

# **ADMINISTRATION**

# **Bases de données**

# **MYSQL**

**LICENCE PROFESSIONNELLE D'INFORMATIQUE**

Par

**Edouard Ngor SARR**

**Enseignant à UCAO- SAINT MICHEL**

## SOMMAIRE

PARTIE 1 : INTRODUCTION et INSTALLATION

PARTIE 2 : LES MODES DE CONNEXION, DE DECONNEXION, DEMARRAGE ET ARRET

PARTIE 3 : CREATION BASE DE DONNEES ET ORDRES SQL

PARTIE 4 : GESTION DES USERS ET DES PRIVILEGES

PARTIE 5 : LES TRANSACTION

PARTIE 6 : LES VEROUS

PARTIE 7 : LES TRIGGERS

PARTIE 8 : CHARGEMENT DE DONNEES EXTERNES AVEC LOAD

PARTIE 9 : SAUVEGARDE ET RESTAURATION (IMPORT/ EXPORT)

PARTIE 10 : INFORMATION\_SCHEMA

PARTIE 11 : PRATIQUE SUR PHPMYADMIN

PARTIE 12 : PRATIQUE SUR Mysql Workbench

**PARTIE 1 :**  
**INTRODUCTION et INSTALLATION**

## 1. INTRODUCTION

### **BASE DE DONNEES :**

Une base de données informatique est un ensemble de données qui ont été stockées sur un support informatique, et organisées et structurées de manière à pouvoir faciliter l'exploitation (ajout, mise à jour, recherche de données). Elle se traduit physiquement par un ensemble de fichiers sur disque.

Une base de données permet donc de Classer, trier et filtrer de larges quantités d'informations structurées concernant un sujet donné et gérées par un SGBD.

Une base de données seule ne suffit donc pas, il est nécessaire d'avoir également : un système permettant de gérer cette base ; un langage pour transmettre des instructions à la base de données (par l'intermédiaire du système de gestion).

Mais une base de données seule ne suffit donc pas, il est nécessaire d'avoir également :

- un système permettant de gérer cette base : SGBD
- un langage pour transmettre des instructions à la base de données (par l'intermédiaire du système de gestion). SQL

### **Le SGBD :**

Un Système de Gestion de Base de Données (SGBD) est un logiciel (ou un ensemble de logiciels) permettant de manipuler les données d'une base de données. Manipuler, c'est-à-dire sélectionner et afficher des informations tirées de cette base, modifier des données, en ajouter ou en supprimer (ce groupe de quatre opérations étant souvent appelé "CRUD", pour Create, Read, Update, Delete). MySQL est un système de gestion de bases de données.

C'est-à-dire un logiciel qui permet de gérer des bases de données, et donc de gérer de grosses quantités d'informations. Il utilise pour cela le langage SQL.

Il s'agit d'un des SGBDR les plus connus et les plus utilisés.

MySQL peut donc s'utiliser seul, mais est la plupart du temps combiné à un autre langage de programmation : PHP par

Cette capacité à gérer ces différentes relations porte le nom de Système de Gestion de Base de Données Relationnelle (SGBDR).

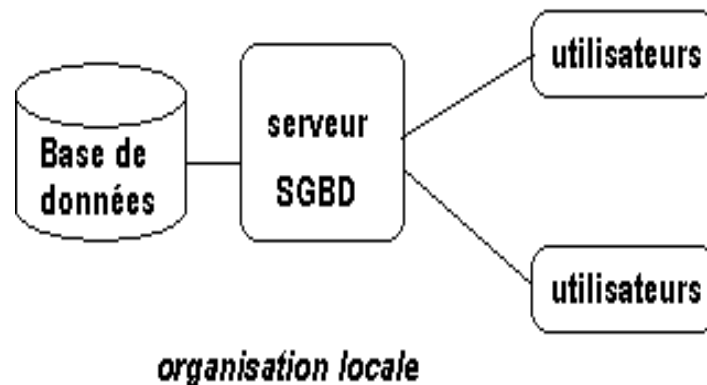
Un SGBDR est un SGBD qui implémente la théorie relationnelle. MySQL implémente la théorie relationnelle ; c'est donc un SGBDR.

Dans un SGBDR, les données sont contenues dans ce qu'on appelle des relations, qui sont représentées sous forme de tables. Une relation est composée de deux parties, l'en-tête et le corps. L'en-tête est lui-même composé de plusieurs attributs. Quant au corps, il s'agit d'un ensemble de lignes (ou n-uplets).

Un premier prototype de Système de gestion de bases de données relationnelles (SGBDR) est construit dans les laboratoires d'IBM. Depuis les années 1980. (Source : <http://www.adproxima.fr/glossaire-208-base-donnees.html> )

En somme : elle réalise les fonctionnalités permettant d'assurer le bon fonctionnement d'une base de données :

- stockage des données et gestion de l'espace disque
- gestion du dictionnaire de données
- recherche et modification des données
- sécurité des données ainsi que leur intégrité et leur confidentialité
- gestion de la concurrence d'accès).



## **LE SQL (STRUCTURED QUERY LANGUAGE) :**

C'est un langage informatique qui permet d'interagir avec des bases de données relationnelles. C'est le langage pour base de données le plus répandu, et c'est bien sûr celui utilisé par MySQL.

C'est donc le langage que nous allons utiliser pour dire au client MySQL d'effectuer des opérations sur la base de données stockée sur le serveur MySQL.

Il a été créé dans les années 1970 et c'est devenu standard en 1986 (pour la norme ANSI - 1987 en ce qui concerne la norme ISO). Il est encore régulièrement amélioré.

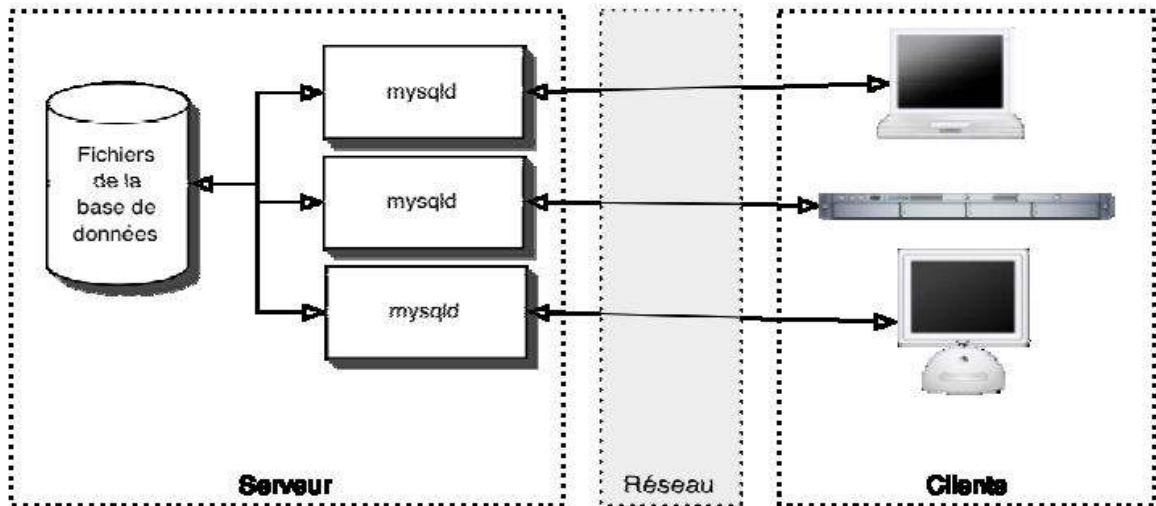
## **MYSQL**

Il s'agit d'un des SGBDR les plus connus et les plus utilisés (Wikipédia et Adobe utilisent par exemple MySQL). Et c'est certainement le SGBDR le plus utilisé à ce jour pour réaliser des sites web dynamiques.

### **Le Modèle client - serveur :**

La plupart des SGBD sont basés sur un modèle Client - Serveur. C'est-à-dire que la base de données se trouve sur un serveur qui ne sert qu'à ça, et pour interagir avec cette base de données, il faut utiliser un logiciel "client" qui va interroger le serveur et transmettre la réponse que le serveur lui aura donnée.

Le serveur peut être installé sur une machine différente du client ; c'est souvent le cas lorsque les bases de données sont importantes.



L'objet le plus représenté d'une base de données est la table. Chaque table (appelées encore « relation ») est caractérisée par une ou plusieurs colonnes (ou « attributs »). Le langage qui permet de gérer ces objets est appelé « Langage de Description des Données » (LDD).

Les données sont stockées dans les tables sous forme de lignes (ou « tuples »). Le langage qui permet de manipuler les données est appelé « Langage de Manipulation des Données » (LMD).

## UTILISATION DE MYSQL

Nous avons deux modes d'utilisation de MYSQL :

### 1. MYSQL AVEC L'INTERFACE : PHPMYADMIN

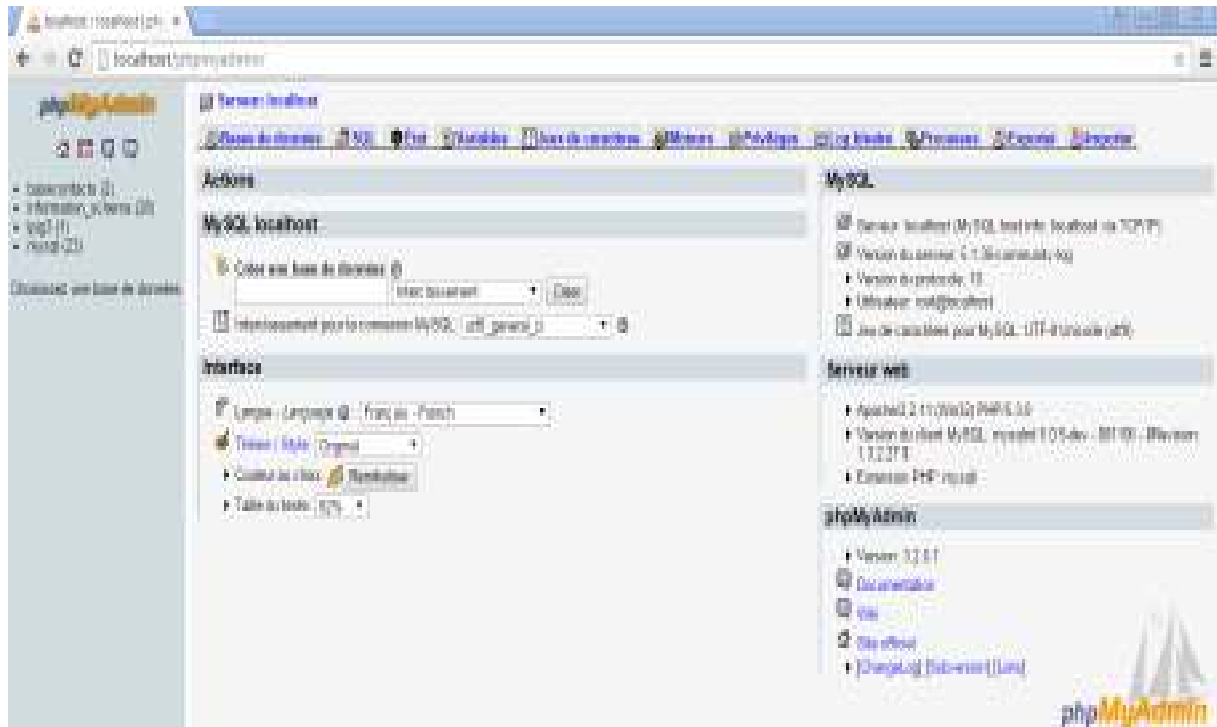
Cependant, grâce à la grande popularité de MySQL, des interfaces plus conviviales ont vu le jour. Les plus utilisées sont :

- **phpMyAdmin** est, comme son nom l'indique, une interface Web écrite en PHP
- **Webmin**, il existe un module d'administration de MySQL pour Webmin
- **WinMySQLAdmin** est une interface pour les systèmes Windows ;
- MySQLCC (« MySQL Control Center », <http://www.mysql.com/products/mysqlcc>) .

L'interface la plus répandue est phpMyAdmin.

phpMyAdmin est disponible sur ce site : <http://www.phpmyadmin.net>

Pour y accéder taper sur la navigateur <http://localhost/phpmyadmin/>

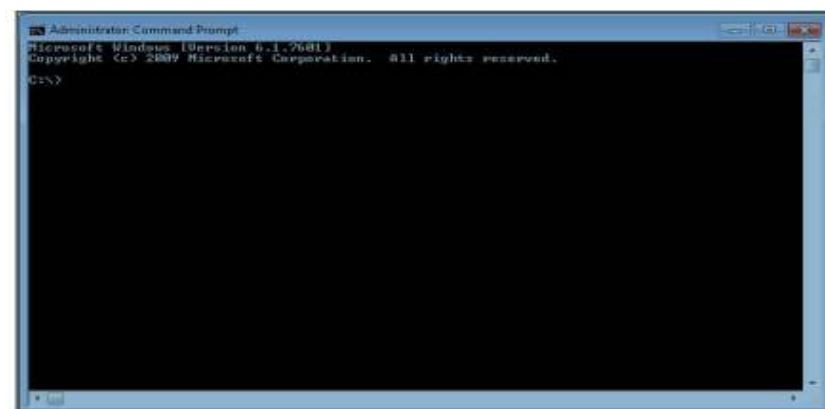


## 2. MYSQL AVEC UNE CONSOLE

Plus compliqué mais plus souple et plus légère. Il nous permet aussi :

- Création d'une base de données et des tables nécessaires à la gestion des données
- Gestion des relations entre les différentes tables d'une base
- Sélection des données selon de nombreux critères
- Manipulation des données (modification, suppression, calculs divers)
- Utilisation des triggers et des procédures stockées pour automatiser certaines actions
- Utilisation des vues et des tables temporaires
- Gestion des utilisateurs de la base de données

Mais faudra avant configurer les variable d'environnement. Voir la fin du cours.



Nous allons nous appesantir sur l'utilisation en ligne de commande.

Pourquoi utiliser la ligne de commande ? Quatre raisons :

- Pour que maîtrisez vraiment les commandes SQL.
- Parce qu'il est fort probable que vous désiriez utiliser MySQL en combinaison avec un autre langage de programmation (Php par exemple).  
Or, dans du code PHP on est obligé d'utiliser le SQL.
- En entreprise on utilise plus la console.
- La console est plus simple et plus légère.

## **LES CONCURRENTS DE MYSQL**

Il existe des dizaines de SGBDR, chacun ayant ses avantages et ses inconvénients.

- ORACLE
- PostgreSQL
- MS Access
- SQLite

## **EN RESUME**

- MySQL est un Système de Gestion de Bases de Données Relationnelles (SGBDR) basé sur le modèle client-serveur.
- Le langage SQL est utilisé pour communiquer entre le client et le serveur.
- Dans une base de données relationnelle, les données sont représentées sous forme de tables.

## **INSTALLATION DE MYSQL**

Pour installer MySQL nous pouvons soit :

- Installer directement le logiciel Mysql
- Installer des paquets (logiciel) contenant MySQL

Exemple de paquets sous Windows :

- EASYPHP
- WAMP SERVER
- ....

Exemple de paquets sous Unix :

- LAMP C'est la version WAMP sous linux

Exemple de paquets sous MAC :

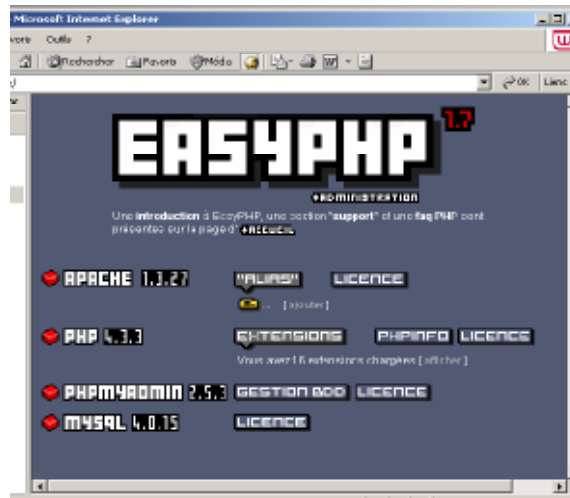
- MAMP C'est la version WAMP sous MAC OS

EasyPHP v 1.7 contient en même temps :

- PHP v4.3.3 : pour développer en PHP et interagir avec la base de données
- MySQL v4.0.15 : le SGBDR pour la gestion de nos bases de données



- PhpMyAdmin v2.5.3 : le logiciel graphique d'administration la base de données
- Apache v1.3 : le serveur d'application



Système Windows : Le « package » WAMP SERVER 2.0 contient :

- PHP v4.3.3
- MySQL v4.0.15 : le SGBDR
- PhpMyAdmin : le logiciel graphique administration la base de données
- Apache v1.3 : le serveur web

Dans tous les cas l'installation est très simple. Dans le cadre de ce cours nous allons travailler avec WAMP en administration console. Donc télécharger et installer le .exe de Wamp Server. Par la suite nous apprendrons : Comment l'utiliser.

## **PARTIE 2 : CONNEXION, DECONNEXION, DEMARRAGE ET ARRET**

## CONNEXION A MYSQL

Nous allons nous connecter via notre client à notre serveur Mysql afin de pouvoir y insérer, consulter et modifier des données.

La commande pour lancer le client est tout simplement son nom. Taper simplement Mysql sous CMD

```
C:\Users\edvaldo>> mysql
```

Exemple :

```
C:\Windows\system32\cmd.exe
Microsoft Windows [version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

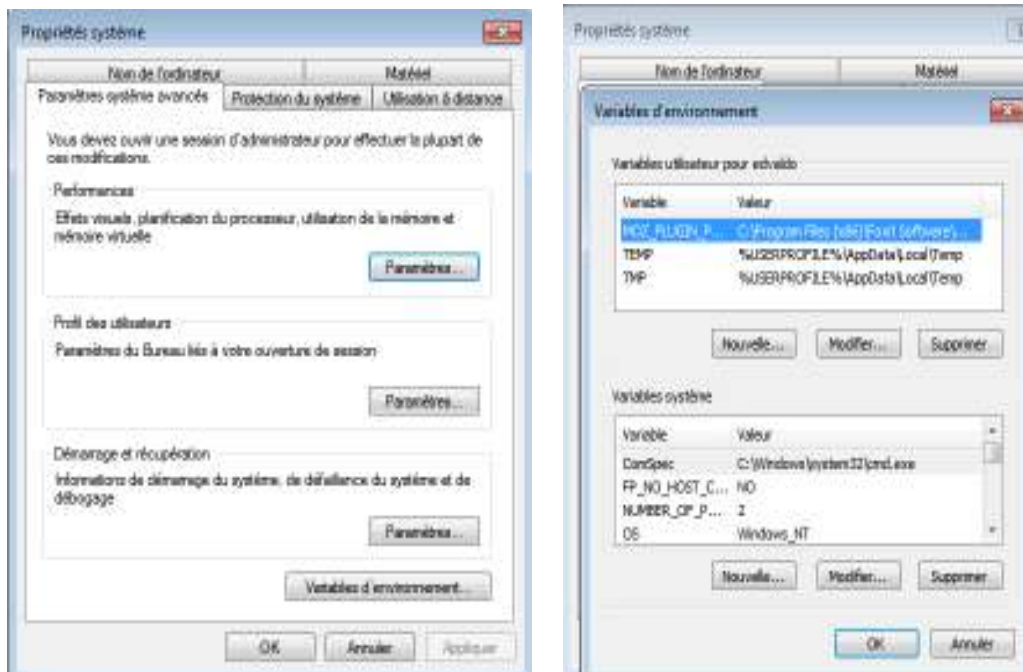
C:\Users\edvaldo>mysql
'mysql' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.

C:\Users\edvaldo>
```

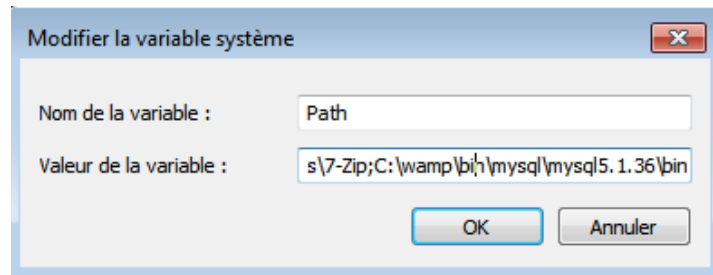
### MySQL non reconnu par ligne de commande Windows

#### **RESOLUTION : Changer la variable d'environnement.**

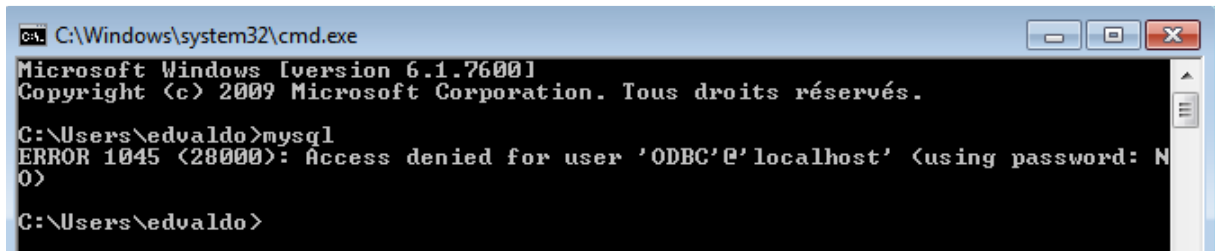
- Ouvrir le panneau de configuration / Système / Paramètres Système avancé / Variables d'environnement.



- Sélectionner PATH puis modifier et y coller (après un point-virgule) le chemin d'accès du répertoire bin de MySQL dans le repertoire d'installation de Mysql  
C:\wamp\bin\mysql\mysql5.1.36\bin



- Fermer la console et relancer le puis taper Mysql à nouveau



C'est bon maintenant.

Cependant cela ne suffit pas. Il vous faut également préciser un certain nombre de paramètres. Le serveur MySQL a besoin d'au minimum trois paramètres :

- **l'hôte** : c'est-à-dire l'endroit où est localisé le serveur ;
- **le nom d'utilisateur** ;
- **le mot de passe de l'utilisateur**.

En effet :

L'hôte et l'utilisateur ont des valeurs par défaut, et ne sont donc pas toujours indispensables. La valeur par défaut de l'hôte est "**localhost**", ce qui signifie que le serveur est sur le même ordinateur que le client. C'est bien notre cas, donc nous n'aurons pas à préciser ce paramètre.

Pour le nom d'utilisateur, la valeur par défaut dépend de votre système.

- Sous Windows, l'utilisateur courant est "**ODBC**",
- Sous Unix (Mac et Linux), il s'agit de votre nom d'utilisateur (le nom qui apparaît dans l'invite de commande).

**NB :** Pour votre première connexion à MySQL, il faudra vous connecter avec l'utilisateur "**root**", pour lequel vous avez normalement défini un mot de passe. Par la suite, nous créerons un nouvel utilisateur.

La commande complète pour se connecter est donc :

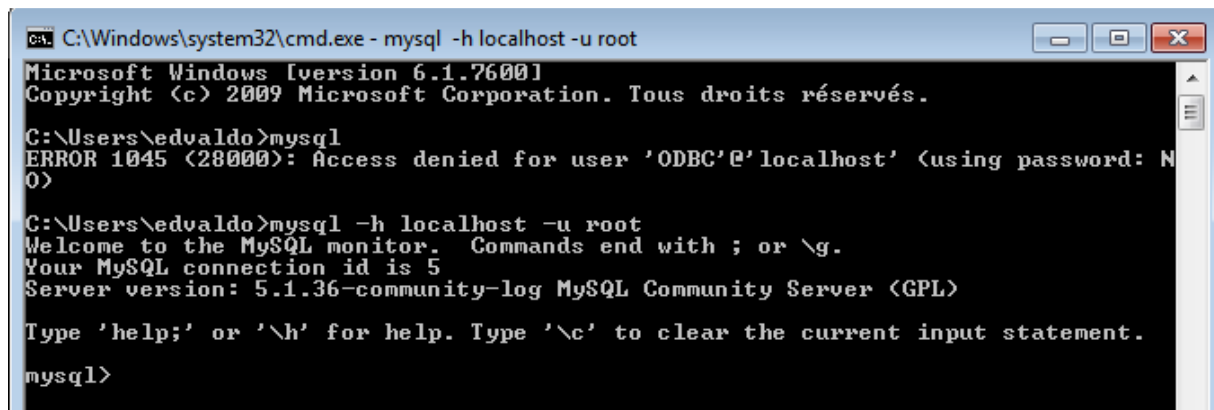
### Méthode 1 :

```
mysql -h localhost -u root -psarr
```

On suppose ici que sarr est votre mot de passe.

Dans notre cas notre User est root qui n'a pas de mot de passe.

```
mysql -h localhost -u root
```



```
C:\Windows\system32\cmd.exe - mysql -h localhost -u root
Microsoft Windows [version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. Tous droits réservés.

C:\Users\edvaldo>mysql
ERROR 1045 (28000): Access denied for user 'ODBC'@'localhost' (using password: NO)

C:\Users\edvaldo>mysql -h localhost -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.1.36-community-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

### Méthode 2 :

```
mysql --host=localhost --user=root --password=sarr
```

Remarquez l'absence d'espace entre -p et le mot de passe ceci est souvent source d'erreurs.

### Méthode 3 :

Notez que pour le mot de passe, il est possible (et c'est même très conseillé) de préciser uniquement que vous utilisez le paramètre, sans lui donner de valeur :

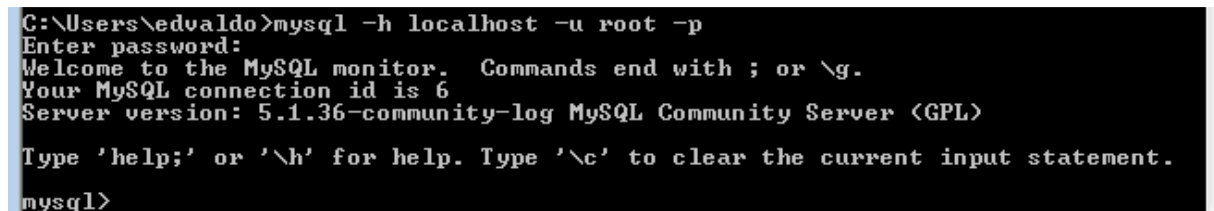
```
mysql -h localhost -u root -p
```

Apparaissent alors dans la console les mots suivants :

```
Enter password:
```

Tapez donc votre mot de passe, et là, vous pouvez constater que les lettres que vous tapez ne s'affichent pas. C'est normal, pour plus de sécurité.

Exemple :



```
C:\Users\edvaldo>mysql -h localhost -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.1.36-community-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Dans notre cas :

Serveur : localhost  
User : Root  
Password : (Rien)  
La commande devient

#### **Méthode 4 : On peut aussi omettre le -h localhost vue que nous somme en localhost**

```
mysql -u root -p
Enter password:
Musql>
```

#### **Exemple :**

```
C:\Users\edvaldo>mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7
Server version: 5.1.36-community-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

#### **TESTE DE CONNEXION**

Vous pouvez constater que vous êtes connectés. Faut aussi noter le changement de l'invite de commande. On voit maintenant :

```
mysql>
```

#### **TEST avec "Hello World !"**

Traditionnellement, lorsque l'on apprend un langage informatique, la première chose que l'on fait, c'est afficher le célèbre message "Hello World !".

```
SELECT 'Hello World !';
```

SELECT est la commande qui permet la sélection de données, mais aussi l'affichage. Vous devriez donc voir s'afficher deux fois "Hello World !" parce que Mysql représente les données sous forme de tables.

#### **DECONNEXION :**

On tape simplement Quit

```
mysql> quit
Bye
C:\Users\edvaldo>
```

#### **REGLES DU SQL**

##### **Fin d'une instruction :**

Pour signifier à MySQL qu'une instruction est terminée, il faut mettre le caractère **;**. Tant qu'il ne rencontre pas ce caractère, le client MySQL pense que vous n'avez pas fini d'écrire votre commande et attend gentiment que vous continuiez.

## Commentaires :

Les commentaires sont des parties de code qui ne sont pas interprétées. Ils servent principalement à vous repérer dans votre code.

En SQL, les commentaires sont introduits par `--` (deux tirets). Cependant, MySQL déroge un peu à la règle SQL et accepte deux syntaxes :

`#` : tout ce qui suit ce caractère sera considéré comme commentaire

`--` : la syntaxe normale est acceptée uniquement si les deux tirets sont suivis d'une espace au moins

## Chaînes de caractères :

Lorsque vous écrivez une chaîne de caractères dans une commande SQL, il faut absolument l'entourer de guillemets simples (donc des apostrophes).

Exemple : Pour afficher un message.

```
SELECT 'Bonjour SARR !' ;
```

Par ailleurs, si vous désirez utiliser un caractère spécial dans une chaîne, il vous faudra l'échapper avec `\`. Par exemple, si vous entourez votre chaîne de caractères de guillemets simples mais voulez utiliser un tel guillemet à l'intérieur de votre chaîne :

```
SELECT 'Salut l'ami'; -- Pas bien !  
SELECT 'Salut l\'ami'; -- Bien !
```

Quelques autres caractères spéciaux :

`\n` retour à la ligne

`\t` tabulation

`_` pour souligner

Exemple : Afficher la phrase suivante :

*Je suis l'ami de M sarr*

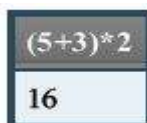
```
SELECT ' Je suis l'ami de M sarr'; -- Bien !
```

## UN PEU DE MATH :

MySQL est également doué en calcul :

```
SELECT (5+3)*2;
```

Pas de guillemets cette fois puisqu'il s'agit de nombres. MySQL calcule pour nous et nous affiche le résultat qui est de 16.



MySQL est sensible à la priorité des opérations, comme vous pourrez le constater en tapant cette commande :

```
SELECT (5+3)*2, 5+3*2;
```

Résultat :

$(5+3)*2$	$5+3*2$
16	11

Mettre une virgule dans le select c'est comme aller à un colonne suivante .

**DEMARRER LE SERVEUR MYSQL**

**ARRETER MYSQL**



## **PARTIE 3 :**

# **CREATION BASE DE DONNEES ET ORDRES SQL**

- **Création de bases de données**

L'une des tâches de l'administration base de données est de pouvoir créer une base de données et aussi être à mesure d'y créer des tables.

```
CREATE DATABASE NOM_BASE;
```

```
mysql> create database lpig;  
Query OK, 1 row affected (0.01 sec)
```

- **Vérifier la création.**

```
SHOW DATABASES ;
```

```
mysql> show databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| basebrioche |  
| basecontacts |  
| baselpig |  
| bclients |  
| bcomptes |  
| bemployes |  
| bdload |  
| bdls |  
| bdsde |  
| bdsde3 |  
| bdsenelec2 |  
| bdsenelec3 |  
| bdventes |  
| bemployes |  
| btransactions |  
| clients |  
| commandes |
```

Il est possible de définir l'encodage utilisé (l'UTF-8 par exemple). Voici donc la commande complète à taper pour créer votre base :

```
CREATE DATABASE NomBase CHARACTER SET 'utf8';
```

```
mysql> CREATE DATABASE baselpig CHARACTER SET 'utf8';  
Query OK, 1 row affected (0.02 sec)
```

- **Suppression de la base de données**

```
DROP DATABASE NomBase ;
```

```
mysql> drop database lpig3;  
Query OK, 1 row affected (0.16 sec)  
  
mysql>
```

- Se Connecter à une base de données :

```
USE NOMBASE ;
```

```
mysql>  
mysql> use lpig;  
Database changed  
mysql>
```

NB : USE permet de passer d'une base à une autre.

- Lister les tables d'une base de données

```
SHOW TABLES;
```

```
mysql> use gnotes;  
Database changed  
mysql> show tables;  
+-----+  
| Tables_in_gnotes |  
+-----+  
| annees            |  
| classes           |  
| departements      |  
| etudiants         |  
| evaluations       |  
| filieres          |  
| jour              |  
| login             |  
| module            |  
| mois              |  
| notes             |  
| unite_enseignements |  
+-----+  
12 rows in set (0.02 sec)
```

### Création de tables

```
CREATE TABLE Etudiant (  
  nom VARCHAR(20),  
  prenom VARCHAR(20),  
  adresse VARCHAR(20),  
  sexe CHAR(1),  
  naissance DATE  
);
```

```
mysql> use lpig;
Database changed
mysql> CREATE TABLE Etudiant (nom VARCHAR(20), prenom VARCHAR(20), ad
(1), naissance DATE);
Query OK, 0 rows affected (0.01 sec)

mysql> show tables;
+-----+
| Tables_in_lpig |
+-----+
| etudiant       |
+-----+
1 row in set (0.00 sec)

mysql>
```

## NB : Les moteurs de tables

Les moteurs de tables sont une spécificité de MySQL. Ce sont des moteurs de stockage. Cela permet de gérer différemment les tables selon l'utilité qu'on en a. Je ne vais pas Les deux moteurs les plus connus sont MyISAM et InnoDB.

### 1. MyISAM

C'est le moteur par défaut. Les commandes d'insertion et sélection de données sont particulièrement rapides sur les tables utilisant ce moteur. Cependant, il ne gère pas certaines fonctionnalités importantes comme les clés étrangères, qui permettent de vérifier l'intégrité d'une référence d'une table à une autre table, les transactions, qui permettent de réaliser des séries de modifications "en bloc" ou au contraire d'annuler ces modifications.

### 2. InnoDB

Plus lent et plus gourmand en ressources que MyISAM, ce moteur gère les clés étrangères et les transactions. Étant donné que nous nous servons des clés étrangères dès la deuxième partie, c'est celui-là que nous allons utiliser. De plus, en cas de crash du serveur, il possède un système de récupération automatique des données.

Préciser un moteur lors de la création de la table

Pour qu'une table utilise le moteur de notre choix, il suffit d'ajouter ceci à la fin de la commande de création :

ENGINE = moteur;

En remplaçant "moteur" par le nom du moteur que nous voulons utiliser, ici

InnoDB :

ENGINE = INNODB;

## Suppression de Table :

```
mysql> drop table etudiant;
Query OK, 0 rows affected (0.01 sec)
```

**PRINCIPAUX TYPE DE DONNÉES**

Type de données	Signification / valeur	Taille
TINYINT	-128 à 127 ou 0 à 255	1 bit
SMALLINT	-32768 à 32767 ou 0 à 65535	2 bits
MEDIUMINT	-8388608 à 8388607 ou 0 à 16 777 215	3 bits
INT ou INTEGER	-2.14 E9 à 2.14 E9 ou 0 à 4.29 E9	4 bits
BIGINT	-9.22 E18 à 9.22 E18 ou 0 à 18.44 E18	8 bits
FLOAT	-3.40 <sup>E38</sup> à 1.17 <sup>E-38</sup>	8 bits
DOUBLE	-1.79 <sup>E308</sup> à 2.22 <sup>E308</sup>	8 bits
NUMERIC	-1.79 <sup>E308</sup> à 2.22 <sup>E308</sup>	<= 8 bits
DATETIME	1000-01-01 00:00:00' à 9999-12-31 23:59:59'	8 bits
DATE	1000-01-01 à 9999-12-31	3 bits
TIMESTAMP	La durée en seconde depuis le 1-01-1970	4 bits
TIME	L'heure	3 bits
YEAR	L'année	1 bit
CHAR	Chaîne de longueur fixe 1 à 255	1 à 255 octets

VARCHAR	Chaîne de longueur variable (1 à 255)	1 à 255 octets
BLOB ou TEXT	Chaîne de longueur variable (1 à 65535)	2 à 65537 octets
MEDIUMBLOB ou MEDIUMTEXT	Chaîne de longueur variable (1 à plus de 16 millions)	3 à plus de 16 millions d'octets
LOB ou LONGTEXT	Chaîne de longueur variable (1 à plus de 4 milliards)	4 à plus de 4 milliards d'octets
ENUM	Liste de valeurs (65535 max)	1 à 2 octets
SET	Liste de valeurs (64 max)	1 à 8 octets

**Exemple : Supprimons la table Etudiant et recreons la avec comme moteur INNODB**

```
CREATE TABLE Etudiant (
  nom VARCHAR(20),
  prenom VARCHAR(20),
  adresse VARCHAR(20),
  sexe CHAR(1),
  naissance DATE
)
ENGINE = INNODB;
```

```
mysql> drop table Etudiant;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE Etudiant (
  ->   nom VARCHAR(20),
  ->   prenom VARCHAR(20),
  ->   adresse VARCHAR(20),
  ->   sexe CHAR(1),
  ->   naissance DATE
  -> )
  -> ENGINE = INNODB;
Query OK, 0 rows affected (0.09 sec)
```

**Pourquoi ?** Simplement parce que nous aurons à effectuer des transactions sur cette table.

### Description d'une table

```
DESCRIBE NomTable;
```

```
mysql> describe etudiant;
```

Field	Type	Null	Key	Default	Extra
nom	varchar(20)	YES		NULL	
prenom	varchar(20)	YES		NULL	
adresse	varchar(20)	YES		NULL	
sexe	char(1)	YES		NULL	
naissance	date	YES		NULL	

5 rows in set (0.01 sec)

### Insérer des données

```
INSERT INTO Etudiant VALUES ('SARR','Edouard','Mbour','M','1999-03-30');
```

```
mysql> INSERT INTO Etudiant VALUES ('SARR','Edouard','Mbour','M','1999-03-30')
Query OK, 1 row affected (0.06 sec)
```

```
mysql>
```

### Lister les informations

```
SELECT quoi_selectionner
FROM quel_table
WHERE conditions_a_satisfaire ;
```

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30

3 rows in set (0.00 sec)

```
mysql> select prenom,nom from etudiant;
```

prenom	nom
Edouard	SARR
Anissa	SAGNE
Abdou	Tall

3 rows in set (0.00 sec)

On peut avoir :

- Plusieurs conditions "AND" / « OR »
- DISTINCT

### Pour trouver les noms commençant par la lettre 'S' :

WHERE nom LIKE "S%";

```
mysql> select * from etudiant where nom like "S%";
```

nom	prenom	adresse	sexe	naissance
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30

2 rows in set (0.00 sec)

- Pour trouver les noms finissant par 'RR' :

WHERE nom LIKE "%RR";

```
mysql> select * from etudiant where nom like "%RR";
```

nom	prenom	adresse	sexe	naissance
SARR	Edouard	Mbour	M	1999-03-30

1 row in set (0.00 sec)

- Pour trouver les noms contenant le caractère 'a' :

WHERE nom LIKE "%a%";

```
mysql> select * from etudiant where nom like "%a%";
```

nom	prenom	adresse	sexe	naissance
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30

3 rows in set (0.00 sec)

- Pour trouver les noms contenant exactement 4 caractères, utilisez le caractère de recherche '\_' :

WHERE nom LIKE "\_\_\_\_";

```
mysql> select * from etudiant where nom like "____";
```

nom	prenom	adresse	sexe	naissance
SARR	Edouard	Mbour	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30

2 rows in set (0.00 sec)

- Pour trouver les noms qui commencent par la lettre 'S', utilisez '^' pour trouver le début du nom :

```
WHERE nom REGEXP "^S";
```

```
mysql> select * from etudiant where nom REGEXP "^S";;
+-----+-----+-----+-----+-----+
| nom   | prenom | adresse | sexe | naissance |
+-----+-----+-----+-----+-----+
| SARR  | Edouard | Mbour   | M    | 1999-03-30 |
| SAGNE | Anissa  | Dakar   | M    | 1999-03-30 |
+-----+-----+-----+-----+-----+
2 rows in set (0.13 sec)
```

- Pour trouver les noms finissant par 'E', utilisez '\$' pour trouver la fin du nom :

```
WHERE nom REGEXP "fy$";
```

```
mysql> select * from etudiant where nom REGEXP "E$";;
+-----+-----+-----+-----+-----+
| nom   | prenom | adresse | sexe | naissance |
+-----+-----+-----+-----+-----+
| SAGNE | Anissa | Dakar   | M    | 1999-03-30 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

- Pour trouver les noms contenant la lettre 'G' minuscule ou majuscule, utilisez la requête suivante :

```
WHERE nom REGEXP "G";
```

```
mysql> select * from etudiant where nom REGEXP "g";;
+-----+-----+-----+-----+-----+
| nom   | prenom | adresse | sexe | naissance |
+-----+-----+-----+-----+-----+
| SAGNE | Anissa | Dakar   | M    | 1999-03-30 |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

## Colonne AUTO\_INCREMENT

L'attribut AUTO\_INCREMENT peut être utile pour générer un id unique pour les nouvelles lignes.

```
Create Table ListeEtudiant(
    Num MediumInt Not Null AUTO_INCREMENT,
    prenom varchar(33),
    Nom Varchar(44),
    primary key(Num));
```



```
mysql> Create Table ListeEtudiant(
-> Num MediumInt Not Null AUTO_INCREMENT,
-> prenom varchar(33),
-> Nom Varchar(44),
-> primary key(Num));
Query OK, 0 rows affected (0.08 sec)
```

NB : Une colonne Auto increment est toujours NOT NULL.  
Lors de l'insertion nous n'avons pas besoin de donner une valeur à cette colonne.

### Modification d'un enregistrement : UPDATE

```
mysql> Update etudiant set prenom='Edouard Ngor' where nom='Sarr';
Query OK, 1 row affected (0.11 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
SARR	Edouard Ngor	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30

```
3 rows in set (0.00 sec)
```

### TP 1: Connexion, création Base, Ordre SQL

Créer la base de données LPIG3 avec la table suivante :

Etudiants (Matricule, Nom, Classe, Adresse, Age) ;

- 0- insérer 10 étudiants
- 1- Lister les étudiants dont les noms commence par N
- 2- Lister les étudiants dont les noms finissent par R
- 3- lister des étudiants dont les prénoms contient la lettre 'a '
- 4- Liste les étudiants dont les noms commence par S et se termine par R
- 5- Modifier le nom de l'étudiant de matricule 0001 et le nommé SARR
- 6- Supprimer les étudiants Mbourois
- 7- Quels sont les étudiants qui habite la même ville que l'étudiant 0001 et qui sont dans la meme classe que 0008.

## **Partie 4 : LA GESTION DES USERS ET DES PRIVILEGES**

Lorsque l'on se connecte à MySQL, on le fait avec un utilisateur. Chaque utilisateur possède une série de privilèges, relatifs aux données stockées sur le serveur : le privilège de sélectionner les données, de les modifier, de créer des objets, etc.

Ces privilèges peuvent exister à plusieurs niveaux :

- base de données
- tables
- colonnes
- procédures
- etc.

Toutes ces informations sont stockées dans la base de données MYSQL.

En faisant un `SHOW DATABASES ;` sous ROOT vous verrez cette base.

Les utilisateurs sont stockés dans la table user, avec leurs privilèges globaux (c'est-à-dire valables au niveau du serveur, sur toutes les bases de données). La base mysql possède par ailleurs quatre tables permettant de stocker les privilèges des utilisateurs à différents niveaux.

- db : privilèges au niveau des bases de données.
- tables\_priv : privilèges au niveau des tables.
- columns\_priv : privilèges au niveau des colonnes.
- proc\_priv : privilèges au niveau des routines (procédures et fonctions stockées).

```
mysql> use mysql;
Database changed
mysql> show tables;
+-----+
| Tables_in_mysql |
+-----+
| columns_priv     |
| db               |
| event            |
| func             |
| general_log      |
| help_category    |
| help_keyword     |
| help_relation    |
| help_topic       |
| host             |
| ndb_binlog_index |
| plugin           |
| proc             |
| procs_priv       |
| servers          |
| slow_log         |
| tables_priv      |
| time_zone        |
| time_zone_leap_second |
| time_zone_name   |
| time_zone_transition |
| time_zone_transition_type |
| user             |
+-----+
23 rows in set (0.13 sec)

mysql>
```

Il est tout à fait possible d'ajouter, modifier et supprimer des utilisateurs en utilisant des requêtes INSERT, UPDATE ou DELETE directement sur la table mysql.user. De même pour leurs privilèges, avec les tables mysql.db, mysql.tables\_priv, mysql.columns\_priv et mysql.procs\_priv.

```
mysql> describe user;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| 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 | | |
| Event_priv      | enum('N','Y') | NO | | N | | |
| x509_subject     | blob          | NO | | NULL | | |
| 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 | | |
+-----+-----+-----+-----+-----+-----+
39 rows in set (0.01 sec)
```

Cependant, en général, on utilise plutôt des commandes dédiées à cet usage.

## CREATION ET SUPPRESSION AVEC CREATE USER ET DROP USER

-- Création

```
CREATE USER 'login'@'hote' [IDENTIFIED BY 'mot_de_passe'];
```

L'utilisateur est donc défini par trois éléments :

- son login est le nom du user et son mot de passe
- l'hôte est le nom ou l'adresse IP du serveur

-- Suppression

```
DROP USER 'login'@'hote';
```

Exemple :

```
mysql>
mysql> create user Ami@localhost identified by 'ami';
Query OK, 0 rows affected (0.17 sec)

mysql> create user awa@localhost;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> create user awa@192.168.0.2 identified by 'awa';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql>
mysql> drop user awa@localhost;
Query OK, 0 rows affected (0.06 sec)
```

Il est également possible de permettre à un utilisateur de se connecter à partir de plusieurs hôtes différents (sans devoir créer un utilisateur par hôte) : en utilisant le joker %, on peut préciser des noms d'hôtes partiels, ou permettre à l'utilisateur de se connecter à partir de n'importe quel hôte.

### Exemples

-- fatouFall peut se connecter à partir de n'importe quel hôte dont l'adresse IP commence par 194.28.12.

```
CREATE USER FatouFall@'194.28.12.%' IDENTIFIED BY 'fatou';
```

-- maxime peut se connecter à partir de n'importe quel hôte du domaine brab.net

```
CREATE USER maxime @'%ucao.sn' IDENTIFIED BY 'maxime';
```

-- berou peut se connecter à partir de n'importe quel hôte

```
CREATE USER 'berou'@'%' IDENTIFIED BY 'berou';
```

### RENOMMER L'UTILISATEUR

Pour modifier l'identifiant d'un utilisateur (login et/ou hôte), on peut utiliser

```
RENAME USER ancien_utilisateur TO nouvel_utilisateur ;
```

Exemple : on renomme max en maxime, en gardant le même hôte.

```
RENAME USER max@localhost TO maxime@localhost;
```

### MODIFIER LE MOT DE PASSE

Pour modifier le mot de passe d'un utilisateur, on peut utiliser la commande SET PASSWORD (à condition d'avoir les privilèges nécessaires, ou d'être connecté avec l'utilisateur dont on veut changer le mot de passe).

Cependant, cette commande ne hashé pas le mot de passe automatiquement. Il faut donc utiliser la fonction PASSWORD().

### Exemple

```
SET PASSWORD FOR edou@'194.28.12.%' = PASSWORD('edou');
```

## LES PRIVILEGES

Lorsque l'on crée un utilisateur avec CREATE USER, celui-ci n'a au départ aucun privilège, aucun droit. En SQL, avoir un privilège, c'est avoir l'autorisation d'effectuer une action sur un objet.

Un utilisateur sans aucun privilège ne peut rien faire d'autre que se connecter. Il n'aura pas accès aux données, ne pourra créer aucun objet (base/table/procédure/autre), ni en utiliser.

## LES DIFFERENTS PRIVILEGES

Il existe de nombreux privilèges dont voici une sélection des plus utilisés (à l'exception des privilèges particuliers ALL, USAGE et GRANT OPTION que nous verrons plus loin).

### PRIVILEGES DU CRUD

Les privilèges SELECT, INSERT, UPDATE et DELETE permettent aux utilisateurs d'utiliser ces mêmes commandes. Privilèges concernant les tables, les vues et les bases de données

Privilège	Action autorisée
CREATE TABLE	Création de tables
CREATE TEMPORARY TABLE	Création de tables temporaires
CREATE VIEW	Création de vues (il faut également avoir le privilège SELECT sur les colonnes sélectionnées par la vue)
ALTER	Modification de tables (avec ALTER TABLE)
DROP	Suppression de tables, vues et bases de données

### *Autres privilèges*

Privilège	Action autorisée
CREATE ROUTINE	Création de procédures stockées (et de fonctions stockées - non couvert dans ce cours)
ALTER ROUTINE	Modification et suppression de procédures stockées (et fonctions stockées)
EXECUTE	Exécution de procédures stockées (et fonctions stockées)
INDEX	Création et suppression d'index

TRIGGER	Création et suppression de triggers
LOCK TABLES	Verrouillage de tables (sur lesquelles on a le privilège SELECT)
CREATE USER	Gestion d'utilisateur (commandes CREATE USER, DROP USER, RENAME USER et SET PASSWORD)



## LES DIFFERENTS NIVEAUX D'APPLICATION DES PRIVILEGES

Lorsque l'on accorde un privilège à un utilisateur, il faut également préciser à quoi s'applique ce privilège.

Niveau	Application du privilège
<b>*.*</b>	Privilège global : s'applique à toutes les bases de données, à tous les objets. Un privilège de ce niveau sera stocké dans la table <i>mysql.user</i> .
<b>*</b>	Si aucune base de données n'a été préalablement sélectionnée (avec <code>USE nom_bdd</code> ), c'est l'équivalent de *.* (privilège stocké dans <i>mysql.user</i> ). Sinon, le privilège s'appliquera à tous les objets de la base de données qu'on utilise (et sera stocké dans la table <i>mysql.db</i> ).
<i>nom_bdd.*</i>	Privilège de base de données : s'applique à tous les objets de la base <i>nom_bdd</i> (stocké dans <i>mysql.db</i> ).
<i>nom_bdd.nom_table</i>	Privilège de table (stocké dans <i>mysql.tables_priv</i> ).
<i>nom_table</i>	Privilège de table : s'applique à la table <i>nom_table</i> de la base de données dans laquelle on se trouve, sélectionnée au préalable avec <code>USE nom_bdd</code> (stocké dans <i>mysql.tables_priv</i> ).
<i>nom_bdd.nom_routine</i>	S'applique à la procédure (ou fonction) stockée <i>nom_bdd.nom_routine</i> (privilège stocké dans <i>mysql.procs_priv</i> ).

Les privilèges peuvent aussi être restreints à certaines colonnes, auquel cas, ils seront stockés dans la table *mysql.columns\_priv*.

Nous verrons comment restreindre un privilège à certaines colonnes avec la commande `GRANT`.

## AJOUT ET REVOCATION DE PRIVILEGES

### • Ajout de privilèges

Pour pouvoir ajouter un privilège à un utilisateur, il faut posséder le privilège `GRANT OPTION`. Pour l'instant, seul l'utilisateur "root" le possède. Étant donné qu'il s'agit d'un privilège un peu particulier, nous n'en parlerons pas tout de suite. Connectez vous donc avec "root" pour exécuter les commandes de cette partie.

Syntaxe

```
GRANT privilege [(liste_colonnes)] [, privilege [(liste_colonnes)], ...]
ON [type_objet] niveau_privilege
TO utilisateur [IDENTIFIED BY mot_de_passe];
```

- `privilege` : le privilège à accorder à l'utilisateur (`SELECT`, `CREATE VIEW`, `EXECUTE`,...);
- `(liste_colonnes)` : facultatif - liste des colonnes auxquelles le privilège s'applique
- `niveau_privilege` : niveau auquel le privilège s'applique (\*.\*, *nom\_bdd.nom\_table*,...);

- type\_objet : en cas de noms ambigus, il est possible de préciser à quoi se rapporte le niveau : TABLE ou PROCEDURE.

On peut accorder plusieurs privilèges en une fois : il suffit de séparer les privilèges par une virgule. Si l'on veut restreindre tout ou partie des privilèges à certaines colonnes, la liste doit en être précisée pour chaque privilège.

Exemple :

```
mysql>
mysql> create user sarr@localhost identified by 'sarr';
Query OK, 0 rows affected (0.01 sec)

mysql> grant all privileges on gnotes.* to sarr@localhost identified by 'sarr';
Query OK, 0 rows affected (0.06 sec)

mysql>
```

NB :

- Si l'utilisateur auquel on accorde les privilèges n'existe pas, il sera créé. Auquel cas, il vaut mieux ne pas oublier la clause IDENTIFIED BY pour donner un mot de passe à l'utilisateur. Sinon, il pourra se connecter sans mot de passe.
- Si l'utilisateur existe, et qu'on ajoute la clause IDENTIFIED BY, son mot de passe sera modifié.

Exemples :

1. On crée un utilisateur anna@localhost, en lui donnant les privilèges SELECT, INSERT, DELETE et UPDATE sur la colonne prenom sur la table etudiant de la base LPIG.

```
mysql> use lpig;
Database changed
mysql> create user anna@localhost identified by 'anna';
Query OK, 0 rows affected (0.00 sec)

mysql> grant UPDATE(prenom),SELECT, INSERT, DELETE ON etudiant
-> to anna@localhost
-> identified by 'anna';
Query OK, 0 rows affected (0.00 sec)
```

2. Revoquons le INSERT de anna

```
mysql> Revoke insert on etudiant from anna@localhost;
Query OK, 0 rows affected (0.01 sec)
```

3. On se connecte en tant que anna

```
C:\Users\edvaldo>mysql -u anna -panna
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.1.36-community-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql>
```

#### 4. Testons un INSERT

```
mysql> use lpig
Database changed
mysql> INSERT INTO Etudiant VALUES ('mane','anna','Dakar','M','1999-03-30');
ERROR 1142 (42000): INSERT command denied to user 'anna'@'localhost' for table 'Etudiant'
mysql>
```

NB : on a révoqué le INSERT de anna

#### 5. Testons le select

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30
Diop	Ngone	Dakar	F	0000-00-00
Soumare	Ami	Dakar	M	1993-08-30
Sarr	Fatou	Dakar	F	1991-01-30
Diop	Ali	Dakar	M	1990-06-30

```
8 rows in set (0.00 sec)
```

Le SELECT réussit car anna dispose du privilège SELECT.

#### 6. Testons un Update sur la colonne PRENOM

```
mysql> update etudiant set prenom='anna marie' where nom='Soumare';
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30
Diop	Ngone	Dakar	F	0000-00-00
Soumare	anna marie	Dakar	M	1993-08-30
Sarr	Fatou	Dakar	F	1991-01-30
Diop	Ali	Dakar	M	1990-06-30

```
8 rows in set (0.00 sec)
```

Le UPDATE marche sur cette colonne

7. Faisons maintenant un UPDATE sur la colonne SEXE

```
mysql> update etudiant set sexe='F' where nom='Soumare';  
ERROR 1143 (42000): UPDATE command denied to user 'anna'@'localhost' for column 'sexe'  
mysql>
```

ERREUR : Anna ne peut pas faire un UPDATE sur une colonne autre que PRENOM.

## PRIVILEGES PARTICULIERS

- **Le privilège ALL** (ou ALL PRIVILEGES), comme son nom l'indique, représente tous les privilèges. Accorder le privilège ALL revient donc à accorder tous les droits à l'utilisateur.

Il faut évidemment préciser le niveau auquel tous les droits sont accordés (on octroie tous les privilèges possibles sur une table, ou sur une base de données, etc.).

Un privilège fait exception : GRANT OPTION n'est pas compris dans les privilèges représentés par ALL.

Exemple : on accorde tous les droits sur la table Client à 'anna'@'localhost'.

```
GRANT ALL ON LPIG.Etudiant TO 'anna'@'localhost';
```

Ou

```
GRANT ALL ON LPIG.Etudiant TO 'anna'@'localhost';
```

- **Le privilège USAGE** à l'inverse de ALL signifie "aucun privilège".

À première vue, utiliser la commande GRANT pour n'accorder aucun privilège peut sembler un peu ridicule. En réalité, c'est extrêmement utile : USAGE permet en fait de modifier les caractéristiques d'un compte avec la commande GRANT, sans modifier les privilèges du compte.

USAGE est toujours utilisé comme un privilège global (donc ON \*.\*).

Exemple : modification du mot de passe de 'john'@'localhost'. Ses privilèges ne changent pas.

```
GRANT USAGE ON *.* TO anna@'localhost' IDENTIFIED BY 'anna';
```

- **GRANT OPTION**

Un utilisateur ayant ce privilège est autorisé à utiliser la commande GRANT, pour accorder des privilèges à d'autres utilisateurs.

Ce privilège n'est pas compris dans le privilège ALL. Par ailleurs, un utilisateur ne peut accorder que les privilèges qu'il possède lui-même.

Exemple : on accorde les privilèges SELECT, UPDATE, INSERT, DELETE et GRANT OPTION sur la base de données LPIG à 'joseph'@'localhost'.

```
GRANT SELECT, UPDATE, INSERT, DELETE ON LPIG.* TO anna@'localhost'  
IDENTIFIED BY 'anna' WITH GRANT OPTION;
```

Ou:

```
GRANT SELECT, UPDATE, INSERT, DELETE, GRANT OPTION ON LPIG.* TO  
anna@'localhost' IDENTIFIED BY 'anna' WITH GRANT OPTION;
```

### OPTIONS SUPPLEMENTAIRES :

La commande GRANT possède encore deux clauses facultatives supplémentaires, permettant de limiter les ressources serveur de l'utilisateur, et d'obliger l'utilisateur à se connecter via SSL.

- **Limitation des ressources**

On peut limiter trois choses différentes pour un utilisateur :

- le nombre de requêtes par heure (MAX\_QUERIES\_PER\_HOUR) : limitation de toutes les commandes exécutées par l'utilisateur ;
- le nombre de modifications par heure (MAX\_UPDATES\_PER\_HOUR) : limitation des commandes entraînant la modification d'une table ou d'une base de donnée ;
- le nombre de connexions au serveur par heure (MAX\_CONNECTIONS\_PER\_HOUR).

Pour cela, on utilise la clause WITH MAX\_QUERIES\_PER\_HOUR nb ou MAX\_UPDATES\_PER\_HOUR nb ou MAX\_CONNECTIONS\_PER\_HOUR nb de la commande GRANT. On peut limiter une des ressources, ou deux, ou les trois en une fois, chacune avec un nombre différent.

```
GRANT ALL ON NomBase.* TO NomUser@'localhost' IDENTIFIED BY 'pwd'  
WITH MAX_QUERIES_PER_HOUR X  
MAX_CONNECTIONS_PER_HOUR Y;
```

X et Y des entiers.

Exemple : créons un user Ngor avec Maximum 1 requête par heure dans la base LPIG

```
GRANT ALL ON LPIG.* TO banna@'localhost' IDENTIFIED BY 'ngor' WITH  
MAX_QUERIES_PER_HOUR 1;
```

Vérification:

```
C:\Users\edvaldo>mysql -u ngor -pngor  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 9  
Server version: 5.1.36-community-log MySQL Community Server (GPL)  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement  
  
mysql> show databases;  
ERROR 1226 (42000): User 'ngor' has exceeded the 'max_questions' resource (cu  
mysql>
```

Exemple 2: Creons un User Siga avec maximum de connexion 1 par heure

```
GRANT ALL ON LPIG.* TO siga@'localhost' IDENTIFIED BY 'siga' WITH  
MAX_CONNECTIONS_PER_HOUR 1;
```

```
C:\Users\edvaldo>mysql -u siga -psiga  
Welcome to the MySQL monitor. Commands end with ; or \g.  
Your MySQL connection id is 11  
Server version: 5.1.36-community-log MySQL Community Server (GPL)  
  
Type 'help;' or '\h' for help. Type '\c' to clear the current input state  
mysql> Bye  
  
C:\Users\edvaldo>mysql -u siga -psiga  
ERROR 1226 (42000): User 'siga' has exceeded the 'max_connections_per_hou  
1)  
  
C:\Users\edvaldo>
```

Pour limiter les ressources d'un utilisateur existant sans modifier ses privilèges, on peut utiliser le privilège USAGE.

```
GRANT USAGE ON *.* TO demba@'localhost'  
WITH MAX_UPDATES_PER_HOUR 15;
```

Pour supprimer une limitation de ressources, il suffit de la mettre à zéro.

```
GRANT USAGE ON *.* TO demba@'localhost' WITH MAX_UPDATES_PER_HOUR 0;
```

## **Partie 5 : LES TRANSACTIONS**

Dans cette partie, nous allons aborder trois notions qui permettent de sécuriser une base de données :

- les transactions ;
- les verrous ;
- les requêtes préparées.

### **LES TRANSACTIONS :**

Les transactions sont une fonctionnalité absolument indispensable, permettant de sécuriser une application utilisant une base de données.

Sans transactions, certaines opérations risqueraient d'être à moitié réalisées, et la moindre erreur, la moindre interruption pourrait avoir des conséquences énormes.

En effet, les transactions permettent de regrouper des requêtes dans des blocs, et de faire en sorte que tout le bloc soit exécuté en une seule fois, cela afin de préserver l'intégrité des données de la base.

Une transaction, c'est un ensemble de requêtes qui sont exécutées en un seul bloc. Ainsi, si une des requêtes du bloc échoue, on peut décider d'annuler tout le bloc de requêtes (ou de quand même valider les requêtes qui ont réussi).

Supposons que nous devons ajouter faire un virement bancaire. le traitement d'un virement est plutôt simple, deux étapes suffisent :

- étape 1 : on retire le montant du virement du compte du donneur d'ordre ;
- étape 2 : on ajoute le montant du virement au compte du bénéficiaire.

Supposons que nous souhaitons faire le virement sans transaction. Ainsi on doit faire deux requêtes :

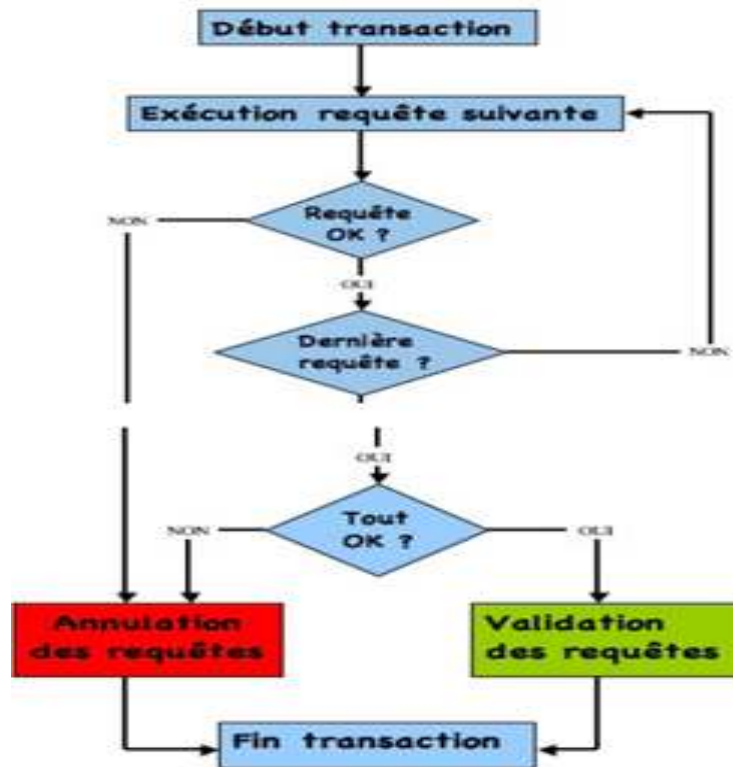
- Un Update pour le retrait
- Un update pour le dépôt dans l'autre compte

Supposons maintenant que juste après le deuxième UPDATE (le dépôt) survient une coupure de courant.

La seconde requête du traitement n'ayant jamais été exécutée, la première requête n'aurait jamais été validée.

PROBLEME, vous avez retirer sur un compte sans verser l'argent dans l'autre compte.





## Vocabulaire

Lorsque l'on valide les requêtes d'une transaction, on dit aussi que l'on committe les changements. À l'inverse, l'annulation des requêtes s'appelle un rollback.

NB : par défaut MySQL ne travaille pas avec les transactions. Chaque requête effectuée est directement committée (validée). On ne peut pas revenir en arrière. On peut donc en fait considérer que chaque requête constitue une transaction, qui est automatiquement committée. Par défaut, MySQL est donc en mode "autocommit".

Pour quitter ce mode, il suffit de lancer la requête suivante :

**SET autocommit=0;**

```
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
```

Une fois que vous n'êtes plus en autocommit, chaque modification de donnée devra être committée pour prendre effet. Tant que vos modifications ne sont pas validées, vous pouvez à tout moment les annuler (faire un rollback).

Les commandes pour commiter et faire un rollback sont relativement faciles à retenir :

COMMIT; -- pour valider les requêtes  
 ROLLBACK; -- pour annuler les requêtes

NB : N'oubliez pas le moteur qu'on avait choisit lors de la création de la table Etudiant pour pouvoir faire des transactions sur cette table. Sans cela, vous ne pourrez pas faire la suite du cours.

```
mysql> drop table Etudiant;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TABLE Etudiant (
  ->   nom VARCHAR(20),
  ->   prenom VARCHAR(20),
  ->   adresse VARCHAR(20),
  ->   sexe CHAR(1),
  ->   naissance DATE
  -> )
  -> ENGINE = INNODB;
Query OK, 0 rows affected (0.09 sec)
```

Exemple de transaction :

1. Faisons un select pour voir l'état de la table étudiant avant la transaction :

```
mysql> select * from etudiant;
+----+-----+-----+-----+-----+
| nom  | prenom | adresse | sexe | naissance |
+----+-----+-----+-----+-----+
| Diop | Ali    | Dakar   | M    | 1990-06-30 |
| SARR | Edouard | Mbour   | M    | 1999-03-30 |
| SAGNE | Anissa | Dakar   | M    | 1999-03-30 |
| Tall | Abdou  | Dakar   | M    | 1999-03-30 |
+----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

2. Ecrivons notre transaction sur NOTEPAD++

```
INSERT INTO Etudiant VALUES ('Ka','Abdou','Dakar','M','1988-03-30');
INSERT INTO Etudiant VALUES ('Diop','Ngone','Dakar','F','1999-02-30');
INSERT INTO Etudiant VALUES ('Soumare','Ami','Dakar','M','1993-08-30');
INSERT INTO Etudiant VALUES ('Sarr','Fatou','Dakar','F','1991-01-30');
INSERT INTO Etudiant VALUES ('Diop','Ali','Dakar','M','1990-06-30');
```

Enregistrer dans le disque D sous le nom trans.sql. Moi je vais le mettre dans  
E:\DONNEES\UCAO\MYSQL

3. Appelons le fichier par \.

```
mysql> \. E:\DONNEES\UCAO\MYSQL\trans.sql
Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected, 1 warning (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)
```

## 4. Faisons un nouveau select

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30
Ka	Abdou	Dakar	M	1988-03-30
Diop	Ngone	Dakar	F	0000-00-00
Soumare	Ami	Dakar	M	1993-08-30
Sarr	Fatou	Dakar	F	1991-01-30
Diop	Ali	Dakar	M	1990-06-30

```
9 rows in set (0.00 sec)
```

## 5. Un simple ROLLBACK annule toute les opérations :

```
mysql> rollback;
Query OK, 0 rows affected (0.06 sec)
```

## 6. Faisons un nouveau select

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30

```
4 rows in set (0.00 sec)
```

La transaction a été annulée.

## 7. Appelons a nouveau la transaction

```
mysql> \. E:\DONNEES\UCAO\MYSQL\trans.sql
Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected, 1 warning (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)
```

## 8. Faisons un nouveau select

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30
Ka	Abdou	Dakar	M	1988-03-30
Diop	Ngone	Dakar	F	0000-00-00
Soumare	Ami	Dakar	M	1993-08-30
Sarr	Fatou	Dakar	F	1991-01-30
Diop	Ali	Dakar	M	1990-06-30

```
9 rows in set (0.00 sec)
```

#### 9. Faisons un COMMIT

```
mysql> commit;
```

Query OK, 0 rows affected (0.06 sec)

La transaction à été validée. Un ROLLBACK n'y peut rien maintenant

#### 10. Faisons un ROLLBACK puis un SELECT

```
mysql> ROLLBACK;
```

Query OK, 0 rows affected (0.00 sec)

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30
Ka	Abdou	Dakar	M	1988-03-30
Diop	Ngone	Dakar	F	0000-00-00
Soumare	Ami	Dakar	M	1993-08-30
Sarr	Fatou	Dakar	F	1991-01-30
Diop	Ali	Dakar	M	1990-06-30

```
9 rows in set (0.00 sec)
```

#### NB :Démarrer explicitement une transaction

En désactivant le mode autocommit, en réalité, on démarre une transaction. Et chaque fois que l'on fait un rollback ou un commit (ce qui met fin à la transaction), une nouvelle transaction est créée automatiquement, et ce tant que la session est ouverte.

Il est également possible de démarrer explicitement une transaction, auquel cas on peut laisser le mode autocommit activé, et décider au cas par cas des requêtes qui doivent être faites dans une transaction.

Pour Repasser donc en mode autocommit :

```
SET autocommit=1;
```

Pour démarrer une transaction, il suffit de lancer la commande suivante :

```
START TRANSACTION;
```

Avec MySQL, il est également possible de démarrer une transaction avec BEGIN ou BEGIN WORK. Cependant, il est conseillé d'utiliser plutôt START TRANSACTION, car il s'agit de la commande SQL standard.

Une fois la transaction ouverte, les requêtes devront être validées pour prendre effet. Attention au fait qu'un COMMIT ou un ROLLBACK met fin automatiquement à la transaction, donc les commandes suivantes seront à nouveau committées automatiquement si une nouvelle transaction n'est pas ouverte.

### **Jalon de transaction : SAVE POINT**

Lorsque l'on travaille dans une transaction, et que l'on constate que certaines requêtes posent problème, on n'a pas toujours envie de faire un rollback depuis le début de la transaction, annulant toutes les requêtes alors qu'une partie aurait pu être validée.

Il n'est pas possible de démarrer une transaction à l'intérieur d'une transaction. Par contre, on peut poser des jalons de transaction. Il s'agit de points de repère, qui permettent d'annuler toutes les requêtes exécutées depuis ce jalon, et non toutes les requêtes de la transaction.

Trois nouvelles commandes suffisent pour pouvoir utiliser pleinement les jalons :

```
SAVEPOINT nom_jalon; -- Crée un jalon avec comme nom "nom_jalon"
```

```
ROLLBACK [WORK] TO [SAVEPOINT] nom_jalon; -- Annule les requêtes exécutées depuis le jalon "nom_jalon", WORK et SAVEPOINT ne sont pas obligatoires
```

```
RELEASE SAVEPOINT nom_jalon; -- Retire le jalon "nom_jalon" (sans annuler, ni valider les requêtes faites depuis)
```

### **Validation implicite et commandes non-annulables**

Vous savez déjà que pour terminer une transaction, il faut utiliser les commandes COMMIT ou ROLLBACK, selon que l'on veut valider les requêtes ou les annuler. Un certain nombre d'autres commandes auront aussi pour effet de clôturer une transaction. Et pas seulement la clôturer, mais également valider toutes les requêtes qui ont été faites dans cette transaction. Exactement comme si vous utilisiez COMMIT.

Par ailleurs, ces commandes ne peuvent pas être annulées par un ROLLBACK.

Ce sont les Commandes DDL

Toutes les commandes qui créent, modifient, suppriment des objets dans la base de données valident implicitement les transactions. Ces commandes forment ce qu'on appelle les requêtes DDL, pour Data Definition Language.

Cela comprend donc :

- la création et suppression de bases de données : CREATE DATABASE, DROP DATABASE ;
- la création, modification, suppression de tables : CREATE TABLE, ALTER TABLE, RENAME TABLE, DROP TABLE ;
- la création, modification, suppression d'index : CREATE INDEX, DROP INDEX ;
- la création d'objets comme les procédures stockées, les vues, etc., dont nous parlerons plus tard.

## Une transaction doit être **ACID**

Il a été défini que ces critères sont au nombre de quatre : Atomicité, Cohérence, Isolation et Durabilité. Soit, si on prend la première lettre de chaque critère : ACID.

- **A pour Atomicité**

Atome signifie étymologiquement "qui ne peut être divisé". Une transaction doit être atomique, c'est-à-dire qu'elle doit former une entité complète et indivisible. Chaque élément de la transaction, chaque requête effectuée, ne peut exister que dans la transaction. Si l'on reprend l'exemple du virement bancaire, en utilisant les transactions, les deux étapes (débit du compte donneur d'ordre, crédit du compte bénéficiaire) ne peuvent exister indépendamment l'une de l'autre. Si l'une est exécutée, l'autre doit l'être également. Il s'agit d'un tout.

Peut-on dire que nos transactions sont atomiques ?

Oui. Si une transaction en cours est interrompue, aucune des requêtes exécutées ne sera validée. De même, en cas d'erreur, il suffit de faire un ROLLBACK pour annuler toute la transaction. Et si tout se passe bien, un COMMIT validera l'intégralité de la transaction en une fois.

- **C pour cohérence**

Les données doivent rester cohérentes dans tous les cas : que la transaction se termine sans encombre, qu'une erreur survienne, ou que la transaction soit interrompue. Un virement dont seule l'étape de débit du donneur d'ordre est exécutée produit des données incohérentes (la disparition de 300 euros jamais arrivés chez le bénéficiaire). Avec une transaction, cette incohérence n'apparaît jamais. Tant que la totalité des étapes n'a pas été réalisée avec succès, les données restent dans leur état initial.

Nos transactions permettent-elles d'assurer la cohérence des données ?

Oui, les changements de données ne sont validés qu'une fois que toutes les étapes ont été exécutées. De l'extérieur de la transaction, le moment entre les deux étapes d'un virement n'est jamais visible.

- **I pour Isolation**

Chaque transaction doit être isolée, donc ne pas interagir avec une autre transaction. Lorsqu'une transaction accède en écriture sur une donnée A, alors un verrou est posé sur cette donnée empêchant ainsi une autre transaction d'y accéder elle aussi en écriture. Ce blocage a pour effet d'empêcher la deuxième session d'écraser un changement fait par la première. Donc, ce blocage a bien pour effet l'isolation des transactions.

Exemple :

- Root supprime l'étudiant Ka sans faire un COMMIT ;

```
mysql> delete from etudiant where nom='Ka';  
Query OK, 1 row affected (0.00 sec)  
mysql>
```

- Edou modifie le même étudiant

Résultat : la session de Edou est bloquée jusqu'à ce que root termine.

```
mysql> update etudiant set prenom='Amadou nar' where nom='ka';
```

Si root tarde à faire un commit, alors la transaction de Edou est annulée.

```
mysql> update etudiant set prenom='Amadou nar' where nom='ka';  
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction  
mysql>  
mysql>  
mysql>
```

- **D pour Durabilité**

Une fois la transaction terminée, les données résultant de cette transaction doivent être stockées de manière durable, et pouvoir être récupérées, en cas de crash du serveur par exemple. Nos transactions modifient-elles les données de manière durable ?

Oui, une fois les changements commités, ils sont stockés définitivement (jusqu'à modification par une nouvelle transaction).

## **PARTIE 6 : LES VERROUS**



Complément indispensable des transactions, les verrous permettent de sécuriser les requêtes en bloquant ponctuellement et partiellement l'accès aux données. Lorsqu'une session MySQL pose un verrou sur un élément de la base de données, cela veut dire qu'il restreint, voire interdit, l'accès à cet élément aux autres sessions MySQL qui voudraient y accéder.

Il est possible de poser un verrou sur une table entière, ou seulement sur une ou plusieurs lignes d'une table.

Étant donné qu'un verrou empêche l'accès d'autres sessions, il est en général plus intéressant de poser un verrou sur la plus petite partie de la base possible.

Par exemple, si l'on travaille avec les étudiants de Dakar de la table Etudiant.

- On peut poser un verrou sur toute la table Etudiant. Dans ce cas, les autres sessions n'auront pas accès à cette table, tant que le verrou sera posé. Qu'elles veuillent en utiliser les étudiants de Mbour ou autre, tout leur sera refusé.
- On peut aussi poser un verrou uniquement sur les lignes de la table qui contiennent des étudiants de Dakar. De cette manière, les autres sessions pourront accéder aux étudiants de Mbour, à ceux de Fatick, etc. Elles pourront toujours travailler, tant qu'elles n'utilisent pas les étudiants de Dakar.

Cette notion d'accès simultané aux données par plusieurs sessions différentes s'appelle la concurrence. Plus la concurrence n'est possible, donc plus le nombre de sessions pouvant accéder aux données simultanément est grand, mieux c'est. En effet, prenons l'exemple d'un site web. En général, on préfère permettre à plusieurs utilisateurs de surfer en même temps, sans devoir attendre entre chaque action de pouvoir accéder aux informations chacun à son tour.

Or, chaque utilisateur crée une session chaque fois qu'il se connecte à la base de données (pour lire les informations ou les modifier). Préférez donc (autant que possible) les verrous de ligne aux verrous de table !

#### Avertissements

En effet, les verrous sont implémentés différemment selon les SGDB, et même selon le moteur de table en ce qui concerne MySQL. Si le principe général reste toujours le même, certains comportements et certaines options peuvent différer d'une implémentation à l'autre.

### **VERROUS DE TABLE**

Les verrous de table sont les seuls supportés par MyISAM. Ils sont d'ailleurs principalement utilisés pour pallier en partie l'absence de transactions dans MyISAM.

Les tables InnoDB peuvent également utiliser ce type de verrou. Pour verrouiller une table, il faut utiliser la commande LOCK TABLES :

**LOCK TABLES nom\_table [AS alias\_table] [READ | WRITE] [, ...];**

- En utilisant READ, un verrou de lecture sera posé ; c'est-à-dire que les autres sessions pourront toujours lire les données des tables verrouillées, mais ne pourront plus les modifier.
- En utilisant WRITE, un verrou d'écriture sera posé. Les autres sessions ne pourront plus ni lire ni modifier les données des tables verrouillées.

Pour déverrouiller les tables, on utilise UNLOCK TABLES. Cela déverrouille toutes les tables verrouillées. Il n'est pas possible de préciser les tables à déverrouiller. Tous les verrous de table d'une session sont relâchés en même temps.

```
mysql> UNLOCK TABLES;  
Query OK, 0 rows affected (0.00 sec)
```

### Session ayant obtenu le verrou

Lorsqu'une session acquiert un ou plusieurs verrous de table, cela a plusieurs conséquences pour cette session :

- elle ne peut plus accéder qu'aux tables sur lesquelles elle a posé un verrou ;
- elle ne peut accéder à ces tables qu'en utilisant les noms qu'elle a donnés lors du verrouillage (soit le nom de la table, soit le/les alias donné(s)) ; s'il s'agit d'un verrou de lecture (READ), elle peut uniquement lire les données, pas les modifier.

Exemples : Soit la base de données LPIG avec un user lpig.

Soient les trois tables Etudiants et professeurs et classes

```
mysql> drop table Etudiant;  
Query OK, 0 rows affected (0.02 sec)  
  
mysql> CREATE TABLE Etudiant (  
-> nom VARCHAR(20),  
-> prenom VARCHAR(20),  
-> adresse VARCHAR(20),  
-> sexe CHAR(1),  
-> naissance DATE  
-> )  
-> ENGINE = INNODB;  
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> create table professeurs(
-> prenom varchar(55),
-> nom varchar(66),
-> salaire int) engine INNODB;
Query OK, 0 rows affected (0.08 sec)

mysql> insert into professeurs values ('pape','sarr',200000);
Query OK, 1 row affected (0.05 sec)

mysql> insert into professeurs values ('aly','diouf',300000);
Query OK, 1 row affected (0.02 sec)

mysql> insert into professeurs values ('ami','Fall',600000);
Query OK, 1 row affected (0.17 sec)
```

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SARR	Edouard	Mbour	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30
Diop	Ngone	Dakar	F	0000-00-00
Soumare	anna marie	Dakar	M	1993-08-30
Sarr	Fatou	Dakar	F	1991-01-30
Diop	Ali	Dakar	M	1990-06-30
Ali	Ka	Dakar	M	2012-12-05
Astou	Lo	Dakar	M	1995-12-05
Tendeng	Vero	Dakar	M	1993-12-05

```
11 rows in set (0.00 sec)
```

```
mysql> select * from professeurs;
```

prenom	nom	salaire
pape	sarr	200000
aly	diouf	300000
ami	Fall	600000

```
3 rows in set (0.00 sec)
```

```
mysql> create table classes (nom_classe varchar(55));
Query OK, 0 rows affected (0.06 sec)

mysql> insert into classes values ('LPIG1'),('LPIG2'),('LPIG3');
Query OK, 3 rows affected (0.02 sec)
Records: 3 Duplicates: 0 Warnings: 0

mysql> select * from classes;
```

nom_classe
LPIG1
LPIG2
LPIG3

```
3 rows in set (0.00 sec)
```

Mais faisons un SHOW TABLES

```
mysql> show tables;
+-----+
| Tables_in_lpig |
+-----+
| classes        |
| etudiant       |
| professeurs    |
+-----+
3 rows in set (0.00 sec)
```

Les trois tables sont visibles. Insérons dans la table classe.

```
mysql> insert into classes values ('MPIG1');
Query OK, 1 row affected (0.00 sec)
```

Ca passe car il n'y a pas encore de verrous.

On va poser deux verrous, l'un READ, l'autre WRITE, l'un en donnant un alias au nom de la table, l'autre sans.

READ : Lire mais ne pas écrire

WRITE : écrire mais ne pas lire

- **LOCK TABLES Etudiants READ ;** -- On pose un verrou de lecture sur etudiant

Vous pouvez lire mais ne pas modifier.

User1 :

```
mysql> LOCK TABLES Etudiant READ ;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from etudiant;
+-----+-----+-----+-----+-----+
| nom    | prenom | adresse | sexe | naissance |
+-----+-----+-----+-----+-----+
| Diop   | Ali    | Dakar   | M    | 1990-06-30 |
| SARR   | Edouard | Ngohe   | M    | 1999-03-30 |
| SAGNE  | Anissa | Dakar   | M    | 1999-03-30 |
| Tall   | Abdou  | Dakar   | M    | 1999-03-30 |
| Diop   | Ngone  | Dakar   | F    | 0000-00-00 |
| Soumare | anna marie | Dakar   | M    | 1993-08-30 |
| Sarr   | Fatou  | Ngohe   | F    | 1991-01-30 |
| Diop   | Ali    | Dakar   | M    | 1990-06-30 |
| Ali    | Ka     | Dakar   | M    | 2012-12-05 |
| Astou  | Lo     | Dakar   | M    | 1995-12-05 |
| Tendeng | Vero   | Dakar   | M    | 1993-12-05 |
+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)
```

User2 :

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SARR	Edouard	Ngohe	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30
Diop	Ngone	Dakar	F	0000-00-00
Soumare	anna marie	Dakar	M	1993-08-30
Sarr	Fatou	Ngohe	F	1991-01-30
Diop	Ali	Dakar	M	1990-06-30
Ali	Ka	Dakar	M	2012-12-05
Astou	Lo	Dakar	M	1995-12-05
Tendeng	Vero	Dakar	M	1993-12-05

```
11 rows in set (0.00 sec)
```

Il tente d'écrire :

```
mysql> delete from etudiant where nom='Ali';
```

Il est bloqué.

- **LOCK TABLES Etudiants READ ; -- et un verrou d'écriture**

User 1 :

```
mysql> LOCK TABLES Etudiant WRITE;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from etudiant;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SARR	Edouard	Ngohe	M	1999-03-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30
Diop	Ngone	Dakar	F	0000-00-00
Soumare	anna marie	Dakar	M	1993-08-30
Sarr	Fatou	Ngohe	F	1991-01-30
Diop	Ali	Dakar	M	1990-06-30
Astou	Lo	Dakar	M	1995-12-05
Tendeng	Vero	Dakar	M	1993-12-05

```
10 rows in set (0.00 sec)
```

User 2

```
mysql> select * from etudiant;
```

```
mysql> delete from etudiant where nom='Ali';
```

NB : Un user qui pose un verrou ne peut plus accéder aux autres tables de la base de données avec sa session.

Insérons dans la table classes qui ne fait pas partie du verrou.

```
mysql> insert into classes values ('MPIG1');  
ERROR 1100 (HY000): Table 'classes' was not locked with LOCK TABLES  
mysql>
```

Impossible car le user ne peut travailler que sur les tables en verrou.

## LES VERROUS DE TABLE

Comme les verrous de table, les verrous de ligne peuvent être de deux types :

- Les verrous partagés : permettent aux autres sessions de lire les données, mais pas de les modifier (équivalents aux verrous de table de lecture) ;
- Les verrous exclusifs : ne permettent ni la lecture ni la modification des données (équivalents aux verrous d'écriture).

## Requêtes de modification, insertion et suppression

Les requêtes de modification et suppression des données posent automatiquement un verrou exclusif sur les lignes concernées, à savoir les lignes sélectionnées par la clause WHERE, ou toutes les lignes s'il n'y a pas de clause WHERE (ou s'il n'y a pas d'index, sur les colonnes utilisées comme nous verrons plus loin).

Les requêtes d'insertion quant à elles posent un verrou exclusif sur la ligne insérée.

## Requêtes de sélection

Les requêtes de sélection, par défaut, ne posent pas de verrous. Il faut donc en poser explicitement au besoin.

## Verrou partagé

Pour poser un verrou partagé, on utilise LOCK IN SHARE MODE à la fin de la requête SELECT. Cela signifie que je suis entrainé de lire, lisez mais ne modifier pas.

Exemple :

User1:

```
mysql> SELECT * FROM etudiant WHERE nom = 'sarr' LOCK IN SHARE MODE;
```

nom	prenom	adresse	sexe	naissance
SARR	Edouard	Ngohe	M	1999-03-30
Sarr	Fatou	Ngohe	F	1991-01-30

```
2 rows in set (0.00 sec)
```

Cette requête pose donc un verrou partagé sur les lignes de la table Etudiant pour lesquelles le nom vaut sarr. Ce verrou signifie en fait, pour les autres sessions : "Je suis en train de lire ces données. Vous pouvez venir les lire aussi, mais pas les modifier tant que je n'ai pas terminé."

### Verrou exclusif

Pour poser un verrou exclusif, on utilise FOR UPDATE à la fin de la requête SELECT.

User 1. :

```
mysql> select * from etudiant FOR UPDATE;
```

nom	prenom	adresse	sexe	naissance
Diop	Ali	Dakar	M	1990-06-30
SAGNE	Anissa	Dakar	M	1999-03-30
Tall	Abdou	Dakar	M	1999-03-30
Diop	Ngone	Dakar	F	0000-00-00
Diop	Ali	Dakar	M	1990-06-30
Astou	Lo	Dakar	M	1995-12-05
Tendeng	Vero	Dakar	M	1993-12-05

```
7 rows in set (0.00 sec)
```

Ce verrou signifie aux autres sessions : "Je suis en train de lire ces données dans le but probable de faire une modification. Ne les lisez pas avant que j'ai fini (et bien sûr, ne les modifiez pas).".

User 2 : Modifie et est bloqué

```
mysql> delete from etudiant where nom='Tall';
```

## **PARTIE 7 : LES TRIGGERS**



Les triggers (ou déclencheurs) sont des objets de la base de données. Attachés à une table, ils vont déclencher l'exécution d'une instruction, ou d'un bloc d'instructions, lorsqu'une, ou plusieurs lignes sont insérées, supprimées ou modifiées dans la table à laquelle ils sont attachés.

Tout comme les procédures stockées, les triggers servent à exécuter une ou plusieurs instructions. Mais à la différence des procédures, il n'est pas possible d'appeler un trigger : un trigger doit être déclenché par un événement.

Un trigger est attaché à une table, et peut être déclenché par :

- une insertion dans la table (requête INSERT) ;
- la suppression d'une partie des données de la table (requête DELETE) ;
- la modification d'une partie des données de la table (requête UPDATE).

Par ailleurs, une fois le trigger déclenché, ses instructions peuvent être exécutées soit juste avant l'exécution de l'événement déclencheur, soit juste après.

Un trigger exécute un traitement pour chaque ligne insérée, modifiée ou supprimée par l'événement déclencheur. Donc si l'on insère cinq lignes, les instructions du trigger seront exécutées cinq fois, chaque itération permettant de traiter les données d'une des lignes insérées.

Un trigger peut modifier et/ou insérer des données dans n'importe quelle table sauf les tables utilisées dans la requête qui l'a déclenché. En ce qui concerne la table à laquelle le trigger est attaché (qui est forcément utilisée par l'événement déclencheur), le trigger peut lire et modifier uniquement la ligne insérée, modifiée ou supprimée qu'il est en train de traiter.

Avec des triggers se déclenchant avant l'INSERT et avant l'UPDATE, on peut vérifier les valeurs d'une colonne lors de l'insertion ou de la modification, et les corriger si elles ne font pas partie des valeurs acceptables, ou bien faire échouer la requête. On peut ainsi pallier l'absence de contraintes d'assertion.

Les triggers sont parfois utilisés pour remplacer les options des clés étrangères ON UPDATE RESTRICT|CASCADE|SET NULL et ON DELETE RESTRICT|CASCADE|SET NULL. Notamment pour des tables MyISAM, qui sont nontransactionnelles et ne supportent pas les clés étrangères.

Cela peut aussi être utilisé avec des tables transactionnelles, dans les cas où le traitement à appliquer pour garder des données cohérentes est plus complexe que ce qui est permis par les options de clés étrangères.

Par exemple, dans certains systèmes, on veut pouvoir appliquer deux systèmes de suppression :

- une vraie suppression pure et dure, avec effacement des données, donc une requête DELETE ;
- un archivage, qui masquera les données dans l'application mais les conservera dans la base de données.

Dans ce cas, une solution possible est d'ajouter aux tables contenant des données archivables une colonne archive, pouvant contenir 0 (la ligne n'est pas archivée) ou 1 (la ligne est archivée). Pour une vraie suppression, on peut utiliser simplement un `ON DELETE RESTRICT|CASCADE|SET NULL`, qui se répercutera sur les tables référençant les données supprimées.

Par contre, dans le cas d'un archivage, on utilisera plutôt un trigger pour traiter les lignes qui référencent les données archivées, par exemple en les archivant également.

### **Historisation des actions**

On veut parfois garder une trace des actions effectuées sur la base de données, c'est-à-dire par exemple, savoir qui a modifié telle ligne, et quand. Avec les triggers, rien de plus simple, il suffit de mettre à jour des données d'historisation à chaque insertion, modification ou suppression. Soit directement dans la table concernée, soit dans une table utilisée spécialement et exclusivement pour garder un historique des actions.

### **Création des triggers**

Pour créer un trigger, on utilise la commande suivante :

**CREATE TRIGGER nom\_trigger moment\_trigger evenement\_trigger**

**ON nom\_table FOR EACH ROW**

**corps\_trigger**

- `CREATE TRIGGER nom_trigger` : les triggers ont donc un nom.
- `moment_trigger evenement_trigger` : servent à définir quand et comment le trigger est déclenché.
- `ON nom_table` : c'est là qu'on définit à quelle table le trigger est attaché.
- `FOR EACH ROW` : signifie littéralement "pour chaque ligne", sous-entendu "pour chaque ligne insérée/supprimée/modifiée" selon ce qui a déclenché le trigger.
- `corps_trigger` : c'est le contenu du trigger. Comme pour les procédures stockées, il peut s'agir soit d'une seule instruction, soit d'un bloc d'instructions.

### **Événement déclencheur**

Trois événements différents peuvent déclencher l'exécution des instructions d'un trigger.

- L'insertion de lignes (INSERT) dans la table attachée au trigger.
- La modification de lignes (UPDATE) de cette table.
- La suppression de lignes (DELETE) de la table.

Un trigger est soit déclenché par INSERT, soit par UPDATE, soit par DELETE. Il ne peut pas être déclenché par deux événements différents. On peut par contre créer plusieurs triggers par table pour couvrir chaque événement.

### Avant ou après

Lorsqu'un trigger est déclenché, ses instructions peuvent être exécutées à deux moments différents. Soit juste avant que l'événement déclencheur n'ait lieu (BEFORE), soit juste après (AFTER). Donc, si vous avez un trigger BEFORE UPDATE sur la table A, l'exécution d'une requête UPDATE sur cette table va d'abord déclencher l'exécution des instructions du trigger, ensuite seulement les lignes de la table seront modifiées.

Exemple :

Pour créer un trigger sur la table Professeurs, déclenché par une insertion, et s'exécutant après ladite insertion, on utilisera la syntaxe suivante :

```
CREATE TRIGGER after_insert_Professeurs  
AFTER INSERT ON Professeurs FOR EACH ROW  
corps_trigger;
```

### Règle et convention

- Il ne peut exister qu'un seul trigger par combinaison moment\_trigger/événement\_trigger par table. Donc un seul trigger BEFORE UPDATE par table, un seul AFTER DELETE, etc.
- Étant donné qu'il existe deux possibilités pour le moment d'exécution, et trois pour l'événement déclencheur, on a donc un maximum de six triggers par table.

Cette règle étant établie, il existe une convention quant à la manière de nommer ses triggers, que je vous encourage à suivre :

nom\_trigger = moment\_evenement\_table.

Donc le trigger BEFORE UPDATE ON Professeurs aura pour nom :

before\_update\_ptofesseurs.

## OLD et NEW

Dans le corps du trigger, MySQL met à disposition deux mots-clés : OLD et NEW.

- OLD : représente les valeurs des colonnes de la ligne traitée avant qu'elle ne soit modifiée par l'événement déclencheur. Ces valeurs peuvent être lues, mais pas modifiées.
- NEW : représente les valeurs des colonnes de la ligne traitée après qu'elle a été modifiée par l'événement déclencheur. Ces valeurs peuvent être lues et modifiées.

Il n'y a que dans le cas d'un trigger UPDATE que OLD et NEW coexistent. Lors d'une insertion, OLD n'existe pas, puisque la ligne n'existe pas avant l'événement déclencheur ; dans le cas d'une suppression, c'est NEW qui n'existe pas, puisque la ligne n'existera plus après l'événement déclencheur.

### Exemple:

Créons la table archiveProfesseurs qui va contenir les anciennes valeurs de professeurs supprimées ou modifier

```
mysql> create table archiveProfesseurs (  
-> prenom varchar(66),  
-> nom varchar(66),  
-> salaire int  
-> );  
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> show tables;  
+-----+  
| Tables_in_lpig |  
+-----+  
| archiveprofesseurs |  
| classes |  
| etudiant |  
| professeurs |  
+-----+  
4 rows in set (0.00 sec)
```

- **Trigger delete sur professeurs.**

```
DELIMITER |  
CREATE TRIGGER after_Delete_Professeurs  
AFTER DELETE ON Professeurs FOR EACH ROW  
BEGIN  
  Insert Into archiveProfesseurs(prenom,nom,salaire)  
  values(OLD.prenom,OLD.nom,OLD.salaire);  
END |  
DELIMITER ;
```

Enregistrons le dans le D sous le nom trigger, moi par contre je vois le mettre dans E:\DONNEES\UCAO\MYSQL

Sous root, dans notre base de données LPIG, appelons le fichier avec \.

```
mysql> use lpig
Database changed
mysql> \. E:\DONNEES\UCAO\MYSQL\trigger.sql
Query OK, 0 rows affected (0.09 sec)
```

Testons un DELETE dans notre table professeurs:

```
mysql> delete from professeurs where nom='fall';
Query OK, 1 row affected (0.08 sec)

mysql> select * from professeurs;
+-----+-----+-----+
| prenom | nom   | salaire |
+-----+-----+-----+
| pape   | sarr  | 200000  |
| aly    | diouf | 200000  |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

Allons verifier dans la table archiveProfesseurs avec un SELECT

```
mysql> select * from archiveprofesseurs;
+-----+-----+-----+
| prenom | nom   | salaire |
+-----+-----+-----+
| ami    | Fall | 200000  |
+-----+-----+-----+
1 row in set (0.02 sec)
```

Vidons maintenant la table Professeurs;

```
mysql> delete from professeurs;
Query OK, 2 rows affected (0.06 sec)

mysql> select * from professeurs;
Empty set (0.00 sec)
```

Vérifions à nouveau la table archiveProfesseurs avec un SELECT

```
mysql> select * from archiveprofesseurs;
+-----+-----+-----+
| prenom | nom   | salaire |
+-----+-----+-----+
| ami    | Fall | 200000  |
| pape   | sarr  | 200000  |
| aly    | diouf | 200000  |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

**Partie 8 :**  
**Chargement de données externes**  
**LOAD DATA LOCAL**

C'est un outil permettant l'importation de données en masse depuis un fichier texte sur disque. Pour importer on utilise la commande `Load` en lui spécifiant quelques paramètres :

- Le chemin du fichier : `LOCAL INFILE`
- Le nom du fichier
- La table : `INTO TABLE`
- Le séparateur de colonne : `FIELDS TERMINATED BY`

SYNTAXE :

**LOAD DATA LOCAL INFILE 'Chemin/nomchier' INTO TABLE  
NomTable FIELDS TERMINATED BY "," (Colonne1,Colonne2,...);**

EXEMPLE 1 : soit le fichier externe `load.txt`

```
Ali,Ka,Dakar,M,2012/12/05
Astou,Lo,Dakar,M,1995/12/05
Tendeng,Vero,Dakar,M,1993/12/05
```

**LOAD DATA LOCAL INFILE 'D:/etudiant.txt' INTO TABLE etudiant FIELDS  
TERMINATED BY "," (nom,prenom,adresse,sexe,naissance);**

```
mysql> LOAD DATA LOCAL INFILE 'D:/etudiant.txt' INTO TABLE etudiant FIELDS
nom,prenom,adresse,sexe,naissance);
Query OK, 3 rows affected (0.06 sec)
Records: 3 Deleted: 0 Skipped: 0 Warnings: 0

mysql> select * from etudiant;
+----+-----+-----+-----+-----+
| nom | prenom | adresse | sexe | naissance |
+----+-----+-----+-----+-----+
| Diop | Ali | Dakar | M | 1990-06-30 |
| SARR | Edouard | Mbour | M | 1999-03-30 |
| SAGNE | Anissa | Dakar | M | 1999-03-30 |
| Tall | Abdou | Dakar | M | 1999-03-30 |
| Diop | Ngone | Dakar | F | 0000-00-00 |
| Soumare | anna marie | Dakar | M | 1993-08-30 |
| Sarr | Fatou | Dakar | F | 1991-01-30 |
| Diop | Ali | Dakar | M | 1990-06-30 |
| Ali | Ka | Dakar | M | 2012-12-05 |
| Astou | Lo | Dakar | M | 1995-12-05 |
| Tendeng | Vero | Dakar | M | 1993-12-05 |
+----+-----+-----+-----+-----+
11 rows in set (0.02 sec)

mysql>
```

EXEMPLE:

Soient les fichiers externes suivants :

**Clients.txt**

165566,Edouard ngor ,sarr, Dakar,779808833  
 265566,pape,diop, Dakar,78665544  
 365566,amie ,fall, Fatick,778865544  
 465566,Petit,diop, Mbour,764445544  
 565566,Babacar,ly,touba,705566666

**Clients2.txt**

106556;ami ngor ;sarr; Dakar;779808833  
 20655;pape;diop; Dakar;78665544  
 3055;amie ;fall; Fatick;778865544  
 405;Petit;diop; Mbour;764445544  
 5055;ali;ly;touba;705566666

**Clients3.txt**

0106556/anna ngor /sarr/ Dakar/779808833  
 020655/fatou/diop/ Dakar/78665544  
 03055/edou /fall/ Fatick/778865544  
 0405/ngor/diop/ Mbour/764445544  
 05055/malick/ly/touba/705566666

**Clients4.txt**

20106556--elisa--sarr-- Dakar--779808833  
 3020655--dame--diop-- Dakar--78665544  
 403055--anissa --fall-- Fatick--778865544  
 20405--helene--diop-- Mbour--764445544  
 205055--pape--ly--touba—705566666

Sous Mysql :

```
mysql> create table Temployes (
```

```
-> Num int,  
-> prenom varchar(44),  
-> nom varchar(66),  
-> Adresse varchar(66),  
-> telephone varchar(66)  
-> );
```

Query OK, 0 rows affected (0.06 sec)

```
mysql> LOAD DATA LOCAL INFILE 'E:/DONNEES/UCAO/MYSQL/clients.txt' INTO  
TABLE Temployes FIELDS TE  
RMINATED BY "," (Num,prenom,nom,adresse,telephone);
```

Query OK, 5 rows affected (0.06 sec)

Records: 5 Deleted: 0 Skipped: 0 Warnings: 0

```
mysql> select * from temployes;
```

```
+-----+-----+-----+-----+  
| Num   | prenom   | nom    | Adresse | telephone |
```



```
+-----+-----+-----+-----+
| 65566 | Edouard ngor | sarr | Dakar | 779808833
| 5566 | pape      | diop | Dakar | 78665544
| 65566 | amie       | fall | Fatick | 778865544
| 65566 | Petit     | diop | Mbour | 764445544
| 65566 | Babacar   | ly   | touba | 705566666
+-----+-----+-----+-----+
```

5 rows in set (0.00 sec)

```
mysql> LOAD DATA LOCAL INFILE 'E:/DONNEES/UCAO/MYSQL/clients2.txt' INTO
TABLE Temployes FIELDS T
ERMINATED BY ";" (Num,prenom,nom,adresse,telephone);
```

Query OK, 5 rows affected (0.00 sec)

Records: 5 Deleted: 0 Skipped: 0 Warnings: 0

```
mysql> select * from temployes;
```

```
+-----+-----+-----+-----+
| Num  | prenom   | nom  | Adresse | telephone |
+-----+-----+-----+-----+
| 65566 | Edouard ngor | sarr | Dakar | 779808833
| 5566 | pape      | diop | Dakar | 78665544
| 65566 | amie       | fall | Fatick | 778865544
| 65566 | Petit     | diop | Mbour | 764445544
| 65566 | Babacar   | ly   | touba | 705566666
| 06556 | ami ngor   | sarr | Dakar | 779808833
| 0655 | pape      | diop | Dakar | 78665544
| 3055 | amie       | fall | Fatick | 778865544
| 405 | Petit     | diop | Mbour | 764445544
| 5055 | ali       | ly   | touba | 705566666
+-----+-----+-----+-----+
```

10 rows in set (0.00 sec)

```
mysql> LOAD DATA LOCAL INFILE 'E:/DONNEES/UCAO/MYSQL/clients3.txt' INTO
TABLE Temployes FIELDS T
ERMINATED BY "/" (Num,prenom,nom,adresse,telephone);
```

Query OK, 5 rows affected (0.00 sec)

Records: 5 Deleted: 0 Skipped: 0 Warnings: 0

```
mysql> select * from temployes;
```

```
+-----+-----+-----+-----+
| Num  | prenom   | nom  | Adresse | telephone |
+-----+-----+-----+-----+
| 65566 | Edouard ngor | sarr | Dakar | 779808833
| 5566 | pape      | diop | Dakar | 78665544
| 65566 | amie       | fall | Fatick | 778865544
| 65566 | Petit     | diop | Mbour | 764445544
| 65566 | Babacar   | ly   | touba | 705566666
| 06556 | ami ngor   | sarr | Dakar | 779808833
| 0655 | pape      | diop | Dakar | 78665544
```

```
| 3055 | amie      | fall | Fatick | 778865544
| 405 | Petit      | diop | Mbour  | 764445544
| 5055 | ali        | ly   | touba  | 705566666
| 06556 | anna ngor | sarr | Dakar  | 779808833
| 0655 | fatou      | diop | Dakar  | 78665544
| 3055 | edou       | fall | Fatick | 778865544
| 405 | ngor       | diop | Mbour  | 764445544
| 5055 | malick     | ly   | touba  | 705566666
+-----+-----+-----+-----+
15 rows in set (0.00 sec)
```

```
mysql> LOAD DATA LOCAL INFILE 'E:/DONNEES/UCAO/MYSQL/clients4.txt' INTO
TABLE Temployes FIELDS T
ERMINATED BY "--" (Num,prenom,nom,adresse,telephone);
```

Query OK, 5 rows affected (0.00 sec)

Records: 5 Deleted: 0 Skipped: 0 Warnings: 0

```
mysql> select * from temployes;
```

```
+-----+-----+-----+-----+-----+
| Num   | prenom    | nom  | Adresse | telephone |
+-----+-----+-----+-----+-----+
| 165566 | Edouard  | ngor | sarr    | Dakar      | 779808833
| 265566 | pape     | diop | Dakar   | 78665544
| 365566 | amie     | fall | Fatick  | 778865544
| 465566 | Petit    | diop | Mbour   | 764445544
| 565566 | Babacar  | ly   | touba   | 705566666
| 106556 | ami ngor | sarr | Dakar   | 779808833
| 20655 | pape     | diop | Dakar   | 78665544
| 3055 | amie     | fall | Fatick  | 778865544
| 405 | Petit    | diop | Mbour   | 764445544
| 5055 | ali      | ly   | touba   | 705566666
| 106556 | anna ngor | sarr | Dakar   | 779808833
| 20655 | fatou    | diop | Dakar   | 78665544
| 3055 | edou     | fall | Fatick  | 778865544
| 405 | ngor     | diop | Mbour   | 764445544
| 5055 | malick   | ly   | touba   | 705566666
| 0106556 | elisa   | sarr | Dakar   | 779808833
| 020655 | dame    | diop | Dakar   | 78665544
| 403055 | anissa  | fall | Fatick  | 778865544
| 20405 | helene   | diop | Mbour   | 764445544
| 205055 | pape    | ly   | touba   | 705566666
+-----+-----+-----+-----+
20 rows in set (0.00 sec)
```

```
mysql>
```

## **PARTIE 9**

### **IMPORT / EXPORT**

Les Systèmes de Gestion de Bases de Données tels que MySQL permettent de manipuler facilement et avec beaucoup de souplesse un très important volume de données.

Toutefois, aussi robuste soit MySQL, il peut être intéressant de récupérer l'ensemble des données que contient notre base de données, pour faire une sauvegarde (backup) ou bien tout simplement pour passer à une autre base de données.

On appelle "exportation" le fait de formater dans un fichier (appelé dump) toutes les informations nécessaires à la création d'une base de données identique. À l'inverse, on appelle importation le fait de créer dans un SGBD une nouvelle base de données à partir d'un fichier d'exportation (dump).

MySQL offre un certain nombre d'outils permettant d'exporter ses bases vers d'autres SGBD ou bien de les importer.

- **Exporter une base de données MySQL avec mysqldump**

La commande mysql permet d'exporter l'intégralité d'une base de données hébergée par MySQL de façon efficace mais n'offre pas la souplesse nécessaire à l'exportation de plusieurs bases de données ou au contraire d'une partie de la base de données (table ou partie d'une table).

La commande mysqldump répond à ce besoin en offrant la possibilité de spécifier plus précisément les données à exporter.

Voici la syntaxe de cette commande :

```
Mysqldump [options] base_de_donnees [tables]
```

NB : il n'y a pas de ; à la fin

Voici les options généralement utilisées :

```
mysqldump -h host -u user -ppass -rfichier base_de_donnees [tables]
```

Voici un exemple d'exportation des tables membres et invites de la base nommée UCAO située sur la machine localhost et appartenant à l'utilisateur sarr (dont le mot de passe est sarr) :

Exporter toutes les objets de la base UCAO

```
C:\Windows\System32>mysqldump -h localhost -u sarr -psarr  
-r D:/Sauv.sql UCAO
```

Exporter seulement la table étudiant de la base UCAO:

```
C:\Windows\System32>mysqldump -h localhost -u sarr -psarr  
-r D:/Sauv.sql UCAO Etudiant
```

Exporter seulement les tables étudiant et Cours de la base UCAO:

```
C:\Windows\System32>mysqldump -h localhost -u sarr -psarr  
-r D:/Sauv.sql UCAO Etudiant Cours
```

Pour dumper toutes les bases de données il faut lancer la commande : Ici root n'a pas de mot de passe

```
C:\Windows\System32>mysqldump -h localhost -u root -r D:/SauvAll.sql -A
```

- **Importer une base de données sous MySQL**

La commande en ligne mysql permet également d'importer des données. Il suffit pour cela d'utiliser la redirection < et d'indiquer le fichier dump contenant les instructions SQL à importer. Pour restaurer un dump il suffit de lancer la commande :

↘.

Exemple :

```
Mysql> Create database ucao2;  
Mysql> Use UAO2  
Mysql> \. D:/sauv1.SQL
```

## **PARTIE 10 : INFORMATION\_SCHEMA**

La base de données information\_schema comme son nom l'indique, contient des informations sur les schémas.

En MySQL, un schéma est une base de données. Ce sont des synonymes.

La base information\_schema contient donc des informations sur les bases de données.

Cette définition de "schéma" n'est pas universelle. Dans certains SGBD la notion de schéma est plus proche de celle d'utilisateur que de base de données.

Pour Oracle par exemple, un schéma représente l'ensemble des objets appartenant à un utilisateur.

Voyons ce qu'on trouve comme tables dans cette base de données.

```
C:\Users\edvaldo>mysql -u root

mysql> use information_schema
Database changed

mysql> show tables;
+-----+
| Tables_in_information_schema |
+-----+
| CHARACTER_SETS               |
| COLLATIONS                   |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                     |
| COLUMN_PRIVILEGES            |
| ENGINES                      |
| EVENTS                       |
| FILES                        |
| GLOBAL_STATUS                |
| GLOBAL_VARIABLES             |
| KEY_COLUMN_USAGE             |
| PARTITIONS                   |
| PLUGINS                      |
| PROCESSLIST                  |
| PROFILING                    |
| REFERENTIAL_CONSTRAINTS     |
| ROUTINES                     |
| SCHEMATA                     |
| SCHEMA_PRIVILEGES            |
| SESSION_STATUS               |
| SESSION_VARIABLES            |
| STATISTICS                   |
| TABLES                      |
| TABLE_CONSTRAINTS           |
| TABLE_PRIVILEGES            |
| TRIGGERS                     |
| USER_PRIVILEGES              |
| VIEWS                        |
+-----+
```

Les tables les plus importantes dans cette liste sont:

CHARACTER\_SETS  
 COLUMNS  
 COLUMN\_PRIVILEGES  
 REFERENTIAL\_CONSTRAINTS  
 ROUTINES  
 PROCESSLIST  
 SESSION\_VARIABLES  
 STATISTICS  
 TABLES  
 TABLE\_CONSTRAINTS  
 TABLE\_PRIVILEGES  
 TRIGGERS  
 USER\_PRIVILEGES  
 VIEWS

Le tableau ci-dessus ne reprend qu'une partie des tables d'information\_schema. Cette base contient donc des informations sur les tables, les colonnes, les contraintes, les vues, etc., des bases de données stockées sur le serveur MySQL.

#### 1. Gestion des colonnes :

Elle fait via la table :

##### - COLUMN

```
mysql> describe columns;
```

Field	Type	Null	Key	Default	Extra
TABLE_CATALOG	varchar(512)	YES		NULL	
TABLE_SCHEMA	varchar(64)	NO			
TABLE_NAME	varchar(64)	NO			
COLUMN_NAME	varchar(64)	NO			
ORDINAL_POSITION	bigint(21) unsigned	NO		0	
COLUMN_DEFAULT	longtext	YES		NULL	
IS_NULLABLE	varchar(3)	NO			
DATA_TYPE	varchar(64)	NO			
CHARACTER_MAXIMUM_LENGTH	bigint(21) unsigned	YES		NULL	
CHARACTER_OCTET_LENGTH	bigint(21) unsigned	YES		NULL	
NUMERIC_PRECISION	bigint(21) unsigned	YES		NULL	
NUMERIC_SCALE	bigint(21) unsigned	YES		NULL	
CHARACTER_SET_NAME	varchar(32)	YES		NULL	
COLLATION_NAME	varchar(32)	YES		NULL	
COLUMN_TYPE	longtext	NO		NULL	
COLUMN_KEY	varchar(3)	NO			
EXTRA	varchar(27)	NO			
PRIVILEGES	varchar(80)	NO			
COLUMN_COMMENT	varchar(255)	NO			

Cette table est plus complète car elle liste toutes les informations sur l'ensemble des colonnes de la base avec les privilèges que les users en disposent.

Exemple : listons les noms de colonne, les tables et les types de la base de données Gnotes.



```
mysql> select column_name,table_name,column_type
-> from columns
-> where table_schema='gnotes';
```

column_name	table_name	column_type
annees	annees	int(11)
idclasse	classes	varchar(222)
NomClasse	classes	varchar(45)
Filieres_idFiliere	classes	varchar(222)
annees_annees	classes	int(11)
Departement	departements	varchar(222)
Description	departements	varchar(405)
MatriculeEtudiants	etudiants	varchar(333)
prenom	etudiants	varchar(455)
nom	etudiants	varchar(45)
date_naissance	etudiants	date
lieu_naissance	etudiants	varchar(45)
Telephone	etudiants	varchar(45)
Email	etudiants	varchar(45)
Adresse	etudiants	varchar(45)
Classes_idclasse	etudiants	varchar(222)

Exemple : listons les noms de colonne et les privilèges de la base de données Gnotes.

```
mysql> select column_name,privileges
-> from columns
-> where table_schema='gnotes';
```

column_name	privileges
annees	select,insert,update,references
idclasse	select,insert,update,references
NomClasse	select,insert,update,references
Filieres_idFiliere	select,insert,update,references
annees_annees	select,insert,update,references
Departement	select,insert,update,references
Description	select,insert,update,references
MatriculeEtudiants	select,insert,update,references
prenom	select,insert,update,references
nom	select,insert,update,references
date_naissance	select,insert,update,references
lieu_naissance	select,insert,update,references
Telephone	select,insert,update,references
Email	select,insert,update,references
Adresse	select,insert,update,references
Classes_idclasse	select,insert,update,references
Classes_Filieres_idFiliere	select,insert,update,references

## 2. Gestion des tables :

Consultons la table TABLES.

```
mysql> describe tables;
```

Field	Type	Null	Key	Default	Extra
TABLE_CATALOG	varchar(512)	YES		NULL	
TABLE_SCHEMA	varchar(64)	NO			
TABLE_NAME	varchar(64)	NO			
TABLE_TYPE	varchar(64)	NO			
ENGINE	varchar(64)	YES		NULL	
VERSION	bigint(21) unsigned	YES		NULL	
ROW_FORMAT	varchar(10)	YES		NULL	
TABLE_ROWS	bigint(21) unsigned	YES		NULL	
AVG_ROW_LENGTH	bigint(21) unsigned	YES		NULL	
DATA_LENGTH	bigint(21) unsigned	YES		NULL	
MAX_DATA_LENGTH	bigint(21) unsigned	YES		NULL	
INDEX_LENGTH	bigint(21) unsigned	YES		NULL	
DATA_FREE	bigint(21) unsigned	YES		NULL	
AUTO_INCREMENT	bigint(21) unsigned	YES		NULL	
CREATE_TIME	datetime	YES		NULL	
UPDATE_TIME	datetime	YES		NULL	
CHECK_TIME	datetime	YES		NULL	
TABLE_COLLATION	varchar(32)	YES		NULL	
CHECKSUM	bigint(21) unsigned	YES		NULL	
CREATE_OPTIONS	varchar(255)	YES		NULL	
TABLE_COMMENT	varchar(80)	NO			

Exemple :

```
mysql> select table_name from tables where table_schema='gnotes';
```

table_name
annees
classes
departements
etudiants
evaluations
filieres
jour
login
module
mois
notes
unite_enseignements

12 rows in set (0.00 sec)

### 3. Gestion des Processus ou connexions

Consultons la table PROCESSLIST. Elle nous donne des informations sur les processus des user connectes.

```
mysql> describe processlist;
```

Field	Type	Null	Key	Default	Extra
ID	bigint(4)	NO		0	
USER	varchar(16)	NO			
HOST	varchar(64)	NO			
DB	varchar(64)	YES		NULL	
COMMAND	varchar(16)	NO			
TIME	int(7)	NO		0	
STATE	varchar(64)	YES		NULL	
INFO	longtext	YES		NULL	

8 rows in set (0.02 sec)

```
mysql> select * from processlist;
+-----+-----+-----+-----+-----+-----+-----+
| ID | USER | HOST          | DB          | COMMAND | TIME | STATE |
+-----+-----+-----+-----+-----+-----+-----+
| 6 | sarr | localhost:49168 | test       | Sleep   | 29 |      |
| 5 | ngor | localhost:49167 | NULL      | Sleep   | 56 |      |
| 2 | root | localhost:49164 | information_schema | Query   | 0 | executing |
+-----+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Pour tuer le processus du user sarr on fait un KILL de son ID

```
mysql> kill 6;
Query OK, 0 rows affected (0.00 sec)
```

#### 4. Gestion des contraintes

Avec la table TABLE\_CONSTRAINTS

```
mysql> describe TABLE_CONSTRAINTS;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| CONSTRAINT_CATALOG | varchar(512) | YES | | NULL | |
| CONSTRAINT_SCHEMA | varchar(64) | NO | | | |
| CONSTRAINT_NAME | varchar(64) | NO | | | |
| TABLE_SCHEMA | varchar(64) | NO | | | |
| TABLE_NAME | varchar(64) | NO | | | |
| CONSTRAINT_TYPE | varchar(64) | NO | | | |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.01 sec)
```

Exemple :

```
mysql> select table_name, constraint_name, constraint_type
-> from table_constraints
-> where table_schema='gnotes';
+-----+-----+-----+
| table_name | constraint_name | constraint_type |
+-----+-----+-----+
| annees | PRIMARY | PRIMARY KEY |
| classes | PRIMARY | PRIMARY KEY |
| classes | fk_Classes_annees1 | FOREIGN KEY |
| classes | fk_Classes_Filieres1 | FOREIGN KEY |
| departements | PRIMARY | PRIMARY KEY |
| etudiants | PRIMARY | PRIMARY KEY |
| etudiants | fk_Etudiants_Classes1 | FOREIGN KEY |
| evaluations | PRIMARY | PRIMARY KEY |
| evaluations | fk_Evaluations_annees1 | FOREIGN KEY |
| evaluations | fk_Evaluations_jour1 | FOREIGN KEY |
| evaluations | fk_Evaluations_Module1 | FOREIGN KEY |
| evaluations | fk_Evaluations_mois1 | FOREIGN KEY |
| filieres | PRIMARY | PRIMARY KEY |
| filieres | fk_Filieres_Departements1 | FOREIGN KEY |
| jour | PRIMARY | PRIMARY KEY |
| login | PRIMARY | PRIMARY KEY |
| module | PRIMARY | PRIMARY KEY |
| module | fk_Module_Unite_Enseignements1 | FOREIGN KEY |
| mois | PRIMARY | PRIMARY KEY |
```

## 5. Gestion des users

On utilise la table USER\_PRIVILEGES

```
mysql> desc user_privileges;
```

Field	Type	Null	Key	Default	Extra
GRANTEE	varchar(81)	NO			
TABLE_CATALOG	varchar(512)	YES		NULL	
PRIVILEGE_TYPE	varchar(64)	NO			
IS_GRANTABLE	varchar(3)	NO			

4 rows in set (0.02 sec)

Exemple :

Lister les user disposant du privilège SHUTDOWN

```
mysql> select * from user_privileges where privilege_type='shutdown';
```

GRANTEE	TABLE_CATALOG	PRIVILEGE_TYPE	IS_GRANTABLE
'edou'@'localhost'	NULL	SHUTDOWN	NO
'moumi'@'localhost'	NULL	SHUTDOWN	YES
'absa'@'localhost'	NULL	SHUTDOWN	YES
'root'@'127.0.0.1'	NULL	SHUTDOWN	YES
'lpigroot'@'localhost'	NULL	SHUTDOWN	YES
'fatou'@'localhost'	NULL	SHUTDOWN	YES
'root'@'localhost'	NULL	SHUTDOWN	YES

7 rows in set (0.00 sec)

Lister les user disposant du privilège DELETE

```
mysql> select * from user_privileges where privilege_type='delete';
```

GRANTEE	TABLE_CATALOG	PRIVILEGE_TYPE	IS_GRANTABLE
'edou'@'localhost'	NULL	DELETE	NO
'moumi'@'localhost'	NULL	DELETE	YES
'absa'@'localhost'	NULL	DELETE	YES
'root'@'127.0.0.1'	NULL	DELETE	YES
'lpigroot'@'localhost'	NULL	DELETE	YES
'fatou'@'localhost'	NULL	DELETE	YES
'root'@'localhost'	NULL	DELETE	YES

7 rows in set (0.00 sec)

Lister les user disposant du privilège WITH GRANT OPTION

```
mysql> select distinct(grantee) from user_privileges where is_grantable='yes';
```

grantee
'moumi'@'localhost'
'absa'@'localhost'
'root'@'127.0.0.1'
'lpigroot'@'localhost'
'fatou'@'localhost'
'root'@'localhost'

6 rows in set (0.00 sec)

## 6. Gestion des vues

Avec la table VIEWS

```
mysql> describe views;
```

Field	Type	Null	Key	Default	Extra
TABLE_CATALOG	varchar(512)	YES		NULL	
TABLE_SCHEMA	varchar(64)	NO			
TABLE_NAME	varchar(64)	NO			
VIEW_DEFINITION	longtext	NO		NULL	
CHECK_OPTION	varchar(8)	NO			
IS_UPDATABLE	varchar(3)	NO			
DEFINER	varchar(77)	NO			
SECURITY_TYPE	varchar(7)	NO			
CHARACTER_SET_CLIENT	varchar(32)	NO			
COLLATION_CONNECTION	varchar(32)	NO			

```
10 rows in set (0.03 sec)
```

## Les commandes de base :

Pour avoir de l'aide :

```
mysql> \?
```

? (\?)      Synonym for `help`.  
clear (\c)   Clear the current input statement.  
connect (\r)   Reconnect to the server. Optional arguments are db and host.  
delimiter (\d)   Set statement delimiter.  
ego (\G)      Send command to mysql server, display result      vertically.  
exit (\q)    Exit mysql. Same as quit.  
go (\g)      Send command to mysql server.  
help (\h)    Display this help.  
notee (\t)   Don't write into outfile.  
print (\p)   Print current command.  
prompt (\R)   Change your mysql prompt.  
quit (\q)    Quit mysql.  
rehash (\#)   Rebuild completion hash.  
source (\.)   Execute an SQL script file. Takes a file name as an argument.  
status (\s)   Get status information from the server.  
tee (\T)      Set outfile [to\_outfile]. Append everything into given outfile.  
use (\u)      Use another database. Takes database name as argument.  
charset (\C)   Switch to another charset. Might be needed for processing binlog with multi-byte charsets.  
warnings (\W)   Show warnings after every statement.  
nowarning (\w)   Don't show warnings after every statement.  
For server side help, type 'help contents' mysql>

---

## PROJET MYSQL

### Nom de la base : BASEAERIENNE

#### Tables :

Pilote (Numpilote, nom, ville, salaire, Telephone)

Avion (numero, type, capacite)

Service ( NumeroAvion ,Numpilote, NumVol)

Nb : une adresse se résume au nom de la ville.

Nb : prévoir un nom de pilote « concorde », certains pilotes n'ont pas de téléphone.

Villes : Nice, Paris, Lille, Bordeaux

Capacités des avions de 50 à 700

Type d'avion : airbus, concorde, boeing, un\_coucou

Avion : A300, B727, A320 etc

#### Questions :

- 1- Créer la base
- 2- Créer les tables en ligne de commande
- 3- Insérer 15 enregistrements dans chaque table
- 4- Créer un utilisateur portant votre nom pour administrer cette base avec tous les privilèges nécessaires sur cette base.
- 5- Créer un superadmin pour cette base autre que root
- 6- Créer un utilisateur stagiereNafi qui ne fait que des selects sur notre base.

#### 7- Ecrire les requêtes :

- liste de tous les pilotes . le listing comportera les champs : numpilote, nom, adresse, salaire
- calculer le salaire annuel des pilotes, le lister pour chaque pilote
- calculer la somme des salaires des pilotes
- donner tous les types d'avion de la compagnie
- donner les numeros d'avions et type d'avion de capacité sup à 300
- donner les noms de pilotes habitants Paris ou Nice
- quels pilotes ont un 't' dans leur nom en 3eme position?
- Quels sont les vols au depart de Nice, Paris ou Bordeaux ?
- donner les noms des pilotes ayant un a et un e dans leur nom ?
- donner les noms et no tel des pilotes qui ont un No de telephone
- donner le salaire le + eleve et l'afficher sous la forme : <valeur du salaire max> « MAX SALAIRE »
- quels sont les noms, adresse et salaire des pilotes triés par ordre croissant sur l adresse et pour une meme adresse ordre decroissant de salaire
- donner les paires de pilotes habitant la même ville Paris
- Donner les noms de pilotes qui conduisent un A300 ou un B727
- donner les noms des pilotes qui assurent un service, afficher les no de vols (jointure)(+)
- quels sont les pilotes qui ont assuré au moins 2 vols
- mettre à jour le salaire du pilote no 1 à 3200 : valider
- supprimer le pilote no 3
- ajouter les champs « age » à la table des pilotes

Nous avons une base oracle. Nous avons exporté d'autres pilotes dans un fichier pilote.txt

Voici le contenu du fichier :

Pilote (Numpilote, nom, ville, salaire,Telephone)

"11", "edouard sarr", "Dakar", "200000", "779082238", "28"

"11", "aly diop", "Dakar", "100000", "776678899", "28"

"11", "ami fall", "Dakar", "300000", "779445568", "28"

"11", "amadou mbaye", "Dakar", "400000", "773334238", "28"

- Importer les données
- Exporter la base de données dans un fichier pilote1.sql
- Importer la dans une autre base nommée: BASE2AVION
- Exporter BASE2AVION pilote2.sql

### **TAF**

Rendre un rapport complet format PDF et papier comprenant la réponse aux questions.

Rendre les fichiers pilote1.sql et pilote2.sql