COURS INTEGRAL

# BASE DE DONNEES MySQL

Cours Intégral



## 1. MCD et MLD

## 2. BASE DE DONNEES MYSQL (p 16/78)

**Installation avec Linux:**

```
$ sudo apt-get install mysql-server
```

Check version:

```
$ mysql -V
```

Redemarrer mysql

```
$ /etc/init.d# service mysql [ start | restart | stop | reload ]
```

Lancer une console mysql

```
$ sudo mysql -u root [nomBDD]
$ sudo mysql -u root -D [nomBDD]  # --user=root --Database=
```

```
> C:\xampp\mysql\bin\mysql.exe [OPTIONS] [database]
```

Afficher l'aide complete

```
$ mysql --verbose --help
```

**OPTIONS de mysql.exe (les plus utiles pour l'instant)**

```
  --init-command=name       SQL Command to execute when connecting to MySQL server.
                            Will automatically be re-executed when reconnecting.
  -h, --host=name           Connect to host.
  -H, --html                Produce HTML output.
  -X, --xml                 Produce XML output.
  -D, --database=name       Database to use.
  --delimiter=name          Delimiter to be used.
  -e, --execute=name        Execute command and quit. (Disables --force and history file.)
  -p, --password[=name]     Password to use when connecting to server. If password is not given it's
                            asked from the tty.
  -W, --pipe                Use named pipes to connect to server.
  -P, --port=#              Port number to use for connection or 0 for default to, in order of
                            preference, my.cnf, $MYSQL_TCP_PORT, /etc/services, built-in default
                           (3306).
  --prompt=name             Set the mysql prompt to this value.
  --protocol=name           The protocol to use for connection (tcp, socket, pipe, memory).
  --reconnect               Reconnect if the connection is lost. Disable with --disable-reconnect.
                            This option is enabled by default. (Defaults to on; use --skip-reconnect
                            to disable.)
  -s, --silent              Be more silent. Print results with a tab as separator, each row on new
                            line.
  -S, --socket=name         The socket file to use for connection.
  -t, --table               Output in table format.
  --tee=name                Append everything into outfile. See interactive help (\h) also.
                            Does not work in batch mode. Disable with --disable-tee. This option is
                            disabled by default.
  -u, --user=name           User for login if not current user.
  -U, --safe-updates        Only allow UPDATE and DELETE that uses keys.
  -U, --i-am-a-dummy        Synonym for option --safe-updates, -U.
  -v, --verbose             Write more. (-v -v -v gives the table output format).
  -V, --version             Output version information and exit.
  -w, --wait                Wait and retry if connection is down.
  --connect-timeout=#       Number of seconds before connection timeout.
  --show-warnings           Show warnings after every statement.
```

Dans la console mysql

```
mysql> show databases;
mysql> use <dbname>;
mysql> show tables;
```

```
show databases;                    -- afficher les noms des bdd existantes
show tables;                       -- afficher les noms des tables dans la bdd connecté
connect <nom_bdd>;                 -- se connecter à une base de donnees
use <nom_bdd>;                     -- se connecter à une base de donnees
select database();                 -- afficher le nom de la base de donnees en cours
source <nom_fichier>               -- lire l'entree a partir du fichier et l'executer comme si elle
                                      avait ete saisie sur le clavier
\. <nom_fichier>                   -- meme chose que `source nomfichier`
show index from <nom_table>        -- afficher les noms des index à la table nom_table
tee <nom_fichier>                  -- sauvegarder les resultats des requetes dans le fichier
                                      nom_fichier (=ouverture d'un flux)
notee                              -- terminer la sauvegarde (=fermeture du flux)
select * INTO OUTFILE '/path/to/file' from <nom_table>
                                   -- exporter le resultat de la requete vers le fichier indique
desc nom_table                     -- afficher la structure de la table <nom_table> (schema en SQLite)
```

## 3. LANGAGE SQL (p 25/78)

Le langage SQL comporte :

- une partie sur la définition des données: Le **LDD (Langage de Définition des Données)** qui permet de définir des tables, des vues et des contraintes d'intégrité (`CREATE, ALTER, TRUNCATE, DROP, RENAME`)
- une partie sur les requêtes: Le **LMD (Langage de Manipulation des Données)** qui permet d'interroger une base de données (`SELECT, INSERT, UPDATE, DELETE`)
- une partie sur le contrôle des données: le **LCD (Langage de Contrôle des Données)** qui permet de contrôler la sécurité et les accès aux données (`GRANT, REVOKE`)

#### CREER UNE BASE DE DONNEES

Vous pouvez créer une base de données en utilisant la commande **CREATE DATABASE**
ex.:

```
mysql> CREATE DATABASE base_dd;
```

#### SE CONNECTER A UNE BASE DE DONNEES EXISTANTE

Vous pouvez vous connecter à une base de données en utilisant les commandes **CONNECT** ou **USE**.
ex.:

```
mysql> CONNECT base_dd;
mysql> USE base_dd;
```

#### CREER UNE TABLE

Vous pouvez créer une table en spécifiant le nom de la table, suivi du nom de toutes les colonnes et de leur type.
ex:

```
CREATE TABLE client (
   id integer auto_increment primary key,
   nomClient varchar(80),
   adresseClient varchar(150)
   );
```

Les options:

```
AUTO_INCREMENTE          - incrémente la valeur de l'attribut id à chaque insertion
PRIMARY KEY              - précise que id est la clé primaire[1] de la table client
```

1 ***Clé primaire**: Dans une base de données relationnelle, une clé primaire est la donnée qui permet d'identifier de manière unique un enregistrement dans une table.*
*Une clé primaire peut être composée d'une ou de plusieurs colonnes de la table. Deux lignes distinctes de la table ne peuvent pas avoir les mêmes valeurs dans les colonnes définies comme clé primaire. Il est possible de définir pour une même table plusieurs contraintes d'unicité, mais au plus une seule clé primaire. Une clé primaire est choisie parmi les clés candidates. Suivant les cas il peut être nécessaire ou préférable d'utiliser une clé artificielle ajoutée aux données comme clé primaire. (source: Wiki)*

### CLES ETRANGERES

Soient les tables client et commande. Il s'agit maintenant de s'assurer que personne n'insère de ligne dans la table commande qui ne corresponde à une entrée dans la table `Client`.

```
CREATE TABLE commande (
   id integer auto_increment primary key,
   client_id integer,
   foreign key (client_id) references client (id),
   dateCommande date);
```

A cet effet, toute insertion d'enregistrement dans la table `commande` provoquera une erreur si la valeur de l'attribut `client_id` n'existe pas dans la table `client`.

Une clé étrangère peut aussi contraindre et référencer un groupe de colonnes. Il faut alors l'écrire sous forme d'une contrainte de table.
Syntaxe:

```
   CREATE TABLE tbl (
      a INTEGER PRIMARY KEY,
      b INTEGER,
      c INTEGER,
      FOREIGN KEY (b,c) REFERENCES autre_tbl(c1, c2)
      );
```

Exemple:

```
   CREATE TABLE commande_produits (
      no_produit INTEGER REFERENCES produits ON DELETE RESTRICT,
      id_commande INTEGER REFERENCES commandes ON DELETE CASCADE,
      quantite INTEGER,
      PRIMARY KEY (no_produit, id_commande)
      );
```

Les options:

```
   RESTRICT  - empêche la suppression d'une ligne référencée.
   CASCADE   - indique que, lors de la suppression d'une ligne référencée, les lignes la référençant
doivent être automatiquement supprimées.
```

### SUPPRIMER UNE TABLE: `DROP TABLE`

Si vous n'avez plus besoin d'une table ou que vous voulez la recréer différemment, vous pouvez la supprimer en utilisant la commande suivante:

```
   DROP TABLE nom_table;
```

### REMPLIR UNE TABLE: `INSERT INTO`

L'instruction `INSERT` est utilisée pour remplir une table.
Exemple:

```
   INSERT INTO client (nomClient, adresseClient)
   VALUES
      ('Joe', 'Saint-Nazaire'),
      ('Richard', 'Angres');
```

Il est aussi possible de remplir une table avec une table de même structure:
Exemple:

```
   INSERT INTO client (nomClient, adresseClient)
      (SELECT nom, adresse FROM Fournisseur);
```

### SUPPRIMER DES ENREGISTREMENTS D'UNE TABLE: `DELETE` ou `TRUNCATE`

L'instruction `DELETE` est utilisée pour supprimer des enregistrements.
Prenons comme exemple la suppression de tous les enregistrements de la table `CLIENT` dans `adresseClient='Saint-Nazaire'` :

```
   DELETE FROM Client WHERE adresseClient='Saint-Nazaire'
```

ATTENTION ! `DELETE FROM Client;` sans conditions effacerait toute les entrées de la table `Client` !

Il est possible de vider complètement la table en utilisant la clause `TRUNCATE` :

```
   TRUNCATE TABLE Client;
```

### TABLES JOINTES

Une table jointe est une table dérivée de deux autres tables suivant les règles du type de jointure particulier.

`--> CROSS JOIN / INNER JOIN`
Soit t1 et t2 telles que:

```
   create table t1 (no integer auto_increment primary key, nom varchar(20));
   insert into t1 (nom) values ('a'), ('b'), ('c');
   create table t2 (no integer primary key, valeur varchar(20));
   insert into t2 (no, valeur) values (1,'xxx'), (3,'yyy'), (5,'zzz');
```

Table t1

```
+----+------+
| no | nom  |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
+----+------+
```

Table t2

```
+----+--------+
| no | valeur |
+----+--------+
|  1 | xxx    |
|  3 | yyy    |
|  5 | zzz    |
+----+--------+
```

--> CROSS JOIN

```
SELECT * FROM t1, t2;             /* methode 1 */
SELECT * FROM t1 CROSS JOIN t2;   /* methode 2 CROSS JOIN */
+----+------+----+--------+
| no | nom  | no | valeur |
+----+------+----+--------+
|  1 | a    |  1 | xxx    |
|  2 | b    |  1 | xxx    |
|  3 | c    |  1 | xxx    |
|  1 | a    |  3 | yyy    |
|  2 | b    |  3 | yyy    |
|  3 | c    |  3 | yyy    |
|  1 | a    |  5 | zzz    |
|  2 | b    |  5 | zzz    |
|  3 | c    |  5 | zzz    |
+----+------+----+--------+
```

--> INNER JOIN

```
SELECT * FROM t1, t2 WHERE t1.no=t2.no;          /* methode 1 */
SELECT * FROM t1 INNER JOIN t2 ON t1.no=t2.no;   /* methode 2 INNER JOIN */
+----+------+----+--------+
| no | nom  | no | valeur |
+----+------+----+--------+
|  1 | a    |  1 | xxx    |
|  3 | c    |  3 | yyy    |
+----+------+----+--------+
```

--> LEFT JOIN

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.no=t2.no;
+----+------+------+--------+
| no | nom  | no   | valeur |
+----+------+------+--------+
|  1 | a    |    1 | xxx    |
|  2 | b    | NULL | NULL   |
|  3 | c    |    3 | yyy    |
+----+------+------+--------+
```

--> RIGHT JOIN

```
SELECT * FROM t1 RIGHT JOIN t2 ON t1.no=t2.no;
+------+------+----+--------+
| no   | nom  | no | valeur |
+------+------+----+--------+
|    1 | a    |  1 | xxx    |
|    3 | c    |  3 | yyy    |
| NULL | NULL |  5 | zzz    |
+------+------+----+--------+
```

**TRANSACTIONS**

Une transaction assemble plusieurs étapes en une seule opération tout-ou-rien.
Si un échec survient, alors aucune des étapes n'affecte la base de données.

--> BEGIN; ...; COMMIT;
Une transaction est déclarée en entourant les commandes SQL par les clauses BEGIN et COMMIT (ou END).

Exemple -- une transaction bancaire

```
BEGIN;
   UPDATE comptes
      SET balance = balance - 100.00
      WHERE nom = 'Alice';
   UPDATE branches
      SET balance = balance -100.00
      WHERE nom = (SELECT nom_branche FROM comptes WHERE nom = 'Alice');
   UPDATE comptes
      SET balance = balance + 100.00
      WHERE nom = 'Bob';
   UPDATE branches
      SET balance = balance + 100.00
      WHERE nom = (SELECT nom_branche FROM comptes WHERE nom = 'Bob');
COMMIT;
```

--> ROLLBACK (annulation) et SAVEPOINT
La commande ROLLBACK peut être utilisée à la place de COMMIT pour annuler la transaction.

Un groupe d'instructions entourées par BEGIN et COMMIT est parfois appelé bloc transactionnel.

Il est possible d'augmenter la granularité du contrôle des instructions au sein d'une transaction en utilisant des points de retournement (`SAVEPOINT`).

Ceux-ci permettent d'annuler des parties de la transaction tout en validant le reste. Après avoir défini un point de retournement à l'aide de `SAVEPOINT`, les instructions exécutées depuis ce point peuvent, au besoin, être annulées avec `ROLLBACK TO`.

Exemple:

```
BEGIN;

   UPDATE comptes
      SET balance = balance - 100.00
      WHERE nom = 'Alice';

   SAVEPOINT mon_pointdesauvegarde;

   UPDATE comptes
      SET balance = balance + 100.00
      WHERE nom = 'Bob';

   /* oops! Annulons à partir de mon SAVEPOINT
    * et créditons le compte de wally */
   ROLLBACK TO mon_pointdesauvegarde;

   UPDATE comptes
      SET balance = balance + 100.00
      WHERE nom = 'Wally';

COMMIT;
```

## VUES

Syntaxe:

```
CREATE [ OR REPLACE ] VIEW <nomVue> AS <requete>;
CREATE VIEW [IF NOT EXISTS ] <nomVue> AS <requete>;
```

Les options:

`OR REPLACE` : à utiliser si on veut remplacer une vue existante
`IF NOT EXISTS` : la vue ne sera créée que si elle n'existe pas déjà

A quoi servent les vues ?
Les vues peuvent être utilisées pour différentes raisons. Elles permettent par exemple de :

- restreindre l'accès aux données confidentielles par type d'utilisateur
- masquer la complexité du schéma
- mettre à jour automatiquement des données calculées ou d'aggrégations telles que `sum()`, `avg()` `max()`, etc.

## EXPRESSIONS CONDITIONNELLES

`--> CASE`
Syntaxe:

```
CASE
   WHEN <condition> THEN <action>    /* pas de virgule entre les WHEN */
    [ WHEN ... ]
  [ ELSE <action> ]
END;
```

Exemple:

```
CREATE TABLE test (a int);
INSERT INTO test (a) VALUES (1), (2), (3);

SELECT
  a,
  CASE
     WHEN a=1 THEN 'un'
     WHEN a=2 THEN 'deux'
     ELSE 'autres'
  END as description
FROM test;
```

```
/* RESULTAT:
   +---+-------------+
   | a | description |
   +---+-------------+
   | 1 | un          |
   | 2 | deux        |
   | 3 | autres      |
   +---+-------------+ */
```

`--> COALESCE`
Syntaxe:

```
COALESCE (valeur [, ...]);
```

La fonction `COALESCE` renvoie le premier de ses arguments qui n'est pas nul. Une valeur `NULL` est renvoyée seulement si tous les arguments sont nuls.

--> NULLIF
Syntaxe:

```
NULLIF (valeur1, valeur2)
```

La fonction NULLIF renvoie une valeur NULL si valeur1 et valeur2 sont égales, sinon il renvoie valeur1.

--> GREATEST et LEAST
Syntaxe:

```
GREATEST (valeur [, ...])   /* Plus grand que */
LEAST (valeur [, ...])      /* Au moins */
```

Les fonctions GREATEST et LEAST selectionnent respectivement la valeur la plus grande et la valeur la plus petite à partir d'une liste d'un certain nombre d'expressions

### CLAUSES GROUP BY ET HAVING

--> GROUP BY et HAVING
GROUP BY permet de grouper les lignes d'enregistrement
HAVING spécifie une condition à vérifier sur les lignes groupées

Exemple:

```
SELECT ville, max(t_basse)
FROM temps
GROUP BY ville;



SELECT ville, max(t_basse)
FROM temps
GROUP BY ville
   HAVING max(t_basse) < 40;
```

```
/* +---------------+-----+
   | ville         | max |
   +---------------+-----+
   | Hayward       |  37 |
   | San Francisco |  46 |
   +---------------+-----+ */

/* => GROUP BY ville HAVING max low temps < 40
   +---------------+-----+
   | ville         | max |
   +---------------+-----+
   | Hayward       |  37 |
   +---------------+-----+ */
```

### COMBINER DES REQUETES

Les résultats de deux requêtes peuvent être combinés en utilisant les opérations d'ensemble: union, intersection et différence.

Syntaxe:

```
requete1 UNION [ ALL ] requete2
requete1 INTERSECT [ ALL ] requete2
requete1 EXCEPT [ ALL ] requete2
```

--> UNION
UNION elimine les lignes dupliquées du résultat, de la même façon que DISTINCT, sauf si UNION ALL est utilisée.

--> INTERSECT
INTERSECT renvoie toutes les lignes qui sont à la fois dans le résultat de requete1 ET dans le résultat de requete2. Les lignes dupliquées sont éliminées sauf si INTERSECT ALL est utilisée.

--> EXCEPT
EXCEPT renvoie toutes les lignes qui sont dans le résultat de requete1 mais pas dans le résultat de requete2 (difference entre 2 requêtes). De nouveau, les lignes dupliquées sont éliminées sauf si EXCEPT ALL est utilisée.

### CREATION INDEX

L'utilisation d'un index simplifie et accélère les opérations de recherche, de tri, de jointure ou d'agrégation effectuées par le SGBD.

Syntaxe:

```
CREATE INDEX nomIndex
   ON nomTable(col1 [, col2] [, ...]);
```

Exemple:
-- Créer un index sur la colonne titre dans la table films

```
CREATE INDEX title_idx
   ON films(title);
```

-- Supprimer un index créé:

```
ALTER TABLE nomTable
   DROP INDEX nomIndex;
```

### AJOUTER UNE CONTRAINTE

-- Pour ajouter une contrainte, la syntaxe de contrainte de table est utilisée

```
ALTER TABLE produits
   ADD CONSTRAINT nom_contrainte_unique
   UNIQUE (no_produit);
```

```
ALTER TABLE produits
   ADD CONSTRAINT nom_contrainte_foreign_key
   FOREIGN KEY (id_group_produits) REFERENCES groupe_produits;
```

-- Pour enlever respectivement les contraintes:

```
ALTER TABLE produits
   DROP INDEX nom_contrainte_unique;
ALTER TABLE produits
   DROP FOREIGN KEY nom_contrainte_foreign_key;
```

-- Pour ajouter une contrainte `NOT NULL`, qui ne peut pas être écrite sous forme d'une contrainte de table:

```
ALTER TABLE produits
   MODIFY no_produit VARCHAR(20) NOT NULL;
```

-- Pour enlever cette même contrainte `NOT NULL`:

```
ALTER TABLE produit
   MODIFY no_produit VARCHAR(20) NULL;
```

La contrainte étant immédiatement vérifiée, les données de la table doivent satisfaire la contrainte avant qu'elle ne soit ajoutée.

### SUPPRIMER UNE CONTRAINTE

La suppression se fait par son si elle a été explicitement nommée.
Dans le cas contraire, le système engendre et attribue un nom qu'il faut découvrir à partir de la commande `SHOW CREATE TABLE <nomTable>`

Syntaxe:

```
ALTER TABLE nomTable
   DROP FOREIGN KEY nomContrainte;
```

Exemple:

```
ALTER TABLE commande
   DROP FOREIGN KEY commande_fki_client_id;
```

### MODIFIER LA VALEUR PAR DEFAUT D'UNE COLONNE

-- Ajouter une valeur par defaut
Syntaxe:

```
ALTER TABLE nomTable
   ALTER COLUMN nomColonne
   SET DEFAULT valeur;
```

Exemple:

```
ALTER TABLE produits
   ALTER COLUMN prix SET DEFAULT 7.77;
```

-- Retirer toute valeur par defaut
Syntaxe:

```
ALTER TABLE nomTable
   ALTER COLUMN nomColonne
   DROP DEFAULT;
```

Exemple:

```
ALTER TABLE produits
   ALTER COLUMN prix DROP DEFAULT;
```

### MODIFIER LE TYPE DE DONNEE D'UNE COLONNE

Syntaxe:

```
ALTER TABLE nomTable
   MODIFY nomColonne type1 [type2 ...];
```

Exemples:

```
ALTER TABLE produits MODIFY prixUnitaire NUMERIC(10,2);
ALTER TABLE client MODIFY nom VARCHAR(150) NOT NULL;
```

### RENOMMER UNE COLONNE

Syntaxe:
```
ALTER TABLE nomTable
   CHANGE COLUMN nomColonne nouveauNomColonne <type>;
```

Exemple:
```
ALTER TABLE client
   CHANGE COLUMN nom nomClient VARCHAR(100);
```

### RENOMMER UNE TABLE:

Syntaxe:
```
ALTER TABLE nomTable
   RENAME TO nouveauNomTable;
```

Exemple:
```
ALTER TABLE produits
   RENAME TO articles;
```

### AFFICHER LA REQUETE DE CREATION D'UNE TABLE

Syntaxe:
```
SHOW CREATE TABLE nomTable;
```

Exemple:
```
SHOW CREATE TABLE article;

/* =>
+---------+-------------------------------------------------------------+
| Table   | Create Table                                                |
+---------+-------------------------------------------------------------+
| article | CREATE TABLE `article` (
|         |   `id` int(11) NOT NULL AUTO_INCREMENT,
|         |   `numeroArticle` varchar(20) NOT NULL,
|         |   `designation` varchar(250) DEFAULT NULL,
|         |   `prixUnitaire` decimal(10,2) DEFAULT NULL,
|         |   `prixRevient` decimal(10,2) DEFAULT NULL,
|         |   PRIMARY KEY (`id`),
|         |   UNIQUE KEY `numeroArticle` (`numeroArticle`)
|         |   ) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1 |
+---------+-------------------------------------------------------------+
*/
```

### AFFICHER LES CONTRAINTES A UNE TABLE

Syntaxe:
```
SELECT *
FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'nomTable';
```

Exemple:
```
SELECT * FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
WHERE TABLE_NAME = 'article';
/* =>
+--------------------+-------------------+-----------------+--------------+------------+-----------------+
| CONSTRAINT_CATALOG | CONSTRAINT_SCHEMA | CONSTRAINT_NAME | TABLE_SCHEMA | TABLE_NAME | CONSTRAINT_TYPE |
+--------------------+-------------------+-----------------+--------------+------------+-----------------+
| def                | poecjava          | PRIMARY         | poecjava     | article    | PRIMARY KEY     |
| def                | poecjava          | numeroArticle   | poecjava     | article    | UNIQUE          |
+--------------------+-------------------+-----------------+--------------+------------+-----------------+
*/
```

## 4. SAUVEGARDE ET RESTAURATION (p 50/78)

### SAUVEGARDES -- `mysqldump`

Le principe est de produire un fichier texte de commandes SQL appelé "fichier dump[2]" à partir de la commande externe[3] `mysqldump`

L'usage basique est:
```
$ mysqldump -u nomUtilisateur [options] base_de_donnée > fichier_de_sortie
```

2 ***Dump**: anglais pour "dépôt". To dump - déposer / déverser / décharger*
3 ***commande externe** = invite de commande MS-DOS (cmd.exe), par opposition à invite de commande mysql (lancer mysql.exe depuis cmd.exe)*

Les options de `mysqldump` sont (les plus utilisées):

```
    --all-databases         selectionne toutes les bdd existantes
    --database bd1 bd2      selectionne les bdd bd1 bd2 etc.
    -d                      permet de ne tenir compte que des structures des tables de la bdd
```

Exemple:

```
    -- Sauvegarder la bdd maBoutique situé dans le répertoire C:\Test

    -- Données + structures
    $ mysqldump -u root maBoutique > c:\test\maBoutique_dump.sql

    -- Structure seulement
    $ mysqldump -u root -d maBoutique > c:\test\maBoutique_structure.sql
```

Quelques exemples avec les options:

```
    -- Sauvegarder toutes les bases de données
    $ mysqldump --user=nomUser --password=monPassword --all-databases > fichier_destination.sql

    -- Sauvegarder une base de données précise
    $ mysqldump --user=nomUser --password=monPassword --databases nomBDD > fichier_destination.sql

    -- Sauvegarder plusieurs base de données
    $ mysqldump --user=nomUser --password=monPassword --databases nomBDD1 nomBDD2 >
fichier_destination.sql

    -- Sauvegarder une table en particulier
    $ mysqldump --user=nomUser --password=monPassword --databases nomBDD --tables nomTable >
fichier_destination.sql

    -- Sauvegarder plusieurs tables
    $ mysqldump --user=nomUser --password=monPassword --databases nomBDD --tables nomTable1 nomTable2 >
fichier_destination.sql
```

### SAUVEGARDE AUTOMATIQUE DE BDD EN WINDOWS

Pour se faire, mettre dans une commande batch (fichier avec extension `.bat`) les différentes lignes de commande suivantes (exemple `c:\sauvegardeSQL.bat`)

```
REM Sauvegarde automatique de base de donnees dans un script .bat
SET JOUR=%date:~-10,2%
SET ANNEE=%date:~-4%
SET MOIS=%date:~-7,2%
SET HEURE=%time:~0,2%
SET MINUTE=%time:~3,2
SET SECONDE=%time:~-5,2%
SET REPERTOIR=C:\SauvegardeMysql
SET FICHIER=%REPERTOIR%\Sauvegarde_du_%JOUR%_%MOIS%_%ANNEE%_a_%HEURE%h%MINUTE%mn.sql
IF NOT EXIST %REPERTOIR% md %REPERTOIR%
mysqldump -u nomUtilisateur -ppassword maBDD > "%FICHIER%"
```

Remarques

- Les guillemets sur la variable `FICHIER` nous permet d'accepter les espaces dans le nom du fichier de sauvegarde.
- Utilisez le Planificateur de Tâches de Windows pour l'automatisation.
- Un exemple de fichier résultat: `Sauvegarde_du_03_01_2014_a_16h27mn.sql`

### SAUVEGARDE AUTOMATIQUE DE BDD SOUS LINUX

Mettre le script suivant dans un fichier portant l'extension `.sh`

```
#! /bin/bash
# Script qui dump la bdd pour sauvegarde
bdd=mabasededonnee
hm=$(date +%X)
h=${hm:0:2}
m=${hm:3:2}
fichier=${bdd}_$(date +%Y%m%d)_${h}h${m}.sql
mysqldump -u root -ppassword mabasededonnee > $fichier
```

Rendre le fichier executable:

```
$ chmod +x monFichier
```

Remarque:
Utiliser le planificateur de taches `gnome-schedule` pour l'automatisation.
Exemple de fichier resultat: `mabasededonnee_20180509_22h18.sql`

### RESTAURATIONS

Les fichiers crees par `mysqldump` peuvent etre lus par le programme `mysql`. La syntaxe generale d'une commande de restauration est :

```
$ mysql -u nomUtilisateur maBDD < fichier_d_entree
```

où le `fichier_d_entree` est le fichier en sortie de la commande `mysqldump`. La base de donnees `maBDD` n'est pas creee par cette commande. Elle doit etre creee avant d'executer `mysql`.

Exemple:

```
$ mysql -u root maBoutique1 < c:\test\maBoutiqueDump.sql
```

*(`maBoutique1` est deja creee avant le lancement de la commande)*

## 5. GERER LES UTILISATEURS ET LEURS DROITS * (p 56/78)

### INTRODUCTION

Il existe deux phases lors d'un controle d'acces:

1. la connection au serveur
2. la verification des privileges

L'ensemble des informations relatives a la gestion des utilisateurs est stocke dans une base de donnees nommee `mysql`. C'est l'une des bases creees automatiquement lors de l'installation de MySQL. Nous pouvons y ajouter, modifier et supprimer des donnees, de 2 manieres differentes:

- - en modifiant directement le contenu des tables avec les ordres "classiques" (`INSERT, UPDATE, DELETE`) – **METHODE TRES DELICATE -- DECONSEILLEE**
- - en utilisant les ordres **DCL (Data Control Language)** de MySQL comme `CREATE USER, SET PASSWORD, GRANT, REVOKE`. C'est la methode conseillee.

Pour vous connecter a un serveur MySQL, vous devez disposer d'un nom d'utilisateur et du mot de passe associe.

Syntaxe:

```
$ mysql [-n nomHote] [-u nomUtilisateur] [-pVotreMotDePasse]
```

La liste des utilisateurs, ainsi que leur eventuel mot de passe, correspond au contenu de la table `user` de la base `mysql`.

### NOTION DE COMPTE UTILISATEUR

Si la plupart des SGBD caracterisent un compte par son nom d'utilisateur (`login`), MySQL prend en consideration un parametre supplementaire: le nom (ou l'adresse IP) de la machine depuis laquelle l'utilisateur essaie de se connecter (appelee "hote").

Un compte utilisateur est donc l'association entre le nom utilisateur et l'hote de cet utilisateur.

Ainsi, vous pouvez avoir par exemple:

- `root@localhost` : le super administrateur travaillant directement sur le serveur
- `root@provider.com` : le super administrateur travaillant chez lui, avec sa connexion Internet.

### LES DROITS D'ACCES DANS MYSQL

La base de donnees `mysql` est

- creee automatiquement a l'installation
- la base qui gere les utilisateurs et leurs privileges d'acces sur l'ensemble des bases de donnees, y compris `mysql` elle-meme.

"`root`" est l'utilisateur privilegie qui est en charge d'administrer la base `mysql`. C'est donc le seul initialement qui pourra ajouter des utilisateurs et modifier leurs droits.

Comme toutes les autres bases, `mysql` constituee de plusieurs tables.
Nous allons decrire les 3 plus importantes a savoir:

1) table `USER`
2) table `DB`
3) table `HOST`

### 1. LA TABLE USER

La table `user` recense tous les utilisateurs. Elle **contient les privileges de chacun** d'entre eux. En pratique, les privileges les plus importants sont reserves a un administrateur de la base.

```
+---------------+--------+-------------------------------------------------+
| Attribut      | Valeur | Fonction
+---------------+--------+-------------------------------------------------+
| Host          | host   | serveur sur lequel mysql tourne (localhost)
| User          | user   | login mysql de l'utilisateur
| Password      | pass   | mot de passe mysql (crypte) de l'utilisateur
| Select_priv   | y/n    | selection
| Insert_priv   | y/n    | insertion
| Update_priv   | y/n    | modification (de valeur)
| Delete_priv   | y/n    | effacement
| Index_priv    | y/n    | indexation
| Alter_priv    | y/n    | modification (de table ou de champs)
| Create_priv   | y/n    | creation
| Drop_priv     | y/n    | suppression
| Grant_priv    | y/n    | permission
| Reload_priv   | y/n    | relancer mysql
| Shutdown_priv | y/n    | arreter mysql
| Process_priv  | y/n    | processus
| File_priv     | y/n    | lecture et ecriture de fichier (import/export)
+---------------+--------+-------------------------------------------------+
```

### 2. LA TABLE DB

La table `db` permet de lier un utilisateur a une ou plusieurs bases. C'est dans cette table que l'on peut **donner des privileges a un utilisateur.**

```
+---------------+--------+-------------------------------------------------+
| Attribut      | Valeur | Fonction
+---------------+--------+-------------------------------------------------+
| Host          | host   | serveur sur lequel mysql tourne (localhost)
| Db            | db     | base de donnees
| User          | user   | login mysql de l'utilisateur
| Select_priv   | y/n    | selection
| Insert_priv   | y/n    | insertion
| Update_priv   | y/n    | modification (de valeur)
| Delete_priv   | y/n    | effacement
| Index_priv    | y/n    | indexation
| Alter_priv    | y/n    | modification (de table ou de champs)
| Create_priv   | y/n    | creation
| Drop_priv     | y/n    | suppression
| Grant_priv    | y/n    | permission
+---------------+--------+-------------------------------------------------+
```

### 3. LA TABLE HOST

La table `host` permet de lier un hote a une base de donnees.
Cette table permet de **gerer les privileges non pas par utilisateur mais par machine**.

```
+---------------+--------+-------------------------------------------------+
| Attribut      | Valeur | Fonction
+---------------+--------+-------------------------------------------------+
| Host          | host   | serveur sur lequel mysql tourne (localhost)
| Db            | db     | base de donnees
| Select_priv   | y/n    | selection
| Insert_priv   | y/n    | insertion
| Update_priv   | y/n    | modification (de valeur)
| Delete_priv   | y/n    | effacement
| Index_priv    | y/n    | indexation
| Alter_priv    | y/n    | modification (de table ou de champs)
| Create_priv   | y/n    | creation
| Drop_priv     | y/n    | suppression
| Grant_priv    | y/n    | permission
+---------------+--------+-------------------------------------------------+
```

### GESTION DES DROITS D'ACCES: Manipulation des tables de la base mysql

Donner des droits d'acces aux utilisateurs peut etre fait:

- - en utilisant directement des requetes `INSERT`
- - puis en demandant au serveur de recharger les tables de droits en utilisant la commande `FLUSH PRIVILEGES`.

Exemple:

```
-- ajouter un utilisateur "toto" et lui donner acces uniquement a la base "donnees_de_toto"

INSERT INTO user (host, user, password)
   VALUES ('localhost', 'toto', PASSWORD('toto_pass'));
```

```
 INSERT INTO db (host, user, db, Select_priv, Insert_priv, Update_priv, Delete_priv, Create_priv,
Index_priv, Alter_priv)
    VALUES ('localhost', 'toto', 'donnees_de_toto', 'Y','Y','Y','Y','Y','Y','Y');

 FLUSH PRIVILEGES;
```

N'oulbiez pas que le mot de passe doit être encrypté !!
Il est généré à l'aide de la fonction `PASSWORD()` de `mysql`.

Pour le changer:

```
   UPDATE user
      SET password=PASSWORD('nouveau_mdp')
      WHERE user='toto';
```

### AFFICHER TOUS LES UTILISATEURS DU SERVEUR MYSQL

```
 SELECT user, host FROM mysql.user;
```

???????? manque rien ?

### GESTION DES DROITS D'ACCES: Utilisation des ordres DCL (Data Control Language)

Donner des droits d'accès aux utilisateurs en utilisant les ordres DCL se déroule en 2 étapes:

1. Création de l'utilisateur
2. Assignation des privilèges

#### 1. Creation des utilisateurs

La commande `CREATE USER` cree un nouveau compte MySQL.
Pour chaque compte, `CREATE USER` crée un nouvel enregistrement dans la table `mysql.user`, sans aucun droit. Une erreur survient si le compte existe déjà. Le compte peut recevoir un mot de passe avec la clause exceptionnelle `IDENTIFIED BY`.

Syntaxe de `CREATE USER`:

```
   CREATE USER user [ IDENTIFIED BY [PASSWORD] 'password'] [, user [IDENTIFIED BY [PASSWORD]
'password']] ... ;
```

Exemples:

```
   -- création d'un utilisateur 'moi' avec mdp 'oim' sur le serveur local
   CREATE USER 'moi'@'localhost' IDENTIFIED BY 'oim';

   -- l'ordinateur appelé monOrdi
   CREATE USER moi@monOrdi;

   -- l'ordinateur dont l'IP est 192.168.1.123
   CREATE USER moi@192.168.1.123;

   -- n'importe quel ordinateur dont l'IP est de la classe 192.168
   CREATE USER moi@'192.168.%';

   -- n'importe quel ordinateur du domaine microapp.com
   CREATE USER moi@'%.microapp.com';

   -- n'importe quel ordinateur
   CREATE USER moi@'%';
```

#### 2. Ajout de droits

Les droits donnés à un utilisateur sont définis directement selon qu'on leur autorise un accès large ou restreint aux bases, aux tables, et même aux colonnes d'une table. Ces droits pourront être insertion, consultation, destruction, ...

Syntaxe de `GRANT`

```
   GRANT priv_type [(column_list) [, priv_type [(column_list)]] ...
      ON [ tbl_name | * | *.* | db_name.* ]
      TO user1 [, user2] ...
     [ WITH [GRANT OPTION | MAX_QUERIES_PER_COUNT count | MAX_UPDATES_PER_HOUR count |
MAX_CONNECTIONS_PER_HOUR count] ]
```

Les options `MAX_QUERIES_PER HOUR`, `MAX_UPDATES_PER_HOUR`, `MAX_CONNECTIONS_PER_HOUR` permettent de définir respectivement le nombre de requêtes maximal, le nombre de mises à jour maximal et le nombre de connexions maximal (le tout en nombre par heure).

Exemple:

```
   GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP
      ON tutorials.*
      TO 'zara'@'localhost'
      IDENTIFIED BY 'zara123';
```

### Ajout de droits: Liste non-exhaustives des privilèges pouvant être accorder aux utilisateurs

```
-- autorise tous les privilèges simples
GRANT ALL [PRIVILEGES]

-- Autorise l'usage de ...
GRANT ALTER ...
GRANT CREATE ...
GRANT DELETE ...
GRANT DROP ...            /* DROP TABLE uniquement */
GRANT INDEX ...           /* [CREATE | DROP] INDEX */
GRANT INSERT ...
GRANT SELECT ...
GRANT SHOW DATABASES      /* autorise l'usage de SHOW DATABASES */
GRANT SHUTDOWN            /* autorise l'usage de l'arrêt par mysqladmin */
GRANT UPDATE ...
etc.
```

Exemples:

```
   -- donner à l'utilisateur 'toto' tous les droits d'accès à toutes les bases à partir de n'importe
quelle machine
   GRANT ALL PRIVILEGES ON * TO toto@'%' IDENTIFIED BY 'password';

   -- donner à tout le monde les droits de consultation de la base Fournisseurs
   GRANT SELECT ON Fournisseurs TO PUBLIC;

   -- donner à Albert et Marcel le droit de mise à jour sur la base Fournisseurs
   GRANT UPDATE ON Fournisseurs TO Albert, Marcel;
```

Remarque: La commande `GRANT` permet aussi de créer un compte si ce dernier n'existe pas encore.

### GESTION DES DROITS D'ACCES: Suppression des droits

Pour enlever les droits sur les utilisateurs, on utilise la commande `REVOKE`.
Elle est sensiblement la même que celle de l'ajout.

Syntaxe:

```
   REVOKE priv_type [column_list)][, priv_type [(column_list)]] ...
   ON {tbl_name|*|*.*|db_name.*}
   FROM user[, user] ...
```

Exemple:

```
   -- Supprimer à Albert le droit de mise à jour sur la base Fournisseurs.
   REVOKE UPDATE ON Fournisseurs FROM Albert@localhost;
```

### AFFICHER LES DROITS A UN UTILISATEUR

```
   SHOW GRANTS FOR nomUtilisateur@'nomHote';
```

### AFFICHER TOUS LES DROITS DE L'UTILISATEUR EN COURS

```
   SHOW GRANTS;
```

### AFFICHER L'UTILISATEUR EN COURS

```
   SELECT CURRENT_USER;
```

### GESTION DES DROITS D'ACCES: Changement de mot de passe

Pour assigner un mot de passe à un compte, on utilise la commande `SET PASSWORD`

```
   mysql> SET PASSWORD FOR 'jeffrey'@'%'=PASSWORD('biscuit');
```

Seuls les utilisateurs root ayant des accès en écriture à la base `mysql` peuvent changer les mots de passe des autres utilisateurs.

Pour modifier le mot de passe de l'utilisateur courant, on peut omettre la clause `FOR`

```
   mysql> SET PASSWORD = PASSWORD('biscuit');
```

SOURCES

Livres
"*Audit et Optimisation MySQL 5: Bonne pratique pour l'administrateur*", par Pascal Borghino, Olivier Dasini, Arnaud Gadal
"*MySQL*", par Paul Dubois

Sites
http://www.developpez.com
http://dev.mysql.com
http://sql.sh

*-- FIN --*