# 1 Installation serveur et client MySQL sous Linux

## 1.1 Verification du démarrage du serveur MySQL

Après une installation, le démarrage du serveur à chaque démarrage de l'OS et activé. Il est possible manuellement de démarrer ce service avec le script : *mysqld_safe*
Il est possible de vérifier l'activité du service :

```
ps ax | grep mysql
/bin/sh /usr/bin/mysqld_safe
/usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --user=mysql ... --port=3306 --socket=/var/run/mysqld/mysqld.sock
logger -p daemon.err -t mysqld_safe -i -t mysqld
```

*mysqladmin variables -u root ...* : permet de consulter les variables d'environnement
*mysqladmin status -u root ...* : permet de connaître l'état
*mysqladmin version -u root ...* : permet de connaître la version.

## 1.2 Affectation d'un mot de passe au root de MySQL

L'installation initiale demande désormais (depuis 5.0) un mot de passe pour l'administrateur . Il est possible de modifier le mot de passe à l'utilisateur root du serveur 'mysql'. Un utilitaire permet cette attribution/modification :

```
mysqladmin –u root password nouveau_mon_de_passe
```

# 2 Exploitation du client fourni

Chaque accès au serveur doit être demandé via un nom d'utilisateur

## 2.1 Les Utilisateurs

La gestion des utilisateurs est primordiale dans la vie d'une base de données relationnelle. Protéger les données est ce qui motive en premier lieu la mise en place de comptes utilisateur et de mesures de sécurité.

- L'utilisateur doit disposer des rôles et des privilèges inhérents à l'accomplissement de son travail.
- Aucun utilisateur ne doit bénéficier de droits d'accès dépassant l'étendue de ses fonctions.
- Lorsqu'un utilisateur n'a plus besoin d'un accès, son compte doit être soit supprimé, soit désactivé.
- Tout utilisateur a sa place dans la base de données, mais certains ont plus de responsabilités que d'autres.
- Des objets de la base de données sont associés à des comptes utilisateur ; c'est ce que l'on appelle des schémas. Un schéma est un jeu d'objets qui appartient à un utilisateur. Ce dernier est également appelé propriétaire du schéma. La différence entre un utilisateur normal et un propriétaire de schéma est que ce dernier possède des objets dans une base de données, ce qui n'est pas le cas de la plupart des utilisateurs. Beaucoup disposent de comptes qui permettent d'accéder aux données contenues dans d'autres schémas.
- La création d'utilisateurs implique l'utilisation de commandes de type SQL dans la base de données. Il n'existe pas de standard dans SQL ; chaque implémentation possède sa propre méthode.

Les noms d'utilisateurs, utilisés par MySQL pour l'authentification, n'ont rien à voir avec les noms d'utilisateur de Unix (Nom de login) ou de Windows. Ils peuvent avoir jusqu'à 16 caractères de long. Les mots de passe MySQL sont cryptés avec un cryptage différent de celui d'Unix. Voir commandes `PASSWORD()et ENCRYPT()`.
Syntaxe d'accès au serveur MySQL via le client 'mysql' avec identification:

- *mysql [-h host_name] [-u user_name] [-p[mdp]]* (!pas d'espace entre -p et mot de passe)
- `mysql --host=host_name --user=user_name --password=mdp`

!!! transmettre le mot de passe dans la ligne de commande en clair??? L'option '-p ' seule indique qu'il sera demandé après, et non affiché.

Exemple d'accés à la base mysql via l'utilisateur root avec demande de mot de passe.

```
mysql –p –u root mysql
Enter password:
mysql>
```

## 2.2 Information sur les tables (d'administration ou ) de la base 'mysql'

Les droits sont conservés dans les tables `user`, `db`, `host`, `tables_priv` et `columns_priv` de la base de données. Le serveur **MySQL** lit le contenu de ces tables au démarrage, et à chaque fois que c'est nécessaire.

La table `user` détermine le droit de connexion.
Les tables `db` et `host` sont utilisées ensemble :

- La table `db` détermine quelles bases sont accessibles à quels utilisateurs.
- La table `host` est utilisée comme une extension de `db` lorsque vous voulez qu'une ligne de `db` s'applique à plusieurs hôtes. Par exemple, si vous voulez qu'un utilisateur soit capable d'accéder à la base depuis plusieurs hôtes différents, laissez le champs `Host` de la table `db` vide, puis ajoutez un enregistrement dans la table `host` pour chaque hôte à autoriser.

Les tables `tables_priv` et `columns_priv` sont similaires à la table `db`, mais s'appliquent au niveau table et colonne, plutôt qu'au niveau base.

Les différents droits (Selection, Update…) pour une colonne sont calculés à partir des privilèges comme suit :

**Global Privileges OR (Database Privileges AND Host Privileges) OR Table Privileges OR Column Privileges**

**Niveau général (Global level) :** Les droits de niveau général s'appliquent à toutes les bases du serveur. Ils sont stockés dans la table **`mysql.user`**.
**Niveau base de données (Database level):** Les droits de niveau base de données s'appliquent aux tables d'une base. Ils sont stockés dans les tables **`mysql.db`** et **`mysql.host`**.
**Niveau table (Table level) :** Les droits de niveau table s'appliquent aux colonnes d'une table donnée. Ils sont stockés dans la table **`mysql.tables_priv`** .

**Niveau colonne (Column level) :** Les droits de niveau colonne s'appliquent à une colonne donnée d'une table donnée. Ils sont stockés dans la table **`mysql.columns_priv`** .

### 2.2.1 Les droits des utilisateurs pour les connexions

```
mysql> show tables from mysql;
+---------------------------+
| Tables_in_mysql           |
+---------------------------+
| columns_priv              |
| db                        |
| func                      |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| host                      |
| proc                      |
| procs_priv                |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

```
mysql> describe user;
+------------------------+-----------------------------------+------+-----+---------+
| Field                  | Type                              | Null | Key | Default |
+------------------------+-----------------------------------+------+-----+---------+
| Host                   | char(60)                          | NO   | PRI |         |
| User                   | char(16)                          | NO   | PRI |         |
| Password               | char(41)                          | NO   |     |         |
| Select_priv            | enum('N','Y')                     | NO   |     | N       |
| Insert_priv            | enum('N','Y')                     | NO   |     | N       |
| Update_priv            | enum('N','Y')                     | NO   |     | N       |
| Delete_priv            | enum('N','Y')                     | NO   |     | N       |
| Create_priv            | enum('N','Y')                     | NO   |     | N       |
| Drop_priv              | enum('N','Y')                     | NO   |     | N       |
| Reload_priv            | enum('N','Y')                     | NO   |     | N       |
| Shutdown_priv          | enum('N','Y')                     | NO   |     | N       |
| Process_priv           | enum('N','Y')                     | NO   |     | N       |
| File_priv              | enum('N','Y')                     | NO   |     | N       |
| Grant_priv             | enum('N','Y')                     | NO   |     | N       |
| References_priv        | enum('N','Y')                     | NO   |     | N       |
| Index_priv             | enum('N','Y')                     | NO   |     | N       |
| Alter_priv             | enum('N','Y')                     | NO   |     | N       |
| Show_db_priv           | enum('N','Y')                     | NO   |     | N       |
| Super_priv             | enum('N','Y')                     | NO   |     | N       |
| Create_tmp_table_priv  | enum('N','Y')                     | NO   |     | N       |
| Lock_tables_priv       | enum('N','Y')                     | NO   |     | N       |
| Execute_priv           | enum('N','Y')                     | NO   |     | N       |
| Repl_slave_priv        | enum('N','Y')                     | NO   |     | N       |
| Repl_client_priv       | enum('N','Y')                     | NO   |     | N       |
| Create_view_priv       | enum('N','Y')                     | NO   |     | N       |
| Show_view_priv         | enum('N','Y')                     | NO   |     | N       |
| Create_routine_priv    | enum('N','Y')                     | NO   |     | N       |
| Alter_routine_priv     | enum('N','Y')                     | NO   |     | N       |
| Create_user_priv       | enum('N','Y')                     | NO   |     | N       |
| ssl_type               | enum('','ANY','X509','SPECIFIED') | NO   |     |         |
| ssl_cipher             | blob                              | NO   |     |         |
| x509_issuer            | blob                              | NO   |     |         |
| x509_subject           | blob                              | NO   |     |         |
| max_questions          | int(11) unsigned                  | NO   |     | 0       |
| max_updates            | int(11) unsigned                  | NO   |     | 0       |
| max_connections        | int(11) unsigned                  | NO   |     | 0       |
| max_user_connections   | int(11) unsigned                  | NO   |     | 0       |
+------------------------+-----------------------------------+------+-----+---------+
```

```
mysql> desc mysql.host;
+-----------------------+---------------+------+-----+---------+
| Field                 | Type          | Null | Key | Default |
+-----------------------+---------------+------+-----+---------+
| Host                  | char(60)      | NO   | PRI |         |
| Db                    | char(64)      | NO   | PRI |         |
| Select_priv           | enum('N','Y') | NO   |     | N       |
| Insert_priv           | enum('N','Y') | NO   |     | N       |
| Update_priv           | enum('N','Y') | NO   |     | N       |
| Delete_priv           | enum('N','Y') | NO   |     | N       |
| Create_priv           | enum('N','Y') | NO   |     | N       |
| Drop_priv             | enum('N','Y') | NO   |     | N       |
| Grant_priv            | enum('N','Y') | NO   |     | N       |
| References_priv       | enum('N','Y') | NO   |     | N       |
| Index_priv            | enum('N','Y') | NO   |     | N       |
| Alter_priv            | enum('N','Y') | NO   |     | N       |
| Create_tmp_table_priv | enum('N','Y') | NO   |     | N       |
| Lock_tables_priv      | enum('N','Y') | NO   |     | N       |
| Create_view_priv      | enum('N','Y') | NO   |     | N       |
| Show_view_priv        | enum('N','Y') | NO   |     | N       |
| Create_routine_priv   | enum('N','Y') | NO   |     | N       |
| Alter_routine_priv    | enum('N','Y') | NO   |     | N       |
| Execute_priv          | enum('N','Y') | NO   |     | N       |
+-----------------------+---------------+------+-----+---------+
```

```
mysql> desc mysql.db;
+-----------------------+---------------+------+-----+---------+
| Field                 | Type          | Null | Key | Default |
+-----------------------+---------------+------+-----+---------+
| Host                  | char(60)      | NO   | PRI |         |
| Db                    | char(64)      | NO   | PRI |         |
| User                  | char(16)      | NO   | PRI |         |
| Select_priv           | enum('N','Y') | NO   |     | N       |
| Insert_priv           | enum('N','Y') | NO   |     | N       |
| Update_priv           | enum('N','Y') | NO   |     | N       |
| Delete_priv           | enum('N','Y') | NO   |     | N       |
| Create_priv           | enum('N','Y') | NO   |     | N       |
| Drop_priv             | enum('N','Y') | NO   |     | N       |
| Grant_priv            | enum('N','Y') | NO   |     | N       |
| References_priv       | enum('N','Y') | NO   |     | N       |
| Index_priv            | enum('N','Y') | NO   |     | N       |
| Alter_priv            | enum('N','Y') | NO   |     | N       |
| Create_tmp_table_priv | enum('N','Y') | NO   |     | N       |
| Lock_tables_priv      | enum('N','Y') | NO   |     | N       |
| Create_view_priv      | enum('N','Y') | NO   |     | N       |
| Show_view_priv        | enum('N','Y') | NO   |     | N       |
| Create_routine_priv   | enum('N','Y') | NO   |     | N       |
| Alter_routine_priv    | enum('N','Y') | NO   |     | N       |
| Execute_priv          | enum('N','Y') | NO   |     | N       |
+-----------------------+---------------+------+-----+---------+
```

```
mysql> desc mysql.tables_priv;
+-------------+-----------------------------------------------------------------------------------+------+-----+-------------------+
| Field       | Type                                                                              | Null | Key | Default           |
+-------------+-----------------------------------------------------------------------------------+------+-----+-------------------+
| Host        | char(60)                                                                          | NO   | PRI |                   |
| Db          | char(64)                                                                          | NO   | PRI |                   |
| User        | char(16)                                                                          | NO   | PRI |                   |
| Table_name  | char(64)                                                                          | NO   | PRI |                   |
| Grantor     | char(77)                                                                          | NO   | MUL |                   |
| Timestamp   | timestamp                                                                         | NO   |     | CURRENT_TIMESTAMP |
| Table_priv  | set('Select','Insert','Update','Delete','Create','Drop','Grant','References',     | NO   |     |                   |
|             |     'Index','Alter','Create View','Show view')                                    |      |     |                   |
| Column_priv | set('Select','Insert','Update','References')                                      | NO   |     |                   |
+-------------+-----------------------------------------------------------------------------------+------+-----+-------------------+
```

```
mysql> desc mysql.columns_priv;
+-------------+---------------------------------------------+------+-----+-------------------+
| Field       | Type                                        | Null | Key | Default           |
+-------------+---------------------------------------------+------+-----+-------------------+
| Host        | char(60)                                    | NO   | PRI |                   |
| Db          | char(64)                                    | NO   | PRI |                   |
| User        | char(16)                                    | NO   | PRI |                   |
| Table_name  | char(64)                                    | NO   | PRI |                   |
| Column_name | char(64)                                    | NO   | PRI |                   |
| Timestamp   | timestamp                                   | NO   |     | CURRENT_TIMESTAMP |
| Column_priv | set('Select','Insert','Update','References') | NO   |     |                   |
+-------------+---------------------------------------------+------+-----+-------------------+
```

- Les droits **select**, **insert**, **update** et **delete** permettent d'exécuter des opérations sur les lignes dans les tables existantes d'une base.
- Le droit **index** vous permet de créer ou de détruire des index.
- Le droit **alter** vous permet d'utiliser la commande `ALTER TABLE`.
- Les droits **create** et **drop** vous permet de créer et détruire de nouvelles tables et bases. !!! Un utilisateur ayant le droit **drop** pour la base `mysql`, il pourra effacer la table des droits **MySQL**!

- Le droit **grant** vous permet de donner des droits que vous possédez à un autre utilisateur.
- Le droit **file** permet de lire et écrire des fichiers sur le serveur, en utilisant les commandes `LOAD DATA INFILE` et `SELECT INTO OUTFILE`.

Pour les opérations administratives, disponibles aussi avec `mysqladmin`.

Le droit **reload** permet d'effectuer les commandes suivantes. La commande reload force le serveur à relire les tables de droits. Voir refresh, flush-privileges, flush-hosts, flush-logs, flush-tables.

Le droit **shutdown** permet, grâce à la commande shutdown, d'éteindre le serveur.

Le droit **process** permet d'effectuer les commandes suivantes. La commande processlist affiche la liste des threads courant du serveur. La commande kill arrête les threads du serveur. Vous pouvez toujours afficher et détruire les threads que vous possédez, mais vous devez avoir les droits de **process** pour afficher ou terminer les threads d'un autre utilisateur.

Le champs `Host` peut être un nom d'hôte, ou une adresse IP ou `'localhost'`. Pour accepter tous les hôtes, `Host` vaut `'%'`.
Exemple d'association de champs Hôte et utilisateur.

```
Host                  User   Connexions corresponsdantes
'thomas.loc.gov'      'fred' fred, se connectant depuis thomas.loc.gov
'thomas.loc.gov'      ''     Utilisateur anonyme, se connectant depuis thomas.loc.gov
'%'                   'fred' fred, se connectant depuis n'importe quel hôte
'%'                   ''     Utilisateur anonyme se connectant depuis n'importe quel hôte
'%.loc.gov'           'fred' fred, se connectant depuis un hôte du domaine loc.gov
'144.155.166.177'     'fred' fred, se connectant depuis l'hôte avec une adresse IP 144.155.166.177
'144.155.166.%'       'fred' fred depuis n'importe quel hôte du sous réseau classe C 144.155.166
```

!!! L'ordre est important lors de l'identification de l'accès.
`USER(), SYSTEM_USER(), SESSION_USER()` Retourne le nom de l'utilisateur **MySQL** courant. :
`mysql> select USER();` Voir aussi current_date(), current_time(), version() …

### 2.2.2 Création d'un utilisateur

Plusieurs syntaxes sont possibles. Soit des commandes SQL :

```
mysql> INSERT INTO user VALUES('localhost','superutilisateur',PASSWORD('motdepasse'),
'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','', ...);
mysql> INSERT INTO user SET Host='localhost',User='admin', Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User,Password) VALUES('localhost','invite','');
mysql> INSERT INTO db (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv, Create_priv,Drop_priv) VALUES
('localhost','basededonnees', 'invite','Y','Y','Y','Y','Y','Y');
```

L'utilisateur 'invite' n'a aucun privilège de connexion mais va obtenir des droits sur la base de données *'basededonnees'*

`PASSWORD(str)` Calcule un mot de passe à partir du texte de `str`. C'est une fonction d'encryption utilisée par **MySQL** pour stocker les mots de passes dans la colonne `Password`, de la table des droits des utilisateurs.

```
mysql> select PASSWORD('mot_de_passe_en_clair');
```

Le cryptage de `PASSWORD()` n'est pas réversible. `PASSWORD()` effectue un cryptage différent de celui d'Unix.

`ENCRYPT(chaine[,clé])` Crypte la chaîne `chaine` en utilisant le cryptage d'Unix( `crypt()`)

### 2.2.3 Mise en service des nouveaux droits

- Les modifications des tables de droits qui se font avec différentes requêtes. Les requêtes `GRANT, REVOKE,` ou `SET PASSWORD` sont immédiatement prises en compte.

- Les requêtes permettant de travailler directement sur les tables (`INSERT, UPDATE,` etc.), peuvent modifier les privilèges mais ne forcent pas le serveur à considérer les nouveaux droits.

- Pour forcer le serveur à considérer les changements de tables,
  - soit, exécuter la commande suivant dans le client : mysql> FLUSH PRIVILEGES;
  - soit lancer l'exécutable `mysqladmin flush-privileges`
  - sinon, les changements ne seront pas effectifs avant le prochain rechargement.

## 2.3 Exploitation d'une base de données sous le serveur MySQL

La commande **`USE`** `Nom_bdd` indique à **MySQL** le nom de la base de données par défaut pour les requêtes suivantes. Cette base restera courante jusqu'à la fin de la session, ou jusqu'à la prochaine utilisation de la commande `USE.`

### 2.3.1 Création et suppression d'une base de données

**CREATE DATABASE** Nom_bdd : Avec **MySQL**, les bases sont représentées sous la forme de dossiers qui contiendront des fichiers, qui représenteront les tables. Cette fonction se contente de créer un dossier dans le dossier données de **MySQL**.

`DATABASE()` Retourne le nom de la base de données courante : `mysql> select DATABASE();`

DROP DATABASE [IF EXISTS] Nom_bdd : efface toutes les tables de la base , et efface aussi la base.
Étant donné que chaque table est constituée de trois fichiers, (`'.ISD', '.ISM', '.frm'`) cette fonction effacera 3*nb_tables fichiers.

`Nom_table.frm` : Définition de la table (forme).
`Nom_table.ISD` : Fichier de données.
`Nom_table.ISM` Fichier d'index.

### 2.3.2 Création de tables

Syntaxe :
***CREATE TABLE** [IF NOT EXISTS] Nom_table (**create_definition**,...) [**opts_de_table**] [cmd_de_selection]*

**create_definition** peut avoir les définitions suivantes :
*Nom_col **type** [NOT NULL | NULL] [DEFAULT val_defaut] [AUTO_INCREMENT] [PRIMARY KEY] [reference_def]* : avec par défaut `NULL, AUTO_INCREMENT` pour 1 seule colonne de type entier devant être indexée
***PRIMARY KEY** (index_Nom_col,...)*
***KEY** [Nom_index] KEY(index_Nom_col,...)* identique à ***INDEX** [Nom_index] (index_Nom_col,...)*
***UNIQUE** [INDEX] [Nom_index] (index_Nom_col,...)*
*[**CONSTRAINT** symbole] **FOREIGN KEY** Nom_index(index_Nom_col,...) [reference_definition]*
***CHECK** (expression )*

l'attribut **UNIQUE** force la valeur à toujours prendre une valeur distincte. Erreur si valeur déjà utilisée.
l'attribut **PRIMARY KEY** est identique à l'attribut `KEY` qui porterait le nom de `PRIMARY.` Une table ne peut avoir qu'une seule colonne avec l'attribut table `PRIMARY KEY`. Si aucune colonne n'a de `PRIMARY KEY` et qu'une application requiert la colonne de `PRIMARY KEY`, **MySQL** retournera la première colonne ayant l'attribut `UNIQUE.` Une `PRIMARY KEY` peut être un index multi colonne.
La liste des noms d'index est accessible avec la requête : `SHOW INDEX FROM Nom_table.` Si aucun nom n'est affecté à un index, un nom sera automatiquement généré et assigné, celui de la première colonne. Seule les tables de type MyISAM acceptent les index sur les colonnes qui contiennent la valeur `NULL`. Dans les autres cas, il faut absolument déclarer la colonne de type `NOT NULL` pour ne pas générer une erreur.
L'attribut `FOREIGN KEY`, ajoute une contrainte de clé étrangère.
L'attribut `CHECK`, ajoute une contrainte de vérification sur l'affectation.
L'attribut `REFERENCES` ne font en réalité rien de spécial.
Ces 3 derniers attributs sont actifs en fonction de la version de MySQL (à vérifier)

**opts_de_table** peut avoir les définitions suivantes:

| | |
|---|---|
| *type = [ISAM \| MYISAM \| HEAP]* | (ISAM Gestion originale de la table MyISAM Nouvelle gestion binaire et portable de la table HEAP Gestion de la table exclusivement en mémoire vive) |
| *auto_increment = #* | prochaine valeur à affecter pour la colonne |
| *avg_row_longueur = #* | approximation de la taille moyenne de la longueur d'une ligne |
| *checksum = [0 \| 1]* | utilisé pour la vérification de l'intégrité de la table en calculant un somme (ralenti la table) |
| *comment = "string"* | 60 caractères max de commentaire pour la table |
| *rows_max= #* | nombre maximal de ligne dans la table |
| *rows_min = #* | nombre minimal de ligne dans la table |
| *pack_keys = [0 \| 1]* | Réduit la taille des index |

#### 2.3.2.1 Les types de champs possibles sous MySQL

**MySQL** dispose d'un grand nombre de type de colonnes. Ces types peuvent être regroupés en trois catégories : les types numériques, les types date et heure, et les types chaînes de caractères.

**M** Indique la taille maximale d'affichage. La taille maximale autorisée par défaut est 255.
**D** S'applique aux types à virgule flottante, et précise le nombre de chiffre après la virgule.
**Crochets (' ['et ']')** indique que cet argument est optionnel.
NB : Il est toujours possible de spécifier `ZEROFILL` pour une colonne. **MySQL** ajoutera alors automatiquement l'attribut `UNSIGNED` à la colonne.

***TINYINT[(M)] [UNSIGNED] [ZEROFILL]*** Un très petit entier. Signé, il couvre l'intervalle -128 à 127 ; non signé, il couvre 0 à 255.
***SMALLINT[(M)] [UNSIGNED] [ZEROFILL]*** Un petit entier. Signé, il couvre l'intervalle -32768 à 32767; non signé, il couvre 0 à 65535.
***MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]*** Un entier de taille intermédiaire. Signé, il couvre l'intervalle -8388608 à 8388607; non signé, il couvre 0 à 16777215.
***INT[(M)] [UNSIGNED] [ZEROFILL]*** Un entier de taille normale. Signé, il couvre l'intervalle -2147483648 à 2147483647; non signé, il couvre 0 à 4294967295.
***INTEGER[(M)] [UNSIGNED] [ZEROFILL]*** Un synonyme pour INT.
***BIGINT[(M)] [UNSIGNED] [ZEROFILL]*** Un entier de grande taille. Signé, il couvre l'intervalle -9223372036854775808 à 9223372036854775807; non signé, il couvre 0 à 18446744073709551615.
NB : toutes les opérations arithmétiques effectuée en interne, utilise des `BIGINT` signés ou `DOUBLE` , donc il ne faut pas utiliser les grands entiers non signé au delà de 9223372036854775807 (63 bits), hormis pour les fonctions sur les bits.
***FLOAT(precision) [ZEROFILL]*** Un nombre à virgule flottante. Il est obligatoirement signé. `precision` peut prendre les valeurs de 4 ou 8. `FLOAT(8)` est un nombre à précision double. Ces types sont identiques aux types FLOAT et DOUBLE décrit ci-dessous, mais leur précision peut être paramétrée. Avec **MySQL** 3.23, ce sont de vrais nombres à virgule flottante, alors qu'avec les anciennes versions, FLOAT(precision) n'avait que 2 décimales. Cette syntaxe a été ajoutée pour assurer la compatibilité ODBC.
***FLOAT[(M,D)] [ZEROFILL]*** Un nombre à virgule flottante, en précision simple. Il est toujours signé. Les valeurs sont comprises -3.402823466E+38 et -1.175494351E-38.

***DOUBLE[(M,D)] [ZEROFILL]*** Un nombre à virgule flottante, en précision double. Il est toujours signé. Les valeurs sont comprises -1.7976931348623157E+308 et 2.2250738585072014E-308.

***DOUBLE PRECISION[(M,D)] [ZEROFILL]***

***REAL[(M,D)] [ZEROFILL]*** Des synonymes pour DOUBLE.

***DECIMAL(M,D) [ZEROFILL]*** Un nombre à virgule flottante. Il est toujours signé. Il se comporte comme une colonne CHAR . Il n'est pas paqué, c'est à dire que le nombre est stocké comme une chaîne de chiffre. Chaque chiffre, le signe moins, la virgule occupe un caractère. Si D vaut 0, le nombre n'aura pas de décimales, ni de virgule. La taille maximale pour les décimales est la même que pour les DOUBLE , mais en il peut être limité par le choix de of M et D. Avec **MySQL** 3.23, M n'inclut plus le signe moins`-', ni la virgule des nombres décimaux (norme ANSI SQL.).

***NUMERIC(M,D) [ZEROFILL]*** Un synonyme pour DECIMAL.

***DATE*** Une date. L'intervalle valide de date va de '1000-01-01' à '9999-12-31'. **MySQL** affiche les DATE avec le format , mais il est possible d'affecter des DATE en utilisant indifféremment des chaînes ou des nombres.

***DATETIME*** Une combinaison de date et d'heure. L'intervalle valide va de '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. **MySQL** affiche DATETIME avec le format 'YYYY-MM-DD HH:MM:SS', mais il est possible d'affecter des **DATETIME** en utilisant indifféremment des chaînes ou des nombres.

***TIMESTAMP[(M)]*** Un timestamp : la date et l'heure, exprimée en secondes, depuis le 1er janviers 1970. Il permet de couvrir un intervalle allant de'1970-01-01 00:00:00' à quelque part, durant l'année 2037. **MySQL** affiche les TIMESTAMP avec les format YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD ou YYMMDD, suivant que M vaut 14 (ou absent), 12, 8 ou 6, mais il est possible d'affecter des TIMESTAMP en utilisant indifféremment des chaînes ou des nombres. Une colonne de type TIMESTAMP est très pratique pour enregistrer des dates et heures lors d'un INSERT ou UPDATE, car cette colonne sera automatiquement mis à la date et heure de l'opération la plus récente, si aucune valeur n'est précisée. Il est aussi possible d'affecter l'heure courante en assignant la valeur NULL à une colonne de type .

***TIME*** Une mesure de l'heure. L'intervalle valide est '-838:59:59' à '838:59:59'. **MySQL** affiche TIME au format 'HH:MM:SS'**,** mais il est possible d'affecter des TIME en utilisant indifféremment des chaînes ou des nombres.

***YEAR*** Un an. L'intervalle valide est 1901 à 2155, et 0000. **MySQL** affiche YEAR au format YYYY, mais il est possible d'affecter des YEAR en utilisant indifféremment des chaînes ou des nombres(Le type YEAR est nouveau en **MySQL** 3.22.)

***CHAR(M) [BINARY]*** Une chaine de caractère de taille fixe, et toujours complétée à droite par des espaces. M va de 1 à 255 caractères. Les espaces supplémentaires sont supprimés lorsque la valeur est retournée dans une requête. Les tris et comparaisons effectués sur des valeurs de type CHAR sont insensibles à la casse, à moins que le mot clé BINARY soit précisé.

***VARCHAR(M) [BINARY]*** Une chaine de caractère de longueur variable. Les espaces en fin de chaîne sont supprimés lorsque la chaine est stockée (ce n'est pas conforme à la norme ANSI SQL). Va de 1 a 255 caractères . Les tris et comparaisons effectués sur des valeurs de type VARCHAR sont insensibles à la casse, à moins que le mot clé BINARY soit précisé.

***TINYBLOB***

***TINYTEXT*** Un objet BLOB ou TEXT avec une longueur maximale de 255 (2^8 - 1).

***BLOB*** Un objet BLOB avec une longueur maximale de 65535 (2^16 - 1). Sensible à la casse

***TEXT*** Un objet TEXT avec une longueur maximale de 65535 (2^16 - 1). Insensible à la casse

***MEDIUMBLOB***

***MEDIUMTEXT*** Un objet BLOB ou TEXT avec une longueur maximale de 16777215 (2^24 - 1).

***LONGBLOB***

***LONGTEXT*** Un objet BLOB ou TEXT avec une longueur maximale de 4294967295 (2^32 - 1). Voir aussi la section

***ENUM('value1','value2',...)*** Une énumération. Un objet chaîne peut prendre une des valeurs contenue dans une liste de valeur 'value1', 'value2', ..., ou NULL . Une ENUM peut avoir un maximum de 65535 valeurs distinctes.

***SET('value1','value2',...)*** Un ensemble. Un objet chaîne peut prendre une ou plusieurs valeurs, chacune de ces valeurs devant être contenue dans une liste de valeurs 'value1', 'value2', .... Un SET peut prendre jusqu'à 64 éléments.

| **Type de colonne** | **Taille mémoire** |
|---|---|
| TINYINT | 1 octet |
| SMALLINT | 2 octets |
| MEDIUMINT | 3 octets |
| INT | 4 octets |
| INTEGER | 4 octets |
| BIGINT | 8 octets |
| FLOAT(4) | 4 octets |
| FLOAT(8) | 8 octets |
| FLOAT | 4 octets |
| DOUBLE | 8 octets |
| DOUBLE PRECISION | 8 octets |
| REAL | 8 octets |
| DECIMAL(M,D) | M octets (D+2, si M < D) |
| NUMERIC(M,D) | M octets (D+2, si M < D) |
| DATETIME | 8 octets |
| DATE | 3 octets |
| TIMESTAMP | 4 octets |
| TIME | 3 octets |
| YEAR | 1 octet |
| CHAR(M) | M octets, 1 <= M <= 255 |
| VARCHAR(M) | L+1 octets, avec L <= M et 1 <= M <= 255 |
| TINYBLOB, TINYTEXT | L+1 octets, avec L < 2^8 |
| BLOB, TEXT | L+2 octets, avec L < 2^16 |
| MEDIUMBLOB, MEDIUMTEXT | L+3 octets, avec L < 2^24 |
| LONGBLOB, LONGTEXT | L+4 octets, avec L < 2^32 |
| ENUM('value1','value2',...) | 1 à 2 octets, suivant le nombre de valeurs dans l'énumération (65535 valeurs au maximum) |
| SET('value1','value2',...) | 1, 2, 3, 4 ou 8 octets, suivant le nombre de valeurs dans l'ensemble (64 valeurs au maximum) |

### 2.3.2.2 Modification d'une table

Syntaxe : **ALTER** [IGNORE] **TABLE** Nom_tbl **alter_spec** [**, alter_spec ...**] **:** modifie la structure d'une table existante.

**alter_spec** peut avoir les définitions suivantes :

***-> ADD [COLUMN] Col1 [FIRST | AFTER Col2 ]***

***-> ADD INDEX [Nom_index] (Col,...)***

***-> ADD PRIMARY KEY** (Col,...)*

***-> ADD UNIQUE** [Nom_index] (Col,...)*

***-> ALTER** [**COLUMN**] Col {SET DEFAULT literal | DROP DEFAULT}* spécifie une nouvelle valeur par défaut, ou bien efface l'ancienne valeur

***-> CHANGE** [**COLUMN**] Ancien_Nom_Col Nouv_Nom_Col Type:* permet de changer le nom/type d'une colonne

***-> MODIFY** [**COLUMN**] Col*

***-> DROP** [COLUMN] Col*

***-> DROP PRIMARY KEY :*** efface la colonne qui porte l'attribut `PRIMARY KEY` . Si une telle colonne n'existe pas, la première colonne de type `UNIQUE` est effacée à la place.

***-> DROP INDEX** Nom_index :* efface un index.

***-> RENAME** [AS] Nouveau_Nom_table* : permet de renommer la table

***-> table_option***

Pour pouvoir utiliser la commande `ALTER TABLE`, il faut avoir les droits pour **select**, **insert**, **delete**, **update**, **create** et **drop** sur la table.

Pour changer le type de la colonne mais pas son nom, la syntaxe de `CHANGE` requiert deux noms de colonnes, même si ils sont identiques. Par exemple : *mysql> ALTER TABLE t1 CHANGE Col Col Type NOT NULL;* Sinon utiliser MODIFY : *mysql> ALTER TABLE t1 MODIFY Col Type NOT NULL;*

Ajoute une nouvelle colonne Col, de type `Type.` : *mysql> ALTER TABLE t2 ADD Col Type;*

Ajoute un index sur la colonne Col1, et fait de la colonne Col2 une `PRIMARY KEY` : *mysql> ALTER TABLE t2 ADD INDEX (Col1), ADD PRIMARY KEY (Col2);*

Efface la colonne Col: *mysql> ALTER TABLE t2 DROP COLUMN Col;*

#### 2.3.2.3 Référence à une colonne

| Référence à une colonne | Signification |
|---|---|
| `Nom_col` | La colonne Nom_col de la table courante de la base de données courante. |
| `Nom_tab.Nom_col` | La colonne Nom_col de la table Nom_tab de la base de données courante. |
| `Nom_base.Nom_tab.Nom_col` | La colonne Nom_col de la table Nom_tab de la base de données `Nom_base`. |

#### 2.3.2.4 Indexation d'une colonne

Avec **MySQL**, tous les types de colonnes peuvent être indexés, à l'exception des types `BLOB` et `TEXT`. L'utilisation d'index est le meilleur moyen d'accélérer les performances des clauses `SELECT`.

Une table peut avoir jusqu'à 16 index.

Il n'est pas possible d'indexer une colonne contenant des valeurs `NULL`. Une colonne indexée doit être déclarée `NOT NULL`.

Pour les colonnes de type `CHAR` et `VARCHAR`, il est possible de préfixer la colonne. C'est un moyen beaucoup plus rapide et qui requiert moins d'espace disque qu'indexer une colonne complète. La syntaxe pour créer une telle colonne est la suivante :

```
KEY Nom_index(Nom_col(longueur))
```

L'exemple suivant créer un index pour les 10 premiers caractères de la colonne Nom_col.

```
mysql> CREATE TABLE test (
           nom  CHAR(200) NOT NULL,
           KEY Nom_index(nom(10)));
```

**MySQL** peut créer des indexes sur plusieurs colonnes en même temps. Un index peut contenir jusqu'à 15 colonnes

```
mysql> CREATE TABLE test ( id INT NOT NULL,
                           nom CHAR(30) NOT NULL,
                           prenom CHAR(30) NOT NULL,
                           PRIMARY KEY (id),
                           INDEX nom_complet (nom(15), prenom(15)));
```

Ainsi, l'index nom_complet est un index sur les deux colonnes nom et prénom. L'index sera utilisé lors des requêtes qui recherchent un nom, ou un nom et un prénom.

**CREATE** [**UNIQUE**] **INDEX** Nom_index **ON** Nom_table (Nom_col[(longueur)],... ) : raccourci de `ALTER TABLE` qui crée des index. Une liste de nom de colonne de format `(col1,col2,...)` créer un index de multiples colonnes. Les index sont formés en concaténant les différentes valeurs en une ligne.

**DROP INDEX** Nom_index : Supprime l'indexation

### 2.3.3 Contraintes d'intégrité

Les contraintes d'intégrité servent à assurer la précision et la logique des données dans une base de données relationnelle. L'intégrité est gérée par le concept d'intégrité référentielle (IR).

Il est nécessaire de nommée une contrainte par exemple pour la supprimer.

#### 2.3.3.1 Contraintes sur la clé primaire

PRIMARY KEY : La clé primaire identifie une ou plusieurs colonnes d'une table pour créer des lignes de données uniques. L'objectif est que chaque enregistrement ait une clé primaire ou valeur unique comme numéro d'identification.
Une clé primaire peut être composée de plusieurs colonnes.

```
ALTER TABLE Nom_Table ADD PRIMARY KEY (Col1, Col2…);
```

#### 2.3.3.2 Contraintes uniques

Une contrainte de colonne unique est comparable à une clé primaire dans la mesure où la valeur située dans cette colonne doit être unique pour chaque ligne de données de la table. Alors que la contrainte sur la clé primaire est placée dans une colonne, vous pouvez placer une contrainte unique sur une autre colonne même si celle-ci ne fait pas office de clé primaire

#### 2.3.3.3 Contraintes sur la clé extérieure

Une clé extérieure (foreign key) est une colonne que l'on place dans une table enfant pour référencer une clé primaire de la table parent. Ce type de contrainte sur clé extérieure est le principal mécanisme utilisé pour assurer l'intégrité référentielle entre les tables d'une base de données relationnelle. La colonne définie comme clé extérieure sert à référencer une colonne définie comme clé primaire dans une autre table.

```
ALTER TABLE Nom_Table_Enfant ADD CONSTRAINT Nom_Contrainte FOREIGN KEY (ColE) REFERENCES
Nom_Table_Reference (ColR);
```

La colonne ColE fait référence à la colonne ColR. Donc :

- pour qu'une ligne de Nom_Table_Enfant existe, elle doit avoir dans ColE une valeur que possède Nom_Table_Reference.ColR
- pour supprimer une valeur Nom_Table_Reference.ColR, toutes les valeurs ColE correspondantes doivent d'abord être supprimées de la table enfant.

#### 2.3.3.4 Contraintes NOT NULL

Permet d'interdire une saisie de valeurs NULL dans la colonne. Il est donc obligatoire de saisir des données pour chaque ligne d'une colonne NOT NULL.

#### 2.3.3.5 Contraintes de vérification (CHK)

Les contraintes de vérification (check) servent à vérifier la validité des données saisies dans des colonnes de table.

```
CONSTRAINT Nom_Contrainte CHECK (Colonne ='46000');
CONSTRAINT Nom_Contrainte CHECK ( Colonne in ('46100','46200','46700') );
CONSTRAINT Nom_Contrainte CHECK (Colonne > 75.00 ) );
```

## 3 Des commandes SQL sur MySQL

### 3.1 Obtenir des informations

**SHOW DATABASES** [**LIKE** wild] : fait la liste des bases de donnés sur le serveur **MySQL. Wild =** `%' ou '_'
**SHOW TABLES** [**FROM** Nom_bdd] [**LIKE** wild] : fait la liste des tables sur une base de données.
**SHOW COLUMNS FROM** Nom_table [**FROM** Nom_bdd] [**LIKE** wild] : fait la liste des colonnes d'une table. `SHOW FIELDS` en est un synonyme.
**SHOW INDEX FROM** Nom_table [**FROM** Nom_bdd] : `SHOW KEYS` est un synonyme de `SHOW INDEX`
**SHOW STATUS** : fournit des informations sur le serveur
**SHOW VARIABLES** [**LIKE** wild] : affiche quelques variables système de **MySQL**
**SHOW PROCESSLIST** : affiche la liste des process en ligne
**SHOW TABLE STATUS** [**FROM** Nom_bdd] [**LIKE** wild]

{**DESCRIBE** | **DESC**} Nom_table {Nom_col | wild} : fournit des informations à propos des colonnes d'une table

### 3.2 Ajouts et suppression de droits aux utilisateurs par les administrateurs

**GRANT** priv_type [(column_list)] [, priv_type [(column_list)] ...]
**ON** {Nom_table | * | *.* | Nom_bdd.*}
**TO** user_nom [**IDENTIFIED BY** 'password'] [, user_nom [**IDENTIFIED BY** 'password'] ...]
[**REQUIRE** [{**SSL**| **X509**}] [**CIPHER** cipher [**AND**]] [**ISSUER** issuer [**AND**]] [**SUBJECT** subject]]
[**WITH GRANT OPTION**]

**REVOKE** priv_type [(column_list)] [, priv_type [(column_list)] ...]
**ON** {Nom_table | * | *.* | Nom_table.*}
**FROM** user_nom [, user_nom ...]

`priv_type` peut avoir une des valeurs suivantes :
`ALL PRIVILEGES(ALL), ALTER, CREATE, DELETE, DROP, FILE, GRANT, INDEX, INSERT, PROCESS, REFERENCES, RELOAD, SELECT, SHUTDOWN, UPDATE, USAGE(aucun droit)`
La clause `WITH GRANT OPTION` donne à l'utilisateur la capacité de donner à d'autres utilisateurs des droits, d'un niveau égal à ceux qu'il a déjà.
Avec ces 2 requêtes, les droits sont immédiatement considérés (nul besoin de flush privileges).

### 3.3 Obtention de contenus de tables

La requête `SELECT` est utilisée pour obtenir des lignes à partir d'une ou plusieurs tables. Toutes les options doivent impérativement être dans l'ordre indiqué ci dessus.

**SELECT** [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [DISTINCT | DISTINCTROW | ALL]
select_expression,... : indique les colonnes à lire.
[**INTO OUTFILE** 'Nom_fichier' export_options] : écrit les lignes sélectionnées dans un fichier sur le serveur qui ne doit pas exister (cf. §3.9).
[**FROM** table_references : indique les noms des tables qu'il faut interroger. Si il y a plusieurs tables, il vaut mieux utiliser la clause `join`.
[**WHERE** where_definition] : condition pour qu'une ligne soit sélectionnée (=,<,>…) voir ci dessous
[**GROUP BY** Nom_col,...] : Doit précéder ORDER BY. Classe des données identiques par groupe
[**HAVING** condition] : Indique à GroupBy le groupe à inclure. Identique à Where de Select
[**ORDER BY** {unsigned_integer | Nom_col | formule} [ASC | DESC] ,...] : indique l'ordre ASCendant ou DESCendant pour les colonnes voulues en sortie.
[**LIMIT** [offset,] rows] :retourne les lignes comprises entre (offset+1) à (offset+rows)
[PROCEDURE procedure_name] ]

### 3.4 Clause Where

Permet de donner des critères de sélection en utilisant divers opérateurs tels que **=, <, >, <>, <=, >=, !=, BETWEEN, LIKE** et **NOT**.
Exemple : **SELECT** mysql.user.User, mysql.user.Host, mysql.user.Password **FROM** mysql.user **WHERE** User='root' **AND** Host='localhost';
**SELECT** mysql.user.User, mysql.user.Host, mysql.user.Password **FROM** mysql.user **GROUP BY** Host, User order by Host, User **ASC**;
**DELETE FROM** mysql.user **WHERE** User='admin_bd' **AND** Host!='localhost';

### 3.5 Modifier des références (lignes)

```
UPDATE [LOW_PRIORITY] Nom_table SET Nom_col1=expr1,Nom_col2=expr2,...
        [WHERE where_definition]
```

**UPDATE** met à jour une ligne existante dans une table. La clause **SET** indique quelles colonnes modifier, et quelles valeurs mettre dans ces colonnes. La condition **WHERE** permet de choisir quelles lignes sont à mettre à jour. Sinon, toutes les lignes sont mises à jour.
L'option **LOW_PRIORITY**, permet de retarder l'exécution de la requête jusqu'au moment où il n'y a plus de client qui lisent la table.
Lors de l'accès à une colonne de la table `Nom_table` dans une expression, **UPDATE** utilise la valeur courante de la colonne. Par exemple, la requête suivante ajoute 1 à la colonne `age`.

```
mysql> UPDATE persondata SET age=age+1;
```

Les commandes `UPDATE` sont évaluées de gauche à droite. Par exemple, la requête suivante double la colonne `age`, puis l'incrémente d'une unité :

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

Affecter la valeur courante d'une colonne lors d'une commande `UPDATE` conduit **MySQL** à ignorer cette mise à jour.
La commande `UPDATE` retourne le nombre de lignes qui ont été effectivement modifiées.

### 3.6 Insérer une référence (ligne)

**INSERT** [**LOW_PRIORITY | DELAYED**] [**IGNORE**] [**INTO**] tbl_nom [(col_nom,...)] **VALUES** (expression,...), (...),...
**INSERT** [**LOW_PRIORITY | DELAYED**] [**IGNORE**] [**INTO**] tbl_nom [(col_nom,...)] **SELECT** ...
**INSERT** [**LOW_PRIORITY | DELAYED**] [**IGNORE**] [**INTO**] tbl_nom **SET** col_nom=expression, col_nom=expression, ...
Insert une ligne dans une table existante. La deuxième écriture insert des lignes dans une table à partir d'une ou plusieurs autres tables sélectionnées. On utilisera DELAYED pour ne pas attendre la fin de l'insertion.

### 3.7 Remplacer une référence (ligne)

**REPLACE** [**LOW_PRIORITY | DELAYED**] [**INTO**] Nom_table [(Nom_col,...)] **VALUES** (expression,...)
**REPLACE** [**LOW_PRIORITY | DELAYED**] [**INTO**] Nom_table [(Nom_col,...)] **SELECT** ...
**REPLACE** [**LOW_PRIORITY | DELAYED**] [**INTO**] Nom_table **SET** Nom_col=expression, Nom_col=expression,...

`REPLACE` fonctionne exactement comme `INSERT`, mais si une ligne d'une table a une colonne qui porte l'attribut unique , `REPLACE` pourra remplacer cette ligne par une nouvelle ligne, tout en gardant la même valeur dans la colonne de type unique.

### 3.8 Supprimer des références (lignes)

**DELETE** [**LOW_PRIORITY**] **FROM** Nom_table [**WHERE** where_definition] [**LIMIT** rows] : retourne le nombre lignes effacées.
**!!!Sans clause WHERE, la commande s'applique à toutes les lignes.**

L'option **`LOW_PRIORITY`** permet de reporter l'effacement jusqu'au moment ou il ne reste plus personne qui lise la table.
L'option **`LIMIT`**, spécifique à **MySQL,** permet d'indiquer au serveur le nombre maximum de ligne à effacer. Cela permet d'éviter qu'une commande `DELETE` ne prenne trop de temps. Il suffit alors de répéter la commande jusqu'à ce qu'elle ait effacé moins de ligne que `LIMIT`.

### 3.9 Lire des lignes à partir d'un fichier, et transforme le contenu du fichier en table, à très grande vitesse.

**LOAD DATA** [LOCAL] **INFILE** 'Nom_fichier.txt' [**REPLACE | IGNORE**] : LOCAL pour indiquer que le fichier est sur le poste client. Avec `REPLACE`, la nouvelle ligne remplacera l'ancienne si elles ont la même valeur d'index. Avec `IGNORE`, la nouvelle ligne sera ignorée. Si rien n'est précisé, une erreur surviendra lors de la tentative d'insertion du doublon, et le reste du fichier sera ignoré.

**INTO TABLE** Nom_table
[**FIELDS**
{[**TERMINATED BY** '\t'] | :Les champs d'une ligne sont séparés grâce aux tabulations '\t' ou par une virgule ',' …
[**OPTIONALLY**] **ENCLOSED BY** ''] | :Ne pas Entourer les champs si ''. Insérés dans des doubles guillemets '"'. L'option OPTIONALLY force l'utilisation du caractère ENCLOSED BY seulement pour les champs de type CHAR et VARCHAR.
[**ESCAPED BY** '\\' ]}] :Indique le caractère d'échappement. Le caractère suivant '\' ne sera pas interpréter.
[**LINES TERMINATED BY** '\n'] :Les lignes sont séparées par nouvelle-ligne '\n', retour-chariot/nouvelle-ligne '\r\n' …
[**IGNORE** nombresde **LINES**] : permet d'ignorer les premières lignes, qui contiendrait un entête, par exemple.
[(Nom_col,...)] :indique l'ordre des colonnes de la table à respecter

### 3.10 Sélection des données dans plusieurs tables : Les jointures

Une des fonctionnalités les plus puissantes du langage SQL réside dans la capacité à sélectionner des données dans plusieurs tables. Sans elle, l'ensemble du concept de base de données relationnelle serait pure abstraction.
Les tables liées sont générées à l'aide de colonnes communes contenant les clés primaires.
Une jointure combine deux ou plusieurs tables afin d'en extraire des données.

#### 3.10.1 Types de jointures

**EQUIJOIN** (jointure d'égalité) encore appelée INNER JOIN (jointure interne)
**NATURAL JOIN** (jointure naturelle)
**NON-EQUIJOIN** (jointure de non égalité)
**OUTER JOIN** (jointures externe)
**SELF JOIN** (jointure réflexive)
L'instruction SELECT et la clause FROM sont toutes deux des éléments obligatoires d'une instruction SQL ; la clause WHERE est un élément obligatoire de la jointure de tables. Les tables à joindre sont répertoriées dans la clause FROM et la jointure s'accomplit dans la clause WHERE. Pour joindre des tables, il est possible d'utiliser divers opérateurs tels que =, <, >, <>, <=, >=, !=, BETWEEN, LIKE et NOT. Toutefois, l'opérateur le plus utilisé reste le signe égal.

#### 3.10.2 Jointures d'égalité

Syntaxe correspondant à la jointure **EQUIJOIN** :

**SELECT** TABLE1.COL1, TABLE2.COL2... retourne les valeurs des colonnes indiquées
**FROM** TABLE1, TABLE2 [, TABLE3 ] des tables indiquées
**WHERE** TABLE1.NOM_COL **=** TABLE2.NOM_COL ou cette condition et vrai
[ **AND** TABLE1.NOM_COL **=** TABLES.NOM_COL ] et celle-ci aussi

#### 3.10.3 Jointures naturelles

Une jointure **NATURAL JOIN** équivaut presque à une jointure EQUIJOIN, si ce n'est qu'elle permet d'éliminer les valeurs dupliquées dans les colonnes jointes. Il en va de même pour la condition JOIN, mais les colonnes sélectionnées diffèrent.
Syntaxe :

**SELECT** TABLE1.*, TABLE2.NOM_COL [ TABLE3.NOM_COL ] retourne toutes les colonnes de table1
**FROM** TABLE1, TABLE2 [ TABLE3 ]
**WHERE** TABLE1.NOM_COL = TABLE2.NOM_COL
[ **AND** TABLE1.NOM_COL = TABLE3.NOM_COL ]

#### 3.10.4 Jointures de non égalité

Les jointures de non égalité **NON-EQUIJOIN** joignent deux tables ou plus à partir d'une valeur de colonne spécifiée différente de la valeur de colonne spécifiée dans l'autre table. Voici la syntaxe de cette jointure :

**FROM** TABLE1, TABLE2 [, TABLE3 ]
**WHERE** TABLE1.NOM_COL **!=** TABLES.NOM_COL renvoie les références n'ayant pas de correspondance
[ **AND** TABLE1.NOM_COL **!=** TABLES.NOM_COL ]

### 1.1.1 Jointures externes (OUTER JOIN)

Retourner toutes les lignes d'une table, même si celles-ci ne possèdent pas de correspondance dans la table jointe.
Syntaxe la plus courante :
**FROM** TABLE 1 {RIGHT ; LEFT ; FULL} [OUTER] JOIN ON TABLE2

```
mysql> select table1.* from table1 LEFT JOIN table2 ON table1.id=table2.id where table2.id is NULL;
```

Cet exemple recherche toutes les lignes dans la `table1` avec une colonne `id` qui n'est pas présent dans la table `table2` (i.e., toutes les lignes de la table `table1` qui n'ont pas de ligne correspondante dans la table `table2`).

A LEFT JOIN B USING (C1,C2,C3,...) Correspond à l'utilisation d'une clause ON comme ceci :
`A.C1=B.C1 AND A.C2=B.C2 AND A.C3=B.C3,...`

### 1.1.2 Jointures réflexibles ou autojointures

La jointure réflexive SELF JOIN s'utilise pour joindre une table à elle-même, comme s'il existait deux tables en une et en renommant provisoirement au moins une table dans l'instruction SQL.
En voici la syntaxe :
**SELECT** A.NOM_COL, B.NOM_COL, [C.NOM_COL ] **FROM** TABLE1 A, TABLE1 B [, TABLE1 C ]
**WHERE** A.NOM_COL **=** B.NOM_COL [ **AND** A.NOM_COL **=** C.NOM_COL ]
Les jointures réflexives sont utiles lorsque toutes les données que vous souhaitez extraire résident dans une seule table.

## 3.11 Bloquer une table

**LOCK TABLES** Nom_tbl [**AS** alias] {**READ** | [**LOW_PRIORITY**] **WRITE**} [, Nom_tbl {**READ** | [**LOW_PRIORITY**] **WRITE**} ...]
...
**UNLOCK TABLES**

Si un thread obtient le verrou de lecture (`READ`) sur une table, le thread qui a le verrou est le seul à pouvoir lire ou écrire dans la table.
Les autres threads attendent (sans limite) que le verrou se libère.
Si un thread obtient un verrou de lecture, et qu'un autre thread obtient un verrou d'écriture, alors le thread au verrou de lecture devra attendre la libération du verrou d'écriture.

Aucun autre thread ne peut interférer avec la commande en cours d'exécution.
Les cas où il serait nécessaire de verrouiller une table :

- Si un grand nombre d'opérations vont être menées sur un bon nombre de tables, il est plus rapide de verrouiller les tables utilisées. L'inconvénient est qu'aucun autre thread ne pourra accéder aux informations, ni les modifier.
- **MySQL** ne supporte pas d'environnement transactionnel, il faut absolument verrouiller une table, pour s'assurer que d'autres thread n'interviennent entre une commande `SELECT` et une commande `UPDATE` .

```
mysql> LOCK TABLES table1 READ, table2 WRITE;
mysql> select sum(value) from table1 where table2_id= un_id;
mysql> update table2 set champy=sum_from_previous_statement
           where table2_id=un_id;
mysql> UNLOCK TABLES;
```

Sans la commande `LOCK TABLES`, il se peut qu'un autre thread insère une nouvelle ligne dans la table `table1` entre les deux commandes `SELECT` et `UPDATE` .

## 3.12 Optimisation d'une table

**`OPTIMIZE TABLE`** `Nom_table`
`OPTIMZE TABLE` est à utiliser après avoir effacer de grandes parties d'une table. Les index des lignes effacées sont conservés, et l'opération `INSERT` les réutilise, et écrase les anciennes lignes. La commande `OPTIMIZE TABLE` permet de récupérer l'espace inutilisé des tables.
`OPTIMIZE TABLE` fonctionne en effectuant une copie temporaire de la table originale. La modification est effectuée sur la copie, puis l'originale est remplacée par la copie modifiée. Toutes les mises à jours sont automatiquement appelées, et sont faites dans un mode sans erreur. Pendant la modification, la table originale est toujours accessible en lecture par les autres clients. Les mises à jour et les écritures sont reportées jusqu'à la fin de la modification.

## 3.13 Informations après un SELECT

**`EXPLAIN SELECT`** `select_options`
L'option `EXPLAIN` force **MySQL** à expliquer la façon avec laquelle il va traiter la requête `SELECT`, en détaillant les opérations de regroupement.

### 3.14 Ajout et suppression de fonction

```
CREATE FUNCTION function_name RETURNS {STRING|REAL|INTEGER}
       SONAME shared_library_name

DROP FUNCTION function_name
```

Une fonction définie par l'utilisateur est un bon moyen d'ajouter de nouvelles fonctionnalités à **MySQL** avec de nouvelles fonctions natives, comme par exemple `ABS()` et `CONCAT()`.
`CREATE FUNCTION` sauve le nom de la fonction, le type et le point d'entrée de la fonction dans la table système `mysql.func` . Il faut avoir les droits **insert** et **delete** pour pouvoir créer et effacer des fonctions.
Toutes les fonctions actives sont rechargées à chaque démarrage du serveur, à moins de lancer `mysqld` avec l'option `--skip-grant-tables` . Dans ce cas, l'initialisation des fonctions utilisateurs est oubliée, et les fonctions sont inutilisables. (Une fonction active est une fonction créée par `CREATE FUNCTION` et pas effacée avec `DROP FUNCTION`.)

## 4 Les opérateurs

Ils permettent de spécifier des conditions dans une instruction SQL.

### 4.1 Opérateurs de Comparaison

On y trouve : =, <>, < et >
On peut les combiner : <=, >=
On les utilise avec la clause "WHERE"
Exemple : `SELECT * FROM table WHERE colonne = '2004';`
Exemple : `SELECT * FROM table WHERE colonne <> '2003';`
Exemple : `SELECT * FROM table WHERE colonne > 49.99;`

### 4.2 Logique

#### 4.2.1 IS NULL

Compare avec une valeur de type "NULL". Cet opérateur teste le type et non une valeur égale à 'NULL'.
Exemple : `SELECT * FROM table WHERE colonne IS NULL;` renvoie un résultat différent de `WHERE colonne IS NULL`

#### 4.2.2 BETWEEN

Cherche des valeurs situées à l'intérieur d'une plage de valeurs dont les bornes sont incluses.
Exemple : `SELECT * FROM table WHERE colonne BETWEEN 49.99 AND 99.99;`

#### 4.2.3 IN

Compare une valeur à une liste de valeurs spécifiées et retourne TRUE si il y a une correspondance.
Exemple : `SELECT * FROM table WHERE colonne IN ('2004', '2000', '2002');`

#### 4.2.4 LIKE

Compare une valeur à des valeurs similaires avec des caractères génériques : '%' pour aucun ou plusieurs caractères et '_' pour un caractère unique.
Exemple : `SELECT * FROM table WHERE colonne LIKE 'DUP_';`
Ou `'%du%', '_up%', d_%_%'` pour commencer par 'd' avec au moins 3 caractères, `'_u%t', 'd____t', ...`

#### 4.2.5 EXISTS

Rechercher la présence d'une ligne répondant à certains critères. On utilisera ici des sous requêtes :
SELECT cols FROM tbls WHERE col opérateur (SELECT cols FROM tbls WHERE conditions);
Exemple : `SELECT * FROM tbl WHERE EXISTS (SELECT col FROM tbl WHERE COL > 49.9);`

#### 4.2.6 UNIQUE

Rechercher toutes les lignes uniques (sans doublon). On utilisera ici des sous requêtes.
Exemple : `SELECT * FROM tbl WHERE UNIQUE (SELECT col FROM tbl WHERE COL = 50);`

#### 4.2.7 ALL, ANY

ALL compare une valeur à toutes celles d'un jeu de valeurs. On utilisera ici des sous requêtes.
ANY compare une valeur à toutes les valeurs applicables d'une liste en fonction de la condition. On utilisera ici des sous requêtes.
Exemple : `SELECT * FROM tbl WHERE col > ALL (SELECT col FROM tbl WHERE COL = val);`

### 4.3 Autres

AND, OR,
NOT : NOT BETWEEN, NOT IN, NOT LIKE, IS NOT NULL, NOT EXISTS, NOT UNIQUE

### 4.4 Arithmétiques

+, -, *, /
Exemple : `SELECT COL * COL2 * 100 FROM tbl WHERE col3 < COL4*3;`

## 5 Fonctions MySQL

Elles peuvent être utilisées dans des requêtes SQL. Pour la plupart, elles constituent un ajout de MySQL à la norme SQL ANSI

ABS (nombre) : renvoie la valeur absolue
SIGN(nombre) : renvoie le signe de nombre
MOD( nb1, nb2) : renvoie nb1 modulo nb2
SQRT(nombre) : renvoie la racine carré
CEILING(nombre) : renvoie le plus petit entier >= à nombre.
FLOOR(nombre) : plus petit entier inférieur ou égal à nombre.
CONV(nombre, base1, base2) : renvoie la conversion de nombre de base1 en base2. (2<= base i <36)
BIN(decimal) : renvoie la valeur binaire d'un nombre décimal.
HEX(decimal) : renvoie la valeur hexadécimal d'un nombre décimal.
OCT(decimal) : renvoie la valeur octale d'un nombre décimal.
BIT_COUNT(nombre) : renvoie le nombre de bits à 1 dans la représentation binaire de nombre.
FORMAT(nombre, decimales) : formate un numérique arrondi avec un nombre de décimales.
GREATEST(nombre1, nombre2 [, nombre3, ...]) : renvoie le nombre le plus grand
LEAST(nombre1, nombre2 [, nombre3, ...]) : renvoie le nombre le plus petit
INTERVAL(val, v1, v2, v3...) : renvoie 0 si val est la plus petite valeur, 1 si v1<=val< v2, 2 si v2<=val< v3, etc.
RAND([generateur]) : renvoie un nombre aléatoire entre 0 et 1. Le générateur peut être spécifié.
ROUND(nombre [, nbdec]) : arrondit nombre à l'entier le plus proche ou au nombre de décimales donné.
TRUNCATE(nombre, nbdec) : tronque nombre (sans arrondi)
LOG(nombre) : Logarithme népérien
LOG10(nombre) : Logarithme base 10

COS(radians) : renvoie le cosinus
ACOS (nombre) : renvoie le cosinus inverse de nombre exprimé en radians
SIN(radians) : renvoie le sinus
ASIN (nombre) : renvoie le sinus inverse de nombre exprimé en radians
TAN (nombre) : renvoie la tangente de nombre exprimée en radians
ATAN (nombre) : renvoie la tangente inverse de nombre exprimée en radians
ATAN2 (x, y) : renvoie la tangente inverse du point x,y
COT(radians) : renvoie la cotangente
DEGREES(radians) : conversion de radians en degrés.
RADIANS(degre) : conversion de degrés en radians.
PI() : renvoie Π

ASCII (car) : renvoie le code ASCII
CHAR( code1, [code2, ...]) : renvoie une chaîne obtenue par conversion de chaque code ASCII vers le caractère correspondant.
CONCAT(chaine1, [chaine2, ...]) : renvoie la concaténation de tous les arguments.
ELT(nb, chain1, chain2, ...) : retourne la chaîne dont la position est nb ou NULL si la position n'est pas valide
ENCRYPT(chain [, clé]) : Crypte la chaîne, en utilisant la clé si elle est fournie
PASSWORD(chain) : cryptage de chain selon MySQL
FIELD(chain, chain1, chain2, ...) : renvoie la position de la première chaîne identique à chain parmi {chain1, chain2 ...}, 0 si rien
FIND_IN_SET(chain, ensembl) : Renvoie la position de chain dans ensembl.
INSERT(chain, pos, long, chain2) : renvoie une chaîne obtenue en remplaçant la sous-chaîne de chain de longueur long et commençant à pos par chain2
INSTR(chain, souschain) : renvoie la position de souschain dans chain.
LOCATE(souschain, chain [, pos]) : comme INSTR mais avec las arguments inversés et commence à partir de la position pos
LCASE(chain) : renvoie chain en minuscules
UCASE(chain) : renvoie chain en majuscules
LEFT(chain, nb) : renvoie les nb premiers caractères de chain
RIGHT(chain, nb) : renvoie les nb derniers caractères de chain
LENGTH(chain) : renvoie la longueur de chain
LPAD(chain, long, motif) : renvoie chain complétée à gauche de motif pour que la longueur totale soit long.
RPAD(chain, long, motif) : renvoie chain complétée à droite de motif pour que la longueur totale soit long.
LTRIM(chain) : retire tous les caractères blancs au début de chain
RTRIM(chain) : retire tous les caractères blancs en fin de chain
TRIM([BOTH | LEADING | TRAILING] [car] [FROM] chain) : retire le caractère blanc ou car en début OU|ET en fin de chain
MID(chain, pos, long), SUBSTRING(chain, pos, long) : renvoie la sous-chaîne de longueur long de chain, débutant de pos

REPEAT(chain, n) : renvoie une chaîne constituée de n fois chain
REPLACE(chain, nouveau, ancien) : renvoie une chaîne où toutes les occurrences de ancien sont remplacées par nouveau
REVERSE(chain) : renvoie la chaîne miroir de chain
SPACE(nb); renvoie une chaîne avec nb caractères blancs
STRCMP(chain1, chain2) : renvoie 0 si les chaînes sont identiques, -1 si chain1 est avant chain2 (a avant b), sinon 1.
SUBSTRING_INDEX(chain, car, n) : renvoie sous-chaîne obtenue en comptant n * car dans chain, et en prenant tout ce qui est à gauche si n<0 ou à droite sinon.

CURDATE() : renvoie la date courante
CURTIME() : renvoie l'heure courante
DATE_ADD(date, INTERVAL nb période) : Ajoute nb*période à date. Ex : DATE_ADD(CURDATE(), INTERVAL 40 DAY). Période : SECOND, MINUTE, HOUR, DAY, MONTH ou YEAR.
DATE_SUB (date INTERVAL nb période) : soustrait une durée
DAYNAME(date) : nom du jour en anglais.
DOYOFMONTH (date) : numéro du jour dans le mois.
DAYOFWEEK(date) : numéro du jour dans la semaine.
DAYOFYEAR (date) : numéro du jour dans l'année.
FROM_DAYS(jours) : renvoie la date correspondant à jours. Ex : select from_days(732570) => 2005-09-16
FROM_UNIXTIME(seconde [,format]) : renvoie la date GMT correspondant au nombre de seconde depuis le 01/01/1970
HOUR(temps) : renvoie l'heure de temps
MINUTE(temps) : renvoie les minutes de temps
SECOND(temps) : renvoie les secondes de temps
WEEK(date) : renvoie le numéro de la semaine de date
YEAR(date) : renvoie l'année de date
MONTH(date) : renvoie le numéro du mois de date
MONTHNAME(data) : renvoie le nom de mois de date en anglais
NOW() : renvoie la date et l'heure courante
PERIOD_ADD(date, nbmois) : ajoute nbmois à date qui doit être au format AAMM ou AAAMM
QUARTER(date) : renvoie le numéro du trimestre de date
SEC_TO_TIME(secondes) : convertit des secondes en heures, minutes et secondes.
TIME_TO_SEC(temps) : renvoie le nombre de secondes dans temps.
TIME_FORMAT(temps, format) : formate temps selon format (voir DATE_FORMAT)
TO_DAYS(date) : renvoie le nombre de jours entre le 1/1/1 et date
UNIX_TIMESTAMP([date]) : renvoie le nombre de secondes entre le 1/1/1970GMT et date (si non précisée alors date courante)
DATE_FORMAT(date, format) : formate la date selon le format :

- '%a', '%W' : nom court ou complet du jour en anglais.
- '%b', '%M' : nom court ou complet du mois en anglais.
- '%d', '%D': jour du mois sans ou avec suffixe ($1^{st}$, $2^{nd}$)
- '%h', '%H' : heure sur 12 ou 24 heures (2 chiffres)
- '%l', '%k' : heure sur 12 ou 24 heures (1 | 2 chiffres)
- '%i' : minutes
- '%j' : jour de l'année
- '%m' : numéros du mois
- '%p' : AM ou PM
- '%r', '%T' : heure complète (hh:mm:ss) sur 12 heures avec AM | PM ou 24 heures
- '%S', '%s': secondes sur 2 chiffres voir 1 ou 2 chiffres
- '%U' : numéro de la semaine
- '%w' : numéro du jour de la semaine (0:Dimanche)
- '%Y', '%y' :année sur 4 ou 2 chiffres
- '%%' : affiche % Ex : select DATE_FORMAT(CURDATE(), '%a %d %b %Y'); ou '%W %d %M %Y'

IF(test, val1, va2) : si test est vrai alors val1 sinon val2
IFNULL(valeur, valeur2) : si valeur n'est pas NULL alors valeur sinon valeur2
ISNULL(expression) : renvoie vrai (1) si expression est à NULL, sinon 0

## 6 Synthèses des résultats des requêtes

### 6.1 La fonction COUNT

Comptage du nombre de lignes ou de valeurs d'une colonne ne contenant pas de valeur NULL.
Syntaxe : `COUNT ( [ *) | [DISTINCT | ALL (nom_colonne) ] )` ou DISTINCT permet de compter les lignes différentes. ALL par défaut. Avec '*' toutes lignes comptées même si la valeur d'une colonne vaut NULL

### 6.2 La fonction SUM

Retourne le total des valeurs d'une colonne pour un groupe de lignes.
Syntaxe : `SUM [( DISTINCT] (nom_colonne) [)]`

### 6.3 La fonction AVG

Retourne la moyenne des valeurs d'une colonne pour un groupe de lignes.
Syntaxe : `AVG [( DISTINCT] (nom_colonne) [)]`

### 6.4 La fonction MAX

Retourne la valeur maximale d'une colonne pour un groupe de lignes.
Syntaxe : `MAX (nom_colonne)`

### 6.5 La fonction MIN

Retourne la valeur minimale d'une colonne pour un groupe de lignes.
Syntaxe : `MIN (nom_colonne)`

## 7 Versions et fonctionnalités

| *Fonctionnalités \ version* | *MySQL 3.23* | *MySQL 4.1* | *MySQL 5.0* | *MySQL 5.1* |
|---|---|---|---|---|
| Clés étrangères/ Intégrité référentielle | Avec le moteur innoDB | Avec le moteur innoDB | Avec le moteur innoDB | Avec le moteur innoDB |
| Requêtes imbriquées | | oui | oui | oui |
| Vues | | | oui | oui |
| Procédures stockées | | | oui | oui |
| Déclencheurs | | | limité | oui |

### 7.1 Les Vues

Une vues est une table virtuelle. Les données de la vue sont des champs de différentes tables regroupées, ou des résultats d'opérations sur ces champs. Elles sont utiles pour donner aux utilisateurs l'accès à un ensemble de relations représentées sous la forme d'une table.

Pour l'insertion et la modification de données dans une vue, ce n'est pas aussi simple que la mise à jour d'une table. Pour cette faisabilité, il faut l'envisager au niveau du MCD. Pour cela il faut que:

- Tous les champs des tables possédant des contraintes d'intégrités (index unique, clés primaires ...) doivent être présents.
- La vue ne doit pas posséder de regroupement ou d'exclusion (GROUP BY, DISTINCT, UNION)
- Le plan d'exécution de la vue ne doit pas passer par une table temporaire.

Syntaxe générale : `CREATE VIEW nom_de_la_vue [(liste_colonnes)] AS criteres_de_selection`
Syntaxe pour une table :
```
CREATE VIEW nom_de_la_vue
      AS SELECT * | col1 [,col2] FROM nom_table [criteres_selection]
```

Créer une vue revient à appliquer un filtre à une ou plusieurs tables. Ex :
```
CREATE VIEW etudiantrsc AS SELECT etudiant.nom AS nom, etudiant.prenom AS prenom,
section.nom AS section FROM etudiant,section WHERE filtrer_ici
```

### 7.2 Les procédures stockées

Ce sont des listes de commandes qui peuvent être compilées et stockées sur le serveur. Le client fera référence à la procédure stockée plutôt que de soumettre la commande complète.
Cela se traduit par de meilleures performances car

- les commandes sont déjà analysées et se trouvent dans un format exécutable.
- Il y a moins de trafic réseau

A partir de la version 5, elles respectent les recommandations du standard SQL2003.
Syntaxe générale : `CREATE PROCEDURE nom_procedure ([parametres]) [caracteristiques] routines`

### 7.3 Les déclencheurs

Ou triggers, sont des ordres de déclenchement d'opérations quand un évènement survient sur une table.
Il sont souvent utilisés pour assurer la cohérence des données dans la base, en réalisant des contraintes qui doivent porter sur plusieurs tables.
Syntaxe :
```
CREATE TRIGGER nom_trigger {BEFORE|AFTER} {INSERT|UPDATE|DELETE}
        ON table
        FOR EACH ROW ce_qu'il_faut_faire
```
Le premier élément entrant en compte est le nom du trigger.
Le second paramètre (before ou after)indique si le déclencheur doit être activé avant ou après l'évènement.
Un déclencheur peut être activé durant l'appel à un INSERT, UPDATE ou DELETE.
Ils sont lié à une table
Une fois activé, on définit les instructions à effectuer

## 8 Récapitulatif des requêtes usuelles

**CREATE DATABASE** Nom_bdd ;
**DROP DATABASE** [**IF EXISTS**] Nom_bdd;

**CREATE TABLE** [**IF NOT EXISTS**] Nom_table (**create_definition**,...) [**opts_de_table**] [cmd_de_selection]
**-> create_definition** peut avoir les définitions suivantes :
***->*** *Nom_col* ***type*** *[NOT NULL | NULL] [DEFAULT valeur_par_defaut] [AUTO_INCREMENT] [PRIMARY KEY] [reference_definition]*
***-> PRIMARY KEY*** *(index_Nom_col,...)*
***-> KEY*** *[Nom_index] KEY(index_Nom_col,...)* identique à ***INDEX*** *[Nom_index] (index_Nom_col,...)*
***-> UNIQUE*** *[INDEX] [Nom_index] (index_Nom_col,...)*
***-> [CONSTRAINT*** *symbole]* ***FOREIGN KEY*** *Nom_index(index_Nom_col,...) [reference_definition]*
***-> CHECK*** *(expression )*

**ALTER** [**IGNORE**] **TABLE** Nom_table **alter_spec** [, **alter_spec** ...]
**alter_spec** peut avoir les définitions suivantes :
***-> ADD [COLUMN]*** *Col1 [FIRST | AFTER Col2 ]*
***-> ADD INDEX*** *[Nom_index] (Col,...)*
***-> ADD PRIMARY KEY*** *(Col,...)*
***-> ADD UNIQUE*** *[Nom_index] (Col,...)*
***-> ALTER [COLUMN]*** *Col {SET DEFAULT literal | DROP DEFAULT}* spécifie une nouvelle valeur par défaut, ou bien efface l'ancienne valeur
***-> CHANGE [COLUMN]*** *Ancien_Nom_Col Nouveau_Nom_Col Type:* permet de changer le nom d'une colonne. Voir le type
***-> MODIFY [COLUMN]*** *Col*
***-> DROP*** *[COLUMN] Col*
***-> DROP PRIMARY KEY*** *:* efface la colonne portant l'attribut PRIMARY KEY . Si elle n'existe pas, la première colonne de type UNIQUE est effacée à la place.
***-> DROP INDEX*** *Nom_index :* efface un index.
***-> RENAME*** *[AS] Nouveau_Nom_table* : permet de renommer la table
***-> table_option***

**CREATE** [**UNIQUE**] **INDEX** Nom_index **ON** Nom_table (Nom_col[(longueur]),... )
**DROP INDEX** Nom_index

**GRANT** priv_type [(column_list)] [, priv_type [(column_list)] ...]
**ON** {Nom_table | * | *.* | Nom_bdd.*}
**TO** user_name [IDENTIFIED BY 'password'] [, user_name [IDENTIFIED BY 'password'] ...] [**REQUIRE** [{**SSL**| **X509**}] [**CIPHER** cipher [**AND**]]
[**ISSUER** issuer [**AND**]]
[**SUBJECT** subject]]
[**WITH GRANT OPTION**]

**REVOKE** priv_type [(column_list)] [, priv_type [(column_list)] ...]
**ON** {Nom_table | * | *.* | Nom_table.*}
**FROM** user_name [, user_name ...]

**SELECT** [**STRAIGHT_JOIN**] [**SQL_SMALL_RESULT**] [**DISTINCT | DISTINCTROW | ALL**] Nom_col1,Nom_col2,...
[**INTO OUTFILE** 'Nom_fichier' export_options]
[**FROM** table_references
[**WHERE** where_definition]
[**GROUP BY** Nom_col,...]
[**HAVING** condition]
[**ORDER BY** {unsigned_integer | Nom_col | formule} [ASC | DESC] ,...]
[**LIMIT** [offset,] rows]
[**PROCEDURE** procedure_name] ]

**INSERT** [**LOW_PRIORITY | DELAYED**] [**IGNORE**] [**INTO**] tbl_nom [(col_nom,...)] **VALUES** (expression,...), (...),...
**INSERT** [**LOW_PRIORITY | DELAYED**] [**IGNORE**] [**INTO**] tbl_nom [(col_nom,...)] **SELECT** ...
**INSERT** [**LOW_PRIORITY | DELAYED**] [**IGNORE**] [**INTO**] tbl_nom **SET** col_nom=expression, col_nom=expression, ...

**UPDATE** [**LOW_PRIORITY**] Nom_table **SET** col1=expr1,col2=expr2,... [**WHERE** definition]

**REPLACE** [**LOW_PRIORITY | DELAYED**] [**INTO**] Nom_table [(Nom_col,...)] **VALUES** (expression,...)
**REPLACE** [**LOW_PRIORITY | DELAYED**] [**INTO**] Nom_table [(Nom_col,...)] **SELECT** ...
**REPLACE** [**LOW_PRIORITY | DELAYED**] [**INTO**] Nom_table **SET** Nom_col=expression, Nom_col=expression,...

**DELETE** [**LOW_PRIORITY**] **FROM** Nom_table [**WHERE** where_definition] [**LIMIT** rows]

**LOAD DATA** [LOCAL] **INFILE** 'Nom_fichier.txt' [**REPLACE | IGNORE**] **INTO TABLE** Nom_table
[**FIELDS** {[**TERMINATED BY** '\t'] | [**OPTIONALLY**] **ENCLOSED BY** ''] | [**ESCAPED BY** '\\' ]}]
[**LINES TERMINATED BY** '\n']
[**IGNORE** nombresde **LINES**]
[(Nom_col,...)]

**LOCK TABLES** Nom_table [**AS** alias] {**READ** | [**LOW_PRIORITY**] **WRITE**} [, Nom_table {**READ** | [**LOW_PRIORITY**] **WRITE**} ...]
...
**UNLOCK TABLES**