

MySQL 5 - SQL



Sommaire

Chapitre 1 - INTRODUCTION.....	7
1.1 - Prémisses.....	8
1.1.1 - Quelques icônes utilisées.....	8
1.1.2 - Les contacts de mon smartphone Samsung !.....	9
1.1.3 - Ma vidéothèque.....	11
1.2 - Définition d'un SGBDR.....	12
1.3 - Architecture du système applicatif.....	13
1.4 - SQL.....	14
1.5 - Les normes SQL.....	16
1.6 - BDs utilisées.....	17
Chapitre 2 - INSTALLATION et CONFIGURATION.....	18
2.1 - Stratégies et fichiers à télécharger.....	19
2.2 - Installations.....	20
2.2.1 - Installation du SGBDR.....	20
2.2.2 - Installation des outils-clients.....	21
2.3 - Travailler avec MySQL Query Browser.....	26
2.4 - Travailler avec Workbench.....	26
2.5 - Travailler avec phpMyAdmin.....	26
2.6 - Travailler avec TOAD FOR MySQL.....	26
2.7 - Travailler avec l'utilitaire mysql.....	27
Chapitre 3 - LES OBJETS MySQL.....	29
3.1 - Les bases de données.....	30
3.1.1 - Création.....	30
3.1.2 - Modification d'une base de données.....	31
3.1.3 - Suppression d'une base de données.....	31
3.2 - Les tables.....	32
3.2.1 - Les types de données.....	32
3.2.2 - Création d'une table.....	36
3.2.3 - Création d'une table à partir d'une autre table.....	39
3.2.4 - Renommage d'une table.....	39
3.2.5 - Suppression d'une table.....	39
3.2.6 - Modification de la structure d'une table.....	40
3.3 - Les contraintes (Constraint).....	43
3.3.1 - Création d'une contrainte.....	43
3.3.2 - Le cas particulier des clés étrangères.....	45
3.3.3 - Suppression d'une contrainte.....	47

3.3.4 - Liste des contraintes.....	47
3.4 - Les index (Index).....	48
3.4.1 - Création d'un index.....	48
3.4.2 - Suppression d'un index.....	49
3.4.3 - Visualisation des Index.....	50
3.4.4 - Utilisation des index.....	52
Chapitre 4 - MISE A JOUR DES DONNEES.....	53
4.1 - Présentation.....	54
4.2 - Insertion de données (INSERT).....	55
4.3 - Insertion de données (REPLACE).....	58
4.4 - Suppression de données (DELETE).....	59
4.5 - Suppression de toutes les données (TRUNCATE).....	60
4.6 - Modification de données (UPDATE).....	61
Chapitre 5 - L'EXTRACTION DES DONNEES (L'ORDRE SELECT).....	62
5.1 - SELECT mono table sans condition.....	63
5.2 - Extraire un certain nombre d'enregistrements (Restriction).....	64
5.3 - Le tri.....	65
5.4 - Les opérateurs de comparaison.....	66
5.5 - Les opérateurs logiques.....	67
5.6 - Les opérateurs ensemblistes.....	69
5.6.1 - IN.....	70
5.6.2 - BETWEEN.....	71
5.6.3 - LIKE.....	72
5.6.4 - IS NULL – IS NOT NULL.....	73
5.7 - Les requêtes calculées.....	74
5.7.1 - Numériques.....	74
5.7.2 - Chaînes.....	75
5.8 - Quelques fonctions propriétaires sur les chaînes.....	76
5.9 - Autres fonctions sur les chaînes.....	79
5.10 - Quelques fonctions propriétaires sur les numériques.....	82
5.11 - Quelques fonctions propriétaires sur les dates.....	83
5.12 - Autres fonctions.....	87
5.12.1 - IF.....	87
5.12.2 - IFNULL.....	88
5.12.3 - WHERE MATCH.....	89
Chapitre 6 - LES VUES (VIEWS).....	90
6.1 - Création, suppression, modification d'une view.....	91
6.1.1 - Création d'une view.....	92
6.1.2 - Suppression d'une view.....	92
6.1.3 - Modification d'une view.....	92
6.1.4 - Stockage.....	93
6.1.5 - Les vues matérialisées.....	94
6.2 - Les vues et les mises à jour.....	95

Chapitre 7 - LES JOINTURES.....	97
7.1 - Le produit cartésien et la jointure.....	98
7.2 - La conception d'une jointure.....	99
7.3 - L'equi-jointure.....	102
7.3.1 - Syntaxe simplifiée.....	102
7.3.2 - Syntaxe ANSI.....	103
7.4 - La jointure naturelle.....	104
7.5 - Auto-jointure.....	105
7.6 - Auto-jointure et généalogie.....	107
7.7 - Les jointures externes.....	111
7.8 - Plus loin avec la syntaxe ANSI.....	114
7.9 - Le produit cartésien.....	115
7.9.1 - Définition.....	115
7.9.2 - Premier exemple.....	116
7.9.3 - Deuxième exemple.....	117
Chapitre 8 - LES FONCTIONS AGREGATS.....	118
8.1 - Les fonctions agrégats.....	119
8.2 - La clause GROUP BY.....	121
8.3 - La clause HAVING.....	124
8.4 - La clause WITH ROLLUP.....	125
8.5 - GROUP_CONCAT.....	127
Chapitre 9 - LES REQUETES ENSEMBLISTES.....	130
9.1 - Principes.....	131
9.2 - Union.....	133
9.3 - Intersection (N'existe pas en MySQL).....	134
9.4 - Différence (N'existe pas en MySQL).....	135
Chapitre 10 - LES REQUETES IMBRIQUEES.....	138
10.1 - Présentation.....	139
10.2 - Format 1 : la sous-requête renvoie un seul résultat.....	140
10.3 - Format 2 : la sous-requête renvoie plusieurs résultats.....	145
10.4 - La requête renvoie vrai ou faux.....	147
10.5 - EXISTS et la mise en place des contraintes.....	148
10.6 - La sous requête renvoie un agrégat.....	153
10.7 - Les opérateurs ANY, ALL.....	154
10.8 - Mises à jour en fonction d'une sous-requête.....	159
10.8.1 - Insertion.....	159
10.8.2 - Suppression.....	160
10.8.3 - Modification.....	161
Chapitre 11 - LES REQUETES CORRELEES.....	162
11.1 - Présentation.....	163
11.2 - Epuisement de la corrélation !!!.....	166
11.3 - EXISTS et NOT EXISTS corrélés en substitution d'une jointure externe.....	170







11.4 - UPDATE et corrélation.....	172
11.5 - DELETE et corrélation.....	173
Chapitre 12 - LES TABLEAUX CROISES DYNAMIQUES.....	174
12.1 - Préparation.....	175
12.2 - TCD sur une table.....	178
12.3 - TCD sur une jointure statique.....	180
12.4 - TCD sur une jointure dynamique.....	182
Chapitre 13 - LES TRANSACTIONS.....	183
13.1 - Principes.....	184
13.2 - L'état de la gestion des transactions.....	185
13.3 - Validation.....	185
13.4 - Annulation.....	185
13.5 - Les savepoints.....	188
13.6 - Le verrouillage de table.....	189
13.7 - Le verrouillage de ligne.....	190
Chapitre 14 - LES TYPES SQL3.....	191
14.1 - Les blobs.....	192
Chapitre 15 - DIVERS.....	195
15.1 - Les événements (> 5.1).....	196
15.2 - Quelques éléments meta-basiques.....	198
15.2.1 - La commande SHOW.....	198
15.2.2 - La commande DESC.....	199
Chapitre 16 - ANNEXES.....	200
16.1 - Bibliographie.....	201
16.2 - Documentation (site officiel).....	202
16.3 - Critiques de MySQL.....	202
16.4 - A retenir ... absolument.....	203
16.5 - La BD COURS.....	204
16.6 - La BD cours_reduit_2014.....	205
16.7 - La BD Ingénieurs.....	206
16.8 - La BD Ingénieurs Light.....	207
16.9 - Les Charsets et les collations.....	208
16.10 - Les colonnes de type texte binaire.....	212
16.11 - MySQL et les expressions régulières.....	214
16.12 - IF.....	215
16.13 - CASE WHEN.....	217
16.14 - FIELD, ELT FIND_IN_SET(), SUBSTRING_INDEX() & co.....	219
16.15 - WITH RECURSIVE.....	222
16.16 - Les espaces intervallaires.....	223
16.17 - Exploder une colonne.....	228
16.18 - La division en SQL.....	229
16.18.1 - Définition de la division en algèbre relationnelle.....	229
16.18.2 - La division en SQL - exemple.....	230

16.18.3 - Autre exemple.....	234
16.19 - Les tables MERGE.....	237
16.20 - Les grands nombres.....	241
16.21 - La table des communes de France.....	242
16.22 - Importation/Exportation au format CSV.....	243
16.23 - Jeux de caractères et unicité des valeurs.....	244
16.24 - Accéder à distance à un serveur MySQL.....	245

CHAPITRE 1 - INTRODUCTION

1.1 - PRÉMISSSES

1.1.1 - Quelques icônes utilisées

A connaître par cœur	
Obsolète	 ou  ou  ou 
A retenir	

1.1.2 - Les contacts de mon smartphone Samsung !

Nom	Prénom	Tél fixe	Tél mobile	Tél pro	E-mail	E-mail pro	Photo	Type
Epouse		01 ...	06 ...	01 ...				Famille
Fils1								Famille
Fille1								Famille
Fille2								Famille
Copain1								Amis
Copain2								Amis
Comptable								Pro
Etc								

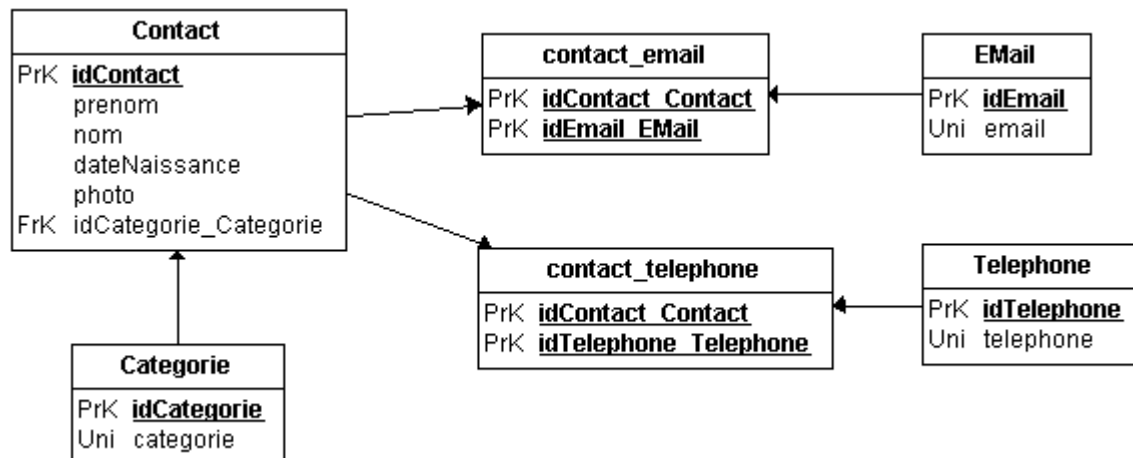
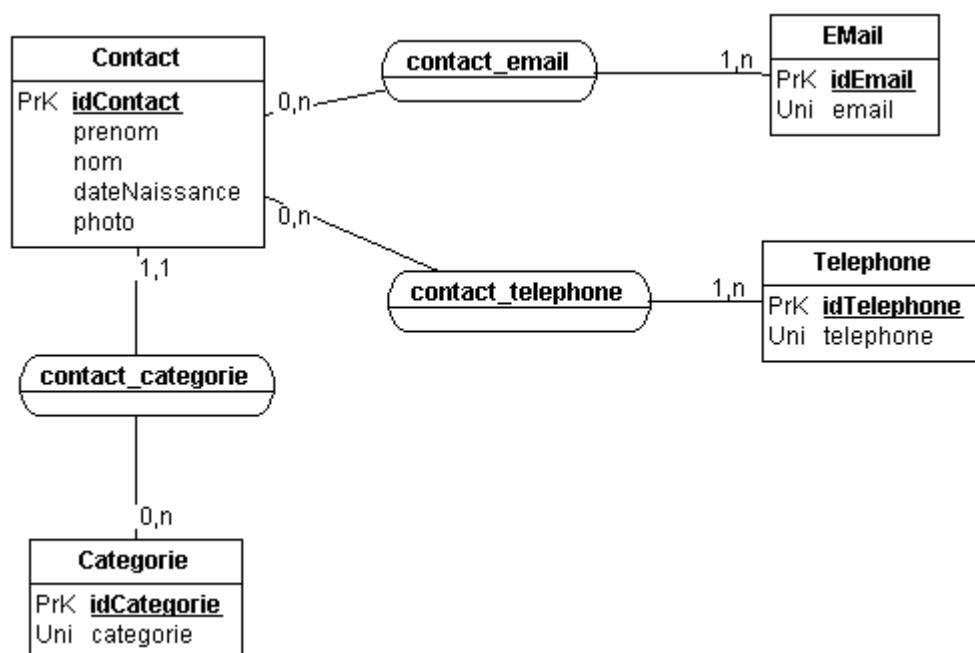
Dans le monde fichier (TXT, CSV, Excel, ...) les données sont stockées dans un seul fichier.
 Dans le monde des SGBD(R) les données seraient stockées dans une seule table.

Avec cette structure-ci :

Contacts
Nom Prénom TelFixe TelMobile TelPro Email EmailPro Photo Type

Que se passe-t-il si ma fille achète un 2^{ème} téléphone portable ?
 Que se passe-t-il si mon comptable dispose de 2 lignes fixes ?
 Que se passe-t-il si un copain est joignable avec 3 e-mails ?

Un contact appartient à une seule catégorie,
 Un contact peut avoir plusieurs numéros de téléphone,
 Un contact peut avoir plusieurs e-mails.



1.1.3 - Ma vidéothèque

Bof, bof, bof !

Titre	Titre original	Acteurs	Réalisateurs	Année de production	Producteur	Distributeur	Support	Pays	Genre

Passage à la BD.

Cf les exercices vidéothèque.

1.2 - DÉFINITION D'UN SGBDR



Un **SGBDR** est un **Système de Gestion de Bases de Données Relationnelles**. C'est-à-dire un ensemble de logiciels capable de gérer une base de données relationnelle.

Une **BDR** (base de données relationnelle) est un ensemble de **tables** bien souvent reliées entre elles (il peut exister des tables paramètre indépendantes) qui modélisent un domaine du SI d'une organisation.

Une table est composée de **colonnes** et de **lignes**.

Une table doit posséder une **clé primaire**, composée d'une ou plusieurs colonnes; celle-ci permet d'identifier chaque ligne. Chaque valeur est unique et doit être renseignée (elle est NOT NULL). La clé primaire est indexée.

Une table peut comprendre zéro, une ou plusieurs **clé(s) étrangère(s)** qui sont des colonnes correspondant à une clé primaire dans une autre table. Une clé étrangère dans une table (enfant) permet de faire un lien vers une autre table (une table parent). Mais c'est surtout une contrainte, dans la mesure où les valeurs de la clé primaire de la table parent sont références pour la table enfant. On parle de **contrainte d'intégrité référentielle**. Une valeur dans la colonne de la clé étrangère de la table enfant doit nécessairement correspondre à une valeur présente dans la colonne clé primaire de la table parent.

Le **R** signifie **relationnel**. CODD a créé une algèbre relationnelle, qui est une extension de l'algèbre ensembliste. Les opérandes sont des relations et les opérations sont la projection, la restriction, le produit cartésien, la jointure, l'union, l'intersection, la différence, la division.

SQL (Structured Query Language), le langage standard d'interrogation des BDR et l'objet de ce support de cours, est une implémentation de cette algèbre.

Un SGBDR doit garantir les propriétés **ACID** (Atomicité, Cohérence, Isolation, Durabilité).

- ✓ Atomicité : toutes les actions atomiques sont validées ou aucune.
- ✓ Cohérence : une transaction doit permettre de passer d'un état cohérent N à un état cohérent N+1.
- ✓ Isolation : lors de la transaction seule l'application qui effectue la transaction a accès aux nouvelles valeurs.
- ✓ Durabilité : une transaction validée ne peut être défaite.

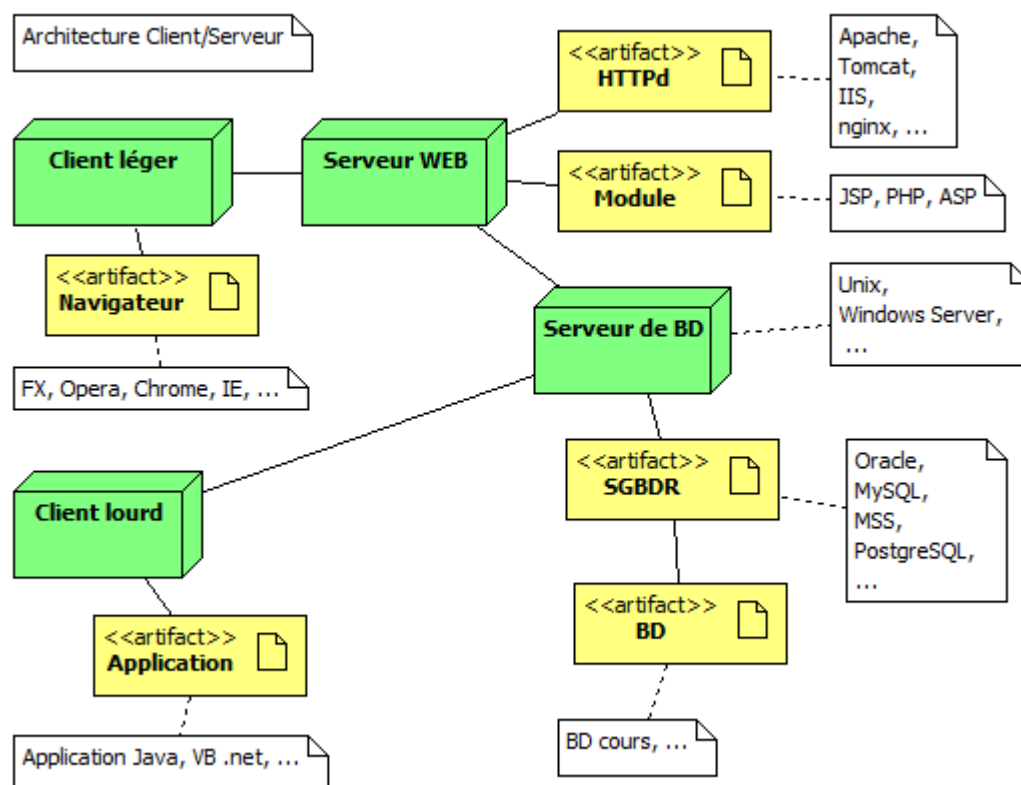
Cf le chapitre sur les transactions.

MySQL est un **SGBDR**.

1.3 - ARCHITECTURE DU SYSTÈME APPLICATIF

Contextes basiques d'utilisation d'un SGBDR.

Diagramme de déploiement UML.



1.4 - SQL

SQL (Structured Query Language) est un **langage algébrique** qui permet de définir des objets, de manipuler les données d'une base de données.

Note : QBE (Query By Example) est un **langage prédictif**. Ces deux langages sont équivalents au niveau de la puissance d'expression.

<http://www-inf.it-sudparis.eu/cours/bd/?idr=36#RTFToC12>

SQL est une implémentation de l'algèbre relationnelle, une extension de l'algèbre ensembliste, qui comporte **8 opérations** :

- ✓ la projection,
- ✓ la restriction,
- ✓ le produit cartésien,
- ✓ la jointure,
- ✓ l'union,
- ✓ l'intersection,
- ✓ la différence,
- ✓ la division.

Note : il est possible de classer les opérations en 3 catégories :

opérations unaires : projection et restriction,

opérations binaires de même schéma : union, intersection, différence,

opérations binaires de schémas différents : produit cartésien, jointure, division.

SQL est composé de 4 sous-langages (LDD, LMD, LCD, LCT).



LDD (Langage de Définition des Données) : permet de créer, modifier, supprimer des objets de la BD (DataBases, Users, Tables, Index, Views, Procedures, Functions, ...). Il gère des **contenants**.

Instruction	Description
CREATE	Crée un objet
ALTER	Modifie un objet
DROP	Supprime un objet



LMD (Langage de Manipulation des Données) : permet de créer, de sélectionner, de modifier et de supprimer des données. Il gère des **contenus**.
Les instructions sont SELECT, INSERT, UPDATE, DELETE.
Le LMD correspond aux quatre actions fondamentales exercées sur les données.

Instruction	Description
INSERT	Ajoute un enregistrement
SELECT	Extrait un ou plusieurs enregistrements
UPDATE	Modifie un enregistrement
DELETE	Supprime un enregistrement



LCD (Langage de Contrôle des Données) : permet de gérer les droits sur les données. Il gère l'**accès** aux contenants et/ou aux contenus.

Instruction	Description
GRANT	Distribue un droit
REVOKE	Révoque un droit



LCT (Langage de Contrôle de Transactions) : permet de gérer les transactions. Il gère la **validation** ou l'**invalidation** des modifications des contenus.

Instruction	Description
COMMIT	Valide toutes les mises à jour
ROLLBACK	Invalide toutes les mises à jour

Sous-langages et métiers :

Administrateur de Bases de Données : LDD et LCD.
Développeur : LMD et LCT.

1.5 - LES NORMES SQL

Naissance de SQL

Année	Événement
1970	Edgar Frank Codd publia l'article «A Relational Model of Data for Large Shared Data Banks» ("Un modèle de données relationnel pour de grandes banques de données partagées") dans la revue Communications of the ACM (Association for Computing Machinery).
1970	Donald Chamberlain et Raymond Boyce ont conçu chez IBM System R. Le langage est nommé SEQUEL.
1975	Naissance de SQL.
1979	Relational Software, Inc. (actuellement Oracle Corporation) présente la première version commercialement disponible de SQL.
1986	SQL est adopté comme recommandation par l'Institut de normalisation américaine (ANSI), puis comme norme internationale par l'ISO en 1987 sous le nom de ISO/CEI 9075.

Normes SQL

Année	Normes	Fonctionnalités, commentaires ...
1986	SQL-86	Édité par l'ANSI puis adopté par l'ISO en 1987.
1989	SQL-89 ou SQL1	Révision mineure.
1992	SQL-92 ou SQL2	Révision majeure.
1999	SQL-99 ou SQL3	Expressions rationnelles, requêtes récursives, déclencheurs, types non-scalaires et quelques fonctions orientées objet.
2003	SQL:2003	Introduction de fonctions pour la manipulation XML, « window functions », ordres standardisés et colonnes avec valeurs auto-produites (y compris colonnes d'identité).
2008	SQL:2008	Ajout de quelques fonctions de fenêtrage (ntile, lead, lag, first value, last value, nth value), limitation du nombre de lignes (OFFSET/FETCH). Amélioration mineure sur les types distincts, curseurs et mécanismes d'auto incrément.
2011	SQL:2011	La possibilité d'utiliser un INSERT, UPDATE, DELETE ou MERGE comme table dérivée. L'apparition de paramètres nommés lors d'une instruction CALL avec ou sans valeur par défaut. De nouvelles fonctionnalités pour mieux gérer la pagination des résultats.

1.6 - BDs UTILISÉES

Cours : cours, genealogies.

Exercices : coursReducit2014, pariscopes, bd_light_ingenieurs, bd_ingenieurs.

Les schémas et scripts de création des bases et d'insertion des données sont en fin de support ou sous forme de fichiers externes.

CHAPITRE 2 - INSTALLATION ET CONFIGURATION

2.1 - STRATÉGIES ET FICHIERS À TÉLÉCHARGER

Il existe au moins deux possibilités :

- ✓ Soit installer des produits séparément (MySQL, WorkBench, anciennement MySQL Query Browser et MySQL Administrator).
- ✓ Soit installer un AMP (Apache, MySQL, PHP) : LAMP, WAMP ou MAMP. Un AMP installe automatiquement un serveur HTTPd (Apache), un module PHP, un SGBDR (MySQL) et une interface d'administration et de requêtage (PHPMyAdmin).

L'installation séparée de MySQL consiste à n'installer que le SGBDR MySQL.

Il est possible de ne travailler qu'avec l'utilitaire mysql en mode commande, mais vous pouvez installer WorkBench (ou MySQL Query Browser) qui est une interface conviviale client lourd multi-plateforme.

D'autres outils existent : des AGL (Atelier de Génie Logiciel), des outils de migration, ..., des pilotes pour des connexions applicatives, des références hors-ligne.

Sites pour MySQL :

<http://www-fr.mysql.com/>

<http://dev.mysql.com/downloads/>

Tableau des téléchargements possibles (non exhaustifs et versions non définitives).

La liste correspond aux versions à une certaine date. Les versions évoluent rapidement.

Fonction	Fichier
MySQL	mysql-x.y.z-winXX.msi
Interface d'administration (MySQL Administrator), Interface d'interrogation (MySQL Query Browser), Interface de migration (MySQL Migration Toolkit).	mysql-gui-tools-noinstall-5.0-r15-win32.zip
MySQL WorkBench	Concepteur graphique de BD. Nécessite sous Windows le framework .NET 2.0 minimum.
DBDesigner	Concepteur graphique de BD (Obsolète mais intéressant dans la mesure où il possède un QBE).
Connecteur ODBC 5.1	mysql-connector-odbc-x.y.z-win32.msi
Connecteur JDBC	mysql-connector-java-x.y.z.zip
Connecteur PHP	php_mysql.dll for PHP x.y.z
Connecteur .NET	mysql-connector-net-x.y.z.zip
La référence en français (> 1500 pages)	mysql_5_reference_fr.pdf
Le fichier d'aide (format Winhelp)	http://downloads.mysql.com/docs/refman-5.0-fr.chm ou http://www.placeoweb.com/chm/

Sites pour les AMPs :

<http://www.apachefriends.org/fr/xampp-windows.html>

<http://www.wampserver.com/>

<http://www.easyphp.org/>

2.2 - INSTALLATIONS

2.2.1 - Installation du SGBDR

2.2.1.1 - Différents types d'installation

- ✓ **MySQL** : installation rapide et assistée (Cf support Administration).
- ✓ XAMPP, EasyPHP, WAMP5 : installations rapides et assistées (WAMP).

Cf le support mysql_5_administration.doc pour plus de détails.

2.2.1.2 - Paramétrage pour la France

Notes sur la francisation : modifications dans le paragraphe [mysqld] de my.ini des valeurs par défaut de variables système globales.

Le timezone :

```
| default-time-zone = "Europe/Paris"
```

Les noms des jours et des mois en français via les fonctions sur les dates (cf aussi le paragraphe dédié) :

```
| lc_time_names = fr_FR
```

Les messages d'erreur en français :

```
| language=french
```

Pour tester ces valeurs

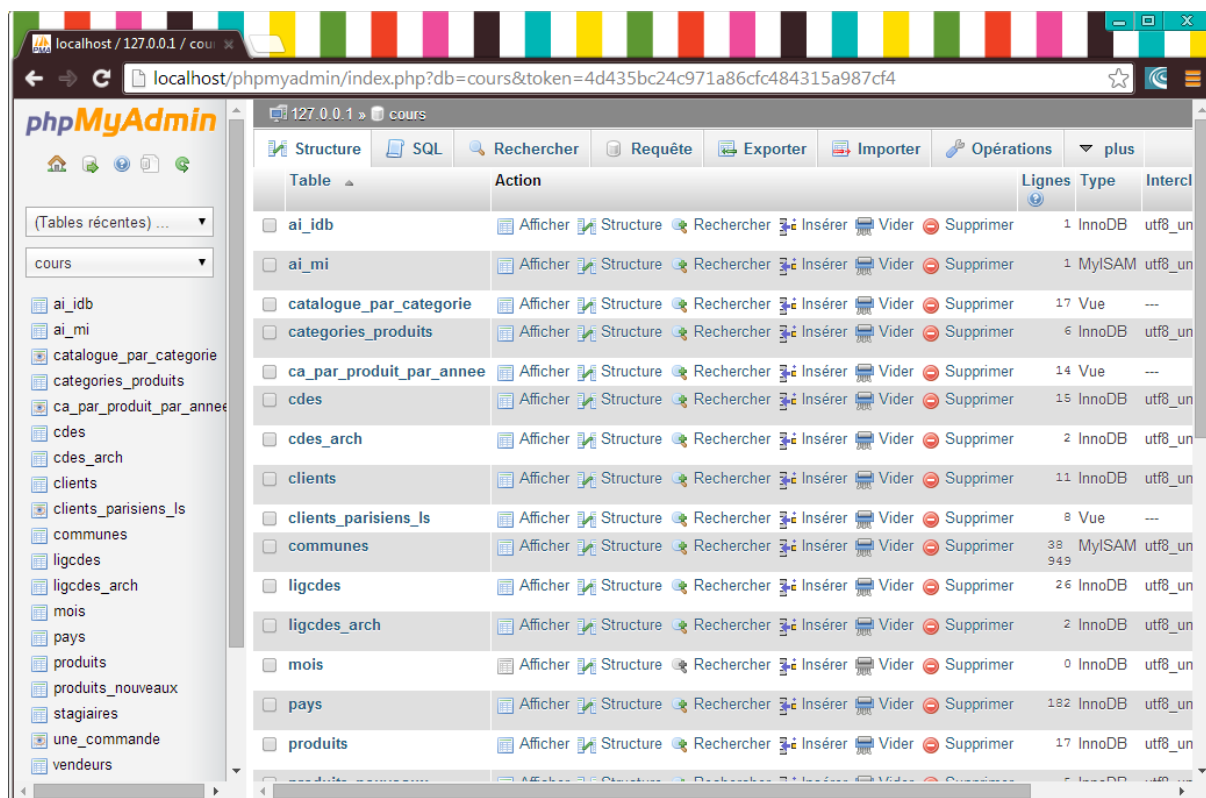
```
| SELECT @@time_zone;  
| SELECT @@lc_time_names;
```

2.2.2 - Installation des outils-clients

2.2.2.1 - phpMyAdmin

Avec EasyPHP, WAMP et XAMPP l'outil phpMyAdmin est fourni.

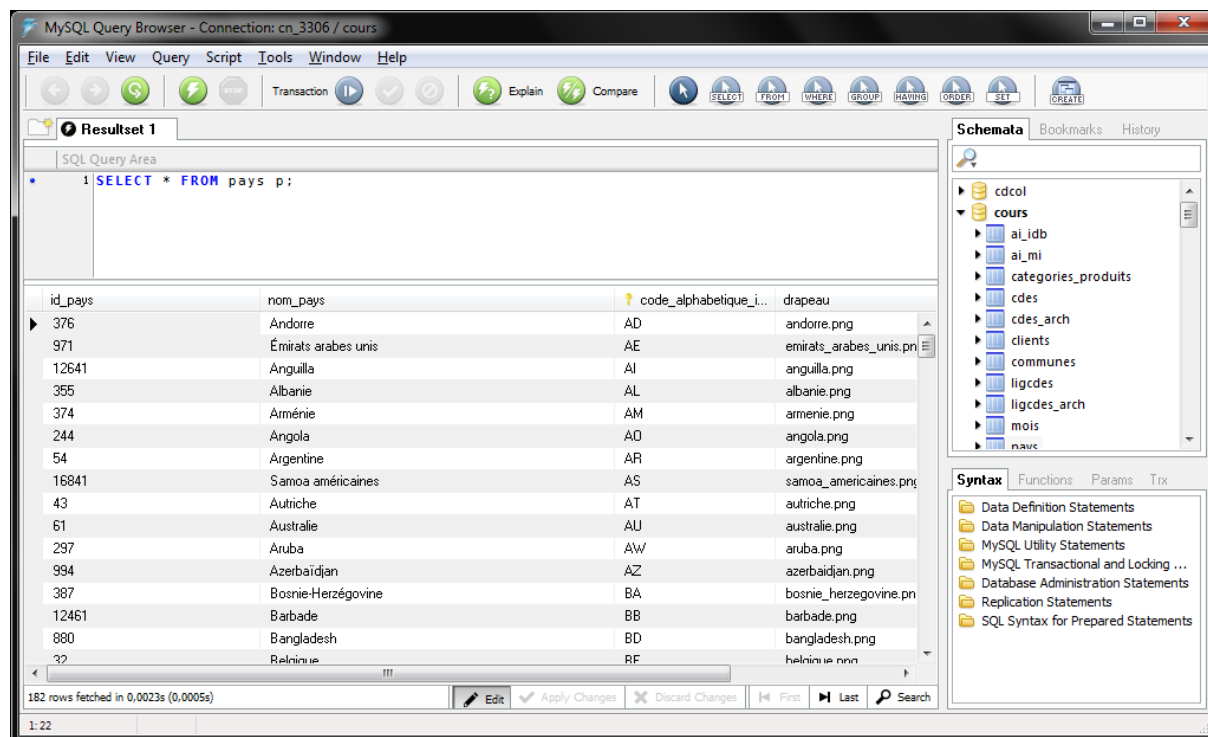
Pour l'installer de façon indépendante, phpMyAdmin-x.y.z-all-languages.zip est téléchargeable à http://www.phpmyadmin.net/home_page/downloads.php



2.2.2.2 - MySQL Query Browser de MySQL AB



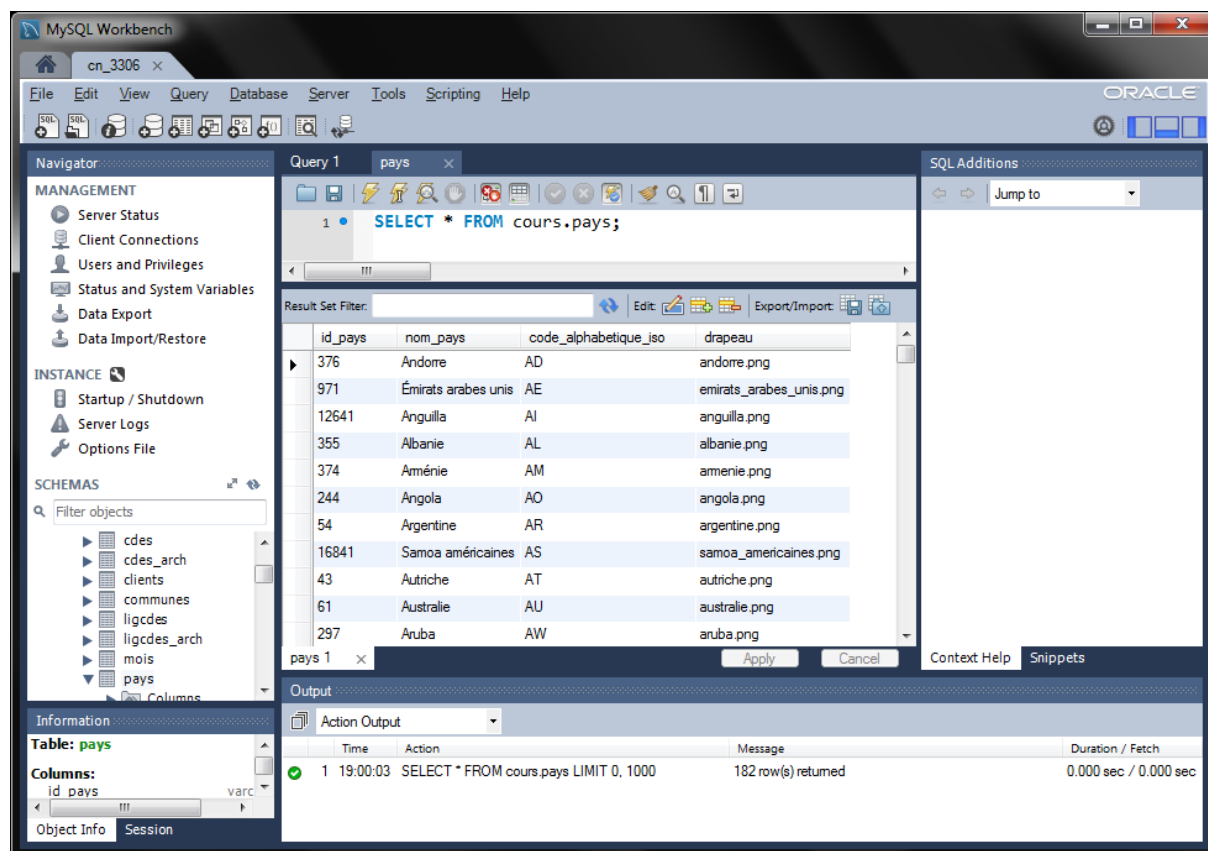
Client lourd plus convivial : MySQL Query Browser (pour Windows, Mac et Linux).
C'est un utilitaire gratuit édité par MySQL AB.
Cf le support `mysql_query_browser.doc`.



2.2.2.3 - MySQL Workbench 6.x.x d'Oracle Corporation

La dernière version intègre tous les outils cités précédemment plus quelques autres.

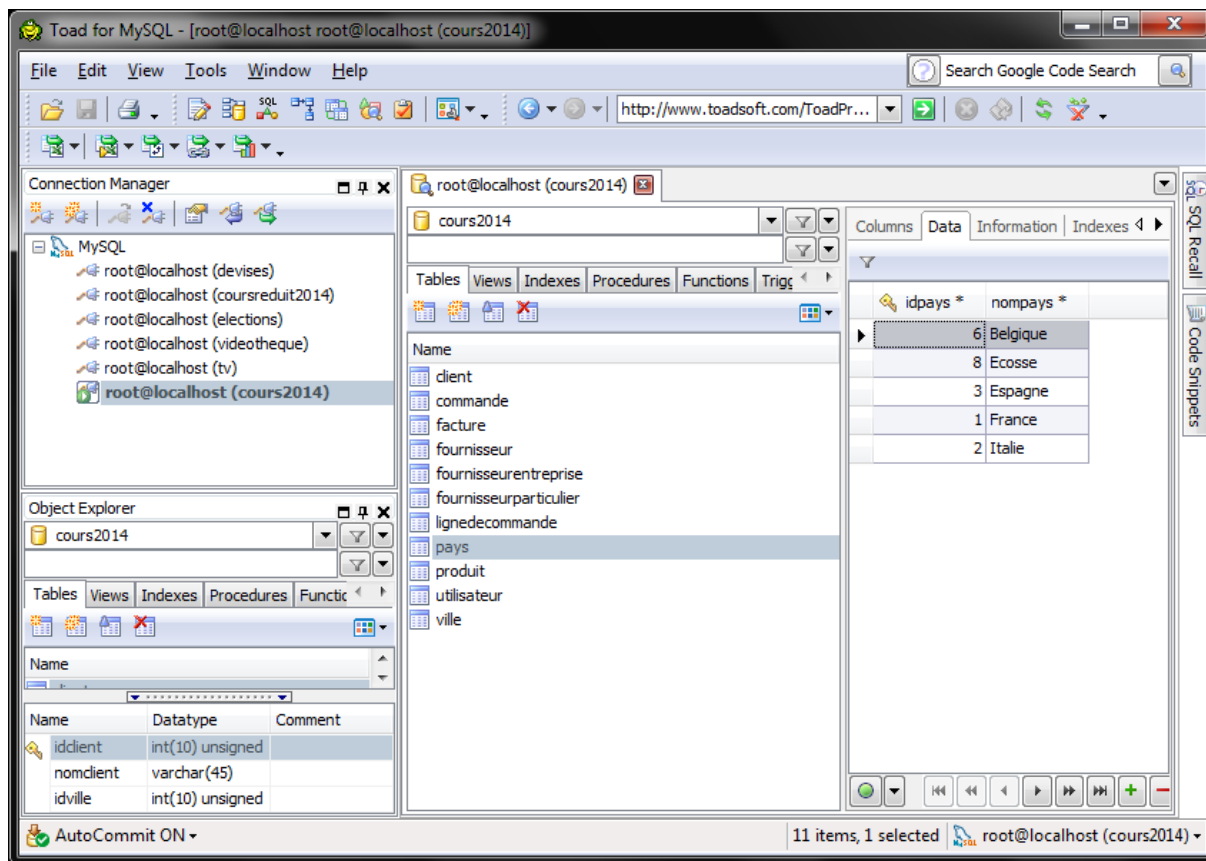
<http://www.mysql.fr/products/workbench/>



2.2.2.4 - TOAD for MySQL

Outil possédant un QBE.

<http://www.quest.com/toad-for-mysql/>



2.2.2.5 - DBDesigner de FabForge.net



AGL pour la conception de la BD (AGL de conception, reverse engineering, QBE). Non maintenu depuis 2005. Il possède un QBE absent dans Workbench.

2.2.2.6 - MySQL Administrator de MySQL AB



Outil pour l'administration des BD.

2.3 - TRAVAILLER AVEC MYSQL QUERY BROWSER

Cf le support mysql_query_browser.odt.

2.4 - TRAVAILLER AVEC WORKBENCH

Cf le support mysql_workbench_6x.odt.

2.5 - TRAVAILLER AVEC PHPMYADMIN

<http://localhost/phpmyadmin/>

2.6 - TRAVAILLER AVEC TOAD FOR MYSQL

Cf le support toad_for_mysql.odt.

2.7 - TRAVAILLER AVEC L'UTILITAIRE MYSQL

Se connecter au serveur

```
mysql -h serveur -u utilisateur -p
```

- **Exemples**

```
C:\...\mysql\bin> mysql --host=localhost --user=root --password= --database=cours
```

ou

```
C:\...\mysql\bin> mysql -h localhost -u root -D cours -p
```

ou

```
C:\...\mysql\bin> mysql -h localhost -u root -p
```

Ensuite il vous sera demandé de saisir le mot de passe.

Enfin vous êtes dans l'interface de l'utilitaire mysql avec son prompt : mysql>.

Lister les bases de données du serveur

```
mysql> show databases;
```

Pour sélectionner une base

```
USE nom_de_la_bd;
```

- **Exemple**

```
mysql> use mysql;
```

Puis les commandes SQL

```
mysql> SELECT * FROM user;
```

Lister les tables

```
SHOW tables;
```

Lister la structure d'une table

```
DESC nom_de_table;
```

- **Exemple ... dans la base nommée MySQL**

```
mysql>DESC user;
```

Pour sortir

```
mysql>exit;
```

CHAPITRE 3 - LES OBJETS MYSQL

3.1 - LES BASES DE DONNÉES

3.1.1 - Création

- **Syntaxe**



```
CREATE DATABASE [IF NOT EXISTS] nom_de_base
[DEFAULT CHARACTER SET jeu_de_caractères
COLLATE collation];
```

- **Exemples**

```
CREATE DATABASE IF NOT EXISTS base_exos;
```

```
CREATE DATABASE IF NOT EXISTS base_exos
DEFAULT CHARACTER SET utf8
COLLATE utf8_unicode_ci;
```

```
USE base_exos;
```

Autres jeux de caractères :

```
CREATE DATABASE bd1 DEFAULT CHARACTER SET utf8 COLLATE utf8_general_cs;
CREATE DATABASE bd2 DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
CREATE DATABASE bd3 DEFAULT CHARACTER SET latin1 COLLATE latin1_general_ci;
```

Notes :

Un CHARSET (jeu de caractères) est un ensemble de symboles et de codes. Exemples : utf8, latin1, greek, hebrew, ...
 Une collation est un ensemble de règles permettant la comparaison de caractères dans un jeu de caractères.
 Un encodage UTF-8 encode un ensemble de caractères (plus d'un million) Unicode sur 1, 2 ou 3 octets.
 Il permet le stockage de textes de 650 langages.
 La collation general_ci est basée sur un algorithme simple et rapide de comparaison.
 La collation unicode_ci sur un algorithme plus complexe.

Cf : <http://dev.mysql.com/doc/refman/5.1/en/charset-unicode-sets.html>

Un encodage ISO 8859-1 correspond à la norme ISO 8859-1 (Alphabet de l'Europe occidentale). C'est une table de 191 caractères. Elle comprend les caractères accentués du français.
 La collation pour le français est latin1_general_ci ou latin1_general_cs.

COLLATE

La COLLATION c'est l'algorithme utilisé pour les comparaisons de chaînes de caractères.

Pour utf8 : utf8_general_ci, utf8_bin, utf8_unicode_ci, ...
 Pour latin1 : latin1_swedish_ci, latin1_bin, latin1_general_cs, latin1_general_ci, ...
 Une comparaison binaire est une comparaison exacte : Å est différent de a, Ä est différent de A.
 Lors d'une comparaison "general" Å est égal à A.
 utf8_unicode_ci gère les ligatures (fusion de deux graphèmes d'une écriture pour en former un) : par exemple oe. Ce que ne gère pas utf8_general_ci.
 Attention, par défaut de nombreuses plate-formes MySQL sont en utf8 collation utf8_swedish_ci.

Cf d'autres détails et des exemples dans les annexes.

3.1.2 - Modification d'une base de données

- **Syntaxe**

```
ALTER DATABASE nom_de_base DEFAULT CHARACTER SET jeu COLLATE collation;
```

- **Exemple**

Modification de la collation du jeu de caractères.

```
| ALTER DATABASE base_exos DEFAULT CHARACTER SET utf8 COLLATE utf8_general_cs;
```

3.1.3 - Suppression d'une base de données

- **Syntaxe**



```
DROP DATABASE [IF EXISTS] nom_de_base;
```

- **Exemples**

```
| DROP DATABASE base_exos;
```

ou

```
| DROP DATABASE IF EXISTS base_exos;
```

Le schéma de la BD Cours est dans les annexes.

3.2 - LES TABLES

3.2.1 - Les types de données

<http://dev.mysql.com/doc/refman/5.0/fr/column-types.html?ff=nopfls>

Les grandes catégories de types de données sont :

- ✓ Numériques,
- ✓ Caractères,
- ✓ Dates,
- ✓ Binaires,
- ✓ Ensemble (Enumeration ou Set),
- ✓ Géométriques.

Les types numériques peuvent être signés ou non (UNSIGNED).

Ils peuvent aussi être complétés par des 0 non significatifs (ZEROFILL). Un numérique avec ZEROFILL est nécessairement UNSIGNED.

Pour les numériques de type Int, TinyInt, ... la taille de l'affichage est paramétrable avec l'attribut (M). Ceci est visible si c'est combiné avec ZEROFILL.

Pour les numériques de type Float et Double la taille de l'affichage est paramétrable avec l'attribut (M,D). Ceci est visible si c'est combiné avec ZEROFILL. M correspond au nombre total de chiffres et D au nombre de chiffres de la partie décimale.

Petit entier TINYINT(5) UNSIGNED ZEROFILL DEFAULT NULL,

entier INT(5) UNSIGNED ZEROFILL DEFAULT NULL,

Dans le cas des valeurs suivantes 100, 100 affichera 00100, 00100

Dans le cas des valeurs suivantes 255, 1 000 000 affichera 00255, 1000000.

Type	Sous-types	Octets	Extensions
Numériques	TinyInt	1	0 à 255 ou -127 à 128
	SmallInt	2	-32 768 à 32 767 ou 0 à 65 535
	MediumInt	3	-8 388 608 à 8 388 607 ou 0 à 16 777 215
	Int[(M)]	4	-2 147 483 648 à 2 147 483 647
	BigInt	8	0 à 18 446 744 073 709 551 615 (> 18 trillions)
	Float, Float[(M,D)]	4	Ce type de données permet de stocker des nombres flottants à précision simple. Va de -1.175494351E-38 à 3.402823466 ^E +38. Donc entre le sextillion et le sextilliard. Float est à éviter (Les calculs renvoyés sont incertains) Float(M,D); M pour le nombre de chiffres à afficher, D pour le nombre de chiffres décimaux. M doit être supérieur à D.
	Double, Double[(M,D)]	8	Stocke des nombres flottants à double précision de -1.7976931348623157E+308 à -2.2250738585072014E-308, 0, et de 2.2250738585072014E-308 à 1.7976931348623157E+308.
Date	Date	3	'AAAA-MM-JJ'

	Time	3	'HH:MM:SS'
	DateTime	8	'AAAA-MM-JJ HH:MM:SS'
	Timestamp	4	Le type TIMESTAMP est prévu pour stocker automatiquement l'heure courante lors d'une commande INSERT ou UPDATE si vous envoyez NULL ou même sans mentionner la colonne. Les TIMESTAMP sont affichés comme les DATETIME.
	Year[(2 4)]	1	MySQL extrait et affiche la valeur de YEAR au format YYYY. L'échelle va de 1901 à 2155. Vous pouvez spécifier un nombre de 2 ou de 4 chiffres. http://dev.mysql.com/doc/refman/5.0/fr/year.html
Caractères	Char(N)	N	255 caractères maximum
	Varchar(N)	N	255 caractères maximum (MySQL convertit en TEXT ou MEDIUMTEXT si vous dépassez 255)
Binaire	TinyBlob	255	
	Blob	65 535	
	MediumBlob	16 Mo	Ou 16 777 215 d'octets
	LongBlob	4 Go	Ou 4 294 967 295 d'octets
Textes	TinyText	255	
	Text	65 535	
	MediumText	16 Mo	Ou 16 777 215 d'octets
	LongText	4 Go	Ou 4 294 967 295 d'octets
Enumération	Enum('valeur1', 'valeur2',...)		Choix unique et obligatoire. Une chaîne qui doit prendre une valeur , sélectionnée parmi une liste 'valeur1', 'valeur2', ..., NULL ou la valeur spéciale d'erreur ''. 65535 valeurs distinctes sont autorisées. L'interface : liste déroulante ou boutons radio
Ensemble	Set('valeur1', 'valeur2',...)		Choix facultatif ou multiple. Une chaîne, qui peut prendre zéro, une ou plusieurs valeurs, choisies parmi une liste de valeurs 'valeur1', 'valeur2', ... Une valeur SET peut avoir un maximum de 64 membres. L'interface : liste à choix multiples.

Il existe aussi les types géométriques (Geometry, Point, MultiPoint, LineString, MultiLineString, Polygon, MultiPolygon, GeometryCollection).

Notes pour les chaînes :

L'attribut BINARY signifie que les valeurs sont classées et triées en tenant compte de la casse, suivant l'ordre des caractères ASCII de la machine.

L'attribut ASCII peut être spécifié pour assigner le jeu de caractères utf8 à une colonne de type CHAR.

L'attribut UNICODE peut être spécifié pour assigner le jeu de caractères ucs2 à une colonne CHAR.

Référence pour les entiers:

cf <http://dev.mysql.com/doc/refman/5.0/en/integer-types.html>

Références diverses sur les types

Les bits : <http://dev.mysql.com/doc/refman/5.0/en/bit-field-literals.html>

Les booléens : <http://dev.mysql.com/doc/refman/5.0/en/numeric-type-overview.html>

Un booléen est un tinyint(1) où 0 correspond à false et !=0 correspond à true.

Référence sur les colonnes auto_increment

<http://dev.mysql.com/doc/refman/5.0/fr/innodb-auto-increment-column.html>

Un auto_increment dans une table InnoDB est stocké temporairement dans la table INFORMATION_SCHEMA.TABLES et n'est pas stocké définitivement.

Car MySQL au redémarrage exécute un SELECT MAX(colonne auto_increment) + 1 pour recréer la nouvelle valeur.

Un auto_increment avec une table MyISAM est stocké définitivement.

```
SELECT TABLE_NAME, TABLE_TYPE, AUTO_INCREMENT
FROM information_schema.TABLES T
WHERE TABLE_SCHEMA = 'cours'
AND TABLE_TYPE = 'BASE TABLE'
AND AUTO_INCREMENT IS NOT NULL;
```

Tests :

```
DROP TABLE IF EXISTS cours.ai_mi;
CREATE TABLE  cours.ai_mi (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom varchar(45) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

DROP TABLE IF EXISTS cours.ai_idb;
CREATE TABLE  cours.ai_idb (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom varchar(45) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

INSERT INTO ai_idb(nom) VALUES('a');
INSERT INTO ai_idb(nom) VALUES('b');
INSERT INTO ai_idb(nom) VALUES('c');

INSERT INTO ai_mi(nom) VALUES('a');
INSERT INTO ai_mi(nom) VALUES('b');
INSERT INTO ai_mi(nom) VALUES('c');

DELETE FROM ai_idb WHERE id > 1;
DELETE FROM ai_mi WHERE id > 1;
```

Exécutez le SELECT sur la table [TABLES] avant d'arrêter le serveur ; auto_increment est à 4 pour les 2 tables.

Arrêtez MySQL, redémarrez MySQL, exécutez le SELECT sur la table [TABLES].

auto_increment est à 2 pour la table InnoDB et à 4 pour la table MyISAM.

Note : si vous affectez 0 ou NULL à une colonne auto_increment lors d'un INSERT l'auto_increment est généré. Mais si la clause SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO"; est spécifiée vous pouvez affecter la valeur 0 à une colonne de ce type.

3.2.2 - Création d'une table

- **Syntaxe**



```
CREATE TABLE nom_de_table (
  nom_de_colonne TYPE [CONTRAINTE]
  [, col2 ... ,
  [Contrainte de table]])
[ENGINE moteur_de_table]
[DEFAULT CHARSET=jeu_de_caractères COLLATE=collation];
```

- **Exemple de base (sans contrainte)**

```
CREATE TABLE villes (cp CHAR(5), nom_ville VARCHAR(50), id_pays CHAR(4));
```

- **Exemple avec une clé primaire, un auto_increment et un moteur spécifique**

```
CREATE TABLE clients (
  id_client INT(5) NOT NULL AUTO_INCREMENT ,
  nom VARCHAR(50) NOT NULL ,
  prenom VARCHAR(50) NULL ,
  adresse VARCHAR(100) NULL ,
  date_naissance DATE NULL ,
  cp CHAR(5) NOT NULL ,
  PRIMARY KEY (id_client)
) ENGINE = InnoDB;
```

- **Exemple avec en plus un test d'existence et un jeu de caractères spécifique**

```
CREATE TABLE IF NOT EXISTS clients (
  id_client INT(5) NOT NULL AUTO_INCREMENT ,
  nom VARCHAR(50) NOT NULL ,
  prenom VARCHAR(50) NULL ,
  adresse VARCHAR(100) NULL ,
  date_naissance DATE NULL ,
  cp CHAR(5) NOT NULL ,
  PRIMARY KEY (id_client)
) ENGINE = InnoDB
DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

ENUM et SET :

Ces deux dernières possibilités sont à utiliser avec précaution pour des raisons de souplesse et de maintenabilité.

- **Exemple avec une énumération (Choix obligatoire parmi la liste de valeurs)**

... connu_par **ENUM**('Internet','Bouche à oreille','Presse') default NULL,

- **Exemple avec un ensemble (Choix facultatif dans la liste, le choix peut être multiple)**

... lectures **SET**('Romans','Essais','Scolaires','Art','Techniques') default NULL,

- **Les contraintes**

Les contraintes prises en charge par MySQL sont (D'autres sont acceptées mais non validées : pages, ...) :

- ✓ NOT NULL,
- ✓ PRIMARY KEY,
- ✓ INDEX UNIQUE.

Exemple avec des colonnes NOT NULL

```
CREATE TABLE villes (cp CHAR(5) NOT NULL, nom_ville VARCHAR(50) NOT NULL, site VARCHAR(50) NULL , photo VARCHAR(50) NULL , id_pays CHAR(3) NOT NULL);
```

Exemple avec la création d'une clé primaire.

```
CREATE TABLE villes (  
cp CHAR(5) NOT NULL ,  
nom_ville VARCHAR(50) NOT NULL ,  
site VARCHAR(50) NULL ,  
photo VARCHAR(50) NULL ,  
id_pays CHAR(3) NOT NULL ,  
PRIMARY KEY (cp)  
)  
ENGINE = InnoDB;
```

Exemple de création d'une table avec une clé primaire et une colonne indexée

```
CREATE TABLE villes (  
cp CHAR(5) NOT NULL ,  
nom_ville VARCHAR(50) NOT NULL ,  
site VARCHAR(50) NULL ,  
photo VARCHAR(50) NULL ,  
PRIMARY KEY (cp) ,  
INDEX (nom_ville)  
) ENGINE = InnoDB;
```

3.2.3 - Création d'une table à partir d'une autre table

```
CREATE TABLE table_a_creer  
AS SELECT * | colonnes  
FROM table_source | jointure  
[WHERE condition];
```

```
| CREATE TABLE villes_bis AS SELECT * FROM villes;
```



Seule la structure de base est copiée. Les clés ... sont omises dans la copie.

3.2.4 - Renommage d'une table

- **Syntaxe**

```
ALTER TABLE ancien_nom RENAME TO nouveau_nom;
```

- **Exemple**

```
| ALTER TABLE villes_bis RENAME TO villes_2;
```

3.2.5 - Suppression d'une table

- **Syntaxe**



```
DROP TABLE [IF EXISTS] nom_de_table;
```

- **Exemple**

```
| DROP TABLE villes_2;
```

3.2.6 - Modification de la structure d'une table

- **Syntaxes**

Pour ajouter une colonne ou une contrainte.

```
ALTER TABLE nom_de_table ADD ...[, ADD ...]
```

Pour supprimer une colonne.

```
ALTER TABLE nom_de_table DROP nom_de_colonne [, DROP nom_de_colonne];
```

Pour modifier une colonne ou une contrainte.

```
ALTER TABLE nom_de_table CHANGE ...
```


- **Exemples**

Ajout d'une clé primaire

```
CREATE TABLE villes (  
  cp CHAR(5) NOT NULL ,  
  nom_ville VARCHAR(50) NOT NULL  
) ENGINE = InnoDB;  
  
ALTER TABLE villes ADD PRIMARY KEY (cp);
```

Ajout d'une colonne

```
ALTER TABLE villes ADD id_pays CHAR(4) NOT NULL;
```

Suppression d'une colonne

```
ALTER TABLE villes DROP id_pays;
```

Ajout d'un index

Cf plus loin

Modification d'un type

```
ALTER TABLE villes CHANGE id_pays id_pays INT(4) NOT NULL;
```

Modification d'une contrainte (NOT NULL -> NULL)

```
ALTER TABLE villes CHANGE id_pays id_pays CHAR(4) NULL;
```

Et inversement (Si aucune données n'est dans la table)

```
ALTER TABLE villes CHANGE id_pays id_pays CHAR(4) NOT NULL;
```

- **Autre syntaxe pour la modification d'une colonne**

```
ALTER TABLE nomDeTable MODIFY COLUMN nomDeColonne Type Contrainte;
```

- **Exemple**

```
ALTER TABLE villes MODIFY COLUMN id_pays CHAR(4) CHARACTER SET utf8 COLLATE  
utf8_general_ci NOT NULL;
```

Ou

```
ALTER TABLE villes MODIFY COLUMN id_pays CHAR(4) NOT NULL;
```

3.3 - LES CONTRAINTES (CONSTRAINT)

3.3.1 - Création d'une contrainte

- **Syntaxe**

```
ALTER TABLE nom_de_table
ADD CONSTRAINT nom_contrainte Contrainte [, ADD CONSTRAINT ...];
```

Note : bien entendu une contrainte peut être créée lors de la création de la table comme nous l'avons vu précédemment.

Admettons la table suivante **STAGIAIRES**(#id_stagiaire, nom, statut, #cp).

```
CREATE TABLE stagiaires(id_stagiaire INT(5), nom VARCHAR(50), age INT, cp CHAR(5))
ENGINE = innnoDB;
```

Clé primaire

```
ALTER TABLE stagiaires
ADD CONSTRAINT pk_stagiaires
PRIMARY KEY (id_stagiaire);

ALTER TABLE stagiaires
MODIFY COLUMN id_stagiaire INT(5) NOT NULL AUTO_INCREMENT;
```

Clé étrangère

```
ALTER TABLE stagiaires
ADD CONSTRAINT fk_stagiaires_cp
FOREIGN KEY (cp)
REFERENCES villes(cp)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Note : SET FOREIGN_KEY_CHECKS = 0; // Désactive les clés étrangères

Valeur unique

```
ALTER TABLE stagiaires
ADD CONSTRAINT stagiaires_nom_u
UNIQUE (nom);
```

Validité (Plage, ...) MySQL ne supporte pas encore ce type de contrainte.

ALTER TABLE stagiaires ADD CONSTRAINT cst_age CHECK age BETWEEN 18 AND 25;
Cf le type ENUM.

Clé étrangère réflexive

```
ALTER TABLE vendeurs ADD CONSTRAINT FK_vendeurs_id_vendeur FOREIGN KEY  
FK_vendeurs_id_vendeur (chef)  
    REFERENCES vendeurs (id_vendeur)  
    ON DELETE RESTRICT  
    ON UPDATE RESTRICT  
, ROW_FORMAT = DYNAMIC;
```

3.3.2 - Le cas particulier des clés étrangères

Nous avons vu qu'une clé étrangère est une contrainte d'intégrité référentielle.

Quelles sont les possibilités de mises à jour d'une table enfant lorsque l'on supprime (DELETE) une ligne dans une table parent ? (les règles s'appliquent aussi pour la mise à jour – UPDATE –).

Admettons les tables Pays(#id_pays, nom_pays) et Villes(#id_ville, nom_ville, #id_pays).



```
DROP TABLE IF EXISTS pays;
CREATE TABLE pays (
  id_pays int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom_pays varchar(45) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (id_pays) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

```
DROP TABLE IF EXISTS ville;
CREATE TABLE ville (
  id_ville int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom_ville varchar(45) COLLATE utf8_unicode_ci NOT NULL,
  id_pays int(10) unsigned DEFAULT NULL,
  PRIMARY KEY (id_ville) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Pour la clé étrangère dans la table Ville il existe 4 possibilités : RESTRICT, NO ACTION, CASCADE, SET NULL.

RESTRICT : interdiction de supprimer une ligne dans la table parent s'il y a une ou des lignes correspondantes dans la table enfant

```
ALTER TABLE ville ADD CONSTRAINT FK_ville_pays FOREIGN KEY FK_ville_pays (id_pays)
REFERENCES pays (id_pays)
ON DELETE RESTRICT
ON UPDATE RESTRICT;
```

NO ACTION : interdiction de supprimer une ligne dans la table parent s'il y a une ou des lignes correspondantes dans la table enfant (idem au précédent)

```
ALTER TABLE ville ADD CONSTRAINT FK_ville_pays FOREIGN KEY FK_ville_pays (id_pays)
REFERENCES pays (id_pays)
ON DELETE NO ACTION
ON UPDATE NO ACTION;
```

SET NULL : on affecte la valeur NULL à la colonne FK (agrégation UML) dans les lignes de la table enfant.

```
ALTER TABLE ville ADD CONSTRAINT FK_ville_pays FOREIGN KEY FK_ville_pays (id_pays)
REFERENCES pays (id_pays)
ON DELETE SET NULL
ON UPDATE SET NULL;
```

CASCADE : on supprime les lignes dans la table enfant (composition UML)

```
ALTER TABLE ville ADD CONSTRAINT FK_ville_pays FOREIGN KEY FK_ville_pays (id_pays)
REFERENCES pays (id_pays)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

3.3.3 - Suppression d'une contrainte

```
ALTER TABLE nom_de_table DROP TYPE_DE_CONTRAINTE [nom_de_contrainte];
```

```
ALTER TABLE stagiaires DROP PRIMARY KEY;  
ALTER TABLE stagiaires DROP FOREIGN KEY fk_stagiaires_cp;  
ALTER TABLE stagiaires DROP INDEX stagiaires_nom_u;
```

3.3.4 - Liste des contraintes

On trouve ceci dans la BD information_schema.

```
SELECT constraint_name, table_schema, table_name, constraint_type  
FROM information_schema.TABLE_CONSTRAINTS T  
WHERE constraint_schema = 'cours' AND table_name = 'pays';
```

ou

```
SELECT * FROM KEY_COLUMN_USAGE  
WHERE constraint_schema='cours' AND table_name='pays';  
  
SELECT * FROM KEY_COLUMN_USAGE  
WHERE constraint_schema='cours' AND table_name='clients';
```

3.4 - LES INDEX (INDEX)

3.4.1 - Création d'un index

Un index est un accélérateur.

Un index peut être créé dans l'instruction de création de table (Cela permet d'en créer plusieurs) ou via CREATE INDEX (Un seul à la fois).

Un index peut s'appliquer à une colonne ou plusieurs.

Un index peut être UNIQUE (Mais accepte les valeurs NULL, à la différence des Primary Key) ou NOT UNIQUE.

Un index peut être appliqué à un préfixe de colonne (colonne(n)).

Un index FULL TEXT peut être appliqué à un texte long sur des tables MyISAM.

• Syntaxe de CREATE INDEX.

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX nom_d_index ON nom_de_table
(nom_de_colonne[(n)],...)
```

FULLTEXT précise que l'index, même sur les textes longs, doit être créé sur toute la valeur du texte.

SPATIAL indexe les colonnes dont les types sont géométriques : Geometry(), Point(x,y), MultiPoint(pt1,pt2,...), LineString(pt1,pt2,...), MultiLineString(ls1,ls2,...), Polygon(ls1,ls2,...), MultiPolygon(poly1,poly2,...), GeometryCollection(g1,g2,...).

• Exemples

```
CREATE INDEX idx_villes_ville ON villes(nom_ville);
CREATE INDEX idx_clients_nom_prenom ON clients(nom, prenom);
CREATE INDEX idx_nom_ville ON villes(nom_ville(5)); -- Préfixe de colonne
```

• Syntaxe de ALTER TABLE ... ADD INDEX ...

```
ALTER TABLE nom_de_table ADD INDEX [nom_d_index] (nom_de_colonne [(n)] [,
nom_de_colonne]);
```

• Exemples

```
ALTER TABLE villes ADD INDEX idx_villes_nom_ville (nom_ville); -- Un index

ALTER TABLE villes ADD INDEX idx_villes_nom_ville (nom_ville), add index
i_villes_id_pays(id_pays); -- Deux index

ALTER TABLE villes ADD INDEX idx_villes_nom_ville_id_pays (nom_ville,id_pays); --
Un index sur 2 colonnes
```


3.4.2 - Suppression d'un index

- **Syntaxe**

```
DROP INDEX nom_d_index ON nom_de_table;
```

- **Exemple**

```
| DROP INDEX idx_villes_ville ON villes;
```

- **Syntaxe**

```
ALTER TABLE nom_de_table DROP INDEX nom_d_index;
```

- **Exemple**

```
| ALTER TABLE villes DROP INDEX idx_villes_nom_ville;
```

3.4.3 - Visualisation des Index

- **Via la structure de la table**

```
mysql>SHOW CREATE TABLE nom_de_table \G
```

```
mysql>SHOW CREATE TABLE villes \G
      Table: villes
Create Table: CREATE TABLE villes (
  cp varchar(5) NOT NULL,
  nom_ville varchar(50) NOT NULL,
  site varchar(50) DEFAULT NULL,
  photo varchar(50) DEFAULT NULL,
  id_pays char(3) DEFAULT NULL,
  PRIMARY KEY (cp),
  KEY index_id_pays (id_pays)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC
```

- **Via la structure des index d'une table**

```
SHOW INDEX FROM nom_de_table \G
```

```
mysql>SHOW INDEX FROM villes \G
***** 1. row *****
      Table: villes
      Non_unique: 0
      Key_name: PRIMARY
Seq_in_index: 1
Column_name: cp
Collation: A
Cardinality: 12
      Sub_part: NULL
      Packed: NULL
      Null:
      Index_type: BTREE
***** 2. row *****
      Table: villes
      Non_unique: 1
      Key_name: index_id_pays
Seq_in_index: 1
Column_name: id_pays
Collation: A
Cardinality: NULL
      Sub_part: NULL
      Packed: NULL
      Null: YES
      Index_type: BTREE
```

- **Index sur préfixe**

Les index sur préfixe sont plus rapides.

```
| CREATE INDEX i_nom_ville ON villes(nom_ville(5));
```

```
|      Table: villes  
|      Non_unique: 1  
|      Key_name: i_nom_ville  
|      Seq_in_index: 1  
|      Column_name: nom_ville  
|      Collation: A  
|      Cardinality: NULL  
|      Sub_part: 5  
|      Packed: NULL  
|      Null:  
|      Index_type: BTREE
```

Notes :

Les index par défaut sont de type B-TREE (arbre balancé).

Les index FULLTEXT ne peuvent indexer que des colonnes de type CHAR, VARCHAR ou TEXT, et seulement dans les tables MyISAM.

Les index SPATIAL peuvent indexer les colonnes spatiales, et uniquement avec les tables MyISAM.

Si une base de données est suffisamment petite pour tenir en mémoire, alors le plus rapide pour faire des requêtes est d'utiliser les index hash.

"R-trees are tree data structures that are similar to B-trees, but are used for spatial access methods i.e., for indexing multi-dimensional information".

3.4.4 - Utilisation des index

Par défaut les index sont utilisés avec un SELECT comportant une clause WHERE sur une colonne indexée.

Si un index est créé sur plusieurs colonnes l'index est utilisé lorsqu'une clause WHERE contient au moins la première des colonnes de l'index. S'il ne contient que la deuxième et/ou d'autres il est inactif.

Donc avec l'index crée plus haut sur nom et prenom l'index est utilisé dans les 3 cas suivants :

WHERE nom = '...'

WHERE nom = '...' AND prenom = '...'

WHERE nom = '...' AND prenom = '...' AND adresse = '...'

Et avec n'importe quel opérateur de comparaison.

Les index FULLTEXT sont utilisés avec les fonctions MATCH() et AGAINST().

Avec MATCH(liste des colonnes de l'index)

Et AGAINST(valeur recherchée)

```
CREATE TABLE clients_myisam(
  id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  nom VARCHAR(45) NOT NULL,
  prenom VARCHAR(45),
  PRIMARY KEY (id),
  FULLTEXT INDEX i_full_clients_nom_prenom(nom,prenom)
)
ENGINE = MyISAM;

INSERT INTO clients_myisam(nom, prenom) VALUES ('Tintin', 'Albert');
INSERT INTO clients_myisam(nom, prenom) VALUES ('Milou', 'Le chien');
INSERT INTO clients_myisam(nom, prenom) VALUES ('Casta', 'Laetitia');
```

**SELECT * FROM clients_myisam WHERE MATCH(nom,prenom)
AGAINST('Tintin');**

Pour les désactiver vous pouvez utiliser une fonction ou une opération neutre (+0, CONCAT "", ...).

Ou utiliser une clause d'usage d'index :

USE INDEX, **IGNORE INDEX** et FORCE INDEX affectent les index qui sont utilisés lors du choix de la méthode de sélection des lignes dans la table lors d'une jointure. Elles n'affectent pas l'utilisation de l'index dans les clauses ORDER BY ou GROUP BY.

```
SELECT * FROM villes USE INDEX (i_villes_nom_ville) WHERE nom_ville = 'Lyon';
```

Ou

```
SELECT * FROM villes IGNORE INDEX (i_villes_nom_ville) WHERE nom_ville = 'Lyon';
```

Cf aussi EXPLAIN.

CHAPITRE 4 - MISE A JOUR DES DONNEES

4.1 - PRÉSENTATION

Il existe 4 verbes (CRUD : Create, Read, Update, Delete) du LMD (Langage de manipulation de données) dont 3 permettant la mise à jour des données.

Verbe générique	Verbe SQL	Fonctionnalité
Create	INSERT	Pour ajouter un ou des enregistrements.
Read	SELECT	Pour extraire un ou des enregistrements.
Update	UPDATE	Pour modifier un ou des enregistrements.
Delete	DELETE	Pour supprimer un ou des enregistrements.

4.2 - INSERTION DE DONNÉES (INSERT)

4.2.1.1 - Présentation

L'ordre SQL INSERT permet d'ajouter un ou plusieurs enregistrements dans une table.

4.2.1.2 - Première syntaxe



Insérer un enregistrement.

```
INSERT INTO nom_de_table [(col1, col2, ...)]
VALUES (valeur1, valeur2, ...);
```

• Exemples

```
-- Insère le 21ème arrondissement
INSERT INTO villes(cp, nom_ville, id_pays) VALUES('75021', 'Paris 21', '033');
INSERT INTO villes(cp, nom_ville, site, photo, id_pays) VALUES('75022', 'Paris 22',
'www.paris22.fr', 'paris22.jpg', '033');
INSERT INTO villes(cp, nom_ville, site, photo, id_pays) VALUES('75023', 'Paris 23',
NULL, NULL, '033');
```

```
-- Insertion dans une table avec un auto_increment
-- Et récupération de la nouvelle valeur.
INSERT INTO clients(nom, prenom, cp)
VALUES('Tournesol', 'Bruno', '75011');
```

```
SELECT LAST_INSERT_ID();
```

```
-- Insertion dans une table ayant une colonne de type ENUM.
-- Une interface graphique présentera des boutons radio.
INSERT INTO personnels (nom ,categorie)
VALUES ('Tintin', 'M');
```

```
INSERT INTO personnels (nom ,categorie)
VALUES ('Casta', 'C');
```

```
-- Insertion dans une table ayant une colonne de type SET.
-- On insère la valeur de type SET entre ' (Quote simple).
-- Si plusieurs valeurs doivent être insérées on les sépare par une virgule.
-- Une interface graphique présentera une liste à choix multiples.
```

```
INSERT INTO personnes (nom ,hobbies ,cp)
VALUES ('Tintin', 'S', '75011');
```

```
INSERT INTO personnes (nom ,hobbies ,cp)
VALUES ('Haddock', 'C,T', '75011');
```

Détails sur last_insert_id()

http://dev.mysql.com/doc/refman/5.0/en/information-functions.html#function_last-insert-id

La fonction LAST_INSERT_ID() renvoie une valeur de type BIGINT (64-bit) qui représente la première valeur du dernier ordre INSERT effectué sur une table ayant une clé primaire de type INT AUTO_INCREMENT.

Ce qui veut dire que si vous exécutez 2 ordres SQL INSERT consécutifs sur deux tables ce sera le dernier ID généré qui sera renvoyé par la fonction (cf exemple 1).

Et que si vous exécutez un ordre SQL INSERT façon MYSQL pour insérer plusieurs enregistrements ce sera le premier ID généré par cet ordre SQL qui sera renvoyé par la fonction (cf exemple 2).

Exemple 1 :

```
INSERT INTO clients(liste de colonnes) VALUES(liste de valeurs);  
INSERT INTO cdes(liste de colonnes) VALUES(liste de valeurs);
```

```
SELECT LAST_INSERT_ID();
```

renverra le nouvel ID de la table cdes.

4.2.1.3 - Deuxième syntaxe

Insérer plusieurs enregistrements via un seul ordre INSERT (Syntaxe non standard propre à MySQL).

```
INSERT INTO nom_de_table(col1, col2, ...)  
VALUES (valeur1, valeur2, ...) , (valeur3, valeur4, ...) ...;
```

- **Exemple**

```
-- Insère Caen et Lille  
INSERT INTO villes (cp, nom_ville, id_pays)  
VALUES  
( '14000', 'Caen', '033'),  
( '59000', 'Lille', '033');
```

4.3 - INSERTION DE DONNÉES (REPLACE)

<http://dev.mysql.com/doc/refman/5.0/fr/replace.html>

REPLACE fonctionne comme INSERT, sauf que si une ligne dans la table à la même valeur qu'une nouvelle pour un index UNIQUE ou une PRIMARY KEY, la ligne sera supprimée avant que la nouvelle ne soit insérée.

- **Objectif**

Insérer des données en remplaçant – éventuellement - les données qui créent des conflits d'unicité.

C'est l'équivalent d'un INSERT si la ligne n'existe pas et d'un DELETE + INSERT si la ligne existe. D'ailleurs le message de MySQL est soit "1 row affected" ... soit "2 rows affected".

- **Syntaxes**

```
REPLACE INTO nom_de_table(colonne1 [, colonne2])
VALUES(valeur1 [, valeur2]);
```

```
REPLACE INTO nom_de_table [(colonne1 [, colonne2])]
SELECT colonne1 [, colonne2]
FROM nom_de_table
[WHERE condition];
```

- **Exemple 1**

```
REPLACE INTO villes(cp, nom_ville, id_pays)
VALUES('75031', 'Paris XXI', '033');
```

Si '75031', '...', '...' n'existe pas l'enregistrement est créé.

Si '75031', '...', '...' existe l'enregistrement est modifié (supprimé + inséré).

- **Exemple 2**

Si '7', 'Russie' existe ...

cet ordre modifiera l'enregistrement (sur clé primaire).

```
REPLACE INTO pays(id_pays, nom_pays)
VALUES('7', 'Russia');
```

Si '7', 'Russia' existe ...

cet ordre modifiera l'enregistrement (sur clé unique donc sur le nom du pays)

```
REPLACE INTO pays(id_pays, nom_pays)
VALUES('777', 'Russia');
```

4.4 - SUPPRESSION DE DONNÉES (DELETE)

- **Présentation**

L'ordre SQL DELETE permet de supprimer un ou plusieurs ou tous les enregistrements.

- **Syntaxe**



```
DELETE FROM nom_de_table  
[WHERE condition];
```

- **Exemples**

```
-- Supprime la ou les villes dont le cp est égal à 13000  
DELETE FROM villes_bis  
WHERE cp = '13000';
```

```
-- Supprime toutes des villes  
DELETE FROM villes_bis;
```

4.5 - SUPPRESSION DE TOUTES LES DONNÉES (TRUNCATE)

- **Présentation**

TRUNCATE (instruction du LDD) supprime toutes les données de la table. Le résultat est le même que DELETE FROM nom_de_table.

L'exécution est plus rapide.

En fait TRUNCATE droppe la table et la recrée.

De ce fait avec TRUNCATE les id auto_increment sont remis à 0.

- **Syntaxe**

```
TRUNCATE [TABLE] nom_de_table;
```

- **Exemple**

```
| TRUNCATE TABLE villes_bis;
```

Si la table villes_bis n'existe pas lancez les commandes suivantes :

```
| CREATE TABLE villes_bis AS SELECT * FROM villes;  
| SELECT * FROM villes_bis;  
| TRUNCATE villes_bis;  
| SELECT * FROM villes_bis;
```

4.6 - MODIFICATION DE DONNÉES (UPDATE)

- **Présentation**

L'ordre SQL UPDATE permet de modifier un ou plusieurs ou tous les enregistrements.

- **Syntaxe**



```
UPDATE nom_de_table  
SET col1 = valeur1 [, col2 = valeur2]  
[WHERE condition];
```

- **Exemples**

```
-- Met en majuscule les noms des villes  
UPDATE villes  
SET nom_ville = UPPER(nom_ville);
```

```
-- Modifie le nom de la ville en question  
UPDATE villes  
SET nom_ville = 'Paris'  
WHERE nom_ville = 'Paris 12';
```

```
-- Augmentation du prix de tous les produits d'1 euro  
UPDATE produits  
SET prix = prix + 1;
```

CHAPITRE 5 - L'EXTRACTION DES DONNEES (L'ORDRE SELECT)

5.1 - SELECT MONO TABLE SANS CONDITION

- **Syntaxe**



```
SELECT [DISTINCT] * | col1 [[AS] alias de colonne], col2 [[AS] alias de colonne], ...  
FROM nom_de_table [[AS] alias de table];
```

- **Exemples**

Tout sur les villes

```
SELECT *  
FROM villes;
```

```
SELECT *  
FROM villes v;
```

Les noms et cp des clients (Projection)

```
SELECT nom, cp  
FROM clients;
```

```
SELECT c.nom, c.cp AS "Code postal"  
FROM clients AS c;
```

```
SELECT DISTINCT cp  
FROM clients;
```

5.2 - EXTRAIRE UN CERTAIN NOMBRE D'ENREGISTREMENTS (RESTRICTION)

- **Syntaxe**



```
SELECT * | col1, col2, ...  
FROM nom_de_table  
LIMIT début, nombre;
```

Note : syntaxe spécifique à MySQL. Cf plus loin pour le WHERE.

- **Exemples**

Les 3 premiers :

```
SELECT *  
FROM clients  
LIMIT 0,3;
```

Les trois suivants :

```
SELECT *  
FROM clients  
LIMIT 3,3;
```


5.3 - LE TRI

- **Syntaxe**



```
SELECT * | col1, col2, ...  
FROM nom_de_table  
ORDER BY col1 [ASC | DESC] [, col2 ...];
```

Il est possible d'utiliser le rang de la colonne dans la clause ORDER.
Il est possible d'utiliser un alias la colonne dans la clause ORDER avec éventuellement des back-quotes [`].

- **Exemples**

```
SELECT id_client, nom  
FROM clients  
ORDER BY nom;
```

```
SELECT id_client, nom  
FROM clients  
ORDER BY 2;
```

```
SELECT id_client, nom AS `Nom du client`  
FROM clients  
ORDER BY `Nom du client`;
```

```
SELECT id_client, nom  
FROM clients  
ORDER BY nom DESC;
```

```
SELECT cp, nom, id_client  
FROM clients  
ORDER BY cp, nom;
```

Note : la clause ORDER BY est toujours la dernière clause d'un ordre SELECT sauf si vous ajoutez une clause LIMIT.

```
SELECT id_client, nom  
FROM clients  
ORDER BY nom  
LIMIT 0,3;
```

5.4 - LES OPÉRATEURS DE COMPARAISON

- Les opérateurs de comparaison



OPÉRATEUR	DESCRIPTION
=	Egal
!= , <>	Différent de
>	Supérieur à
>=	Supérieur ou égal à
<	Inférieur à
<=	Inférieur ou égal à

- Syntaxe

```
SELECT col1, col2, ...  
FROM nom_de_table  
WHERE condition;
```

- Exemples

```
SELECT *  
FROM PRODUITS  
WHERE prix = 12;
```

```
SELECT *  
FROM PRODUITS  
WHERE prix != 12;
```

```
SELECT *  
FROM PRODUITS  
WHERE prix > 5;
```

```
SELECT *  
FROM clients  
WHERE cp = '75011';
```

5.5 - LES OPÉRATEURS LOGIQUES

Tableau des opérateurs logiques



Opérateur	Description
AND ou &&	Et logique
OR ou	Ou logique
NOT	La négation logique

Rappel sur les opérateurs logiques : table de vérités

C1	C2	AND	OR	NOT C1	XOR
Vrai	Vrai	Vrai	Vrai	Faux	Faux
Vrai	Faux	Faux	Vrai	Faux	Vrai
Faux	Vrai	Faux	Vrai	Vrai	Vrai
Faux	Faux	Faux	Faux	Vrai	Vrai

AND : Le résultat est VRAI si les opérandes C1 et C2 sont VRAI.

OR : Le résultat est VRAI si un des opérandes C1 et C2 est VRAI.

NOT : Le résultat est VRAI l'opérande C1 FAUX.

XOR : Le résultat est VRAI si un et un seul des opérandes C1 et C2 est VRAI.

Exemples :

AND : pour être authentifié le pseudo et mot de passe doivent être valides.

OR : pour afficher les produits des catégories Eaux et Sodas il faut utiliser l'opérateur OR.

NOT : pour afficher les villes hors de France il faut utiliser l'opérateur NOT.

XOR : n'existe pas en SQL. Il faudra utiliser la clause exists et des requêtes ensemblistes.

- **Syntaxes**

```
SELECT col1, col2, ...  
FROM nom_de_table  
WHERE condition1 OR | AND condition2;
```

```
SELECT col1, col2, ...  
FROM nom_de_table  
WHERE NOT condition1;
```

- **Exemples**

Les Tintin du 75011.

```
SELECT *  
FROM clients  
WHERE nom = 'Tintin' AND cp = '75011';
```

Les clients du 11ème et du 12ème

```
SELECT *  
FROM clients  
WHERE cp = '75011' OR cp = '75012';
```

Les clients qui ne sont pas du 11

```
SELECT *  
FROM clients  
WHERE NOT cp = '75011';
```

L'opérateur de comparaison <> donnera le même résultat.
On utilise plutôt l'opérateur NOT avec IS et les opérateurs ensemblistes (cf plus loin).

5.6 - LES OPÉRATEURS ENSEMBLISTES

Tableau des opérateurs ensemblistes



Opérateur	Descripteur
[NOT] IN(V1, V2, ...)	Egal à n'importe quelle valeur d'une liste de valeurs
[NOT] BETWEEN x AND y	$x \geq \text{valeur} \leq y$
[NOT] LIKE	Comparer deux chaînes de caractères avec l'utilisation des caractères génériques : _ pour un caractère et % pour une chaîne de caractères
IS [NOT] NULL	Tester la valeur NULL (ou non) dans une colonne Le NULL est la valeur dite indéterminée indépendamment du type

5.6.1 - IN

- **Syntaxe**



```
SELECT col1, col2, ...  
FROM nom_de_table  
WHERE col1 IN (v1, v2, ...);
```

- **Exemple**

Les clients du 11ème et du 12ème

```
SELECT nom, cp  
FROM clients  
WHERE cp IN('75011','75012');
```

5.6.2 - BETWEEN

- **Syntaxe**



```
SELECT col1, col2, ...  
FROM nom_de_table  
WHERE colonne BETWEEN v1 AND v2;
```

- **Exemple**

Les produits d'une fourchette de prix

```
SELECT designation, prix  
FROM produits  
WHERE prix BETWEEN 1 AND 13;
```

5.6.3 - LIKE

- **Syntaxe**



```
SELECT col1, col2, ...  
FROM nom_de_table  
WHERE colonne LIKE '...%';
```

% : pour une chaîne de caractères,
_ : pour un caractère.

- **Exemple**

Les clients parisiens

```
SELECT nom, cp  
FROM clients  
WHERE cp LIKE '75%';
```

Selon le standard LIKE n'est applicable qu'à des chaînes de caractères. Mais parfois en pratique, à cause de transtypage implicite, l'opérateur LIKE peut s'appliquer à des numériques ou même des dates.

Cas particulier (échappement) :

```
SELECT * FROM tests WHERE nom LIKE '%\%%';
```


5.6.4 - IS NULL – IS NOT NULL

- **Syntaxe**



```
SELECT col1, col2, ...  
FROM nom_de_table  
WHERE colonne IS NULL | IS NOT NULL;
```

- **Exemples**

Les clients sans date de naissance

```
SELECT nom, date_naissance  
FROM clients  
WHERE date_naissance IS NULL;
```

Les clients avec une date de naissance

```
SELECT nom, date_naissance  
FROM clients  
WHERE date_naissance IS NOT NULL;
```

```
SELECT nom, date_naissance  
FROM clients  
WHERE date_naissance;
```

```
SELECT nom, date_naissance  
FROM clients  
WHERE date_naissance IS TRUE;
```

renvoient le même résultat.

5.7 - LES REQUÊTES CALCULÉES

5.7.1 - Numériques

- **Syntaxe**



```
SELECT col1, col2 opérateur col | opérande, ...
FROM nom_de_table
```

- **Exemple**

Calcul de la TVA

designation	Prix HT	TVA	Prix TTC
Evian	0.99	0.19	1.18
Badoit	2.31	0.45	2.76
Ruinard	131.77	25.83	157.60
Coca	2.09	0.41	2.50
Fanta	3.63	0.71	4.34
Crémant	6.16	1.21	7.37

```
SELECT designation,
prix `Prix HT`,
FORMAT(prix * 0.196, 2) `TVA`,
FORMAT(prix * 1.196, 2) `Prix TTC`
FROM produits p;
```

5.7.2 - Chaînes

- **Syntaxe**



```
SELECT CONCAT(col1, col2, ...)
FROM nom_de_table
```

- **Exemple**

L'affichage du nom et du prénom dans la même colonne d'affichage.

```
SELECT id_client, CONCAT(nom, "-", prenom) `Nom et prénom`
FROM clients
ORDER BY nom DESC;
```

Autre

```
SELECT id_client, CONCAT(prenom, " ", UPPER(nom)) `Nom et prénom`
FROM clients
WHERE cp = '75011'
ORDER BY nom ASC;
```

5.8 - QUELQUES FONCTIONS PROPRIÉTAIRES SUR LES CHAÎNES

<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html>

Syntaxe	Action	Exemple
Char = CHAR(n)	Retourne le caractère dont la valeur ASCII = n	SELECT CHR(65) ; affiche A
Int = ASCII(char)	Retourne le code ASCII	SELECT ASCII('A') ; affiche 65
Ch = CONCAT (ch1, ch2)	Retourne une seule chaîne (La concaténation)	SELECT CONCAT('PA','RIS') ; affiche PARIS
Ch = UPPER (ch) ou UCASE(ch)	Transforme ch en majuscule	SELECT UPPER('paris') ; affiche PARIS
Ch = LOWER (ch) ou LCASE(ch)	Retourne la chaîne en minuscule	SELECT LOWER('ORACLE') ; affiche oracle
Ch = REPLACE (ch1, ch2, ch3)	Cherche dans ch1 la ch2 puis la remplace par ch3	SELECT REPLACE('JACK et JUE','J','BL') ; affiche BLACK et BLUE
Ch = TRIM (ch) et RTRIM(ch) et LTRIM(ch)	TRIM supprime des caractères espace devant et derrière. RTRIM à droite LTRIM à gauche	SELECT TRIM(' PARIS ') ; affiche PARIS
Ch = SOUNDEX (ch)	Retourne la représentation phonétique de ch	SELECT nom FROM clients WHERE SOUNDEX(nom) = SOUNDEX('Dupaunt') ; trouvera Dupont, Dupond, ...
Ch = LEFT (ch, n)	Extrait n caractères	SELECT LEFT('75011',2) ;
Ch = RIGHT (ch, n)	Extrait n caractères	SELECT RIGHT('75011',3) ;
Ch = SUBSTR (ch,n[,m]) SUBSTRING(ch,n[,m]) SUBSTRING(ch, FROM n FOR m)	Extrait de la chaîne ch1 à partir de la position n, le nombre de caractères m. La 3 ^{ème} est standardisée Si le 2ème argument est négatif cela commence par la fin	SELECT SUBSTR('PARIS LA DEFENSE',7,2) ; retourne LA. Commence à 1. SELECT SUBSTR('Evian', -4) ; renvoie 'vian'
Int = LENGTH (ch) Ou OCTET_LENGTH (ch)	Calcule la longueur d'une chaîne en octets . A partir du code ASCII 128 ou 0x80 un caractère occupe 2 octets	SELECT LENGTH('Tintin') ; retourne 6 SELECT LENGTH('Elève') ; retourne 6
Int = CHAR_LENGTH (ch)	Calcule la longueur d'une chaîne en caractères	SELECT CHAR_LENGTH('Tintin') ; renvoie 6 SELECT CHAR_LENGTH('Elève') ; renvoie 5
Int = INSTR ('chaîne','chaîne recherchée')	Renvoie la position de la chaîne recherchée	SELECT 'AZERTY', INSTR('AZERTY', 'Z') ; retourne 2
Int = LOCATE ('Chaîne recherchée', 'Chaîne' [,position])	Renvoie la position de la chaîne recherchée à partir de position. Inversion des paramètres par rapport à INSTR	SELECT LOCATE('ve','Elève Eve', 6); renvoie 8
Int = POSITION ('sous-chaîne' IN 'chaîne');	Idem	SELECT POSITION('Z' IN 'AZERTY'); -- Renvoie 2

Exemples :

```
SELECT LENGTH('AZERTY'); -- Renvoie 6
```

```
SELECT LENGTH('Pepe'); -- Renvoie 4
```

```
SELECT LENGTH('Pépé'); -- Renvoie 6
```

```
SELECT CHAR_LENGTH('Pépé'); -- Renvoie 4
```

```
SELECT cp, CONCAT(prenom, ' ', nom) AS `Nom et prénom`, id_client  
FROM clients  
ORDER BY cp, nom;
```

```
SELECT cp, nom_ville, LEFT(cp,2) AS `Département` FROM villes;
```

```
SELECT SUBSTRING('AZERTY' FROM 1 FOR LENGTH('AZERTY')-1); -- Renvoie AZERT
```

```
SELECT DISTINCT nom, SOUNDEX(nom)  
FROM clients  
ORDER BY SOUNDEX(nom);
```

```
SELECT SOUNDEX('Dupont'), SOUNDEX('Dupond'), SOUNDEX('Doupond'), SOUNDEX('Dopond');  
-- Renvoie D153 pour les 4.
```

```
SELECT SOUNDEX('paris'), SOUNDEX('parisse'), SOUNDEX('parice'), SOUNDEX('pariss'),  
SOUNDEX('pariz'), SOUNDEX('pari'); -- Renvoie P620 pour tous sauf le dernier P600
```

SOUNDEX – Un autre exemple

```

DROP TABLE IF EXISTS cours.mots_soundex;
CREATE TABLE  cours.mots_soundex (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  mot varchar(45) NOT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO mots_soundex VALUES(1, 'Dupont');
INSERT INTO mots_soundex VALUES(2, 'Dupond');
INSERT INTO mots_soundex VALUES(3, 'Doupont');
INSERT INTO mots_soundex VALUES(4, 'Dopond');
INSERT INTO mots_soundex VALUES(5, 'Doupo');
INSERT INTO mots_soundex VALUES(6, 'Paris');
INSERT INTO mots_soundex VALUES(7, 'Pariss');
INSERT INTO mots_soundex VALUES(8, 'Pariz');
INSERT INTO mots_soundex VALUES(9, 'Parice');
INSERT INTO mots_soundex VALUES(10, 'Pari');

SELECT * FROM mots_soundex;

```

id	mot
1	Dupont
2	Dupond
3	Doupont
4	Dopond
5	Doupo
6	Paris
7	Pariss
8	Pariz
9	Parice
10	Pari

```

SELECT *
FROM mots_soundex
WHERE SOUNDEX(mot) = SOUNDEX('Paris');

```

id	mot
6	Paris
7	Pariss
8	Pariz
9	Parice

```

SELECT *
FROM mots_soundex
WHERE SOUNDEX(mot) = SOUNDEX('Dupont');

```

id	mot
1	Dupont
2	Dupond
3	Doupont
4	Dopond

5.9 - AUTRES FONCTIONS SUR LES CHAÎNES

<http://dev.mysql.com/doc/refman/5.0/en/string-functions.html>

Syntaxe	Action	Exemple
Ch = FORMAT(X,D)	Formate un nombre avec N chiffres après la virgule	SELECT FORMAT(123.456, 2); Affiche 123.45
Ch = BIN(n)	Renvoie la valeur binaire de n	SELECT BIN(3); Affiche 11
Ch = HEX(n)	Renvoie la valeur hexadécimale de n	SELECT HEX(10); Affiche A
Ch = OCT(n)	Renvoie la valeur octale de n	SELECT OCT(8); Affiche 10
Int = BIT_LENGTH(ch)	Renvoie la longueur de la chaîne en bits	SELECT BIT_LENGTH("a"); Affiche 8 SELECT BIT_LENGTH("é"); Affiche 16
Int = STRCMP(ch1, ch2)	Compare 2 chaînes de caractères. Renvoie 0 si elles sont égales. Renvoie 1 si ch1 > ch2. Renvoie -1 si ch1 < ch2 Plus de détails pour les jeux de caractères	SELECT STRCMP('Le loup', 'Le loup'); Affiche 0 SELECT STRCMP('Le loup', 'La louve'); Affiche 1 SELECT STRCMP('La louve', 'Le loup'); Affiche -1
Ch = CONCAT_WS("séparateur", ch1, ch2[, ch3,...])	Renvoie la concaténation des chaînes avec un caractère séparateur	SELECT CONCAT_WS(' ', 'Tintin', 'le reporter'); Affiche Tintin le reporter
Ch = INSERT(ch, départ, longueur, nouvelle chaîne)	Insère une chaîne de caractères	SELECT INSERT('Il fait beau', 8, 1, ' très '); Affiche "Il fait très beau"
Int = FIND_IN_SET('Chaîne recherchée', 'SET')	Renvoie la position de la chaîne dans le SET	Cf plus bas la page des exemples
MAKE_SET(bits, ch1, ch2[, ch3, ...]))	Renvoie une valeur de type SET	
EXPORT_SET()		
Ch = ELT(n, liste de chaînes)	Renvoie la chaîne n	Cf plus bas la page des exemples
REPEAT(ch, n);	Répète n fois la chaîne	SELECT REPEAT('Z', 5); Affiche ZZZZZ
SPACE(n)	Répète n fois ' '	SELECT " ", SPACE(3), " "; Affiche " * *"
Ch = RPAD(ch1, n, ch2) Ch = LPAD(ch1, n, ch2)	Complète la chaîne ch1 avec la chaîne ch2 jusqu'à ce que la chaîne résultante soit de taille n	SELECT RPAD('Le', 5, '*'); Affiche Le*** SELECT LPAD('21', 5, '0'); Affiche 00021
Ch = QUOTE(ch)	Renvoie la chaîne avec les quotes	SELECT QUOTE('Le Mas d\'Azil'); Affiche 'Le Mas d\'Azil'

		alors que SELECT 'Le Mas d\'Azil'; Affiche Le Mas d\'Azil
Ch = REVERSE('chaîne')	Inverse la chaîne	AZERTY → YTREZA

Exemples

FIND_IN_SET

Remarque : les valeurs sont "exactes" dans les SET donc pas d'espace.

```
| SELECT FIND_IN_SET('Mardi' , 'Lundi,Mardi,Mercredi,Jeudi,Samedi,Dimanche');
```

Affiche 2

ELT

```
| SELECT ELT(2 , 'Lundi', 'Mardi', 'Mercredi', 'Jeudi', 'Vendredi', 'Samedi',  
| 'Dimanche');
```

Affiche Mardi

5.10 - QUELQUES FONCTIONS PROPRIÉTAIRES SUR LES NUMÉRIQUES

<http://dev.mysql.com/doc/refman/5.0/en/numeric-functions.html>

Syntaxe	Action	Exemple
Num = ABS(n)	Extrait la valeur absolue de n	SELECT ABS(-125) [FROM dual]; Affiche 125
Num = TRUNCATE(m, n)	Tronque m à n décimales	SELECT TRUNCATE(12.3456, 2); Affiche 12.34
Num = ROUND(m, n)	Arrondit m à n décimales	SELECT ROUND(12.3456, 2); Affiche 12.35
Num = CEIL (n)	Calcule l'entier >= n	SELECT CEIL(15.4) [FROM dual]; Affiche 16
Num = FLOOR(n)	La partie entière de n	SELECT FLOOR(15.8) [FROM dual]; Affiche 15
Num = MOD (m, n)	Renvoie le reste de la division de m par n	SELECT MOD(25,7) [FROM dual]; Affiche 4
Num = POWER(m, n)	Calcule m à la puissance n	SELECT POWER(3,2) "Puissance" [FROM dual]; Affiche 9
Num = SQRT(n)	Racine carrée d'un nombre	SELECT SQRT(9); Affiche 3
Num = SIN(), COS(), TAN()	Calcule le sinus, cosinus tangente, etc.	
Num = RAND()	Renvoie un nombre aléatoire à virgule flottante compris entre 0 et 1.0	SELECT RAND(); SELECT ROUND(RAND() * 100); Affiche un entier compris entre 0 et 100
Num = CONV(valeur, base source, base cible)	Convertit une valeur d'une base vers une autre base	SELECT CONV(15, 10, 16); Affiche F SELECT CONV('F', 16, 10); Affiche 15 SELECT CONV('F', 16, 2); Affiche 1111 SELECT CONV(3, 10, 2); Affiche 11 SELECT CONV(10, 2, 10); Affiche 2

Cf aussi LOG(), LOG2, LOG10(), EXP(), ...

5.11 - QUELQUES FONCTIONS PROPRIÉTAIRES SUR LES DATES

<http://dev.mysql.com/doc/refman/5.0/en/date-and-time-functions.html>

Récupérer la date système



CURDATE()	Date du serveur
NOW()	Date et heure du serveur
SYSDATE()	Date et heure du serveur

Paramétrer les noms utilisés dans les dates (jour et mois)

<http://dev.mysql.com/doc/refman/5.0/en/locale-support.html>

```
-- Affiche la locale  
SELECT @@lc_time_names;
```

```
-- Permettra, pour la session, d'afficher les noms en français  
SET lc_time_names = 'fr_FR';
```

Ou dans my.ini dans le paragraphe [mysqld]

```
lc_time_names = fr_FR
```

cf aussi

```
default-time-zone = "Europe/Paris"
```

Fonctions sur les dates



les 3 premières

Syntaxe	Action	Exemple
Num = YEAR(d)	Extrait l'année	SELECT YEAR(date_cde), date_cde FROM cdes;
Num = MONTH(d)	Extrait le mois	SELECT MONTH(date_cde), date_cde FROM cdes;
Num = DAY(d)	Extrait le quantième du mois	SELECT DAY(date_cde), date_cde FROM cdes;
Ch = MONTHNAME(d)	Extrait le nom du mois	SELECT MONTHNAME(date_cde), date_cde FROM cdes;
Ch = DAYNAME(d)	Extrait le nom du jour	SELECT DAYNAME(date_cde), date_cde FROM cdes;
Num = DAYOFWEEK(d)	Extrait le jour de la semaine. Dimanche = 0	SELECT DAYOFWEEK(date_cde), date_cde FROM cdes;
Num = DAYOFYEAR(d)	Extrait le jour de l'année. 1er janvier = 1	SELECT DAYOFYEAR(date_cde), date_cde FROM cdes;
Num = WEEKOFYEAR(d)	Extrait la semaine de l'année. Commence à 1.	SELECT WEEKOFYEAR(date_cde), date_cde FROM cdes;

Formater une date

DATE_FORMAT(date, 'format')

Quelques formats :

Format	Description
%W	nom du jour
%a	nom abrégé du jour
%d	Quantième de 00 à 31
%e	Quantième de 0 à 31
%D	quantième avec suffixe anglais (st, th, rd)
%m	numéro du mois
%M	mois en toutes lettres
%b	nom du mois abrégé
%Y	année sur 4 chiffres
%y	année sur 2 chiffres

%k pour les heures, %i pour les minutes.

Cf la documentation officielle pour plus de détails.

Afficher la date du jour formatée.

```
SELECT NOW(), DATE_FORMAT(NOW(), '%W %d %M %Y');
```

Affiche : Tuesday 30 September 2008

Afficher une date au format français

Nom	Date de naissance USF	Date de naissance FRF
Buguet	1955-10-03	03/10/1955
Buguet	1948-08-22	22/08/1948
Fassiola	1985-05-10	10/05/1985
Napoléon-Bonaparte		
Fournier de Sarlovèse		

```
SELECT nom "Nom", date_naissance "Date de naissance USF",
       DATE_FORMAT(date_naissance, '%d/%m/%Y') "Date de naissance FRF"
FROM clients;
```

Calculs sur les dates

Syntaxe	Description
DATE_ADD(date, INTERVAL expression Type)	Ajoute à une date
DATE_SUB(date, INTERVAL expression Type)	Soustrait à une date
DATEDIFF(date1, date2)	Différence entre deux dates (MySQL utilise la date, pas les heures et minutes ...)

Il existe de nombreuses autres fonctions sur les dates. Cf la documentation officielle pour plus de détails.

Exemples

Ajoute 31 jours à la date de commande

```
| SELECT date_cde, DATE_ADD(date_cde, INTERVAL 31 DAY) FROM cdes;
```

Soustrait 24 heures à la date de commande

```
| SELECT date_cde, DATE_SUB(date_cde, INTERVAL 24 HOUR) FROM cdes;
```

Calcule le nombre de jours entre deux dates

```
| SELECT DATEDIFF(date_cde, NOW()), date_cde, NOW() FROM cdes;
```

La date d'il y a 30 jours

```
| SELECT DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

Les commandes des 30 derniers jours

```
| SELECT *  
| FROM cdes c  
| WHERE date_cde > DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

5.12 - AUTRES FONCTIONS

5.12.1 - IF

Il permet de faire un test sur une valeur d'une colonne, d'un calcul, ...

IF(condition, vrai, faux)

```
SELECT nom AS `Nom`,
  IF(date_naissance IS NOT NULL, date_naissance, 'Date de naissance inconnue') AS
  `Date de naissance`
FROM cours.clients;
```

Nom	Date de naissance
Fassiola	1985-05-10
Roux	1950-10-10
Tintin	Date de naissance inconnue
Sordi	Date de naissance inconnue
Muti	Date de naissance inconnue
Milou	1955-10-03

```
SELECT id_produit, designation, prix, IF(prix > 5, prix, 0)
FROM cours.produits;
```

id_produit	designation	prix	IF(prix > 10, prix, 0)
1	Evian	1.50	0.00
2	Graves	5.50	5.50
3	Badoit	1.20	0.00
4	Médoc	10.00	10.00

5.12.2 - IFNULL

Affiche la valeur de la colonne si la valeur est différente de NULL, autrement affiche la valeur du 2^{ème} paramètre.

IFNULL(colonne, 'Texte' | valeur)

Affiche 'Date inconnue' quand la date de naissance est NULL.

```
SELECT nom, IFNULL(date_naissance, 'Date inconnue')  
FROM clients;
```

Affiche 0 quand la qte_stockee est NULL.

```
SELECT designation, IFNULL(qte_stockee, 0)  
FROM produits;
```


5.12.3 - WHERE MATCH

Recherche sur des colonnes de type CHAR, VARCHAR ou TEXT indexé avec un index de type FULLTEXT.

```
SELECT * | colonne FROM nom_de_table WHERE MATCH (colonne) AGAINST  
( 'valeur_recherchee' );
```

Note : le moteur de table doit être MyISAM.

Recherche les articles dont le texte contient 'Tintin'. La 3ème colonne affiche un indicateur de pertinence (de 0 à 1).

```
SELECT id_article,  
texte_article,  
MATCH (texte_article) AGAINST ('Tintin')  
FROM articles  
WHERE MATCH (texte_article) AGAINST ('Tintin');
```

CHAPITRE 6 - LES VUES (VIEWS)

6.1 - CRÉATION, SUPPRESSION, MODIFICATION D'UNE VIEW

Une vue est une requête stockée - un ordre SELECT - dans la BD.

Une vue permet de ne pas avoir à ressaisir les requêtes statiques.

Elles permettent aussi de garantir l'intégrité des données et d'assurer la confidentialité.
Elles correspondent aux vues des Modèles Organisationnels des Données.
Elles en facilitent l'implémentation.

Pensez qu'il est impossible de donner des droits de lecture sur des données d'une table correspondant à certaines lignes et certaines colonnes sans créer une vue qui correspond au SELECT de ce type :

```
SELECT colonne1, colonne2  
FROM nomDeTable  
WHERE colonne3 = valeur.
```

Par exemple :

```
SELECT nom, prenom, email  
FROM clients  
WHERE cp LIKE '75%';
```

L'administrateur de la Base de Données, en concordance avec le résultat de la conception de la BD - faite par un concepteur Merise ou UML - , distribuera aux USERS – des développeurs ou des utilisateurs finals - , via la commande GRANT, des autorisations sur des vues plutôt que sur des tables.

On utilise une vue comme une table (on parle de **table virtuelle**); ainsi les ordres SELECT, les jointures, la création d'une vue à partir d'une autre vue, les insertions, les suppressions, les modifications, ..., sont des actions possibles sur les données de la BD à partir de views.

6.1.1 - Création d'une view



```
CREATE OR REPLACE VIEW nom_de_vue  
AS SELECT * | colonnes  
FROM nom_de_table  
[WHERE condition]  
[WITH CHECK OPTION];
```

CHECK OPTION permet de contrôler les MAJ à partir des vues (cf plus loin pour les détails).

Il est impossible d'insérer, de modifier ou de supprimer un enregistrement via la VIEW si l'enregistrement ne correspond pas au prédicat de celle-ci. Cf plus loin.

```
| CREATE VIEW clients_parisiens AS SELECT * FROM clients WHERE cp LIKE '75%';
```

6.1.2 - Suppression d'une view



```
DROP VIEW nom_de_vue;
```

6.1.3 - Modification d'une view

```
ALTER VIEW nom_de_la_vue  
AS  
SELECT ... ;
```

6.1.4 - Stockage

Avec MySQL les vues sont des fichiers .frm stockés dans le dossier /mysql/data/bd.

Exemple de code stocké :

```
CREATE VIEW v_villes_francaises
AS SELECT cp, nom_ville, id_pays FROM villes
WHERE id_pays = '33';
```

Stocke ceci :

```
TYPE=VIEW
query=select cours.villes.cp AS cp,cours.villes.nom_ville AS
nom_ville,cours.villes.id_pays AS id_pays from cours.villes where
(cours.villes.id_pays = \'33\')
md5=19b2c9196ce219a89e6b8708cf423f17
updatable=1
algorithm=0
definer_user=root
definer_host=localhost
suid=2
with_check_option=0
timestamp= timestamp=2011-11-11 11:11:11
create-version=1
source=SELECT cp, nom_ville, id_pays FROM villes
\nWHERE id_pays = \'33\'
client_cs_name=utf8
connection_cl_name=utf8_general_ci
view_body_utf8=select cours.villes.cp AS cp,cours.villes.nom_ville AS
nom_ville,cours.villes.id_pays AS id_pays from cours.villes where
(cours.villes.id_pays = \'33\')
```

AVEC CHECK OPTION cela rajoute \nWITH CHECK OPTION et with_check_option passe à 2.

En revanche updatable ne change pas même si vous créez une requête qui ne permet pas la mise à jour.

6.1.5 - Les vues matérialisées

Elles n'existent pas à proprement parlé avec MySQL.

Elles peuvent être en partie simulées ... avec l'instruction `CREATE TABLE AS SELECT ...`

Elles remplacent les anciens snapshots.

6.2 - LES VUES ET LES MISES À JOUR

Les views permettent, éventuellement, les mises à jour.

Pour cela il faut qu'elles soient mono-table, que toutes les colonnes NOT NULL soient présentes dans le SELECT et qu'aucun calcul n'ait été effectué dans le SELECT.

A contrario les requêtes suivantes dans la création d'une VIEW ne seront pas susceptibles de mises à jour :

jointures,
requêtes calculées,
requêtes agrégats,
etc.

- **Exemples**

-- Les clients parisiens avec jointure Clients X Villes
-- Aucun ajout possible

```
CREATE OR REPLACE VIEW clients_parisiens
AS SELECT nom, prenom, c.cp, nom_ville
FROM clients c, villes v
WHERE c.cp = v.cp
AND nom_ville LIKE 'PARIS%';
```

```
SELECT * FROM clients_parisiens;
```

-- Les clients parisiens mono-table avec toutes les colonnes NOT NULL
-- Tous les clients peuvent être ajoutés

```
CREATE OR REPLACE VIEW clients_parisiens_ajout
AS SELECT nom, prenom, cp
FROM clients
WHERE cp LIKE '75%';
```

```
SELECT * FROM clients_parisiens_ajout;
```

-- L'enregistrement sera inséré, toutes les colonnes NOT NULL sont renseignées

```
INSERT INTO clients_parisiens_ajout(nom, prenom, cp)
VALUES ('Casta', 'Laetitia', '75012');
```

-- Les clients parisiens mono-table avec toutes les colonnes NOT NULL mais CHECK
OPTION
-- Seuls les clients parisiens peuvent être ajoutés

```
CREATE OR REPLACE VIEW clients_parisiens_ajout
AS SELECT nom, prenom, cp
FROM clients
WHERE cp LIKE '75%'
WITH CHECK OPTION;
```

-- L'enregistrement ne sera pas inséré, le cp ne correspond pas au prédicat ; le message d'erreur est le suivant : CHECK OPTION failed 'cours.clients_parisiens_ajout'

```
INSERT INTO clients_parisiens_ajout(nom, prenom, cp, email)
VALUES ('Bullock', 'Sandra', '69000', 'sb@free.fr');
```

Pour la suppression de même mais il n'y a pas de message d'erreur seulement le message suivant : Query returned no resultset

```
DELETE
FROM clients_parisiens_ajout
WHERE nom = 'Sordi';
```

-- Les clients parisiens avec blocage
-- Aucun client ne peut être ajouté, il y a un calcul dans la requête

```
CREATE OR REPLACE VIEW clients_parisiens_ls
AS SELECT id_client + 0 "id_client", nom, prenom, cp
FROM clients
WHERE cp LIKE '75%';
```

-- L'enregistrement ne sera pas inséré, la View est en Lecture Seule

```
INSERT INTO clients_parisiens_ls(nom, prenom, cp)
VALUES ('Roberts', 'Julia', '75012');
```

Le message est : The target table clients_parisiens_ls of the INSERT is not insertable-into

CHAPITRE 7 - LES JOINTURES

7.1 - LE PRODUIT CARTÉSIEN ET LA JOINTURE

Le produit cartésien est la concaténation de tous les enregistrements d'une table T1 avec chaque enregistrement d'une table T2.

```
SELECT * | table1.colonne1 [, table2.colonne2 [, ...]]
FROM table1 CROSS JOIN table2;
```

Exemple de produit cartésien :

```
SELECT *
FROM villes CROSS JOIN pays
ORDER BY nom_ville;
```

cp	nom_ville	site	photo	id_pays	id_pays	nom_pays
14000	Caen	www.caen.fr	caen.jpg	33	33	France
14000	Caen	www.caen.fr	caen.jpg	33	39	Italie
14000	Caen	www.caen.fr	caen.jpg	33	35	Angleterre
14000	Caen	www.caen.fr	caen.jpg	33	1	USA
59000	Lille	www.lille.fr	lille.jpg	33	33	France
59000	Lille	www.lille.fr	lille.jpg	33	39	Italie
59000	Lille	www.lille.fr	lille.jpg	33	35	Angleterre
59000	Lille	www.lille.fr	lille.jpg	33	1	USA

Une jointure est un sous-produit du produit cartésien.

C'est la concaténation de tous les enregistrements d'une table T1 avec chaque enregistrement d'une table T2 quand une condition de jointure est satisfaite.

La requête suivante affiche chaque ville avec son pays.

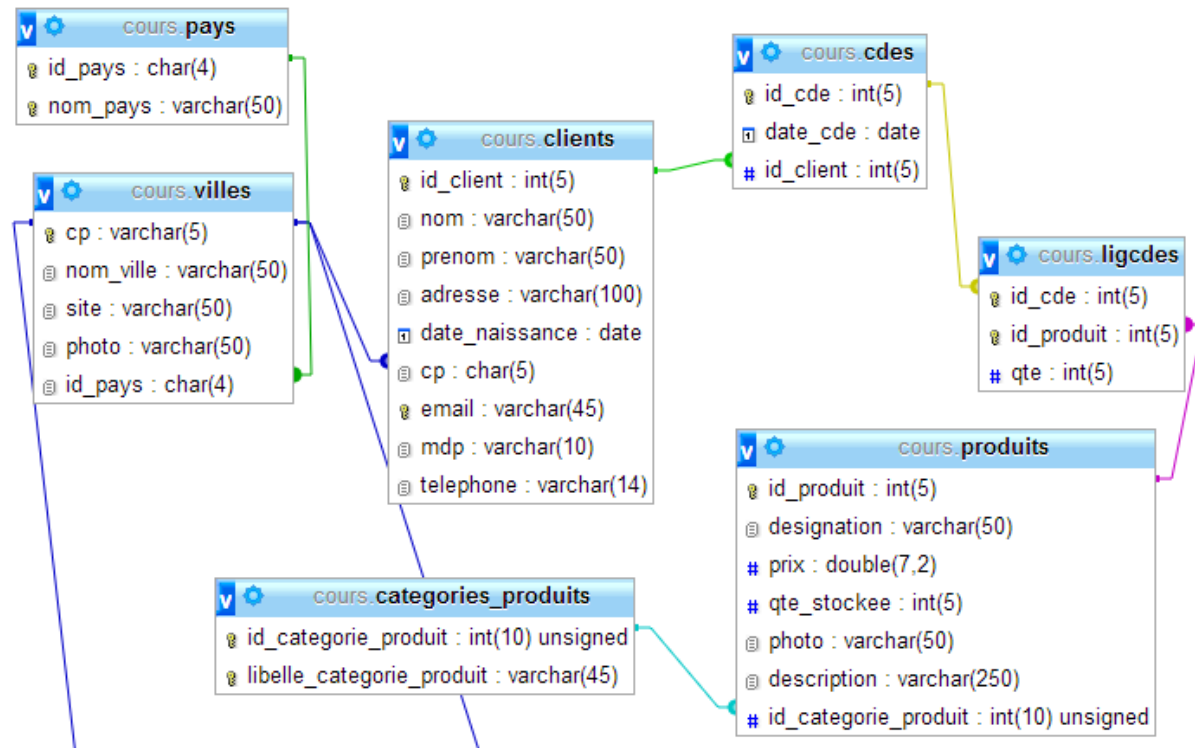
```
SELECT *
FROM villes v JOIN pays p
ON v.id_pays = p.id_pays
ORDER BY nom_ville;
```

Il existe plusieurs types de jointures :

- ✓ Equi-jointure quand l'opérateur de comparaison est =,
- ✓ La jointure naturelle,
- ✓ Theta-jointure quand l'opérateur de comparaison est différent de =,
- ✓ Auto-jointure quand la jointure s'effectue sur la même table,
- ✓ Jointures externes quand tous les enregistrements d'une table T1 sont dans le résultat même s'ils n'ont pas de correspondant dans une table T2.

Cf <http://dev.mysql.com/doc/refman/5.0/fr/join.html>

7.2 - LA CONCEPTION D'UNE JOINTURE



Démarche :

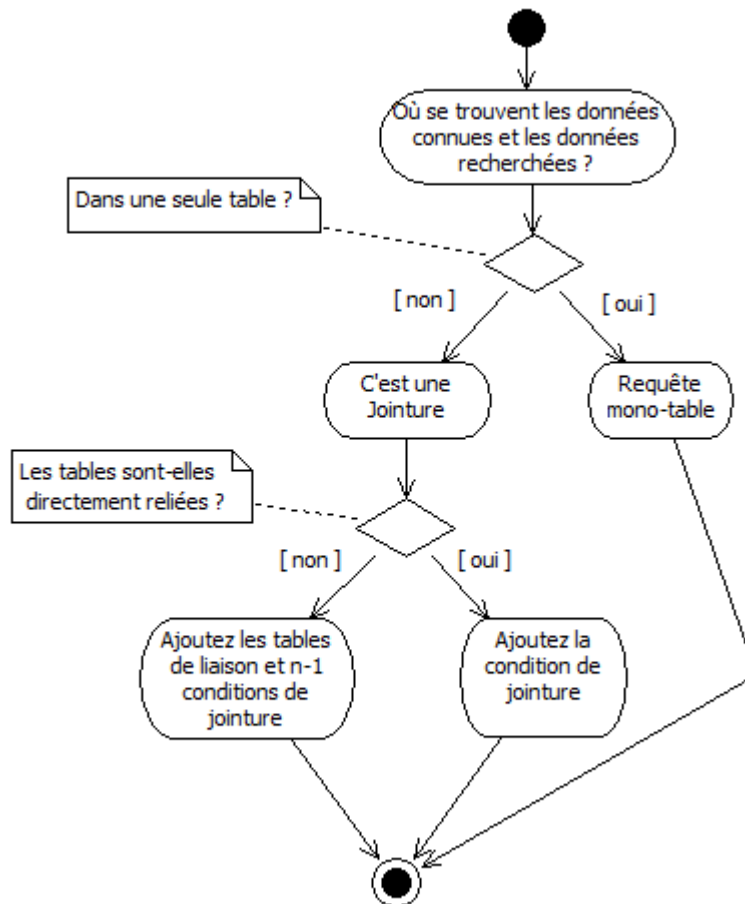
Les questions à se poser ... et les réponses à donner :

Où se trouvent les données connues et les données recherchées ?

Si elles se trouvent dans une table c'est une requête mono-table sinon c'est une jointure.

Si c'est une jointure, est-ce que les tables sont directement reliées entre elles ?

Si oui elles sont suffisantes sinon il faut ajouter après le FROM toutes les tables de liaison et autant de conditions de jointure.



Exemples d'énoncés et conception de requête :

Les noms des clients ?

Le nom du client est dans la table CLIENTS, donc c'est une requête mono-table.

Les clients romains ?

Le nom du client est dans la table CLIENTS, Rome est dans la table VILLES, donc c'est une jointure avec 2 tables et une condition de jointure + une restriction.

Les clients italiens ?

Le nom du client est dans la table CLIENTS, Italie est dans la table PAYS, donc c'est une jointure avec 2 tables et 2 conditions de jointures + une restriction.

Les buveurs de coca italiens ?

Le nom du client est dans la table CLIENTS, coca est dans la table PRODUITS, Italie est dans la table PAYS, donc c'est une jointure avec 3 tables et 5 conditions de jointures + deux restrictions.

7.3 - L'EQUI-JOINTURE

7.3.1 - Syntaxe simplifiée

Cette syntaxe n'est pas normalisée par l'ANSI mais fonctionne avec tous les SGBDR.

```
SELECT * | table1.col1, table1.col2, table2.col3, ...
FROM table1 , table2
WHERE condition_de_jointure;
```

Bien entendu il est possible, même conseillé, d'utiliser des alias de table.

- **Exemples**

Clients et villes : equi-jointure sans alias

id_client	nom	cp	nom_ville
1	Buguet	75011	Paris 11
4	Buguet	75011	Paris 11
5	Fassiola	75011	Paris 11
6	Roux	59000	Lille

```
SELECT clients.id_client, clients.nom, clients.cp, villes.nom_ville
FROM clients, villes
WHERE clients.cp = villes.cp;
```

Liste des produits dont la quantité commandée est supérieure ou égal à 5.

Désignation	Quantité
Evian	6
Evian	5
Evian	10
Graves	10

```
SELECT p.designation "Désignation", l.qte "Quantité"
FROM produits p , ligcdes l
WHERE p.id_produit = l.id_produit
AND l.qte >= 5;
```

Note : si l'on veut une seule fois le même produit il faut ajouter la clause DISTINCT après le SELECT et ne pas sélectionner la colonne qte :

```
SELECT DISTINCT p.designation "Désignation"
FROM produits p , ligcdes l
WHERE p.id_produit = l.id_produit
AND l.qte >= 5;
```

7.3.2 - Syntaxe ANSI



```
SELECT table1.col1, table1.col2, table2.col3, ...  
FROM table1 [INNER] JOIN table2  
ON table1.col1 = table2.col2;
```

- **Exemples**

Les villes et les clients (sans alias de tables)

```
SELECT villes.nom_ville, clients.nom  
FROM villes JOIN clients  
ON villes.cp = clients.cp;
```

Les pays et les villes (avec des alias de tables)

```
SELECT p.nom_pays, v.nom_ville  
FROM pays p JOIN villes v  
ON p.id_pays = v.id_pays;
```

Note :

STRAIGHT_JOIN est identique à JOIN, sauf que la table de gauche est toujours lue avant celle de droite. Cela peut être utilisé dans les cas (rares) où l'optimiseur de requêtes place les tables dans le mauvais ordre.

7.4 - LA JOINTURE NATURELLE

Jointure sur 2 ou N tables sans spécifier les colonnes de jointure. La colonne de jointure, lorsque l'on utilise *, n'est pas dupliquée.

Avec MySQL il faut que les noms des colonnes pour la jointure soient identiques même si vous n'avez pas créé de clé étrangère; il se fiche de l'identité des types des colonnes (VARCHAR -> CHAR, VARCHAR -> INTEGER, ...). Si vous avez créé une clé étrangère et que les noms sont différents il produira un produit cartésien.



```
SELECT liste de colonnes | *  
FROM table1 NATURAL JOIN table2;
```

Note : la syntaxe suivante donne le même résultat sur 2 tables

```
SELECT liste de colonnes | *  
FROM table1 JOIN table2  
ON (condition de jointure);
```

Deux tables

```
SELECT *  
FROM villes NATURAL JOIN clients;
```

Trois tables (en cascade ...)

```
SELECT nom_pays, nom_ville, nom  
FROM cours.pays NATURAL JOIN cours.villes NATURAL JOIN cours.clients;
```

Encore trois tables ... (avec une table de "liaison")

```
SELECT designation, qte, date_cde  
FROM cours.produits NATURAL JOIN cours.ligcdes NATURAL JOIN cours.cdes  
ORDER BY designation;
```


7.5 - AUTO-JOINTURE

C'est une jointure sur la même table. Deux exemples suivent.

Rapport hiérarchique : les vendeurs et leur chef direct

La table vendeurs

```
DROP TABLE IF EXISTS vendeurs;

CREATE TABLE vendeurs (
  id_vendeur int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom varchar(45) COLLATE utf8_unicode_ci NOT NULL,
  chef int(10) unsigned NOT NULL DEFAULT '0',
  cp char(5) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (id_vendeur),
  KEY FK_vendeurs_cp (cp),
  KEY FK_vendeurs_id_vendeur (chef),
  CONSTRAINT FK_vendeurs_id_vendeur FOREIGN KEY (chef) REFERENCES vendeurs
(id_vendeur) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

INSERT INTO vendeurs (id_vendeur, nom, chef, cp) VALUES
(1, 'Lucky', 1, '75011'),
(2, 'Dalton', 1, '75012'),
(3, 'Mickey', 1, '75012'),
(4, 'Donald', 2, '75011');
```

id_vendeur	nom	chef	cp
1	Lucky	1	75011
2	Dalton	1	75012
3	Mickey	1	75012
4	Donald	2	75011

L'auto-jointure : les vendeurs et leur chef direct

id_vendeur	Nom du vendeur	chef	Nom du chef
1	Lucky	1	Lucky
2	Dalton	1	Lucky
3	Mickey	1	Lucky
4	Donald	2	Dalton

```
CREATE VIEW vendeurs_et_chef AS
SELECT v1.id_vendeur, v1.nom "Nom du vendeur", v1.chef, v2.nom "Nom du chef"
FROM vendeurs v1 JOIN vendeurs v2
ON v1.chef = v2.id_vendeur;
```

```
SELECT * FROM vendeurs_et_chef v;
```

Note : pour mieux voir et mieux comprendre ajoutez la colonne v2.id_vendeur.

Championnat match aller (auto theta jointure)

Tout le monde rencontre tout le monde une seule fois.

ID_1	Nom_1	ID_2	Nom_2
1	Lucky	3	Mickey
1	Lucky	4	Donald
1	Lucky	2	Dalton
2	Dalton	3	Mickey
2	Dalton	4	Donald
3	Mickey	4	Donald

```
CREATE VIEW championnat_match_aller AS
SELECT v1.id_vendeur AS "ID_1",
v1.nom AS "Nom_1",
v2.id_vendeur AS "ID_2",
v2.nom AS "Nom_2"
FROM vendeurs v1 JOIN vendeurs v2
ON v1.id_vendeur < v2.id_vendeur
ORDER BY v1.id_vendeur;
```

```
SELECT * FROM championnat_match_aller;
```

Note : pour mieux voir et mieux comprendre commencez par le produit cartésien puis, les matches aller-retour ...

7.6 - AUTO-JOINTURE ET GÉNÉALOGIE

Objectif

Stocker un arbre généalogique et afficher des ascendants et des descendants directs.

La BD



Les tables

La première table contient les informations sur les personnes (id, nom, prenom, sexe). La deuxième table contient les liens de parenté, ie code parent + code enfant et le rôle (père ou mère). Donc en principe pour chaque enfant il y aura 2 lignes dans cette table.

```
DROP DATABASE IF EXISTS genealogies;

CREATE DATABASE genealogies
DEFAULT CHARACTER SET utf8
COLLATE utf8_unicode_ci;

USE genealogies;

SET FOREIGN_KEY_CHECKS = 0;

CREATE TABLE IF NOT EXISTS genealogies.personnes (
  id_personne int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom varchar(45) NOT NULL,
  prenom varchar(45) NOT NULL,
  sexe varchar(45) NOT NULL,
  PRIMARY KEY (id_personne)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO genealogies.personnes (id_personne, nom, prenom, sexe) VALUES
(1, 'Buguet', 'Pascal', 'M'),
(2, 'Buguet', 'Sophie', 'F'),
(3, 'Frick', 'Emmanuel', 'M'),
(4, 'Frick', 'Gérard', 'M'),
(5, 'Robinet', 'Geneviève', 'F'),
(6, 'Buguet', 'André', 'M'),
(7, 'Fassiola', 'Annabelle', 'F'),
(8, 'Roux', 'MJ', 'F'),
(9, 'Roux', 'Josette', 'F'),
(10, 'Roux', 'Arthur', 'M'),
(11, 'Loyet', 'Aline', 'M'),
(12, 'Robinet', 'André', 'M'),
```

```
(13, 'Buguet', 'Roger' 'M'),
(14, 'Puyol', 'Juliette' 'M');

CREATE TABLE IF NOT EXISTS genealogies.parentes (
  id_parent int(10) unsigned NOT NULL,
  id_enfant int(10) unsigned NOT NULL,
  role varchar(45) NOT NULL,
  PRIMARY KEY (id_parent,id_enfant)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE genealogies.parentes ADD CONSTRAINT FK_parentes_parent FOREIGN KEY
FK_parentes_parent (id_parent)
  REFERENCES personnes (id_personne)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT;

ALTER TABLE genealogies.parentes ADD CONSTRAINT FK_parentes_enfant FOREIGN KEY
FK_parentes_enfant (id_enfant)
  REFERENCES personnes (id_personne)
  ON DELETE RESTRICT
  ON UPDATE RESTRICT;

INSERT INTO parentes (id_parent, id_enfant, role) VALUES(1, 7, 'Père');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(2, 3, 'Mère');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(4, 3, 'Père');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(5, 1, 'Mère');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(5, 2, 'Mère');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(6, 1, 'Père');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(6, 2, 'Père');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(8, 7, 'Mère');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(9, 8, 'Mère');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(10, 8, 'Père');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(11, 5, 'Mère');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(12, 5, 'Père');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(13, 6, 'Père');
INSERT INTO parentes (id_parent, id_enfant, role) VALUES(14, 6, 'Mère');

SET FOREIGN_KEY_CHECKS = 1;
```

Les parents de quelqu'un :

ID enfant	ID parent	Prénom parent	Prénom parent
1	5	Geneviève	Robinet
1	6	André	Buguet

Equi-jointure entre les personnes (table personnes) et la table des liens de parenté (parentes) et condition de restriction sur la colonne id_enfant.

Les parents de la personne dont l'ID est égal à 1.

```
CREATE VIEW parents_de_1 AS
SELECT pa.id_enfant AS "ID enfant",
pe.id_personne AS "ID parent", pe.prenom AS "Prénom parent", pe.nom AS "Nom parent"
FROM personnes pe INNER JOIN parentes pa
ON pe.id_personne = pa.id_parent
AND pa.id_enfant = 1;

SELECT * FROM parents_de_1;
```

Les enfants de quelqu'un (donc les frères et sœurs éventuellement) :

ID parent	ID enfant	Prénom enfant	Nom parent
5	1	Pascal	Buguet
5	2	Sophie	Buguet

Equi-jointure entre les personnes (table personnes) et la table des liens de parenté (parentes) et condition de restriction sur la colonne id_parent.

Les enfants de la personne dont l'ID est égal à 5.

```
CREATE VIEW enfants_de_5 AS
SELECT pa.id_parent AS "ID parent",
pe.id_personne AS "ID enfant",
pe.prenom AS "Prénom enfant",
pe.nom AS "Nom enfant"
FROM personnes pe INNER JOIN parentes pa
ON pe.id_personne = pa.id_enfant
AND pa.id_parent = 5;

SELECT * FROM enfants_de_5;
```

La même avec le nom du parent affiché

ID parent	Prénom parent	Nom parent	ID enfant	Prénom enfant	Nom enfant
5	Geneviève	Robinet	1	Pascal	Buguet
5	Geneviève	Robinet	2	Sophie	Buguet

```
CREATE VIEW enfants_de_5_bis AS
SELECT pe2.id_personne AS "ID parent",
pe2.prenom AS "Prénom parent",
pe2.nom AS "Nom parent",
pe.id_personne AS "ID enfant",
pe.prenom AS "Prénom enfant",
pe.nom AS "Nom enfant"
FROM personnes pe INNER JOIN parentes pa
ON pe.id_personne = pa.id_enfant
INNER JOIN personnes pe2
ON pe2.id_personne = pa.id_parent
AND pa.id_parent = 5;

SELECT * FROM enfants_de_5_bis;
```

7.7 - LES JOINTURES EXTERNES

- **Objectif**

Une jointure externe permet d'extraire les enregistrements d'une table ainsi que ceux d'une deuxième table de jointure mais aussi ceux de la première table qui n'ont pas de correspondants dans la deuxième table.

Dans la table Villes il y a des villes qui n'ont pas de clients; ainsi une équi-jointure n'affiche que les villes où il y a des clients.

La jointure externe permettra d'afficher ainsi toutes les villes même celles sans clients.

Une jointure externe permettra avec une condition supplémentaire d'afficher les enregistrements sans correspondances.

Une jointure externe peut l'être à gauche ou à droite.

Une jointure externe **à gauche** ajoute un enregistrement NULL à droite (deuxième table) pour le concaténer à l'enregistrement de la table de gauche (première table) qui n'a pas de correspondants.

- **Syntaxe (ANSI)**



```
SELECT table1.col1, table1.col2, table2.col3 ...  
FROM table1 RIGHT | LEFT [OUTER] JOIN table2  
ON table1.col1 = table2.col2;
```

ou

```
SELECT table1.col1, table1.col2, table2.col3 ...  
FROM table1 NATURAL RIGHT | LEFT [OUTER] JOIN table2;
```

- Exemples

Toutes les villes avec leurs clients ainsi que celles qui n'ont pas de clients

```
SELECT nom_ville, nom
FROM villes v LEFT OUTER JOIN clients c
ON v.cp = c.cp
ORDER BY nom;
```

ou

```
SELECT nom_ville, nom
FROM villes v NATURAL LEFT OUTER JOIN clients c
ORDER BY nom;
```

nom_ville	nom
Vincennes	
Saint-Mandé	
Paris 11	Buguet
Paris 11	Buguet
Paris 11	Fassiola
Paris 12	Milou
Milan	Muti
Lille	Roux
Rome	Sordi
Paris 12	Tintin

Toutes les villes qui n'ont pas de clients

```
SELECT nom_ville, nom
FROM villes v LEFT OUTER JOIN clients c
ON v.cp = c.cp
WHERE nom IS NULL;
```

nom_ville	nom
Lyon	
Nanterre	
Neuilly	
Saint-Mandé	
Versailles	
Vincennes	

Les vendeurs et le chef

Si la table vendeurs est ainsi :

Eventuellement CREATE TABLE vendeurs_bis AS SELECT * FROM vendeurs si la table vendeurs contient une FK sur elle-même sur l'id du vendeur.

id_vendeur	nom	chef	cp
1	Lucky		75011
2	Dalton	1	75012
3	Mickey	1	75012
4	Donald	2	75011

Pour afficher tout le monde il faut une auto-jointure externe :

```
SELECT v1.id_vendeur, v1.nom_vendeur, v1.chef, v2.nom_vendeur "Nom du chef"
FROM vendeurs_bis v1 LEFT OUTER JOIN vendeurs_bis v2
ON v1.chef = v2.id_vendeur;
```

7.8 - PLUS LOIN AVEC LA SYNTAXE ANSI

Equi-jointure avec 3 tables

```
SELECT p.nom_pays, v.nom_ville, c.nom
FROM pays p JOIN villes v
JOIN clients c
ON p.id_pays = v.id_pays
AND v.cp = c.cp
ORDER BY p.nom_pays, v.nom_ville, c.nom;
```

ou

```
SELECT p.nom_pays, v.nom_ville, c.nom
FROM pays p JOIN villes v
ON p.id_pays = v.id_pays
JOIN clients c
ON v.cp = c.cp
ORDER BY p.nom_pays, v.nom_ville, c.nom;
```

Jointure externe avec 3 tables

```
SELECT p.nom_pays, v.nom_ville, c.nom
FROM pays p LEFT OUTER JOIN villes v
ON p.id_pays = v.id_pays
LEFT OUTER JOIN clients c
ON v.cp = c.cp
WHERE UPPER(p.nom_pays) = 'FRANCE'
ORDER BY p.nom_pays, v.nom_ville, c.nom;
```

Les vendeurs et leur chef (Auto-jointure externe)

id_vendeur	nom	chef	Nom du chef
1	Lucky	0	
2	Dalton	1	Lucky
3	Mickey	1	Lucky
4	Donald	2	Dalton

```
SELECT v1.id_vendeur, v1.nom_vendeur, v1.chef, v2.nom_vendeur "Nom du chef"
FROM vendeurs v1 LEFT OUTER JOIN vendeurs v2
ON v1.chef = v2.id_vendeur;
```

7.9 - LE PRODUIT CARTÉSIEN

7.9.1 - Définition

Le produit cartésien est la concaténation de chaque ligne d'une table avec toutes les autres lignes d'une autre table.

D'emploi rare il peut être très utile pour des "saisies de masse".

7.9.2 - Premier exemple

Admettons deux tables : Listes (Listes candidates) et Bureaux (Bureaux de votes).

```
CREATE TABLE listes (
  id_liste int(10) unsigned NOT NULL default '0',
  nom_liste varchar(50) NOT NULL default '',
  PRIMARY KEY (id_liste)
) ENGINE=InnoDB;
```

```
INSERT INTO listes (id_liste, nom_liste) VALUES
(1, 'MODEM'),
(2, 'UMP'),
(3, 'PS'),
(4, 'PCF'),
(5, 'VERTS');
```

```
CREATE TABLE bureaux (
  id_bureau int(10) unsigned NOT NULL default '0',
  nom_bureau varchar(50) NOT NULL default '',
  adresse varchar(100) NOT NULL default '',
  inscrits int(10) unsigned NOT NULL default '0',
  PRIMARY KEY (id_bureau)
) ENGINE=InnoDB;
```

```
INSERT INTO bureaux (id_bureau, nom_bureau, adresse, inscrits) VALUES
(1, 'Mairie', 'Place Léon Blum', 1101),
(2, 'Ecole Roquette', 'Rue de la Roquette', 1200),
(3, 'Ecole Folie Régnault', 'Rue de la Folie Régnault', 1001),
(4, 'Ecole Faidherbe', 'Rue Faidherbe', 1000),
(11, 'Ecole élémentaire Pihet', 'Rue Pihet', 800),
(12, 'Ecole Popincourt', 'Rue Popincourt', 1300),
(52, 'Ecole Roubo', 'Rue Roubo', 800);
```

Le produit cartésien sera

```
SELECT id_bureau, nom_bureau, adresse, inscrits, id_liste, nom_liste
FROM bureaux, listes;
```

Vous créez une table nommée 'resultats' ainsi

```
CREATE TABLE resultats
AS
SELECT id_bureau, nom_bureau, adresse, inscrits, id_liste, nom_liste, 0 "Voix"
FROM bureaux, listes;
```

Il ne restera plus qu'à saisir les résultats dans la colonne voix. Dans une commune comme Paris où il y a environ 1000 bureaux de votes et de 10 à 20 candidats ou listes par élection cela permettra de générer de 10 000 à 20000 enregistrements à chaque élection à partir d'une saisie d'environ 1000 enregistrements. Donc un gain de 90%.

7.9.3 - Deuxième exemple

Admettons la table 'Fourchettes' de prix

```
CREATE TABLE fourchettes (
  plancher int(10) unsigned NOT NULL default '0',
  plafond int(10) unsigned NOT NULL default '0',
  libelle VARCHAR(45) NOT NULL default '',
  PRIMARY KEY (plancher, plafond)
) ENGINE=InnoDB;
```

```
INSERT INTO fourchettes (plancher, plafond, libelle) VALUES
(0, 10, 'Economique'),
(11, 100, 'Moyen'),
(101, 1000, 'Cher');
```

La requête qui permet d'afficher les produits et leur libellé de fourchette.

```
SELECT designation, prix, libelle
FROM produits, fourchettes
WHERE prix BETWEEN plancher AND plafond
ORDER BY plancher, prix;
```

designation	prix	libelle
Evian	0,99	Economique
Coca	1	Economique
Badoit	1,3	Economique
Picpoul	4,95	Economique
Crémant	4,99	Economique
Graves	15,94	Moyen
Ruinard	131,77	Cher
Dom Pérignon	165	Cher

CHAPITRE 8 - LES FONCTIONS AGREGATS

8.1 - LES FONCTIONS AGRÉGATS

- **Objectif**

Faire des calculs sur des ensembles ou sous-ensembles d'enregistrements.
Ce sont principalement des fonctions statistiques.

- **Syntaxe**

```
SELECT fonction_agregat([DISTINCT | ALL] col1 | *), ...
FROM nom_de_table
[WHERE ...]
[ORDER BY ...]
[LIMIT ...];
```

- **Fonctions**



Fonction	Description
COUNT(*)	Nombre de lignes d'une table ou satisfaisant une condition.
COUNT([DISTINCT ALL] colonne)	Nombre de valeurs NOT NULL de la colonne.
SUM([DISTINCT ALL] colonne)	Somme des valeurs de la colonne. La colonne est nécessairement numérique. Avec DISTINCT somme les valeurs uniques (sans les doublons).
AVG([DISTINCT ALL] colonne)	Moyenne des valeurs d'une colonne. Même remarque que précédemment.
MAX([DISTINCT ALL] colonne)	Maximum des valeurs de la colonne. Même remarque que précédemment.
MIN([DISTINCT ALL] colonne)	Minimum des valeurs de la colonne. Même remarque que précédemment.
STDDEV([DISTINCT ALL] colonne)	Ecart type des valeurs de la colonne. Même remarque que précédemment.
VARIANCE([DISTINCT ALL] colonne)	La variance des valeurs de la colonne. Même remarque que précédemment.

Note : en principe ces fonctions sont utilisées sur des colonnes de type numériques mais MIN et MAX peuvent être utilisées sur des colonnes d'autres types.

- **Exemples**

Nombre de villes

```
SELECT COUNT(*) "Nombre de villes"  
FROM villes;
```

Nombre de clients dont on connaît la date de naissance

```
SELECT COUNT(date_naissance)  
FROM clients;
```

Nombre de noms de clients différents

```
SELECT COUNT(DISTINCT nom) "Nombre de noms de client différents"  
FROM clients;
```

Moyenne des prix

```
SELECT AVG(prix) "Moyenne des prix"  
FROM produits;
```

Prix le plus élevé

```
SELECT MAX(prix) "Prix le plus élevé"  
FROM produits;
```

Nombre de clients résidant dans une ville

```
SELECT COUNT(*) "Nombre de clients"  
FROM clients c INNER JOIN villes v  
ON c.cp = v.cp  
WHERE UPPER(v.nom_ville) = 'LILLE';
```


8.2 - LA CLAUSE GROUP BY

Regrouper les enregistrements lorsque l'on applique une fonction agrégat.

- **Syntaxe**



```
SELECT fonction_agregat(colonne | *) [, colonne | fonction_agregat() ]  
FROM nom_de_table  
[WHERE ...]  
GROUP BY colonne, ...  
[ORDER BY ...]  
[LIMIT ...];
```

Le SELECT ne doit contenir que des fonctions agrégats ou des colonnes présentes dans la clause GROUP BY.

- **Exemples**

Nombre de clients par cp.

cp	Nombre de clients
59000	1
75011	5
75012	2
99391	1
99392	1

```
SELECT cp, COUNT(*) "Nombre de clients"  
FROM clients  
GROUP BY cp;
```

Nombre de clients par ville.

Ville	Nombre de clients
Lille	1
Milan	1
Paris 11	5
Paris 12	2
Rome	1

```
SELECT v.nom_ville "Ville", COUNT(*) "Nombre de clients"  
FROM clients c INNER JOIN villes v  
ON c.cp = v.cp  
GROUP BY v.nom_ville;
```

Note : pour trier en fonction du nombre de clients ajoutez ORDER BY 2 ou ORDER BY Nombre de clients (avec des backquotes).

Nombre de commandes par année par client

Année	ID Client	Nom	Nombre de commandes
2000	1	Buguet	1
2005	1	Buguet	1
2005	3	Fassiola	1
2008	1	Buguet	1
2011	3	Fassiola	1
2011	4	Roux	1
2012	2	Buguet	4
2013	1	Buguet	1

```

SELECT YEAR(cd.date_cde) "Année", c.id_client "ID Client", c.nom "Nom",
COUNT(*) "Nombre de commandes"
FROM clients c INNER JOIN cdes cd
ON c.id_client = cd.id_client
GROUP BY c.id_client, c.nom, YEAR(cd.date_cde)
ORDER BY YEAR(cd.date_cde);

```

Nombre de villes par pays trié par ordre décroissant

Pays	Nombre de villes
France	13
Italie	2
Angleterre	0
USA	0
Erythrée	0

```

SELECT p.nom_pays "Pays", COUNT(v.nom_ville) "Nombre de villes"
FROM pays p LEFT JOIN villes v
ON p.id_pays = v.id_pays
GROUP BY p.nom_pays
ORDER BY 2 DESC;

```

8.3 - LA CLAUSE HAVING

Appliquer une condition à un groupe.

- **Syntaxe**

```
SELECT fonction_agregat(col1 | *) [, colonne | fonction_agregat() ]
FROM nom_de_table
[WHERE ...]
GROUP BY col1, ...
HAVING fonction_agregat(col1 | *) opérateur de comparaison Valeur
[ORDER BY ...]
[LIMIT ...];
```

- **Exemples**

Nombre de clients en fonction du CP quand ce nombre est supérieur à 1

cp	Nombre de clients
75011	5
75012	2

```
SELECT cp, COUNT(*) "Nombre de clients"
FROM clients
GROUP BY cp
HAVING COUNT(cp) > 1;
```

ou

```
SELECT cp, COUNT(*) `Nombre de clients`
FROM clients
GROUP BY cp
HAVING `Nombre de clients` > 1;
```

Clients ayant passé commande plus d'une fois en 2005

Année	Code client	Nom	Nombre de commandes
2005	1	Buguet	4

```
SELECT YEAR(cd.date_cde) "Année", c.id_client "Code client", c.nom "Nom", COUNT(*)
"Nombre de commandes"
FROM clients c INNER JOIN cdes cd
ON c.id_client = cd.id_client
WHERE YEAR(cd.date_cde) = '2005'
GROUP BY c.id_client, nom, YEAR(cd.date_cde)
HAVING COUNT(*) > 1;
```

8.4 - LA CLAUSE WITH ROLLUP

La clause WITH ROLLUP (non standard) produit un effet complexe avec la clause GROUP BY. Elle produit un "break", une rupture sur le regroupement. La requête produit une ligne qui est un "extra super-aggregate summary".

Utile pour créer des rapports.

Moyenne des prix par Catégorie et Moyenne générale des prix :

id_categorie	Moyenne
1	1,65
2	3,26
3	79,67
	30,76

```
SELECT id_categorie, AVG(prix) AS Moyenne
FROM produits
GROUP BY id_categorie
WITH ROLLUP;
```

Chiffres d'affaires par produit et CA global :

Désignation	CA
Badoit	18
Coca-Cola	14,4
Coca-Cola light	12,25
Dom Pérignon	141
Evian	62
Graves	32
Ruinard	604
Vichy Célestins	48
	923,7

```
SELECT p.designation AS `Désignation`, SUM(p.prix * l.qte) AS CA
FROM produits p INNER JOIN ligcdes l
ON p.id_produit = l.id_produit
GROUP BY p.designation
WITH ROLLUP;
```

Double rupture (CA par catégorie et par produit)

ID Catégorie	Désignation	CA
1	Badoit	28,12
1	Evian	27,72
1		55,84
2	Coca	58,21
2		58,21
3	Dom Pérignon	165
3	Graves	191,28
3	Ruinard	131,77
3		487,45
		601,5

```

SELECT p.id_categorie AS `ID Catégorie`,
p.designation AS `Désignation`,
SUM(p.prix * l.qte) AS CA
FROM produits p INNER JOIN ligcdes l
ON p.id_produit = l.id_produit
GROUP BY p.id_categorie, p.designation
WITH ROLLUP;

```

8.5 - GROUP_CONCAT

GROUP_CONCAT (non standard) permet de concaténer les valeurs d'une colonne – avec par défaut une virgule comme séparateur – correspondant à un regroupement (GROUP BY).

Syntaxe

```
SELECT colonneDeRegroupement,  
GROUP_CONCAT(colonne [ ORDER BY colonne [ SEPARATOR 'séparateur']]) [AS alias]  
FROM table  
GROUP BY colonneDeRegroupement;
```

Afficher la liste des clients correspondant à un CP.

Pour chaque CP on affiche la liste des noms des clients.

cp	liste				
59000	Roux				
75011	Buguet	Buguet	Fassiola	Napoléon-Bonaparte	Fournier de Sarlovèse
75012	Tintin	Milou			
99391	Sordi				
99392	Muti				

```
SELECT cp, GROUP_CONCAT(nom) AS liste  
FROM clients  
GROUP BY cp;
```

Afficher la liste des clients correspondant à une ville.

Pour chaque Ville on affiche la liste des noms des clients dans la deuxième colonne.

nom_ville	clients
Lille	Roux
Paris 11	Buguet+Napoléon-Bonaparte+Fournier de Sarlovèse+Fassiola+Buguet
Paris 12	Tintin+Milou
Rome	Sordi
Milan	Muti

```
SELECT v.nom_ville, GROUP_CONCAT(c.nom SEPARATOR '+') AS clients
FROM villes v INNER JOIN clients c
ON v.cp = c.cp
GROUP BY c.cp, v.nom_ville;
```


Afficher la liste des clients en fonction de la première lettre de leur nom.

Les noms commençant par la même lettre sont concaténés dans la deuxième colonne.

Debut	Liste
B	Buguet+Buguet
F	Fassiola+Fournier de Sarlovèse
M	Milou+Muti
N	Napoléon-Bonaparte
R	Roux
S	Sordi
T	Tintin

```
SELECT LEFT(nom, 1) AS Debut,  
GROUP_CONCAT(nom ORDER BY nom SEPARATOR '+') AS Liste  
FROM clients  
GROUP BY Debut;
```

CHAPITRE 9 - LES REQUETES ENSEMBLISTES

9.1 - PRINCIPES

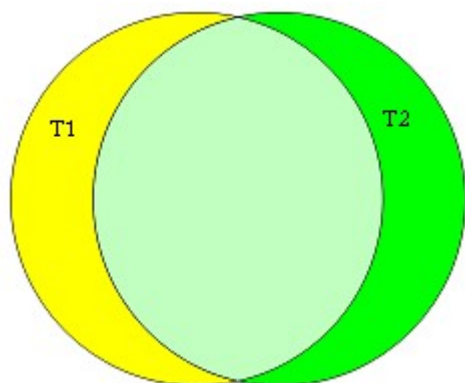
Les opérations et les opérateurs ensemblistes standards sont :

- ✓ L'UNION (**UNION**) qui renvoie l'ensemble des enregistrements des tables de l'union,
- ✓ L'INTERSECTION (INTERSECT) qui renvoie l'ensemble des enregistrements communs aux tables de l'intersection,
- ✓ La différence (MINUS ou EXCEPT) qui renvoie l'ensemble des enregistrements qui appartiennent à une table et seulement à cette table.

Ces opérateurs doivent avoir comme opérandes des **schémas identiques**.

Les SGBDR n'implémentent pas nécessairement toutes ces opérateurs.
C'est le cas de MySQL qui n'implémente que l'UNION.

Admettons deux tables T1 et T2.



- **Union : T1 UNION T2**

Tout.

- **Intersection : T1 INTERSECT T2**

L'ensemble vert clair.

- **Différence : T1 – T2**

L'ensemble jaune.

- **Différence : T2 – T1**

L'ensemble vert.

9.2 - UNION

- **Syntaxe**



```
SELECT schéma FROM table1  
UNION  
SELECT schéma FROM table2;
```

- **Exemple**

Les clients qui résident en France et à l'étranger

Admettons que la table Villes contienne des noms de villes étrangères dont le CP commence par 99.
Et quelques clients habitant ces villes.

Admettons la table **clients_etrangers** créée avec une requête du style :

```
CREATE TABLE clients_etrangers  
AS SELECT * FROM clients  
WHERE cp LIKE '99%';
```

Ce sont des clients qui résident aussi bien en France qu'à l'étranger.
Ajoutons dans la table clients_etrangers un nouvel enregistrement (ce sera un client qui ne réside qu'à l'étranger).

```
INSERT INTO clients_etrangers(id_client, nom, prenom, cp)  
VALUES (22, 'Loren', 'Sofia', '99391');
```

```
SELECT nom, cp  
FROM clients  
UNION  
SELECT nom, cp  
FROM clients_etrangers  
ORDER BY cp, nom;
```

Cette requête affiche l'ensemble des clients présents dans la table [clients] ainsi que ceux de la table [clients_etrangers] et élimine les lignes en double.

9.3 - INTERSECTION (N'EXISTE PAS EN MySQL)

- **Syntaxe SQL Standard**

```
SELECT schéma FROM table1  
INTERSECT  
SELECT schéma FROM table2;
```

- **Exemple en MySQL avec une jointure ou un SELECT imbriqué.**

Les clients communs à Clients et clients_etrangers

Equi-Jointure (Mais on ne récupérera que les colonnes d'une table).

On joint – avec une équi-jointure - les enregistrements qui sont aussi bien dans la table [clients] que dans la table [clients_etrangers].

```
SELECT c.id_client, c.nom, c.prenom, c.cp  
FROM clients c INNER JOIN clients_etrangers ce  
ON c.id_client = ce.id_client;
```

Requête imbriquée.

On récupère en premier – dans la sous-requête - les ID des clients étrangers puis à partir de cette liste d'ID on récupère la liste des clients qui possède cet ID dans la table [clients].

```
SELECT c.id_client, c.nom, c.prenom, c.cp  
FROM clients c  
WHERE c.id_client  
IN (SELECT ce.id_client  
    FROM clients_etrangers ce  
    );
```

9.4 - DIFFÉRENCE (N'EXISTE PAS EN MySQL)

- **Syntaxe SQL Standard**

```
SELECT schéma FROM table1 ...  
MINUS | EXCEPT  
SELECT schéma FROM table2 ...;
```

- **Exemples en MySQL avec un SELECT imbriqué ou une jointure.**

Les clients strictement français

Avec une jointure

On exécute une equi-jointure externe sur les tables [clients] et [clients_etrangers], ensuite on extrait seulement les lignes qui n'ont pas de correspondant dans la table [clients_etrangers].

```
SELECT c.id_client, c.nom, ce.id_client, ce.nom  
FROM clients c LEFT JOIN clients_etrangers ce  
ON c.id_client = ce.id_client  
WHERE ce.id_client IS NULL  
ORDER BY c.id_client;
```

Avec une requête imbriquée

C'est plus simple!

On récupère d'abord dans la sous-requête les ID des clients étrangers.

Ensuite on récupère dans la table [clients] toutes les lignes sauf celles n'ont pas l'ID d'un client étranger.

```
SELECT *  
FROM clients  
WHERE id_client  
NOT IN (SELECT id_client FROM clients_etrangers);
```

Les clients strictement étrangers

Avec une jointure

```
SELECT ce.id_client, ce.nom
FROM clients_etrangers ce LEFT JOIN clients c
ON ce.id_client = c.id_client
WHERE c.id_client IS NULL
ORDER BY ce.id_client;
```

Avec une requête imbriquée

```
SELECT *
FROM clients_etrangers
WHERE id_client
NOT IN (SELECT id_client FROM clients);
```

Note : en Oracle ce serait :

```
SELECT * FROM clients
MINUS
SELECT * FROM clients_etrangers;
```


Les villes (plus exactement les CP) où il n'y a pas de clients

```
SELECT DISTINCT v.cp
FROM villes v
LEFT JOIN clients c ON v.cp = c.cp
WHERE c.cp IS NULL;
```

Ou (cf requêtes imbriquées)

```
SELECT cp
FROM villes
WHERE cp
NOT IN (SELECT cp FROM clients);
```

En Oracle ce serait :

```
SELECT cp FROM villes
MINUS
SELECT cp FROM clients;
```

Si l'on veut avoir la liste des villes il faut faire une jointure pour présenter 2 schémas identiques :

```
SELECT DISTINCT v.cp, v.nom_ville
FROM villes v
LEFT JOIN clients c ON v.cp = c.cp
WHERE c.cp IS NULL;
```

```
SELECT cp, nom_ville
FROM villes
WHERE cp
NOT IN (SELECT cp FROM clients);
```

CHAPITRE 10 - LES REQUETES IMBRIQUEES

10.1 - PRÉSENTATION

Une requête imbriquée est une requête qui utilise dans sa clause WHERE le résultat d'une autre requête (une sous-requête).

Les requêtes imbriquées sont proches des jointures à la différence que l'on ne peut afficher que les colonnes du premier SELECT.

Une sous-requête peut renvoyer :

- ✓ Une seule ligne et une seule colonne (une seule valeur),
- ✓ Une seule colonne (plusieurs valeurs),
- ✓ Une seule ligne (plusieurs valeurs),
- ✓ Plusieurs lignes et plusieurs colonnes (plusieurs valeurs).

10.2 - FORMAT 1 : LA SOUS-REQUÊTE RENVOIE UN SEUL RÉSULTAT

- **Syntaxe**



```
SELECT col1, col2, ...  
FROM table1  
WHERE colonne =  
    (SELECT colonne  
     FROM table2  
     WHERE condition  
    );
```

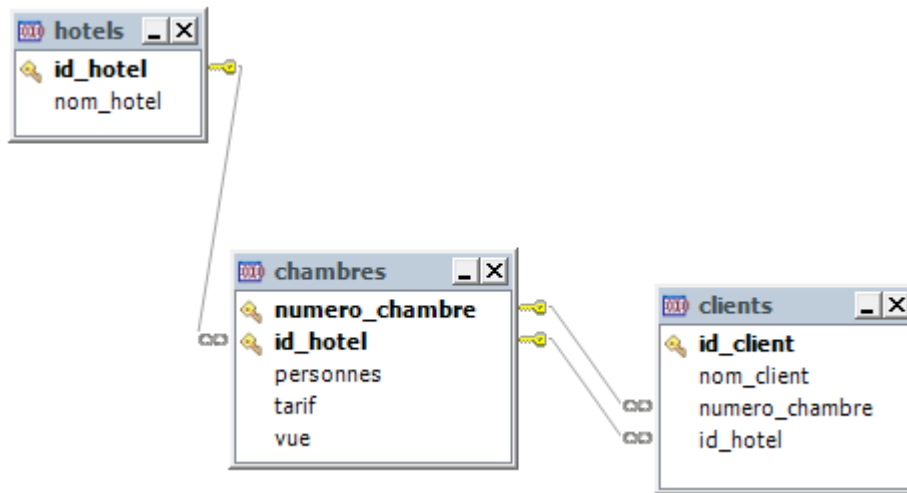
L'opérateur est = parce que l'on sait que la sous-requête ne renvoie qu'une seule valeur.

- **Exemple**

Les villes de France parce qu'on est sûr que France renvoie un seul ID; jointure possible.

```
SELECT * FROM villes  
WHERE id_pays =  
    (SELECT id_pays  
     FROM pays  
     WHERE UPPER(nom_pays) = 'FRANCE');
```

La requête renvoie un seul résultat (une seule ligne mais plusieurs colonnes)



Hotels

id_hotel	nom_hotel
1	Astoria
2	Cambridge
3	Oxford
4	Regina

Chambres (avec une clé double pour l'identification relative)

numero_chambre	id_hotel	personnes	tarif	vue
1	1	2	100	Mer
1	2	2	120	Mer
2	1	2	100	Mer
2	2	1	100	Mer
3	1	1	80	Jardin
3	2	1	90	Ville

Clients (avec une clé étrangère double)

id_client	nom_client	numero_chambre	id_hotel
1	Tintin	1	1
2	Milou	1	1
3	Haddock	2	1
4	Casta	1	2
5	Tournesol	2	2

Création des tables et inserts

```

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET FOREIGN_KEY_CHECKS=0;

DROP DATABASE IF EXISTS hotels;

CREATE DATABASE IF NOT EXISTS hotels
DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci;

USE hotels;

CREATE TABLE hotels.hotels (
  id_hotel INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  nom_hotel VARCHAR(45) NOT NULL,
  PRIMARY KEY (id_hotel)
)
ENGINE = InnoDB;

INSERT INTO hotels (id_hotel, nom_hotel) VALUES(1, 'Astoria');
INSERT INTO hotels (id_hotel, nom_hotel) VALUES(2, 'Cambridge');
INSERT INTO hotels (id_hotel, nom_hotel) VALUES(3, 'Oxford');
INSERT INTO hotels (id_hotel, nom_hotel) VALUES(4, 'Regina');

CREATE TABLE hotels.chambres (
  numero_chambre INTEGER UNSIGNED NOT NULL,
  id_hotel INTEGER UNSIGNED NOT NULL,
  personnes VARCHAR(45) NOT NULL,
  tarif DOUBLE NOT NULL,
  vue VARCHAR(45) NOT NULL,
  PRIMARY KEY (numero_chambre, id_hotel)
)
ENGINE = InnoDB;

ALTER TABLE hotels.chambres
  ADD CONSTRAINT FK_chambres_hotel FOREIGN KEY FK_chambres_hotel (id_hotel)
    REFERENCES hotels (id_hotel)
    ON DELETE CASCADE
    ON UPDATE CASCADE;

INSERT INTO chambres (numero_chambre, id_hotel, personnes, tarif, vue) VALUES(1, 1,
'2', 100, 'Mer');
INSERT INTO chambres (numero_chambre, id_hotel, personnes, tarif, vue) VALUES(1, 2,
'2', 120, 'Mer');
INSERT INTO chambres (numero_chambre, id_hotel, personnes, tarif, vue) VALUES(2, 1,
'2', 100, 'Mer');
INSERT INTO chambres (numero_chambre, id_hotel, personnes, tarif, vue) VALUES(2, 2,
'1', 100, 'Mer');
INSERT INTO chambres (numero_chambre, id_hotel, personnes, tarif, vue) VALUES(3, 1,
'1', 80, 'Jardin');
INSERT INTO chambres (numero_chambre, id_hotel, personnes, tarif, vue) VALUES(3, 2,
'1', 90, 'Ville');

CREATE TABLE hotels.clients (
  id_client INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  nom_client VARCHAR(45) NOT NULL,
  numero_chambre INTEGER UNSIGNED NOT NULL,
  id_hotel INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY (id_client)
)
ENGINE = InnoDB;

ALTER TABLE hotels.clients ADD CONSTRAINT FK_clients_chambre FOREIGN KEY
FK_clients_chambre (numero_chambre, id_hotel)

```

```
REFERENCES chambres (numero_chambre, id_hotel)
ON DELETE RESTRICT
ON UPDATE RESTRICT;

INSERT INTO clients (id_client, nom_client, numero_chambre, id_hotel) VALUES(1,
'Tintin', 1, 1);
INSERT INTO clients (id_client, nom_client, numero_chambre, id_hotel) VALUES(2,
'Milou', 1, 1);
INSERT INTO clients (id_client, nom_client, numero_chambre, id_hotel) VALUES(3,
'Haddock', 2, 1);
INSERT INTO clients (id_client, nom_client, numero_chambre, id_hotel) VALUES(4,
'Casta', 1, 2);
INSERT INTO clients (id_client, nom_client, numero_chambre, id_hotel) VALUES(5,
'Tournesol', 2, 2);

SET FOREIGN_KEY_CHECKS=1;
```

La requête imbriquée : les clients de l'hôtel Cambridge

```
SELECT *
FROM clients c
WHERE (numero_chambre, id_hotel) IN
      (SELECT numero_chambre, id_hotel
       FROM chambres
       WHERE id_hotel =
         (SELECT id_hotel
          FROM hotels
          WHERE nom_hotel = 'Cambridge'
         )
      )
);
```

id_client	nom_client	numero_chambre	id_hotel
4	Casta	1	2
5	Tournesol	2	2

10.3 - FORMAT 2 : LA SOUS-REQUÊTE RENVOIE PLUSIEURS RÉSULTATS

- **Syntaxe**



```
SELECT col1, col2, ...
FROM table1
WHERE colonne IN
  (SELECT colonne
   FROM table2
   WHERE condition
  );
```

- **Exemples**

Les clients de Paris (2 niveaux sur 2 tables; une jointure est possible)

```
SELECT c.nom, c.cp
FROM clients c
WHERE c.cp IN
  (SELECT v.cp
   FROM villes v
   WHERE UPPER(v.nom_ville) LIKE 'PARIS%'
  );
```

```
SELECT c.nom, v.cp
FROM clients c JOIN villes v
ON v.cp = c.cp
WHERE UPPER(v.nom_ville) LIKE 'PARIS%';
```

Les clients qui habitent la même ville que les Buguet (2 niveaux sur la même table).

```
SELECT c.id_client, c.nom, c.prenom
FROM clients c
WHERE c.cp IN
  (SELECT cp
   FROM clients c
   WHERE UPPER(c.nom) = 'BUGUET'
  );
```

Infos complètes sur les commandes d'Evian (3 niveaux; jointure possible).

```
SELECT *
FROM cdes cd
WHERE cd.id_cde IN
    (SELECT l.id_cde
     FROM ligcdes l
     WHERE l.id_produit =
        (SELECT p.id_produit
         FROM produits p
         WHERE UPPER(p.designation) = 'EVIAN'
        )
    );
```

Requête imbriquée et BETWEEN

En attendant mieux ...

Liste des produits dont le prix est plus ou moins proche de la moyenne des prix.

```
SELECT p.designation, p.prix
FROM produits p
WHERE prix BETWEEN (SELECT AVG(prix) - 50 FROM produits)
AND (SELECT AVG(prix) + 50 FROM produits);
```

10.4 - LA REQUÊTE RENVOIE VRAI OU FAUX

La clause EXISTS teste la présence ou l'absence de résultats de la requête imbriquée pour chaque enregistrement du premier SELECT. Elle renvoie True ou False.

- **Syntaxe**

```
SELECT colonnes
FROM table(s)
WHERE [NOT] EXISTS
  (SELECT colonne(s)
   FROM table(s)
   [WHERE (conditions)])
);
```

- **Exemples**

Tester l'existence d'une valeur : existe-t-il des commandes de l'an 2000?

```
SELECT 'Trouvé' AS "Résultat"
FROM dual
WHERE EXISTS
  (SELECT *
   FROM cdes
   WHERE YEAR(date_cde) = 2000
  );
```

Renverra Trouvé s'il existe au moins une commande pour cette année-là.

Ne renverra aucun résultat s'il n'existe aucune commande pour cette année-là.

Tester l'inexistence d'une valeur : existe-t-il des commandes de l'an 1999?

```
SELECT 'Non Trouvé' AS "Résultat"
FROM dual
WHERE NOT EXISTS
  (SELECT *
   FROM cdes
   WHERE YEAR(date_cde) = 1999
  );
```

Renverra 'Non trouvé' s'il n'existe aucune commande pour cette année-là.

Ne renverra aucun résultat s'il existe au moins une commande pour cette année-là.

10.5 - EXISTS ET LA MISE EN PLACE DES CONTRAINTES

- **Objectif**

Contrôler les insertions : contrôler qu'une valeur existe dans une table avant de faire quelque chose.

- **Démarche**

On utilise une clause WHERE NOT EXISTS sur la table dual pour contrôler l'existence d'un enregistrement de référence.

- **Exemple 1 : Interdire les doublons sur une table sans clé primaire !!!**

Cette requête ajoutera ('75011', 'Paris 11', '33') si l'enregistrement n'existe pas et ne l'ajoutera pas s'il existe.

```
INSERT INTO villes(cp, nom_ville, id_pays)
SELECT '75011', 'Paris 11', '33'
FROM dual
WHERE NOT EXISTS
  (SELECT *
   FROM villes
   WHERE cp = '75011'
  );
```

Ce type de requête peut aussi servir pour mettre en place une contrainte d'intégrité référentielle quand les clés étrangères ne peuvent pas être mises en place (Tables MyISAM).

- **Exemple 2 : CIR sur une table MyISAM**

Insérera l'enregistrement Clients puisque '75011' existe dans la table villes.

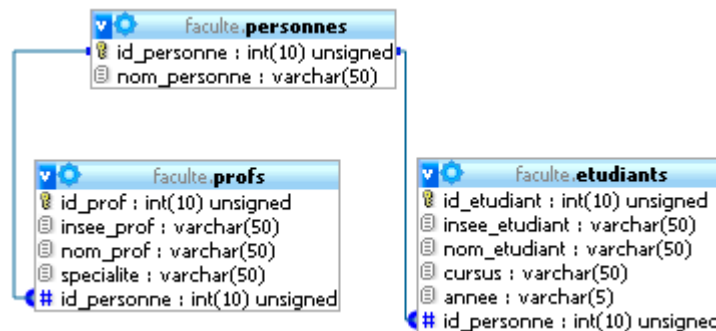
```
INSERT INTO clients(nom, prenom, cp)
SELECT 'Milou', 'Le chien', '75011'
FROM dual
WHERE EXISTS
  (SELECT *
   FROM villes
   WHERE cp = '75011'
  );
```

N'insérera pas l'enregistrement Clients puisque '75016' n'existe pas dans la table villes.

```
INSERT INTO clients(nom, prenom, cp)
SELECT 'Milou', 'Le chien', '75016'
FROM dual
WHERE EXISTS
  (SELECT *
   FROM villes
   WHERE cp = '75016'
  );
```

• Exemple 3 : mise en place de contraintes d'héritage et d'exclusion

Une personne peut-elle être prof et étudiant en même temps ?



La BD nommée 'faculte'

La table personnes(id_personne, nom_personne)

La table profs(id_prof, insee_prof, nom_prof, specialite, id_personne)

La table etudiants(id_etudiant, insee_etudiant, nom_etudiant, annee, cursus, id_personne)

Scripts

```
DROP DATABASE IF EXISTS faculte;
CREATE DATABASE faculte
DEFAULT CHARACTER SET utf8;
USE faculte;
```

```
DROP TABLE IF EXISTS profs;
DROP TABLE IF EXISTS etudiants;
DROP TABLE IF EXISTS personnes;
```

```
CREATE TABLE IF NOT EXISTS personnes (
  id_personne int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom_personne varchar(50) NOT NULL,
  PRIMARY KEY (id_personne)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS profs (
  id_prof int(10) unsigned NOT NULL AUTO_INCREMENT,
  insee_prof varchar(50) NOT NULL,
  nom_prof varchar(50) NOT NULL,
  specialite varchar(50) NOT NULL,
  id_personne int(10) unsigned NOT NULL,
  PRIMARY KEY (id_prof),
  KEY FK_profs_personnes (id_personne)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
ALTER TABLE profs
  ADD CONSTRAINT FK_profs_personnes FOREIGN KEY (id_personne) REFERENCES personnes
  (id_personne) ON DELETE CASCADE ON UPDATE CASCADE;
```

```
CREATE TABLE IF NOT EXISTS etudiants (
  id_etudiant int(10) unsigned NOT NULL AUTO_INCREMENT,
  insee_etudiant varchar(50) NOT NULL,
```

```
nom_etudiant varchar(50) NOT NULL,  
cursus varchar(50) NOT NULL,  
annee varchar(5) NOT NULL,  
id_personne int(10) unsigned NOT NULL,  
PRIMARY KEY (id_etudiant),  
KEY FK_etudiants_personnes (id_personne)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
ALTER TABLE etudiants  
ADD CONSTRAINT FK_etudiants_personnes FOREIGN KEY (id_personne) REFERENCES  
personnes (id_personne) ON DELETE CASCADE ON UPDATE CASCADE;
```

-- Insertion des personnes

```
INSERT INTO personnes (id_personne, nom_personne) VALUES  
(1, 'Tintin'),  
(2, 'Milou'),  
(3, 'Haddock'),  
(4, 'Dupont'),  
(5, 'Dupond'),  
(6, 'Castafiore'),  
(7, 'Tournesol');
```

Insertions avec contraintes (la partition)

```
-- Insere un prof s'il existe dans Personnes : OK
INSERT INTO profs(insee_prof, nom_prof, specialite, id_personne)
SELECT '44444', 'Dupont', 'Maths', 4
FROM dual
WHERE EXISTS
  (SELECT *
   FROM personnes
   WHERE id_personne = 4
  );

-- Insere un etudiant s'il existe dans Personnes : OK
INSERT INTO etudiants(insee_etudiant, nom_etudiant, cursus, annee, id_personne)
SELECT '55555', 'Dupond', 'Droit', '1ere', 5
FROM dual
WHERE EXISTS
  (SELECT *
   FROM personnes
   WHERE id_personne = 5
  );

-- Insere un prof s'il existe dans Personnes et n'existe pas dans etudiants
(Disjonction) : OK
INSERT INTO profs(insee_prof, nom_prof, specialite, id_personne)
SELECT '33333', 'Haddock', 'Statistiques', 3
FROM dual
WHERE EXISTS
  (SELECT *
   FROM personnes
   WHERE id_personne = 3
  )
AND NOT EXISTS
  (SELECT *
   FROM etudiants
   WHERE id_personne = 3
  );
```

Insertions en échec

La personne 8 n'existe pas

```
INSERT INTO profs(insee_prof, nom_prof, specialite, id_personne)
SELECT '88888', 'Obelix', 'Maths', 8
FROM dual
WHERE EXISTS
    (SELECT *
     FROM personnes
     WHERE id_personne = 8
    );
```

La personne 5 est dans la table Etudiants et donc ne pourra pas être insérée dans la table profs

```
INSERT INTO profs(insee_prof, nom_prof, specialite, id_personne)
SELECT '55555', 'Milou', 'Statistiques', 5
FROM dual
WHERE EXISTS
    (SELECT *
     FROM personnes
     WHERE id_personne = 5
    )
AND NOT EXISTS
    (SELECT *
     FROM etudiants
     WHERE id_personne = 5
    );
```


10.6 - LA SOUS REQUÊTE RENVOIE UN AGRÉGAT

Si l'agrégat est mono-valué on emploie l'opérateur **=** ou **>** ou **...**.
Autrement l'opérateur IN.

- **Exemples mono-valués**

Caractéristiques du produit le plus cher.

```
SELECT *
FROM produits
WHERE prix =
    (SELECT MAX(prix)
     FROM produits
    );
```

Caractéristiques des produits dont le prix est supérieur à la moyenne des prix

```
SELECT designation "Désignation"
FROM produits
WHERE prix >
    (SELECT AVG(prix)
     FROM produits
    );
```

- **Exemples multi-valués**

Caractéristiques des produits qui ont été commandés plus d'une fois

```
SELECT designation "Désignation"
FROM produits
WHERE id_produit IN
    (SELECT p.id_produit
     FROM produits p , ligcdes l
     WHERE p.id_produit = l.id_produit
     GROUP BY p.id_produit
     HAVING COUNT(*) > 1
    );
```


10.7 - LES OPÉRATEURS ANY, ALL

• Fonctionnalités

Les opérateurs ANY et ALL sont combinés aux opérateurs =, >, >=, < et <=.

ALL : la condition est vraie SI la comparaison est vraie pour CHACUNE des valeurs ramenées par la sous-requête.

ANY : la condition est vraie SI la comparaison est vraie pour AU MOINS une des valeurs ramenées par la sous-requête.

	ALL	ANY
=	Ne renverra aucun enregistrement	Renvoie toute la table de la Requête 1
>	Ne renverra aucun enregistrement	Tous sauf le plus bas
>=	=MAX sur la même table	Renvoie toute la table de la Requête 1
<	Ne renverra aucun enregistrement	Renvoie toute la table de la Requête 1
<=	=MIN sur la même table	Tous sauf le plus élevé

• Syntaxe

```
SELECT liste_expression FROM liste_de_tables
WHERE
<expression> <opérateur_de_comparaison> {ALL | ANY}
  (SELECT liste_expression
   FROM liste_de_tables
   [WHERE (conditions)]
  );
```

La sous-requête située après les mots clés ALL ou ANY doit avoir le même nombre d'éléments dans sa clause SELECT qu'il y a d'éléments dans expression.

- **Exemples**

ANY

-- Même résultat que SELECT nom FROM clients

```
SELECT nom
FROM clients
WHERE cp = ANY (SELECT cp FROM villes);
```

```
SELECT cp
FROM villes
WHERE cp = ANY (SELECT cp FROM clients);
```

ou encore

```
SELECT cp
FROM villes
WHERE cp IN (SELECT cp FROM clients);
```

-- Le produit le plus cher

-- Le même résultat est obtenu avec la fonction agrégat MAX

SELECT * FROM produits WHERE prix >= ALL (SELECT prix FROM produits);	SELECT * FROM produits WHERE prix = (SELECT MAX (prix) FROM produits);
--	---

-- Tous les produits dont le prix est supérieur au mini (à n'importe quel autre)

-- Le même résultat est obtenu avec la fonction agrégat MIN

SELECT * FROM produits WHERE prix > ANY (SELECT prix FROM produits);	SELECT * FROM produits WHERE prix > (SELECT MIN (prix) FROM produits);
--	--

L'année où il y a eu le plus grand nombre de commandes

AVEC LES OPERATEURS ALL, ANY ou pas dans certains cas ...

Là MAX(COUNT(*)) est impossible; il faut passer par ce type de requête.

On calcule le nombre de commandes par année (requête de niveau 2).

Ensuite on sélectionne l'année où il y a le plus de commandes (requête de niveau 1).

```
# Pour vérifier
SELECT YEAR(date_cde) AS `Année`,
COUNT(*) AS `Nombre de commandes`
FROM cdes
GROUP BY YEAR(date_cde)
ORDER BY 2 DESC;
```

```
# Version standard
SELECT YEAR(date_cde) AS `Année`,
COUNT(*) AS `Nombre de commandes`
FROM cdes
GROUP BY YEAR(date_cde)
HAVING COUNT(*) >= ALL
    (SELECT COUNT(*)
     FROM cdes
     GROUP BY YEAR(date_cde));
```

Explain : 12+12

ou

```
# Version MySQL avec LIMIT
SELECT YEAR(date_cde) AS `Année`,
COUNT(*) AS `Nombre de commandes`
FROM cdes
GROUP BY YEAR(date_cde)
ORDER BY `Nombre de commandes` DESC
LIMIT 0,1;
```

Explain : 12

ou avec

```
# Version avec table dérivée
SELECT YEAR(date_cde) AS `Année`,
COUNT(*) AS `Nombre de commandes`
FROM cdes cd_1
GROUP BY YEAR(date_cde)
HAVING COUNT(*) =
    (SELECT MAX(lemax) FROM
     (SELECT COUNT(*) AS `lemax`
      FROM cdes cd_2
      GROUP BY YEAR(date_cde)
     ) table_derivee
    );
```

Notes : le nommage de la table dérivée, avec n'importe quel nom, est obligatoire ainsi que l'aliasage de la colonne `lemax`.

Explain : 12+3+12

La commande dont le total est le plus élevé.

On calcule le total de chaque commande (requête de niveau 2).

Ensuite on sélectionne la commande dont le total est le plus élevé (requête de niveau 1).

```
# La commande la plus élevée
SELECT l.id_cde `Code commande`,
SUM(p.prix * l.qte) `Total de la commande`
FROM ligcdes l INNER JOIN produits p
ON l.id_produit = p.id_produit
GROUP BY l.id_cde
HAVING SUM(p.prix * l.qte) >= ALL
  (SELECT SUM(p.prix * l.qte)
   FROM ligcdes l INNER JOIN produits p
   ON l.id_produit = p.id_produit
   GROUP BY l.id_cde
  );
```

Le produit le + commandé

```
SELECT p.id_produit, p.designation,
COUNT(*) AS compte
FROM ligcdes l INNER JOIN produits p
ON l.id_produit = p.id_produit
GROUP BY p.id_produit, p.designation
HAVING COUNT(*) >= ALL
  (SELECT COUNT(*)
   FROM ligcdes
   GROUP BY id_produit
  );
```

Le produit le + commandé avec les quantités commandées et la date de commande

```
SELECT p.id_produit, p.designation, l.qte, cd.date_cde
FROM ligcdes l INNER JOIN produits p
ON l.id_produit = p.id_produit
INNER JOIN cdes cd
ON l.id_cde = cd.id_cde
AND p.id_produit =
  (SELECT id_produit
   FROM ligcdes l
   GROUP BY l.id_produit
   HAVING COUNT(*) >= ALL
    (SELECT COUNT(*)
     FROM ligcdes l
     GROUP BY l.id_produit
    )
  )
ORDER BY 3 DESC
;
```

Les ventes par client par année triées par Année/Client/CA DESC

```
SELECT YEAR(cd.date_cde) AS `Année`,
       c.id_client,
       SUM(l.qte * p.prix) AS `CA`
FROM clients c INNER JOIN cdes cd INNER JOIN ligcdes l INNER JOIN produits p
ON c.id_client = cd.id_client
AND cd.id_cde = l.id_cde
AND l.id_produit = p.id_produit
GROUP BY c.id_client, YEAR(cd.date_cde)
ORDER BY `Année`, `CA` DESC, c.id_client;
```

10.8 - MISES À JOUR EN FONCTION D'UNE SOUS-REQUÊTE

10.8.1 - Insertion

Insérer un ou plusieurs enregistrements à partir d'un SELECT (donc d'une autre table).

```
INSERT INTO table1 [(col1, col2, ...)]  
SELECT col1, col2, ... FROM table2 ...;
```

- **Exemple**

```
-- Insère dans la table villes_bis tous des enregistrements de la table Villes  
INSERT INTO villes_bis  
SELECT *  
FROM villes;
```

Utile pour l'archivage !

10.8.2 - Suppression

DELETE FROM nom_de_table WHERE colonne Opérateur (SELECT ...);

- **Exemples**

```
-- Suppression les lillois
DELETE FROM clients
WHERE cp IN
  (SELECT cp
   FROM villes
   WHERE nom_ville LIKE 'Lille%'
  );
```

```
-- Suppression des villes d'Angleterre
DELETE
FROM villes
WHERE id_pays =
  (SELECT id_pays
   FROM pays
   WHERE UPPER(nom_pays) = 'ANGLETERRE'
  );
```


10.8.3 - Modification

```
UPDATE nom_de_table SET col1 = valeur1 [, col2 = valeur2]
WHERE colonne Opérateur (SELECT ... );
```

- **Exemple**

Mettre en majuscules les noms des clients qui habitent Lille.

```
UPDATE clients SET nom = UPPER(nom)
WHERE cp IN
  (SELECT cp
   FROM villes
   WHERE UPPER(nom_ville) LIKE UPPER('lille%'))
);
```

WARNING : il est impossible de faire un UPDATE sur une table avec dans la sous-requête un SELECT sur la table à modifier.

Par exemple :

```
UPDATE clients SET nom = UPPER(nom)
WHERE cp IN
  (SELECT cp
   FROM clients
   WHERE UPPER(nom) = 'BUGUET')
);
```

ErrNo 1093 : You can't specify target table 'clients' for update in FROM clause

Une solution est de faire la modification dans une procédure stockée avec la création d'une table temporaire ou d'une TEMPORARY TABLE ou d'une MEMORY TABLE.

CHAPITRE 11 - LES REQUETES CORRELEES

11.1 - PRÉSENTATION

Une requête corrélée est une sous-requête qui utilise la requête principale.

Pas à pas ... sur la BD [sdp]. Cf le script sdp_create_insert.sql.

Tous les articles

id_article	titre_article
1	Tintin au Tibet
2	Tintin au Congo
3	On a marché sur la Lune
4	Le temple du Soleil
5	Le Sceptre d'Ottokar
6	Le secret de la licorne

```
SELECT id_article, titre_article FROM sdp.articles a;
```

Tous les commentaires sur les articles

1	Sur Tintin au Tibet	Pas mal	2012-10-21 09:59:04	1
2	Encore sur Tintin au Tibet	Etrange	2012-10-22 09:59:04	1
7	Tintin au Tibet l'actualité	Excellente présentation	2012-12-02 09:59:04	1
8	Dalai Lama	On ne crache pas dan	2012-12-03 09:59:04	1
9	Tibet ... Lama	A rire	2012-12-04 09:59:04	1
3	Sur la Lune	Très bien	2012-11-16 09:59:04	3
4	Encore sur la Lune!!!	Cet article commence	2012-11-29 09:59:04	3
5	Dans la Lune!!!	Cet article commence	2012-11-30 09:59:04	3
6	Sur Tintin au Temple	Bien	2012-12-01 09:59:04	4

```
SELECT id_commentaire, titre_commentaire, texte_commentaire, date_commentaire,
id_article
FROM sdp.commentaires c
ORDER BY id_article;
```

Tous les commentaires concernant l'article 3

```
SELECT *
FROM sdp.commentaires c
WHERE id_article = 3;
```

Exemples de requêtes corrélées :

Extraire le dernier commentaire concernant un article, l'article 1 en l'occurrence.

id_commentaire	texte_commentaire	date_commentaire	id_article
9	A rire	2012-12-04 09:59:04	1

```
SELECT c.id_commentaire, c.texte_commentaire, c.date_commentaire, c.id_article
FROM sdp.commentaires c
WHERE c.id_article = 1
AND c.date_commentaire =
    (SELECT MAX(date_commentaire)
     FROM sdp.commentaires cInterne
     WHERE cInterne.id_article = c.id_article
    )
;
```

(EXPLAIN 4 rows).

MAIS avec MySQL il est possible d'obtenir le même résultat avec :

```
SELECT c.id_commentaire, c.id_article, c.texte_commentaire, c.date_commentaire
FROM sdp.commentaires c
WHERE c.id_article = 1
ORDER BY c.date_commentaire DESC
LIMIT 0,1;
```

(EXPLAIN 4 rows).

Tous les derniers commentaires ... donc sans la condition a.id_article = ...

id_commentaire	titre_commentaire	date_commentaire	id_article
5	Dans la Lune!!!	2012-11-30 09:59:04	3
6	Sur Tintin au Temple	2012-12-01 09:59:04	4
9	Tibet ... Lama	2012-12-04 09:59:04	1

```
SELECT c.id_commentaire, c.titre_commentaire, c.date_commentaire, c.id_article
FROM sdp.commentaires c
WHERE c.date_commentaire =
    (SELECT MAX(date_commentaire)
     FROM sdp.commentaires cInterne
     WHERE cInterne.id_article = c.id_article
    )
;
```

11.2 - EPUISEMENT DE LA CORRÉLATION !!!

Exemples avec les deux tables vendeurs et vendeurs_villes : requêtes corrélées (la sous-requête fait appel à la requête principale dans les 3 derniers cas).

Vendeurs

id_vendeur	nom	chef	cp
1	Lucky	0	75011
2	Dalton	1	75012
3	Mickey	1	75011
4	Donald	2	75011

Vendeurs_villes

id_vendeur	cp	Date_debut	date_fin
1	75011	2006-01-02	2006-12-31
1	75011	2007-01-02	2007-12-31
2	75012	2007-01-02	2007-12-31
2	75011	2007-01-02	2007-12-31
3	75012	2006-01-02	2006-12-31
3	75012	2007-01-02	2007-12-31

Donc :

Lucky, Dalton et Mickey ont travaillé (1,2,3),
 Lucky a toujours travaillé là où il habite (75011),
 Dalton a parfois travaillé là où il habite (75012),
 Mickey n'a jamais travaillé là où il habite (75011),
 Donald n'a jamais travaillé !!!

Si les tables n'existent pas dans la BD voici les scripts :

```
SET FOREIGN_KEY_CHECKS=0;

DROP TABLE IF EXISTS vendeurs;

CREATE TABLE IF NOT EXISTS vendeurs (
  id_vendeur int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom_vendeur varchar(45) NOT NULL,
  chef int(10) unsigned NOT NULL DEFAULT '0',
  cp char(5) NOT NULL,
  PRIMARY KEY (id_vendeur),
  KEY FK_vendeurs_cp (cp),
  KEY FK_vendeurs_id_vendeur (chef)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO vendeurs (id_vendeur, nom_vendeur, chef, cp) VALUES
(1, 'Lucky', 0, '75011'),
(2, 'Dalton', 1, '75012'),
(3, 'Mickey', 1, '75011'),
(4, 'Donald', 2, '75011');

DROP TABLE IF EXISTS vendeurs_villes;

CREATE TABLE IF NOT EXISTS vendeurs_villes (
  id_vendeur int(10) unsigned NOT NULL AUTO_INCREMENT,
  cp varchar(5) NOT NULL,
  date_debut date NOT NULL DEFAULT '0000-00-00',
  date_fin date NOT NULL DEFAULT '0000-00-00',
  PRIMARY KEY (id_vendeur,cp,Date_debut),
  KEY cp (cp)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO vendeurs_villes (id_vendeur, cp, date_debut, date_fin) VALUES
(1, '75011', '2006-01-02', '2006-12-31'),
(1, '75011', '2007-01-02', '2007-12-31'),
(2, '75011', '2007-01-02', '2007-12-31'),
(2, '75012', '2007-01-02', '2007-12-31'),
(3, '75012', '2006-01-02', '2006-12-31'),
(3, '75012', '2007-01-02', '2007-12-31');

SET FOREIGN_KEY_CHECKS=1;
```

Les vendeurs qui ont travaillé au moins une fois (ils sont dans Vendeurs_villes)

id_vendeur	nom	chef	cp
1	Lucky	0	75011
2	Dalton	1	75012
3	Mickey	1	75011

```
SELECT *
FROM vendeurs v
WHERE id_vendeur IN
  (SELECT DISTINCT id_vendeur
   FROM vendeurs_villes
  );
```

Les vendeurs qui n'ont jamais travaillé (ils ne sont pas dans Vendeurs_villes)

id_vendeur	nom	chef	cp
4	Donald	2	75011

```
SELECT *
FROM vendeurs v
WHERE id_vendeur NOT IN
  (SELECT DISTINCT id_vendeur
   FROM vendeurs_villes
  );
```


Les vendeurs qui ont travaillé (au moins une fois) dans la ville où ils habitent.

id_vendeur	nom	chef	cp
1	Lucky	0	75011
2	Dalton	1	75012

```
SELECT * FROM vendeurs v
WHERE cp IN
  (SELECT cp
   FROM vendeurs_villes
   WHERE id_vendeur = v.id_vendeur
  );
```

ou

```
SELECT * FROM vendeurs v
WHERE cp = ANY
  (SELECT cp
   FROM vendeurs_villes
   WHERE id_vendeur = v.id_vendeur
  );
```

Les vendeurs qui ont toujours travaillé dans la ville où ils habitent.

id_vendeur	nom	cp
1	Lucky	75011

```
SELECT DISTINCT v.id_vendeur, v.nom_vendeur, v.cp
FROM vendeurs v INNER JOIN vendeurs_villes vv
ON v.id_vendeur = vv.id_vendeur
WHERE v.cp = ALL
  (SELECT cp
   FROM vendeurs_villes
   WHERE id_vendeur = v.id_vendeur
  );
```

Les vendeurs qui n'ont jamais travaillé dans la ville où ils habitent.

id_vendeur	nom	cp
3	Mickey	75011

```
SELECT DISTINCT v.id_vendeur, v.nom_vendeur, v.cp
FROM vendeurs v INNER JOIN vendeurs_villes vv
ON v.id_vendeur=vv.id_vendeur
WHERE v.cp <> ALL
  (SELECT cp
   FROM vendeurs_villes
   WHERE id_vendeur = v.id_vendeur
  );
```

11.3 - EXISTS ET NOT EXISTS CORRÉLÉS EN SUBSTITUTION D'UNE JOINTURE EXTERNE

EXISTS + une corrélation.

EXISTS

Renverra la liste des villes où il y a au moins un client.
La jointure est moins coûteuse.

```
SELECT nom_ville
FROM villes v
WHERE EXISTS
  (SELECT *
   FROM clients
   WHERE cp = v.cp
  );
```

EXPLAIN : 14+1 (avec communes 24973+1 ; avec communes + index sur codepos : 47258+1)

C'est la même chose que ceci (jointure, et la jointure est nettement moins coûteuse) :

```
SELECT DISTINCT nom_ville
FROM villes v INNER JOIN clients c
ON v.cp = c.cp;
```

EXPLAIN : 11+1 (avec communes 24973+1 ; avec communes + index sur codepos : 11+6)

Exemples avec la table communes

```
EXPLAIN SELECT commune
FROM communes co
WHERE EXISTS
  (SELECT *
   FROM clients
   WHERE cp = co.codepos
  );
```

```
EXPLAIN SELECT DISTINCT commune
FROM communes co INNER JOIN clients c
ON cp = co.codepos;
```

NOT EXISTS

Renverra la liste des villes où il n'y a pas de clients.
Corrélée et jointure ont le même coût.
Et c'est moins coûteux sans index qu'avec l'index.

```
SELECT nom_ville
FROM villes v
WHERE NOT EXISTS
  (SELECT *
   FROM clients
   WHERE cp = v.cp
  );
```

EXPLAIN : 14+1 (avec communes 24973+1 ; avec index sur codepos : 47258+1)

C'est la même chose que ceci (jointure externe) :

```
SELECT DISTINCT nom_ville
FROM villes v
LEFT OUTER JOIN clients c
ON v.cp = c.cp
WHERE c.cp IS NULL;
```

EXPLAIN : 14+1 (avec communes 24973+1 ; avec index sur codepos : 47258+1)

Exemples avec la table communes

```
EXPLAIN SELECT commune
FROM communes co
WHERE NOT EXISTS
  (SELECT *
   FROM clients
   WHERE cp = co.codepos
  );
```

```
EXPLAIN SELECT DISTINCT commune
FROM communes co LEFT OUTER JOIN clients c
ON co.codepos = c.cp
WHERE c.cp IS NULL;
```

11.4 - UPDATE ET CORRÉLATION

Rappel une corrélation sur la même table est impossible avec un UPDATE.

Admettons que l'on dénormalise une table. Par exemple la table clients où l'on ajoute la colonne nom_ville.

Pour mettre à jour les valeurs de la nouvelle colonne il faut exécuter cet ordre-ci :

```
UPDATE clients c
SET c.nom_ville =
(SELECT v.nom_ville
FROM villes v
WHERE v.cp = c.cp);
```

ou celui-ci si la table possède comme clé étrangère id_ville et non pas cp :

```
UPDATE clients c
SET c.nom_ville =
(SELECT v.nom_ville
FROM villes v
WHERE v.id_ville = c.id_ville);
```

ou la procédure stockée [denormalisation_clients] qui contient l'ordre UPDATE.

```
CALL denormalisation_clients();
```

Pour vérifier :

```
SELECT c.nom_client, c.cp, c.nom_ville FROM clients c;
```

ou

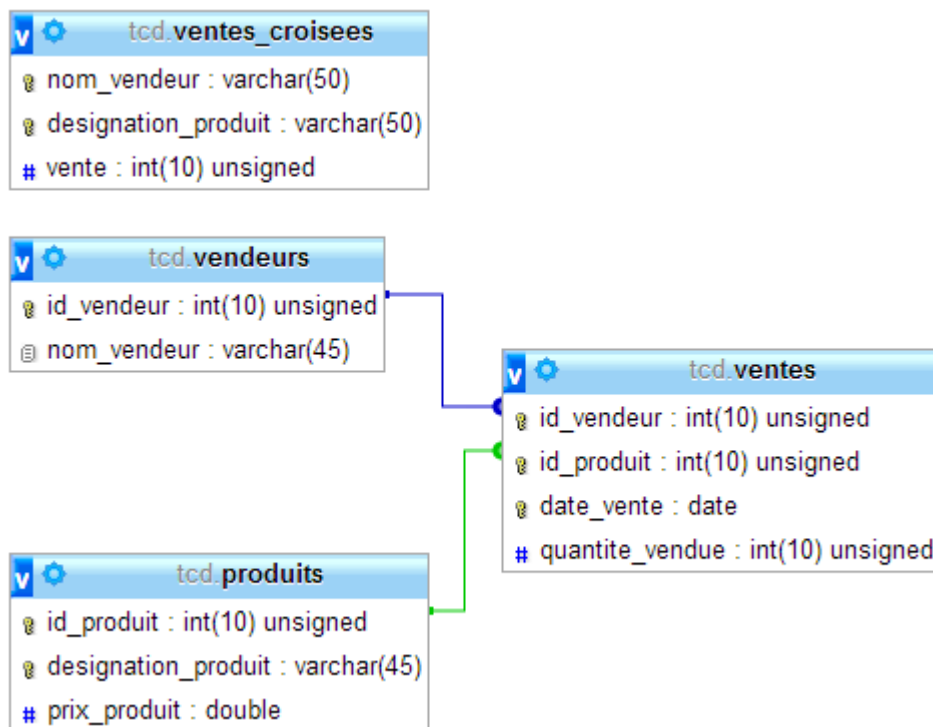
```
SELECT c.nom_client, c.id_ville, c.nom_ville FROM clients c;
```

11.5 - DELETE ET CORRÉLATION

CHAPITRE 12 - LES TABLEAUX CROISES DYNAMIQUES

12.1 - PRÉPARATION

Le schéma de la BD.



La BD

```
DROP DATABASE IF EXISTS tcd;

CREATE DATABASE IF NOT EXISTS tcd
DEFAULT CHARACTER SET utf8
COLLATE utf8_general_ci;

USE tcd;
```

La table 'ventes_croisees'

```
CREATE TABLE tcd.ventes_croisees (
  nom_vendeur varchar(50) NOT NULL default '',
  designation_produit varchar(50) NOT NULL default '',
  vente int(10) unsigned NOT NULL default '0',
  PRIMARY KEY (nom_vendeur, designation_produit)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO tcd.ventes_croisees (nom_vendeur, designation_produit, vente)
VALUES
('Casta', 'Evian', 20),
('Casta', 'Graves', 5),
('Haddock', 'Badoit', 1),
('Haddock', 'Evian', 1),
('Haddock', 'Graves', 10),
('Tintin', 'Badoit', 5),
('Tintin', 'Evian', 10),
('Tintin', 'Graves', 10);
```

Les tables vendeurs, produits, ventes

```
DROP TABLE IF EXISTS tcd.vendeurs;

CREATE TABLE tcd.vendeurs (
  id_vendeur int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom_vendeur varchar(45) NOT NULL,
  PRIMARY KEY (id_vendeur)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS tcd.produits;

CREATE TABLE tcd.produits (
  id_produit int(10) unsigned NOT NULL AUTO_INCREMENT,
  designation_produit varchar(50) NOT NULL,
  prix_produit float(7,2) NOT NULL,
  PRIMARY KEY (id_produit),
  UNIQUE KEY index_designation (designation_produit)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS tcd.ventes;

CREATE TABLE tcd.ventes (
  id_vendeur int(10) unsigned NOT NULL,
  id_produit int(10) unsigned NOT NULL,
  date_vente date NOT NULL,
```



```
    quantite_vendue int(10) unsigned NOT NULL,  
    PRIMARY KEY (id_vendeur,id_produit,date_vente)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;  
  
ALTER TABLE tcd.ventes ADD CONSTRAINT FK_ventes_vendeur FOREIGN KEY  
FK_ventes_vendeur (id_vendeur)  
    REFERENCES vendeurs (id_vendeur)  
    ON DELETE RESTRICT  
    ON UPDATE RESTRICT;  
  
ALTER TABLE tcd.ventes ADD CONSTRAINT FK_ventes_produit FOREIGN KEY  
FK_ventes_produit (id_produit)  
    REFERENCES produits (id_produit)  
    ON DELETE RESTRICT  
    ON UPDATE RESTRICT;  
  
INSERT INTO tcd.produits (id_produit, designation_produit, prix_produit)  
VALUES  
(1, 'Evian', 1.50),  
(2, 'Graves', 5.50),  
(3, 'Badoit', 1.20);  
  
INSERT INTO tcd.vendeurs (id_vendeur, nom_vendeur)  
VALUES  
(1, 'Casta'),  
(2, 'Haddock'),  
(3, 'Tintin'),  
(4, 'Tournesol');  
  
INSERT INTO tcd.ventes (id_vendeur, id_produit, date_vente, quantite_vendue)  
VALUES  
(1, 1, '2007-04-16', 1),  
(1, 2, '2007-04-16', 1),  
(1, 3, '2007-04-16', 1),  
(2, 1, '2007-04-16', 1),  
(2, 2, '2007-04-16', 1),  
(3, 1, '2007-04-16', 1);
```

Remarques : chaque vendeur a fait au moins une vente sauf Tournesol qui n'en a fait aucune.

12.2 - TCD SUR UNE TABLE

- **Définition**

Un TCD (Tableau croisé dynamique) permet de passer d'une représentation 1D à une représentation 2D.

Exemple :

Les ventes des vendeurs par produit. En 1D c'est peu lisible ...

nom_vendeur	designation_produit	vente
Casta	Evian	20
Casta	Graves	5
Haddock	Badoit	1
Haddock	Evian	1
Haddock	Graves	10
Tintin	Badoit	5
Tintin	Evian	10
Tintin	Graves	10

Vers 2D (Vendeur en ligne et produit en colonne), c'est plus lisible ...

nom_vendeur	Evian	Graves	Badoit
Casta	20	5	0
Haddock	1	10	1
Tintin	10	10	5

Syntaxe du TCD

```
SELECT colonne à afficher en ligne,  
    SUM(IF(colonne à afficher en colonne= 'valeur', valeur à afficher à l'intersection si  
elle existe, 0)) AS alias,  
    SUM(IF(colonne à afficher en colonne = 'valeur', valeur à afficher à l'intersection si  
elle existe, 0)) AS alias,  
...  
GROUP BY colonne à afficher en ligne;
```

Note : cf le IF() au paragraphe 5.12.1.

-- La vue qui crée le TCD statique à partir de la table

```
CREATE OR REPLACE VIEW tcd.v_ventes_tcd_statique_table AS  
SELECT nom_vendeur,  
SUM(IF(designation_produit = 'Evian' , vente, 0)) AS 'Evian',  
SUM(IF(designation_produit = 'Graves' , vente, 0)) AS 'Graves',  
SUM(IF(designation_produit = 'Badoit' , vente, 0)) AS 'Badoit'  
FROM tcd.ventes_croisees  
GROUP BY nom_vendeur;
```

```
SELECT * FROM tcd.v_ventes_tcd_statique_table v;
```

12.3 - TCD SUR UNE JOINTURE STATIQUE

Avec les tables Vendeurs, Ventes et Produits

nom_vendeur	Evian	Graves	Badoit
Casta	2	1	1
Haddock	1	1	0
Tintin	1	0	0

Même syntaxe mais sur une equi-jointure sur 3 tables.

```
CREATE OR REPLACE VIEW tcd.v_ventes_tcd_statique_jointure_equi AS
SELECT nom_vendeur,
SUM(IF(designation_produit = 'Evian', quantite_vendue, 0)) AS `Evian`,
SUM(IF(designation_produit = 'Graves', quantite_vendue, 0)) AS `Graves`,
SUM(IF(designation_produit = 'Badoit', quantite_vendue, 0)) AS `Badoit`
FROM vendeurs JOIN ventes JOIN produits
ON vendeurs.id_vendeur = ventes.id_vendeur
AND ventes.id_produit = produits.id_produit
GROUP BY nom_vendeur;
```

```
SELECT * FROM tcd.v_ventes_tcd_statique_jointure_equi v;
```

Pour faire apparaître tous les vendeurs ... même ceux qui n'ont rien vendu ...

nom_vendeur	Evian	Graves	Badoit
Casta	2	1	1
Haddock	1	1	0
Tintin	1	0	0
Tournesol	0	0	0

utilisez une jointure externe

```
CREATE OR REPLACE VIEW tcd.v_ventes_tcd_statique_jointure_externe AS
SELECT nom_vendeur,
SUM(IF(designation_produit = 'Evian', quantite_vendue, 0)) AS 'Evian',
SUM(IF(designation_produit = 'Graves', quantite_vendue, 0)) AS 'Graves',
SUM(IF(designation_produit = 'Badoit', quantite_vendue, 0)) AS 'Badoit'
FROM (vendeurs LEFT JOIN ventes ON vendeurs.id_vendeur = ventes.id_vendeur)
LEFT JOIN produits ON ventes.id_produit = produits.id_produit
GROUP BY nom_vendeur;
```

```
SELECT * FROM tcd.v_ventes_tcd_statique_jointure_externe v;
```

Rajoutez une vente sur un produit existant:

```
INSERT INTO tcd.ventes(id_vendeur, id_produit, date_vente, quantite_vendue)
VALUES(1, 1, '2007-04-18', 1);
```

Ré-exécutez la vue ... la nouvelle vente est intégrée ...

Si vous ajoutez un vendeur il est pris en compte mais si vous ajoutez un produit les résultats ne sont pas mis à jour (cf paragraphe suivant).

```
INSERT INTO tcd.vendeurs(nom_vendeur) VALUES('Rasto');
```

12.4 - TCD SUR UNE JOINTURE DYNAMIQUE

La limite de ce qui vient d'être fait, c'est que les TCD sont statiques dans la mesure où la liste des produits est statiquement inscrite dans la requête. Si l'on ajoute un produit il faut modifier la requête.

Pour tester ajoutez un produit et ajoutez une vente concernant ce produit :

```
INSERT INTO tcd.produits(designation_produit, prix_produit)
VALUES ('Médoc', 10);
```

```
INSERT INTO tcd.ventes(id_vendeur, id_produit, date_vente, quantite_vendue)
VALUES (1, 4, '2007-04-21', 1);
```

Les résultats de la view ne changent pas.

Pour réaliser un TCD vraiment dynamique il faut travailler avec un langage de programmation procédurale.

Soit avec une procédure stockée,
Soit avec un langage serveur dynamique de type PHP, ASP, JSP, ...

En créant d'abord la liste des désignations de la table produits.
Puis le SELECT dynamique.

```
CREATE VIEW tcd.v_produits
AS SELECT id_produit, designation_produit
FROM produits;
```

```
CREATE VIEW tcd.v_vendeurs
AS SELECT id_vendeur, nom_vendeur
FROM vendeurs;
```

```
SELECT * FROM tcd.v_produits;
SELECT * FROM tcd.v_vendeurs;
```

cf les procédures stockées ou des exemples avec du code Java ou PHP.

CHAPITRE 13 - LES TRANSACTIONS

13.1 - PRINCIPES

Une transaction est l'espace-temps qui s'écoule entre deux états stables et cohérents de la base de données.

Les transactions doivent garantir les propriétés **ACID** (Atomicité, Cohérence, Isolation, Durabilité).

Atomicité : garantit le fait que toutes les actions élémentaires sont effectuées ou aucune.

Cohérence : garantit le fait que l'on passe d'un état cohérent à un autre état cohérent.

Isolation : garantit le fait que d'autres actions ne peuvent accéder aux données pendant la transaction.

Durabilité : garantit le fait qu'une fois la transaction validée elle ne peut être défaite.

Lors des MAJ la BD est en état instable et incohérent. Ce n'est que lorsque toutes les MAJ atomiques d'une transaction (Ajout d'une commande avec ses lignes de commandes, opération de transfert de débit-crédit d'un compte vers un autre) que la BD retrouve un état stable et cohérent.

Seules les tables InnoDB, BDB, Berkeley supportent les transactions.

Par défaut MySQL est lancé avec l'option `AutoCommit=true` ou `AutoCommit=1`. C'est-à-dire qu'à chaque instruction de MAJ la MAJ dans la BD est définitive. Ce n'est pas un bon paramétrage.

Ceci est modifiable dans le fichier `my.ini` :

```
| init_connect='SET autocommit=0'
```

mais ceci n'aura d'effet que pour les users qui n'ont pas les droits SUPER donc root est toujours par défaut en `autocommit=true`.

13.2 - L'ÉTAT DE LA GESTION DES TRANSACTIONS

Pour connaître l'état de l'autocommit tapez :

```
| SELECT @@AUTOCOMMIT;
```

Par défaut renvoie 1.

Pour modifier l'état basculez avec SET AUTOCOMMIT = 0;

La gestion des transactions peut aussi être réalisée grâce à la commande **START TRANSACTION** depuis MySQL 4.0.11. Dans ce cas-ci il n'est pas nécessaire de modifier la valeur du paramètre de session autocommit.

13.3 - VALIDATION



Valide toutes les mises à jour depuis le dernier commit ou l'ouverture de la session.

```
COMMIT;
```

13.4 - ANNULATION



Annule toutes les mises à jour depuis le dernier commit ou le dernier rollback ou l'ouverture de la session.

```
ROLLBACK;
```

Exemple : virement d'un compte d'épargne à un compte courant**Création des tables et insertions pour l'exemple.**

```

DROP TABLE IF EXISTS cours.compte_courant;
CREATE TABLE cours.compte_courant (
  id_compte int(10) unsigned NOT NULL AUTO_INCREMENT,
  titulaire varchar(45) NOT NULL,
  solde int(10) unsigned NOT NULL,
  PRIMARY KEY (id_compte)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS cours.compte_epargne;
CREATE TABLE cours.compte_epargne (
  id_compte int(10) unsigned NOT NULL AUTO_INCREMENT,
  titulaire varchar(45) NOT NULL,
  solde int(10) unsigned NOT NULL,
  PRIMARY KEY (id_compte)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO cours.compte_courant (id_compte, titulaire, solde) VALUES
(1, 'Tintin', 1000);

INSERT INTO cours.compte_epargne (id_compte, titulaire, solde) VALUES
(1, 'Tintin', 10000);

```

Transaction.

Opération	Description
SELECT * FROM compte_courant; SELECT * FROM compte_epargne;	Vérification du solde de chaque compte
SET AUTOCOMMIT = 0;	Passage à autocommit false
SELECT @@AUTOCOMMIT;	Vérification de l'état de la gestion des transactions
UPDATE compte_epargne SET solde = solde - 1000 WHERE id_compte = 1;	Débit
UPDATE compte_courant SET solde = solde + 1000 WHERE id_compte = 1;	Crédit
COMMIT;	Validation
SET AUTOCOMMIT = 1;	Passage à autocommit true
SELECT @@AUTOCOMMIT;	Vérification de l'état de la gestion des transactions
SELECT * FROM compte_courant; SELECT * FROM compte_epargne;	Vérification du solde de chaque compte

Remarque liminaire : avec START TRANSACTION; les instructions ne sont pas validées tant qu'il n'y pas eu de COMMIT; donc SET AUTOCOMMIT = 0; est inutile.

```
SELECT * FROM cours.compte_courant;

SELECT * FROM cours.compte_epargne;

-- SET AUTOCOMMIT = 0;

-- SELECT @@AUTOCOMMIT;

START TRANSACTION;

UPDATE cours.compte_epargne
SET solde = solde - 1000
WHERE id_compte = 1;

UPDATE cours.compte_courant
SET solde = solde + 1000
WHERE id_compte = 1;

COMMIT;

-- SET AUTOCOMMIT = 1;

-- SELECT @@AUTOCOMMIT;

SELECT * FROM cours.compte_courant;

SELECT * FROM cours.compte_epargne;
```

Testez l'isolation de la transaction avec 2 clients MySQL (2 MySQL Query Browser par exemple).

13.5 - LES SAVEPOINTS

Note : depuis la version 4.0.14 MySQL supporte (avec les tables InnoDB) les commandes SAVEPOINT point_de_sauvegarde; et ROLLBACK TO point_de_sauvegarde;

- **Objectif**

Diviser une transaction en plusieurs sous-parties.

- **Syntaxes**

Création d'un point de sauvegarde

```
SAVEPOINT point_de_sauvegarde;
```

Annule les MAJ jusqu'au point de sauvegarde

```
ROLLBACK TO SAVEPOINT point_de_sauvegarde;
```

Suppression d'un point de sauvegarde

```
RELEASE SAVEPOINT point_de_sauvegarde;
```

- **Exemple**

Au final vous n'aurez que 75031.

```
SELECT * FROM villes;
SET AUTOCOMMIT=0;

START TRANSACTION;
SAVEPOINT sp1;
INSERT INTO villes(cp, nom_ville) VALUES('75031','Paris 31');

SAVEPOINT sp2;
INSERT INTO villes(cp, nom_ville) VALUES('75032','Paris 32');

ROLLBACK TO SAVEPOINT sp2;

COMMIT;
SELECT * FROM villes;
```

13.6 - LE VERROUILLAGE DE TABLE

Les commandes LOCK et UNLOCK permettent de verrouiller et de déverrouiller une ou plusieurs tables en lecture ou en lecture/écriture.

- **Syntaxes**

```
LOCK TABLES nom_de_table verrouillage [, nom_de_table verrouillage];
```

Verrouillage prend les valeurs READ ou WRITE.

Un verrouillage de type READ autorise les lectures mais pas les écritures de la part des autres utilisateurs.

Un verrouillage de type WRITE n'autorise ni les lectures ni les écritures.

```
UNLOCK TABLES;
```

- **Exemple**

Ouvrez deux sessions clients (mysql et MySQL Query Browser par exemple).

Utilisateur	Autre utilisateur
SELECT * FROM villes; START TRANSACTION; LOCK TABLES villes READ ; UPDATE villes SET nom_ville = 'Marsiglia' WHERE cp = '13000'; UNLOCK TABLES ; COMMIT;	SELECT * FROM villes; -- OK UPDATE villes SET nom_ville = 'Marsilia' WHERE cp = '13000'; -- KO

Utilisateur	Autre utilisateur
SELECT * FROM villes; START TRANSACTION; LOCK TABLES villes WRITE ; UPDATE villes SET nom_ville = 'Marsiglia' WHERE cp = '13000'; UNLOCK TABLES ; COMMIT;	SELECT * FROM villes; -- KO UPDATE villes SET nom_ville = 'Marsilia' WHERE cp = '13000'; -- KO

13.7 - LE VERROUILLAGE DE LIGNE

La clause FOR UPDATE appliquée à un SELECT permet de verrouiller une ou plusieurs lignes en écriture.

- **Syntaxe**

```
SELECT * FROM nomDeTable WHERE condition FOR UPDATE;
```

- **Exemple**

Ouvrez deux sessions clients (mysql et MySQL Query Browser par exemple).

Utilisateur	Autre utilisateur
START TRANSACTION; SELECT * FROM pays WHERE id_pays = '033' FOR UPDATE ; UPDATE pays SET nom_pays = 'FR' WHERE id_pays = '033'; COMMIT;	START TRANSACTION; UPDATE pays SET nom_pays = 'fr' WHERE id_pays = '033'; -- Attente bloquante (*) COMMIT;

(*) UPDATE sur un autre pays c'est OK.

CHAPITRE 14 - LES TYPES SQL3

14.1 - LES BLOBS

- **Objectif**

Travailler avec les Binary Large Objects de SQL3.

IN BD		OUT BD	
+	-	+	-
Centralisation Sauvegarde aisée Sécurité Maintenance aisée	Lent Codage applicatif plus complexe	Rapide Codage applicatif facile	Sauvegarde en 2 temps

MySQL dispose de 4 sous-types :

TinyBlob	256 octets (2^8)
Blob	64 ko ou 65 536 octets (2^{16})
MediumBlob	16 Mo ou 16 777 215 d'octets (2^{24})
LongBlob	4 Go Ou 4 294 967 295 d'octets (2^{32})

Avantages et inconvénients du stockage des BLOBS dans la BD :

Avantages	Inconvénients
En cas de réplication	Taille de la table et donc lenteur d'accès sauf si ce sont des icônes

Création de la table :

Id, nom du fichier, données, type de données (type MIME).

```
DROP TABLE IF EXISTS cours.blobs;


CREATE TABLE  cours.blobs (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom varchar(45) COLLATE utf8_unicode_ci NOT NULL,
  blobdata mediumblob NOT NULL,
  blobtype varchar(45) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (id)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Remarques :

Blobtype prend les valeurs suivantes : image/png, image/jpg, ..., application/pdf, etc.

```
| SELECT * FROM blobs b;
```

Insertion et visualisation d'un objet avec MySQL Query Browser.

BLOB  : insérer, visualiser, modifier, enregistrer, supprimer.



CHAPITRE 15 - DIVERS

15.1 - LES ÉVÉNEMENTS (> 5.1)

- **Objectif**

Créer un événement pour effectuer une action, une commande SQL.
L'événement peut gérer une action récurrente (un archivage tous les mois) ou unique (une suppression de données dans 2 minutes ...).

- **Syntaxes**

Vérifier le statut (Version de MySQL, état de l'activation du scheduler, éventuellement modification de son statut).

```
SELECT version();  
  
SHOW GLOBAL VARIABLES LIKE 'event_scheduler';  
  
SET GLOBAL event_scheduler = 1;
```

Créer un événement.

```
CREATE EVENT [bd.]nom_d_evenement  
ON SCHEDULE horaire  
DO commandeSQL;
```

Horaire peut être défini de deux façons : absolu ou relatif.
ON SCHEDULE AT heure [+ INTERVAL intervalle]
ON SCHEDULE intervalle

Modifier un événement.

```
ALTER EVENT [bd.]nom_d_evenement ...;
```

Supprimer un événement.

```
DROP EVENT [IF EXISTS] [bd.]nom_d_evenement;
```

Lister les événements de la BD courante.

```
SHOW EVENTS [FROM bd];
```

• Exemples

```
DROP TABLE IF EXISTS cours.villes_bis;  
CREATE TABLE cours.villes_bis AS SELECT * FROM villes;
```

```
CREATE EVENT cours.dans_2_minutes  
ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 2 MINUTE  
DO DELETE FROM cours.villes_bis;
```

```
SHOW EVENTS FROM cours;
```

```
SELECT * FROM cours.villes_bis v;
```

```
CREATE EVENT cours.tous_les_mois  
ON SCHEDULE EVERY 1 MONTH  
DO DELETE FROM cours.villes_bis;
```

15.2 - QUELQUES ÉLÉMENTS META-BASIQUES

15.2.1 - La commande SHOW

- **Objectif**

Permet de récupérer les caractéristiques d'un objet.

- **Syntaxe**

```
SHOW objet [FROM objet_parent [LIKE '%...']];
```

- **Exemples**

```
-- Liste des bases
SHOW DATABASES;

-- Liste des tables d'une base
SHOW TABLES FROM cours;
SHOW TABLES FROM cours LIKE 'V%';

-- Liste des colonnes d'une table
SHOW FIELDS FROM villes;
SHOW COLUMNS FROM villes;

-- Liste des clés d'une table
SHOW KEYS FROM villes;

-- Liste des index d'une table
SHOW INDEX FROM villes;

-- Visualisation de l'instruction de création d'une table
SHOW CREATE TABLE villes;
```

15.2.2 - La commande DESC

Objectif :

Afficher la structure d'une table.

Syntaxe :

```
DESC nom_de_table;
```

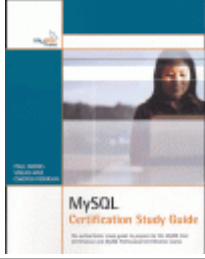
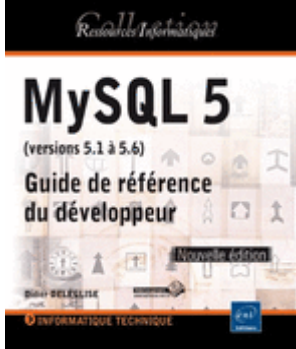
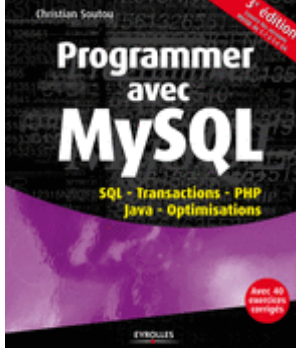
Exemple :

```
DESC villes;
```

Field	Type	Null	Key	Default	Extra
cp	varchar(5)	NO	PRI		
nom_ville	varchar(50)	NO			
site	varchar(50)	YES			
photo	varchar(50)	YES			
id_pays	varchar(7)	NO	MUL		

CHAPITRE 16 - ANNEXES

16.1 - BIBLIOGRAPHIE

<p>MySQL 5 - Guide officiel de Paul DuBois, Stefan Hinz, Carsten Pedersen</p> <p>Editions MySQL Press, juillet 2006 700 pages, 1285 g</p> <p>La référence</p>	
<p>MySQL 5 Guilde de référence du développeur de Didier DELEGLISE</p> <p>Editions ENI, Avril 2013 484 pages EAN13 : 9782746079724.</p>	
<p>Programmer avec MySQL de Christian Soutou</p> <p>Editions Eyrolles, Février 2011, 450 pages, 1040 g, Format : 19 x 23 ISBN13: 978-2-212-12869-7. Excellent.</p>	

16.2 - DOCUMENTATION (SITE OFFICIEL)

<http://dev.mysql.com/doc/refman/5.0/fr/>

16.3 - CRITIQUES DE MySQL

http://blog.developpez.com/sqlpro/p9136/langage-sql-norme/mysql_un_sgbdr_poudre_aux_yeux

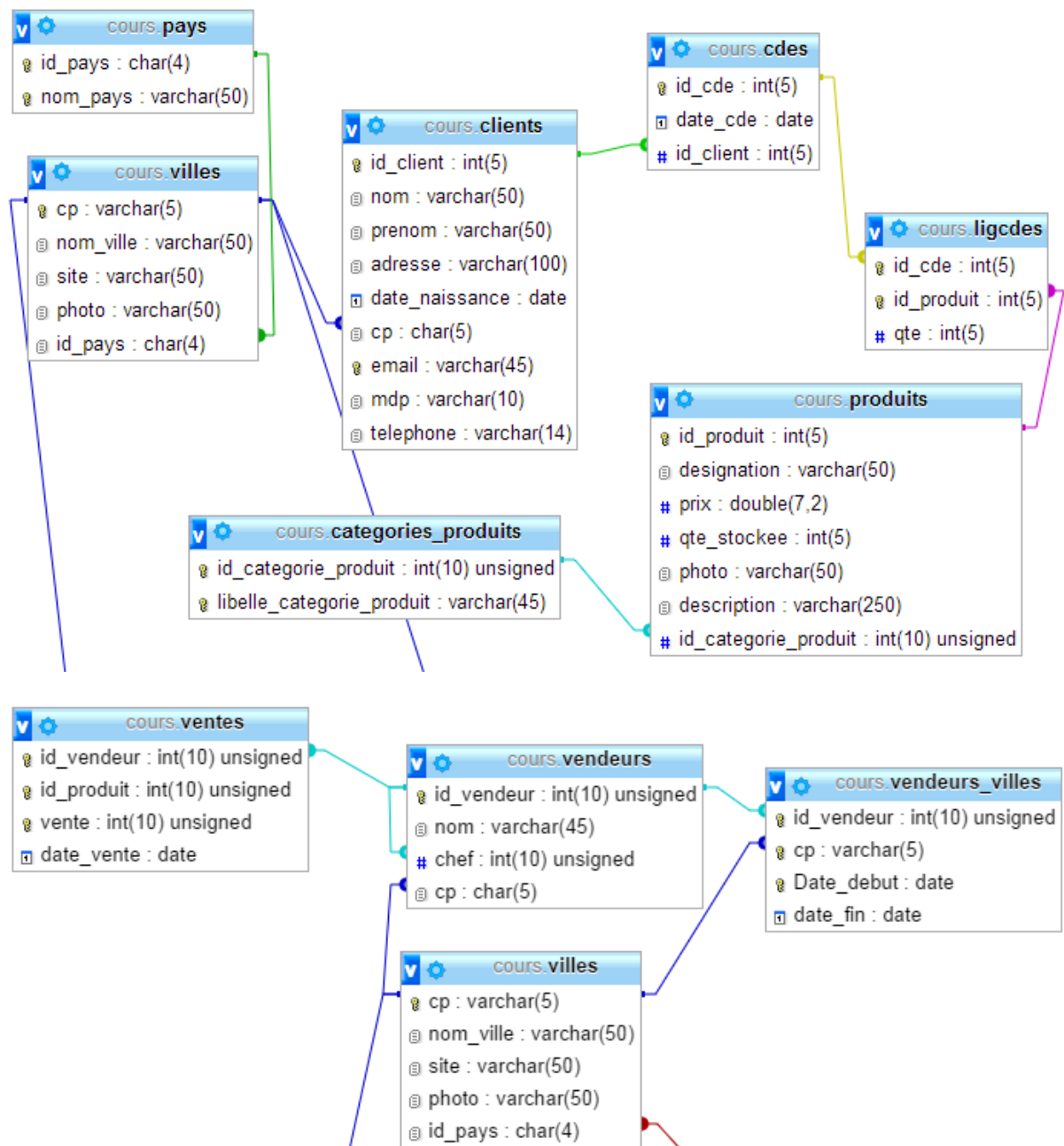
16.4 - A RETENIR ... ABSOLUMENT

Les principaux concepts ...



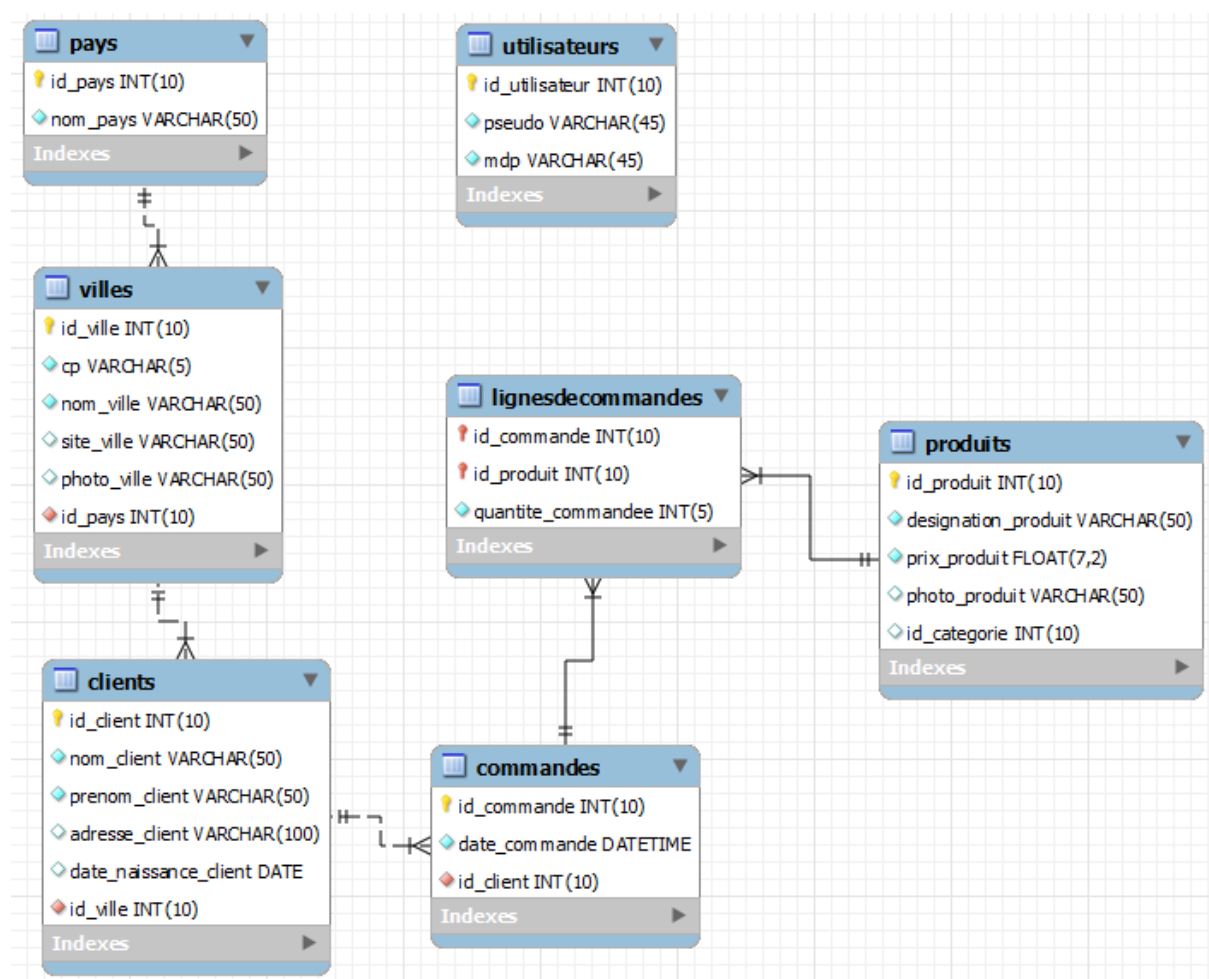
Concept	Description
Architecture n-tiers	Architecture matérielle multi-machines supportant une application (par exemple client, serveur d'application, serveur de BD).
Définition d'un serveur de BD	Machine hébergeant un SGBD.
Définition d'un SGBDR	Un SGBDR est un Système de Gestion de Bases de Données Relationnelles . C'est-à-dire un ensemble de logiciels capable de gérer une base de données relationnelle.
Définition d'une BDR	Une BDR (base de données relationnelle) est un ensemble de tables bien souvent reliées entre elles (il peut exister des tables paramètre indépendantes) qui modélisent un domaine du SI d'une organisation.
Gestion des objets	CREATE, DROP, ALTER
Ajout de données	INSERT
Récupération de données	SELECT
Suppression de données	DELETE
Modification de données	UPDATE
Récupération de données provenant de plusieurs tables	Jointure
Faire des calculs	Requêtes calculées ou requêtes agrégats
Fusionner plusieurs tables	Requêtes ensemblistes
Obtenir des résultats en fonction d'une requête	Requêtes imbriquées
Gérer les mises à jour	Les transactions
Filtrer les données et gérer les accès	Les vues et GRANT

16.5 - LA BD COURS

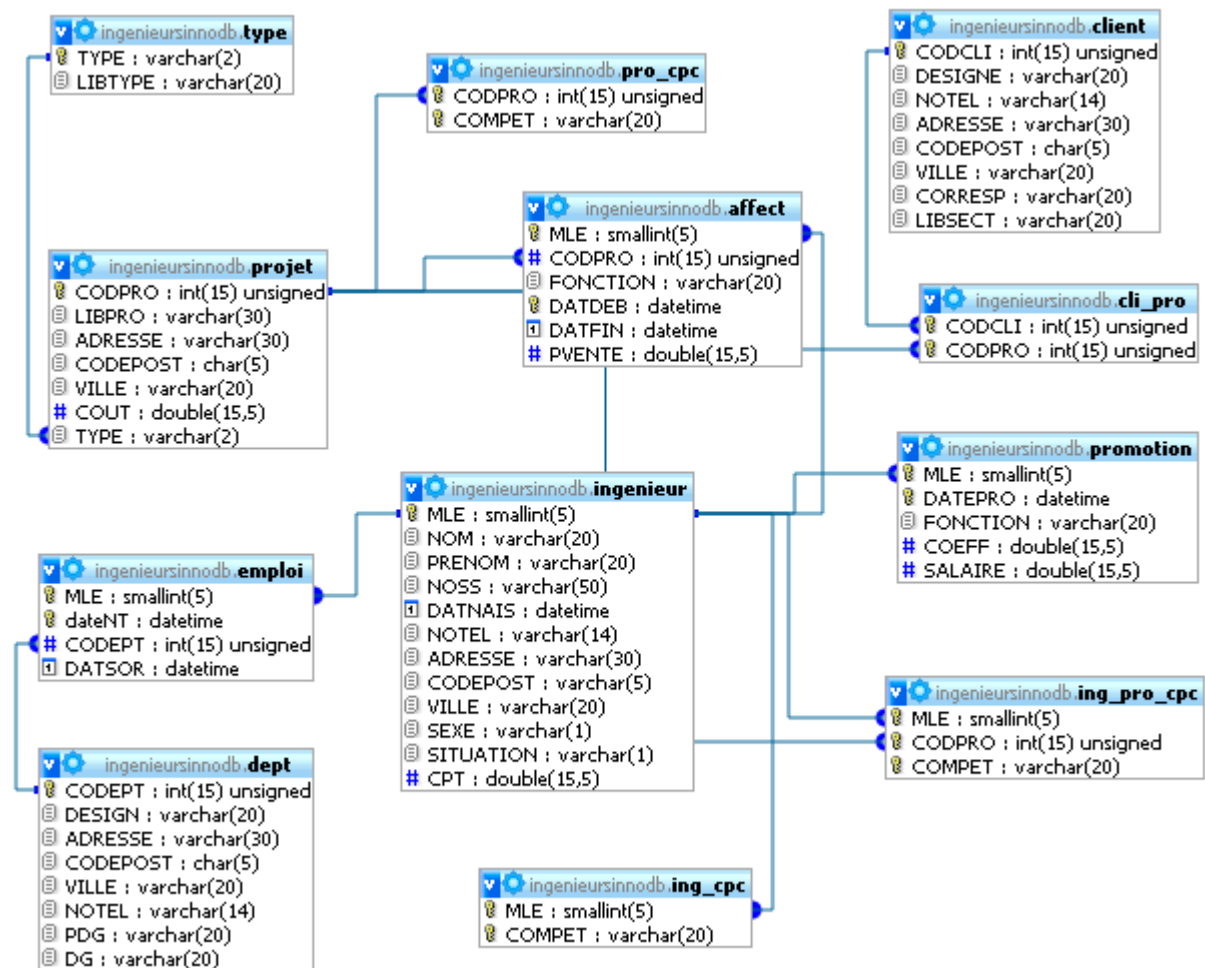


cf cours_create.sql et cours_insert.sql.

16.6 - LA BD COURS_REDUIT_2014



16.7 - LA BD INGÉNIEURS

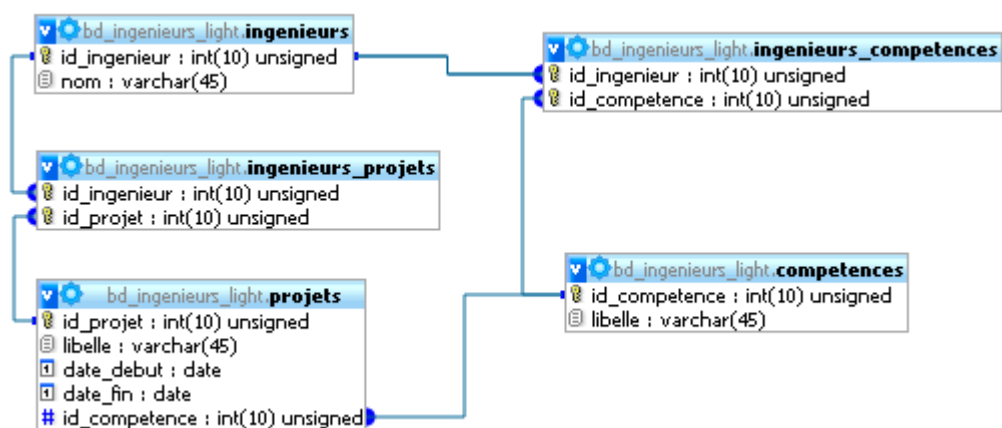


Cf les scripts :

bd_ingenieurs_innodb.sql,
bd_ingenieurs_myisam.sql,

bd_ingenieurs_inserts.sql.

16.8 - LA BD INGÉNIEURS LIGHT



cf bd_ingenieurs_light_create_inserts.sql.

16.9 - LES CHARSETS ET LES COLLATIONS

<http://dev.mysql.com/doc/refman/5.0/en/charset-general.html>

Character Sets and Collations in General

A character set is a set of symbols and encodings. A collation is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: "A", "B", "a", "b". We give each letter a number: "A" = 0, "B" = 1, "a" = 2, "b" = 3. The letter "A" is a symbol, the number 0 is the encoding for "A", and the combination of all four letters and their encodings is a character set.

Suppose that we want to compare two string values, "A" and "B". The simplest way to do this is to look at the encodings: 0 for "A" and 1 for "B". Because 0 is less than 1, we say "A" is less than "B". What we've just done is apply a collation to our character set. The collation is a set of rules (only one rule in this case): "compare the encodings." We call this simplest of all possible collations a binary collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters "a" and "b" as equivalent to "A" and "B"; (2) then compare the encodings. We call this a case-insensitive collation. It is a little more complex than a binary collation.

In real life, most character sets have many characters: not just "A" and "B" but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether to distinguish lettercase, but also for whether to distinguish accents (an "accent" is a mark attached to a character as in German "Ö"), and for multiple-character mappings (such as the rule that "Ö" = "OE" in one of the two German collations).

MySQL can do these things for you:

Store strings using a variety of character sets

Compare strings using a variety of collations

Mix strings with different character sets or collations in the same server, the same database, or even the same table

Enable specification of character set and collation at any level

In these respects, MySQL is far ahead of most other database management systems. However, to use these features effectively, you need to know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

Cf aussi <http://fr.wikipedia.org/wiki/UTF-8>

Cf aussi <http://www.utf8-chartable.de/>

Exemples :

```
CREATE TABLE cours.charset_tests (  
  id int(10) unsigned NOT NULL AUTO_INCREMENT,  
  nom_utf8_general_ci varchar(45) NOT NULL,  
  nom_utf8_unicode_ci varchar(45) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT  
NULL,  
  nom_utf8_bin varchar(45) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,  
  nom_latin1_ci varchar(45) CHARACTER SET latin1 COLLATE latin1_general_ci NOT  
NULL,  
  nom_latin1_cs varchar(45) CHARACTER SET latin1 COLLATE latin1_general_cs NOT  
NULL,  
  PRIMARY KEY (id)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
INSERT INTO charset_tests (id, nom_utf8_general_ci, nom_utf8_unicode_ci,  
nom_utf8_bin, nom_latin1_ci, nom_latin1_cs) VALUES  
(1, 'Dupont', 'Dupont', 'Dupont', 'Dupont', 'Dupont'),  
(2, 'Tintin', 'Tintin', 'Tintin', 'Tintin', 'Tintin'),  
(3, 'Milou', 'Milou', 'Milou', 'Milou', 'Milou'),  
(4, 'Daphnée', 'Daphnée', 'Daphnée', 'Daphnée', 'Daphnée'),  
(5, 'milou', 'milou', 'milou', 'milou', 'milou'),  
(6, 'pepe', 'pepe', 'pepe', 'pepe', 'pepe'),  
(7, 'pépé', 'pépé', 'pépé', 'pépé', 'pépé'),  
(8, 'pèpè', 'pèpè', 'pèpè', 'pèpè', 'pèpè'),  
(9, 'pfff', 'pfff', 'pfff', 'pfff', 'pfff'),  
(10, 'pêpê', 'pêpê', 'pêpê', 'pêpê', 'pêpê');
```

SELECT ... WHERE ...

```
SELECT * FROM charset_tests c WHERE c.nom_utf8_general_ci = 'pepe';
SELECT * FROM charset_tests c WHERE c.nom_utf8_unicode_ci = 'pepe';

SELECT * FROM charset_tests c WHERE c.nom_utf8_bin = 'pepe';
SELECT * FROM charset_tests c WHERE c.nom_latin1_ci = 'pepe';
SELECT * FROM charset_tests c WHERE c.nom_latin1_cs = 'pepe';
```

Les 2 premiers affichent pepe, p    , p     et p    .
Les 3 derniers affichent pepe.

Changement de COLLATION à la volée :

```
-- Donne le meme resultat que 1 et 2
SELECT * FROM charset_tests c WHERE c.nom_utf8_bin = 'pepe'
COLLATE utf8_general_ci;

-- Donne le meme resultat que 3
SELECT * FROM charset_tests c WHERE c.nom_utf8_general_ci = 'pepe'
COLLATE utf8 bin;
```

Notes : la collation doit correspondre au CHARSET (Par exemple COLLATE latin1_general_cs provoque une erreur dans ce cas-ci).

```
SELECT * FROM charset_tests c WHERE c.nom_utf8_bin = 'pepe'
COLLATE latin1_general_ci;
Ceci est KO.
```

Mais ceci est OK

```
-- Donne le meme resultat que 1 et 2
SELECT * FROM charset_tests c WHERE c.nom_latin1_cs = 'pepe'
COLLATE utf8 general ci;
```

SELECT ... ORDER BY ...

```
SELECT * FROM charset_tests c ORDER BY c.nom_utf8_general_ci;  
SELECT * FROM charset_tests c ORDER BY c.nom_utf8_unicode_ci;  
SELECT * FROM charset_tests c ORDER BY c.nom_utf8_bin;  
SELECT * FROM charset_tests c ORDER BY c.nom_latin1_ci;  
SELECT * FROM charset_tests c ORDER BY c.nom_latin1_cs;
```

- 1 - pèpè, pépé, pepe, pêpê, pfff
- 2 - pepe, pépé, pèpè, pêpê, pfff
- 3 - pepe, pff, pèpè, pépé, pêpê
- 4 - pepe, pèpè, pépé, pêpê, pfff
- 5 - pepe, pèpè, pépé, pêpê, pfff

4 et 5 identiques

ASCII étendue version IBM :

e = 101 = x65
f = 102 = x66
é = 130 = x82
ê = 136 = x88
è = 138 = x8A

UTF-8 :

e = 101 = x65
f = 102 = x66
è = 232 = xE8
é = 233 = xE9
ê = 234 = xEA

cf aussi <http://hapax.qc.ca/conversion.fr.html>

16.10 - LES COLONNES DE TYPE TEXTE BINAIRE

Admettons la table `textesbinairesnonbinaires(id, texteBinaire, texteNonBinaire)`.

```
DROP TABLE IF EXISTS textesbinairesnonbinaires;

CREATE TABLE IF NOT EXISTS textesbinairesnonbinaires (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  texteBinaire varbinary(45) DEFAULT NULL,
  texteNonBinaire varchar(45) DEFAULT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO textesbinairesnonbinaires (texteBinaire, texteNonBinaire) VALUES
('event', 'event'),
('événement', 'événement'),
('evenement', 'evenement'),
('ève', 'ève'),
('Eve', 'Eve');
```

Tri binaire

```
SELECT *  
FROM textesbinairesnonbinaires t  
ORDER BY texteBinaire;
```

id	texteBinaire	texteNonBinaire
5	Eve	Eve
3	evenement	evenement
1	event	event
4	ève	ève
2	événement	événement

Le tri est effectué en fonction de la table UTF-8.
(E : x45=69, e : x65=101, è : xE8=232, é : xE9=233)

Tri non binaire

```
SELECT *  
FROM textesbinairesnonbinaires t  
ORDER BY texteNonBinaire;
```

id	texteBinaire	texteNonBinaire
4	ève	ève
5	Eve	Eve
2	événement	événement
3	evenement	evenement
1	event	event

Le tri est effectué en faisant abstraction des caractères accentués.

16.11 - MySQL ET LES EXPRESSIONS RÉGULIÈRES

Une expression régulière permet de réaliser une recherche complexe.

MySQL utilise l'implémentation de Henry Spencer des expressions régulières qui tend à être conforme à la norme POSIX 1003.2.

La fonction MySQL qui permet d'utiliser les ER est REGEXP. Elle renvoie 1 si la chaîne de caractères correspond au motif et 0 dans le cas contraire.

A partir de la version 3.23.4 de MySQL, REGEXP est insensible à la casse pour les comparaisons de chaînes non binaires.

Le motif d'une expression régulière décrit un jeu de chaînes de caractères, une structure. Il est composé d'une chaîne de caractères et de meta-caractères.

Le motif le plus simple est celui qui ne comporte pas de caractères spéciaux.

Par exemple, l'expression régulière 'Paris' trouvera les enregistrements contenant 'Paris' ou 'paris' etc.

```
SELECT *
FROM villes
WHERE nom_ville REGEXP 'paris';
```

Mais dans ce cas l'utilisation de REGEXP ne sert à rien, de plus elle est plus coûteuse qu'une recherche exacte ou même qu'une recherche avec l'opérateur LIKE.

Les expressions régulières non-triviales utilisent des meta-caractères.

cf la documentation officielle pour la liste des meta-caractères.

Par exemple, l'expression régulière 'paris|rome' trouve la chaîne 'paris' ou la chaîne 'rome'.

```
SELECT *
FROM villes
WHERE nom_ville REGEXP 'paris|rome';
```

Autre exemple avec l'utilisation des meta-caractères '.' (n'importe quel caractère) et '?' (0,n).

```
-- Affiche les noms des clients et 0 ou 1 si celui-ci contient un trait d'union ou
"de" ou "De".
SELECT nom, nom REGEXP '.*-|de.*'
FROM clients;
```

Autre exemple pour le contrôle du CP dans la table villes :

```
SELECT cp, cp REGEXP '^[0-9]{5}$'
FROM villes;
```

Pour plus de détails cf la documentation officielle.

16.12 - IF

Le IF implémente une ternaire.

Point de départ : une jointure entre les tables [produits] et [categories_produits].

```
SELECT c.libelle_categorie_produit AS Catégorie, p.designation AS Désignation
FROM produits p INNER JOIN categories_produits c
ON p.id_categorie_produit = c.id_categorie_produit;
```

EXPLAIN : 5+1

Catégorie	Désignation
Champagnes	Ruinard
Champagnes	Dom Pérignon
Eaux	Evian
Eaux	Badoit
Eaux	Vichy Célestins
Eaux	Vichy Saint-Yorre
Sodas	Coca-Cola
Sodas	Coca-Cola light
Sodas	Coca-Cola zéro

Rappel de la fonction GROUP_CONCAT : concatène en fonction d'un regroupement.

```
SELECT c.libelle_categorie_produit AS Catégorie, GROUP_CONCAT(designation ORDER BY
designation SEPARATOR ' - ') AS Désignation
FROM produits p INNER JOIN categories_produits c
ON p.id_categorie_produit = c.id_categorie_produit
GROUP BY c.libelle_categorie_produit;
```

EXPLAIN : 5+1

Catégorie	Désignation
Champagnes	Dom Pérignon - Ruinard
Eaux	Badoit - Evian - Vichy Célestins - Vichy Saint-Yorre
Sodas	Coca-Cola - Coca-Cola light - Coca-Cola zéro

Syntaxe du IF

`IF(condition, vrai, faux) [AS] alias]`

Une expression ternaire. Faux est obligatoire.

```
SELECT p.designation AS Désignation,  
IF (id_categorie_produit=1, 'Eaux', '') Eaux,  
IF (id_categorie_produit=2, 'Sodas', '') Sodas  
FROM produits p;
```

EXPLAIN : 12

Désignation	Eaux	Sodas
Evian	Eaux	
Badoit	Eaux	
Ruinard		
Dom Pérignon		
Coca-Cola		Sodas
Coca-Cola light		Sodas
Coca-Cola zéro		Sodas
Vichy Célestins	Eaux	
Vichy Saint-Yorre	Eaux	

16.13 - CASE WHEN

Implémentation d'une structure conditionnelle de type CASE.

Syntaxe

```
SELECT colonne1 [[AS] alias1],
CASE colonne2
WHEN 1 THEN 'Constante1'
WHEN 2 THEN 'Constante2'
ELSE 'Constante3'
END
[[AS] alias2]
FROM table alias ;
```

```
SELECT p.designation AS Désignation,
CASE p.id_categorie_produit
WHEN 1 THEN 'Eaux'
WHEN 2 THEN 'Sodas'
ELSE 'Autre'
END
AS Catégorie
FROM produits p ;
```

EXPLAIN : 12

Désignation	Catégorie
Evian	Eaux
Badoit	Eaux
Ruinard	Autre
Dom Pérignon	Autre
Coca-Cola	Sodas
Coca-Cola light	Sodas
Coca-Cola zéro	Sodas
Vichy Célestins	Eaux
Vichy Saint-Yorre	Eaux

Cf aussi BOF !!!

```
SELECT p.designation AS Désignation, `Eaux`  
  FROM produits p  
 WHERE id_categorie_produit=1  
UNION  
  SELECT designation, `Sodas`  
  FROM produits  
 WHERE id_categorie_produit=2  
UNION  
  SELECT designation, `Champagnes`  
  FROM produits  
 WHERE id_categorie_produit= 6  
;
```

EXPLAIN : 4+3+2

Désignation	Catégorie
Evian	Eaux
Badoit	Eaux
Vichy Célestins	Eaux
Vichy Saint-Yorre	Eaux
Coca-Cola	Sodas
Coca-Cola light	Sodas
Coca-Cola zéro	Sodas
Ruinard	Champagnes
Dom Pérignon	Champagnes

16.14 - FIELD, ELT FIND_IN_SET(), SUBSTRING_INDEX() & CO

FIELD(), ELT(), FIND_IN_SET(), SUBSTRING_INDEX().

Objectif :

La fonction **FIELD()** : retourne l'index de la chaîne str dans la liste str1, str2, str3, ... et retourne 0 si str n'est pas trouvé. La fonction FIELD() est le complément de la fonction ELT().

```
FIELD('str', 'str1','str2','str3')
```

```
| SELECT FIELD('b', 'a','b','c'); -- Renvoie 2
```

La fonction **ELT()** : retourne la chaîne située à la position p dans une liste de chaînes.

Retourne str1 si N = 1, str2 si N = 2, et ainsi de suite. Retourne NULL si N est plus petit que 1 ou plus grand que le nombre d'arguments. La fonction ELT() est un complément de la fonction FIELD().

```
ELT(position, 'str1','str2','str3')
```

```
| SELECT ELT(2, 'a','b','c'); -- Renvoie 'b'
```

```
| SELECT ELT(1, designation, prix) AS `Désignation`,  
| ELT(2, designation, prix) AS `Prix`  
| FROM produits p;
```

Fonction identique :

```
MAKE_SET(position, 'str1','str2','str3')
```

```
| SELECT MAKE_SET(2, 'a','b','c'); -- Renvoie 'b'
```

La fonction **FIND_IN_SET()** : retourne la position d'une sous-chaîne dans une chaîne.

Retourne une valeur de 1 à N si la chaîne str se trouve dans la liste strlist constituée de N chaînes. Une liste de chaînes est une chaîne composée de sous-chaînes séparées par une virgule ','. Si le premier argument est une chaîne constante et le second, une colonne de type SET, la fonction FIND_IN_SET() est optimisée pour utiliser une recherche binaire très rapide. Retourne 0 si str n'est pas trouvé dans la liste strlist ou si la liste strlist est une chaîne vide. Retourne NULL si l'un des arguments est NULL. Cette fonction ne fonctionne pas correctement si le premier argument contient une virgule.

```
FIND_IN_SET('str', 'str1,str2,str3');
```

```
SELECT FIND_IN_SET('b', 'a,b,c'); -- Renvoie 2
```

La fonction **SUBSTRING_INDEX()** : retourne la sous-chaîne située avant le séparateur.

```
SUBSTRING_INDEX('chaîne des sous-chaînes séparées', 'séparateur', position)
```

Note : il faut imbriquer deux SUBSTRING_INDEX() pour récupérer une sous-chaîne au milieu ou la à fin de la chaîne.

```
SELECT SUBSTRING_INDEX('Tintin,1,75011', ',', 1); -- Renvoie 'Tintin'
SELECT SUBSTRING_INDEX('Tintin,1,75011', ',', 2); -- Renvoie 'Tintin, 1'
SELECT SUBSTRING_INDEX(SUBSTRING_INDEX('Tintin,1,75011', ',', 2), ',', -1); --
Renvoie '1'
SELECT SUBSTRING_INDEX(SUBSTRING_INDEX('Tintin,1,75011', ',', 3), ',', -1); --
Renvoie '75011'
```

Exemple dans une procédure stockée

```
DELIMITER $$

DROP PROCEDURE IF EXISTS coursmysql2013.vendeurs_insert $$
CREATE PROCEDURE coursmysql2013.vendeurs_insert (IN donnees VARCHAR(200))
BEGIN

    -- CALL vendeurs_insert('Blanche-Neige,2,75011');

    DECLARE lsNom VARCHAR(50);
    DECLARE liChef INTEGER;
    DECLARE lsCP VARCHAR(50);

    SET lsNOM = SUBSTRING_INDEX(donnees, ',', 1);
    SET liChef = SUBSTRING_INDEX(SUBSTRING_INDEX(donnees, ',', 2), ',', -1);
    SET lsCP = SUBSTRING_INDEX(SUBSTRING_INDEX(donnees, ',', 3), ',', -1);

    INSERT INTO vendeurs(nom_vendeur, chef, cp_vendeur) VALUES(lsNom, liChef, lsCP);

END $$

DELIMITER ;
```

16.15 - WITH RECURSIVE

Implémentation d'une récursive.

N'existe pas en MySQL.

16.16 - LES ESPACES INTERVALLAIRES

Cf <http://sqlpro.developpez.com/cours/arborescence/>

Objectif :

Niveau 1	Niveau 2	Niveau 3	Niveau 4	Borne gauche	Borne droite
Racine	Méthodes				
		UML			
		Merise			
		Gestion de projet			
	Langages				
		PHP			
			Intro PHP		
			PDO		
		Java			
			Intro Java		
			JDBC		
		HTML			
		CSS			

Opérations :

afficher la racine (id=1),
 afficher tous les nœuds,
 afficher les nœuds d'un nœud (descendants directs),
 afficher les nœuds d'un nœud (tous les descendants),
 afficher le parent direct d'un nœud (ascendant direct),
 afficher les feuilles (borne_droite - borne_gauche = 1,

Ajout,
 suppression.

Racine

```
DELIMITER $$

DROP PROCEDURE IF EXISTS hierarchies.racine $$

CREATE PROCEDURE hierarchies.racine ()
BEGIN
    SELECT m.libelle_module
    FROM modules m
    WHERE id_module = 1;
END $$

DELIMITER ;

CALL racine();
```

Parent direct (Méthodes pour UML, Langages pour Java, Java pour JSP, ...)

```
DELIMITER $$

DROP PROCEDURE IF EXISTS hierarchies.parent_direct $$

CREATE PROCEDURE hierarchies.parent_direct (asModule VARCHAR(50))
BEGIN
    SELECT m.libelle_module
    FROM modules m
    WHERE id_module =
        (SELECT id_module_parent FROM modules WHERE libelle_module = asModule);
END $$

DELIMITER ;

CALL parent_direct('merise'); -- Renvoie Méthodes
```


Toutes les feuilles

```
DELIMITER $$

DROP PROCEDURE IF EXISTS hierarchies.feuilles $$

CREATE PROCEDURE hierarchies.feuilles ()
BEGIN
    SELECT m.libelle_module
    FROM modules m
    WHERE m.borne_droite - m.borne_gauche = 1;
END $$

DELIMITER ;

CALL feuilles();
```

Compter le nombre de feuilles

```
DELIMITER $$

DROP PROCEDURE IF EXISTS hierarchies.feuilles_compte $$
CREATE PROCEDURE hierarchies.feuilles_compte ()
BEGIN
    SELECT COUNT(*)
    FROM modules m
    WHERE m.borne_droite - m.sborne_gauche = 1;
END $$

DELIMITER ;

CALL feuilles_compte();
```

Tous les nœuds (Racine, Méthodes, Langages, Java, PHP)

```
DELIMITER $$

DROP PROCEDURE IF EXISTS noeuds $$

CREATE DEFINER=root@localhost PROCEDURE noeuds()
BEGIN
    SELECT m.libelle_module
    FROM modules m
    WHERE borne_droite - borne_gauche > 1;
END $$

DELIMITER ;

CALL noeuds();
```

Enfants (directs et indirects)

```
DELIMITER $$

DROP PROCEDURE IF EXISTS hierarchies.enfants_de $$

CREATE PROCEDURE hierarchies.enfants_de (asParent VARCHAR(50))
BEGIN
    SELECT m.libelle_module
    FROM modules m
    WHERE m.borne_gauche > (SELECT borne_gauche FROM modules WHERE libelle_module =
asParent)
    AND borne_droite < (SELECT borne_droite FROM modules WHERE libelle_module =
asParent);
END $$

DELIMITER ;

CALL enfants_de('Méthodes'); -- Renvoie Merise, UML, gestion de projet
CALL enfants_de('Langages'); -- Renvoie PHP, PHP intro, PDO, Java, Java intro,
JDBS, JSP
```

Enfants directs

```
CALL enfants_de('méthodes'); -- Renvoie Merise, UML, gestion de projet
```

Insertion

Nous faisons l'insertion à droite en précisant le nom du module à insérer et le nom du parent considérant que le nom d'un module, d'un sous-module est unique.

Il faut donc augmenter de 2 unités les bornes droites et de 2 unités les bornes gauche des éléments à décaler.

Puis insérer l'élément avec comme valeur de la borne gauche l'ancienne valeur de la borne droite de l'élément parent et comme valeur de la borne gauche la valeur de l'ancienne valeur de la borne gauche du parent + 1.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS insertion $$
CREATE DEFINER=root@localhost PROCEDURE insertion(asModule VARCHAR(50),
asModuleParent VARCHAR(50))
BEGIN
-- INSERTION A DROITE
DECLARE liModuleParent INTEGER;
DECLARE liBorneDroiteParent INTEGER;
-- DECLARE liBorneGaucheParent INTEGER;

SELECT id_module, borne_droite INTO liModuleParent, liBorneDroiteParent
FROM modules
WHERE libelle_module = asModuleParent;

-- CREATE TABLE IF NOT EXISTS modules_tempo AS SELECT * FROM modules;

-- UPDATE modules
-- SET borneDroite = borneDroite + 2
-- WHERE borneDroite >= (SELECT borneDroite FROM modules_tempo WHERE libelle_module
= asModule);

-- UPDATE modules
-- SET borneGauche = borneGauche + 2
-- WHERE borneGauche >= (SELECT borneGauche FROM modules_tempo WHERE libelle_module
= asModule);

-- On decale toutes les bornes droites de 2 unites
-- pour les bornes droites allant de la borne droite du parent
-- a toutes les autres qui sont superieures
UPDATE modules
SET borne_droite = borne_droite + 2
WHERE borne_droite >= liBorneDroiteParent;

-- On decale toutes les bornes gauches de 2 unites
-- pour les bornes gauches allant de la borne gauche du premier suivant du parent
-- a toutes les autres qui sont superieures
-- sauf pour la RACINE
UPDATE modules
SET borne_gauche = borne_gauche + 2
WHERE borne_droite > liBorneDroiteParent + 2
AND borne_gauche <> 1;

-- Insertion dans la table du nouveau
INSERT INTO modules(borne_gauche, borne_droite, libelle_module, id_module_parent)
VALUES (liBorneDroiteParent, liBorneDroiteParent+1, asModule, liModuleParent);

-- DROP TABLE IF EXISTS modules_tempo;
END $$

DELIMITER ;
```

16.17 - EXPLOSER UNE COLONNE

Objectif

Séparer une colonne en 2 colonnes en fonction d'un caractère séparateur.

designation	SUBSTRING(designation, 1, INSTR(designation, ' '))	SUBSTRING(designation, INSTR(designation, ' ')+1)
Badoit		Badoit
Coca		Coca
Coca Light	Coca	Light
Coca Zéro	Coca	Zéro
Crémant		Crémant
Dom Pérignon	Dom	Pérignon
Evian		Evian
Fanta		Fanta
Graves		Graves
Picpoul		Picpoul
Ruinard		Ruinard

Exemple

```
SELECT designation,  
SUBSTRING(designation, 1, INSTR(designation, ' ')),  
SUBSTRING(designation, INSTR(designation, ' ')+1)  
FROM produits;
```

Utilisation de la fonction INSTR() pour récupérer la position du caractère séparateur et de la fonction SUBSTRING() pour extraire une sous-chaîne.

Cf aussi ce qui a été fait avec SUBSTRING_INDEX().

16.18 - LA DIVISION EN SQL

16.18.1 - Définition de la division en algèbre relationnelle

La Division relationnelle :

Objectif :

$R' = R1 \text{ div } R2$

La division est une opération portant sur deux relations, la seconde étant une sous relation de la première (c'est à dire que tous les attributs de la seconde font partie de la première).

La relation résultante est composée des attributs de la première relation qui ne sont pas attributs de la seconde.

Les n-uplets composant cette relation sont le résultat de tests égalitaires entre les valeurs des attributs de la seconde relation et les valeurs des attributs correspondants dans la première ; on recopie les valeurs des attributs non concernés par le test qui vont devenir les n-uplets de la relation résultant de la division.

Exemple :

A partir des relations suivantes :

Competences(id_competence, formateur, competence)

Modules(id_module, module)

où les attributs [competence] et [module] s'appliquent au même domaine ('SQL', 'Merise', 'PHP', 'Java', ...),

il sera possible de faire la division $R' = \text{competences.competence DIV modules.module}$.

Cf aussi la définition de la division de Laurent Audibert ("Bases de données - de la modélisation au SQL (cours et exercices)" ed. Ellipses) sur Wikipedia.

16.18.2 - La division en SQL - exemple

N'existe pas comme opérateur SQL.

Exemple avec MODULES et COMPETENCES

```
DROP DATABASE IF EXISTS division;

CREATE DATABASE IF NOT EXISTS division
DEFAULT CHARACTER SET utf8
COLLATE utf8_unicode_ci;

USE division;

DROP TABLE IF EXISTS division.competences;

CREATE TABLE IF NOT EXISTS division.competences (
  id_competence int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom_formateur varchar(45) NOT NULL,
  competence varchar(45) NOT NULL,
  PRIMARY KEY (id_competence) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS division.modules;

CREATE TABLE IF NOT EXISTS division.modules (
  id_module int(10) unsigned NOT NULL AUTO_INCREMENT,
  module varchar(45) NOT NULL,
  PRIMARY KEY (id_module)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO division.competences (id_competence, nom_formateur, competence) VALUES
(1, 'Pascal', 'SQL'),
(2, 'Pascal', 'UML'),
(3, 'Pascal', 'Java'),
(4, 'Pascal', 'Merise'),
(5, 'Arno', 'UML'),
(6, 'Arno', 'Java'),
(7, 'Romain', 'Java'),
(8, 'Romain', 'UML'),
(9, 'Romain', 'Merise'),
(10, 'Romain', 'SQL');

INSERT INTO division.modules (id_module, module) VALUES
(1, 'SQL'),
(2, 'UML'),
(3, 'MERISE'),
(4, 'JAVA');
```

[module] de [modules] correspond à [competence] de [competences] !!!

Table Competences

id_competence	nom_formateur	competence
1	Pascal	UML
2	Pascal	Merise
3	Pascal	SQL
4	Pascal	Java
5	Arno	UML
6	Arno	Java
7	Romain	UML
8	Romain	Merise
9	Romain	SQL
10	Romain	Java

Table Modules

id_module	module
1	UML
2	MERISE
3	SQL
4	Java

Le résultat de la division : la liste des formateurs qui ont toutes les compétences de la table Modules.

nom_formateur
Pascal
Romain

Donc on va pouvoir extraire tous les enregistrements (les noms des formateurs) de COMPETENCES qui correspondent à tous ceux de MODULES.

La division :

```
SELECT nom_formateur
FROM competences
WHERE competence IN
  (SELECT module FROM modules)
GROUP BY nom_formateur
HAVING COUNT(*) =
  (SELECT COUNT(DISTINCT competence) FROM competences);
```

La division exacte (tous ceux qui ont toutes compétences pour tous les modules et seulement eux et pas d'autres) :

```
SELECT nom_formateur
FROM competences c LEFT JOIN (
  SELECT DISTINCT module
  FROM modules) m
ON c.competence = m.module
GROUP BY nom_formateur
HAVING COUNT(*) =
  (SELECT COUNT(DISTINCT module) FROM modules)
AND COUNT(c.competence) =
  (SELECT COUNT(DISTINCT module) FROM modules);
```

Résultat :

nom_formateur
Pascal
Romain

Si vous ajoutez dans la table [competences] :

'Romain', 'Zend'.

Le résultat sera :

nom_formateur
Pascal

puisque 'Romain' a une compétence supplémentaire qui n'est pas dans la liste des modules.

Avec une requête corrélée :

```
SELECT c1.nom_formateur FROM competences c1
WHERE NOT EXISTS (
  SELECT * FROM modules m
  WHERE NOT EXISTS (
    SELECT * FROM competences c2
    WHERE (c1.nom_formateur = c2.nom_formateur)
    AND (c2.compétence = m.module))
GROUP BY c1.nom_formateur
HAVING COUNT(*) =
(SELECT COUNT(DISTINCT module) FROM modules);
```

Tableau récapitulatif

Requête	Si aucune autre	Si une autre
R1	2	0
R2	2	1
R3	2	1

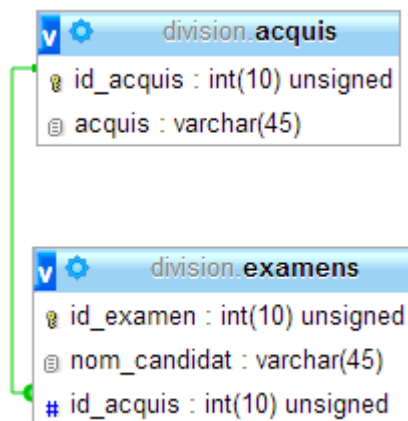
16.18.3 - Autre exemple

Autre exemple plus simple ... trop simple ... la réussite à l'examen.

Admettons ...

La question est : qui a réussi l'examen ?

Les tables :



Avec dans la table [acquis] les valeurs suivantes : Interface, Persistance et Architecture.
Et dans la table [examens] un enregistrement par candidat et par acquis.

La création et le remplissage des tables.

```
DROP TABLE IF EXISTS acquis;

CREATE TABLE IF NOT EXISTS acquis (
  id_acquis int(10) unsigned NOT NULL AUTO_INCREMENT,
  acquis varchar(45) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (id_acquis)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

INSERT INTO acquis (id_acquis, acquis) VALUES(1, 'Interface');
INSERT INTO acquis (id_acquis, acquis) VALUES(2, 'Persistance');
INSERT INTO acquis (id_acquis, acquis) VALUES(3, 'Architecture');


DROP TABLE IF EXISTS examens;

CREATE TABLE IF NOT EXISTS examens (
  id_examen int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom_candidat varchar(45) COLLATE utf8_unicode_ci NOT NULL,
  id_acquis int(10) unsigned NOT NULL,
  PRIMARY KEY (id_examen),
  KEY FK_examens_acquis (id_acquis)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

INSERT INTO examens (id_examen, nom_candidat, id_acquis) VALUES(1, 'Philippe', 1);
INSERT INTO examens (id_examen, nom_candidat, id_acquis) VALUES(2, 'Philippe', 2);
INSERT INTO examens (id_examen, nom_candidat, id_acquis) VALUES(3, 'Philippe', 3);
INSERT INTO examens (id_examen, nom_candidat, id_acquis) VALUES(4, 'Didier', 1);
INSERT INTO examens (id_examen, nom_candidat, id_acquis) VALUES(5, 'Didier', 2);
INSERT INTO examens (id_examen, nom_candidat, id_acquis) VALUES(6, 'Marc', 1);
INSERT INTO examens (id_examen, nom_candidat, id_acquis) VALUES(7, 'Marc', 2);
INSERT INTO examens (id_examen, nom_candidat, id_acquis) VALUES(8, 'Marc', 3);


ALTER TABLE examens
  ADD CONSTRAINT FK_examens_acquis FOREIGN KEY (id_acquis) REFERENCES acquis
(id_acquis);
```

La division : qui a le diplôme ?

```
SELECT nom_candidat
FROM examens
GROUP BY nom_candidat
HAVING COUNT(*) =
(SELECT COUNT(DISTINCT acquis) FROM acquis);
```

Maintenant ajoutons une colonne nommée [id_diplome] aux 2 tables.

Dans la table [acquis] affectez 1 à la colonne [id_diplome] à tous les enregistrements puis ajoutez des acquis et affectez 2 à la colonne [id_diplome] pour les nouveaux enregistrements.

Dans la table [examens] affectez 1 à tous les enregistrements.

La requête devient celle-ci :

```
SELECT e.nom_candidat, e.id_diplome
FROM examens e
GROUP BY e.nom_candidat
HAVING COUNT(*) =
(SELECT COUNT(DISTINCT a.acquis)
FROM acquis a
WHERE a.id_diplome = e.id_diplome);
```

```
SELECT nom_candidat
FROM examens e LEFT JOIN (
SELECT DISTINCT acquis
FROM acquis) a
ON e.acquis = a.acquis
GROUP BY nom_candidat
HAVING COUNT(*) =
(SELECT COUNT(DISTINCT acquis) FROM acquis)
AND COUNT(e.acquis) =
(SELECT COUNT(DISTINCT acquis) FROM acquis);
```

16.19 - LES TABLES MERGE

Objectif

Fusionner 2 tables MyISAM. Car il est préférable d'avoir plusieurs petites tables que de temps en temps l'on doit fusionner qu'une grande table.

Code

```
DROP TABLE IF EXISTS villes_france;
CREATE TABLE villes_france ENGINE=MyISAM AS SELECT cp, nom_ville FROM villes
WHERE id_pays = '33';
DESC villes_france;
SELECT * FROM villes_france;

DROP TABLE IF EXISTS villes_etrangeres;
CREATE TABLE villes_etrangeres ENGINE=MyISAM AS SELECT cp, nom_ville FROM villes
WHERE id_pays != '33';
DESC villes_etrangeres;
SELECT * FROM villes_etrangeres;

DROP TABLE IF EXISTS villes_tous_pays;
CREATE TABLE villes_tous_pays
(cp VARCHAR(5),
nom_ville VARCHAR(50)
)
ENGINE=MERGE
UNION=(villes_france,villes_etrangeres);
```

BOGUE : Toutes les tables de la table de type MERGE n'ont pas la même définition !!!

```
SELECT * FROM villes_tous_pays;
```

OK pour l'UNION

```
SELECT * FROM villes_france
UNION
SELECT * FROM villes_etrangeres;
```

Exercice :

Faites la même chose pour clients_francais et clients_etrangers.

```
SELECT * FROM clients_etrangers c;  
CREATE TABLE clients_francais ENGINE MYISAM AS SELECT * FROM clients ;  
SELECT * FROM clients_francais;
```

```
SELECT * FROM clients_tous_pays;
```

```
SELECT * FROM clients_tous_pays c;  
DROP TABLE IF EXISTS clients_tous_pays;
```

```
CREATE TABLE clients_tous_pays  
(id_client INT(10) NOT NULL AUTO_INCREMENT,  
nom VARCHAR(50),  
prenom VARCHAR(50),  
adresse VARCHAR(100),  
date_naissance DATE,  
cp VARCHAR(5),  
INDEX(id_client)  
)  
ENGINE=MERGE  
UNION=(clients_francais,clients_etrangers)  
INSERT_METHOD=LAST;
```

Exemple 2 :

```
CREATE TABLE mois (  
  id_mois INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,  
  num_mois TINYINT UNSIGNED NOT NULL,  
  nom_mois VARCHAR(9) NOT NULL,  
  PRIMARY KEY (id_mois)  
)  
ENGINE = InnoDB  
CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

```
INSERT INTO mois (num_mois, nom_mois)  
VALUES  
(1, 'Janvier'),  
(2, 'Février'),  
(3, 'Mars'),  
(4, 'Avril'),  
(5, 'Mai'),  
(6, 'Juin'),  
(7, 'Juillet'),  
(8, 'Août'),  
(9, 'Septembre'),  
(10, 'Octobre'),  
(11, 'Novembre'),  
(12, 'Décembre');
```

```
SELECT LAST_INSERT_ID();
```

Renverra 1 et non pas 12.

Mais si vous aviez exécuté ces commandes SQL :

```
INSERT INTO mois (num_mois, nom_mois)  
VALUES(1, 'Janvier');  
  
INSERT INTO mois (num_mois, nom_mois)  
VALUES(2, 'Février');  
  
INSERT INTO mois (num_mois, nom_mois)  
VALUES(3, 'Mars');  
  
INSERT INTO mois (num_mois, nom_mois)  
VALUES(4, 'Avril');  
  
INSERT INTO mois (num_mois, nom_mois)  
VALUES(5, 'Mai');  
  
INSERT INTO mois (num_mois, nom_mois)  
VALUES(6, 'Juin');  
  
INSERT INTO mois (num_mois, nom_mois)  
VALUES(7, 'Juillet');  
  
INSERT INTO mois (num_mois, nom_mois)  
VALUES(8, 'Août');  
  
INSERT INTO mois (num_mois, nom_mois)  
VALUES(9, 'Septembre');
```

```
INSERT INTO mois (num_mois, nom_mois)
VALUES (10, 'Octobre');
```

```
INSERT INTO mois (num_mois, nom_mois)
VALUES (11, 'Novembre');
```

```
INSERT INTO mois (num_mois, nom_mois)
VALUES (12, 'Décembre');
```

```
SELECT LAST_INSERT_ID();
```

Renverrait 12.

Exemple 3 :

Insertion d'une nouvelle commande et des lignes de commandes afférantes.

```
INSERT INTO cdes(date_cde, id_client) VALUES(CURDATE(), 1);
```

```
INSERT INTO ligcdes(id_cde, id_produit, qte) VALUES(LAST_INSERT_ID(), 1, 5);
INSERT INTO ligcdes(id_cde, id_produit, qte) VALUES(LAST_INSERT_ID(), 2, 10);
```

```
SELECT * FROM cdes c;
SELECT * FROM ligcdes l;
```


16.20 - LES GRANDS NOMBRES

1 000 : Mille = 10^3
1 000 000 : Million = 10^6
1 000 000 000 : Milliard = 10^9
1 000 000 000 000 : Billion = 10^{12}
1 000 000 000 000 000 : Billiard = 10^{15}
1 000 000 000 000 000 000 : Trillion = 10^{18}
1 000 000 000 000 000 000 000 : Trilliard = 10^{21}
1 000 000 000 000 000 000 000 000 : Quadrillion = 10^{24}
1 000 000 000 000 000 000 000 000 000 : Quadrilliard = 10^{27}
1 000 000 000 000 000 000 000 000 000 000 : Quintillion = 10^{30}
1 000 000 000 000 000 000 000 000 000 000 000 : Quintilliard = 10^{33}
1 000 000 000 000 000 000 000 000 000 000 000 000 : Sextillion = 10^{36}
1 000 000 000 000 000 000 000 000 000 000 000 000 000 : Sextilliard = 10^{39}

A mettre en relation avec les types numériques.

16.21 - LA TABLE DES COMMUNES DE FRANCE

Environ 38 950 enregistrements. Utile pour certains tests de performance.

```
DROP TABLE IF EXISTS cours.communes;
```

```
CREATE TABLE cours.communes (  
  commune varchar(45) COLLATE utf8_unicode_ci NOT NULL,  
  codepos varchar(45) COLLATE utf8_unicode_ci NOT NULL,  
  departement varchar(45) COLLATE utf8_unicode_ci NOT NULL,  
  insee varchar(45) COLLATE utf8_unicode_ci NOT NULL,  
  PRIMARY KEY (insee)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Import du fichier communes_insee.csv :

```
LOAD DATA INFILE 'c:/communes_insee.csv'  
INTO TABLE cours.communes  
FIELDS TERMINATED BY ';'   
LINES TERMINATED BY '\r\n'  
IGNORE 1 LINES ;
```

Notes : alors qu'avec phpMyAdmin c'est hyper lent !!!

16.22 - IMPORTATION/EXPORTATION AU FORMAT CSV

Pour plus de détails cf mysql_5_administration.odt.

Et aussi : <http://dev.mysql.com/doc/refman/5.0/en/select-into.html>

et <http://dev.mysql.com/doc/refman/5.0/en/load-data.html>

Exportation :

```
SELECT * INTO OUTFILE 'nom_de_fichier' [options d'export] FROM nom_de_table ...;
```

Importation :

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
[REPLACE | IGNORE]
INTO TABLE tbl_name
[FIELDS
  [TERMINATED BY '\t']
  [[OPTIONALLY] ENCLOSED BY '"']
  [ESCAPED BY '\\']
]
[LINES
  [STARTING BY '"']
  [TERMINATED BY '\n']
]
[IGNORE number LINES]
[(col_name,...)]
```

Exemples :

```
| LOAD DATA INFILE '/tempo/pays.txt' INTO TABLE pays_bis;
```

```
| SELECT * INTO OUTFILE '/tempo/pays.txt' FROM pays;
```

16.23 - JEUX DE CARACTÈRES ET UNICITÉ DES VALEURS

Exemple :

?	id	nom_ci	nom_cs	nom_bin
	1	Dupont	Dupont	Dupont
	2	René	René	Réné
	4	RENe	Rene	Renè

```

DROP TABLE IF EXISTS casse_tests;

CREATE TABLE IF NOT EXISTS casse_tests (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  nom_ci varchar(45) NOT NULL,
  nom_cs varchar(45) CHARACTER SET latin1 COLLATE latin1_general_cs NOT NULL,
  nom_bin varchar(45) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (id),
  UNIQUE KEY idx_uni_nom_cs (nom_cs),
  UNIQUE KEY idx_uni_nom_bin (nom_bin),
  UNIQUE KEY idx_uni_nom_ci (nom_ci) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

INSERT INTO casse_tests (id, nom_ci, nom_cs, nom_bin) VALUES
(1, 'Dupont', 'Dupont', 'Dupont'),
(2, 'René', 'René', 'Réné'),
(4, 'RENEE', 'Rene', 'Renè');

```

(*) La colonne nom_ci est CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL et c'est pour cela que les valeurs René, Rene, ... sont considérées comme identiques et refusées par l'index unique.

16.24 - ACCÉDER À DISTANCE À UN SERVEUR MySQL

Pour cela il faut créer un nouvel user sur le serveur distant qui aura les droits adéquats sur la BD et/ou les tables et/ou les vues.

Rappel des commandes CREATE USER et DROP USER :

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

```
DROP USER nom_du_user;
```

Rappel des commandes GRANT et REVOKE :

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
ON {tbl_name | * | *.* | db_name.*}
TO user [IDENTIFIED BY [PASSWORD] 'password']
[, user [IDENTIFIED BY [PASSWORD] 'password']] ...
[REQUIRE
  NONE |
  [{SSL| X509}]
  [CIPHER cipher [AND]]
  [ISSUER issuer [AND]]
  [SUBJECT subject]]
[WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR count |
      MAX_UPDATES_PER_HOUR count |
      MAX_CONNECTIONS_PER_HOUR count]]
```

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

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

Exemple :

Création du user :

```
CREATE USER p IDENTIFIED BY 'b';
```

Pour autoriser le client distant dont l'IP est 192.168.2.1 à se connecter avec p/b et avoir tous les droits sur toutes les Bds :

```
GRANT ALL ON *.* TO 'p'@'192.168.2.1';  
FLUSH PRIVILEGES;
```

Pour autoriser tous les clients distants à se connecter avec p/b et avoir tous les droits sur toutes les Bds :

```
CREATE USER p IDENTIFIED BY 'b';  
GRANT ALL ON *.* TO 'p'@'%';  
FLUSH PRIVILEGES;
```

Pour autoriser tous les clients distants à se connecter avec p/b à la BD cours :

```
CREATE USER p IDENTIFIED BY 'b';  
GRANT ALL ON cours.* TO 'p'@'%';  
FLUSH PRIVILEGES;
```

Dans le fichier my.ini ou my.cnf

Mettre en commentaire :

```
#bind-address = 192.168.2.96  
#skip-networking
```

Pour supprimer le user :

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM p;  
FLUSH PRIVILEGES;  
DROP USER p;
```

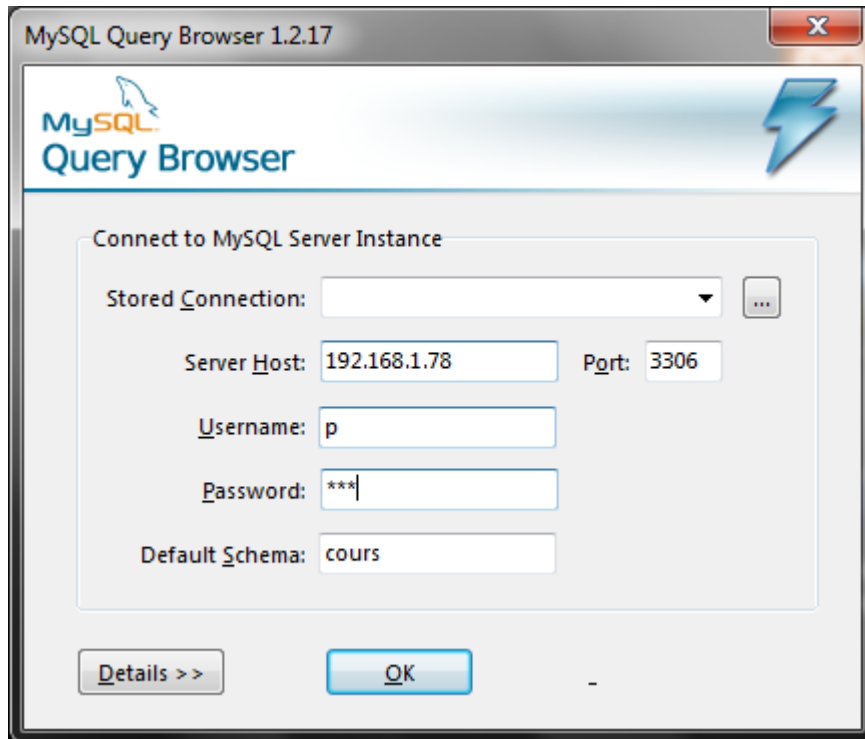
Modifier le mot de passe :

```
# Pour le user courant  
SET PASSWORD = PASSWORD('nouveau mot de passe');  
  
# Pour un autre user  
SET PASSWORD FOR nom_du_user = PASSWORD('nouveau mot de passe');
```

Lister les users

```
select * from mysql.user
```

La connexion avec MySQL Query Browser :



Note : même sur le poste où cela a été autorisé il n'est pas possible d'utiliser localhost ou 127.0.0.1.

Connexion avec PHPMYADMIN :

dans le fichier /dossierServeur/phpmyadmin/config.inc.php

```
/* Authentication type and info */
$config['Servers'][$i]['auth_type'] = 'config';
$config['Servers'][$i]['user'] = 'root';
$config['Servers'][$i]['password'] = '';
$config['Servers'][$i]['extension'] = 'mysql';
$config['Servers'][$i]['AllowNoPassword'] = true;
$config['Lang'] = '';

/* Bind to the localhost ipv4 address and tcp */
$config['Servers'][$i]['host'] = '127.0.0.1';
$config['Servers'][$i]['connect_type'] = 'tcp';
```