1. PRESENTATION DU LANGAGE SQL

a. HISTORIQUE

Le langage SQL (STRUCTURED QUERY LANGUAGE), langage d'interrogation structuré, a été élaboré dans les années 70 d'après les théories d'un informaticien britannique :Edgar Frank Codd.

Le Docteur Codd est considéré comme l'inventeur du modèle relationnel. Il travaillait au laboratoire de recherche d'IBM à San José en Californie. IBM mis du temps à croire en la théorie d'un langage d'interrogation des données structuré et ce sont des entreprises concurrentes telles Oracle qui les premières misèrent sur le langage SQL.

De par sa nature simple et presque naturelle, SQL ne tarda pas à devenir un standard de fait dans la manipulation des données.

Il fut rapidement normalisé, garantissant ainsi son indépendance vis-à-vis des systèmes de gestion de bases de données relationnelle (SGBDR). Ainsi, une requête SQL peut être portée sans modifications (ou alors mineures) de MYSQL à Oracle ou d'Oracle à SQL Server.

Voici un tableau récapitulant les différentes versions successives du langage SQL :

ANNEE	APPELLATION
1986	SQL-86 ou SQL-87
1989	SQL-89 ou SQL1
1992	SQL-92 ou SQL2
1999	SQL-99 ou SQL3
2003	SQL-2003
2008	SQL-2008
2011	SQL-2011 : Ajout du support des tables temporelles

Les versions apportant leurs lots de modifications, il est important avant d'utiliser l'une ou l'autre, de vérifier que le système de gestion de bases de données intègre la même version.

b. STRUCTURATION

Le langage SQL est composé de trois parties majeures et distinctes :

- Le DML (data manipulation language) ou LMD (langage de manipulation des données : le DML permet de consulter ou de modifier le contenu de la base de données.
- Le DDL (data definition language) ou LDD (langage de définition des données)) : le DDL permet de modifier la structure de la base de données.
- Le DCL (data control language) ou LCD (langage de contrôle des données) : le DCL permet de gérer les privilèges, ou les différents droits des utilisateurs sur la base de données.

Voici les principaux ordres employés :

DML	DDL	DCL
SELECT	CREATE	GRANT
INSERT	ALTER	REVOKE
DELETE	RENAME	
UPDATE	DROP	

Une requête SQL peut être employée seule, dans un éditeur de requêtes fourni par le logiciel de gestion de bases de données, ou directement intégré dans le langage de programmation. Par exemple, une requête peut être testée directement avec phpMyAdmin et être ensuite intégrée dans une procédure écrite en langage PHP pour interagir avec le gestionnaire de base de données MySQL.

c. MISE EN PRATIQUE AVEC MySQL

Tous les exemples donnés plus bas peuvent être testés avec un vrai serveur de base de données. Le serveur de base de données MySQL est un serveur réputé pour plusieurs raisons :

- Il existe sur de nombreuses plates-formes (Linux, Unix, Windows)
- Il supporte une grande montée en charge. Il est capable de gérer des entrepôts de données de plusieurs téraoctets.
- MySQL peut répondre aux demandes de performances les plus exigeantes. Il peut traiter un volume de requêtes s'exprimant en milliards de requêtes par jour.
- MySQL offre des fonctions de sécurité qui garantissent une très forte protection des données.
- MySQL est Open Source et gratuit en Licence Développement, payant en exploitation. Le coût de possession est très inférieur à ce que propose la concurrence.

Le site officiel français est : https://www.mysql.com/fr/

Il existe des packs logiciels installant une suite logicielle permettant d'installer un serveur MySQL sans effort. Voici certains de ces packs :

- EasyPHP : http://www.easyphp.org/
- XAMPP : http ://www.apachefriends.org/en/xampp.html
- Wampserver : http://www.wampserver.com/

Ces différents packs intègrent le serveur Web Apache, le langage PHP et MySQL. Ils offrent aussi des outils de configuration et de gestion de MySQL comme phpMyAdmin.

Pour ce qui concerne nos exemples, ce sera l'interface en ligne de commande de MySQL que nous allons utiliser. Elle se lance grâce à l'exécutable MySQL. Cette interface ressemble à une fenêtre DOS et permet de dialoguer très simplement avec la base de données. L'utilisation peut être interactive ou en mode batch. Dans le premier cas (c'est le mode le plus courant), le résultat des extractions est présenté sous une forme tabulaire au format ASCII.

Vous verrez qu'il est notamment possible :

- D'exécuter des instructions SQL (créer des tables, manipuler des données, extraire des informations, etc.) ;
- De compiler des procédures cataloguées et des déclencheurs ;
- De réaliser des tâches d'administration (création d'utilisateurs, attribution de privilèges, etc.).

Le principe général de l'interface est le suivant : après une connexion locale ou distante, des instructions sont saisies et envoyées à la base qui retourne des résultats affichés dans la même fenêtre de commande.

2. LE LANGAGE DE DEFINITION DES DONNEES

a. GESTION DES BASES DE DONNEES

Création d'une base de données

Pour pouvoir créer une base de données, vous devez posséder le privilège CREATE sur la nouvelle base (ou au niveau global pour créer toute table).

CREATE DATABASE [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 (auquel cas elle ne sera pas remplacée).
- nomBase désigne le nom de la base (64 caractères maximum, caractères compris par le système de gestion de fichier du système d'exploitation, notamment respectant les règles de nommage des répertoires). Les caractères « / », « \ », ou « . » sont proscrits.
- CHARACTER SET indique le jeu de caractères associé aux données qui résideront dans les tables de la base.
- COLLATE définit la collation associée au jeu de caractères précédemment choisi.

Une fois créée, vous constaterez la présence d'un répertoire portant le nom de cette nouvelle base (par défaut dans C:/ProgrammesData/MySQL/MySQL Server x.y/Data avec Windows 7 et 10). Ce répertoire contiendra les données de la nouvelle base.

L'instruction suivante décrit la création d'une base de données dont le jeu de caractères est compatible avec le standard Unicode, en restant compatible avec la norme ASCII sensible aux accents et à la casse.

```
CREATE DATABASE bd_utf8
        DEFAULT CHARACTER SET utf8
        COLLATE utf8_bin ;
```

Sélection d'une base de données

L'instruction USE sélectionne une base de données qui devient active dans une session.

USE nomBase ;

Remarque : si vous désirez travailler simultanément dans différentes bases de données, faites toujours préfixer le nom des tables par celui de la base par la notation pointée (nomBase.nomTable).

***Modification d'une base* (ALTER DATABASE)**

ALTER DATABASE vous permet de modifier le jeu de caractères par défaut d'une base de données. Pour pouvoir changer ainsi une base, vous devez avoir le privilège ALTER sur la base de données en question.

```
ALTER DATABASE nomBase [ [DEFAULT] CHARACTER SET nomjeu]
        [ [DEFAULT] COLLATE nomCollation ] ;
```

L'instruction suivante modifie la collation de la nouvelle base en la rendant insensible à la casse et à l'accentuation.

ALTER DATABASE bd_utf8 DEFAULT COLLATE utf8_general_ci ;

Suppression d'une base **(DROP DATABASE)**

Pour pouvoir supprimer une base de données, vous devez posséder le privilège DROP sur la base (ou au niveau global pour effacer toute base). Cette commande détruit tous les objets (tables, index, etc.) et le répertoire contenu dans la base.

DROP DATABASE [IF EXISTS] nomBase ;

- IF EXISTS évite une erreur dans le cas où la base de données n'existerait pas.
- Cette instruction retourne le nombre de tables qui ont été supprimées (fichiers à l'extension « .frm »).

L'instruction suivante supprime la base récemment créée.

DROP DATABASE bd_utf8 ;

b. GESTION DES TABLES

Création d'une table

Une table est créée en SQL par l'instruction CREATE TABLE, modifiée au niveau de sa structure par l'instruction ALTER TABLE et supprimée par la commande DROP TABLE. La syntaxe SQL simplifiée 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 | …] ;

- TEMPORARY : pour créer une table qui n'existera que durant la session courante (la table sera supprimée à la déconnexion). Deux connexions peuvent ainsi créer deux tables temporaires de même nom sans risquer de conflit. Il faut posséder le privilège CREATE TEMPORARY TABLES.
- IF NOT EXISTS : permet d'éviter qu'une erreur se produise si la table existe déjà (si c'est le cas, elle n'est aucunement affectée par la tentative de création).
- nomBase : (jusqu'à 64 caractères permis dans un nom de répertoire ou de fichier sauf (« / », « \ » et « . ») s'il est omis, il sera assimilé à la base connectée. S'il est précisé, il désigne soit la base connectée soit une autre base (dans ce cas il faut que l'utilisateur courant ait le droit de créer une table dans l'autre base).
- nomTable : mêmes limitations que pour le nom de la base.
- Colonne1 type1 : nom d'une colonne (mêmes caractéristiques que pour les noms des tables) et son type (INTEGER, CHAR, DATE,…). La directive DEFAULT fixe une valeur par défaut. La directive NOT NULL oblige la colonne à contenir une donnée pour chaque ligne (information obligatoire). A l'inverse, NULL permet à la colonne de ne pas être renseignée pour toutes les lignes.
- COMMENT : (jusqu'à 60 caractères) permet de commenter une colonne. Ce texte sera ensuite automatiquement affiché à l'aide des commandes SHOW CREATE TABLE et SHOW FULL COLUMNS.

- nomContraintei typeContraintei : nom de la contrainte et son type (clé primaire, clé étrangère, etc.).
- ENGINE : définit le type de table (par défaut InnoDB, bien adapté à la programmation de transaction). Le moteur MyISAM est très utile du fait de sa robustesse, mais il ne supporte ni les transactions ni l'intégrité référentielle. D'autres types existent, citons MEMORY pour les tables temporaires, ARCHIVE, etc.
- « ; » : symbole par défaut qui termine une instruction MySQL en mode ligne de commande (en l'absence d'un autre délimiteur).

REMARQUE : NULL n'est pas une valeur mais un concept (on parle de « marqueur » et non de valeur) qui permet de signifier « non disponible », « non affecté », « inconnu » ou « inapplicable ». NULL n'est pas la chaîne vide pour un caractère ou zéro pour un nombre. Vous devrez considérer avec attention toutes les données facultatives car, dans une comparaison SQL, deux NULL ne sont pas considérés comme identiques. Par ailleurs, pour d'autres opérateurs (regroupements ou ensemblistes), deux NULL sont cette fois considérés comme identiques.

Commentaires

Dans toute instruction SQL (déclaration, manipulation, interrogation et contrôle), il est possible d'inclure des retours chariot, des tabulations, espaces et commentaires (sur une 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 */). Les scripts suivants décrivent la déclaration d'une même table en utilisant différentes conventions (on crée auparavant la base de données BD_Test dans laquelle sera créée la table Test) :

```
CREATE TABLE Test (numero
DECIMAL(38,8)) ;
```

```
CREATE TABLE
Test
(numero
DECIMAL(38,8)
) ;
```

```
CREATE TABLE
        -- Nom de la table
Test ( #début de la description
        Numero DECIMAL(38,8)
)
-- Fin, ne pas oublier le point-virgule.
;
```

```
CREATE TABLE Test (
/* une plus grande description
Des colonnes */
Numero /* type : */ DECIMAL(38,8)) ;
```

Exemples : les tables suivantes serviront pour les exemples du cours

Client

numcli	nom	prenom	adresse	cp	ville	telephone

Article

numart	designation	categorie	prix

Achat

numcli	numart	dateAchat	qte

Créons d'abord la base de données « gestionCom » qui devra contenir les trois tables

CREATE DATABASE gestionCom ;

USE gestionCom ;

L'instruction SQL suivante permet de créer la table Client dans la base de données courante « gestionCom » :

CREATE TABLE Client (numcli int(5), nom VARCHAR(25),prenom VARCHAR(25),
Adresse VARCHAR(30), cp VARCHAR (10), #cp code postale du client
Ville VARCHAR (25) DEFAULT 'Bamako' COMMENT 'ville par défaut Bamako',
Telephone CHAR (8),
CONSTRAINT pk_client PRIMARY KEY (numcli)) ;

Contraintes

Les contraintes ont pour but de programmer des règles de gestion au niveau des colonnes des tables. Elles peuvent alléger un développement côté client (si on déclare qu'une note doit être comprise entre 0 et 20, les programmes de saisie n'ont plus à tester les valeurs en entrée mais seulement le code retour après connexion à la base ; on déporte les contraintes côté serveur).

Les contraintes peuvent être déclarées de deux manières.

- En même temps que la colonne (valeur pour les contraintes monocolonnes) ; ces contraintes sont dites « en ligne » (inline constraints). Exemples : DEFAULT, NOT NULL
- Après que la colonne soit déclarée : ces contraintes ne sont pas limitées à une colonne et peuvent être personnalisés par un nom (out-of-line constraints).

Il est recommandé de déclarer les contraintes NOT NULL en ligne, les autres peuvent soit être déclarées en ligne, soit être nommées. Etudions à présent les types de contraintes nommées (out-of-line). Les quatre types de contraintes les plus utilisées sont les suivants :

CONSTRAINT nomContrainte
UNIQUE (colonne1 [,colonne2]…)

CONSTRAINT nomContrainte
PRIMARY KEY (colonne1 [,colonne2]…)

CONSTRAINT nomContrainte
FOREIGN KEY (colonne1 [,colonne2]…)
REFERENCES nomTablePere [(colonne1 [,colonne2]…)]

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

CONSTRAINT nomContrainte
CHECK (condition)

- La contrainte UNIQUE impose une valeur distincte au niveau de la table (les valeurs nulles font exception à moins que NOT NULL soit aussi appliqué sur les colonnes).
- La contrainte PRIMARY KEY déclare la clé primaire de la table. Un index est généré automatiquement sur la ou les colonnes concernées. Les colonnes clés primaires ne peuvent être ni nulles ni identiques (en totalité si elles sont composées de plusieurs colonnes).
- La contrainte FOREIGN KEY déclare une clé étrangère entre une table enfant(child) et une table père (parent). Ces contraintes définissent l'intégrité référentielle. Les directives ON UPDATE ET ON DELETE disposent de quatre options que nous détaillons. Plusieurs scénarios sont possibles pour assurer la cohérence de la table « père » vers la table « fils » :
  - Prévenir la modification ou la suppression d'une clé primaire (ou candidate) de la table « père ». cette alternative est celle par défaut. Soit vous n'ajoutez pas d'option à la clause REFERENCES, soit vous utilisez NO ACTION pour les directives ON DELETE et ON UPDATE. Dans ce cas, la suppression ou la modification d'une clé primaire (ou candidate) n'est possible tant qu'elle est référencée par une clé étrangère.
  - Propager la suppression des enregistrements « fils » associés à l'enregistrement « père » supprimé. Ce mécanisme est réalisé par la directive ON DELETE CASCADE. Dès qu'un enregistrement de la table « père » est supprimé, tous les enregistrements de la table « fils » qui le référencent seront supprimés.
  - Etendre la modification de la clé primaire de l'enregistrement « père » aux enregistrements « fils » associés. Ce mécanisme est réalisé par la directive ON UPDATE CASCADE.
  - Propager l'affectation de la valeur nulle aux clés étrangères des enregistrements « fils » associés à l'enregistrement « père » supprimé ou modifié. Ce mécanisme est réalisé par la directive ON DELETE SET NULL (ou ON UPDATE SET NULL en cas de modification de la clé primaire du « père »). Dans ces deux cas, il ne faut pas poser de contraintes NOT NULL sur la clé étrangère.
- La contrainte CHECK impose un domaine de valeurs à une colonne ou une condition (exemples : CHECK (note BETWEEN 0 AND 20), CHECK (grade='Maître Assistant' or grade='Maître de Conférence'). Il est permis de déclarer des contraintes CHECK lors de la création d'une table mais le mécanisme de vérification des valeurs n'est toujours pas implémenté.

REMARQUE : il est recommandé de ne pas définir de contraintes sans les nommer (bien que cela soit possible), car il sera difficile de les faire évoluer (désactivation, réactivation, suppression), et la lisibilité des programmes en sera affectée.

Créons les tables article et achat dans la base de données gestionCom . si gestionCom n'est pas la base de données courante, voici les instructions qui permettent de créer les tables article et achat en mode ligne de commande

CREATE TABLE gestionCom.article
(numart INT(5), designation VARCHAR(60) NOT NULL, categorie VARCHAR (30),
prix INT(6) DEFAULT 0 COMMENT '0 est le prix par défaut',

```
CONSTRAINT pk_numart PRIMARY KEY (numart)) ;

CREATE TABLE gestionCom.achat
(numcli INT(5), numart INT(5), dateachat DATE, qte INT(4),
CONSTRAINT fk_numcli FOREIGN KEY (numcli) REFERENCES client(numcli),
CONSTRAINT fk_numart FOREIGN KEY (numart) REFERENCES article(numart),
CONSTRAINT pk_numcli_numart PRIMARY KEY (numcli,numart),
CONSTRAINT ch_qte CHECK (qte>=1)) ;
```

Types des colonnes

Pour décrire les colonnes d'une table, MySQL fournit les types prédéfinis suivants (built-in datatypes) :

- Caractères (CHAR, VARCHAR, TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT),
- Valeurs numériques (TINYINT, SMALLINT, MEDIUMINT, INT, INTEGER, BIGINT, FLOAT, DOUBLE, REAL, DECIMAL, NUMERIC et BIT),
- Date/heure (DATE, DATETIME, TIME, YEAR, TIMESTAMP),
- Données binaires (BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB),
- Enumérations (ENUM, SET)

Caractères

Le type CHAR permet de stocker des chaînes de caractères de taille fixe. Les valeurs sont stockées, en ajoutant, s'il le faut, des espaces (trailing spaces) à concurrence de la taille définie. Ces espaces ne seront pas considérés après extraction à partir de la table.

Le type VARCHAR permet de stocker des chaînes de caractères de taille variable. Les valeurs sont stockées sans l'ajout d'espaces à concurrence de la taille définie. Depuis la version 5.0.3 de MySQL, les éventuels espaces de fin de chaîne seront stockés et extraits en conformité avec la norme SQL. Des caractères Unicode (méthode de codage universelle qui fournit une valeur de code unique pour chaque caractère quels que soient la plate-forme, le programme ou la langue) peuvent aussi être stockés.

Type	Description	Commentaire pour une colonne
CHAR(n) [BINARY | ASCII | UNICODE]	chaîne fixe de n octets ou caractères.	Taille fixe (maximum de 255 caractères).
VARCHAR(n) [BINARY]	Chaîne variable de n caractères ou octets.	Taille variable (maximum de 65535 caractères).

Valeurs numériques

De nombreux types sont proposés par MySQL pour définir des valeurs exactes (entiers ou décimaux, positifs ou négatifs : INTEGER et SMALLINT), et des valeurs à virgule fixe ou flottante (FLOAT, DOUBLE et DECIMAL).

Type	Description
BIT[(n)]	Ensemble de n bits. Taille de 1 à 64 (par défaut 1).
BOOL et BOOLEAN	La valeur zéro est considérée comme fausse. Le non-zéro est considéré comme vrai

Type	Description
SMALLINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 2 octets) de -32 768 à 32 767 signé, 0 à 65 535 non signé
INTEGER[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 4 octets) de -2 147 483 648 à 2 147 483 647 signé, 0 à 4 294 967 295 non signé.
FLOAT[(n[,p])] [UNSIGNED] [ZEROFILL]	Flottant (de 4 à 8 octets) p désigne la précision simple (jusqu'à 7 décimales) de $-3.4\ 10^{+38}$ à $-1.1\ 10^{-38}$, 0, signé, et de $1.1\ 10^{-38}$ à $3.4\ 10^{+38}$ non signé
DOUBLE [(n[,p])] [UNSIGNED] [ZEROFILL]	Flottant (sur 8 octets) p désigne la précision double (jusqu'à 15 décimales) de $-1.7\ 10^{+308}$ à $-2.2\ 10^{-308}$, 0, signé, et de $2.2\ 10^{-308}$ à $1.7\ 10^{+308}$ non signé
DECIMAL [(n[,p])] [UNSIGNED] [ZEROFILL]	Décimal (sur 4 octets) à virgule fixe, p désigne la précision (nombre de chiffres après la virgule, maximum 30). Par défaut n vaut 10, p vaut 0. Nombre maximal de chiffres pour un décimal 65.

n indique le nombre de positions de la valeur à l'affichage.

La directive UNSIGNED permet de considérer seulement des valeurs positives

La directive ZEROFILL complète par des zéros à gauche une valeur (par exemple : soit un INTEGER(5) contenant 4, si ZEROFILL est appliqué, la valeur extraite sera « 00004 ». En déclarant une colonne ZEROFILL, MySQL l'affecte automatiquement à UNSIGNED aussi.

INT est synonyme de INTEGER.

REMARQUE : Dans toute instruction SQL, écrivez la virgule avec un point (7/2 retourne 3.5).

Dates et heures

Les types suivants permettent de stocker des moments ponctuels (dates, dates et heures, années, et heures). Les fonctions NOW() et SYSDATE() retournent la date et l'heure courantes. Dans une procédure ou un déclencheur SYSDATE est réévalué en temps réel, alors que NOW désignera toujours l'instant de début de traitement.

Type	Description	Commentaire pour une colonne
DATE	Dates du 1er janvier de l'an 1000 au 31 décembre 9999 après Jésus Christ	Sur 3 octets. L'affichage est au format 'YYYY-MM-DD'.
DATETIME	Dates et heures (de 0 h de la première date à 23 h 59 minutes 59 secondes de la dernière date)	Sur 8 octets. L'affichage est au format 'YYYY-MM-DD HH :MM :SS'.
YEAR [(2|4)]	Sur 4 positions : de 1901 à 2155 (incluant 0000). Sur 2 positions (autorisé jusqu'en version 5.5) : de 70 à 69 désignant 1970 à 2069	Sur 1 octet ; l'année est considérée sur 2 ou 4 positions (4 par défaut). Le format d'affichage est 'YYYY'.
TIME	Heures de -838 h 59 minutes 59 secondes à 838 h 59 minutes 59 secondes.	L'heure au format 'HHH :MM :SS' sur 3 octets
TIMESTAMP	Instants du 1er janvier 1970 0 h 0 minute 0 seconde à l'année 2037	Estampille sur 4 octets (au format 'YYYY-MM-DD HH :MM :SS') ; mise à jour à chaque modification sur la table

MySQL peut considérer les données de type date et heures soit en tant que chaînes de caractères soit comme numériques :

- Chaînes de caractères 'YYYY-MM-DD HH :MM :SS' ou 'YY-MM-DD HH :MM :SS' (pour les colonnes DATE 'YYYY-MM-DD' ou 'YY-MM-DD'). Tout autre délimiteur est autorisé comme : '2005.12.31 11%30%45' (pour les colonnes DATE, '65.12.31', '65/12/31', et '65@12@31' sont équivalents et désignent tous le réveillon de l'année 1965).
- Considérés comme chaînes de caractères dans les formats suivants : 'YYYYMMDDHHMMSS' ou 'YYMMDDHHMMSS' (pour les dates 'YYYYMMDD' ou 'YYMMDD') en supposant que la chaîne ait un sens en tant que date. Ainsi '19650205063000' est interprété comme le 5 février 1965 à 6 h 30 minutes. Par contre '19650255' n'a pas de sens du fait du jour (55), l'erreur sera retournée.
- Considérés comme numériques dans les formats suivants : YYYYMMDDHHMMSS ou YYMMDDHHMMSS (pour les DATE YYYYMMDD ou YYMMDD) en supposant que le nombre ait un sens en tant que date. Ainsi 19650205063000 est interprété comme le 5 février 1965 à 6 h et 30 minutes. Par contre 19650205069000 ou 19650205250000 n'ont pas de sens du fait des minutes (6 h et 90 mn) dans le premier cas et des heures (25h) pour le second (l'erreur sera retournée).

Données binaires

Les types BLOB (Binary Large Object) permettent de stocker des données non structurées comme le multimédia (images, sons, vidéo, etc.). Les quatre types de colonnes BLOB sont TINYBLOB, BLOB, MEDIUMBLOB, LONGBLOB. Ces types sont traités comme des flots d'octets sans jeu de caractère associé.

Type	Description	Commentaire pour une colonne
TINYBLOB(n)	Flot de n octets	Taille fixe (maximum de 255 octets)
BLOB(n)	Flot de n octets	Taille fixe (maximum de 65 535 octets)
MEDIUMBLOB(n)	Flot de n octets	Taille fixe (maximum de 16 mégaoctets)
LONGBLOB(n)	Flot de n octets	Taille fixe (maximum de 4,29 gigaoctets)

Enumération

Deux types de collections sont proposés par MySQL.

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

Type	Description
ENUM ('valeur1', 'valeur2',…)	Liste de 65 535 valeurs au maximum
SET ('valeur1', 'valeur2',…)	Ensemble de référence (maximum de 64 valeurs)

Les collations et jeux de caractères

Une collation est liée à un jeu de caractère et permet de classer les caractères dans le jeu (lors d'un tri ou d'une comparaison). Par exemple, il sera possible de différencier « à » de « a » (sensibilité diacritique). Chaque jeu de caractère possède plusieurs collations, dont une par défaut. Toutes les collations ont un nom qui commence par le jeu de caractères auquel elles sont liées et se termine par

_bin (binary), _cs(case sensitive) ou _ci (case insensitive). Concernant les collations binaires, les caractères sont ordonnés selon leurs numéros de code (d'abord les majuscules, puis les minuscules, puis les lettres accentuées). Les collations non binaires permettent de trier selon le langage, en choisissant la sensibilité à la casse.

Le tableau suivant résume les caractéristiques de deux jeux de caractères : le latin qui concerne l'Europe de l'ouest et l'UTF8 qui est conforme au standard Unicode.

Jeu de caractères	Collation	Sensible à la casse	Sensible aux accents
Latin1	Latin1_bin	oui	oui
	Latin1_general_cs	oui	oui
	Latin1_general_ci	non	oui
Utf8	Utf8_bin	oui	oui
	Utf8_general_ci	non	non
	Utf8_general_bin	non	non

Dans l'exemple suivant, le jeu de caractères de la table est positionné à utf8, mais deux colonnes sont fixées à latin1 en assurant une sensibilité à la casse pour le nom de la compagnie.

```
CREATE DATABASE Bdutil ;

CREATE TABLE Bdutil.compagnie (comp CHAR (4), nrue INTEGER (3),Rue VARCHAR (20),
        Ville VARCHAR (15) CHARACTER SET latin1 COLLATE latin1_general_ci,
        nomComp VARCHAR (15) CHARACTER SET latin1 COLLATE latin1_general_cs)
        DEFAULT CHARACTER SET utf8 COLLATE utf8_bin ;
```

Les données utilisent le jeu de caractères et la collation de leur colonne. Si la collation n'a pas été spécifiée pour la colonne, elle adopte celle de la table. Si la collation de la table n'a pas été spécifiée, elle adopte la collation de la base. Si la collation de la base n'a pas été spécifiée non plus, le jeu de caractères et la collation par défaut sont ceux du serveur.

La commande SHOW CHARACTER SET vous donnera la liste des jeux de caractères à votre disposition et SHOW VARIABLES LIKE 'collation%' indiquera les collations par défaut au niveau de la connexion, de la base et du serveur.

Structure d'une table

DESCRIBE permet d'extraire la structure brute d'une table ou d'une vue.

DESCRIBE [nomBase.] nomTableouVue [colonne] ;

Si la base n'est pas indiquée, il s'agit de celle en cours d'utilisation. Retrouvons la structure des tables client et article précédemment créées. Le type de chaque colonne apparait :

```
mysql> DESCRIBE gestionCom.client ;

mysql> DESCRIBE gestionCom.article ;
```

REMARQUE : la commande SHOW CREATE TABLE [nomBase.]nomTable restitue l'instruction complète qui a permis la création de la table en question. La commande SHOW FULL COLUMNS FROM [nomBase.]nomTable ; est plus complète que l'instruction DESCRIBE.

```
mysql> SHOW CREATE TABLE gestionCom.client ;
```

mysql> SHOW FULL COLUMNS FROM gestionCom.client ;

Suppression d'une table [DROP TABLE]

Les tables « père » doivent être créées avant les tables « fils » si des contraintes (contrainte d'intégrité référentielle) sont définies en même temps que les tables. L'ordre de destruction des tables, pour des raisons de cohérence, est inverse (il faut détruire les tables « fils » puis les tables « pères »).

Pour pouvoir supprimer une table, il faut posséder le privilège DROP sur cette base. L'instruction 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.

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

- TEMPORARY : pour supprimer des tables temporaires. Les transactions en cours ne sont pas affectées. L'utilisation de TEMPORARY peut être un bon moyen de s'assurer qu'on ne détruit pas accidentellement une table non temporaire.
- IF EXISTS : permet d'éviter qu'une erreur se produise si la table n'existe pas.
- RESTRICT (par défaut) permet de vérifier qu'aucun autre élément n'utilise la table (autre table via une clé étrangère, vue, déclencheur, etc.)
- CASCADE, option qui n'est pas encore opérationnelle, doit répercuter la destruction à toutes les autres tables référencées par des clés étrangères.

Les éléments qui utilisaient la table (vues, synonymes, fonctions et procédures) ne sont pas supprimés mais sont temporairement inopérants. Attention, une suppression ne peut pas être par la suite annulée.

Exemple :

DROP TABLE test ;

3. LE LANGAGE DE MANIPULATION DES DONNEES

SQL propose trois instructions pour manipuler des données :

- L'insertion d'enregistrements : INSERT ;
- La modification de données : UPDATE ;
- La suppression d'enregistrements : DELETE (et TRUNCATE).

Il existe d'autres possibilités pour insérer des données dans une base. Vous pouvez recourir à des commandes en ligne d'administration et de programmation (comme LOAD DATA INFILE), ou à des outils d'importation et de migration, parmi lesquels nous pouvons citer MYSQL Workbench, phpMyAdmin, Navicat et EMS SQL Management Studio.

a. INSERTION D'ENREGISTREMENTS [INSERT]

Pour pouvoir insérer des enregistrements dans une table, il faut que vous ayez reçu le privilège INSERT. Il existe plusieurs possibilités d'insertion : l'insertion monoligne qui ajoute un enregistrement par instruction et l'insertion multiligne qui insère plusieurs enregistrements par une requête.

La syntaxe simplifiée de l'instruction INSERT monoligne est la suivante :

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
        [INTO] [nomBase.] {nomTable | nomVue} [(nomColonne,…)]
        VALUES ({expression | DEFAULT},…),(…),…
        [ON DUPLICATE KEY UPDATE nomColonne = expression,…]
```

- DELAYED : indique que l'insertion est différée (si la table est modifiée par ailleurs, le serveur attend qu'elle se libère pour y insérer périodiquement de nouveaux enregistrements si elle redevient active entre-temps).
- LOW_PRIORITY diffère l'exécution de la commande tant qu'il existe un client qui accède à la table. Cela ne concerne que les moteurs de stockage qui verrouillent au niveau table (MyISAM, MEMORY et MERGE)
- HIGH_PRIORITY annule l'option low priority du serveur.
- IGNORE indique que les éventuelles erreurs déclenchées suite à l'insertion seront considérées en tant que warnings
- ON DUPLICATE KEY UPDATE permet de mettre à jour l'enregistrement présent dans la table, qui a déclenché l'erreur de doublon (dans le cas d'un index UNIQUE ou d'une clé primaire). Dans ce cas le nouvel enregistrement n'est pas inséré, seul l'ancien est mis à jour.

Les doublons

Lorsqu'une insertion déclenche une valeur en doublon concernant une colonne unique (clé primaire ou contrainte UNIQUE), par défaut l'insertion est refusée (à juste titre). En revanche si l'option ON DUPLICATE KEY UPDATE est présente, le nouvel enregistrement n'est pas inséré, mais l'ancien peut être éventuellement modifié. Le script suivant présente cette option.

```
-- Création de la table sgbd
CREATE TABLE sgbd
        (a INTEGER(4) PRIMARY KEY, b CHAR(6)) ;

-- insertion d'un premier enregistrement

INSERT INTO sgbd (a,b) VALUES (1995,'Oracle') ;
```

-- Affichage de l'enregistrement

mysql > SELECT a,b FROM sgbd ;

a	b
1995	Oracle

-- Insertion d'un deuxième enregistrement

INSERT INTO sgbd (a,b) VALUES (1995, 'MySQL')

ON DUPLICATE KEY UPDATE a=a+15 ;

-- Affichage du résultat

mysql> SELECT a,b FROM sgbd ;

a	b
2010	Oracle

Le premier et le deuxième enregistrement ayant la même valeur (1995) pour la colonne clé primaire (a), le problème de doublon se pose. Le deuxième enregistrement n'est pas inséré. Mais à cause de la présence de l'option ON DUPLICATE KEY UPDATE, le premier enregistrement est modifié (a prend la valeur de a+15, ce qui fait que a=2010).

Renseigner toutes les colonnes

Ajoutons trois lignes dans la table client en alimentant toutes les colonnes de la table par des valeurs. La deuxième insertion utilise le mot-clé DEFAULT pour affecter explicitement la valeur par défaut à la colonne ville. La troisième insertion attribue explicitement la valeur NULL à la colonne telephone.

```
INSERT INTO client
        VALUES (1,'Diarra','Oumar','1 Rue Droite','30000','Segou','23104750')
/* Toutes les valeurs sont renseignées dans l'ordre de la structure de la table. */;

INSERT INTO client
        VALUES (2,'Coulibaly','Karim','7 Rue Titi','12000',DEFAULT,'76483901') /* DEFAULT explicite */ ;

INSERT INTO client
        VALUES (3,'Coulibaly','Aminata','Avenue Travélé','12000','Kayes',NULL) /* NULL explicite */ ;
```

Renseigner certaines colonnes

Insérons deux lignes dans la table client en ne précisant pas toutes les colonnes. Le nombre de valeurs de la requête (dans la clause VALUES) doit coïncider avec le nombre de colonnes de destination.

```
INSERT INTO client (numcli,nom,prenom,adresse,cp,ville)
        VALUES (4,'Sidibé','Kadia','Rue Tombouctou','6600','Mopti') ;
```

Cette insertion affecte implicitement la valeur NULL à la colonne telephone.

```
INSERT INTO client (numcli,nom,prenom,adresse,cp,telephone)
        VALUES (5,'Touré','Sidiki','Rue Maridjé','30000','66485423') ;
```

Cette insertion affecte implicitement la valeur par défaut à la colonne ville.

La table client contient à présent les lignes suivantes :

mysql> SELECT * FROM client ;

numcli	nom	prenom	adresse	cp	ville	Telephone
1	Diarra	Oumar	1 Rue Droite	30000	Segou	23104750
2	Coulibaly	Karim	7 Rue Titi	12000	Bamako	76483901
3	Coulibaly	Aminata	Avenue Travélé	12000	Kayes	NULL
4	Sidibé	Kadia	Rue Tombouctou	66000	Mopti	NULL
5	Touré	Sidiki	Rue Maridjé	30000	Bamako	66485423

Plusieurs enregistrements

L'instruction suivante ajoute six lignes à la table article en une seule instruction INSERT.

```
INSERT INTO article VALUES
        (1,'Guimba','CD',2000),
        (2,'Sodia','DVD',3000),
        (3,'WebCam','Informatique',4000),
        (4,'Graveur','Informatique',9000),
        (5,'Clé Usb 16 Go','Informatique',2500),
        (6,'Befo','DVD',2000) ;
```

mysql> SELECT * FROM article ;

numart	designation	categorie	prix
1	Guimba	CD	2000
2	Sodia	DVD	3000
3	WebCam	Informatique	4000
4	Graveur	Informatique	9000
5	Clé Usb 16 Go	Informatique	2500
6	Befo	DVD	2000

Respecter les contraintes

Toute insertion des enregistrements dans les tables doit respecter les contraintes définies sur les colonnes (contrainte d'unicité, contrainte d'intégrité référentielle, contrainte de non nullité, etc.). Le non-respect des contraintes empêche l'insertion et entraîne le renvoi des messages d'erreur.

Insertions multilignes

Il est possible d'insérer plusieurs enregistrements dans une table en une seule instruction INSERT. On parle alors d'instruction multiligne. La syntaxe de ce mécanisme est présentée ci-après. Les options sont identiques à l'insertion classique mise à part la directive SELECT qui décrit l'extraction des données à insérer dans la table.

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
[INTO] [nomBase.] nomTable [(nomColonne,…)]
SELECT…
[ON DPLICATE KEY UPDATE colonne=expression,…]

Exemple : créons une nouvelle table clientBis dans la base de données gestionCom et insérons dedans tous les enregistrements de la table client.

CREATE TABLE ClientBis (numcli int(5), nom VARCHAR(30),prenom VARCHAR(30),
Adresse VARCHAR(30), cp VARCHAR (10),
Ville VARCHAR (25) DEFAULT 'Bamako' COMMENT 'ville par défaut Bamako',
Telephone CHAR (8),
CONSTRAINT pk_clientBis PRIMARY KEY (numcli)) ;

INSERT INTO ClientBis (numcli,nom,prenom,adresse,cp,ville,telephone)
SELECT numcli,nom,prenom,adresse,cp,ville,telephone FROM Client ;

Affichons maintenant toutes les lignes de la table ClientBis.

mysql> SELECT * FROM ClientBis ;

numcli	nom	prenom	adresse	cp	ville	Telephone
1	Diarra	Oumar	1 Rue Droite	30000	Segou	23104750
2	Coulibaly	Karim	7 Rue Titi	12000	Bamako	76483901
3	Coulibaly	Aminata	Avenue Travélé	12000	Kayes	NULL
4	Sidibé	Kadia	Rue Tombouctou	66000	Mopti	NULL
5	Touré	Sidiki	Rue Maridjé	30000	Bamako	66485423

Données binaires

Le type BIT permet de manipuler des suites variables de bits. Des fonctions sont disponibles pour programmer le « ET », le « OU » exclusif ou inclusif, etc. La table suivante contient deux colonnes de ce type. Notez l'utilisation du préfixe « b » pour initialiser un tel type.

CREATE TABLE Registres (nom CHAR (5), numero BIT (2), adresse BIT (16)) ;

INSERT INTO Registres VALUES ('COM2', b'10', b'0000010011110111') ;

Enumérations

Deux types d'énumérations existent : le type ENUM (énumération simple), qui permet de vérifier la présence d'une valeur dans une liste, et le type SET (énumération multiple), qui permet de vérifier la présence de plusieurs valeurs dans une liste.

Le type ENUM

Le type ENUM se définit par une liste de chaînes de caractères lors de la création de la table. Toute valeur d'une colonne de ce type devra être présente dans cette liste. Supposons que l'on recense 4 diplômes ('BTS', 'DUT', 'License' et 'INSA'), chacun affecté ou non à un étudiant.

Uncursus

num	nom	diplome
E1	Ballo	BTS
E2	Diarra	License
E3	Mallé	NULL

Le script de création de la table et des insertions valides et incorrectes est donné ci-dessous. Notez la présence de NULL à la création de l'énumération (autorise l'absence de valeur) et lors d'insertions. Par ailleurs , les parenthèses sont optionnelles pour renseigner la colonne ENUM.

```
CREATE TABLE Uncursus (num CHAR(4), nom VARCHAR(15),
        diplome ENUM ('BTS', 'DUT', 'License', 'INSA') NULL,
        CONSTRAINT pk_uncursus PRIMARY KEY (num)) ;
```

Insertions valides :

```
INSERT INTO uncursus (num, nom, diplome) VALUES ('E1', 'Ballo',('BTS')) ;

INSERT INTO uncursus (num, nom, diplome) VALUES ('E2', 'Diarra','License') ;

INSERT INTO uncursus (num, nom) VALUES ('E3', 'Mallé') ;
```

Insertions incorrectes :

```
INSERT INTO Uncursus (num, nom, diplome) VALUES ('E4', 'Sarré','Mathsup') ;
```

Un message d'erreur est affiché puisque « Mathsup » n'existe pas dans la liste des diplomes ('BTS', 'DUT', 'License', 'INSA') fournie lors de la création de la table.

```
INSERT INTO Uncursus (num, nom, diplome) VALUES ('E5', 'Danté',('License','BTS')) ;
```

Un message d'erreur est affiché puisqu'une seule valeur d'énumération est permise par la colonne.

```
SELECT * FROM Uncursus ;
```

num	nom	diplome
E1	Ballo	BTS
E2	Diarra	License
E3	Mallé	NULL

Le type SET

Le type SET se définit aussi par une liste de chaînes de caractères lors de la création de la table. Toute valeur (ou liste de valeurs) d'une colonne de ce type devra être incluse dans la liste de référence. Supposons à présent que chaque étudiant puisse être détenteur d'un ou de plusieurs des 4 diplômes ('BTS', 'DUT', 'License' et 'INSA').

Cursus

num	nom	[diplome]
E1	Ballo	BTS, License
E2	Savané	DUT, License, INSA
E3	Coulibaly	BTS
E4	Touré	BTS, DUT, License, INSA
E5	Sylla	NULL

Le script de création de la table et des insertions valides et incorrectes est le suivant. Notez la présence de NULL à la création de l'énumération multiple (ce qui autorise l'absence de valeur) et lors d'insertions. Par ailleurs, les parenthèses sont optionnelles pour renseigner la colonne SET.

Les doublons sont permis à l'insertion et l'ordre importe peu. Les doublons seront ignorés au niveau du stockage et l'ordre de stockage des différentes valeurs suivra l'ordre alphabétique.

```
CREATE TABLE Cursus (num CHAR (4), nom VARCHAR (15),
        diplomes SET ('BTS', 'DUT', 'License', 'INSA') NULL,
        CONSTRAINT pk_cursus PRIMARY KEY (num)) ;
```

Insertions valides :

```
INSERT INTO Cursus (num, nom, diplomes) VALUES ('E1', 'Ballo','BTS,License')) ;

INSERT INTO Cursus (num, nom, diplomes) VALUES ('E2', 'Savané','License,INSA,DUT')) ;

INSERT INTO Cursus (num, nom, diplome) VALUES ('E3', 'Coulibaly','BTS,BTS')) ;

INSERT INTO Cursus (num, nom, diplome) VALUES ('E4', 'Touré',
('License,BTS,INSA,BTS,INSA,INSA,DUT')) ;

INSERT INTO Cursus (num, nom) VALUES ('E5', 'Sylla') ;
```

Insertions incorrectes :

```
INSERT INTO Cursus (num, nom, diplome) VALUES ('E6', 'Traoré','BTS,INSA,ENAC')) ;
```

Un message d'erreur est affiché. Cela s'explique par l'absence d'une valeur (« ENAC ») dans la liste de référence.

```
SELECT * FROM Cursus ;
```

num	nom	[diplome]
E1	Ballo	BTS, License
E2	Savané	DUT, INSA, License
E3	Coulibaly	BTS
E4	Touré	BTS, DUT, INSA, License
E5	Sylla	NULL

Dates et heures

Déclarons la table Employe qui contient deux colonnes de type date/heure : DATE et DATETIME :

```
CREATE TABLE Employe
        (matricule CHAR(6), nom VARCHAR (30), dateNaiss DATE,
        dateEmbauche DATETIME ,
        CONSTRAINT pk_Employe PRIMARY KEY (matricule)) ;

INSERT INTO Employe (matricule,nom,dateNaiss,dateEmbauche)
        VALUES ('Em1', 'Oumar Kéita', '1965-02-05', SYSDATE() ) ;
INSERT INTO Employe (matricule,nom,dateNaiss,dateEmbauche)
        VALUES ('Em2', 'Aoua Koita', '1970/06/21', SYSDATE()) ;
```

INSERT INTO Employe (matricule,nom,dateNaiss,dateEmbauche)
VALUES ('Em3', 'Toumani Oulalé', '19720718', SYSDATE()) ;

Notez que la date de naissance a été saisie de trois manières différentes toutes valides. La fonction SYSDATE() retourne la date du moment (année,mois,jour,heures, minutes, secondes).
SELECT * FROM Employe ;

matricule	nom	dateNaiss	dateEmbauche
Em1	Oumar Kéita	1965-02-05	2020-01-09 09:53:08
Em2	Aoua Koita	1970-06-21	2020-01-09 09:54:12
Em3	Toumani Oulalé	1972-07-18	2020-01-09 09:54:56

Voyons un autre exemple avec TIMESTAMP. Par défaut, toute colonne du type TIMESTAMP est actualisée lors de toute modification de l'enregistrement (la première fois à l'INSERT, puis à chaque UPDATE, quelle que soit la colonne mise à jour). Déclarons la table EmployeBis qui contient une colonne de ce type.

CREATE TABLE EmployeBis (matricule VARCHAR(6), nom VARCHAR(30),
misaJour TIMESTAMP, CONSTRAINT pk_EmpBis PRIMARY KEY (matricule) ;

l'insertion de l'employé suivant initialisera la colonne misaJour à la date et heure système.

INSERT INTO EmployeBis (matricule,nom) VALUES ('Em1','Abdoulaye Traoré') ;

SELECT * FROM employeBis ;

matricule	nom	misaJour
Em1	Abdoulaye Traoré	2020-01-09 10:49:39

Par la suite, et à la différence d'un type DATE et DATETIME, pour chaque modification de cet employé, la colonne misaJour sera réactualisée avec la date et l'instant de la mise à jour. Même si vous désirez forcer à NULL cette colonne avec une instruction UPDATE, elle contiendra toujours l'instant de votre vaine tentative.

Modifions par exemple le nom de l'employé : Abdoulaye Diarra à la place de Abdoulaye Traoré.

UPDATE EmployeBis SET nom='Abdoulaye Diarra' ;

SELECT * FROM employebis ;

matricule	nom	misaJour
Em1	Abdoulaye Diarra	2020-01-09 11:12:02

L'heure de la mise à jour a changé. Elle est actualisée car l'enregistrement concerné a subi une modification. La colonne misaJour donne le jour et l'heure de la dernière modification de l'enregistrement concerné.

Insérons un second employé.

INSERT INTO EmployeBis (matricule,nom) VALUES ('Em2','Ousmane Coulibaly') ;

SELECT * FROM employeBis ;

matricule	Nom	misaJour
Em1	Abdoulaye Diarra	2020-01-09 11:12:02
Em2	Ousmane Coulibaly	2020-01-09 19:54:41

Modifions le second enregistrement : Amara coulibaly en remplacement de Ousmane Coulibaly.

UPDATE EmployeBis SET nom='Amara Coulibaly' WHERE matricule='Em2' ;

SELECT * FROM employebis ;

matricule	Nom	misaJour
Em1	Abdoulaye Diarra	2020-01-09 11:12:02
Em2	Amara Coulibaly	2020-01-09 20:08:26

Comme vous pouvez le constater l'insertion et la mise à jour du second enregistrement n'affecte en rien la valeur de la colonne misaJour du premier enregistrement. Seul l'enregistrement inséré ou modifié est concerné par la mise à jour de la date et de l'heure (colonne misaJour).

Séquences

Dans le vocabulaire de MySQL, pour le moment, séquence n'existe pas. Cependant, MySQL offre la possibilité de générer automatiquement des valeurs numériques. Ces valeurs sont bien utiles pour composer des clés primaires de tables quand vous ne disposez pas de colonnes adéquates à cet effet. Ce mécanisme répond en grande partie à ce qu'on attendrait d'une séquence.

Utilisation en tant que clé primaire

L'instruction SQL suivante illustre la séquence appliquée à la colonne numClient pour initialiser les valeurs de la clé primaire de la table Client. La fonction LAST_INSERT_ID() retourne la dernière valeur de la séquence générée (ici le pas est de 1, la séquence débute par défaut à 1).

```
CREATE TABLE Client (numClient INT AUTO_INCREMENT, prenom VARCHAR(15),
        Nom VARCHAR(15),adresse VARCHAR(20),
        CONSTRAINT pk_Client PRIMARY KEY (numClient)) ;
```

INSERT INTO Client (prenom,nom,adresse) VALUES ('Souleymane','Oulalé,'Segou') ;

INSERT INTO Client (prenom,nom,adresse) VALUES ('Alfred','Sanogo',Sikasso') ;

INSERT INTO Client (numClient,prenom,nom,adresse) VALUES (NULL,
'Boubacar','Bagayoko','Bamako') ;

INSERT INTO Client VALUES (0, 'Ousmane' , 'Taguem' , 'Mopti') ;

Toutes ces insertions sont valides même s'il n'est pas possible de forcer une insertion pour la séquence. Les tentatives d'insérer les valeurs NULL et 0 ne sont pas prises en compte. La numérotation automatique continue.

SELECT * FROM Client ;

numClient	prenom	nom	Adresse
1	Souleymane	Oulalé	Segou
2	Alfred	Sanogo	Sikasso
3	Boubacar	Bagayoko	Bamako
4	Ousmane	Taguem	Mopti

Pour retourner la dernière valeur de la séquence :

SELECT LAST_INSERT_ID() ;

LAST_INSERT_ID()
4

Modification d'une séquence

La seule modification possible d'une séquence est celle qui consiste à changer la valeur de départ de la séquence (avec ALTER TABLE). Seules les valeurs à venir de la séquence modifiée seront changées (heureusement pour les données existantes des tables).

Supposons qu'on désire continuer à insérer des nouveaux clients à partir de la valeur 100. Le prochain client sera estimé à 100 et les insertions suivantes prendront en compte le nouveau point de départ tout en laissant intactes les données existantes des tables.

ALTER TABLE Client AUTO_INCREMENT=100 ;

INSERT INTO Client (prenom,nom,adresse) VALUES ('Moussa' , 'Dembélé' , 'Segou') ;

INSERT INTO Client (prenom,nom,adresse) VALUES ('Mariam' , 'Sissoko' , 'Kayes') ;

SELECT * FROM Client ;

numClient	prenom	nom	Adresse
1	Souleymane	Oulalé	Segou
2	Alfred	Sanogo	Sikasso
3	Boubacar	Bagayoko	Bamako
4	Ousmane	Taguem	Mopti
100	Moussa	Dembélé	Segou
101	Mariam	Sissoko	Kayes

Utilisation en tant que clé étrangère

Soit la table commande ayant les colonnes numCommande, dateCommande et numClient. numCommande est la clé primaire. Utilisons une séquence pour générer ses valeurs. La valeur de départ d'une séquence peut-être définie à la fin de l'ordre CREATE TABLE. Notez également l'utilisation de la fonction LAST_INSERT_ID() dans les insertions pour récupérer la valeur de la clé primaire.

```
CREATE TABLE Commande (numCommande INT AUTO_INCREMENT,
    dateCommande DATE, numClient INT,
    CONSTRAINT pk_Commande PRIMARY KEY (numCommande),
    CONSTRAINT fk_Com_Client FOREIGN KEY (numClient)
    REFERENCES Client (numClient)) ;
INSERT INTO Commande VALUES (NULL , '20190621' , LAST_INSERT_ID()) ;
```

SELECT * FROM Commande ;

numCommande	dateCommande	numClient
1	2019-06-21	101

Insérons un nouveau client dans la table Client. Supposons que la nouvelle commande est passée par le nouveau client. Le script suivant illustre cette situation.

INSERT INTO Client (prenom,nom,adresse) VALUES ('Daouda' , 'Tembely' , 'Mopti') ;

SELECT * FROM Client ;

numClient	prenom	nom	Adresse
1	Souleymane	Oulalé	Segou
2	Alfred	Sanogo	Sikasso
3	Boubacar	Bagayoko	Bamako
4	Ousmane	Taguem	Mopti
100	Moussa	Dembélé	Segou
101	Mariam	Sissoko	Kayes
102	Daouda	Tembely	Mopti

INSERT INTO Commande (dateCommande,numClient)
VALUES ('20190624' , LAST_INSERT_ID()) ;

SELECT * FROM Commande ;

numCommande	dateCommande	numClient
1	2019-06-21	101
2	2019-06-24	102

REMARQUE : Le mécanisme d'auto-incrémentation est très utilisé en production. En effet, il est pratique et optimal que chaque table dispose d'une clé primaire sous la forme d'une séquence (index compact et efficace du fait que les lignes sont proches dans les blocs de données). On parle de clé artificielle.

Par ailleurs, il est souhaitable que chaque table dispose également d'une clé métier (ayant une sémantique forte comme le code d'un produit ou le numéro matricule d'un employé). Cette dernière peut être implémentée sous la forme d'une contrainte unique afin de bénéficier d'un index.

b. MODIFICATIONS DE COLONNES [UPDATE]

L'instruction UPDATE permet la mise à jour des colonnes d'une table. Pour pouvoir modifier des enregistrements d'une table, il faut que cette dernière soit dans votre base ou que vous ayez reçu le privilège UPDATE sur la table.

```
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]
```

- LOW_PRIORITY diffère l'exécution de la commande tant qu'il existe un client qui accède à la table. Cela ne concerne que les moteurs de stockage qui verrouillent au niveau table (MyISAM,MEMORY et MERGE).
- IGNORE signifie que les éventuelles erreurs déclenchées suite aux modifications seront considérées en tant que warnings.
- La clause SET actualise une colonne en lui affectant une expression (valeur, valeur par défaut, calcul ou résultat d'une requête).
- La condition du WHERE filtre les lignes à mettre à jour dans la table. Si aucune condition n'est précisée, tous les enregistrements seront actualisés. Si la condition ne filtre aucune ligne, aucune mise à jour ne sera réalisée.
- ORDER BY indique l'ordre de modification des colonnes.
- LIMIT spécifie le nombre maximum d'enregistrements à changer (par ordre de clé primaire croisante)

Exemple : modifions le client dont le numéro est 1 en affectant la valeur 20231047 à la colonne telephone

UPDATE Client SET telephone ='20231047' WHERE numcli=1 ;

Modifions le Client numéro 3 en affectant simultanément la valeur 15000 à la colonne cp et la valeur par défaut ('Bamako') à la colonne ville.

UPDATE Client SET cp='15000', ville=DEFAULT WHERE numcli=3 ;

La table Client contient à présent les données suivantes

mysql> SELECT * FROM client ;

numcli	nom	prenom	adresse	cp	ville	Telephone
1	Diarra	Oumar	1 Rue Droite	30000	Segou	20231047
2	Coulibaly	Karim	7 Rue Titi	12000	Bamako	76483901
3	Coulibaly	Aminata	Avenue Travélé	15000	Bamako	NULL
4	Sidibé	Kadia	Rue Tombouctou	66000	Mopti	NULL
5	Touré	Sidiki	Rue Maridjé	30000	Bamako	66485423

Modification de plusieurs enregistrements

Modifions les 2 premiers clients (par ordre de clé primaire) en affectant la valeur 'Sikasso' à la colonne ville.

UPDATE client SET ville='Sikasso' LIMIT 2 ;

mysql>SELECT * FROM client ;

numcli	nom	prenom	adresse	cp	ville	Telephone
1	Diarra	Oumar	1 Rue Droite	30000	Sikasso	20231047
2	Coulibaly	Karim	7 Rue Titi	12000	Sikasso	76483901
3	Coulibaly	Aminata	Avenue Travélé	15000	Bamako	NULL
4	Sidibé	Kadia	Rue Tombouctou	66000	Mopti	NULL
5	Touré	Sidiki	Rue Maridjé	30000	Bamako	66485423

Respecter les contraintes

Il faut, comme pour les insertions, respecter les contraintes qui existent au niveau des colonnes. Dans le cas inverse, une erreur est renvoyée (le nom de la contrainte apparait) et la mise à jour n'est pas effectuée.

Restrictions

Il n'est pas possible d'utiliser la table qui est modifiée par UPDATE dans une sous-requête de la clause SET .

Exemple :

```
UPDATE article
        SET prix= (SELECT prix/2 FROM article WHERE numart=4)
        WHERE numart=2 ;
```

Cette instruction renvoie une erreur car la table à modifier est **article** (UPDATE article) et la table utilisée dans la sous-requête de la clause SET est la même table **article**.

c. SUPPRESSION D'ENREGISTREMENTS [DELETE]

L'instruction DELETE permet de supprimer un ou plusieurs enregistrements d'une table. Pour pouvoir supprimer des enregistrements dans une table, il faut que cette dernière soit dans votre base ou que vous ayez reçu le privilège DELETE sur la table.

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM [nomBase.]nomTable
        [WHERE (condition)]
        [ORDER BY listeColonnes]
        [LIMIT nbreLimite]
```

- LOW_PRIORITY, IGNORE et LIMIT ont la même signification que pour les instructions INSERT et UPDATE
- QUICK (pour les tables de type MyISAM) ne met pas à jour les index associés pour accélérer le traitement.
- La condition du WHERE sélectionne les lignes à supprimer dans la table. Si aucune condition n'est précisée, toutes les lignes seront détruites. Si la condition ne sélectionne aucune ligne, aucun enregistrement ne sera supprimé.
- ORDER BY réalise un tri des enregistrements qui seront effacés dans cet ordre.

Exemple : la première commande supprime tous les articles de catégorie informatique, la seconde détruit tous les clients de Bamako de la table clientBis dont le numéro de téléphone n'est pas NULL.

```
DELETE FROM article WHERE categorie='Informatique' ;

DELETE FROM clientBis WHERE ville='Bamako' and telephone IS NOT NULL ;
```

Détruisons enfin les 2 premiers clients de la table client :

```
DELETE FROM client LIMIT 2 ;
```

4. LE LANGAGE D'INTERROGATION DES DONNEES

Ce chapitre traite de l'aspect le plus connu du langage SQL qui concerne l'extraction des 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.

L'instruction SELECT est une commande déclarative (elle décrit ce que l'on cherche sans expliquer le moyen d'opérer). A l'inverse, une instruction procédurale (comme un programme) développerait le moyen pour réaliser l'extraction de données (comme le chemin à emprunter entre des tables ou une itération pour parcourir un ensemble d'enregistrements).

Pour pouvoir extraire des enregistrements d'une table, il faut que celle-ci soit dans votre base ou que vous ayez reçu le privilège SELECT sur la table.

```
SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes
        FROM nomTables1 [,nomTable2]…
        [WHERE condition]
        [clauseRegroupement]
        [HAVING condition]
        [clauseOrdonnacement]
        [LIMIT [rangDépart,] nbLignes] ;
```

a. PROJECTION [éléments du SELECT]

Étudions la partie de l'instruction SELECT qui permet de programmer l'opérateur de projection (surligné dans la syntaxe suivante).

```
SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes
        FROM nomTable [aliasTable]
         [clauseOrdonnacement]
        [LIMIT [rangDépart,] nbLignes] ;
```

- DISTINCT et DISTINCTROW jouent le même rôle, à savoir ne pas inclure les duplicatas.
- ALL prend en compte les duplicatas (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).
- FROM désigne la table (qui porte un alias ou non) à interroger.
- clauseOrdonnancement : tri sur une ou plusieurs colonnes ou expressions.
- LIMIT pour limiter le nombre de lignes après résultat.

Interrogeons la table suivante en utilisant chaque option.

Client

numcli	nom	prenom	adresse	cp	ville	Telephone
1	Diarra	Oumar	1 Rue Droite	30000	Segou	23104750
2	Coulibaly	Karim	7 Rue Titi	12000	Bamako	76483901
3	Coulibaly	Aminata	Avenue Travélé	12000	Kayes	NULL
4	Sidibé	Kadia	Rue Tombouctou	66000	Mopti	NULL
5	Touré	Sidiki	Rue Maridjé	30000	Bamako	66485423

Extraction de toutes les colonnes

mysql> SELECT * FROM client ;

numcli	nom	prenom	adresse	cp	ville	Telephone
1	Diarra	Oumar	1 Rue Droite	30000	Segou	23104750
2	Coulibaly	Karim	7 Rue Titi	12000	Bamako	76483901
3	Coulibaly	Aminata	Avenue Travélé	12000	Kayes	NULL
4	Sidibé	Kadia	Rue Tombouctou	66000	Mopti	NULL
5	Touré	Sidiki	Rue Maridjé	30000	Bamako	66485423

Extraction de certaines colonnes

La liste des colonnes à extraire se trouve dans la clause SELECT.

SELECT prenom,nom,numcli FROM client ;

prenom	nom	numcli
Oumar	Diarra	1
Karim	Coulibaly	2
Aminata	Coulibaly	3
Kadia	Sidibé	4
Sidiki	Touré	5

Alias

Les alias permettent de renommer des colonnes à l'affichage ou des tables dans la requête.
Les alias de colonnes sont utiles pour les calculs. L'utilisation de la directive AS est facultative.

SELECT prenom AS 'Prénom du client', nom AS 'Nom du client',
numcli 'Numéro du client' FROM client ;

Prénom du client	Nom du client	Numéro du client
Oumar	Diarra	1
Karim	Coulibaly	2
Aminata	Coulibaly	3
Kadia	Sidibé	4
Sidiki	Touré	5

SELECT aliasClient.prenom AS Prénom, aliasClient.nom nomCli
FROM client aliasClient ;

Prénom	nomCli
Oumar	Diarra
Karim	Coulibaly
Aminata	Coulibaly
Kadia	Sidibé
Sidiki	Touré

Duplicata

Les directives DISTINCT ET DISTINCTROW éliminent les éventuels duplicatas.

SELECT nom FROM client ;

Nom
Diarra
Coulibaly
Coulibaly
Sidibé
Touré

SELECT DISTINCT nom FROM client ;

Nom
Diarra
Coulibaly
Sidibé
Touré

Expressions simples

Il est possible d'évaluer et d'afficher simultanément des expressions dans la clause SELECT (types numériques, DATE et DATETIME).

Les opérateurs arithmétiques sont évalués par ordre de priorité (* , / , + et -). Le résultat d'une expression comportant un NULL est bien souvent un NULL.

Soit la table article contenant les données suivantes :

numart	designation	categorie	prix
1	Guimba	CD	2000
2	Sodia	DVD	3000
3	WebCam	Informatique	4000
4	Graveur	Informatique	9000
5	Clé Usb 16 Go	Informatique	2500
6	Befo	DVD	2000

SELECT numart, prix,prix*prix AS PrixAuCarré, prix*1.1 'Prix augmenté de 10%'
FROM article ;

numart	prix	PrixAuCarré	Prix augmenté de 10%
1	2000	4000000	2200
2	3000	9000000	3300
3	4000	16000000	4400
4	9000	81000000	9900
5	2500	6250000	2750
6	2000	4000000	2200

Ordonnancement

Pour trier le résultat d'une requête, il faut spécifier la clause d'ordonnancement par ORDER BY de la manière suivante :

ORDER BY
{ expression1 | position1 | alias1 } [ASC | DESC]
[, { expression2 | position2 | alias2 } [ASC | DESC]]

- Expression : nom de colonne, fonction SQL, constante, calcul.
- Position : entier qui désigne l'expression (au lieu de la nommer) dans son ordre d'apparition dans la clause SELECT.
- ASC ou DESC : tri ascendant ou descendant (par défaut ASC)

SELECT numcli,prenom FROM client
ORDER BY prenom ;

numcli	prenom
3	Aminata
4	Kadia
2	Karim
1	Oumar
5	Sidiki

SELECT numcli, prenom FROM client

ORDER BY prenom DESC ;

numcli	prenom
5	Sidiki
1	Oumar
2	Karim
4	Kadia
3	Aminata

Limitation du nombre de 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

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. Ainsi « LIMIT n » équivaut à « LIMIT 0,n ».

SELECT * FROM client LIMIT 1,2 ;

1 correspond au rang de la deuxième ligne (puisque la première ligne est considérée comme présente au rang 0) et 2 est le nombre de lignes à extraire à commencer par la deuxième ligne (rang 1).

numcli	nom	prenom	adresse	cp	ville	Telephone
2	Coulibaly	Karim	7 Rue Titi	12000	Bamako	76483901
3	Coulibaly	Aminata	Avenue Travélé	12000	Kayes	NULL

Les deux articles les plus chers

SELECT * FROM article
ORDER BY prix DESC LIMIT 2 ;

numart	designation	categorie	prix
4	Graveur	Informatique	9000
3	WebCam	Informatique	4000

b. RESTRICTION [WHERE]

Les éléments de la clause WHERE d'une requête permettent de programmer l'opérateur de restriction. Cette clause limite la recherche aux enregistrements qui respectent une condition simple ou complexe. Cette section s'intéresse à la partie surlignée de l'instruction SELECT suivante :

SELECT [{ DISTINCT | DISTINCTROW } | ALL] listeColonnes
FROM nomTable [aliasTable]
[WHERE condition] ;

- condition composée de colonnes, d'expressions, de constantes liées deux à deux entre des opérateurs :
  - de comparaison (>,=,<,>=,<=,<>) ;
  - logiques (NOT, AND ou OR)
  - intégrés (BETWEEN, IN, LIKE, IS NULL)

Interrogeons la table suivante en utilisant chaque catégorie d'opérateur :

numart	designation	categorie	prix
1	Guimba	CD	2000
2	Sodia	DVD	3000
3	WebCam	Informatique	4000
4	Graveur	Informatique	9000
5	Clé Usb 16 Go	Informatique	2500
6	Befo	DVD	2000

Opérateurs de comparaison

Donner la liste des articles dont le prix est 2000

SELECT * FROM article WHERE prix=2000 ;

numart	designation	categorie	prix
1	Guimba	CD	2000
6	Befo	DVD	2000

Donner la liste des articles caractérisés par le numéro article, la désignation et le prix et dont le prix est supérieur ou égal à 3000

SELECT numart,designation,prix FROM article
WHERE prix>=3000 ;

numart	designation	prix
2	Sodia	3000
3	WebCam	4000
4	Graveur	9000

Donner la liste des articles de catégorie informatique

SELECT * FROM article WHERE categorie='Informatique' ;

numart	designation	categorie	prix
3	WebCam	Informatique	4000
4	Graveur	Informatique	9000
5	Clé Usb 16 Go	Informatique	2500

REMARQUE : les écritures « prix=2000 » et « (prix=2000) » sont équivalentes. Les écritures « prix<>3000 », « NOT (prix=3000) » sont équivalentes. Les parenthèses sont utiles pour composer des conditions.

Opérateurs logiques

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.

La première requête contient une condition composée de trois prédicats qui sont évalués par ordre de priorité (d'abord AND puis OR). La conséquence est l'affichage des articles de catégorie 'DVD' avec les articles de catégorie 'Informatique' ayant un prix inférieur à 3000.

La deuxième requête force la priorité avec les parenthèses (AND et OR sur le même pied d'égalité). La conséquence est l'affichage des articles ayant un prix inférieur à 3000 appartenant aux catégories 'DVD' et 'Informatique'.

```
SELECT numart,designation,categorie,prix FROM article
        WHERE (categorie='DVD' OR categorie='Informatique' AND prix<3000) ;
```

numart	designation	categorie	prix
2	Sodia	DVD	3000
5	Clé Usb 16 Go	Informatique	2500
6	Befo	DVD	2000

```
SELECT numart,designation,categorie,prix FROM article
        WHERE (categorie='DVD' OR categorie='Informatique') AND prix<3000 ;
```

numart	designation	categorie	prix
5	Clé Usb 16 Go	Informatique	2500
6	Befo	DVD	2000

Opérateurs intégrés

Les opérateurs intégrés sont BETWEEN, IN, LIKE et IS NULL.

BETWEEN limiteinf AND limitesup

teste l'appartenance à un intervalle de valeurs.

```
SELECT numart, designation, prix FROM article
        WHERE prix BETWEEN 2500 AND 5000 ;
```

numart	designation	categorie	prix
2	Sodia	DVD	3000
3	WebCam	Informatique	4000
5	Clé Usb 16 Go	Informatique	2500

IN (listeValeurs)

Compare une expression avec une liste de valeurs

```
SELECT numcli, prenom, nom, ville FROM client
        WHERE ville IN ('Bamako' , 'Kayes') ;
```

numcli	prenom	nom	ville
2	Karim	Coulibaly	Bamako
3	Aminata	Coulibaly	Kayes
5	Sidiki	Touré	Bamako

LIKE (expression)

Compare de manière générique des chaînes de caractères à une expression. Le symbole « % » remplace zéro ou plusieurs caractères. Le symbole « _ » remplace un caractère. Ces symboles peuvent se combiner. Utilisez de préférence des colonnes VARCHAR ou complétez si nécessaire par des blancs jusqu'à la taille maximale pour des CHAR.

Donner la liste des clients dont la deuxième lettre du nom est « i ».

```
SELECT numcli, prenom,nom,ville FROM client
        WHERE nom LIKE ('_i%') ;
```

numcli	prenom	nom	ville
1	Oumar	Diarra	Segou
4	Kadia	Sidibé	Mopti

Donner la liste des clients dont le prénom contient la lettre « m »

```
SELECT numcli,prenom,nom,ville FROM client
        WHERE prenom LIKE ('%m%') ;
```

numcli	prenom	nom	ville
1	Oumar	Diarra	Segou
2	Karim	Coulibaly	Bamako
3	Aminata	Coulibaly	Kayes

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) ».

Donner la liste des clients dont le numéro de téléphone est NULL.

```
SELECT numcli, prenom, nom, telephone FROM client
        WHERE telephone IS NULL ;
```

numcli	prenom	nom	Telephone
3	Aminata	Coulibaly	NULL
4	Kadia	Sidibé	NULL

Donner la liste des clients dont le numéro de téléphone est différent de NULL.

```
SELECT numcli, prenom, nom, telephone FROM client
        WHERE telephone IS NOT NULL ;
```

On aurait pu écrire la même instruction différemment c.-à-d. de la manière suivante:

```
SELECT numcli, prenom, nom, telephone FROM client
        WHERE NOT (telephone IS NULL) ;
```

Et le résultat est le même

numcli	prenom	nom	Telephone
1	Oumar	Diarra	23104750
2	Karim	Coulibaly	76483901
5	Sidiki	Touré	66485423

Il n'est pas permis d'utiliser un alias de colonne dans la clause WHERE. Cette recommandation de la norme s'explique par le fait que certaines expressions pourraient ne pas être déterminées pendant que la condition WHERE est évaluée.

Regroupements

Cette section traite à la fois des regroupements de lignes (agrégats) et des fonctions de groupe (ou multilignes). Nous étudierons les parties surlignées de l'instruction SELECT suivante :

```
SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes
        FROM nomTable [WHERE condition]
        [clauseRegroupement]
        [HAVING condition]
        [clauseOrdonnacement]
        [LIMIT [rangDépart,] nbLignes] ;
```

- listeColonnes : peut inclure des expressions (présentes dans la clause de regroupement) ou des fonctions de groupe.
- 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).

Fonctions de groupe

Le tableau suivant présente les principales fonctions. L'option DISTINCT évite les duplicatas (sinon pris en compte par défaut). A l'exception de COUNT, toutes les fonctions ignorent les valeurs NULL.

Fonction	Objectif
AVG ([DISTINCT] expr)	Moyenne de expr (nombre)
COUNT ({ * | [DISTINCT] expr })	Nombre de lignes (* toutes les lignes, expr pour les colonnes non nulles)
MAX ([DISTINCT] expr)	Maximum de expr (nombre, date, chaîne)
MIN ([DISTINCT] expr)	Minimum de expr (nombre, date, chaîne)
SUM ([DISTINCT] expr)	Somme de expr (nombre)

Utilisés sans GROUP BY, ces fonctions s'appliquent à la totalité ou à une seule partie d'une table comme le montrent les exemples suivants :

Donner la moyenne des prix des articles

SELECT AVG(prix) 'Moyenne des prix' FROM article ;

Moyenne des prix
3750.0000

Donner le nombre de client

SELECT COUNT(*) 'Nombre de client' FROM client ;

Nombre de client
5

Donner la moyenne des prix des articles de la catégorie Informatique

SELECT AVG(prix) 'Moyenne prix cat informatique' FROM article
WHERE categorie='Informatique' ;

Moyenne prix cat informatique
5166.6667

Donner le nombre de client ayant des numéros de téléphone différent de NULL

SELECT COUNT(*) 'Nombre de client ayant des numéros de téléphone'
FROM client WHERE telephone IS NOT NULL;

Nombre de client ayant des numéros de téléphone
3

Donner le nombre de ville

SELECT COUNT(ville) 'Nombre de ville' FROM client ;

Nombre de ville
5

Donner le nombre de ville distincte

SELECT COUNT(DISTINCT ville) FROM client ;

Nombre de ville distincte
4

Donner le prix le plus élevé et le plus bas

SELECT MAX(prix) 'Prix le plus élévé', MIN(prix) 'Prix le plus bas'
FROM article ;

Prix le plus élévé	Prix le plus bas
9000	2000

Soit la table achat contenant les données suivantes :

numcli	numart	dateachat	Qte
1	1	2018-01-30	1
1	5	2018-01-30	4
4	2	2018-01-30	3
4	3	2018-01-29	1
5	2	2018-02-01	2

Donner la date d'achat la plus ancienne et la date d'achat la plus récente

```
SELECT MIN(dateachat) 'Date d'achat la plus ancienne',
       MAX(dateachat) 'Date d'achat la plus récente' FROM achat ;
```

Date d'achat la plus ancienne	Date d'achat la plus récente
2018-01-29	2018-02-01

Etude du GROUP BY et HAVING

Le groupement de lignes dans une requête se programme au niveau surligné de l'instruction SQL suivante :

```
SELECT col1 [, col2…], fonction1Groupe(…) [, fonction2Groupe(…) …]
       FROM nomTable [WHERE condition]
       [GROUP BY {col1 | expr1 | position1} [, {col2…} ]
       [HAVING condition]
       [ORDER BY {col1 | expr1 | position1} [ASC | DESC] [, {col1…} ] ] ;
```

- La clause WHERE de la requête permet d'exclure des lignes pour chaque groupement, ou de rejeter des groupements entiers. Elle s'applique donc à la totalité de la table.
- La clause GROUP BY liste les colonnes du groupement
- La clause HAVING permet de poser des conditions sur chaque groupement
- La clause ORDER BY permet de trier le résultat (déjà étudiée).

<u>REMARQUE</u> :

- Toutes les colonnes qui doivent être présentes dans le SELECT doivent se trouver dans le GROUP BY.
- Seules des fonctions d'agrégat peuvent être présentes dans le SELECT, en plus des colonnes du regroupement.
- Aucun alias de colonne ne peut être utilisé dans la clause GROUP BY.
- Les éventuels NULL seront regroupés dans le même ensemble de résultats (deux NULL sont considérés ici comme semblables).

Dans l'exemple suivant, en groupant sur la colonne categorie, trois ensembles de lignes (groupements) sont composés puisqu'il existe trois catégories d'articles dans la table (CD,DVD,Informatique). Il est alors possible d'appliquer des fonctions de groupe à chacun de ces ensembles (dont le nombre n'est pas précisé dans la requête ni limité par le système qui parcourt toute la table).

Il est aussi possible de grouper sur plusieurs colonnes.

Utilisés avec GROUP BY, les fonctions s'appliquent désormais à chaque regroupement, comme le montrent les exemples suivants :

Moyenne des prix par catégorie

```
SELECT categorie, AVG(prix) 'Moyenne des prix' FROM article
        GROUP BY categorie ;
```

categorie	Moyenne des prix
Cd	2000.0000
DVD	2500.0000
Informatique	5166.6667

Nombre d'articles par catégorie

```
SELECT categorie, COUNT(*) AS "Nombre d'article" FROM article
        GROUP BY categorie ;
```

categorie	Nombre d'article
Cd	1
DVD	2
Informatique	3

Prix le plus élevé par catégorie

```
SELECT categorie, MAX(prix) 'Prix le plus élevé' FROM article
        GROUP BY categorie;
```

categorie	Prix le plus élevé
Cd	2000
DVD	3000
Informatique	9000

Prix le plus élevé par catégorie et supérieur ou égal à 2500

```
SELECT categorie, MAX(prix) 'Prix le plus élevé et >=2500'
        FROM article GROUP BY categorie
        HAVING MAX(prix)>=2500;
```

categorie	Prix le plus élevé et >=2500
DVD	3000
Informatique	9000

Catégorie ayant moins de 3 articles

```
SELECT categorie, COUNT(*) 'Nombre d'articles ' FROM article
        GROUP BY categorie HAVING COUNT(*)<3;
```

categorie	Nombre d'article
Cd	1
DVD	2

Exemples concernant la table ACHAT

Nombre d'article acheté par client

```
SELECT numcli,COUNT(numart) "Nombre d'article acheté" FROM achat
        GROUP BY numcli;
```

numcli	Nombre d'article acheté
1	2
4	2
5	1

Nombre d'article acheté par client et par date d'achat

```
SELECT numcli, dateachat, COUNT(numart) 'Nombre article acheté'
        FROM achat GROUP BY numcli, dateachat;
```

numcli	dateachat	Nombre d'article acheté
1	2018-01-30	2
4	2018-01-29	1
4	2018-01-30	1
5	2018-02-01	1

c. JOINTURES

Les jointures permettent d'extraire des données issues de plusieurs tables. Le processus de normalisation du modèle relationnel est basé sur la décomposition et a pour conséquence d'augmenter le nombre de tables d'un schéma. Ainsi, la majorité des requêtes utilisent des jointures nécessaires pour pouvoir extraire des données de tables distinctes.

REMARQUE :

- Une jointure met en relation deux tables sur la base d'une clause de jointure (comparaison de colonnes). Généralement, cette comparaison fait intervenir une clé étrangère d'une table avec une clé primaire d'une autre table (car le modèle relationnel est fondamentalement basé sur les valeurs).

Classification

Une jointure peut s'écrire, dans une requête SQL, de différentes manières:

- « relationnelle » (aussi appelée « SQL89 » pour rappeler la version de la norme SQL) ;
- « SQL2 » (aussi appelée « SQL92 ») ;
- « procédurale » (qui qualifie la structure de la requête) ;
- « mixte » (combinaison des trois approches précédentes).

Nous allons principalement étudier les deux premières écritures qui sont les plus utilisées.

Jointure relationnelle

La forme la plus courante de la jointure est la jointure dite « relationnelle » (aussi appelée SQL89 caractérisée par une seule clause FROM contenant les tables et alias à mettre en jointure deux à deux. La syntaxe générale suivante décrit une jointure relationnelle :

```
SELECT [alias1.] col1, [alias2.] col2…
        FROM [nomBase.] nomTable1 [alias1], [nomBase.] nomTable2 [alias2]…
        WHERE (conditionsDeJointure) ;
```

Cette forme est la plus utilisée, car elle est la plus simple à écrire. Un autre avantage de ce type de jointure est qu'elle laisse le soin au SGBD d'établir la meilleure stratégie d'accès (choix du premier index à utiliser, puis du deuxième, etc.) pour optimiser les performances.

Afin d'éviter les ambiguïtés concernant le nom des colonnes, on utilise en général des alias de tables pour suffixer les tables dans la clause FROM et préfixer les colonnes dans les clauses SELECT et WHERE.

Jointures SQL2

Afin de se rendre conforme à la norme SQ2, MySQL propose aussi des directives qui permettent de programmer d'une manière plus verbale les différents types de jointures :

```
SELECT [ ALL | DISTINCT | DISTINCTROW ] listeColonnes
        FROM [nomBase.] nomTable1 [ { INNER | { LEFT | RIGHT } [OUTER] } ]
        JOIN [nomBase.] nomTable2 {ON condition | USING (colonne1 [, colonne2]…)}
        | { CROSS JOIN | NATURAL [ { LEFT | RIGHT } [OUTER] ]
                JOIN [nomBase.] nomTable2 }…
        [ WHERE condition ] ;
```

Cette écriture est moins utilisée que la syntaxe relationnelle. Bien que plus concise pour les jointures à deux tables, elle se complique pour des jointures plus complexes.

Types de jointures

Bien que, dans le vocabulaire courant, on ne parle que de « jointures » en fonction de la nature de l'opérateur utilisé dans la requête, de la clause de jointure et des tables concernées, on distingue :

- Les jointures internes (inner joins) :
    - L'équijointure (equi join) est la plus connue, elle utilise l'opérateur d'égalité dans la clause de jointure. La jointure naturelle est conditionnée en plus par le nom des colonnes. La non équijointure utilise l'opérateur d'inégalité dans la clause de jointure.
    - L'autojointure (self join) est un cas particulier de l'équijointure, qui met en œuvre deux fois la même table (des alias de tables permettront de distinguer les enregistrements entre eux).
    - L'inéquijointure met en relation des lignes de plusieurs tables sans utiliser aucune égalité entre clés primaires (ou candidates) et clés étrangères.
- La jointure externe (outer join) réalise à la fois une jointure interne et permet de relier des lignes sans correspondance. Cette opération favorise une table (dite « dominante ») par rapport à l'autre (dite « subordonnée »).

REMARQUE : pour mettre trois tables T1,T2 et T3 en jointure, il faut utiliser deux clauses de jointures (une entre T1 et T2 et l'autre entre T2 et T3). Pour n tables, il faut n-1 clauses de jointures. Si vous oubliez une clause de jointure, un produit cartésien restreint est généré.

Equijointure

Une équijointure utilise l'opérateur d'égalité dans la clause de jointure et compare généralement des clés primaires avec des clés étrangères.

Soient les tables suivantes :

Client

numcli	nom	prenom	adresse	cp	ville	Telephone
1	Diarra	Oumar	1 Rue Droite	30000	Segou	23104750
2	Coulibaly	Karim	7 Rue Titi	12000	Bamako	76483901
3	Coulibaly	Aminata	Avenue Travélé	12000	Kayes	NULL
4	Sidibé	Kadia	Rue Tombouctou	66000	Mopti	NULL
5	Touré	Sidiki	Rue Maridjé	30000	Bamako	66485423

Article

numart	designation	categorie	prix
1	Guimba	CD	2000
2	Sodia	DVD	3000
3	WebCam	Informatique	4000
4	Graveur	Informatique	9000
5	Clé Usb 16 Go	Informatique	2500
6	Befo	DVD	2000

Achat

numcli	numart	dateachat	Qte
1	1	2018-01-30	1
1	5	2018-01-30	4
4	2	2018-01-30	3
4	3	2018-01-29	1
5	2	2018-02-01	2

Donner l'identité des clients caractérisée par le prénom, le nom et la ville, ayant fait des achats, avec la date d'achat et la quantité achetée.

Ecriture « relationnelle »

```
SELECT prenom, nom, ville, dateachat, qte FROM client,achat
        WHERE client.numcli=achat.numcli ;
```

Ecriture « SQL2 »

```
SELECT prenom,nom,ville,dateachat,qte FROM client
        JOIN achat ON client.numcli=achat.numcli ;
```

prenom	nom	ville	dateachat	qte
Oumar	Diarra	Segou	2018-01-30	1
Oumar	Diarra	Segou	2018-01-30	4
Kadia	Sidibé	Mopti	2018-01-30	3
Kadia	Sidibé	Mopti	2018-01-29	1
Sidiki	Touré	Bamako	2018-02-01	2

Liste des clients caractérisée par prénom, nom et ville habitant Bamako et ayant acheté des articles

Ecriture « relationnelle »

```
SELECT prenom,nom,ville FROM client,achat
        WHERE client.numcli=achat.numcli
        AND ville='Bamako' ;
```

Ecriture « SQL2 »

```
SELECT prenom,nom,ville FROM client
        JOIN achat ON client.numcli=achat.numcli
        WHERE ville='Bamako' ;
```

prenom	nom	ville
Sidiki	Touré	Bamako

Liste des articles achetés en une quantité supérieure ou égale à 3 et caractérisé par la désignation, le prix, la quantité.

Ecriture « relationnelle »

```
SELECT designation,prix,qte FROM article,achat
        WHERE article.numart=achat.numart
        AND qte>=3 ;
```

Ecriture « SQL2 »

```
SELECT designation,prix,qte FROM article
        JOIN achat ON article.numart=achat.numart
        WHERE qte>=3 ;
```

designation	prix	qte
Sodia	3000	3
Clé Usb 16 Go	2500	4

Liste des articles achetés après le 31/01/2018 et caractérisé par désignation, prix et date achat.

Ecriture « relationnelle »

```
SELECT designation,prix,dateachat 'Date achat' FROM article,achat
        WHERE article.numart=achat.numart
        AND dateachat>'2018/01/31';
```

Ecriture « SQL2 »

```
SELECT designation,prix,dateachat 'Date achat' FROM article
        JOIN achat ON article.numart=achat.numart
        WHERE dateachat>'2018/01/31';
```

designation	prix	Date achat
Sodia	3000	2018-02-01

Liste des articles achetés caractérisé par désignation, date achat, prix, quantité et montant (montant=prix*qte).

Ecriture « relationnelle »

```
SELECT designation,dateachat 'Date achat',prix,qte,prix*qte 'Montant'
        FROM article,achat WHERE article.numart=achat.numart ;
```

Ecriture « SQL »

```
SELECT designation,dateachat 'Date achat',prix,qte,prix*qte 'Montant'
        FROM article JOIN achat ON article.numart=achat.numart ;
```

designation	Date achat	prix	qte	Montant
Guimba	2018-01-30	2000	1	2000
Clé Usb 16 Go	2018-01-30	2500	4	10000
Sodia	2018-01-30	3000	3	9000
WebCam	2018-01-29	4000	1	4000
Sodia	2018-02-01	3000	2	6000

5. EVOLUTION DE SCHEMA

L'évolution d'un schéma est un aspect très important à prendre en compte, car il répond aux besoins de maintenance des applicatifs qui utilisent la base de données. Nous verrons qu'il est possible de modifier une base de données d'un point de vue structurel (colonnes et index) mais aussi comportemental (contraintes).

L'instruction principalement utilisée est ALTER TABLE (commande du LDD) qui permet d'ajouter, de renommer, de modifier et de supprimer des colonnes d'une table. Elle permet aussi d'ajouter et de supprimer des contraintes.

a. RENOMMER UNE TABLE

ALTER TABLE [nomBase.]ancienNomTable RENAME TO [nomBase.]nouveauNomTable ;

Exemple :

ALTER TABLE article RENAME TO produit ;

Cette instruction renomme la table article et le nouveau nom est produit. Le nom article n'existe plus dans la base de données.

b. MODIFICATIONS STRUCTURELLES

Considérons la table suivante que nous allons faire évoluer.

CREATE TABLE Etudiant (matricule INT(3), prenom VARCHAR(15)) ;

Ajoutons 2 enregistrements à la table Etudiant.

INSERT INTO Etudiant VALUES (1,'Ousmane'),(2,'Kadia') ;

SELECT * FROM etudiant ;

matricule	prenom
1	Ousmane
2	Kadia

Ajout des colonnes

La directive ADD de l'instruction ALTER TABLE permet d'ajouter une nouvelle colonne à une table. Cette colonne est initialisée à NULL pour tous les enregistrements (à moins de spécifier une contrainte DEFAULT, auquel cas tous les enregistrements de la table sont mis à jour avec une valeur non nulle).

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 (dans le cas inverse, il faudra utiliser MODIFY à la place de ADD).

Le script suivant ajoute trois colonnes à la table etudiant. La première instruction ajoute la colonne nom en l'initialisant à NULL pour tous les étudiants. La deuxième commande ajoute

deux colonnes dont la première numeroclasse est initialisée à NULL et la deuxième ville est initialisée à une valeur non nulle. La colonne ville ne sera jamais nulle.

```
ALTER TABLE Etudiant ADD (nom VARCHAR (15)) ;

ALTER TABLE Etudiant
        ADD (numeroclasse CHAR(5),
        ville VARCHAR(20) DEFAULT 'Bamako' NOT NULL) ;
```

La table est désormais la suivante :

```
SELECT * FROM Etudiant ;
```

matricule	prenom	nom	numeroclasse	Ville
1	Ousmane	NULL	NULL	Bamako
2	Kadia	NULL	NULL	Bamako

Renommer des colonnes

Il faut utiliser l'option CHANGE de l'instruction ALTER TABLE pour renommer une colonne existante. Le nouveau nom de colonne ne doit pas être déjà utilisé dans la table. Le type (avec une éventuelle contrainte) doit être reprécisé. La position de la colonne peut aussi être modifiée en même temps. 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] ;
```

L'instruction suivante permet de renommer la colonne ville en adresse en la positionnant avant la colonne numeroclasse :

```
ALTER TABLE Etudiant CHANGE ville adresse VARCHAR(30) AFTER nom ;

SELECT * FROM Etudiant ;
```

matricule	prenom	nom	adresse	numeroclasse
1	Ousmane	NULL	Bamako	NULL
2	Kadia	NULL	Bamako	NULL

Modifier le type des colonnes

L'option MODIFY de l'instruction ALTER TABLE modifie le type d'une colonne existante sans pour autant la renommer. La syntaxe générale de cette instruction est la suivante, les options sont les mêmes que pour CHANGE :

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] ;

REMARQUE :

- Il est possible d'augmenter la taille d'une colonne numérique (largeur ou précision) ou d'une chaîne de caractère (CHAR et VARCHAR) ou de la diminuer si toutes les données présentes dans la colonne peuvent s'adapter à la nouvelle taille.
- Attention à ne pas réduire les colonnes indexées à une taille inférieure à celle déclarée lors de la création de l'index.
- Les contraintes en ligne peuvent aussi être modifiées par cette instruction. Une fois la colonne changée, les nouvelles contraintes s'appliqueront aux mises à jour ultérieures de la table, et les données présentes devront toutes vérifier cette nouvelle contrainte.

Mettons à jour la table Etudiant en actualisant la colonne numeroclasse (remplacement des NULL par la valeur « ig2 »).

UPDATE Etudiant SET numeroclasse='ig2' ;

SELECT * FROM Etudiant ;

matricule	prenom	nom	adresse	numeroclasse
1	Ousmane	NULL	Bamako	ig2
2	Kadia	NULL	Bamako	ig2

Augmentons la taille de la colonne numeroclasse et modifions son type de CHAR à VARCHAR.

ALTER TABLE Etudiant MODIFY numeroclasse VARCHAR(6) ;

Diminuons la taille de la colonne adresse et mettons comme valeur par défaut « Segou » au lieu de Bamako.

ALTER TABLE Etudiant MODIFY adresse VARCHAR(15) DEFAULT 'Segou' NOT NULL ;

Valeurs par défaut

L'option ALTER COLUMN de l'instruction ALTER TABLE modifie la valeur par défaut d'une colonne existante. La syntaxe générale de cette instruction et la suivante :

ALTER TABLE[nomBase.]nomTable ALTER [COLUMN] nomColonneAmodifier
{SET DEFAULT 'chaine' | DROP DEFAULT}

Le script suivant définit une valeur par défaut pour la colonne numeroclasse puis supprime celle relative à la colonne adresse.

ALTER TABLE Etudiant ALTER COLUMN numeroclasse SET DEFAULT 'ig2' ;

ALTER TABLE Etudiant ALTER COLUMN adresse DROP DEFAULT ;

Supprimer des colonnes

L'option DROP de l'instruction ALTER TABLE permet de supprimer une colonne (aussi un index ou une clé). La possibilité de supprimer une colonne évite aux administrateurs d'exporter les données, de recréer une nouvelle table, d'importer les données et de recréer les éventuels index et contraintes. Lorsqu'une colonne est supprimée, les index qui l'utilisent sont mis à jour voire éliminés si toutes les colonnes qui le composent sont éffacées. La syntaxe de ces options est la suivante :

```
ALTER TABLE [nomBase.]nomTable  DROP
        { [COLUMN] nomColonne | PRIMARY KEY
        | INDEX nomIndex | FOREIGN KEY nomContrainte }
```

REMARQUE : il n'est pas possible de supprimer avec cette instruction :

- Toutes les colonnes d'une table
- Les colonnes qui sont clés primaires (ou candidates par UNIQUE) référencées par des clés étrangères.

La suppression de la colonne adresse de la table Etudiant est programmée par l'instruction suivante :

```
ALTER TABLE Etudiant DROP COLUMN adresse ;
```

Voyons maintenant le contenu de la table.

```
SELECT * FROM Etudiant ;
```

matricule	prenom	nom	numeroclasse
1	Ousmane	NULL	ig2
2	Kadia	NULL	ig2

c. MODIFICATIONS COMPORTEMENTALES

Nous étudions dans cette section les mécanismes d'ajout, de suppression des contraintes.

Ajout de contraintes

Jusqu'à présent, nous avons créé les contraintes en même temps que les tables. Il est possible de créer des tables sans contraintes (dans ce cas l'ordre de génération n'est pas important), puis d'ajouter des contraintes.

La directive ADD CONSTRAINT de l'instruction ALTER TABLE permet d'ajouter une contrainte à une table. Il est aussi possible d'ajouter un index. La syntaxe générale est la suivante.

```
ALTER TABLE [nomBase.]nomTable ADD
        {INDEX [nomIndex] [typeIndex] (nomColonne1,...)
        | 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}]

Actualisons la colonne nom de la table Etudiant.

UPDATE Etudiant SET nom='Konipo' WHERE matricule=1 ;

UPDATE Etudiant SET nom='Koita' WHERE matricule=2 ;

SELECT * FROM Etudiant ;

matricule	prenom	nom	numeroclasse
1	Ousmane	Konipo	ig2
2	Kadia	Koita	ig2

A la table étudiant ajoutons une colonne « surnom ».

ALTER TABLE Etudiant ADD surnom VARCHAR(20) AFTER nom ;

SELECT * FROM etudiant ;

matricule	prenom	nom	Surnom	numeroclasse
1	Ousmane	Konipo	NULL	ig2
2	Kadia	Koita	NULL	ig2

Actualisons la colonne surnom.

UPDATE Etudiant SET surnom='Oussou' WHERE matricule=1 ;

UPDATE Etudiant SET surnom='Kady' WHERE matricule=2 ;

SELECT * FROM Etudiant ;

matricule	prenom	nom	Surnom	numeroclasse
1	Ousmane	Konipo	Oussou	ig2
2	Kadia	Koita	Kady	ig2

Unicité

Ajoutons la contrainte d'unicité portant sur la colonne « surnom ». Un index est automatiquement généré sur cette colonne à présent :

ALTER TABLE Etudiant ADD CONSTRAINT un_surnom UNIQUE (surnom) ;

La colonne surnom peut porter des valeurs nulles mais ses valeurs non nulles ne peuvent pas être répéter.

Clé primaire

Ajoutons à la table Etudiant sa clé primaire.

ALTER TABLE Etudiant ADD CONSTRAINT pk_Etud PRIMARY KEY (matricule) ;

Créons une nouvelle table « Classe » et ajoutons deux enregistrements.

CREATE TABLE Classe (codeClasse CHAR (4), libClasse VARCHAR(15) NOT NULL) ;

INSERT INTO Etudiant VALUES ('ig1','1ère année'),('ig2','2ème année') ;

SELECT * FROM Etudiant ;

codeclasse	libClasse
ig1	1ère année
ig2	2ème année

Ajoutons à la table Classe une clé primaire.

ALTER TABLE Classe ADD CONSTRAINT pk_Classe PRIMARY KEY (codeClasse) ;

Clé étrangère

Ajoutons la clé étrangère à la table Etudiant au niveau de la colonne numeroclasse.

ALTER TABLE Etudiant ADD CONSTRAINT fk_etud_classe
FOREIGN KEY (numeroclasse) REFERENCES Classe (codeClasse) ;

REMARQUE : 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.

Suppression des contraintes

Pour supprimer une de vos contraintes, vous devrez choisir parmi les directives suivantes : MODIFY, DROP INDEX, DROP FOREIGN KEY et DROP PRIMARY KEY.

Contrainte NOT NULL

Il faut utiliser la directive MODIFY de l'instruction ALTER TABLE pour supprimer une contrainte NOT NULL existant sur une colonne. Dans notre exemple, détruisons la contrainte NOT NULL de la colonne libClasse dans la table Classe.

ALTER TABLE Classe MODIFY libClasse VARCHAR(15) NULL ;

Contrainte UNIQUE

Il faut utiliser la directive DROP INDEX de l'instruction ALTER TABLE pour supprimer une contrainte d'unicité. Dans notre exemple, effaçons la contrainte portant sur le surnom de l'étudiant.

ALTER TABLE Etudiant DROP INDEX un_surnom ;

Clé étrangère

L'option DROP FOREIGN KEY de l'instruction ALTER TABLE permet de supprimer une clé étrangère d'une table. La syntaxe générale est la suivante :

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

Le nom de la contrainte est celui qui a été déclaré lors de la création de la table, soit lors de sa modification (dans CREATE TABLE ou ALTER TABLE).

REMARQUE : Si la contrainte a été définie sans être nommée, son nom est généré par le moteur InnoDB et l'instruction SHOW CREATE TABLE [nomBase.]nomTable permet de le découvrir.

Détruisons la clé étrangère de la colonne numeroclasse.

ALTER TABLE Etudiant DROP FOREIGN KEY fk_etud_classe ;

Clé primaire

L'option DROP PRIMARY KEY de l'instruction ALTER TABLE permet de supprimer une clé primaire. Dans des versions précédentes de MySQL, si aucune clé primaire n'existait, la première contrainte UNIQUE disparaissait à la place. Ce n'est plus le cas depuis la version 5.1.

REMARQUE : Si la colonne clé primaire à supprimer contient des clés étrangères, il faut d'abord retirer les contraintes de clé étrangère. Si la clé primaire à supprimer est référencée par des clés étrangères d'autres tables, il faut d'abord ôter les contraintes de clé étrangère de ces autres tables.

Ainsi pour supprimer la clé primaire de la table Classe, il faut d'abord enlever la contrainte de clé étrangère concernant la colonne numeroclasse de la table etudiant. Ce qui a été fait précédemment (ALTER TABLE Etudiant DROP FOREIGN KEY fk_etud_classe ;).

ALTER TABLE Classe DROP PRIMARY KEY;