Système de Gestion de Bases de Données Relationnelles

MySQL

Youssef CHAHIR

PLAN

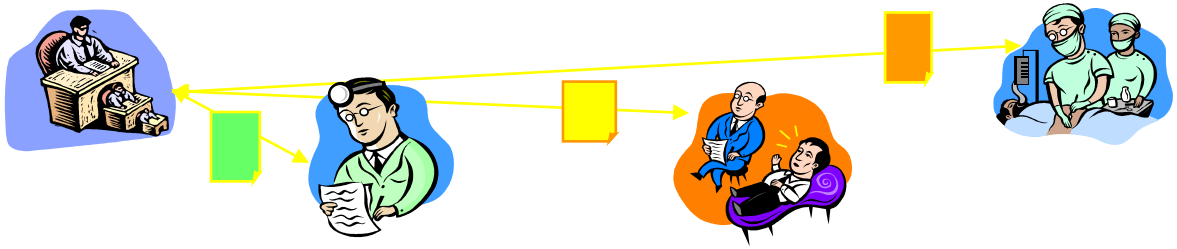
- **Introduction**
- Démarrer MySQL
- Syntaxe de MySQL
  - Types des attributs
  - Identificateurs
  - Effectuer des requêtes
- Fonctions de MySQL

Introduction

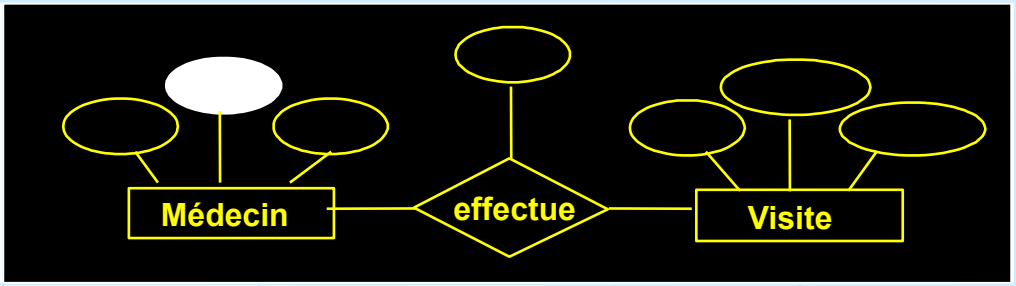
- **C'est un SGBD relationnel**
  - Gratuit (www.mysql.com)
  - Disponible sous Windows et tous les Unix
  - *Propose le minimum vital*
    - (Presque) tout le langage SQL
- **Très prisé pour les sites web**
  - Très bonne intégration avec Apache et PHP
  - Tout est gratuit
- **Beaucoup de "contributions" extérieures**
  - API en C++, Perl, Eiffel,PHP, Java (JDBC)
  - Des clients graphiques, des utilitaires
- **En résumé, un moteur SQL, simple et efficace**

BD: Modélisation du réel

Réel



Modèle conceptuel



Modèle logique

Relationnel **Objet** **XML**

Modèle Physique

- **Organisation physique des données**
- **Structures de stockage des données**
- **Structures accélératrices (index)**

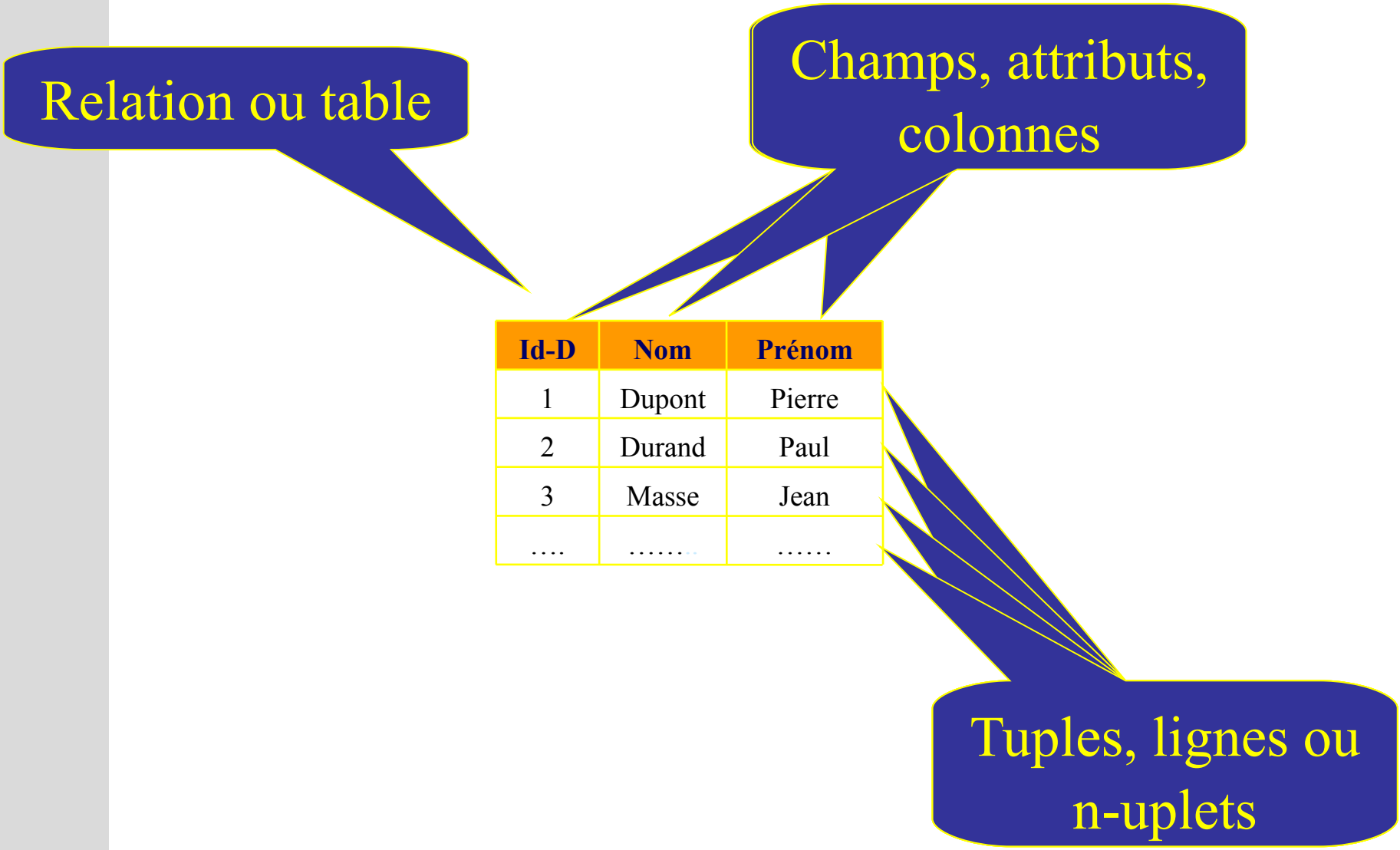
Les Bases de Données Relationnelles

- Définition :
  - Stockage organisé de données ayant des relations entre elles.
- Intérêt :
  - Eviter la redondance des informations
  - Extraire les données de façon pertinente (requête)
  - Faciliter les modifications
- Structure :
  - Constituée de tables (tableaux de données) reliées entre elles par des relations.

Qu’est- ce qu’une table ?

- ✔ Une table correspond à un thème et possède un nom unique dans la base de données.
- ✔ Une table est un tableau de données composée d'une ou plusieurs colonnes et d’une ou plusieurs lignes.
- ✔ Chaque colonne possède un nom unique à l'intérieur de la table.

Modélisation Relationnelle (1)



Modélisation Relationnelle (2)

Docteurs

Id-D	Nom	Prénom
1	Dupont	Pierre
2	Durand	Paul
3	Masse	Jean
....	.....	.....

Visites

Id-D	Id-P	Id-V	Date	Prix
1	2	1	15 juin	250
1	1	2	12 août	180
2	2	3	13 juillet	350
2	3	4	1 mars	250

Prescriptions

Id-V	Ligne	Id-M	Posologie
1	1	12	1 par jour
1	2	5	10 gouttes
2	1	8	2 par jour
2	2	12	1 par jour
2	3	3	2 gouttes
....	....	....	.....

Patients

Id-P	Nom	Prénom	Ville
1	Lebeau	Jacques	Paris
2	Troger	Zoe	Evry
3	Doe	John	Paris
4	Perry	Paule	Valenton
....	.....	.....	.....

Médicaments

Id-M	Nom	Description
1	Aspegic 1000	.....
2	Fluisédal	.....
3	Mucomyst	.....
....	.....	.....

Notion de Clé

- Clé primaire :
  - Identifiant correspondant à une ligne unique d'une table.
- *Clé externe (secondaire)* :
  - Rappel de la clé primaire d'une autre table.

Table des Propriétaires

Propriétaires			
ID	Nom	Code Postal	Ville
1	David	34090	Montpellier
2	Tintin	34090	Montpellier
3	Vladimir	34730	Prades le lez
4	Sophie	34170	Castelnau le lez
5	Farida	34770	Gigean

Table des Animaux

Animaux			
Nom	Espèces	DateNaissance	Propriétaire
Kiki	Chien	1990-10-25	*1*
Milou	Chien	1995-05-25	*2*
Titi	Oiseau	2000-01-28	*3*
Tom	Chat	2001-03-02	*4*
Jerry	Souris	1999-04-18	*5*
Rominet	Chat	2000-08-06	*3*

Conception d'une BD

- ✔ Faire l'inventaire de tous les types de données (colonnes)
- ✔ Regrouper ces types par thèmes (tables)
- ✔ Attribuer une clé primaire par table
- ✔ Définir les relations entre les tables par la mise en place de clés externes

Le langage SQL

- Structured **Q**uery **L**anguage
- **SQL permet de manipuler les informations stockées dans la base de données.**
- Une commande SQL est souvent appelée une **requête**
- SQL est composé de 3 langages:
  - Un langage de définition de données (DDL[1]): pour créer, modifier ou supprimer les définitions des tables : **create table**, **alter table** ...
  - Un langage de manipulation de données (DML[2]) : **select** , **insert** ...
  - Un langage de gestion des protections d'accès aux tables en environnement multi-utilisateurs (DCL[3]) : **grant**

1 Data Definition Language **2** Data Manipultaion Language **3** Data Control Language

En particulier, SQL permet de:

1. interroger (SELECT)
2. manipuler des entrées (UPDATE, INSERT, DELETE)
3. définir des données (CREATE, ALTER, DROP)
4. contrôler les accès (GRANT, REVOKE)

Les requêtes les plus couramment utilisées sont :

CREATE (création d'une table),

SELECT (sélection),

INSERT (insertion),

UPDATE (mise à jour des données),

DELETE (suppression),

ALTER (modification d'une table), etc

Démarrer un client de MySQL

- **Se connecter à MySQL**
  - `$ mysql options nombase`
    - `options:`
      - -h serveur: ordinateur qui héberge le serveur
      - -P port : port à utiliser pour se connecter (optionnel)
      - -u nomUtilisateur: votre nom d'utilisateur MySQL
      - -p : mysql vous demandera votre mot de passe
    - `nombase est optionnel:`
      - `Nom de la base à charger`
      - `Sinon, on utilise USE nombase;`

Client MySQL

- **Nom du serveur:** mysql.info.unicaen.fr
- **Port à utiliser:** 3333
- **Nom utilisateur :** votre nom de login
- **Mot de passe:** xxxx

```
$ mysql -h mysql.info.unicaen.fr -P 3333 -u chahir -p
Enter password: ****
```

ou

```
mysql -h mysql.info.unicaen.fr -P 3333 -u chahir --password="toto"
    chahir_bd
```

Commandes

- Toute commande doit se terminer par un ';' .
- Les résultats sont systématiquement présentés en tables.
- La dernière ligne d'état donne le temps d'exécution et le nombre de lignes du résultat (hors ligne de présentation).
  - Query OK, 0 rows affected (0.01 sec)
- Un script SQL peut être dirigé vers le client *mysql* à l'aide d'une commande de type :

  $ **mysql -h mysql.info.unicaen.fr -u chahir -p chahir_bd**

  < script.sql > resultat.txt

  Exemple: Mode batch

  $ mysql -h mysqlhost -p chahir_bd < script.sql
- Le script peut également être chargée directement depuis le client.

  mysql> \. script.sql

LOAD DATA

- Chargement à partir d'un fichier
  - mysql> LOAD DATA LOCAL INFILE "personnes.txt" INTO TABLE personne fields terminated by ',' enclosed by ' " ' ;

  La souplesse des commandes permet de définir précisément comment lire le fichier original à partir des « fields enclosed by, fields terminated by, ou lines terminated ».
- Les valeurs par défault sont :
  - Fields enclosed by : "
  - Fields escaped by : \\
  - Fields terminated by : \t
  - Lines terminated by : \n

Bases de données

- Activation d'une base de données bd
  - **mysql>USE bd;**
  - La modification reste active jusqu'à la fin de la session ou jusqu'au prochain appel de l'instruction USE nom_base.
- Liste des bases de données présentes sur le serveur
  - **mysql>SHOW DATABASES ;**

Informations sur les tables

- Liste des tables existantes :
  - `SHOW TABLES;`
- Structure d'une table :
  - `DESCRIBE nomtable;`
  - `DESC nomtable;`
    - **mysql>desc** demo1;

Structure d'une table

Exemple : Structure d'une table et des champs qui la définissent:

```
Database: demo  Table: demo1  Rows: 1
+----------+-----------+------+-----+---------+----------------+
| Field    | Type      | Null | Key | Default | Extra          |
+----------+-----------+------+-----+---------+----------------+
| id       | int(10)   |      | PRI | 0       | auto_increment |
| login    | char(10)  |      | MUL |         |                |
| password | char(100) | YES  |     |         |                |
| fullname | char(40)  |      |     |         |                |
| url      | char(60)  |      |     |         |                |
| food     | int(11)   |      |     | 0       |                |
| work     | int(11)   |      |     | 0       |                |
| love     | int(11)   |      |     | 0       |                |
| leisure  | int(11)   |      |     | 0       |                |
| sports   | int(11)   |      |     | 0       |                |
+----------+-----------+------+-----+---------+----------------+
```

- Ignorez les détails (Type, Null, Key, Extra) pour le moment !

Création de tables

```
CREATE [TEMPORARY] TABLE nom_table [IF NOT EXISTS]
  (
    nom_attribut1 TYPE_ATTRIBUT [OPTIONS]
    …
    nom_attributN TYPE_ATTRIBUT [OPTIONS]
    )
```

- L'instruction CREATE TABLE donne au minimum le nom de la table, le nom et le type des attributs.
- TEMPORARY: pour une durée de vie temporaire de la table.
- L'option IF NOT EXIST permet de ne créer cette table que si une table de même nom n'existe pas encore.
- A l'intérieur des parenthèses, il sera listé tous les attributs, clés et indexs de la table.
- Le programmeur conserve la possibilité de faire évoluer la structure des tables à l'aide de l'instruction ALTER TABLE.

Attributs pour une colonne

- ✔ AUTO_INCREMENT : un numéro unique est attribué lors de la création;
- ✔ NOT NULL : le champ doit être obligatoirement rempli ;
- NULL : les valeurs peuvent être omises ;
- PRIMARY KEY : la colonne est une clé primaire ;
- UNIQUE : la colonne est unique ;
- UNSIGNED : pour les entiers, indique que l'entier est non négatif ;
- DEFAULT "valeur" : valeur par défaut du champ.
- . . .

Types pour une colonne (I)

✔ Type alphanumérique

- CHAR(taille): chaîne de longueur fixe qui occupe le nombre d'octets indiqués par taille;
- VARCHAR(taille) chaîne de longueur variable, stocké avec le nombre d'octets nécessaire ;
- BLOB un grand objet binaire (Binary Large Object), permet de stocker des images, textes, (longueur maximale autorisée : 2^{16}-1 octets);
- TEXT chaîne texte, identique à un blob (recherche insensible à la casse);
- ENUM (val1,val2,...valN) la valeur de la colonne doit obligatoirement être une des valeurs de l'énumération ;
- SET (val1,val2,...valN) la valeur de la colonne est aucune, une ou plusieurs valeurs de l'ensemble.
- ...

Types pour une colonne (II)

✔ Types numériques

- INT[(taille)] 4 octets ;
- INTEGER[(taille)] synonyme de INT ;
- FLOAT[(taille,nb_decim)] nombre à virgule 4 octets;
- DOUBLE[(taille,nb_decim)] nombre à virgule 8 octets ;
- REAL[(taille,nb_decim)] synonyme de double;
- DECIMAL(taille,nb_decim) nombre stocké comme une chaîne (un caractère par chiffre) ;
- NUMERIC(taille,nb_decim) synonyme de décimal.
- ...

Types pour une colonne (III)

- **Types de dates et temps**

- DATE une date (Format 'YYYY-MM-DD') ;
- YEAR une année (Format 'YYYY') ;
- TIME une heure (Format 'HH:MM:SS') ;
- DATETIME une date et heure (Format 'YYYY-MM-DD HH:MM:SS')
- TIMESTAMP une durée (Format 'YYYYMMDDHHMMSS', 'YYMMDDHHMMSS', 'YYYYMMDD', ou 'YYMMDD') .

Création de table

Nom	Espèce	DateNaissance	Propriétaire

```
CREATE TABLE Animaux (
        Nom VARCHAR( 30) NOT NULL,
        Espèce VARCHAR( 30) NOT NULL,
        DateNaissance DATE NOT NULL,
        Propriétaire VARCHAR( 30) NOT NULL
) ;
CREATE TABLE article (
    id  INT ,
    titre VARCHAR(80),
    texte TEXT,
    parution DATE,
    auteur VARCHAR(80),
    rubrique ENUM('économie','sports','international','politique','culture')
);
```

Clé primaire (I)

- Clé primaire : Cet attribut ne doit jamais être vide et doit être unique ==> **PRIMARY KEY**
- **AUTO_INCREMENT ==>** signifie que cette valeur doit s'incrémenter automatiquement à chaque insertion d'un enregistrement. Elle débute à 1.

Notre exemple devient :

CREATE TABLE *Personne* (

id **INT UNSIGNED NOT NULL PRIMARY KEY AUTO_INCREMENT,**

nom **VARCHAR(40),**

prenom **VARCHAR(40),**

adresse **TINYTEXT,**

telephone **DECIMAL(10,0)**

)

Personne

Id	*nom*	*prénom*	*adresse*	*téléphone*
1	Dupond	Marc	8 rue de l'octet	0123456789

Clé primaire (II)

Une clé primaire peut être associée simultanément à plusieurs attributs.

Mauvaise syntaxe :

```
CREATE TABLE Personne (
        nom VARCHAR(40) PRIMARY KEY,
        prenom VARCHAR(40) PRIMARY KEY,
        adresse TINYTEXT,
        telephone DECIMAL(10,0)
)
```

Bonne syntaxe :

```
CREATE TABLE Personne (
        nom VARCHAR(40),
        prenom VARCHAR(40),
        adresse TINYTEXT,
        telephone DECIMAL(10,0),
        PRIMARY KEY (nom,prenom)
)
```

Attribut non nul

- Si l'on souhaite que certains attributs aient obligatoirement une valeur. On utilisera l'option **NOT NULL**.
  - Si aucune valeur par défaut n'est déclarée :

  - la chaîne vide **‘’** sera affectée à l'attribut s'il est de type chaîne de caractères
  - la valeur zéro **0** s'il est de type nombre
  - la date nulle 0000-00-00 et/ou l'heure nulle 00:00:00 s'il est de type date, heure ou date et heure.
- Au contraire, on utilisera l'option **NULL** si on autorise l'absence de valeur.

Valeur par défaut

- ✔ Pour donner une valeur par défaut à un attribut, on utilise l'option **DEFAULT**.
- ✔ Lors de l'ajout d'un enregistrement cette valeur sera affectée à l'attribut si aucune valeur n'est donnée.

Exemple :

'ville' ***VARCHAR*** **DEFAULT 'Caen'**

Les attributs de type chaîne de caractères de la famille TEXT et BLOB ne peuvent pas avoir de valeur par défaut.

Attribut sans doublon

- Pour interdire l'apparition de doublon pour un attribut, on utilise l'option **UNIQUE**.

Syntaxe :

UNIQUE [*nomdelacontrainte***](***liste des attributs***)**

Exemples:

- pour interdire tout doublon de l'attribut *nom* :
  - **UNIQUE(***nom***)**
- **Pour interdire tout doublon à un ensemble d'attributs, on passe en paramètre à UNIQUE la liste des attributs concernés:**
  - **UNIQUE(***nom,prenom***)**

CREATE [TEMPORARY] TABLE *nom_relation* [IF NOT EXISTS] (
nom_attribut TYPE_ATTRIBUT [OPTIONS]
…
)

TEMPORARY donne pour durée de vie à la table : le temps de la connexion de l'utilisateur au serveur, après, elle sera détruite. En l'absence de cette option, la table sera permanente à moins d'être détruite par la commande DROP TABLE.

L'option IF NOT EXIST permet de ne créer cette table que si une table de même nom n'existe pas encore.

A l'intérieur des parenthèses, il sera listé tous les attributs, clés et indexs de la table.

SELECT simple

Syntaxe:

SELECT *champs1, champs2,...* **FROM** *table*

OU

SELECT * FROM *table*

Exemple 1: Simple selections

SELECT id, login, fullname, love, sports **FROM** demo1

```
+----+-----------+-------------------+------+---------+
| id | login     | fullname          | love | sports |
+----+-----------+-------------------+------+--------+
|  1 | test      | Tester Test       |    3 |      3 |
| 34 | colin2    | Patrick Jermann2  |    1 |      4 |
....
```

Sélection conditionnelle
(SELECT . . . . WHERE)

Syntaxe: `SELECT .... FROM` *table*
`WHERE` *condition*

Quelques opérateurs pour les conditions:

Opérateur	Explication
opérateurs de comparaison simple	
=	égal
<> or !=	pas égal
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
opérateurs de combinaison	
AND	ET (les 2 propositions doivent être vraie)
OR	OU (une au moins des propositions est vraie)
opérateurs spéciaux	
expr IN (..., ...)	DANS (une liste)
expr NOT IN (..., ..., ...)	PAS dans ....
expr BETWEEN min AND max	Entre ... ET ...
expr NOT BETWEEN ...	(le contraire)

Note:

- Priorités: Opérateurs de comparaison, AND, OR, en cas de doute: mettre des parenthèses (...)
- Les strings doivent être mises entre '... ' ou "..." MAIS: Utilisez des quotes de préférence ('... '), pas des guillemets mal supportés par certaines bases de données.

Exemple

- Si nous désirons savoir pour qui le sport est très important (une valeur supérieure à 4) nous ajoutons une contrainte à la requête à l'aide d'un WHERE.

SELECT *id, login, fullname, love, sports* **FROM** *demo1* **WHERE** *sports>= 4*

id	login	fullname	love	sports
3	colin	Patrick Jermann	6	4
4	schneide	Daniel Schneider	6	6

Exemple : Select ... where

```
SELECT * from demo1 WHERE login = 'colin' AND food < 6
```

Exemple : Select ... where ... IN

• *Pour obtenir juste les noms complets de tous les logins égal à 'colin'* ***ou*** *'blurp'*

```
SELECT fullname from demo1 WHERE login in ('colin', 'blurp')
```

Exemple : Select ... where ... BETWEEN

```
SELECT * from demo1 WHERE food BETWEEN 3 AND 5
```

Fonctions de comparaison de chaînes

Le mot clé **LIKE** permet de comparer deux chaînes.

Le caractère '**%**' est spécial et signifie : 0 ou plusieurs caractères.

Le caractère '**_**' est spécial et signifie : 1 seul caractère, n'importe lequel.

L'exemple suivant permet de rechercher tous les clients sont le prénom commence par 'Jean', cela peut être 'Jean-Pierre', etc… :

```
SELECT nom
FROM clients
WHERE prénom LIKE 'Jean%'
```

Pour utiliser les caractères spéciaux ci-dessus en leur enlevant leur fonction spéciale, il faut les faire précéder de l'antislash : '****'.

Exemple, pour lister les produit dont le code commence par la chaîne '_XE' :

```
SELECT *
FROM produit
WHERE code LIKE '\_XE%'
```

Sélection(2)

- `SELECT * from nom_table ORDER BY nom_col;`
  - affiche toutes les rangées en les classant par ordre en fonction de nom_col

    mysql> select * from personne ORDER BY age DESC;

    ASC/DESC: ordre croissant/ décroissant

- ***SELECT CONCAT(prenom, ' ', nom) as membre FROM personne;***
- ***SELECT CONCAT(prenom, ' ', nom) as membre FROM personne ORDER BY prenom ASC, nom DESC;***

Remarque

SELECT base1.*table5*.*attribut2***
FROM base1.*table5*

Equivalent à

SELECT attribut2
FROM table5

Interrogation (selection)

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO OUTFILE 'file_name' export_options]
    [FROM table_references
        [WHERE where_definition]
        [GROUP BY col_name,...]
        [HAVING where_definition]
        [ORDER BY {unsigned_integer | col_name} [ASC | DESC] ,...]
        [LIMIT [offset,] rows]
        [PROCEDURE procedure_name] ]
```

Selection (2)

Nom	Description
SELECT	Spécifie les attributs dont on souhaite connaître les valeurs.
DISTINCT	Permet d'ignorer les doublons de ligne de résultat.
INTO OUTFILE	Spécifie le fichier sur lequel effectuer la sélection.
FROM	Spécifie le ou les relations sur lesquelles effectuer la sélection.
WHERE	Définie le ou les critères de sélection sur des attributs.
GROUP BY	Permet de grouper les lignes de résultats selon un ou des attributs.
HAVING	Définie un ou des critères de sélection sur des ensembles de valeurs d'attributs après groupement.
ORDER BY	Permet de définir l'ordre (**ASC**endant par défaut ou **DESC**endant) dans l'envoi des résultats.
LIMIT	Permet de limiter le nombre de lignes du résultats

Supprimer une table

La commande **DROP TABLE** prend en paramètre le nom de la table à supprimer. Toutes les données qu'elle contient sont supprimées et sa définition aussi.

Syntaxe :

DROP TABLE *relation*

Exemple :

DROP TABLE *Personnes*

Si on s'aperçoit qu'une relation a été mal définie au départ, plutôt que de la supprimer et de la reconstruire bien comme il faut, on peut la modifier très simplement. Cela évite de perdre les données qu'elle contient.

Insertion

- **Ajout d'informations (lignes) dans une table**
  - `INSERT INTO nom_table (nomcol1, nomcol2, ...) VALUES ('val1', 'val2', ...);`
  - `INSERT INTO nom_table SET nomcol1='val1', nomcol2='val2', ...;`

- `Exemple:`
  - INSERT INTO FilmSimple (titre, annee, nomMES, prenomMES) VALUES ('Pulp Fiction', 1996, 'Quentin', 'Tarantino');

Remarques

- insert into produits values (8,ecrou,5,12,vert);
- insert into produits (pno,design,prix) values (8,ecrou,5);
- insert into produits (pno,design,prix,poids,couleur) values (8,ecrou,5,NULL,NULL);
- insert into bonvins

  select * from vins as v where v.qualite= "A";

Modification et suppression

- **Modification d'informations dans une table**
  - `UPDATE nom_table SET col1='val1', col2='val2',... WHERE expression`
  - Changer le nom *de 'John Woo' en 'Yusen Wu'*.
    - UPDATE FilmSimple SET nomMES='Wu', prenomMES='Yusen' WHERE nomMES = 'Woo';
- **Suppression d'informations dans une table**
  - `DELETE FROM nom_table WHERE expression`
  - Les données détruites sont *vraiment* perdues.
  - Détruire tous les films antérieurs à 1960.
    - DELETE FROM FilmSimple WHERE annee <= 1960;

Modification d'une table (ALTER)

Voici ce qu'il est possible de réaliser avec: **`ALTER TABLE nom_table options.`**

- ajouter/supprimer un attribut
- créer/supprimer une clé primaire
- ajouter une contrainte d'unicité (interdire les doublons)
- changer la valeur par défaut d'un attribut
- changer totalement la définition d'un attribut
- changer le nom de la relation
- ajouter/supprimer un index

✔ `ALTER permet de changer le nom de la table, de modifier le types des colonnes, et les noms des colonnes, d'ajouter des colonnes, d'en supprimer.`

➔ `options: ADD, ALTER, CHANGE, MODIFY, DROP, RENAME.`

Changer le nom de la relation

Syntaxe :

ALTER TABLE *relation* RENAME *nouveau_nom*

Exemple :

ALTER TABLE *Personnes* RENAME *Carnet*

Cela consiste à renommer la table, et donc le fichier qui la stocke.

Ajouter un attribut

Syntaxe :

ALTER TABLE *relation* ADD *definition* **[FIRST | AFTER** *attribut]*

Ajoutons l'attribut *fax* qui est une chaîne représentant un nombre de 10 chiffres:

ALTER TABLE *Personnes* ADD *fax* **DECIMAL(10,0)**

Nous aurions pu forcer la place où doit apparaître cet attribut. Pour le mettre en tête de la liste des attributs de la relation, il faut ajouter l'option **FIRST** en fin de commande. Pour le mettre après l'attribut *'téléphone'*, il aurait fallu ajouter **AFTER** *'téléphone'*.

Note : il ne doit pas déjà avoir dans la relation un attribut du même nom !

Supprimer un attribut (I)

Attention, supprimer un attribut implique la suppression des valeurs qui se trouvent dans la colonne qui correspond à cet attribut, sauf à utiliser l'option IGNORE.

Syntaxe :

ALTER TABLE *relation* DROP *attribut*

Exemple :

ALTER TABLE *Personnes* DROP *'prénom'*

Supprimer un attribut (II)

La suppression d'un attribut peut incidemment provoquer des erreurs sur les contraintes clé primaire (**PRIMARY KEY**) et unique (**UNIQUE**).

CREATE TABLE ***Personne*** **(**

id **SMALLINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,**

nom **VARCHAR(40),**

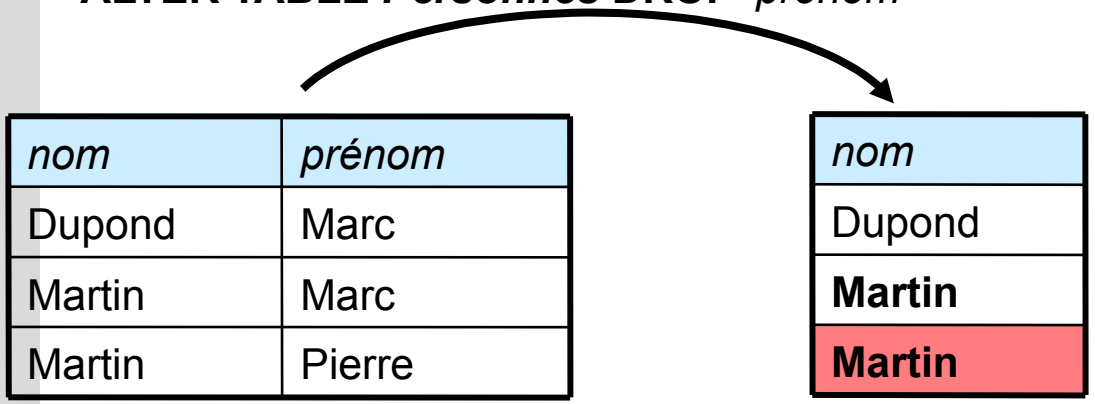
'*prénom*' **VARCHAR(40),**

adresse **TINYTEXT,**

'*téléphone*' **DECIMAL(10,0),**

UNIQUE(*nom***,***'prénom'***)**

)ALTER TABLE ***Personnes*** **DROP** *'prénom'*



nom	*prénom*
Dupond	Marc
Martin	Marc
Martin	Pierre

nom
Dupond
Martin
Martin

Refus d'opérer la suppression, car cela contredirait la contrainte d'unicité qui resterait sur l'attribut *nom*.

Créer une clé primaire

La création d'une clé primaire n'est possible qu'en l'absence de clé primaire dans la relation.

Syntaxe :

ALTER TABLE *relation* ADD PRIMARY KEY (*attribut***)**

Exemple :

ALTER TABLE *Personnes* ADD PRIMARY KEY (*nom,'prénom'*)

Supprimer une clé primaire

Comme une clé primaire est unique, il n'y a aucune ambiguïté lors de la suppression.

Syntaxe :

ALTER TABLE *relation* DROP PRIMARY KEY

Exemple :

ALTER TABLE *Personnes* ADD PRIMARY KEY

S'il n'y a aucune clé primaire lorsque cette commande est exécutée, aucun message d'erreur ne sera généré, le commande sera simplement ignorée.

Ajout d'une contrainte d'unicité

Il est possible (facultatif) de donner un nom à la contrainte.

Cette contrainte peut s'appliquer à plusieurs attributs.

Si les valeurs déjà présentes dans la table sont en contradiction avec cette nouvelle contrainte, alors cette dernière ne sera pas appliquée et une erreur sera générée.

Syntaxe :

ALTER TABLE *relation* ADD UNIQUE [*contrainte***] (***attributs***)**

Exemple pour interdire tout doublon sur l'attribut *fax* de la relation ***Personnes*** :

ALTER TABLE *Personnes* ADD UNIQUE *u_fax* **(***fax***)**

Autre exemple fictif :

ALTER TABLE *Moto* ADD UNIQUE *u_coul_vitre* **(***couleur,vitre***)**

Changer la valeur par défaut d'un attribut

Pour changer ou supprimer la valeur par défaut d'un attribut.

Attention aux types qui n'acceptent pas de valeur par défaut (les familles **BLOB** et **TEXT**).

Syntaxe :

ALTER TABLE *relation* ALTER *attribut* **{ SET DEFAULT** valeur **| DROP DEFAULT }**

Changer sa valeur par défaut :

ALTER TABLE *Personnes* ALTER *'téléphone'* **SET DEFAULT** '9999999999'

Supprimer sa valeur par défaut :

ALTER TABLE *Personnes* ALTER *'téléphone'* **DROP DEFAULT**

Le changement ou la suppression n'affecte en rien les enregistrements qui ont eu recours à cette valeur lors de leur insertion.

Changer la définition d'un attribut

Pour changer la définition de l'attribut sans le renommer :

ALTER TABLE *relation* MODIFY *attribut* **definition_relative**

Exemple 1 :

ALTER TABLE *Personnes* MODIFY *fax* **VARCHAR(14)**

Pour changer sa définition en le renommant :

ALTER TABLE *relation* CHANGE *attribut* **definition_absolue**

Exemple 2 :

ALTER TABLE *Personnes* CHANGE *fax num_fax* **VARCHAR(14)**

Attention, si le nouveau type appliqué à l'attribut est incompatible avec les valeurs des enregistrements déjà présents dans la relation, alors elles risques d'êtres modifiées ou remises à zéro !

Expressions régulières(1)

- **Correspondances et caractères**
  - **LIKE:**
    - % toute chaîne to% = toto, toti...
    - _ tout caractère tou_ = tour, tout...
  - REGEXP
    - ^ correspondance au début: ”chahir" REGEXP "^ch"= vrai
    - $ correspondance à la fin: ”chahir" REGEXP ”ir$" = vrai
    - http://dev.mysql.com/doc/refman/5.0/fr/regexp.html
- Exemples:
  - select * from personne where nom like”%rv%”;
  - select * from personne where nom like”___mple”;

Expressions régulières(2)

- [...] tout caractère dans les crochets

  "tous" REGEXP "tou[rs]" = vrai

  "tout" REGEXP "tou[rs] = faux

- [^...] tout caractère s'il n'appartient pas à ceux des crochets

  "tout" REGEXP "tou[^rs] = vrai

- [a-z] tout caractère dans l'intervalle

select * from personne where nom REGEXP ”^[cC]”;

Expressions régulières(3)

- Indicateur d'occurrences et choix

*	0 ou plusieurs
+	au moins une fois
?	0 ou 1 fois
{n}	n fois
{n,}	n fois ou plus
{n,m}	au moins n fois et au plus m
x | y	x ou y

regexp "^[b|c]...*[bne]"

Fonctions mathématiques

Fonction	Description
ABS(x)	Valeur absolue de X.
SIGN(x)	Signe de X, retourne -1, 0 ou 1.
FLOOR(x)	Arrondi à l'entier inférieur.
CEILING(x)	Arrondi à l'entier supérieur.
ROUND(x)	Arrondi à l'entier le plus proche.
EXP(x), LOG(x), SIN(x), COS(x), TAN(x), PI()	Bon, là c'est les fonctions de maths de base…
POW(x,y)	Retourne X à la puissance Y.
RAND(), RAND(x)	Retourne un nombre aléatoire entre 0 et 1.0 Si x est spécifié, entre 0 et X
TRUNCATE(x,y)	Tronque le nombre X à la Yème décimale.

```
SELECT nom
FROM filiales
WHERE SIGN(ca) = -1
ORDER BY RAND()
```

Cet exemple affiche dans un ordre aléatoire le nom des filiales dont le chiffre d'affaire est négatif.
A noter que : SIGN(ca) = -1 ⇔ ca < 0

Fonctions de chaînes

Fonction	Description
TRIM(x)	Supprime les espaces de début et de fin de chaîne.
LOWER(x)	Converti en minuscules.
UPPER(x)	Converti en majuscules.
LONGUEUR(x)	Retourne la taille de la chaîne.
LOCATE(x,y)	Retourne la position de la dernière occurrence de x dans y. Retourne 0 si x n'est pas trouvé dans y.
CONCAT(x,y,…)	Concatène ses arguments.
SUBSTRING(s,i,n)	Retourne les n derniers caractères de s en commençant à partir de la position i.
SOUNDEX(x)	Retourne une représentation phonétique de x.

```
SELECT UPPER(nom)
FROM clients
WHERE SOUNDEX(nom) = SOUNDEX('Dupond')
```

On affiche en majuscules le nom de tous les clients dont le nom ressemble à 'Dupond'.

Fonctions de dates et heures

Fonction	Description
NOW()	Retourne la date et heure du jour.
TO_DAYS(x)	Conversion de la date X en nombre de jours depuis le 1er janvier 1970.
DAYOFWEEK(x)	Retourne le jour de la semaine de la date x sous la forme d'un index qui commence à 1 (1=dimanche, 2=lundi…)
DAYOFMONTH(x)	Retourne le jour du mois (entre 1 et 31).
DAYOFYEAR(x)	Retourne le jour de l'année (entre 1 et 366).
SECOND(x), MINUTE(x), HOUR(x), MONTH(x), YEAR(x), WEEK(x)	Retournent respectivement les secondes, minutes, heures, mois, année et semaine de la date.

```
SELECT titre
FROM article
WHERE (TO_DAYS(NOW()) – TO_DAYS(parution)) < 30
```

Cet exemple affiche le titre des articles parus il y a moins de 30 jours.

Index

Lors de la recherche d'informations dans une relation, MySQL parcours la table correspondante dans n'importe quel ordre. Dans le cas d'un grand nombre de lignes, cette recherche est très très longue du fait du parcours de TOUTE la table.

Pour y remédier, une optimisation possible et FORTEMENT recommandée, est d'utiliser des indexs.

La création d'un index associé à un attribut ou à un ensemble ordonné d'attributs va créer une liste ordonnée des valeurs de ces attributs et de l'adresse de la ligne associée. C'est sur les valeurs de cette liste que se fera les recherches et les tris. Les algorithmes de recherche et de tri sur des ensembles ordonnées sont énormément plus rapides !

On gagne donc énormément en temps d'accès aux informations. Bien que cela ralentisse les mises à jour (insertion, suppression, modification de clé).

On choisira de créer des indexs sur les attributs qui seront les plus sollicités par les recherches ou utilisés comme critère de jointure.

Syntaxe : **INDEX *index* (*liste des attributs*)**

Exemple: Créer un index sur le couple (*nom*,*prénom*) :

INDEX *idx_nom_prenom* (*nom,'prénom'*)

Ajouter un index

Une table ne peut comporter que 32 indexs.

Et un index ne peut porter que sur 16 attributs maximum à la fois.

Syntaxe :

ALTER TABLE *relation* ADD INDEX *index* (*attributs*)

Exemple :

ALTER TABLE *Personnes* ADD INDEX *nom_complet* (*nom*,*prénom*)

Dans cet exemple, on a ajouté à la relation ***Personnes*** un index que l'on nomme *nom_complet* et qui s'applique aux deux attributs *nom* et '*prénom*'. Ainsi, les recherches et les tris sur les attributs *nom* et '*prénom*' seront grandement améliorés. Car un index apporte les changements sous-jacents permettant d'optimiser les performances du serveur de base de données.

Supprimer un index

Syntaxe :

ALTER TABLE *relation* DROP INDEX *index*

Exemple :

ALTER TABLE *Personnes* DROP INDEX *nom_complet*

Cette exemple permet de supprimer l'index nommé *nom_complet* de la relation ***Personnes***.