



Base de Données Avancées

COURS 1

Introduction à SQL et MySQL



Plan de la présentation

- ▶ SQL langage ou norme
 - Aspects du langage SQL
 - Avantages du langage SQL
 - Modèle des données utilisé
- ▶ Ce quoi MySQL ?
 - Avantage de MySQL
 - Console de MySQL
- ▶ SQL de Base



SQL Langage ou norme

- ▶ IBM a implanté le modèle relationnel au travers du langage **SEQUEL** (*Stuctured English as QUERy Language*), renommé par la suite **SQL** (*Structured Query Language ou langage des requêtes structurées*) au début des années 80.
- ▶ **SQL1** première norme datant de 1987, compromis entre plusieurs constructeurs, mais fortement influencé par le dialecte d'IBM.
- ▶ **SQL2** normalisé en 1992
- ▶ **SQL3** normalisé en 1999....
- ▶ **SQL2011**

Chaque norme a ses propres apports au niveau des caractéristiques et fonctionnalités nouvelles.

Aspects du langage SQL

- ▶ Définition des données
- ▶ Manipulation des données
- ▶ Interrogation des données
- ▶ Contrôle des données

| | |
|---|---------------------------------|
| CREATE - ALTER - DROP - RENAME - TRUNCATE | Définition des données (LDD) |
| INSERT - UPDATE - DELETE - LOCK TABLE | Manipulation des données (LMD) |
| SELECT | Interrogation des données (LID) |
| GRANT - REVOKE - COMMIT - ROLLBACK - SAVEPOINT - SET TRANSACTION | Contrôle des données (LCD) |



Avantages du langage SQL

- ▶ Standard industriel depuis 1987, s'enrichit au fil du temps.
- ▶ Pris en charge par les principaux langages de programmation comme C ou Cobol, mais aussi les langages Orientés Objet comme C++, Java et C#.
- ▶ Possibilité de modifier la structure des données sans pour autant engendrer une refonte importante du programme (Indépendance entre programme et donnée)
- ▶ Adapté aux grandes applications informatiques de gestion
- ▶ Les systèmes basés sur ce langage sont fiables et performants.

Modèle des données: la table

- ▶ SQL utilise le modèle de données relationnelles (MDR).
- ▶ La table relationnelle (*relational table*) est la structure de données de base qui contient des enregistrements appelés aussi « lignes » (*rows*).
- ▶ Une table est composée de colonnes (*columns*) qui décrivent les enregistrements.
- ▶ Ce modèle est essentiellement basé sur les valeurs. Les associations entre tables sont toujours binaires et assurées par les clés étrangères.
- ▶ Exemple : [Compagnie](#)

| comp | nrue | rue | ville | nomComp |
|------|------|-----------|-----------|--------------|
| AF | 10 | Gambetta | Paris | Air France |
| SING | 7 | Camparols | Singapour | Singapore AL |

Pilote

| brevet | nom | nbHVol | compa |
|--------|-------------|--------|-------|
| PL-1 | Louise Ente | 450 | AF |
| PL-2 | Jules Ente | 900 | AF |
| PL-3 | Paul Soutou | 1000 | SING |

Modele des données : les clés d'une table

- ▶ La **clé primaire** (*primary key*) d'une table est l'ensemble minimal de colonnes qui permet d'identifier de manière unique chaque enregistrement.
- ▶ Une clé est dite « **candidate** » (*candidate key*) si elle peut se substituer à la clé primaire à tout instant. Une table peut contenir plusieurs clés candidates ou aucune.
- ▶ Une **clé étrangère** (*foreign key*) référence dans la majorité des cas une clé primaire d'une autre table (sinon une clé candidate sur laquelle un index unique aura été défini). Elle est composée d'une ou plusieurs colonnes. Une table peut contenir plusieurs clés étrangères ou aucune.

Les clés primaires et étrangères sont définies dans les tables en SQL à l'aide de **contraintes**.



Ce quoi MySQL ?

- ▶ Un SGBDR, système de gestion de base de données relationnelle, permettant de gérer un ensemble des données structurées reliées entre-elles.
- ▶ Un système écrit en C et C++, *open source*, fonctionnant sur les systèmes d'exploitation le plus couramment utilisé(Linux, Microsoft, MacOs...etc.).
- ▶ Un serveur développé par une société suédoise dénommé MySQL AB (Uppsala, Suède).
- ▶ Racheté en 2008 par Sun Microsystems (USA), elle-même racheté en 2010 par Oracle Corporation (USA).



Avantages de MySQL :

- ▶ Système *open source*
- ▶ Fonctionnalités de plus en plus riche version après version.
- ▶ Système performant
- ▶ Système ouvert à tous les principaux langages de programmation
- ▶ Système fonctionnant sur le SE le plus couramment utilisé
- ▶ Sa facilité d'utilisation pour des applications Web de taille moyenne.

MySQL: Notion de *database*

- ▶ **Database** (*base de donnée*) un regroupement logique d'objets (*tables, index, vues, déclencheurs, procédures cataloguées, etc.*) pouvant être stockés à différents endroits de l'espace disque.
- ▶ Pour MySQL, un objet appartient à son schéma (*database*), il n'y a pas de notion d'appartenance d'un objet (table, index, etc.) à un utilisateur.
- ▶ Pour Oracle ou d'autres SGBD, chaque objet appartient à son *user*.
- ▶ En MySQL, dès que le serveur est installé dans votre système, il y'a l'existence de trois base de données (*mysql, test et information_schema*).
Ces bases sont dites base système, le serveur ne peut fonctionner sans elles.



MySQL: Notion d'hôte

- ▶ Avec MySQL, *host* c'est le nom de la machine hébergeant le SGBD.
- ▶ Il est possible de distinguer des accès d'un même utilisateur se connectant depuis plusieurs machines différentes.
- ▶ La notion d'identité est basée sur le couple **nom d'utilisateur MySQL (*user*) côté serveur, machine cliente**.
- ▶ Exemple: si l'user *Said* se connecte à partir de la machine *clt1* vers le serveur une fois, et une autre fois, il se connecte à partir de la machine *clt5*, il faudra prévoir deux identités différentes au niveau du serveur (*said@clt1* et *said@clt5*).



MySQL: Interface de commande

- ▶ L'interface de commande en ligne (*Console MySQL*) ressemble à une fenêtre DOS.
- ▶ Elle permet de dialoguer de la plus simple façon avec la **base de données**. L'utilisation peut être souvent *interactive* ou « *batch* ».
- ▶ En mode *interactive* , le résultat des extractions sont représentés sous forme **tabulaire**.
- ▶ Une fois la connexion établie vers le serveur, des **instructions** (ordres SQL) sont envoyées à la base qui retourne des **résultats** affichés dans la même fenêtre de commande.

Une fenêtre Console de MySQL

```
c:\wamp\bin\mysql\mysql5.6.12\bin\mysql.exe
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.6.12-log MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

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

mysql> _
```

Options de base de la Console MySQL

- ▶ Les principales options au lancement de la Console MySQL sont résumées dans le tableau suivant :

| Option | Commentaire |
|---------------------------------------|---|
| --help ou -? | Affiche les options disponibles, l'état des variables d'environnement et rend la main. |
| --batch ou -B | Toute commande SQL peut être lancée dans la fenêtre de commande système sans pour autant voir l'invite ; les résultats (colonnes) sont séparés par des tabulations. |
| --database=nomBD OU -D nomBD | Sélection de la base de données à utiliser après la connexion. |
| --host=nomServeur OU -h nomServeur | Désignation du serveur. |
| --html ou -H | Formatte le résultat des extractions en HTML. |
| --one-database OU -O | Restreint les instructions à la base de données spécifiée initialement. |
| -p | Demande le mot de passe sans l'employer en tant que paramètre. |
| --password=motdePasse | Transmission du mot de passe de l'utilisateur à connecter. Évitez cette option et préférez la précédente... |
| --prompt=parametre | Personnalise l'invite de commande (par défaut mysql>). |
| --silent ou -s | Configure le mode silence pour réduire les messages de MySQL. |

Options de base de la Console MySQL

| | |
|---|--|
| --skip-column-names ou -N | N'écrit aucun en-tête de colonne pour les résultats d'extraction. |
| --table ou -t | Formatte le résultat des extractions en tables à en-tête de colonne (par défaut dans le mode interactif). |
| --tee=cheminNomFichier | Copie la trace de toute la session dans le fichier que vous indiquez. |
| --user=utilisateur ou -u utilisateur | Désigne l'utilisateur devant se connecter. |
| --verbose ou -v | Mode verbeux pour avoir davantage de messages du serveur. |
| --version ou -V | Affiche la version du serveur et rend la main. |
| --vertical ou -E | Affiche les résultats des extractions verticalement (non plus en lignes horizontales). |
| --xml ou -X | Formatte le résultat des extractions en XML. Les noms de balises générées sont <resultset> pour la table résultat, <row> pour chaque ligne et <field> pour les colonnes. |

Il est possible de combiner ces options en les séparant simplement par un espace (exemple :

```
mysql --tee=D:\\dev\\sortiemysql.txt --database=bdtest)
```

Mode Batch de la Console MySQL

- ▶ Pour lancer plusieurs commandes regroupées dans un fichier à extension « `.sql` », il faut préciser le chemin du fichier (`.sql`) et celui qui contiendra les éventuels résultats., c'est le mode batch de la console.
- ▶ Exemple :

```
mysql --user=root --password=tor dbtest
<D:\\testdv\\Testbatch.sql >D:\\testdv\\sortie.txt
```

- ▶ L'instruction suivante exécute dans la base `dbtest`, sous l'autorité de l'utilisateur `root`, les commandes contenues dans le fichier `Testbatch.sql` situé dans le répertoire `D:\\dev`. Le résultat sera consigné dans le fichier `sortie.txt` du même répertoire.

Commandes de base de la Console

- Après connexion au serveur, il est possible d'utiliser des commandes ou faire des copier-coller d'un éditeur de texte dans la Console MySQL.
- Le tableau suivant résume les principales instructions pour manipuler la console d'entrée de l'interface.
- Exemple : prompt (\u0@\h) [\d]>

| Commande | Commentaire |
|-----------------------------|---|
| ? | Affichage des commandes disponibles. |
| delimiter chaîne | Modifie le délimiteur (par défaut « ; »). |
| use nomBase | Rend une base de données courante. |
| prompt chaîne | Modifie l'invite de commande avec les paramètres vus précédemment. |
| quit ou exit | Quitte l'interface. |
| source cheminNomFichier.sql | Charge et exécute dans le buffer le contenu du <i>cheminNomFichier.sql</i> (ex : source D:\\\\dev\\\\Testbatch.sql exécutera le script Testbatch.sql situé dans D:\\dev). |
| tee nomFichiersortie | Création nomFichiersortie dans le répertoire C:\\Program Files\\MySQL\\MySQL Server n.n\\bin qui contiendra la trace de la session. |

Autres caractéristiques de MySQL

- ▶ **Délimiteurs** : la directive `delimiter` permet de choisir le symbole qui terminera chaque instruction , le symbole par défaut de MySQL c'est « ; ». exemple:

```
delimiter #
```

```
CREATE TABLE Test (t CHAR(8)) #
```

- ▶ **Sensibilité à la casse** : MySQL est sensible par défaut à la casse dans la plupart des distributions Unix, il ne l'est pas pour Windows.
- ▶ La variable `lower_case_table_names` permet de forcer la sensibilité à la casse pour les noms des tables et des bases de données.

Les commentaires et conventions

- ▶ **Conventions de MySQL :**
 - Tous les mots-clés de SQL sont notés en majuscules.
 - Les noms de tables sont notés en Minuscules (excepté la première lettre, mais stockés dans le système en minuscules).
 - Les noms de colonnes et de contraintes en minuscules.
- ▶ **Un commentaire** se déclare : sur une même ligne précédée de deux tirets « -- », en fin de ligne à l'aide du dièse « # », au sein d'une ligne ou sur plusieurs lignes entre « /* » et « */ »).

Exemple:

```
CREATE TABLE
    -- nom de la table
    Test( #début de la description
        COLONNE DECIMAL(38,8)
    ) -- fin, ne pas oublier le point-virgule.
;
CREATE TABLE Test (
    /* une plus grande description
    des colonnes */
    COLONNE /* type : */ DECIMAL(38,8));
```



Quelques restrictions de MySQL

- ▶ Nom des colonnes **unique** pour une même table;
- ▶ Les colonnes SET sont évaluées par des chaînes de caractères séparés par des « , » ('Airbus, Boeing'). Ainsi aucune valeur d'un SET ne doit contenir le symbole « , ».
- ▶ Les noms des objets (base, tables, colonnes, contraintes, vues, etc.) ne doivent pas emprunter des mots-clés de MySQL : TABLE, SELECT, INSERT, IF...etc.



SQL de Base de MySQL



SQL de base: CREATE TABLE

- ▶ La syntaxe simplifié est la suivante:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] [nomBase.]nomTable
( colonne1 type1
[NOT NULL | NULL] [DEFAULT valeur1] [COMMENT 'chaine1' ]
[, colonne2 type2
[NOT NULL | NULL] [DEFAULT valeur2] [COMMENT 'chaine2' ] ]
[CONSTRAINT nomContrainte1 typeContrainte1] ... )
[ENGINE= InnoDB | MyISAM | ...];
```



Exemple avec CREATE TABLE

```
CREATE TABLE bdbeile.Compagnie  
(comp CHAR(4),  
n rue INTEGER(3),  
rue CHAR(20),  
ville CHAR(15) DEFAULT 'Paris'  
COMMENT 'Par defaut : Paris',  
nomComp CHAR(15) NOT NULL);
```

Les contraintes dans les tables

- ▶ But: programmer des règles de gestion au niveau des colonnes des tables. Elles peuvent alléger un développement côté client.
- ▶ Elles se déclarent de deux manières:
 - En même temps que la colonne: contraintes « en ligne » et/ou **mono-colonne**,
 - Après déclaration d'une colonne, celles-ci ne sont pas limitées à une seule colonne et personnalisables par un nom;
- ▶ La contrainte NOT NULL est déclarée « en ligne » (recommandé)

Les contraintes les plus utilisées

CONSTRAINT *nomContrainte*

UNIQUE (*colonne1 [,colonne2]...*)

PRIMARY KEY (*colonne1 [,colonne2]...*)

FOREIGN KEY (*colonne1 [,colonne2]...*)

REFERENCES *nomTablePere* [*(colonne1 [,colonne2]...)*]

[ON DELETE {RESTRICT|CASCADE|SET NULL|NO ACTION}]

[ON UPDATE {RESTRICT|CASCADE|SET NULL|NO ACTION}]

CHECK (*condition*)

(exemple: CHECK (note BETWEEN 0 AND 20), CHECK
(grade='Copilote' OR grade='Commandant')).



Les contraintes : exemple

```
CREATE TABLE Pilote
(brevet CHAR(6), nom CHAR(15) NOT NULL,
nbHVol DECIMAL(7,2), compa CHAR(4),
CONSTRAINT pk_Pilote PRIMARY KEY(brevet),
CONSTRAINT ck_nbHVol
CHECK(nbHVol BETWEEN 0 AND 20000),
CONSTRAINT un_nom UNIQUE (nom),
CONSTRAINT fk_Pil_compa_Comp
FOREIGN KEY (compa)
REFERENCES Compagnie(comp));
```

Les types des colonnes:caractères

Pour décrire les colonnes d'une table, MySQL fournit les types prédéfinis suivants (*built-in datatypes*) : parmi les types caractères disponibles, il y'en a:

- ▶ CHAR, chaîne de caractères à taille fixe (max:255 cars);
- ▶ VARCHAR, chaîne de caractères à taille variable (max:65535 cars);
- ▶ BINARY et VARBINARY sont similaires à CHAR et VARCHAR, mais contiennent des chaînes d'octets , pas de jeu de caractères en particulier.
- ▶ Les quatre types permettant aussi de stocker du texte sont TINYTEXT, TEXT, MEDIUMTEXT, et LONGTEXT. Ils sont associés à un jeu de caractères, pas de DEFAULT, ni de suppression d'espaces de fin.

Les types de colonnes: les nombres

Valeurs exactes, positifs ou négatifs

- ▶ INTEGER (entier sur 4 octets. signé ou non)
- ▶ SMALLINT (entier sur 2 octets. signé ou non)
- ▶ BOOL ou BOOLEAN (valeur booléenne 0 ou 1)

Valeurs à virgule fixe ou flottante

- ▶ FLOAT, nombre à virgule flottante (sur 4 à 8 octets
Avec 7decimales max)
- ▶ DOUBLE, nombre à virgule flottante (sur 8 octets
Avec 15decimales max)
- ▶ DECIMAL , décimal à virgule fixe (max 30 chiffres
après virgule)

Les entiers restreints (TINYINT, MEDIUMINT et BIGINT).

Les types de colonnes : Dates/heures

- ▶ DATE (sur 3 octets, du 01/01/1000 au 31/12/9999, format ‘YYYY-MM-DD’);
- ▶ DATETIME (date et heures, sur 8 octets. ‘YYYY-MM-DD HH:MM:SS’)
- ▶ YEAR (année, sur 2 (70 à 69 [1970–2069] ou 4 position (1901–2155), 1 octet, format ‘YYYY’)
- ▶ TIME (heure, sur 3 octets, de -838h59m59s à 838h59m59s, format ‘HHH:MM:SS’)
- ▶ TIMESTAMP (instants du 01/01/1970 0h0m0s à l'an 2037, mis à jour à chaque modification de la table)

Les fonctions NOW() et SYSDATE() retournent la date et l'heure courante du système.

Les autres types de colonnes

Données binaires :

- ▶ Les quatre types BLOB (*Binary Large Object*) [TINYBLOB, BLOB, MEDIUMBLOB et LONGBLOB] permettent de stocker des données non organisées comme le multimédia (images, sons, vidéo, etc.), considérés comme des flux d'octets sans jeu de caractère.

Deux types collections proposés par MySQL:

- ▶ ENUM définit une liste de valeurs permises (chaînes de caractères).
- ▶ SET permet de comparer une liste à une combinaison de valeurs permises à partir d'un ensemble de référence (chaînes de caractères).

DESCRIBE ou structure d'une table

- ▶ Elle permet d'afficher la structure d'une table ou d'une vue. Sa syntaxe est la suivante :

DESCRIBE [nomBase.] nomTableouVue [colonne];

- ▶ Exemple :

```
mysql> DESCRIBE Pilote;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| brevet | char(6) | NO   | PRI |          |        |
| nom    | char(15) | YES  | UNI | NULL    |        |
| nbHVol | double(7,2)| YES |      | NULL    |        |
| compa  | char(4)  | YES  | MUL | NULL    |        |
+-----+-----+-----+-----+-----+
mysql> DESCRIBE Compagnie;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| comp  | char(4) | NO   | PRI |          |        |
| nrue  | int(3)  | YES  |      | NULL    |        |
| rue   | char(20) | YES |      | NULL    |        |
| ville | char(15) | YES |      | Paris   |        |
| nomComp | char(15) | NO   |      |          |        |
+-----+-----+-----+-----+-----+
```

Index d'une colonne avec MySQL

- ▶ Il permet d'accélérer l'accès aux données d'une table, utile pour éviter de parcourir une table séquentiellement .
- ▶ Un index est associé à une table, plus ou moins défini sur une ou plusieurs colonnes (dites « indexées »), peut être déclaré unique dans le besoin.
- ▶ Une table peut « accueillir » plusieurs index ou aucun, les colonnes indexées sont mis à jour obligatoirement après rafraîchissement de la table.

Création d'index sur une colonne

- ▶ Avoir le privilège INDEX sur la base concernée pour créer ou supprimer des index.
- ▶ CREATE INDEX permet de créer un index sur une colonne;
- ▶ DROP INDEX permet de le supprimer;
- ▶ La syntaxe est la suivante :

CREATE [UNIQUE | FULLTEXT | SPATIAL] **INDEX**
nomIndex

[USING BTREE | HASH]

ON *nomTable* (*colonne1* [(*taille1*)] [ASC | DESC], . . .) ;



Exemple de déclaration d'index

```
CREATE INDEX idx_Pilote_compa  
USING BTREE  
ON Pilote (compa);
```

```
CREATE UNIQUE INDEX idx_Pilote_nom3  
USING BTREE  
ON Pilote (nom(3) DESC);
```



Remarques sur les index

- ▶ Un index ralentit l'actualisation de la base, mais accélère l'accès.
- ▶ Il est recommandé de créer des index sur des colonnes utilisées dans les clauses de jointures.
- ▶ Possibilité de créer des index sur les toutes les colonnes d'une table (max 16).
- ▶ Pénalisant sur une table très souvent modifiée, ou contenant peu des enregistrements.

Destruction d'une table : DROP TABLE

- ▶ Posséder le privilège DROP sur cette base.
- ▶ L'ordre de destruction des tables est table « fils » ensuite « père ».
- ▶ DROP TABLE entraîne la suppression des données, de la structure, de la description dans le dictionnaire des données, des index, des déclencheurs associés (*triggers*) et la récupération de la place dans l'espace de stockage. Syntaxe :

```
DROP [TEMPORARY] TABLE [IF EXISTS]  
[nomBase.] nomTable1  
[, [nomBase2.] nomTable2, ...]  
[RESTRICT | CASCADE]
```



Exemple de destruction d'une table

▶ Avec CASCADE

```
DROP TABLE Compagnie CASCADE;
```

```
DROP TABLE Pilote;
```

▶ « fils » puis « père »

```
DROP TABLE Pilote;
```

```
DROP TABLE Compagnie;
```



FIN DU CHAPITRE



Base de Données Avancées

COURS 2

Langage de Manipulation et d'Interrogation
de Données avec MySQL



Plan de la présentation

- ▶ Insertion des données (INSERT INTO)
- ▶ Modification des données (UPDATE)
- ▶ Commande REPLACE
- ▶ Suppression des données (DELETE)
- ▶ L'instruction (TRUNCATE)
- ▶ Intégrité référentiel
- ▶ L'instruction (LOAD DATA IN FILE)
- ▶ Instruction (RENAME ou RENAME TO)
- ▶ Instruction (ALTER TABLE)
- ▶ Interrogation de la base des données (SELECT)



Syntaxe de INSERT INTO

- ▶ Avoir le privilège INSERT sur la base concernée.
- ▶ **INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY]
[IGNORE] [INTO] [nomBase.] { nomTable / nomVue }
[(nomColonne,...)]
VALUES ({expression / DEFAULT},...),(...),...
[ON DUPLICATE KEY UPDATE nomColonne =
expression,...]**



Insertion des données : INSERT

- ▶ INSERT INTO Compagnie VALUES ('SING', 7, 'SingaVille', 'Singapour', 'Singapore AL');
- ▶ INSERT INTO Compagnie VALUES ('AC', 10, 'Gambetta', **DEFAULT**, 'Air France');
- ▶ INSERT INTO Compagnie VALUES ('AN1', **NULL**, 'Janale', 'Djibouti', 'Air Null');
- ▶ INSERT INTO Compagnie(comp, nrue, rue, nomComp) VALUES ('AF', 8, 'Moud.Harbi', 'Djib Air');
- ▶ INSERT INTO Compagnie(comp, rue, ville, nomComp) VALUES ('AN2', 'Gashamaleh', 'Djibouti', 'Air Nul2');

Insertion de plusieurs lignes

```
INSERT INTO Compagnie VALUES
('LUFT', 9, 'Salas', 'Munich', 'Luftansa'),
('QUAN', 1, 'Kangoro', 'Sydney', 'Quantas'),
('SNCM', 3, 'P. Paoli', 'Bastia', 'Corse Air');
```

Quelques erreurs avec INSERT

- ▶ mysql> INSERT INTO Pilote VALUES ('**PL-1**' ,
'Amélie Sulpice' , 100 , 'AF') ;
- ▶ **ERROR 1062 (23000) : Duplicate entry 'PL-1'
for key 1.**

Quelques erreurs avec INSERT

- ▶ mysql> INSERT INTO Pilote VALUES ('PL-4',
'Said Omar', 450, 'AF');
- ▶ ERROR 1062 (23000) : **Duplicate entry** 'Said Omar' for key 2.
- ▶ mysql> INSERT INTO Pilote VALUES ('PL-5', 'Thomas Sulpice', 500, 'TOTO');
- ▶ ERROR 1452 (23000): Cannot add or update a child row:
a foreign key constraint fails (`bdtest/pilote`,
CONSTRAINT `fk_Pil_compa_Comp` FOREIGN KEY
(`compa`) REFERENCES `compagnie`(`comp`))
- ▶ mysql> INSERT INTO Pilote VALUES ('PL-6', NULL, 100,
'AF');
- ▶ ERROR 1048 (23000): Column '**nom**' cannot be null.

Insertion données binaires (BIT)

- Le type BIT permet de manipuler des suites des bits. La table suivante contient deux colonnes de ce type, exemple :

```
CREATE TABLE Registres (nom CHAR(5),  
numero BIT(2), adresse BIT(16)) ;  
INSERT INTO Registres VALUES ('COM2',  
b'10' , b'0000110110001101' ) ;
```



Insertion de données de type ENUM

- ▶ *CREATE TABLE Cursus(num CHAR(4), nom CHAR(15), diplome ENUM ('BTS','DUT','Licence','M1'), CONSTRAINT pk_Cusus PRIMARY KEY(num));*
- ▶ *mysql> INSERT INTO Cursus VALUES ('C1', 'C. Site1', ('BTS'));*
- ▶ *mysql> INSERT INTO Cursus VALUES ('C2', 'C. Site2', 'Licence');*
- ▶ *mysql> INSERT INTO Cursus VALUES ('C3', 'Bug', ('MathInfo));*
- ▶ **ERROR 1265 (01000): Data truncated for column 'diplome' at row 1.**

Insertion de données de type SET

- ▶ `CREATE TABLE Cursus2(num CHAR(4), nom CHAR(15), diplomes SET('BTS', 'DUT', 'Licence', 'M1'), CONSTRAINT pk_Cusus PRIMARY KEY(num));`
- ▶ `mysql> INSERT INTO Cursus VALUES ('C1', 'C. Site1', ('BTS', 'Licence'));`
- ▶ `mysql> INSERT INTO Cursus VALUES ('C2', 'C. Site2', ('Licence', 'DUT', 'M1'));`
- ▶ `mysql> INSERT INTO Cursus VALUES ('C3', 'Bug', ('MathInfo', 'DUT', 'Licence'));`
- ▶ **ERROR 1265 (01000): Data truncated for column 'diplomes' at row 1.**

Insertion type Date/Heure

Pour les type DATETIME, DATE et TIMESTAMP les formats possibles sont les suivants :

- ▶ Chaînes de caractères 'YYYY-MM-DD HH:MM:SS' ou 'YY-MM-DD HH:MM:SS' (format DATE 'YYYY-MM-DD' ou 'YY-MM-DD'). Tout autre délimiteur est autorisé comme '.', '/', '@';
- ▶ Chaînes de caractères dans les formats suivants : 'YYYYMMDDHHMMSS' ou 'YYMMDDHHMMSS', à condition que la chaîne ait un sens en tant que date (interprété ainsi 0000-00-00 00:00:00).
- ▶ Ces 2formats (YYYYMMDDHHMMSS ou YYMMDDHHMMSS,) sont considéré comme des nombres (interprété ainsi 00000000000000).

Insertion de type TIME (Date/heure)

- ▶ Chaîne 'D HH:MM:SS.fraction' avec le nombre de jours (0 à 34) et la fraction de seconde, 'HH:MM:SS.fraction', 'HH:MM:SS', 'HH:MM', 'D HH:MM:SS', 'D HH:MM', 'D HH', ou 'SS'.
- ▶ Chaîne sans les délimiteurs sous réserve que la chaîne ait un sens (format 'HHMMSS').
- ▶ Nombre en raisonnant comme les chaînes (format HHMMSS). Les formats suivants sont aussi corrects : SS, MMSS.

Exemple avec TIME et YEAR

- ▶ CREATE TABLE Pilote (brevet VARCHAR(6), nom VARCHAR(20), pasVolDepuis **TIME**, retraite **YEAR**, CONSTRAINT pk_Pilote PRIMARY KEY(brevet));
- ▶ INSERT INTO Pilote VALUES('PL-1', 'Hassan', '1 23:0:0', '2012');
- ▶ INSERT INTO Pilote VALUES('PL-2', 'Ahmed', '152630', 2016);
- ▶ INSERT INTO Pilote VALUES('PL-3', 'Ali', '4 23:00', 15);
- ▶ INSERT INTO Pilote VALUES('PL-4', 'Abdillahi', 032750, '17');
- ▶ INSERT INTO Pilote VALUES ('PL-5', 'Ammar', '1 23:0:0.457', '18');



L'état de la base après insertion

```
mysql> SELECT * FROM Pilote;
```

| brevet | nom | pasVolDepuis | retraite |
|--------|-----------|--------------|----------|
| PL-1 | Hassan | 47:00:00 | 2012 |
| PL-2 | Ahmed | 15:26:30 | 2016 |
| PL-3 | Ali | 119:00:00 | 2015 |
| PL-4 | Abdillahi | 03:27:50 | 2017 |
| PL-5 | Ammar | 47:00:00 | 2018 |

Exemple avec TIMESTAMP

- ▶ Toute colonne du type **TIMESTAMP** est actualisée à chaque modification de l'enregistrement; exemple :
- ▶

```
CREATE TABLE Pilote (brevet VARCHAR(6), nom VARCHAR(20), misaJour TIMESTAMP, CONSTRAINT pk_Pilote PRIMARY KEY(brevet)) ;
```
- ▶

```
INSERT INTO Pilote(brevet,nom) VALUES ('PL-1','Hait');
```
- ▶ **Les fonctions** CURRENT_TIMESTAMP(), CURRENT_DATE() et CURRENT_TIME(), UTC_TIME(), renseignent sur l'instant, la date, l'heure et l'heure GMT de la session en cours.

La directive AUTO-INCREMENT()

- ▶ Elle est souvent utilisée pour les colonnes de clés primaires, mais peut être utilisé pour toute colonne de type entier indexée unique. Exemple :

```
CREATE TABLE Affreter (numAff SMALLINT  
AUTO_INCREMENT , comp CHAR(4) , immat CHAR(6) ,  
dateAff DATE , nbPax SMALLINT(3) , PRIMARY KEY  
(numAff)) ;
```

Modification de la valeur de départ :

- ▶ ALTER TABLE Affreter **AUTO_INCREMENT = 10**;
- ▶ La fonction `LAST_INSERT_ID()` retourne la dernière valeur de la séquence générée.

Exemple avec LAST_INSERT_ID()

- ▶ CREATE TABLE Passager (numPax SMALLINT **AUTO_INCREMENT** , nom CHAR(15) , siege CHAR(4) , dernierVol SMALLINT , PRIMARY KEY (numPax) , **FOREIGN KEY** (dernierVol) **REFERENCES** Affreter(numAff))
AUTO_INCREMENT = 100;
- ▶ INSERT INTO Affreter
(comp, immat, dateAff, nbPax) VALUES
('SING' , 'F-GAFU' , '2005-02-05' , 155) ;
- ▶ INSERT INTO Passager VALUES
(NULL, 'Payrissat' , '7A' , **LAST_INSERT_ID()**) ;



UPDATE (modification des données)

- ▶ Avoir le privilège UPDATE sur les tables concernées.
- ▶ **UPDATE [LOW_PRIORITY] [IGNORE] [nomBase.]
*nomTable***

SET *col_name1=expr1 [, col_name2=expr2 ...]*

**SET *colonne1 = expression1 / (requête_SELECT) /
DEFAULT[, colonne2 = expression2...]***

[WHERE (*condition*)]

[ORDER BY *listeColonnes*]

[LIMIT *nbreLimite*]

- ▶ ORDER BY : ordre de modification des colonnes
- ▶ LIMIT : nombre max d'enregistrements à modifier.

Exemples avec UPDATE

- ▶ **UPDATE** Compagnie SET nrué = 50
WHERE comp='AN1'; (une colonne, une ligne)
- ▶ **UPDATE** Compagnie SET nrué = 14, ville =
DEFAULT WHERE comp = 'AN2'; (2 cols, une seule
ligne)
- ▶ **UPDATE** Compagnie SET ville = 'Toulouse'
LIMIT 2; (une col, plusieurs lignes)

Quelques Erreurs avec UPDATE

- ▶ mysql>UPDATE Pilote SET brevet='PL-2' WHERE brevet='PL-1';
- ▶ ERROR 1062 (23000): Duplicate entry 'PL-2' for key 1
- ▶ mysql>UPDATE Pilote SET nom = NULL WHERE brevet = 'PL-1';
- ▶ ERROR 1263 (22004): Column set to default value; NULL supplied to NOT NULL column 'nom' at row 1.
- ▶ mysql>UPDATE Pilote SET nom='Said Osman' WHERE brevet='PL-1';
- ▶ ERROR 1062 (23000): Duplicate entry 'Said Omar' for key 2.
- ▶ mysql>UPDATE Pilote SET compa='TOTO' WHERE brevet = 'PL-1';
- ▶ ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`bdtest/pilote`, CONSTRAINT `fk_Pil_compa_Comp` FOREIGN KEY (`compa`) REFERENCES `compagnie`(`comp`))

La commande REPLACE

- ▶ Elle consiste remplacer un enregistrement dans sa totalité (toutes les valeurs de ses colonnes);
- ▶ Avoir le privilège de `INSERT` et `DELETE`
- ▶ Le remplacement se fait selon la clé primaire ou d'un index unique.
- ▶ **REPLACE [LOW_PRIORITY | DELAYED]**
`[INTO] [nomBase.] nomTable [(colonne1,...)]`
`VALUES ({expression1 / DEFAULT},...) [, (...), ...]`
- ▶ **REPLACE INTO Compagnie VALUES ('AN1', 24, 'Salah', 'Randa', 'Air RENATO');**



La commande DELETE

- ▶ Avoir le privilège DELETE sur la table
- ▶ **DELETE** [LOW_PRIORITY] [QUICK] [IGNORE]
FROM [nomBase.] nomTable
[WHERE (*condition*)]
[ORDER BY *listeColonnes*]
[LIMIT *nbreLimite*]
- ▶ ORDER BY: tri sur les enregistrements qui seront effacés.
- ▶ QUICK: (pour les tables de type MyISAM) ne met pas à jour les index associés pour accélérer le traitement.

Exemple avec DELETE

- ▶ **DELETE** FROM Pilote WHERE compa = 'AF' ;
- ▶ **DELETE** FROM Compagnie WHERE comp = 'AF' ;
- ▶ mysql> **DELETE** FROM Compagnie WHERE comp='SING' ;
- ▶ ERROR 1451 (23000): Cannot delete or update a parent row: a **foreign key constraint** fails (`dbtest/pilote`, CONSTRAINT `fk_Pil_compa_Comp` FOREIGN KEY (`compa`) REFERENCES `compagnie`(`comp`));
- ▶ **DELETE** FROM Compagnie **LIMIT** 2; (suppression de 2 premiers rows de la table avec tri croissant selon la clé primaire, à condition qu'aucune clé étrangère ne fait référence).

L'instruction TRUNCATE

- ▶ Suppression de tous enregistrements de la table et libération de la mémoire occupée.
- ▶ **TRUNCATE** [TABLE] [*nomBase.*] *nomTable* ;
- ▶ Avec InnoDB: elle est programmée en DELETE;
- ▶ Avec les autres moteurs, elle diffère:
 - La table est détruite puis récrée, plus rapide que de supprimer les enregistrements un par un.
 - Opération interrompue si table utilisée
 - Pas de nombre d'enregistrements effacés retournés
 - Pas de mémorisation de dernière valeur d'une colonne AUTO_INCREMENT
 - Impossibilité d'exécution , si table référencée par des clés étrangères



Intégrité référentielle

- ▶ Cœur de la cohérence d'une Base de donnée relationnelle;
- ▶ Fondée sur la relation entre clé étrangère et clé primaire;
- ▶ Permet de programmer des règles de gestion et donc de le contrôler à chaque mise à jour.
- ▶ Exemple de règle de gestion: l'affrètement d'un vol doit se faire par une compagnie et pour un avion tous deux existants dans la base de données.

Contrainte d'intégrité référentiel

- ▶ La **contrainte référentielle** concerne toujours deux tables – une table « père » aussi dite « maître » (*parent/referenced*) et une table « fils » (*child/dependent*) – possédant une ou plusieurs colonnes en commun.
- ▶ Pour la table « père », ces colonnes composent la clé primaire.
- ▶ Pour la table « fils », ces colonnes composent une clé étrangère. Elle se programme avec FOREIGN KEY.
- ▶ Une même table peut être « père » pour une contrainte, et « fils » pour une autre.

Schéma intégrité référentiel

Pilote

| brevet | nom | nbHVol | compa |
|--------|-------------|--------|-------|
| PL-1 | Louise Ente | 450 | AF |
| PL-2 | Jules Ente | 900 | AF |
| PL-3 | Paul Soutou | 1000 | SING |

Compagnie *referenced / parent*

dependent / child

| comp | nrue | rue | ville | nomComp |
|------|------|-----------|-----------|--------------|
| AF | 10 | Gambetta | Paris | Air France |
| SING | 7 | Camparols | Singapour | Singapore AL |

Affreter *dependent / child*

| compAff | immat | dateAff | nbPax |
|---------|--------|------------|-------|
| AF | F-WTSS | 2005-05-13 | 85 |
| SING | F-GAFU | 2005-02-05 | 155 |
| AF | F-WTSS | 2005-05-15 | 82 |

Avion

dependent / child

NOT NULL

| immat | typeAvion | nbHVol | proprio |
|--------|-----------|--------|---------|
| F-WTSS | Concorde | 6570 | SING |
| F-GAFU | A320 | 3500 | AF |
| F-GLFS | TB-20 | 2000 | SING |

referenced / parent



Clés composées (composite keys)

- ▶ Les clés primaires ou clés étrangères peuvent être définies sur plusieurs clés, on parle de clé composée ou composite Key;
- ▶ Les clés étrangères peuvent être nulles, si elles ne font pas parties d'une clé primaire, ou si elles ne sont pas définies NOT NULL. Exemple:
- ▶

```
CREATE TABLE Avion (immat CHAR(6), typeAvion
CHAR(15), nbhVol DECIMAL(10,2), proprio
CHAR(4) NOT NULL, PRIMARY KEY (immat),
FOREIGN KEY (proprio) REFERENCES
Compagnie(comp));
```



Exemple avec clés composées

▶ CREATE TABLE Affreter (compAff CHAR(4), immat CHAR(6), dateAff DATE, nbPax INTEGER(3),
PRIMARY KEY (compAff, immat, dateAff),
FOREIGN KEY(immat) **REFERENCES** Avion(immat),
INDEX(compAff), **FOREIGN KEY**(compAff) **REFERENCES**
Compagnie (comp));
-- fils avec père:

INSERT INTO Pilote **VALUES** ('PL-3', 'Said Ali', 10, 'SING');

-- fils sans père:

INSERT INTO Pilote **VALUES** ('PL-4', 'Un Connu', 0, NULL);

-- fils avec pères

INSERT INTO Avion **VALUES** ('F-WTSS', 'Concorde', 6570, 'SING');

INSERT INTO Affreter **VALUES** ('AF', 'F-WTSS', '15-05-2013', 82);



Exemple insertion incorrecte

- ▶ mysql> INSERT INTO Pilote VALUES ('PL-5', 'Pb de Compagnie', 0, '?');
- ▶ INSERT INTO Avion VALUES ('D-YTS', 'Concorde2', 0, NULL);
- ▶ INSERT INTO Affreter VALUES (null, 'D-YTS', '15-05-2018', 52);
- ▶ INSERT INTO Affreter VALUES ('AF', null, '25-05-2013', 52);
- ▶ **ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`dbtest/pilote`, CONSTRAINT `fk_Pil_compa_Comp` FOREIGN KEY (`compa`) REFERENCES `compagnie` (`comp`))**



Options avec FOREIGN KEY

- ▶ [ON DELETE | ON UPDATE] NO ACTION | RESTRICT : (pas d'option), aucune modification ou suppression de la valeur de la clé primaire ne sera exécutée tant qu'elle corresponde à une valeur d'une clé étrangère faisant référence.
- ▶ [ON DELETE | ON UPDATE] CASCADE: toute modification ou suppression d'une valeur de la clé primaire sera étendue aux valeurs de clé étrangère faisant référence.
- ▶ [ON DELETE | ON UPDATE] SET NULL : toute modification ou suppression d'une valeur de la clé primaire entraînera des valeurs nulles pour la clé étrangère faisant référence, si elle l'accepte.

Insertion à partir d'un fichier

- ▶ L'importation de données (au format fichier texte) dans une table peut être programmée à l'aide de la directive `LOAD DATA INFILE`. Elle permet d'insérer tout ou partie des données contenu dans le fichier.
- ▶ **LOAD DATA INFILE** '*nomEtCheminFichier.txt*'
[REPLACE | IGNORE] INTO TABLE *nomTable*
[FIELDS [TERMINATED BY '*string*']
[[OPTIONALLY] ENCLOSED BY '*char*']
[ESCAPED BY '*char*']]
[LINES [STARTING BY '*string*'] [TERMINATED BY '*string*']]
[IGNORE *number* LINES];
- ▶ REPLACE : option à utiliser pour remplacer systématiquement les anciens enregistrements par les nouveaux (valeur de clé primaire ou d'index unique).
- ▶ IGNORE : fait en sorte de ne pas insérer des enregistrements de clé primaire ou d'index unique déjà présents dans la table.

Options avec LOAD DATA INFILE

- ▶ FIELDS décrit comment sont formatées dans le fichier les valeur à insérer dans la table. En l'absence de cette clause, TERMINATED BY vaut '\t', ENCLOSED BY vaut " et ESCAPED BY vaut '\\'.
 - FIELDS TERMINATED BY décrit le caractère qui sépare deux valeurs de colonnes.
 - FIELDS ENCLOSED BY permet de contrôler le caractère qui encadrera chaque valeur de colonne.
 - FIELDS ESCAPED BY permet de contrôler les caractères spéciaux.
- ▶ LINES décrit comment seront écrites dans le fichier les lignes extraites de(s) table(s). En l'absence de cette clause, TERMINATED BY vaut '\n' et STARTING BY vaut ".
- ▶ IGNORE permet de ne pas importer les nb premières lignes du fichier (contenant des éventuelles déclarations).



Exemple avec LOAD DATA INFILE

```
LOAD DATA INFILE 'D:\\dev\\\\pilotes.txt'
REPLACE INTO TABLE Pilote2
FIELDS TERMINATED BY ';' ENCLOSED BY ''''
LINES STARTING BY 'import-Pilote'
TERMINATED BY '$'
```

Instruction RENAME ou RENAME TO

- ▶ L'instruction RENAME renomme une ou plusieurs tables ou vues. Il faut posséder le privilège ALTER et DROP sur la table d'origine, et CREATE sur la base. Sa syntaxe est la suivante :

```
RENAME [nomBase.]oldNameTable TO [nomBase.]  
newNameTable;
```

- ▶ Les contraintes d'intégrité, index et prérogatives associés à l'ancienne table sont automatiquement transférés sur la nouvelle. En revanche, les vues et procédures cataloguées sont invalidées et doivent être recréées.
- ▶ Il est aussi possible d'utiliser l'option RENAME TO de l'instruction ALTER TABLE. Exemples :
- ▶ **RENAME** Pilote **TO** Naviguant;
- ▶ **ALTER TABLE** Pilote **RENAME TO** Naviguant;

ALTER TABLE : ajout des colonnes

- ▶ La directive ADD de l'instruction ALTER TABLE permet d'ajouter une nouvelle colonne à une table.
- ▶ Il est possible d'ajouter une colonne en ligne NOT NULL seulement si la table est vide ou si une contrainte DEFAULT est définie sur la nouvelle colonne. Considérons la table suivante :
- ▶ CREATE TABLE Pilote (brevet VARCHAR(4) , nom VARCHAR(20)) ;
- ▶ INSERT INTO Pilote VALUES ('PL -1' , 'Ahmed ALI') ;
- ▶ **ALTER TABLE** Pilote **ADD** (nbHVol DECIMAL(7,2)) ;
- ▶ **ALTER TABLE** Pilote **ADD** (compa VARCHAR(4) DEFAULT 'AF' , ville VARCHAR(30) DEFAULT 'Paris' NOT NULL) ;

ALTER TABLE : renommer des colonnes

- ▶ Il faut utiliser l'option CHANGE de l'instruction ALTER TABLE pour renommer une colonne existante. La syntaxe générale de cette option est la suivante :
- ▶ **ALTER TABLE** [nomBase].nomTable **CHANGE** [COLUMN] ancienNom nouveauNom typeMySQL [NOT NULL | NULL] [DEFAULT valeur] [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY] [COMMENT 'chaine'] [REFERENCES ...] [FIRST|AFTER nomColonne]; **exemple :**
- ▶ **ALTER TABLE** Pilote **CHANGE** ville adresse VARCHAR(30) **AFTER** nbHVol;

ALTER TABLE: Modifier le type

- ▶ L'option **MODIFY** de l'instruction **ALTER TABLE** modifie le type d'une colonne existante sans pour autant la renommer.
- ▶ **ALTER TABLE** [nomBase] .nomTable **MODIFY** [COLUMN] nomColonneAModifier typeMySQL [NOT NULL | NULL] [DEFAULT valeur] [AUTO INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY] [COMMENT 'chaine'] [REFERENCES ...] [FIRST|AFTER nomColonne];
- ▶ Ainsi, Il est possible d'augmenter la taille d'une colonne ou de la diminuer si toutes les données présentes dans la colonne peuvent s'adapter à la nouvelle taille (attention aux colonnes indexées).
- ▶ Les contraintes en ligne peuvent être aussi modifiées par cette instruction.



Exemple modification de type :

- ▶ **ALTER TABLE Pilote MODIFY compa VARCHAR(6) DEFAULT 'SING';**
- ▶ **INSERT INTO Pilote (brevet, nom) VALUES ('PL-2', 'Liban Ali');**
- ▶ **ALTER TABLE Pilote MODIFY compa CHAR(4) NOT NULL;**
- ▶ **ALTER TABLE Pilote MODIFY compa CHAR(4);**

Modifier valeur par défaut

- ▶ L'option **ALTER COLUMN** de l'instruction **ALTER TABLE** modifie la valeur par défaut d'une colonne existante.
- ▶ **ALTER TABLE** [nomBase].nomTable **ALTER [COLUMN]** nomColonneAModifier { **SET DEFAULT** 'chaine' | **DROP DEFAULT** } ;
- ▶ **ALTER TABLE** Pilote **ALTER COLUMN** adresse **SET DEFAULT** 'Djibouti' ;
- ▶ **ALTER TABLE** Pilote **ALTER COLUMN** compa **DROP DEFAULT** ;

Supprimer une colonne

- ▶ L'option **DROP** de l'instruction **ALTER TABLE** permet de supprimer une colonne. Lorsqu'une colonne est supprimée, les index qui l'utilisent sont mis à jour voire éliminés si toutes les colonnes qui le composent sont effacées.

ALTER TABLE [nomBase].nomTable **DROP**

```
{ [COLUMN] nomColonne | PRIMARY KEY  
| INDEX nomIndex | FOREIGN KEY nomContrainte }
```

- ▶ Limites ; suppression impossible pour :
 - toutes les colonnes d'une table ;
 - les colonnes qui sont clés primaires (ou candidates par UNIQUE) référencées par des clés étrangères.

ALTER TABLE Pilote **DROP COLUMN** adresse;

Ajout d'une contrainte

- ▶ La directive **ADD CONSTRAINT** de l'instruction **ALTER TABLE** permet d'ajouter une contrainte à une table.

```
ALTER TABLE [nomBase] . nomTable ADD {  
CONSTRAINT nomContrainte typeContrainte }
```

- ▶ Trois types de contraintes sont possibles :
 - UNIQUE (*colonne1 [,colonne2]...*)
 - PRIMARY KEY (*colonne1 [,colonne2]...*)
 - FOREIGN KEY (*colonne1 [,colonne2]...*)
 REFERENCES *nomTablePère (col1 [,col2]...)*
 [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
 [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]



Exemples: ajout contraintes

- ▶ **ALTER TABLE Compagnie ADD CONSTRAINT un_nomC UNIQUE (nomComp);**
- ▶ **ALTER TABLE Avion ADD CONSTRAINT fk_Avion_comp_Compag FOREIGN KEY(proprio) REFERENCES Compagnie(comp);**
- ▶ **ALTER TABLE Avion MODIFY proprio CHAR(4) NOT NULL;**
- ▶ **ALTER TABLE Affrete ADD (CONSTRAINT pk_Affreter PRIMARY KEY (compAff, immat, dateAff), CONSTRAINT fk_Aff_na_Avion FOREIGN KEY(immat) REFERENCES Avion(immat), CONSTRAINT fk_Aff_comp_Compag FOREIGN KEY(compAff) REFERENCES Compagnie(comp));**
- ▶ Pour que l'ajout ou la modification d'une contrainte soient possibles, il faut que les données présentes dans la table concernée ou référencée respectent la nouvelle contrainte.

Supprimer une contrainte

▶ Contrainte *NOT NULL*:

```
ALTER TABLE Avion MODIFY proprio CHAR(4) NULL;
```

▶ Contrainte *Unique*:

```
ALTER TABLE Compagnie DROP INDEX un_nomC;
```

▶ Contrainte de *clé étrangère*:

```
ALTER TABLE [nomBase].nomTable DROP FOREIGN KEY  
nomContrainte;
```

▶ Contrainte de *clé primaire*:

```
ALTER TABLE Affreter DROP FOREIGN KEY  
fk_Aff_na_Avion;
```

```
ALTER TABLE Affreter DROP FOREIGN KEY  
fk_Aff_comp_Compag;
```

```
ALTER TABLE Affreter DROP PRIMARY KEY;
```

Désactivation contrainte

- ▶ La désactivation des contraintes référentielles peut être intéressante pour :
 - Accélérer des procédures de chargement d'importation et d'exportation massives de données externes.
 - Améliorer aussi les performances de programmes *batchs* qui ne modifient pas des données concernées par l'intégrité référentielle.
- ▶ L'instruction `SET FOREIGN KEY CHECKS=0` permet de désactiver temporairement (jusqu'à la réactivation) toutes les contraintes référentielles d'une base.
- ▶ L'instruction `SET FOREIGN KEY CHECKS=1` permet de réactiver toutes les contraintes référentielles d'une base.

Interrogation de la base

- ▶ l'extraction de données par requêtes (nom donné aux instructions SELECT).
- ▶ Une requête permet de rechercher des données dans une ou plusieurs tables ou vues, à partir de critères simples ou complexes.
- ▶ Les instructions SELECT peuvent être exécutées dans l'interface de commande ou au sein d'un programme SQL (procédure cataloguée), PHP, Java, C, etc.

Interrogation de la base

- ▶ Avoir le privilège SELECT sur la table. Syntaxe :

```
SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes
FROM nomTable1 [,nomTable2]...
[ WHERE condition ]
[ clauseRegroupement ]
[ HAVING condition ]
[ clauseOrdonnancement ]
[ LIMIT [rangDépart,] nbLignes ] ;
```

- ▶ DISTINCT et DISTINCTROW ne pas inclure les duplicates.
- ▶ ALL option par défaut;
- ▶ listeColonnes : { * | expression1 [[AS] alias1] [, expression2 [[AS] alias2] ...]}
 - * : extrait toutes les colonnes de la table.
 - expression : nom de colonne, fonction SQL, constante ou calcul.
 - alias : renomme l'expression (nom valable pendant la durée de la requête).

Interrogation sur la base (2)

- ▶ FROM désigne la table (qui porte un alias ou non) à interroger.
- ▶ clauseRegroupement : GROUP BY (*expression1[,expression2]...*) permet de regrouper des lignes selon la valeur des expressions (colonnes, fonction, constante, calcul).
- ▶ HAVING condition : pour inclure ou exclure des lignes aux groupes (la condition ne peut faire intervenir que des expressions du GROUP BY).
- ▶ clauseOrdonnancement : tri sur une ou plusieurs colonnes ou expressions.
- ▶ LIMIT pour limiter le nombre de lignes après résultat.



Interrogation avec concaténation

- ▶ La fonction CONCAT est utilisé pour concaténer deux valeurs, elle admet deux chaînes de caractères en paramètre.
- ▶ Elle permet de concaténer différentes expressions (colonnes, calculs, résultats de fonctions SQL ou constantes), sous réserve d'éventuelles conversions (*casting*).
- ▶ Le résultat est considéré comme une chaîne de caractères.

Insertion multilignes

- ▶ Dans l'exemple suivant, il s'agit d'insérer tous les pilotes de la table Pilote (en considérant le nom, le nombre d'heures de vol et la compagnie) dans la table NomsetHVoldesPilotes :
- ▶ CREATE TABLE NomsetHVoldesPilotes (nom VARCHAR(16), nbHVol DECIMAL(7, 2), compa CHAR(4));

```
INSERT INTO NomsetHVoldesPilotes
SELECT nom, nbHVol, compa FROM Pilote;
```

- ▶ CREATE TABLE NomsetHVoldesPilotes AS SELECT nom, nbHVol, compa FROM Pilote;

Limitation des nombres des lignes

- ▶ Pour limiter le nombre de lignes à extraire à partir du résultat d'une requête, il faut spécifier la clause **LIMIT** de la manière suivante :
- ▶ `LIMIT [rangDépart,] nbLignes`
- ▶ `SELECT * FROM Pilote LIMIT 1,2;`
- ▶ `SELECT * FROM Pilote ORDER BY nbHvol DESC LIMIT 2;`
- ▶ Le premier entier précise le **rang de la première ligne sélectionnée** (en fonction du tri du résultat). Le second entier indique le **nombre maximum de lignes à extraire**. La première ligne est considérée comme présente au **rang 0**.

Opérateurs logiques avec WHERE

- ▶ L'ordre de priorité des opérateurs logiques est NOT, AND et OR.
- ▶ Les opérateurs de comparaison (>, =, <, >=, <=, <>) sont prioritaires par rapport à NOT.
- ▶ Les parenthèses permettent de modifier ces règles de priorité. Exemples :
- ▶ SELECT brevet, nom, compa FROM Pilote WHERE (compa = 'SING' **OR** compa = 'AF' **AND** nbHVol < 500) ;
- ▶ SELECT brevet, nom, compa FROM Pilote WHERE ((compa = 'SING' **OR** compa = 'AF') **AND** nbHVol < 500) ;

Opérateurs intégrés pour WHERE

- ▶ BETWEEN *limiteInf AND limiteSup* : teste l'appartenance à un intervalle de valeurs.
- ▶ IN (*listeValeurs*) compare une expression avec une liste de valeurs.
- ▶ LIKE (*expression*) compare de manière générique des chaînes de caractères à une expression. Le symbole % remplace un ou plusieurs caractères. Le symbole _ remplace un caractère. Ces symboles peuvent se combiner.
- ▶ IS NULL compare une expression (colonne, calcul, constante) à la valeur NULL. La négation s'écrit soit « expression IS NOT NULL » soit « NOT (expression IS NULL) ».

Les fonctions MySQL avec SELECT

- ▶ MySQL propose un grand nombre de fonctions qui s'appliquent dans les clauses SELECT ou WHERE d'une requête.
nomFonction(colonne1 | expression1 [, colonne2 | expression2 ...])
- ▶ On distingue 2 catégories des fonctions MySQL :
- ▶ Fonctions monolignes : agit sur une ligne à la fois et ramène un résultat par ligne. On distingue quatre familles de fonctions monolignes : caractères, numériques, dates et conversions de types de données. Ces fonctions peuvent se combiner entre elles (exemple : MAX (COS (ABS (n))))
- ▶ Fonctions multilignes : agit sur un ensemble de lignes pour ramener un résultat.
- ▶ Exemple : CONCAT(c1, c2);
- ▶ SELECT **CONCAT**(**CONCAT**(nom, ' vole pour '), compa) "Personnel" FROM Pilote;

Les fonctions de groupe avec SELECT

| Fonction | Objectif |
|-----------------------------------|--|
| AVG ([DISTINCT] expr) | Moyenne de expr (nombre). |
| COUNT ({ * [DISTINCT] expr}) | Nombre de lignes (* toutes les lignes, expr pour les colonnes non nulles). |
| GROUP_CONCAT (expr) | Composition d'un ensemble de valeurs. |
| MAX ([DISTINCT] expr) | Maximum de expr (nombre, date, chaîne). |
| MIN ([DISTINCT] expr) | Minimum de expr (nombre, date, chaîne). |
| STDDEV (expr) | Écart type de expr (nombre). |
| SUM ([DISTINCT] expr) | Somme de expr (nombre). |
| VARIANCE (expr) | Variance de expr (nombre). |

- À l'exception de COUNT, toutes les fonctions ignorent les valeurs NULL (il faudra utiliser IFNULL pour contrer cet effet).
- Utilisées sans GROUP BY, ces fonctions s'appliquent à la totalité ou à une seule partie d'une table avec WHERE.



SELECT avec GROUP BY

- ▶ SELECT *col1, fonction1Grpe(...)*
FROM *nomTable* [WHERE *condition*]
GROUP BY {*col1* | *expr1* | *position1*}
[HAVING *condition*] [ORDER BY {*col1* | *expr1*
| *position1*} [ASC | DESC]]
- ▶ Utilisées avec GROUP BY, les fonctions s'appliquent désormais à chaque regroupement. Exemples :
- ▶ SELECT compa, **AVG**(nbHVol), **AVG**(prime) FROM Pilote **GROUP BY**(compa) ;
- ▶ SELECT compa, **COUNT**(*) ,**COUNT**(nbHVol) FROM Pilote **GROUP BY**(compa) ;



SELECT avec GROUP BY (autres exemples)

- ▶ SELECT compa, **STDDEV**(prime), **SUM**(nbHVol)
FROM Pilote **WHERE** typeAvion = 'A320'
GROUP BY(compa) ;
- ▶ SELECT compa, typeAvion, **COUNT**(brevet)
FROM Pilote **GROUP BY** compa, typeAvion ;
- ▶ SELECT compa, **COUNT**(brevet) FROM Pilote
GROUP BY(compa) **HAVING COUNT**(brevet) >=2 ;
- ▶ SELECT compa, **STDDEV**(prime), **SUM**(nbHVol)
FROM Pilote **WHERE** typeAvion = 'A320'
GROUP BY(compa) **HAVING SUM**(nbHVol) >=2000 ;



FIN DU CHAPITRE



Base de Données Avancées

COURS 3

Langage de Contrôle de Données et Gestion
des utilisateurs avec MySQL



Plan de la présentation

- ▶ Gestion utilisateurs
 - Classification des utilisateurs
 - Création d'un utilisateur
 - Modification mot de passe
 - Renommer ou supprimer un user
- ▶ Gestion base des données
 - Création base des données
 - Sélection base des données
- ▶ Privilèges d'un user



Classification des utilisateurs

- ▶ **Le DBA (*DataBase Administrator*)**. Il en existe au moins un. Une petite base peut n'avoir qu'un seul administrateur. Une base importante peut en regrouper plusieurs. Parmi les tâches d'administration: – installation et mises à jour de la base et de ses outils, gestion de l'espace disque, gestion des utilisateurs..etc.
- ▶ **L'administrateur réseau** (peut être le DBA) se charge de la configuration des couches client pour les accès distants.
- ▶ **Les développeurs** qui conçoivent et mettent à jour la base, peuvent agir sur leurs objets (création et modification des tables, index, vues, etc.).
- ▶ **Les administrateurs d'application** qui gèrent les données manipulées par la ou les applications. Pour les petites et les moyennes bases, le DBA joue ce rôle.
- ▶ **Les utilisateurs** qui se connectent et interagissent avec la base.



Création d'un utilisateur

- ▶ Avoir le privilège CREATE USER ou INSERT sur la base système mysql. La syntaxe est la suivante :
- ▶ **CREATE USER** utilisateur [IDENTIFIED BY [PASSWORD] '*motdePasse*' [, utilisateur2 [IDENTIFIED BY [PASSWORD] '*motdePasse2*'] ;
- ▶ IDENTIFIED BY *motdePasse* permet d'affecter un mot de passe (16 cars max, sensibles à la casse) à un utilisateur (16 cars max, sensibles à la casse).

CREATE USER beille@localhost IDENTIFIED BY 'asc';

- ▶ Les utilisateurs, une fois créés, ont le droits suivants
 - Lecture écriture sur la base test;
 - Lecture seule sur la base information_schema.

Liste des utilisateurs

- ▶ (root@localhost) [mysql]
mysql> SELECT User, Host FROM user;
- ▶ L'utilisateur vide '' correspond à une connexion *anonyme*.
- ▶ La machine désignée par « % » indique que la connexion est autorisée à partir de tout site.
- ▶ La machine désignée par « *localhost* » spécifie que la connexion est autorisée en local.

| User | Host |
|--------|-----------|
| | % |
| | localhost |
| root | localhost |
| soutou | localhost |

Modification mot de passe User

- ▶ Avoir le privilège CREATE USER (ou le privilège UPDATE sur la base de données mysql).
- ▶ Utilisez l'instruction UPDATE pour modifier le mot de passe d'un user, exemple:
- ▶ **UPDATE** mysql.user

```
SET Password = PASSWORD('salaam')
WHERE User='beille' AND Host= 'localhost';
FLUSH PRIVILEGES;
```
- ▶ La fonction **PASSWORD()** est utilisée pour coder le password.
- ▶ L'instruction **FLUSH PRIVILEGES** recharge les tables systèmes.

Renommer ou supprimer un user

- ▶ Avoir le privilège CREATE USER (ou le privilège UPDATE sur la base de données mysql).
- ▶ **RENAME USER** utilisateur TO nouveauNom;
- ▶ **RENAME USER** beille@localhost **TO** souleh@localhost;
- ▶ **RENAME USER** souleh@localhost **TO** souleh@192.168.143.172;
- ▶ **RENAME USER** souleh@192.168.143.172 **TO** beille@localhost;
- ▶ **DROP USER** beille@localhost;



Gestion base des données



Création d'une base des données

- ▶ Avoir le privilège CREATE au niveau global pour créer des tables.
- ▶ **CREATE {DATABASE | SCHEMA}** [IF NOT EXISTS] *nomBase* [[DEFAULT] CHARACTER SET *nomJeu*] [[DEFAULT] COLLATE *nomCollation*] ;
- ▶ IF NOT EXISTS évite une erreur dans le cas où la base de données existe déjà.
- ▶ *nomBase* désigne le nom de la base (64 caractères max). Les caractères « / », « \ », ou « . » sont exclus.
- ▶ CHARACTER SET indique le jeu de caractères associé aux données qui résideront dans les tables de la base.
- ▶ COLLATE définit la collation du jeu de caractères en question. La collation permet de définir la position des caractères dans le jeu. Par exemple, il sera possible de différencier « à » de « a » ou pas.

Exemple création de base

- ▶ **CREATE DATABASE** bdnew DEFAULT CHARACTER SET ascii COLLATE ascii_general_ci; (pour la langue anglaise)
- ▶ **CREATE DATABASE** bdnew2 DEFAULT CHARACTER SET gb2312; (pour les chinois).
- ▶ Le jeu de caractères par défaut est défini dans my.ini à l'aide de la variable default-character-set. donc possibilité de créer des bases de données associées à différents jeux de caractères au sein d'un même serveur.
- ▶ Possibilité de définir des jeux de caractères différents de celui de la base pour les tables, ou même pour chaque colonne :
- ▶ CREATE TABLE bdnew2.testChap3 (col CHAR(5) **CHARACTER SET** cp850 , col2 CHAR(4)) **CHARACTER SET** latin1;
- ▶ INSERT INTO bdnew2.testChap5 VALUES ('GTR','IUT');

Sélection d'une base des données

- ▶ L'instruction USE sélectionne une base de données qui devient active dans une session. **USE nomBase;**
- ▶ CREATE TABLE bdnew.test (col3 CHAR(5), col4 CHAR(4)) CHARACTER SET latin1;
- ▶ INSERT INTO bdnew.test VALUES ('ACTMP', 'IUT');
- ▶ USE bdnew2;
- ▶ SELECT col ,bdnew.test.col3 FROM testChap3, bdnew.test WHERE col2 = bdnew.test.col4;

Modification/suppression d'une base

- ▶ Avoir le privilège ALTER sur la base de données pour la modifier. Syntaxe :
- ▶ **ALTER {DATABASE** *nomBase* [[DEFAULT]
CHARACTER SET *nomJeu*] [[DEFAULT] COLLATE
nomCollation] ;
- ▶ **ALTER DATABASE** bdnew2 DEFAULT CHARACTER
SET cp850 ;
- ▶ Avoir le privilège DROP sur la base de données pour la supprimer. Syntaxe :
- ▶ **DROP {DATABASE | SCHEMA}** [IF EXISTS]
nomBase ;
- ▶ **DROP DATABASE** bdnew2 ;



Privilèges d'un utilisateur

Privilèges d'un user

- ▶ Un **privilège** est un droit d'exécuter une certaine instruction SQL (privilège système, exemple connexion), ou un droit relatif aux données des tables situées dans différentes bases (privilège objet). Il existe plusieurs niveaux de privilèges :
- ▶ **Global level** : s'appliquent à toutes les bases du serveur et sont stockés dans mysql.user (exemple: GRANT CREATE ON *.*).
- ▶ **Database level** : s'appliquent à tous les objets d'une base de données en particulier, sont stockés dans mysql.db et mysql.host (exemple: GRANT SELECT ON bdtest.*).

Autres niveaux de privilèges

- ▶ **Table level** : s'appliquent à la globalité d'une table d'une base de données en particulier, sont stockés dans la table mysql.tables_priv (exemple : GRANT INSERT ON bdtest.Avion).
- ▶ **Column level** : s'appliquent à une des colonnes d'une table en particulier, sont stockés dans la table mysql.columns_priv (exemple : GRANT UPDATE (nomComp) ON bdtest.Compagnie).
- ▶ **Routine level** : privilèges globaux ou au niveau d'une base (CREATE ROUTINE, ALTER ROUTINE, EXECUTE, et GRANT) s'appliquant aux procédures cataloguées, sont stockés dans la table mysql.procs_priv (mysql) (exemple: GRANT EXECUTE ON PROCEDURE bdtest.sp1)

Table mysql.user

- ▶ *Privilèges objet (LMD) sur toutes les bases de données:*

```
SELECT Host, User, Select_priv,  
Insert_priv, Update_priv, Delete_priv  
FROM mysql.user;
```

| Host | User | Select_priv | Insert_priv | Update_priv | Delete_priv |
|-----------|------|-------------|-------------|-------------|-------------|
| localhost | root | Y | Y | Y | Y |
| localhost | | Y | Y | Y | Y |
| % | | N | N | N | N |
| localhost | Paul | N | N | N | N |

Table mysql.user

- ▶ *Privilèges objet (LDD) sur toutes les bases de données:*

```
SELECT Host, User, Create_priv,  
Drop_priv, Index_priv, Alter_priv FROM  
mysql.user;
```

| Host | User | Create_priv | Drop_priv | Index_priv | Alter_priv |
|-----------|------|-------------|-----------|------------|------------|
| localhost | root | Y | Y | Y | Y |
| localhost | | Y | Y | Y | Y |
| % | | N | N | N | N |
| localhost | Paul | N | N | N | N |

Table mysql.user

- ▶ *Privilèges systèmes (LCD) sur toutes les bases de données:*

```
SELECT Host,User, Create_user_priv,  
Grant_priv,Show_db_priv FROM mysql.user;
```

| Host | User | Create_user_priv | Grant_priv | Show_db_priv |
|-----------|------|------------------|------------|--------------|
| localhost | root | Y | Y | Y |
| localhost | | N | Y | Y |
| % | | N | N | N |
| localhost | Paul | N | N | N |

Table mysql.user

- ▶ *Privilèges à propos des vues sur toutes les bases de données :*

```
SELECT Host,User, Create_view_priv,  
Show_view_priv FROM mysql.user;
```

| Host | User | Create_view_priv | Show_view_priv |
|-----------|------|------------------|----------------|
| localhost | root | Y | Y |
| localhost | | N | N |
| % | | N | N |
| localhost | Paul | N | N |

Table mysql.user

- ▶ *Privilèges à propos des procédures cataloguées sur toutes les bases de données :*

```
SELECT Host,User, Create_routine_priv,  
Alter_routine_priv, Execute_priv FROM  
mysql.user;
```

| Host | User | Create_routine_priv | Alter_routine_priv | Execute_priv |
|-----------|------|---------------------|--------------------|--------------|
| localhost | root | Y | Y | Y |
| localhost | | N | N | Y |
| % | | N | N | N |
| localhost | Paul | N | N | N |

Table mysql.user

- ▶ *Privilèges à propos des restrictions d'utilisateur :*

```
SELECT Host, User, max_questions "Requêtes", max_updates "Modifs" ,  
       max_connections "Connexions", max_user_connections "Cx simult."  
FROM mysql.user;
```

| Host | User | Requêtes | Modifs | Connexions | Cx simult. |
|-----------|------|----------|--------|------------|------------|
| localhost | root | 0 | 0 | 0 | 0 |
| localhost | | 0 | 0 | 0 | 0 |
| % | | 0 | 0 | 0 | 0 |
| localhost | Paul | 0 | 0 | 0 | 0 |

Attribution des privilèges (GRANT)

- ▶ L'instruction GRANT permet d'attribuer un (ou plusieurs) privilège(s) à propos d'un objet à un (ou plusieurs) bénéficiaire(s). L'utilisateur qui exécute cette commande doit avoir reçu lui-même le droit de transmettre ces privilèges (reçu avec la directive GRANT OPTION):

```
GRANT privilege [ (col1 [, col2...]) ] [,privilege2 ... ]
ON [{TABLE | FUNCTION | PROCEDURE}]
{nomTable | * | *.* | nomBase.*}
TO utilisateur [IDENTIFIED BY [PASSWORD] 'password']
[,utilisateur2 ...]
[ WITH [ GRANT OPTION ]
[ MAX_QUERIES_PER_HOUR nb ]
[ MAX_UPDATES_PER_HOUR nb2 ]
[ MAX_CONNECTIONS_PER_HOUR nb3 ]
[ MAX_USER_CONNECTIONS nb4 ] ];
```



Option de GRANT

- ▶ *privilège* : description du privilège (ex : SELECT, DELETE, etc.).
- ▶ *col* précise la ou les colonnes sur lesquelles se portent les privilèges SELECT, INSERT ou UPDATE (exemple : UPDATE (typeAvion) pour n'autoriser que la modification de la colonne typeAvion).
- ▶ GRANT OPTION : permet de donner le droit de retransmettre les privilèges reçus à une tierce personne.

Privilèges pour GRANT et REVOKE

| privilege | Commentaire |
|------------------|---|
| ALL [PRIVILEGES] | Tous les privilèges. |
| ALTER | Modification de base/table. |
| ALTER ROUTINE | Modification de procédure. |
| CREATE | Création de base/table. |
| CREATE ROUTINE | Création de procédure. |
| CREATE USER | Création d'utilisateur. |
| CREATE VIEW | Création de vue. |
| DELETE | Suppression de données de table. |
| DROP | Suppression de base/table. |
| EXECUTE | Exécution de procédure. |
| INDEX | Création/Suppression d'index. |
| INSERT | Insertion de données de table. |
| SELECT | Extraction de données de table. |
| SHOW DATABASES | Lister les bases. |
| SHOW VIEW | Lister les vues d'une base. |
| SUPER | Gestion des déclencheurs. |
| UPDATE | Modification de données de table. |
| USAGE | Synonyme de « sans privilège », USAGE est utilisé pour conserver les privilèges précédemment définis tout en les restreignant avec des options. |

Affectation des privilèges

```
GRANT CREATE, DROP ON bdpaul.*  
    TO 'Paul'@'localhost';
```

Privilège système *database level* :
Paul (en accès local) peut créer ou supprimer des tables dans la base bdpaul.

```
GRANT INSERT, SELECT, DELETE, UPDATE(ISBN)  
    ON bdpaul.Livre  
    TO 'Paul'@'localhost';
```

Privilège objet *table level* :
Paul peut insérer, extraire, supprimer et modifier la colonne ISBN de la table Livre contenue dans la base bdpaul.

```
GRANT ALTER ON bdpaul.Livre  
    TO 'Paul'@'localhost';
```

Privilège système *table level* :
Paul peut modifier la structure ou les contraintes de la table Livre contenue dans la base bdpaul.

```
GRANT SELECT(titre)  
    ON bdpaul.Livre  
    TO 'Jules'@'localhost'  
    WITH GRANT OPTION;
```

Privilège objet *column level* :
Jules peut extraire seulement la colonne titre de la table Livre contenue dans la base bdpaul. Il pourra par la suite retransmettre éventuellement ce droit.

```
GRANT CREATE ON *.*  
    TO 'Jules'@'localhost';
```

Privilège système *global level* :
Jules peut créer des bases de données.

```
GRANT USAGE ON bdpaul.*  
    TO 'Jules'@'localhost'  
    WITH  
        MAX_QUERIES_PER_HOUR 50  
        MAX_UPDATES_PER_HOUR 20  
        MAX_CONNECTIONS_PER_HOUR 6  
        MAX_USER_CONNECTIONS 3;
```

Privilège système *database level* :
Jules ne peut lancer, chaque heure, que 50 SELECT, 20 UPDATE, se connecter 6 fois (dont 3 connexions simultanées) sur la base de données bdpaul.

Voir les privilèges (SHOW GRANTS FOR ,

- ▶ La commande SHOW GRANTS FOR liste les différentes instructions GRANT équivalentes à toutes les prérogatives d'un utilisateur donné.
- ▶ **SHOW GRANTS FOR** *utilisateur*;

```
SHOW GRANTS FOR 'Jules'@'localhost';
+-----+
| Grants for Jules@localhost
+-----+
| GRANT CREATE ON *.* TO 'Jules'@'localhost' IDENTIFIED BY PASSWORD
  '*6AE163FB9EE8BB011EB2E87316AA5BE563A6CDB7' WITH MAX_QUERIES_PER_HOUR 50
    MAX_UPDATES_PER_HOUR 20 MAX_CONNECTIONS_PER_HOUR 6 MAX_USER_CONNECTIONS 3
| GRANT SELECT (titre) ON `bdpaul`.`Livre` TO 'Jules'@'localhost'
  WITH GRANT OPTION
+-----+
```

Voir les privilèges (SHOW GRANTS FOR ,

```
SHOW GRANTS FOR 'Paul'@'localhost';
+-----+
| Grants for Paul@localhost
+-----+
| GRANT USAGE ON *.* TO 'Paul'@'localhost' IDENTIFIED BY PASSWORD
  '**6AE163FB9EE8BB011EB2E87316AA5BE563A6CDB7'
| GRANT CREATE, DROP ON `bdpaul`.* TO 'Paul'@'localhost'
| GRANT SELECT, INSERT, UPDATE (ISBN), DELETE, ALTER ON `bdpaul`.`Livre`
  TO 'Paul'@'localhost'
+-----+
```

```
SHOW GRANTS FOR 'root'@'localhost';
+-----+
| Grants for root@localhost
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' IDENTIFIED BY PASSWORD
  '*387E25FE2CF7ED941E43A76AD9402825401698FC' WITH GRANT OPTION
+-----+
```

Table mysql.db

- La table mysql.db décrit les prérogatives au niveau *database*. Ainsi la colonne Db indique la base de données.

| SELECT Host, User, Db, Create_priv, Drop_priv, Alter_priv FROM mysql.db | | | | | |
|---|------|---------|-------------|-----------|------------|
| Host | User | Db | Create_priv | Drop_priv | Alter_priv |
| % | | test_% | Y | Y | Y |
| % | | test | Y | Y | Y |
| localhost | Paul | bdpaul | Y | Y | N |

Table mysql.tables_priv

- La table mysql.tables_priv décrit les prérogatives objet au niveau *table*.

```
SELECT CONCAT(User,'@',Host) "Compte",
        CONCAT(Db,'.',Table_name) "Objet", Grantor, Table_priv
   FROM mysql.tables_priv;
```

| Compte | Objet | Grantor | Table_priv |
|-----------------|--------------|----------------|----------------------------|
| Jules@localhost | bdpaul.Livre | root@localhost | Grant |
| Paul@localhost | bdpaul.Livre | root@localhost | Select,Insert,Delete,Alter |

Table mysql.columns_priv

- ▶ La table mysql.columns_priv décrit les prérogatives objet au niveau *column*.

```
SELECT CONCAT(User, '@', Host) "Compte", CONCAT(Db, '.', Table_name) "Objet",
       Column_name, Column_priv FROM mysql.columns_priv;
```

| Compte | Objet | Column_name | Column_priv |
|-----------------|--------------|-------------|-------------|
| Jules@localhost | bdpaul.Livre | titre | Select |
| Paul@localhost | bdpaul.Livre | ISBN | Update |

Révocation de privilèges (REVOKE)

- ▶ La révocation d'un ou de plusieurs privilèges est réalisée par l'instruction REVOKE. Pour pouvoir révoquer un privilège, vous devez détenir au préalable ce même privilège avec l'option WITH GRANT OPTION.
- ▶ **REVOKE** *privilege* [(col1 [, col2...])] ON [{TABLE | FUNCTION | PROCEDURE}] {nomTable | * | *.* | nomBase.*} FROM *utilisateur* ;
- ▶ **REVOKE ALL PRIVILEGES**, GRANT OPTION FROM *utilisateur* [, utilisateur2 ...] ; (avoir le privilège CREATE USER au niveau global ou le privilège UPDATE au niveau *database* sur la base mysql).

Révocation de privilèges (REVOKE)

Instruction faite par root

```
REVOKE CREATE  
  ON bdpaul.*  
  FROM 'Paul'@'localhost';
```

```
REVOKE ALTER, INSERT, UPDATE (ISBN)  
  ON bdpaul.Livre  
  FROM 'Paul'@'localhost';
```

```
GRANT USAGE ON bdpaul.*  
  TO 'Jules'@'localhost'  
  WITH  
    MAX_QUERIES_PER_HOUR 0  
    MAX_UPDATES_PER_HOUR 0;
```

Explication

Privilège système *database level* :

Paul (en accès local) ne peut plus créer de tables dans la base bdpaul.

Privilège objet *table level* :

Paul ne peut plus modifier la structure (ou les contraintes), insérer et modifier la colonne ISBN de la table Livre contenue dans la base bdpaul.

Privilège système *database level* :

Jules n'est plus limité en requêtes SELECT et UPDATE sur la base de données bdpaul. Ici c'est un GRANT qu'il faut faire, car il s'agit plus d'une restriction de connexion que d'une instruction SQL.

Accès distant vers MySQL

- ▶ Demander un accès au serveur de MySQL à partir d'une machine autre que celle contenant le serveur MySQL.
- ▶ MySQL doit être installé sur la machine cliente.
- ▶ Les deux machines (client/serveur) doivent être reliée par TCP/IP.
- ▶ Attribution des droits d'accès distant à l'utilisateur sur le serveur (192.168.4.8), exemple:
- ▶ **CREATE USER** userd@192.168.4.3 IDENTIFIED BY 'sitedist';
- ▶ **GRANT SELECT ON** bdtest.tabtest **TO** 'userd'@'192.168.4.3';
- ▶ **mysql -h 192.168.4.8 -u userd -p** (commande de connexion depuis le client (192.168.4.3) vers le serveur).

Table mysql.host

- ▶ La table mysql.host est utilisée conjointement avec mysql.db et concerne les accès distants (plusieurs machines).
- ▶ Elle n'est employée que pour les droits au niveau *database*, indépendamment des utilisateurs.
- ▶ La structure est la même que celle de mysql.db, à l'exception de la colonne User qui n'est pas présente. Le couple de colonnes (Host, Db) est unique.

| Caractère | Signification pour mysql.db | | Signification pour mysql.host | |
|-------------------|-------------------------------|------------|-------------------------------|------------|
| | colonne Host | colonne Db | colonne Host | colonne Db |
| % | toute machine | toute base | toute machine | toute base |
| ' ' (chaîne vide) | consultez la table mysql.host | toute base | toute machine | toute base |

Table mysql.host

- ▶ Pour les (INSERT, UPDATE, etc.), MySQL interroge la table user. Si accès non décrit, recherche dans les tables db et host.
- ▶ Si la colonne Host de la table db est renseignée en fonction de l'accès, l'utilisateur reçoit ses privilèges.
- ▶ Si la colonne Host de la table db n'est pas renseignée (''), cela signifie que la table host énumère les machines qui sont autorisées à accéder à une base de données en particulier.
- ▶ La table mysql.host n'est mise à jour ni par GRANT, ni par REVOKE. Il faudra directement insérer (par INSERT), modifier (par UPDATE) ou supprimer (par DELETE) les lignes de cette table.

| Host | Db | ... |
|------------------|----|-----------------------------|
| camparols.gtr.fr | % | (tous les privilèges à 'N') |
| %.gtr.fr | % | (tous les privilèges à 'Y') |

Les vues en MySQL

- ▶ Depuis la version 5 de MySQL, la confidentialité des données est boosté par l'utilisation de vues (*views*) qui agissent comme des fenêtres sur la base de données.
- ▶ Les vues correspondent à ce qu'on appelle « le niveau externe » qui reflète la partie visible de la base de données pour chaque utilisateur.
- ▶ Elles masquent la structure des tables interrogées, ce qui renforce la confidentialité.
- ▶ Les utilisateurs ne devraient accéder aux données qu'en questionnant ces vues.
- ▶ Dans certains cas, la définition d'une vue temporaire est nécessaire pour écrire une requête qu'il ne serait pas possible de construire à partir des tables seules.

Création d'une vue en MySQL

- ▶ Avoir le privilège CREATE VIEW et les privilèges en SELECT des tables présentes dans la requête de définition de la vue.
- ▶ **CREATE** [OR REPLACE] [ALGORITHM = { UNDEFINED | MERGE | TEMPTABLE}] **VIEW** [*nomBase.*]*nomVue* [*(listecolonne*s)] AS *requêteSELECT* [WITH [CASCDED | LOCAL] CHECK OPTION];
- ▶ OR REPLACE remplace la vue par la nouvelle définition, même si elle existait déjà (avoir le privilège DELETE sur la base).
- ▶ ALGORITHM=MERGE : la définition de la vue et sa requête sont fusionnées en interne (convient aux vues modifiables).
- ▶ ALGORITHM=TEMPTABLE : vue en lecture seule, utilisation d'une table temporaire (TEMPORARY) pour les résultats.

Création d'une vue en MySQL

- ▶ **ALGORITHM=UNDEFINED** (**Aucune option**) : MySQL choisit l'option à adopter (souvent MERGE) .
- ▶ **WITH CHECK OPTION** : garantit que toute mise à jour de la vue par INSERT ou UPDATE s'effectuera conformément au prédictat (condition) contenu dans la requête de définition.
- ▶ **LOCAL et CASCDED** (par défaut): déterminent la portée de la vérification du prédictat (condition) quand une vue est définie à partir d'une autre vue.
 - LOCAL: vérification restreint à la vue elle-même.
 - CASCDED : vérifications étendues aux autres vues source.
- ▶ **La requête de définition** ne peut interroger une table *temporaire*, ni contenir de paramètres ou de variables de session.

Catégories des vues

Il existe deux catégories des vues, en fonction de la nature de la requête de définition:

- ▶ **Vue simple** : vue **monotable**, sa requête de définition ne contient ni des *fonctions*, ni de *regroupement*, mais sa mise à jour est possible.
- ▶ **Vue complexe** : vue **multitable**s, sa requête de définition contient des *fonctions*, un *regroupement*, et sa mise à jour n'est pas toujours possible.

Les vues monatables

- ▶ Une vue **monutable** est définie par une requête SELECT ne comportant qu'une seule table dans sa clause FROM.
- ▶ **CREATE VIEW** PilotesAF **AS** SELECT * FROM Pilote WHERE compa = 'AF';
- ▶ **CREATE VIEW** Etat_civil **AS** SELECT nom, nbHVol, adresse, compa FROM Pilote;
- ▶ Une fois créée, une vue s'interroge comme une table par tout utilisateur détenant le privilège en lecture (GRANT SELECT ON nomVue TO...). Exemple:
- ▶ SELECT SUM(nbHVol) FROM **PilotesAF**;
- ▶ SELECT COUNT(*) FROM **Etat_civil**;

Vues avec utilisation d'alias

- ▶ Les alias, s'ils sont utilisés, désignent le nom de chaque colonne de la vue.
- ▶ Leur utilisation sert à masquer les noms des colonnes de l'objet source.
- ▶ **CREATE OR REPLACE VIEW** PilotesPasAF
(codepil, nomPil, heuresPil, adressePil,
societe)AS SELECT * FROM Pilote WHERE
NOT(compa = 'AF') ;
- ▶ **CREATE OR REPLACE VIEW** PilotesPasAF AS
SELECT brevet codepil, nom nomPil, nbHVol
heuresPil, adresse adressePil, compa societe
FROM Pilote WHERE NOT(compa = 'AF') ;

Vue source d'une autre vue

- ▶ L'objet source d'une vue est en général une table, mais peut aussi être une vue. Il est possible d'utiliser des *alias* pour renommer les colonnes de la nouvelle vue.
- ▶ La vue suivante est définie à partir de la vue PilotesPasAF précédemment créée.
- ▶ **CREATE OR REPLACE VIEW**

*EtatCivilPilotesPasAF AS SELECT nomPil,
heuresPil, adressePil FROM PilotesPasAF;*

Vues en lecture seule

- ▶ L'option ALGORITHM=TEMPTABLE déclare la vue (lecture seule) non modifiable par INSERT, UPDATE, ou DELETE.
- ▶ **CREATE OR REPLACE VIEW ALGORITHM=TEMPTABLE**
PilotesPasAFLS AS SELECT * FROM Pilote WHERE
NOT (compa='AF');
- ▶ INSERT INTO **PilotesPasAFLS** VALUES ('PL-
8', 'FARAH', 5, 'ARTA', 'ASO');
- ▶ **ERROR 1288 (HY000)** : The target table
PilotesPasAFLS of the INSERT is not updatable.
- ▶ UPDATE **PilotesPasAFLS** SET nbHvol=nbHvol+2;
- ▶ **ERROR 1288 (HY000)** : The target table
PilotesPasAFLS of the UPDATE is not updatable.

Vues modifiables

- ▶ L'option ALGORITHM=MERGE convient à la définition d'une vue modifiable (*updatable view*) par INSERT, UPDATE, ou DELETE.
- ▶ Ces modifications sont intrinsèque à cette vue (concerne que ses données).
- ▶ Pour mettre à jour une vue, il doit exister une correspondance réciproque entre les lignes de la vue et celles de l'objet source.
- ▶ Sa requête de définition doit respecter les critères suivants :
 - Pas de directive DISTINCT, de fonction (AVG, COUNT, MAX, MIN, SUM, ou VARIANCE), d'expression dans le SELECT ;
 - Pas de GROUP BY, ORDER BY ou HAVING.

Vues modifiables (exemples)

- ▶ INSERT INTO Etat_civil VALUES ('Radwan',10,'Balbala','ASO') ;
- ▶ INSERT INTO Etat_civil VALUES ('Liban',20,'Arta','ASO') ; (**impossible**)
- ▶ DELETE FROM Etat_civil WHERE compa='ASO' ;
- ▶ UPDATE Etat_civil SET nbHVol=nbHVol*2 WHERE nom='Radwan' ;
- ▶ INSERT INTO PilotesAF VALUES ('PL-8','Farah',5,'Arta','AF') ;

Directive CHECK OPTION

- ▶ La directive WITH CHECK OPTION empêche un ajout ou une modification non conformes à la définition de la vue.
- ▶ **CREATE OR REPLACE VIEW** PilotesAF AS SELECT * FROM pilote WHERE compa = 'AF' **WITH CHECK OPTION**;
- ▶ INSERT INTO PilotesAF VALUES ('PL-11', 'Teste', 900, 'Randa', '**AF**');
- ▶ INSERT INTO PilotesAF VALUES ('PL-9', 'Caboche', 600, 'Rennes', '**ASO**');
- ▶ UPDATE PilotesAF SET compa='ASO';
- ▶ **ERROR 1369 (HY000)**: CHECK OPTION failed 'bdtest.PilotesAF'.

Vues complexes

- ▶ Une vue complexe contient, dans sa définition, plusieurs tables (jointures) ou une fonction appliquée à des regroupements ou à des expressions. Sa mise à jour n'est pas toujours possible.
- ▶ Pour qu'une vue complexe soit modifiable, les restrictions sont les suivantes :
 - La requête de définition ne doit pas contenir de sous-interrogation (jointure procédurale).
 - Il n'est pas possible d'utiliser d'opérateur ensembliste (sauf UNION [ALL]).

Vues complexes (exemples)

- ▶ **CREATE VIEW Pilotes_multi_AF** AS SELECT p.brevet, p.nom, p.nbHVol, c.ville, c.nomComp FROM Pilote p, Compagnie c WHERE p.compa=c.comp AND p.compa='AF';
- ▶ **CREATE VIEW Moyenne_Heures_Pil** AS SELECT compa, AVG(nbHVol) moyenne FROM Pilote GROUP BY compa;
- ▶ **INSERT INTO Moyenne_Heures_Pil** VALUES ('TAT',50);
- ▶ INSERT INTO Pilotes_multi_AF VALUES ('PL-4', 'Test', 400, 'Arta', 'AD'); **ERROR 1394** Can not insert into join view 'bdtest.Pilotes_multi_AF' without fields list.

Mis à jour d'une vue multitable

- ▶ **UPDATE Pilotes_multi_AF**
SET nbHVol=nbHVol*2;
- ▶ **UPDATE Pilotes_multi_AF**
SET ville='Orly';
- ▶ DELETE FROM
Pilotes_multi_AF;
(impossible)
- ▶ *ERROR 1395 (HY000) : Can
not delete from join view
'bdtest.Pilotes_multi_AF'*
.

```
SELECT brevet,nom,nbHVol
      FROM Pilotes_multi_AF;
+-----+-----+-----+
| brevet | nom          | nbHVol |
+-----+-----+-----+
| PL-1   | Louise Ente  | 900.00  |
| PL-2   | Paul Ente    | 1800.00 |
+-----+-----+-----+
```



```
SELECT brevet,nom,ville
      FROM Pilotes_multi_AF;
+-----+-----+-----+
| brevet | nom          | ville   |
+-----+-----+-----+
| PL-1   | Louise Ente  | Orly   |
| PL-2   | Paul Ente    | Orly   |
+-----+-----+-----+
```

Tables protégées par clé

- Une table est dite protégée par sa clé (*key preserved*) si sa clé primaire est préservée dans la clause de jointure et se retrouve en tant que colonne de la vue multitable (elle peut jouer le rôle de clé primaire de la vue).

CREATE VIEW

```
Vue_Multi_Comp_Pil AS SELECT c.comp, p.brevet, p.nom, p.nbHVol
FROM Pilote p,
     Compagnie c
WHERE p.compa=c.comp;
```

| comp | nomComp | brevet | nom | nbHVol |
|------|--------------|--------|-------------|---------|
| AF | Air France | PL-1 | Louise Ente | 450.00 |
| AF | Air France | PL-2 | Paul Ente | 900.00 |
| SING | Singapore AL | PL-3 | Paul Soutou | 1000.00 |

Vue multitable modifiable (Critères)

- ▶ Une vue multitable modifiable (*updatable join view ou modifiable join view*) est une vue qui n'est pas définie avec l'option ALGORITHM=TEMPTABLE et dont sa requête de définition contient plusieurs tables dans la clause FROM.
- ▶ Aucune suppression n'est possible.
- ▶ Les insertions sont permises seulement en isolant toutes les colonnes d'une seule table source.

Vue multitable modifiable

- ▶ DELETE FROM **Vue_Multi_Comp_Pil** WHERE comp='AF'; (impossible, erreur 1395!)
- ▶ UPDATE **Vue_Multi_Comp_Pil** SET nbHVol = nbHVol * 3; Query OK, 3 rows affected.
Rows matched: 3 Changed: 3 Warnings: 0
- ▶ UPDATE **Vue_Multi_Comp_Pil** SET nomComp = 'Dupond'; Query OK, 2 rows affected.
Rows matched: 3 Changed: 2 Warnings: 0
- ▶ INSERT INTO **Vue_Multi_Comp_Pil** (brevet,nom,nbHVol) VALUES ('PL-5', 'Jama', 250);
- ▶ INSERT INTO **Vue_Multi_Comp_Pil** (comp,nomComp) VALUES ('TAT', 'Test');

Transmission des droits sur une vue

- ▶ si un utilisateur désire transmettre des droits sur une partie d'une de ses tables, il utilisera une vue. Seules les données appartenant à la vue seront accessibles aux bénéficiaires.
- ▶ Les privilèges objet qu'il est possible d'attribuer sur une vue sont les mêmes que ceux applicables sur les tables (SELECT, INSERT, UPDATE sur une ou plusieurs colonnes, DELETE).
- ▶ GRANT **SELECT** ON **Vue_Multi_Comp_Pil** TO 'utest'@'localhost';
- ▶ GRANT **INSERT** ON **Vue_Multi_Comp_Pil** TO 'Jama'@'localhost';

Modifier une vue (ALTER VIEW)

- ▶ Avoir les privilèges CREATE VIEW et DELETE au niveau d'une vue pour pouvoir la modifier.
- ▶ **ALTER** [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] **VIEW** [*nomBase.*]*nomVue* [(*listecollettes*)] AS *requêteSELECT* [WITH [CASCDED | LOCAL] CHECK OPTION];
- ▶ Les transformations peuvent concerner toutes les parties d'une vue existante : la politique de création (ALGORITHM), la liste des colonnes, la requête, etc.
- ▶ **ALTER VIEW** ALGORITHM=TEMPTABLE
Vue_Multi_Comp_Pil AS SELECT c.nomComp,
p.brevet, p.nom FROM Pilote p, Compagnie c
WHERE p.compa=c.comp;

Visualisation (SHOW CREATE VIEW)

- ▶ Pour pouvoir visualiser la requête de définition d'une vue, l'instruction que MySQL propose est la suivante :
- ▶ **SHOW CREATE VIEW** [*nomBase.*]*nomVue*;
- ▶ SHOW CREATE VIEW Vue_Multi_Comp_Pil;

Suppression vue (DROP VIEW)

- ▶ Avoir le privilège DROP sur une vue pour pouvoir la supprimer.
- ▶ **DROP VIEW [IF EXISTS]** [*nomBase.*]*nomVue* [*, nomBase2.*]*nomVue2* ;
- ▶ **DROP VIEW IF EXISTS** PilotesPasAFLS;



FIN DU CHAPITRE



Base de Données Avancées

COURS 4

Les Vues, les procédures et les fonctions avec
MySQL



Plan de la présentation

- ▶ Les vues en MySQL
 - Définition
 - Création des vues
 - Vues monotables
 - Vues complexes
 - Modification, visualisation et suppression des vues
- ▶ Les procédures et fonctions stockées

Les vues en MySQL : définition

- ▶ Depuis la version 5 de MySQL, la confidentialité des données est boosté par l'utilisation de vues (*views*) qui agissent comme des fenêtres sur la base de données.
- ▶ Les vues correspondent à ce qu'on appelle « le niveau externe » qui reflète la partie visible de la base de données pour chaque utilisateur.
- ▶ Elles masquent la structure des tables interrogées, ce qui renforce la confidentialité.
- ▶ Les utilisateurs ne devraient accéder aux données qu'en questionnant ces vues.
- ▶ Dans certains cas, la définition d'une vue temporaire est nécessaire pour écrire une requête qu'il ne serait pas possible de construire à partir des tables seules.



Création d'une vue en MySQL

- ▶ Avoir le privilège CREATE VIEW et les privilèges en SELECT des tables présentes dans la requête de définition de la vue.
- ▶ **CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] VIEW** [*nomBase.*]*nomVue* [*(listecolonne*s)] AS *requêteSELECT* [WITH [CASCDED | LOCAL] CHECK OPTION];
- ▶ OR REPLACE remplace la vue par la nouvelle définition, même si elle existait déjà (avoir le privilège DELETE sur la base).
- ▶ ALGORITHM=MERGE : la définition de la vue et sa requête sont fusionnées en interne (convient aux vues modifiables).
- ▶ ALGORITHM=TEMPTABLE : vue en lecture seule, utilisation d'une table temporaire (TEMPORARY) pour les résultats.

Création d'une vue en MySQL

- ▶ **ALGORITHM=UNDEFINED** (**Aucune option**) : MySQL choisit l'option à adopter (souvent MERGE) .
- ▶ **WITH CHECK OPTION** : garantit que toute mise à jour de la vue par INSERT ou UPDATE s'effectuera conformément au prédictat (condition) contenu dans la requête de définition.
- ▶ **LOCAL et CASCDED** (par défaut): déterminent la portée de la vérification du prédictat (condition) quand une vue est définie à partir d'une autre vue.
 - LOCAL: vérification restreint à la vue elle-même.
 - CASCDED : vérifications étendues aux autres vues source.
- ▶ **La requête de définition** ne peut interroger une table *temporaire*, ni contenir de paramètres ou de variables de session.

Catégories des vues

Il existe deux catégories des vues, en fonction de la nature de la requête de définition:

- ▶ **Vue simple** : vue **monotable**, sa requête de définition ne contient ni des *fonctions*, ni de *regroupement*, mais sa mise à jour est possible.
- ▶ **Vue complexe** : vue **multitable**s, sa requête de définition contient des *fonctions*, un *regroupement*, et sa mise à jour n'est pas toujours possible.

Les vues monatables

- ▶ Une vue **monutable** est définie par une requête SELECT ne comportant qu'une seule table dans sa clause FROM.
- ▶ **CREATE VIEW** PilotesAF **AS** SELECT * FROM Pilote WHERE compa = 'AF';
- ▶ **CREATE VIEW** Etat_civil **AS** SELECT nom, nbHVol, adresse, compa FROM Pilote;
- ▶ Une fois créée, une vue s'interroge comme une table par tout utilisateur détenant le privilège en lecture (GRANT SELECT ON nomVue TO...). Exemple:
- ▶ SELECT SUM(nbHVol) FROM **PilotesAF**;
- ▶ SELECT COUNT(*) FROM **Etat_civil**;

Vues avec utilisation d'alias

- ▶ Les alias, s'ils sont utilisés, désignent le nom de chaque colonne de la vue.
- ▶ Leur utilisation sert à masquer les noms des colonnes de l'objet source.
- ▶ **CREATE OR REPLACE VIEW** PilotesPasAF
(codepil, nomPil, heuresPil, adressePil,
societe)AS SELECT * FROM Pilote WHERE
NOT(compa = 'AF') ;
- ▶ **CREATE OR REPLACE VIEW** PilotesPasAF AS
SELECT brevet *codepil*, *nom nomPil*, *nbHVol*
heuresPil, *adresse adressePil*, *compa societe*
FROM Pilote WHERE NOT(compa = 'AF') ;

Vue source d'une autre vue

- ▶ L'objet source d'une vue est en général une table, mais peut aussi être une vue. Il est possible d'utiliser des *alias* pour renommer les colonnes de la nouvelle vue.
- ▶ La vue suivante est définie à partir de la vue PilotesPasAF précédemment créée.
- ▶ **CREATE OR REPLACE VIEW**

*EtatCivilPilotesPasAF AS SELECT nomPil,
heuresPil, adressePil FROM PilotesPasAF;*

Vues en lecture seule

- ▶ L'option ALGORITHM=TEMPTABLE déclare la vue (lecture seule) non modifiable par INSERT, UPDATE, ou DELETE.
- ▶ **CREATE OR REPLACE VIEW ALGORITHM=TEMPTABLE**
PilotesPasAFLS AS SELECT * FROM Pilote WHERE
NOT (compa='AF');
- ▶ INSERT INTO **PilotesPasAFLS** VALUES ('PL-
8', 'FARAH', 5, 'ARTA', 'ASO');
- ▶ **ERROR 1288 (HY000)** : The target table
PilotesPasAFLS of the INSERT is not updatable.
- ▶ UPDATE **PilotesPasAFLS** SET nbHvol=nbHvol+2;
- ▶ **ERROR 1288 (HY000)** : The target table
PilotesPasAFLS of the UPDATE is not updatable.

Vues modifiables

- ▶ L'option ALGORITHM=MERGE convient à la définition d'une vue modifiable (*updatable view*) par INSERT, UPDATE, ou DELETE.
- ▶ Ces modifications sont intrinsèque à cette vue (concerne que ses données).
- ▶ Pour mettre à jour une vue, il doit exister une correspondance réciproque entre les lignes de la vue et celles de l'objet source.
- ▶ Sa requête de définition doit respecter les critères suivants :
 - Pas de directive DISTINCT, de fonction (AVG, COUNT, MAX, MIN, SUM, ou VARIANCE), d'expression dans le SELECT ;
 - Pas de GROUP BY, ORDER BY ou HAVING.

Vues modifiables (exemples)

- ▶ INSERT INTO Etat_civil VALUES ('Radwan',10,'Balbala','ASO') ;
- ▶ INSERT INTO Etat_civil VALUES ('Liban',20,'Arta','ASO') ; **(impossible)**
- ▶ DELETE FROM Etat_civil WHERE compa='ASO' ;
- ▶ UPDATE Etat_civil SET nbHVol=nbHVol*2 WHERE nom='Radwan' ;
- ▶ INSERT INTO PilotesAF VALUES ('PL-8','Farah',5,'Arta','AF') ;

Directive CHECK OPTION

- ▶ La directive WITH CHECK OPTION empêche un ajout ou une modification non conformes à la définition de la vue.
- ▶ **CREATE OR REPLACE VIEW** PilotesAF AS SELECT * FROM pilote WHERE compa = 'AF' **WITH CHECK OPTION**;
- ▶ INSERT INTO PilotesAF VALUES ('PL-11', 'Teste', 900, 'Randa', '**AF**');
- ▶ INSERT INTO PilotesAF VALUES ('PL-9', 'Caboche', 600, 'Rennes', '**ASO**');
- ▶ UPDATE PilotesAF SET compa='ASO';
- ▶ **ERROR 1369 (HY000)**: CHECK OPTION failed 'bdtest.PilotesAF'.

Vues complexes

- ▶ Une vue complexe contient, dans sa définition, plusieurs tables (jointures) ou une fonction appliquée à des regroupements ou à des expressions. Sa mise à jour n'est pas toujours possible.
- ▶ Pour qu'une vue complexe soit modifiable, les restrictions sont les suivantes :
 - La requête de définition ne doit pas contenir de sous-interrogation (jointure procédurale).
 - Il n'est pas possible d'utiliser d'opérateur ensembliste (sauf UNION [ALL]).

Vues complexes (exemples)

- ▶ **CREATE VIEW Pilotes_multi_AF** AS SELECT p.brevet, p.nom, p.nbHVol, c.ville, c.nomComp FROM Pilote p, Compagnie c WHERE p.compa=c.comp AND p.compa='AF';
- ▶ **CREATE VIEW Moyenne_Heures_Pil** AS SELECT compa, AVG(nbHVol) moyenne FROM Pilote GROUP BY compa;
- ▶ **INSERT INTO Moyenne_Heures_Pil** VALUES ('TAT',50);
- ▶ INSERT INTO Pilotes_multi_AF VALUES ('PL-4', 'Test', 400, 'Arta', 'AD'); **ERROR 1394** Can not insert into join view 'bdtest.Pilotes_multi_AF' without fields list.

Mis à jour d'une vue multitable

- ▶ **UPDATE Pilotes_multi_AF**
SET nbHVol=nbHVol*2;
- ▶ **UPDATE Pilotes_multi_AF**
SET ville='Orly';
- ▶ DELETE FROM
Pilotes_multi_AF;
(impossible)
- ▶ *ERROR 1395 (HY000) : Can not delete from join view 'bdtest.Pilotes_multi_AF'.*

```
SELECT brevet,nom,nbHVol
      FROM Pilotes_multi_AF;
+-----+-----+-----+
| brevet | nom          | nbHVol |
+-----+-----+-----+
| PL-1   | Louise Ente  | 900.00  |
| PL-2   | Paul Ente    | 1800.00 |
+-----+-----+-----+
```



```
SELECT brevet,nom,ville
      FROM Pilotes_multi_AF;
+-----+-----+-----+
| brevet | nom          | ville   |
+-----+-----+-----+
| PL-1   | Louise Ente  | Orly   |
| PL-2   | Paul Ente    | Orly   |
+-----+-----+-----+
```

Tables protégées par clé

- Une table est dite protégée par sa clé (*key preserved*) si sa clé primaire est préservée dans la clause de jointure et se retrouve en tant que colonne de la vue multitable (elle peut jouer le rôle de clé primaire de la vue).

CREATE VIEW

Vue_Multi_Comp_Pil
AS SELECT c.comp,

c.nomComp,

p.brevet , p.nom ,

p.nbHVol

FROM Pilote p ,

Compagnie c

WHERE p.compa=c.comp ;

| comp | nomComp | brevet | nom | nbHVol |
|------|--------------|--------|-------------|---------|
| AF | Air France | PL-1 | Louise Ente | 450.00 |
| AF | Air France | PL-2 | Paul Ente | 900.00 |
| SING | Singapore AL | PL-3 | Paul Soutou | 1000.00 |

Vue multitable modifiable (Critères)

- ▶ Une vue multitable modifiable (*updatable join view ou modifiable join view*) est une vue qui n'est pas définie avec l'option ALGORITHM=TEMPTABLE et dont sa requête de définition contient plusieurs tables dans la clause FROM.
- ▶ Aucune suppression n'est possible.
- ▶ Les insertions sont permises seulement en isolant toutes les colonnes d'une seule table source.

Vue multitable modifiable

- ▶ DELETE FROM **Vue_Multi_Comp_Pil** WHERE comp='AF'; (impossible, erreur 1395!)
- ▶ UPDATE **Vue_Multi_Comp_Pil** SET nbHVol = nbHVol * 3; Query OK, 3 rows affected.
Rows matched: 3 Changed: 3 Warnings: 0
- ▶ UPDATE **Vue_Multi_Comp_Pil** SET nomComp = 'Dupond'; Query OK, 2 rows affected.
Rows matched: 3 Changed: 2 Warnings: 0
- ▶ INSERT INTO **Vue_Multi_Comp_Pil** (brevet,nom,nbHVol) VALUES ('PL-5', 'Jama', 250);
- ▶ INSERT INTO **Vue_Multi_Comp_Pil** (comp,nomComp) VALUES ('TAT', 'Test');

Transmission des droits sur une vue

- ▶ si un utilisateur désire transmettre des droits sur une partie d'une de ses tables, il utilisera une vue. Seules les données appartenant à la vue seront accessibles aux bénéficiaires.
- ▶ Les privilèges objet qu'il est possible d'attribuer sur une vue sont les mêmes que ceux applicables sur les tables (SELECT, INSERT, UPDATE sur une ou plusieurs colonnes, DELETE).
- ▶ GRANT **SELECT** ON **Vue_Multi_Comp_Pil** TO 'utest'@'localhost';
- ▶ GRANT **INSERT** ON **Vue_Multi_Comp_Pil** TO 'Jama'@'localhost';

Modifier une vue (ALTER VIEW)

- ▶ Avoir les privilèges CREATE VIEW et DELETE au niveau d'une vue pour pouvoir la modifier.
- ▶ **ALTER** [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] **VIEW** [*nomBase.*]*nomVue* [(*listecolonne*s)] AS *requêteSELECT* [WITH [CASCADED | LOCAL] CHECK OPTION];
- ▶ Les transformations peuvent concerner toutes les parties d'une vue existante : la politique de création (ALGORITHM), la liste des colonnes, la requête, etc.
- ▶ **ALTER** ALGORITHM=TEMPTABLE **VIEW**
Vue_Multi_Comp_Pil AS SELECT c.nomComp,
p.brevet, p.nom FROM Pilote p, Compagnie c
WHERE p.compa=c.comp;

Visualisation (SHOW CREATE VIEW)

- ▶ Pour pouvoir visualiser la requête de définition d'une vue, l'instruction que MySQL propose est la suivante :
- ▶ **SHOW CREATE VIEW** [*nomBase.*]*nomVue*;
- ▶ SHOW CREATE VIEW Vue_Multi_Comp_Pil;

Suppression vue (DROP VIEW)

- ▶ Avoir le privilège DROP sur une vue pour pouvoir la supprimer.
- ▶ **DROP VIEW [IF EXISTS]** [*nomBase.*]*nomVue* [*, nomBase2.*]*nomVue2* ;
- ▶ **DROP VIEW IF EXISTS** PilotesPasAFLS;



Procédures stockées en MySQL



Procédures stockées en MySQL

- ▶ Les procédures stockées sont disponibles depuis la version 5 de MySQL. Elles permettent d'automatiser des actions qui peuvent être très complexes.
- ▶ Une procédure stockée est en fait une série d'**instructions SQL** désignée par un **nom**. Lorsqu'elle est créée, une procédure stockée est enregistrée dans la base de données utilisée, au même titre qu'une table. A partir de ce moment là, il est possible d'appeler celle-ci par son nom. Les instructions de la procédure sont alors exécutées.

Porté d'un objet/sensitivité

- ▶ La portée d'un objet (variable, curseur ou exception) est la zone du programme qui peut y accéder. Un objet déclaré dans un bloc est *accessible* dans les sous-blocs. En revanche, un objet déclaré dans un sous-bloc n'est pas visible du bloc supérieur.
- ▶ Aucun objet manipulé par un sous-programme n'est sensible à la casse (*not case sensitive*). Ainsi numeroBrevet et NumeroBREVET désignent le même identificateur.

Figure 6-3 Visibilité des objets

```
BEGIN
DECLARE v_brevet CHAR(6);
...
-- v_brevet accessible
...
END;
```

v_nom inaccessible

```
BEGIN
DECLARE v_nom VARCHAR (20);
-- v_brevet et v_nom
accessibles
...
END;
```

Identificateur d'un sous-programme

- ▶ Un identificateur (nom des objets du sous-programme) commence par une lettre (ou un chiffre). Un identificateur n'est pas limité en nombre de caractères. Les autres signes toutefois connus du langage sont interdits, exemple :

| Autorisés | Interdits |
|---------------|------------------------------|
| t2 | moi&toi (symbole « & ») |
| code_brevet | debit-credit (symbole « - ») |
| 2nombresMysql | on/off (symbole « / ») |
| _t | code brevet (symbole espace) |

Les variables utilisées

- ▶ Un sous-programme est capable de manipuler des variables qui sont déclarées (et initialisées) par la directive DECLARE.
- ▶ Ces variables permettent de transmettre des valeurs à des sous-programmes via des paramètres, ou d'afficher des états de sortie sous l'interface.
- ▶ Deux types de variables sont disponibles sous MySQL
 - :
 - Scalaires : recevant une seule valeur d'un type SQL (ex : colonne d'une table) ;
 - Externes : définies dans la session et qui peuvent servir de paramètres d'entrée ou de sortie.

Variables scalaires

- ▶ La déclaration d'une **variable scalaire** est de la forme suivante :
- ▶ **DECLARE** *nomVariable1[, nomVariable2...]*
typeMySQL [DEFAULT expression];
- ▶ DEFAULT permet d'initialiser la (ou les) variable(s).
- ▶ Le tableau suivant décrit quelques exemples :

| Déclarations | Commentaires |
|--|---|
| DECLARE v_dateNaissance DATE; | Déclare la variable sans l'initialiser. Équivalent à SET v_dateNaissance = NULL; |
| DECLARE v_capacite SMALLINT(4) DEFAULT 999; | Initialise la variable à 999. |
| DECLARE v_trouve BOOLEAN DEFAULT TRUE; | Initialise la variable à vrai (1). |
| DECLARE v_Dans2jours DATE DEFAULT ADDDATE(SYSDATE(),2); | Initialise la variable à dans 2 jours. |

Affectation/récommendations

- ▶ Il existe plusieurs possibilités pour affecter une valeur à une variable :
 - l'affectation dans les langages de programmation (SET variable := expression).
 - la directive DEFAULT ;
 - la directive INTO d'une requête (SELECT... INTO variable FROM...).
- ▶ Pour éviter les conflits potentiels de noms (variables ou colonnes) dans des instructions SQL, il est recommandé de préfixer ses variables, et de suivre les conventions recommandées comme suivant :
 - Variable : **v_nomVariable** (*exemple : v_compteur*)
 - Constante : **c_nomConstante** (*exemple : c_pi*)
 - Variable de session (globale) : **vs_nomVariable** (*exemple : vs_brevet*)

Utilisation des paramètres

- ▶ Il est possible de passer en paramètres d'entrée d'un bloc des variables externes. Il existe 3 types de paramètre:
 - IN (entrée), OUT (sortie), INOUT (entrée/sortie);
- ▶ Exemple d'une procédure contenant un seul paramètre d'entrée avec utilisation de l'affectation INTO :

```
1 DELIMITER | -- Facultatif si votre délimiteur est toujours |
2 CREATE PROCEDURE afficher_titre_selon_id (IN p_id INT)
3     -- Définition du paramètre p_id
4 BEGIN
5     declare v_titre char(20); --variable interne
6         SELECT titreouvrage into v_titre
7         FROM ouvrage
8         WHERE nidentifiant = p_id; -- Utilisation du paramètre
9     select v_titre;
10 END |
11 DELIMITER ; -- On remet le délimiteur par défaut
```

Variables de session

- ▶ Il est possible de passer en paramètres d'entrée d'un bloc des variables externes. Ces variables sont dites de session (*user variables*). Elles n'existent que durant la session. On déclare ces variables en ligne de commande à l'aide du symbole « @ ».
- ▶ `SET @var1 = expression1 [, @var2 = expression2] ...`

Tableau 6-7 Variables de session

| Code MySQL | Résultat |
|--|---|
| <pre>SET @vs_num = 'PL-4'\$ SET @vs_hvol = 15\$... BEGIN DECLARE v_nom CHAR(16); DECLARE v_nbHVol DECIMAL(7,2); SELECT nom,nbHVol INTO v_nom, v_nbHVol FROM Pilote WHERE brevet = @vs_num; SET v_nbHVol := v_nbHVol + @vs_hvol; SELECT v_nom, v_nbHVol; END;</pre> | <pre>+-----+-----+ v_nom v_nbHVol +-----+-----+ Placide Fresnais 2465.00 +-----+-----+</pre> |

Exemple avec variable de session

Tableau 6-7 Variables de session

Code MySQL

```
SET @vs_num = 'PL-4'$  
SET @vs_hvol = 15$  
...  
BEGIN  
    DECLARE v_nom      CHAR(16);  
    DECLARE v_nbHVol DECIMAL(7,2);  
    SELECT nom,nbHVol  
        INTO v_nom, v_nbHVol FROM Pilote  
        WHERE brevet = @vs_num ;  
    SET v_nbHVol := v_nbHVol + @vs_hvol ;  
    SELECT v_nom, v_nbHVol;  
END;
```

Résultat

| v_nom | v_nbHVol |
|------------------|----------|
| Placide Fresnais | 2465.00 |

Utilisation de la fonction IF

- ▶ MySQL propose deux structures pour programmer des actions conditionnées : la structure IF et la structure CASE. Suivant les tests à programmer, on peut distinguer trois formes de structure IF :
 - IF-THEN (*si alors*),
 - IF-THEN-ELSE (avec le *sinon* à programmer), et
 - IF-THEN-ELSEIF (imbrications de conditions).

Exemple d'utilisation de IF

Tableau 6-10 Structures IF

| IF-THEN | IF-THEN-ELSE | IF-THEN-ELSEIF |
|--|---|---|
| IF condition THEN instructions; END IF ; | IF condition THEN instructions; ELSE instructions; END IF ; | IF condition1 THEN instructions; ELSEIF condition2 THEN instructions2; ELSE instructions3; END IF ; |
| BEGIN | | |
| DECLARE v_telephone CHAR(14) DEFAULT '06-76-85-14-89'; IF SUBSTR(v_telephone,1,2)='06' THEN SELECT "C'est un portable"; ELSE SELECT "C'est un fixe..."; END IF ; END; | | |

La structure CASE

- ▶ Comme l'instruction IF, la structure CASE permet d'exécuter une séquence d'instructions en fonction de différentes conditions.
- ▶ Elle est utile lorsqu'il faut évaluer une même expression et proposer plusieurs traitements pour diverses conditions.

| CASE | searched CASE |
|--|---|
| CASE variable WHEN expr1 THEN instructions1; WHEN expr2 THEN instructions2; ... WHEN exprN THEN instructionsN; [ELSE instructionsN+1;] END CASE; | CASE WHEN condition1 THEN instructions1; WHEN condition2 THEN instructions2; ... WHEN conditionN THEN instructionsN; [ELSE instructionsN+1;] END CASE; |

Exemple avec CASE

IF

```
BEGIN
DECLARE v_mention CHAR(2);
DECLARE v_note DECIMAL(4,2) DEFAULT 9.8; ...

IF v_note >= 16 THEN
    SET v_mention := 'TB';
ELSEIF v_note >= 14 THEN
    SET v_mention := 'B';
ELSEIF v_note >= 12 THEN
    SET v_mention := 'AB';
ELSEIF v_note >= 10 THEN
    SET v_mention := 'P';
ELSE
    SET v_mention := 'R';
END IF;
...
```

CASE

```
CASE
WHEN v_note >= 16 THEN SET v_mention := 'TB';
WHEN v_note >= 14 THEN SET v_mention := 'B';
WHEN v_note >= 12 THEN SET v_mention := 'AB';
WHEN v_note >= 10 THEN SET v_mention := 'P';
ELSE
    SET v_mention := 'R';
END CASE;
...
```



FIN DU COURS....