UNIVERSITE PARIS XII – DESS MASERATI

INITIATION AUX BASE DE DONNEES RELATIONNELLES

LANGAGE SQL

Année 2004-2005

K. Tran-dai & O.Bretel

Avertissement : *les informations présentes dans ce support de cours ne sont pas exhaustives. Pour une documentation complète, il est recommandé de consulter le manuel utilisateur du moteur MYSQL (www.mysql.com).*

Introduction : Le langage SQL

Le langage SQL (Structured Query Language) peut être considéré comme le langage d'accès normalisé aux bases de données. Il est aujourd'hui supporté par la plupart des produits commerciaux que ce soit par les systèmes de gestion de bases de données micro tel que Access ou par les produits plus professionnels tels que Oracle ou Sybase. Il a fait l'objet de plusieurs **normes ANSI/ISO** dont la plus répandue aujourd'hui est la norme SQL2 qui a été définie en 1992. Nous décrivons ici les principaux aspects de cette norme.

Le succès du langage SQL est du essentiellement à sa simplicité et au fait qu'il s'appuie sur le schéma conceptuel pour énoncer des requêtes en laissant le SGBD responsable de la stratégie d'exécution. Le langage SQL propose un langage de requêtes ensembliste et assertionnel. Néanmoins, le langage SQL ne possède pas la puissance d'un langage de programmation : entrées/sorties, instructions conditionnelles, boucles et affectations. Pour certains traitements il est donc nécessaire de coupler le langage SQL avec un langage de programmation complet au sens **Turing** du terme.

Le langage SQL comporte :

- une partie sur la définition des données :
le **langage de définition des données (LDD)** qui permet de créer des tables, les modifier, définir des relations, des vues externes et des contraintes d'intégrité…

- une partie sur les requêtes :
le **langage de manipulation des données (LMD)** qui permet d'interroger une base de données sous forme déclarative sans se préoccuper de l'organisation physique des données…

- une partie sur le contrôle des données :
le **langage de contrôle des données (LCD)** qui permet de contrôler la sécurité et les accès aux données, droits d'accès, mode d'accès…

Une instruction SQL se définie comme un ensemble de mot clé du langage terminé par un « ; ». Le « ; » termine une instruction. La majorité des outils d'exécution SQL sont capable d'exécuter plusieurs commandes SQL séparées par un « ; » à la suite (notion de script SQL).

1. Le Langage de définition des données LDD :

Ces instructions SQL permettent la création d'une base de données, de tables, d'index, d'insérer et modifier les données.

1.1 Création d'une base de données :

Instruction SQL :

CREATE DATABASE [nom base] ;

Instruction SQL sur moteur MYSQL :

CREATE DATABASE (IF NOT EXISTS) [nom base]
DEFAULT CHARACTER SET [nom jeu de caractères] DEFAULT COLLATE [type de collation] ;

L'argument facultatif « IF NOT EXISTS » permet de créer une base seulement si celle-ci n'existe pas. (évite le message d'erreurs en cas d'existence d'une base du même nom).

Le CHARACTER SET : défini le jeu de caractères à utiliser sur cette base de données.
Le COLLATE : défini la méthode de comparaison entre jeu de caractères (permet de jouer sur la sensibilité à la casse par exemple).

1.2 Supprimer une base de données :

Instruction SQL :

DROP DATABASE [nom base] ;

Instruction SQL sur moteur MYSQL :

DROP DATABASE (IF EXISTS) [nom base] ;

L'argument facultatif "IF EXISTS" permet d'éviter le message d'erreur si la base de données n'existe pas.

NB : il n'existe pas d'instruction SQL qui permette de copier une base de données ou la renommer.

1.3 les types de données

Une base de données peut contenir différents types de données. Les types communs les plus courant entre les différents moteurs de base de données sont les suivants :

Types de données courants :

INTEGER[M] : nombre entier
VARCHAR[M] : chaîne de caractères
TEXT : chaîne de caractères longue
DECIMAL[M,D] : nombre décimal
DOUBLE[M] : nombre de la plus grande précision
DATE : date formatée
TIME : heure formatée

Le format de ces types dépend du moteur de base de données.

Types de données MYSQL (les plus courant) :

TINYINT [(M)]	8 bit entier
SMALLINT [(M)]	16 bit entier
INT [(M)]	32 bit entier
BIGINT [(M)]	64 bit entier
FLOAT [(M,D)]	32 bit réelle
DOUBLE [(M,D)]	64 bit réelle
DECIMAL [(M,D)]	réelle, format fixe
DATE	date, format aaaa-mm-jj
TIME	heure, format hh:mm:ss
DATETIME	date et heure, format aaaa-mm-jj hh:mm:ss
CHAR(L)	texte max. 255 octets, longueur fixe
VARCHAR(L)	texte max. 255 octets, longueur variable
TEXT	texte max 2^16-1 octets
MEDIUMTEXT	texte max 2^24-1 octets
LONGTEXT	texte max 2^32-1 octets

ENUM(<valeur1>, <valeur2>, …) Une valeur parmi les valeurs de la liste
SET(<valeur1>, <valeur2>, …) Zéro ou plusieurs valeurs parmi les valeurs de la liste

SET et ENUM : les valeurs sont représentées comme chaînes de caractères, mais stockées comme entiers. Les listes sont limitées à max 64 membres

M … taille d'affichage – total
D … taille d'affichage – chiffre après virgule
L … longueur max de la chaîne de caractère en octets

1.4 Création de tables

Instruction SQL :

CREATE TABLE [Nom Table] ([nom champ 1] [type champ 1] [Options],…..)

Les options que l'on peut définir pour un champ sont les suivantes :

PRIMARY KEY
Clef primaire, valeurs doubles et valeurs NULL interdites.
Si plusieurs colonnes font partie de la clef primaire, utiliser la propriété PRIMARY KEY([champ 1],[champ 2], …) de la table

NOT NULL
Valeur NULL interdite. Utilisation fortement recommandée

UNIQUE
Valeurs doubles interdites (valeur NULL permise)

DEFAULT <valeur>
Spécification de la valeur par défaut sur insertion de ligne

AUTO_INCREMENT
La valeur est incrémentée par défaut (uniquement pour les nombres)

Instruction sur le moteur MYSQL :

CREATE (TEMPORARY) TABLE (IF NOT EXISTS) [nom table]
(
[nom champ1] [type champ1] [options champ1],
… … … …,
[nom champN] [type champN] [options champN],
(DEFINITION DES CLES ET INDEX)
)
(OPTIONS DE LA TABLE)

Exemple d'options de la table :

TYPE = type de table (HEAP|ISAM|InnoDB|MERGE|MRG_MYISAM|MYISAM)
AUTO_INCREMENT = valeur de depart d'un champ autoincrement (1 par défaut)
COMMENT = 'description ou commentaire associé à la table'
MAX_ROWS = valeur nombre maximum de lignes pour la table
MIN_ROWS = valeur nombre minimum de lignes pour la table
ROW_FORMAT = format des lignes dans la table (DEFAULT | DYNAMIC | FIXED | COMPRESSED)
INSERT_METHOD = méthode d'insertion de lignes (NO | FIRST | LAST)
etc…

Les arguments facultatifs :

- TEMPORARY : Ce argument crée une table dite temporaire qui sera alors automatiquement effacée lorsque la connexion l'ayant créée se terminera.

- IF NOT EXISTS : cf. chap. 1.1

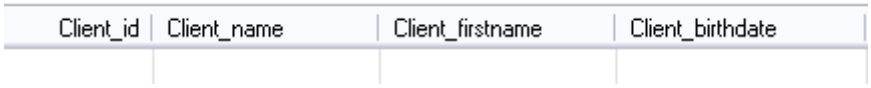
- DEFINITION CLES ET INDEX : permet de définir à la création les clés primaires et les index sur la table :

    - **PRIMARY KEY ([champ1], [champ2], …)**
    - **KEY [nom index] ([champ1], [champ2], …)**

Exemple :

```
CREATE TABLE CLIENT (
        Client_id INT(11) NOT NULL AUTO_INCREMENT,
        Client_name VARCHAR(30) NOT NULL,
        Client_firstname VARCHAR(30) NOT NULL,
        Client_birthdate DATE NOT NULL DEFAULT '1900-01-01',
        PRIMARY KEY (Client_id),
        KEY idx_23 (Client_name,Client_firstname)
) TYPE=MYISAM AUTO_INCREMENT=1000 COMMENT='table clients';
```

Table générée :



Lors de l'insertion d'une ligne la valeur automatique attribuée au premier client_id sera 1000 (option AUTO_INCREMENT=1000).

Le champ Client_id est défini comme clé primaire (client_id unique)
Un index est créé sur les champs Client_name, Client_firstname.
Cet index permet entre autre d'accélérer les recherches sur cette table par nom et prénom de client.

1.5 Supprimer une table

Cette instruction permet de supprimer une table (elle est alors effacée des disques durs des stockage)

Instruction SQL :

DROP TABLE [nom table] ;

Instruction MYSQL :

DROP TABLE (IF EXITS) [nom table1], [nom table2], … ;

Il est possible de supprimer plusieurs tables dans une même instruction. L'instruction facultative « IF EXISTS » permet d'éviter le message d'erreur dans le cas de non existence d'une ou plusieurs tables définies dans l'instruction DROP.

1.6 Modifier une table MYSQL

Les instructions suivantes permettent de modifier une table déjà créée.

1.6.1 Ajouter/Supprimer une colonne

Ajouter une colonne :

```
ALTER TABLE [nom table]
ADD COLUMN [nom col][type col][options col] [POSITION];
```

[POSITION] : définit la position de la nouvelle colonne par rapport aux existantes. Cette option peut prendre les valeurs suivantes :

. FIRST (en première position)
. AFTER [nom colonne] après la colonne spécifiée en [nom colonne]

Supprimer une colonne/un index/ une clé primaire :

```
ALTER TABLE [nom table]
DROP COLUMN [nom colonne]
Ou
DROP INDEX [nom index]
Ou
DROP PRIMARY KEY ;
```

1.6.2 Renommer une table

```
ALTER TABLE [nom table]
RENAME [nouveau nom table] ;
```

(Voir manuel MYSQL pour autres instructions possibles)

1.7 Ajouter un index à une table MYSQL existante

Les instructions suivantes permettent la création d'index sur les tables.

```
CREATE (TYPE) INDEX [nom index]
ON [nom table] ([nom champ1], [nom champ2]) ;
```

L'option TYPE permet de préciser si l'index est de type UNIQUE : si oui, la combinaison des valeurs de champs doit être unique. Autrement dit si deux lignes présentent le même contenu pour les colonnes indexées alors le Moteur MYSQL refusera l'insertion de la ligne, celle-ci étant considérée comme un doublon sur l'index.

Il est possible de créer des index sur une partie seulement de la colonne.

Exemple :

CREATE UNIQUE INDEX idx_13 ON CLIENT (client_name(10), client_firstname(10)) ;

L'index créé sera stocké comme la concaténation des 10 premiers caractères de la colonne **client_name** et les 10 premiers caractères de la colonne **client_firstname**.

Il est fortement recommandé de créer des index sur les colonnes utilisées dans les clauses WHERE (cf chapitre sur les requêtes SELECT) et notamment sur les colonnes de liaisons entre les tables. Le Moteur de base de données utilise alors les index les plus appropriés pour retourner les résultats de façon optimale.

1.8 L'insertion de données : les requêtes INSERT

Une fois les tables créées il faut les remplir ! Les instructions suivantes vous permettent d'insérer des lignes dans une table de différentes façons :

- par chargement de fichier
- par sélection sur autre table
- ligne par ligne

1.8.1 chargement d'un fichier délimité dans une table

Pour pouvoir être « chargée « en table, les données doivent tout de même être ordonnées dans le fichier. 2 caractères sont indispensables pour que le chargement se fasse correctement :

- le séparateur de colonne (« ; », « , », « | »…)
- l'indicateur de fin de ligne (retour chariot…)

Instruction MYSQL :

LOAD DATA (LOCAL) INFILE ‘’[chemin fichier]’’ (GESTION_DOUBLONS)
INTO TABLE [nom table]
FIELDS TERMINATED BY ‘[séparateur de colonne]’
LINES TERMINATED BY ‘[indicateur fin de ligne]’;

L'option (GESTION DOUBLONS) permet de préciser comment l'insertion doit gérer la présence de doublons entre la table destination et le fichier source. Les valeurs de l'option sont :

- REPLACE (remplacement de la ligne déjà dans la table)
- IGNORE (pas de mise à jour de la ligne)

L'option (LOCAL) permet de définir si le fichier à charger est situé sur le poste « local » d'ou est lancé l'instruction LOAD, ou si celui-ci se trouve sur le serveur de base de données.

Exemple :

```
LOAD DATA LOCAL INFILE ‘’c:\\mes données\\données.csv ‘’
INTO TABLE CLIENTS
FIELDS TERMINATED BY ‘ ;’
LINES TERMINATED BY ‘\n’;
```

‘\n’ défini le retour chariot.

1.8.2 insertion ligne par ligne

Cette commande permet d'insérer un seul enregistrement à la fois.

Instruction MYSQL :

INSERT INTO [nom table] ([col1],[col2], …,[colN])
VALUES([valeur_nombre], '[valeur_texte]' , …, '[valeur_date]')
(ON DUPLICATE KEY UPDATE [colX]=[Expression]);

Si l'on ne précise pas le nom des colonnes après le nom de la table, il est nécessaire de donner autant de valeurs dans la clause "VALUES" qu'il existe de colonnes dans la table.
La valeur à donner pour un champ de type AUTOINCREMENT est la valeur vide ''.

L'option facultative « ON DUPLICATE KEY UPDATE » permet de préciser quel action le moteur doit effectuer en cas d'existence dans la table de la ligne que l'on veut insérer. Cette option quels sont les colonnes à modifier et quels actions doivent être exécutées pour que le doublon disparaisse.

Exemple :

```
INSERT INTO TABLE CLIENT (client_name)
VALUES ('DURANT');
(instruction possible si les autres champs n'ont pas l'option "NOT NULL")

INSERT INTO TABLE CLIENT
VALUES ('','DURANT','GILLES','2004-10-01');

INSERT INTO TABLE CLIENT (id_client,client_name,client_firstname,client_birthdate)
VALUES ('','DURANT','GILLES','2004-10-01')
ON DUPLICATE KEY UPDATE client_name=CONCAT(client_name,id_client);
```

1.8.3 insertion par requite SELECT

Il est aussi possible d'insérer des données dans une table à partir des résultats d'une autre commande SQL. Une requête de type « SELECT » retourne un « tableau » de données qu'il est possible d'insérer dans une autre table en une seule commande.

Instruction MYSQL :

INSERT INTO [nom table] ([col1],[col2], …,[colN])
SELECT [col1],[col2], …,[colN] FROM [nom table];

Exemple :

```
INSERT INTO CLIENT (Client_name, Client_firstname, Client_birthdate)
SELECT Client_name, Client_firstname, Client_birthdate
FROM OLD_CLIENT ;
```

Cette instruction insert dans les colonnes Client_name, Client_firstname, Client_birthdate de la table CLIENT toutes les lignes des colonnes Client_name, Client_firstname, Client_birthdate de la table OLD_CLIENT.

1.9 Mise à jour des données, Modification, Suppression de données

Les commandes SQL présentées ci-après permettent de modifier des données déjà existantes dans les tables d'un base de données : ce sont les requêtes « UPDATE »

- Instruction MYSQL pour une seule table :

UPDATE (LOW_PRIORITY) [nom table]
SET [colX]=[expression 1], [colY]=[expression 2], …,[colN]=[expression N]
(WHERE [filtre sur les données])
(ORDER BY [col1],[col2], …,[colN])
(LIMIT [nombre de lignes à traiter])

L'option « LOW_PRIORITY » permet de retarder automatiquement l'exécution de la mise à jour des données au moment ou il n'y a plus aucune lecture en cours sur la table impactée.

La clause « WHERE [filtre sur les données] » permet de sélectionner quelles sont les données à mettre à jour sur la table. Sans cette clause, toutes les lignes de la table seront alors modifiées.

La clause « ORDER BY » permet de trier les données dans la table selon les colonnes indiquées (utile dans le cas de requête UPDATE avec restriction LIMIT)

L'option « LIMIT […] » permet de restreindre la mise à jour à une nombre de lignes spécifié par le paramètre [nombre de lignes à traiter).

Exemple :

```
UPDATE CLIENT
SET Client_name='DUPUIS'          #action à exécuter sur la colonne Client_name#
WHERE Client_name='DUPONT' #sélection des lignes à mettre à jour#
ORDER BY Client_id                #spécifie un tri sur la colonne Client_id#
LIMIT 2                           #restriction de la mise à jour aux 2 premières lignes#
```

Cette commande va modifier le nom des 2 premiers Client d'appelant 'DUPONT' en transformant leur nom en 'DUPUIS'.

Dans l'exemple nous trions les données par la colonne Client_id, qui comme il est un champ numérique de type auto_increment va trier les données selon leur ancienneté dans la table. En effet le numéro client_id est automatiquement incrémenté d'une unité supplémentaire à chaque insertion de ligne. Chaque ligne obtient donc un numéro unique qui augmente avec le nombre de lignes dans la table. Ainsi les numéros les plus faibles sont les premières lignes insérées dans cette table.

- Instruction MYSQL avec plusieurs tables :

UPDATE (LOW_PRIORITY) [nom table1] ,[nom table2],[nom tableN]…
SET [nom table1].[colX]=[expression 1],
[nom table2]. [colY]=[expression 2],
…
WHERE [nom table1].[liaisonA]=[nom table2].[liaisonA]
AND [nom table1].[liaisonX]=[nom tableN].liaisonX]
(AND [filtre sur les données])
(ORDER BY [nom table1].[col1], [nom table1]. [col2], …,[nom tableN].[colN])
(LIMIT [nombre de lignes à traiter])

Une requête UPDATE sur plusieurs tables permet d'utiliser des filtres complexes sur les données (utiliser les colonnes d'une autre table pour filtrer les données à mettre à jour sur la table), ou utiliser une colonne d'un autre table comme [expression] de mise à jour.

Pour supprimer des « enregistrements » (lignes) dans une table il faut utiliser des requêtes « DELETE ».

- Instruction MYSQL pour une seule table :

DELETE (LOW_PRIORITY) FROM [nom table]
(WHERE [filtre sur les données])
(ORDER BY [col1],[col2], …)
(LIMIT [nombre de lignes à effacer])

Les influences des différentes clauses (WHERE, ORDER BY, LIMIT) sont les mêmes que pour la commande « UPDATE »

Exemple :

DELETE FROM CLIENT #supprime toutes le lignes de la table CLIENT)#

DELETE FROM CLIENT
WHERE Client_id=1001 #supprime le client ayant l'id 1001#

- Instruction MYSQL sur plusieurs tables :

DELETE [nom table1].[*], …, [nom tableN].[*]
FROM [nom table1], …, [nom tableN]
WHERE [nom table1].[liaisonA]=[nom table2].[liaisonA]
AND [nom table1].[liaisonX]=[nom tableN].liaisonX]
(AND [filtre sur les données])
(ORDER BY [nom table1].[col1], [nom table1]. [col2], …,[nom tableN].[colN])
(LIMIT [nombre de lignes à traiter])

Cette instruction plus complexe permet de supprimer simultanément des lignes de différentes tables.

De plus amples explications seront fournies dans les chapitres suivants (CLAUSE WHERE et liaisons entre tables).

1.10 Les Autres Commandes MYSQL Utiles

- Sélectionner une base de données :

USE [nom base de données] ;

- Vider tout le contenu d'une TABLE :

TRUNCATE [nom table] ;

- Voir toutes les tables de la base de données :

SHOW TABLES (FROM [nom base de données]) (LIKE '%critère%') ;
Ou (plus détaillé)
SHOW TABLES STATUS FROM [nom base de données]) (LIKE '%critère%') ;

- Voir toutes les colonnes :

SHOW COLUMNS FROM [nom table] (LIKE '%critère%') ;

- Voir la liste des processus actifs sur le moteur de base de données :

SHOW PROCESSLIST

- Obtenir la syntaxe SQL qui a généré une table :

SHOW CREATE TABLE [nom table] ;

2. Le langage de manipulation des données (LMD)

L'ensemble des commandes présentées dans cette partie du cours permettent à l'utilisateur d'une base de données de l'interroger afin d'obtenir les données qu'il désire. Avant d'interroger une base de données, il convient de connaître sa structure et son mode de lecture afin d'être sur de lire correctement les informations contenues dans les tables. Les commandes « SHOW TABLES », « SHOW COLUMNS » sur le moteur MYSQL, donnent un aperçu souvent suffisant de l'architecture de la base de données.

2.1 La lecture des données : SELECT

Cette instruction permet d'interroger une base de données et donc de lire le contenu des tables.

- Instruction SQL générale :

```
SELECT [col1], …. , [colN]
FROM [nom table1], … , [nom table2]
WHERE [liaisons entre tables]
AND [filtres sur les données]
ORDER BY [tri dur les données] ;
```

La définition des colonnes à ramener n'est pas obligatoire il est possible de lire toutes les colonnes de la table en utilisant le caractère '*'.

```
SELECT *
FROM [nom table] ;
```

Cette requête retourne tout le contenu de la table [nom table], cad toutes les colonnes et toutes les lignes. Cette commande peut s'avérer dangereuse si vous n'avez pas la possibilité d'annuler votre commande SQL quand elle est en cours d'exécution (en effet retourner tout le contenu d'une table de plusieurs millions de lignes peut nécessité un certain temps d'exécution).

Le moteur MYSQL permet avec l'option LIMIT [ligne de début], [nombre de ligne] de limiter le nombre de lignes que le moteur doit retourner :

- Instruction MYSQL :

```
SELECT *
FROM [nom table]
LIMIT 1, 1000 ;
```

Cette instruction ne ramène alors que les 1000 premières lignes de la table, ce qui est largement suffisant pour analyser le contenu d'une table.

Une instruction utile pour connaître le nombre de ligne d'une table sans avoir à la parcourir en entier est la commande suivante :

```
SELECT count(*) FROM [nom table] ;
```

Il convient donc avant de manipuler une base de données de se faire une idée des volumétries de données continues dans les tables en utilisant cette instruction.

2.2 Les ALIAS en SQL

Les Alias permettent de renommer les objets d'une base de données dans une instruction SQL afin d'utiliser/obtenir des noms de colonnes plus pratiques ou plus explicites.

- Alias de colonne :

SELECT [col1] AS nom_alias
FROM [nom table] ;

Si l'alias utilise de contient pas de caractères spéciaux (espaces notamment) il n'est pas nécessaire de l'encapsuler avec les caractères d'encapsulation défini sur le moteur SQL ([] sur access, « » pour MYSQL et ORACLE). L'option « AS » est facultative.

Exemple :

SELECT Client_id "Identifiant Client", Client_id FROM CLIENT;
SELECT Client_id Identifiant_Client, Client_id FROM CLIENT;

Identifiant Client	Client_id
1002	1002
1003	1003
1004	1004

- Alias de table :

L'utilisation de ces alias est très utile pour les requêtes utilisant plusieurs tables, car elles évitent de répéter le nom de la table avant chaque nom de colonne.

SELECT *
FROM [nom table1] AS t1,
[nom table2] AS t2 …

Exemple :

SELECT *
FROM Client t1
WHERE t1.Client_id=1002 ;

Client_id	Client_name	Client_firstname	Client_birthdate
1002	DUPUIS	JEAN	1900-01-01

- Alias de requête :

Dans le cas de requêtes complexes imbriquées, il est parfois utile d'aliaser une requête afin de pouvoir utiliser ses résultats dans une autres requêtes. Cette fonctionnalité n'est pas supportée sur tout les moteurs de base de données.

2.3 LES CLAUSES WHERE

Ces clauses sont indispensables dans les requêtes de lecture « SELECT », mise à jour « UPDATE » et suppression « DELETE » pour :

- Sélectionner les lignes à lire, modifier ou supprimer
- Lier les tables entre elles

a. Clauses WHERE de filtrage :

L'instruction précise le début d'une série de conditions dans une requête SQL. La première condition suit le mot « WHERE », les autres s'ajoutent par le biais de l'instruction « AND » ou « OR ».

- Instruction SQL ;

SELECT * FROM [nom table]
WHERE [conditions1]
AND [condition2]
OR [condition4]
AND [condition]

Definition d'une condition :

Une condition utilise au moins une colonne des tables utilisées dans la requête. Cette condition utilise généralement l'un des cinq opérateurs de comparaison classique :

. = défini la correspondance complète
. <> ou (!= sur MYSQL) défini la différence
. > défini la supériorité
. < défini l'infériorité
. LIKE défini la correspondance partielle (pour la chaîne de caractère)
. BETWEEN entre deux valeurs
. IN (liste de valeurs) dans liste de valeurs

L'autre mot clés utiles dans les conditions :

. IS est
. IS NOT n'est pas
. NULL vide

Pour la correspondance partielle il convient d'utiliser un caractère spécial qui permet de définir la terminaison ou le début de la référence de correspondance partielle.

Lorsque l'on manipule des chaînes de caractères dans une conditions il convient d'encapsuler la chaîne avec des quottes ou guillemets (‘chaîne’ , « chaîne ») .

Les opérateurs mathématiques supportés sont : +, - , / , *

Exemple MYSQL :

- Retourne tout les client dont le nom est DUPUIS

```
SELECT * FROM Client
WHERE Client_name='DUPUIS'
```

- Retourne tout les clients dont le nom est DUPUIS et le Client_id est inférieur à 1010

```
SELECT * FROM Client
WHERE Client_name='DUPUIS'
AND Client_id < 1010
```

- Retourne tout les client dont le nom commence par DUP

```
SELECT * FROM Client
WHERE Client_name LIKE 'DUP%'
```

- Retourne tout les clients dont le nom se termine par UIS

```
SELECT * FROM Client
WHERE Client_name LIKE '%UIS'
```

- Retourne tout les client dont le nom est renseigné dans la table

```
SELECT * FROM Client
WHERE Client_name IS NOT NULL
```

- Retourne tout les client dont le nom est renseigné dans la table

```
SELECT * FROM Client
WHERE Client_name IS NULL
```

- Retourne tout les client dont le nom est DUPUIS ou DURAND

```
SELECT * FROM Client
WHERE Client_name='DUPUIS'
 OR Client_name='DURAND'
```

Il est aussi possible d'utiliser des colonnes comme référence dans les conditions. La référence est alors la valeur de la colonne ou une transformation de cette valeur :

- Retourne tout les client dont le nom est égal à leur prénom

SELECT * FROM Client
WHERE Client_name = Client_firstname

- Retourne toutes les commandes sur lesquelles il y a eu une erreur de calcul de TVA

SELECT * FROM Commande
WHERE Montant_TTC<>(Montant_HT+Montant_HT*0.196)

- Retourne tout les clients dont le nom est dans la liste définie :

SELECT * FROM Client
WHERE Client_name IN ('DURAND','DUPONT', …)

b. liaisons entre tables par Clause WHERE

Les clauses WHERE permettent aussi de lier les tables via les clés de liaisons définies dans la base de données.

Syntaxe SQL :

SELECT *
FROM [nom table1] AS t1, [nom table2] as t2
WHERE t1.[clé 1]=t2.[clé 1]
AND t1.[clé 2]=t2.[clé 2]
AND ….

Cette requête retourne toutes les lignes pour toutes les colonnes ou les tables ont une clé commune. Autrement dit seules les données qui ont un équivalent dans toutes les tables liées seront ramenées.

Faire abstraction des CLAUSES WHERE de liaisons revient à faire un « produit cartésien » : le moteur retournera pour chaque ligne de chaque table, l'ensemble de combinaisons de lignes des autres tables (à éviter !!).

SELECT *
FROM [nom table1], [nom table2], …

Nombre de lignes retournées :

Nombre de lignes T1 * nombre de lignes T2 * nombre de lignes TN

c. La liaison Externe

Une liaison externe permet de ramener toutes les lignes d'une table et NULL (si pas d'équivalence) ou ligne (si équivalence) de table liée. Ce point peut être illustré simplement en utilisant 2 tables (ce type de liaison est généralisable à N tables).

La liaison gauche :

SELECT *
FROM [nom table1] LEFT JOIN [nom table2]
ON [nom table1].[clé1]=[nom table2][clé1]
AND [nom table1].[cléN]=[nom table2].[cléN]
WHERE [conditions de filtre]

Cette instruction permet de ramener toutes les lignes de la table 1 et une ligne NULL si la table 2 n'a pas d'équivalent ou la ligne équivalente de la table 2.

La liaison droite :

SELECT *
FROM [nom table1] RIGHT JOIN [nom table2]
ON [nom table1].[clé1]=[nom table2][clé1]
AND [nom table1].[cléN]=[nom table2].[cléN]
WHERE [conditions de filtre]

Cette instruction permet de ramener toutes les lignes de la table 2 et une ligne NULL si la table 2 n'a pas d'équivalent ou la ligne équivalente de la table 1.

Pour les autres types de liaisons et options de jointure, se référer au manuel du moteur MYSQL.

2.4 Les opérations de regroupement

Les instructions SELECT vues précédemment retournent les données lignes par lignes. Le niveau de détail lu est celui enregistré dans les tables. Il existe toutefois des instructions qui permettent d'effectuer de calculs et d'obtenir des données « regroupées » différentes des données présentes dans les tables.

Ces opérations de « regroupement » sont (selon les moteurs de base de données):

- Min : valeur minimum de la colonne
- Count : nombre de
- Max : valeur maximum de la colonne
- Avg : moyenne de la colonne
- Std_dev : écart type de la colonne
- Sum : somme de la colonne
- Variance : variance de la colonne

Instruction SQL :

SELECT [col1],[col2],[colN], Opération[colX]
FROM [nom table1]
GROUP BY [col1],[col2],[colN]

Le regroupement de données s'effectue selon la clé UNIQUE des colonnes sans opérations (col1,col2,colN). C'est sur la base de cette clé UNIQUE que s'effectue l'opération demandée.

Exemple :

SELECT numéro_commande, max(article_montant_ttc) as montant_max
FROM commandes
GROUP BY numéro_commmande

Cette requête retourne, par exemple, le montant de l'article le plus cher présent dans chaque commande.
Instruction MYSQL :

Il n'est pas nécessaire de rappeler le nom des colonnes dans la clause GROUP BY sur le moteur MYSQL (le numéro de la colonne **dans** la requête est suffisant).

SELECT [col1],[col2], …,[colN], Opération[colX]
FROM [nom table]
GROUP BY 1,2, …, N

- La clause HAVING

Pour effectuer un filtre sur le résultat d'une requête GROUP BY, il convient d'utilise l'instruction « HAVING ». Il est alors possible de ramener uniquement les lignes dont l'opération répond à une condition.

Instruction SQL :

SELECT [col1],[col2], …,[colN], Opération[colX]
FROM [nom table]
GROUP BY 1,2, …, N
HAVING [Condition sur la colonne Opération[colX]]

Exemple :

SELECT numéro_commande, max(article_montant_ttc) as montant_max
FROM commandes
GROUP BY numéro_commmande
HAVING max(article_montant_ttc)>100

2.5 La gestion des doublons dans une requête SELECT

Il arrive fréquemment que la lecture d'une table sur un sous ensemble de ses colonnes engendre des doublons (plusieurs lignes identiques pour certains sous ensemble de colonnes). Pour ne pas remonter plusieurs fois les mêmes lignes il existe l'option suivante

- DISTINCT : chaque ligne doit être différente de toutes les autres

- Instruction SQL :

SELECT DISTINCT [col1], … , [colN] FROM [nom table1]

Ainsi chaque ligne retournée par cette requête sera unique (tout les doublons de la table sur les colonnes sélectionnées ne sont présentés qu'une fois dans le résultat).

2.6 Les requêtes Imbriquées

Il est possible d'utiliser des requêtes imbriquées en SQL. Cela permet entre autre d'utiliser le contenu du résultat d'une première requête dans une autre requête.

a. Utilisation d'une sous requête comme une table de données :

Cette instruction permet d'utiliser une table « virtuelle » créée à partir d'une première requête et d'utiliser son contenu pour effectuer une autre requête.

- Instruction SQL :

SELECT [nom Alias de requête1].[col1], …, [nom Alias de requête1].[colN]
FROM
(SELECT [col1],[col2], …, [colN]
FROM [nom table1], …., [nom tableN]
WHERE [Conditions requête1]) AS [nom Alias de requête1]
WHERE [Conditions requête2]
…

La requête est alors exécutée en 2 temps :

- **(SELECT [col1],[col2], …, [colN]**
FROM [nom table1], …., [nom tableN]
WHERE [Conditions requête1]) AS [nom Alias de requête1]

Exécution de la première requête et génération du résultat.

- **SELECT [nom Alias de requête1].[col1], …, [nom Alias de requête1].[colN]**
FROM
[Résultat Requête1]
WHERE [Conditions requête2]

Exécution de la seconde requête sur le résultat de la première requête.

b. Utilisation d'une sous requête dans une conditions de filtre :

Cette type de requête permet d'effectuer une sélection de lignes dans une table dont le résultat sera utilisé comme condition d'une autre requête.

- Instruction SQL :

```
SELECT * FROM [nomtable2]
WHERE [nomtable2].[colX] IN (SELECT [nomtable1].[colY] FROM [nomtable1]
WHERE …. GROUP BY…)
```

Ainsi nous obtenons toutes les lignes de la table [nomtable2] dont la valeur de la colonne X est dans liste créée par les valeurs de la colonne Y de la table [nomtable1].

c. Concaténer le résultat de deux requêtes en une seule :

Cet instruction permet de retourner dans un seul résultat le résultat de plusieurs requêtes SQL.

- Instruction SQL :

```
(SELECT * FROM [nomtable1])
UNION
(SELECT * FROM [nomtable2])
UNION
(SELECT …..)
```

Tout les résultats de chaque requête sont alors introduit les uns à la suite des autres dans un seul résultat. Pour pouvoir exécuter ce type d'instruction, chaque requête doit contenir le même nombre de colonnes (de même type).

d. Utilisation de sous requête dans les requêtes LDD

- Création d'une table à partir d'une requête SQL :

```
CREATE TABLE [nom nouvelle table]
SELECT [col1], …[colN]
FROM [nomtable1]
WHERE [conditions]
GROUP BY [agrégats]
HAVING [conditions sur agrégats]
ORDER BY [tri]
```

Une nouvelle table est alors physiquement créée à partir des résultats de la requête SELECT. Cette nouvelle table reproduit la structure des données de la requête SELECT. Elle ne comportera par contre aucun index ni clé primaire.

2.7 L'export de données à partir d'instruction SELECT

Cette instruction est le complément de l'instruction « LOAD DATA INFILE ». Elle permet de générer un fichier texte formaté à partir du résultat d'une requête SELECT.

Instruction MYSQL :

SELECT * FROM [nom table1]
WHERE [conditions]
GROUP BY ….
INTO OUTFILE "[chemin et nom du fichier]"
FIELDS TERMINATED BY [séparateur de colonnes]
LINES TERMINATED BY [caractères fin de lignes]

Exemple :

SELECT * FROM Client
INTO OUTFILE « c:\\export_client.txt »
FIELDS TERMINATED BY ";"
LINES TERMINATED BY "\n"

3. Les fonctions du moteur MYSQL

Afin de faciliter la manipulation des données contenues dans les colonnes des tables (format, recodage), chaque moteur de base de données possède des fonctions de traitement des valeurs.

3.1 la Manipulation des dates

- **NOW()** : retourne la date et l'heure actuelles
- **DATE()** : retourne la date actuelle
- **TIME()** : retourne l'heure actuelle

- **DATE_ADD** : permet d'ajouter une intervalle de temps à une date

Syntaxe : DATE_ADD ([valeur date], INTERVAL [valeur ajout] [Unité])

[valeur date] peut être soit une colonne, soit une fonction (date() par exemple), soit une constante ('2005-01-01' par exemple).

[valeur ajout] valeur de temps à ajouter.

[unité] : DAY, MONTH, YEAR, HOUR,SECONDE, DAY_SECOND, DAY_HOUR ….

Exemple : SELECT DATE_ADD(NOW(), INTERVAL 60 DAY)
→ retourne maintenant + 60 jours

- **DATE_SUB** : soustrait un intervalle de temps à une date. (même syntaxe que DATA_ADD)

- **DATE_FORMAT** : permet de formater une date

Syntaxe : DATE_FORMAT([valeur date], «format de date voulu »)

Syntaxe	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, ...)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)

%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week
%u	Week (00..53), where Monday is the first day of the week
%V	Week (01..53), where Sunday is the first day of the week; used with %X
%v	Week (01..53), where Monday is the first day of the week; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric, two digits

Exemple :

DATE_FORMAT(‘'1997-10-04’, '%W %M %Y')

Retourne → 'Saturday October 1997'

- **DAY([date])** : Retourne le jour du mois de la date
- **DAYNAME([date])** : Retourne le jour en format semaine (lundi, mardi etc..)
- **DAYOFWEEK([date])** : numéro du jour de la semaine (dimanche=1)
- **DAYOFYEAR([date])** : numéro du jour de l’année
- **DATEDIFF(date1,date2)** : retourne l’intervalle de temps entre 2 dates
- **TIMEDIFF(time1, time2)** : retourne l’intervalle en heure entre 2 dates
- **TIME_TO_SEC([heure])** : conversion du temps en secondes
- **SEC_TO_TIME([secondes])** : conversion des secondes en temps hh :mm :ss
- **YEARWEEK([date])** : retourne le numéro de la semaine dans l’année

Voir le manuel MYSQL pour les autres fonctions de gestion des DATES.

3.2 les fonctions mathématiques

Ces fonctions s'appliquent à des nombres, elles permettent de calculer des transformations sur les données directement sur le moteur de base de données.

- **ABS([nombre])** : valeur absolue
- **ROUND([nombre],[nombre de décimales])** : arrondi de la valeur
- **COS([nombre])** : cosinus du nombre
- **TRUNCATE([nombre],[nombre de chiffres])** : tronque un nombre
- **SQRT([nombre positif])** : racine carré du nombre
- **SIN([nombre])** : retourne le sinue du nombre

Voir le manuel MYSQL pour les autres fonctions mathématiques.

Exemple :

```
SELECT (COS([col1])*ROUND((SIN([col2])),3))/ (SQRT(ABS([col3]))) AS Res
FROM [nom table]
```

3.3 Les fonctions pour chaînes de caractères

Ces fonctions permettent de manipuler des chaînes de caractères (champ varchar, TEXT…)

- **CONCAT([expr1],[expr2], …, [exprN])** : concatène les différentes expressions

Exemple :

CONCAT(« 2005 », «-« , « 02 », «-« , « 01 ») Retourne : « 2005-02-01 »
CONCAT(client_name, client_firstname) retourne : DUPONTPierre

- **LENGTH([expr])** : retourne le nombre de caractères de l'expression [expr]

Exemple :

SELECT LENGTH(« DUPONT») → Retourne 6

- **SUBSTRING([expr],[debut],[longueur])** : retourne la sous chaîne de [expr] commençant au caractère positionné en [début] de longueur [longueur].

Exemple :

SELECT SUBSTRING(« DUPONT », 1 , 3) → Retourne « DUP »

- **LEFT([expr],[longueur]) et RIGHT([expr],[longueur])** : Permet de sélectionner la partie gauche ou la partie droite d'une chaîne de caractères d'une longueur [longueur]

- **LTRIM([expr]) et RTRIM([expr])** : Supprime les espaces en début de chaîne (LTRIM) ou en fin de chaîne de caractère (RTRIM)

- **REPLACE([expr],[search str],[replace str])** : Permet de substituer des expressions par une autre dans une chaîne de caractères.

Exemple :

SELECT REPLACE(« bonjour pierre », « bonjour », « bonne nuit »)

→ retourne « bonne nuit pierre »

- **LOCATE([search str],[expr])** : retourne la position de la [search str] dans [expr].

Exemple :

SELECT LOCATE(« pie », « bonjour pierre ») → retourne 9

Voir le manuel MYSQL pour les autres fonctions pour chaînes de caractères.