pas Unix)
   -p: mot de passe MySQL (pas Unix)
mysql -h tecfasun5 -u schneide -p
Enter password: ********
```

C. Utilisation/changement d'une base de données (USE)

```
mysql> USE demo;
```

ou directement lors de la connexion:

```
mysql -h tecfasun1 -u schneide -p demo
```


© TECFA 10/1/01

D. Lister les tables (SHOW)

```
mysql> SHOW TABLES;
+----------------+
| Tables in demo |
+----------------+
| demo1          |
| test           |
```

E. Décrire la structure d'une table (DESCRIBE)

```
mysql> DESCRIBE demo1;
+----------+-----------+------+-----+---------+----------------+
| Field    | Type      | Null | Key | Default | Extra          |
+----------+-----------+------+-----+---------+----------------+
| id       | int(10)   |      | PRI | 0       | auto_increment |
| login    | char(10)  |      | MUL |         |                |
| password | char(100) | YES  |     | NULL    |                |
| fullname | char(40)  |      |     |         |                |
| url      | char(60)  |      |     |         |                |
| food     | int(11)   |      |     | 0       |                |
| work     | int(11)   |      |     | 0       |                |
| love     | int(11)   |      |     | 0       |                |
| leisure  | int(11)   |      |     | 0       |                |
| sports   | int(11)   |      |     | 0       |                |
+----------+-----------+------+-----+---------+----------------+
```

7.2 Traitement en "batch"

```
mysql -h tecfasun5 -u schneide -p demo < test.mysql
```

- Le contenu du fichier test.mysql contient les commandes SQL pour créer une table, voir par exemple 4.5 “Création de tables (CREATE)” [22]
- N’oubliez pas d’indiquer votre base de données ("demo" ci-dessus) !

- Note: si la table existe déjà, il faut la tuer avec "DROP TABLE if exists" (sinon utiliser ALTER pour modifier sa structure)

```
Syntaxe: DROP TABLE [IF EXISTS] table
Ex: DROP TABLE demo2
Ex: DROP TABLE if exists demo4
```

7.3 Sauvegardes

- On peut "dumper" une base de données en entier
- Cela crée les commandes SQL pour tout restituer (y compris les INSERT)

Avec l’utilitaire ’mysqldump’ (sortir de mysql!):

```
mysqldump -h tecfa -u schneide -p demo > save.mysql
```


© TECFA 10/1/01

7.4 Lister des bases de données, tables, etc.

Note: 'vivian' dans les exemples est à la fois un nom d'utilisateur et de base de données.

Lister toutes les bases de données sur un serveur:

```
mysqlshow -h tecfasun5 -u vivian -p
mysqlshow -h tecfasun5 -u vivian --password=*******
```

Lister les tables dans une base de données

```
mysqlshow -h tecfasun5 -u vivian -p vivian
mysqlshow -h tecfasun5 -u vivian --password=******* vivian
```

Lister la définition d'une table

```
mysqlshow -h tecfasun5 -u vivian -p vivian test
```


© TECFA 10/1/01

© TECFA 10/1/01