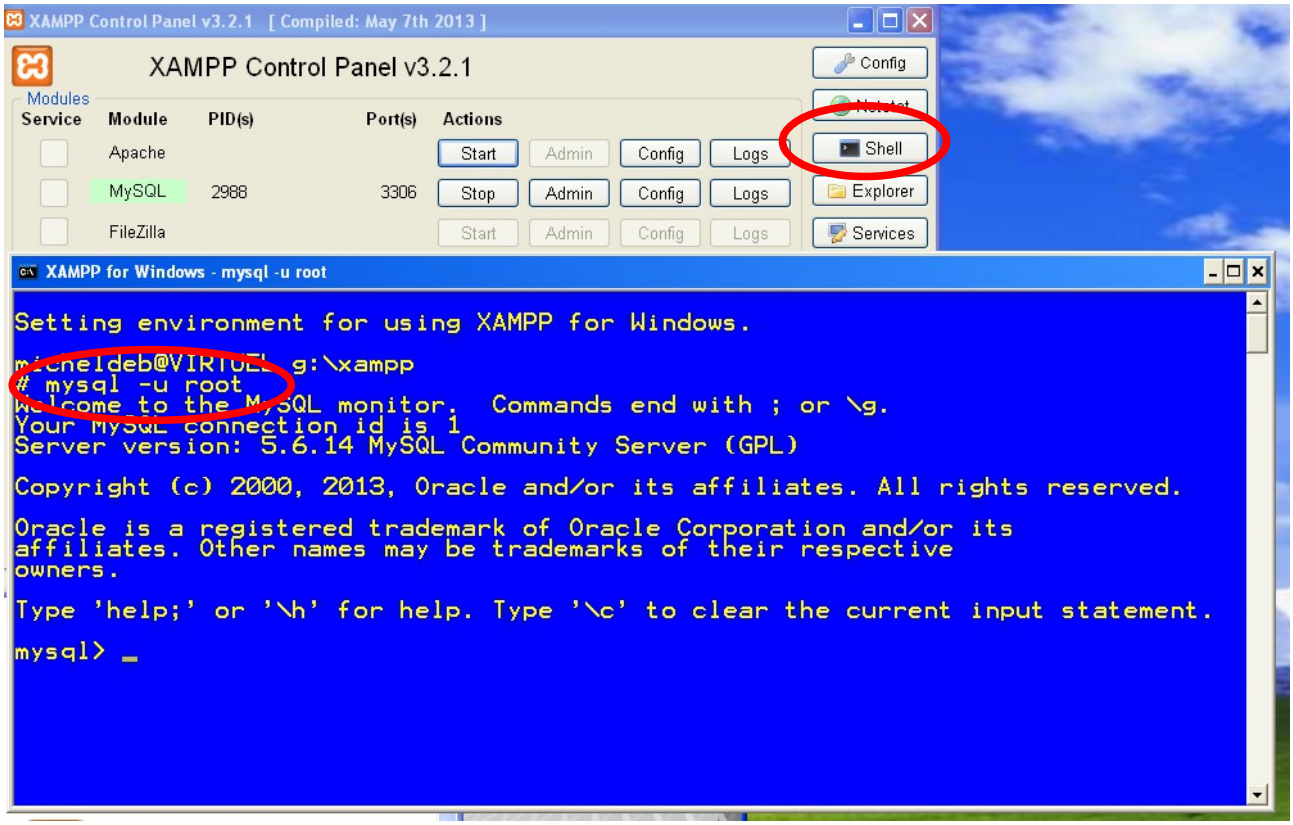
Utilisation de la console mysql avec xampp

Ouvrir une console



shell, ~~se placer dans le répertoire mysql\bin\~~ et taper

mysql -u root

Vous êtes connecté

La commande

`mysql -u root`

exécute un utilitaire qui permet de dialoguer avec le serveur
MySql. Ce mode de gestion s'effectue en ligne de commande et est appelé le mode console.
Lancer cet utilitaire mysql. A ce stade que l'on soit root ou utilisateur quelconque, aucun mot de passe n'est demandé. Et l'on se trouve devant le prompt

`mysql>`

ce qui signifie que l'interpréteur de commandes SQL attend nos requêtes.

```
$ mysql
Welcome to the MySQL monitor. Commands ends with ; or \g
Your Mysql connection id is ...
tape help; ou \h for help
mysql>
```

Quels sont les premiers utilisateurs ?

2 utilisateurs ont déjà été créés par l'installation
Un super-utilisateur du serveur MySql, appelé root@localhost (homonyme du superviseur Linux, mais qui en est bien distinct).
Celui-ci possède **tous les droits** sur les bases de données et en particulier sur la base d'administration nommée elle aussi **mysql**
Or, à l'installation root peut se connecter sans mot de passe ! Il faudrait lui en imposer un !
L'utilisateur anonyme, noté @localhost (il n'a réellement pas de nom)qui ne possède que des droits d'accès très réduits,
Remarque : Les commandes SQL et en particulier celles que l'on passe dans l'utilitaire mysql, ne sont pas sensibles à la casse. Mais par convention, il est d'usage d'écrire les mots-clés SQL en majuscules.

Ouvrir en tant qu'administrateur:

En tant qu'**administrateur (root)** de mysql, cela donne
`mysql -u root -p`
-p car l'administrateur devrait avoir un mot de passe au moins
Remarque : Les commandes SQL et en particulier celles que l'on passe dans l'utilitaire mysql, ne sont pas sensibles à la casse. Mais par convention, il est d'usage d'écrire les mots-clés SQL en majuscules.

Voir les bases

mysql> show databases;

```
+----------+
| Database |
+----------+
| mysql    |
+----------+
1 row in set (0.02 sec)
```

Comment créer une nouvelle base ?

root crée une nouvelle base de données nommée essais et vérifie sa présence dans la liste des bases

```
mysql> create database essai;
Query OK, 1 row affected (0.05 sec)
```

Sélectionner une base de données

La création d'une base de données ne la sélectionne pas pour l'utilisation; vous devez le faire explicitement. Pour rendre `essai` la base courante, utilisez cette commande:

```
mysql> USE essai;
Database changed
```

Créer la base de données est la partie facile, mais jusque-là elle est vide, comme vous le montre

SHOW TABLES;

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

Comment supprimer une base ?

En mode console, root passe la commande sans recours (attention !)

```
mysql> DROP DATABASE  essai;
Query OK,  0 row affected (0.00 sec)
```

Une curieuse requete : "select"

Voilà une commande simple qui demande au serveur de vous donner son numéro de version et la date courante. Entrez-la comme suit, juste après l'invite `mysql>` puis pressez Enter :

```
mysql> SELECT VERSION(), CURRENT_DATE;
```

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------------+--------------+
| VERSION()       | CURRENT_DATE |
+-----------------+--------------+
| 5.1.37-1ubuntu5 | 2010-02-17   |
+-----------------+--------------+
1 row in set (0,00 sec)
```

La requête révèle plusieurs choses à propos de `mysql` :

- Une commande consiste normalement en une commande SQL suivie d'un point-virgule. (Il y a quelques cas ou le point-virgule n'est pas requis. `QUIT`, mentionnée plus tôt, en fait partie. Nous verrons les autres plus tard.)
- Lorsque vous entrez une commande, `mysql` l'envoi au serveur pour l'exécution et affiche le résultat, puis affiche un autre `mysql>` pour indiquer qu'il attend une autre commande.
- `mysql` affiche le résultat des requêtes dans une table (lignes et colonnes). La première ligne contient le nom des colonnes. Les lignes suivantes constituent le résultat de la requête. Normalement, les titres des colonnes sont les noms des champs des tables de la base de données que vous avez récupérés. Si vous récupérez la valeur d'une expression au lieu d'une colonne (comme dans l'exemple précédent), `mysql` nomme la colonne en utilisant l'expression elle-même.
- `mysql` vous indique combien de lignes ont été retournées et combien de temps d'exécution la requête a pris, ce qui vous donnera une approximation des performances du serveur. Ces valeurs sont imprécises car elles représentent le temps logiciel (et non le temps processeur ou matériel), et qu'elles sont affectées par des facteurs tels que la charge du serveur ou l'accessibilité du réseau. (Dans un soucis de brièveté, la ligne contenant ``rows in set" n'est plus montrée dans les exemples suivants de ce chapitre.)

Les mots-clef peuvent être entrés sous n'importe quelle forme de casse. Les requêtes suivantes sont équivalentes :

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Voilà une autre requête. Elle montre que vous pouvez utiliser `mysql` en tant que simple calculatrice :

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+-------------+---------+
|    0.707107 |      25 |
+-------------+---------+
```

Les commandes vues jusqu'à présent ont été relativement courtes, et tenaient sur une seule ligne. Vous pouvez même entrer plusieurs requêtes sur une seule ligne. Il suffit de terminer chacune d'elle par un point-virgule :

```
mysql> SELECT VERSION(); SELECT NOW();
```

```
+-----------------+
| VERSION()       |
+-----------------+
| 5.1.37-1ubuntu5 |
+-----------------+
1 row in set (0,00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2010-02-17 12:20:23 |
+---------------------+
1 row in set (0,00 sec)
```

Une commande ne doit pas être obligatoirement sur une seule ligne ; les commandes qui exigent plusieurs lignes ne sont pas un problème. `mysql` détermine où se situe la fin de votre commande en recherchant le point-virgule de terminaison, et pas l'extrémité de la commande entrée. (Dans d'autres termes, `mysql` accepte des formats libres d'entrée : il collecte les lignes entrées mais ne les exécute qu'une fois le point-virgule trouvé.)

Voilà une seule requête sur plusieurs lignes :

```
mysql> SELECT
    -> user(),
    -> CURRENT_DATE
    -> ;
+----------------+--------------+
| user()         | CURRENT_DATE |
+----------------+--------------+
| root@localhost | 2010-02-17   |
+----------------+--------------+
1 row in set (0,00 sec)
```

Si vous décidez d'annuler une commande que vous êtes en train de taper, faites-le en entrant `\c` :

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Ici aussi, portez votre attention sur l'invite. Elle se transforme à nouveau en `mysql>` après que vous ayez entré `\c`, vous informant que `mysql` est prêt pour une nouvelle requête.

Le tableau suivant montre les différentes invites que vous pourrez voir et résume leur signification quand à l'état dans lequel se trouve `mysql` :

Invite	**Signification**
`mysql>`	Prêt pour une nouvelle commande.
`->`	En attente de la ou des lignes terminant la commande.
`'>`	En attente de la prochaine ligne collectant une chaîne commencée par un guillemet simple (‘'’).
`">`	En attente de la prochaine ligne collectant une chaîne commencée par un guillemet double (‘"’).
`` `> ``	En attente de la prochaine ligne collectant une chaîne commencée par un guillemet oblique (‘`’).

Les commandes sur plusieurs lignes sont la plupart du temps des accidents, lorsque vous voulez faire une commande sur une seule ligne et que vous oubliez le point-virgule de fin. Dans ce cas, `mysql` attend la suite de votre saisie :

```
mysql> SELECT USER()
    ->
```

Si cela vous arrive (vous pensez que votre requête est complète mais la seule réponse est l'invite `->`), il est fort probable que `mysql` attende le point-virgule. Si vous ne notez pas ce que l'invite vous indique, vous pourriez patienter pendant longtemps avant de réaliser ce que vous devez faire. Entrez un point-virgule pour compléter la requête, et `mysql` devrait l'exécuter.

Liste des bases

Utilisez la commande `SHOW` pour trouver quelles bases existent déjà sur le serveur :

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

La liste des bases de données est probablement différente sur votre machine, mais les bases `mysql` et `test` y figurent sûrement. La base `mysql` est requise car elle gère les accès et les privilèges. La base `test` est souvent fournie pour que les utilisateurs y effectuent leurs tests.

Notez que vous ne pourrez voir toutes les bases de données si vous n'avez pas le privilège `SHOW DATABASES`.

Utiliser une base

```
mysql> use mysql;
Database changed
```

Voir les tables

```
mysql> show tables;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
6 rows in set (0.02 sec)
```

Afficher les champs user et password de la table user:

```
mysql> select user,password  from user;
+------+----------+
| user | password |
+------+----------+
| root |          |
|      |          |
+------+----------+
2 rows in set (0.02 sec)
```

idem avec le nom du host en plus

```
mysql> select user,password,host  from user;
+------+----------+-----------+
| user | password | host      |
+------+----------+-----------+
| root |          | localhost |
|      |          | localhost |
+------+----------+-----------+
2 rows in set (0.00 sec)
```

On voit içi clairement l'utilisateur anonyme de mysql.

Changer l'ordre et trier

```
mysql> select host, user,password  from user order by user;
+-----------+------+----------+
| host      | user | password |
+-----------+------+----------+
| localhost |      |          |
| localhost | root |          |
+-----------+------+----------+
```

Tri descendant

```
mysql> select host, user,password  from user order by user desc;
+-----------+------+----------+
| host      | user | password |
+-----------+------+----------+
| localhost | root |          |
| localhost |      |          |
+-----------+------+----------+
2 rows in set (0.02 sec)
```

Creer une base vide (essai)

```
mysql> create database essai;
Query OK, 1 row affected (0.05 sec)
```

Utiliser la nouvelle base et vérifier qu'elle est vide

```
mysql> use essai
Database changed
mysql> show tables;
Empty set (0.00 sec)
```

Creer une table et vérifier son existence - create table

```
mysql> create table carnet(numero int,nom varchar(60),email varchar(80));
Query OK, 0 rows affected (0.08 sec)

mysql> show tables;
+-----------------+
| Tables_in_essai |
+-----------------+
| carnet          |
+-----------------+
```

Vérifier la structure de la table

```
mysql> describe carnet;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| numero | int(11)     | YES  |     | NULL    |       |
| nom    | varchar(60) | YES  |     | NULL    |       |
| email  | varchar(80) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
3 rows in set (0.02 sec)
```

Fichiers créés

On constate que 3 fichiers portant le nom de la table sont créés dans le répertoire de la base */var/lib/mysql/essais*
Il en sera de même pour nouvelle table incluse dans cette base.

carnet.frm	décrit la structure de la table
carnet.MYD	contient les données (vide à la création)
carnet.MYI	contient la descripton des index

`tbl_name.frm`	Définition de la table
`tbl_name.MYD`	Fichier de données
`tbl_name.MYI`	Fichier d'index

Noms des bases de données, tables, index, colonnes et alias

Les noms des bases de données, tables, index, colonnes et alias suivent tous les mêmes règles en MySQL.

La table suivante décrit la taille maximale et les caractères autorisées pour chaque type d'identifiant.

Identifiant	**Longueur maximale**	**Caractères autorisés**
Base de données	64	Tous les caractères autorisés dans un nom de dossier à part ‘/’, ‘\’ et ‘.’.
Table	64	Tous les caractères autorisés dans le nom d'un fichier à part ‘/’ et ‘.’.
Colonne	64	Tous.
Index	64	Tous.
Alias	255	tous.

Notez qu'en plus de ce qui précède, vous n'avez pas droit au caractères ASCII(0) ou ASCII(255) dans un identifiant.

Même si les identifiants Unicode peuvent inclure des caractères multi-octets, notez que les tailles maximales affichées dans la table sont donnés en octets. Si un identifiant contient un caractère multi-octet, le nombre de *caractères* autorisé est alors inférieur aux chiffres affichés.

Un identifiant peut être entre guillemet ou pas. Si un identifiant est un mot réservé, ou qu'il contient des caractères spéciaux, vous *devez* le mettre entre guillemets lorsque vous l'utilisez.

Notez que si un identifiant est un mot réservé, ou contient des caractères spéciaux, vous devez absolument le protéger avec ‘`’ :

Le caractère de protection des identifiants est le guillemet oblique ‘`’ :

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

Faut être un peu tordu pour appeler une table select !!!

Depuis MySQL 4.1, les guillemets peuvent être inclus dans les noms d'identifiants. Si le caractère inclus dans l'identifiant est le même que celui qui est utilisé pour protéger l'identifiant, doublez-le. La commande suivante crée la table a`b, qui contient la colonne c"d :

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

Le jeu de caractères utilisé pour les données et le stockage

Par défaut, MySQL utilise le jeu de caractères ISO-8859-1 (Latin1) avec tri en accord au Suédois/Finnois. C'est le jeu de caractère le mieux adapté pour les USA et l'Europe de l'ouest.

Tous les binaires standards MySQL sont compilés avec `--with-extra-charsets=complex`. Cela ajoutera du code à tous les programmes standards pour qu'ils puissent gérer `latin1` et tous

les jeux de caractères multi-octets compris dans le binaire. Les autres jeux de caractères seront chargés à partir d'un fichier de définition de jeu si besoin.

Le jeu de caractères détermine quels caractères sont autorisés dans les noms et comment s'effectuent les tris dans les clauses `ORDER BY` et `GROUP BY` de la commande `SELECT`.

Cela peut être modifié avec l'option de démarrage `--default-character-set` de `mysqld`.

schéma

ensemble de tables, contraintes et index composant une base de données

Exécuter un script SQL

Si vous avez déjà démarré le client *mysql* , vous pouvez *exécuter un script* SQL en utilisant la commande source : *mysql>* source nom_fichier

Types de champs

Les chaînes de caractères :

CHAR (n)	Chaîne de n caractère, taille fixe	
VARCHAR(M)	Chaîne de caractères variable. M peut être compris entre 1 et 255. longueur de la chaine:nombre de caractères+1	255 caractères. maximum
TINYBLOB, TINYTEXT	Petite zone de texte. Objet d'une longueur maximale de 255 caractères, TINYTEXT aura un contenu de type ASCII (casse insensible) et TINYBLOB aura un contenu de type binaire (casse sensible).	255 caractères. maximum
BLOB, TEXT	Zone de texte standard Objet d'une longueur maximale de 65535 caractères, TEXT aura un contenu de type ASCII (casse insensible) et BLOB aura un contenu de type binaire (casse sensible).	65 535 caractères. maximum
MEDIUMBLOB, MEDIUMTEXT	Zone de texte moyenne. Objet d'une longueur maximale de 16777216 caractères, MEDIUMTEXT aura un contenu de type ASCII (casse insensible) et MEDIUMBLOB aura un contenu de type binaire (casse sensible).	16 millions caractères. maximum
LONGBLOB, LONGTEXT	Grande zone de texte. Objet d'une longueur maximale de 4294967295 caractères, LONGTEXT aura un contenu de type ASCII (casse insensible) et LONGBLOB aura un contenu de type binaire (casse sensible).	4 milliards caractères. maximum
ENUM('valeur','valeur2',...)	Une valeur parmi plusieurs Objet texte qui ne peut avoir qu'une des valeurs 'valeur','valeur2',...	65535 valeurs max.
SET('valeur','valeur2',...)	Une ou plusieurs valeurs parmi plusieurs. Objet texte qui peut avoir une ou plusieurs des valeurs 'valeur','valeur2',...	64 valeurs max.

Les champs numériques :

TINYINT	Entier très petit	1 octet
SMALLINT	Entier petit compris entre -32768 et 32767, si l'option UNSIGNED est utilisée, ce nombre sera compris entre 0 et 65535.	2 octets
MEDIUMINT	Entier moyen compris entre -8388608 et 8388607, si l'option UNSIGNED est utilisée, ce nombre sera compris entre 0 et 16777215	3 octets
INT	Entier standard compris entre –2 147 483 648 et 2 147 483 647. Si l'option UNSIGNED est utilisée, ce nombre sera compris entre 0 et 4 294 967 295	4 octets
BIGINT	Entier grand	8 octets
FLOAT	Décimal de simple précision	4 octets
DOUBLE, REAL	Décimal de double précision	8 octets
DECIMAL (entier,décimal)	Réel, définissez la longueur de chacune des deux parties.	variable

Les champs de Types date et heure :

DATE	Date (ex: 2000-08-24)	3 octets
TIME	Heure (ex: 23:44:05)	3 octets
DATETIME	Date et heure (ex: 2000-08-24 23:44:05)	8 octets
YEAR	Année (ex: 2000)	1 octet

Le type `DATE` est utilisé pour manipuler simplement une date, sans l'heure. *MySQL* retourne et affiche les valeurs de type `DATE` au format `'YYYY-MM-DD'`

Le type `DATETIME` est utile pour manipuler en même temps une date et une heure. . **MySQL** retourne et affiche les valeurs de type `DATETIME` au format `'YYYY-MM-DD HH:MM:SS'.`

Le type `TIMESTAMP` est utilisé automatiquement lors de requête , avec la valeur courante de date et d'heure. Est utilisée pour calculer des différences entre deux dates.

Chacun de ces différents types a un argument optionnel permettant de changer légèrement le

formatage.

NB : MySql représente **les dates** ainsi : **l'année, suivie du mois, puis du jour**. Le *24 août 2000* est donc représenté sous la forme *"2000-08-24* ".

Valeurs `NULL`

La valeur `NULL` signifie ``pas de données'' et est différente des valeurs comme `0` pour les nombres ou la chaîne vide pour les types chaîne.

Ajouter des enregistrements :

```
INSERT INTO nom_table(colonne1, colonne2, colonne3,…)
VALUES ('xx', 'yy', 'zz');
```

La commande INSERT INTO est utilisée pour ajouter des enregistrements dans une base de données. Celle-ci s'emploie avec VALUES pour inclure les données.

```
INSERT INTO nom_table (champ1,champ2,champn) VALUES (val1,val2,valn)
mysql> insert into
carnet(numero,nom,email)values(1,'Plomteux','michel@plomteux.net');
Query OK, 1 row affected (0,00 sec)
```

On vérifie en sélectionnant tous les champs:

```
select * from carnet;
+--------+----------+---------------------+
| numero | nom      | email               |
+--------+----------+---------------------+
|      1 | Plomteux | michel@plomteux.net |
+--------+----------+---------------------+
1 row in set (0,00 sec)
```

Remarque importante : les valeurs chaîne sont entre apostrophes ' (simple quote)

Si les champs sont tous présents et dans l'ordre, il n'est pas nécessaire de préciser la liste avant l'attribut values()

```
mysql> insert into carnet values(2,'Nemard','nemard@savon.net');

mysql> select * from carnet;
+--------+----------+---------------------+
| numero | nom      | email               |
+--------+----------+---------------------+
|      1 | Plomteux | michel@plomteux.net |
|      2 | Nemard   | nemard@savon.net    |
+--------+----------+---------------------+
```

Ajouter un champ

```
ALTER TABLE Nom_table ADD nom_colonne type_données;
```

On a oublié le prenom:

```
ALTER TABLE carnet ADD prenom  VARCHAR( 30 ) ;
```

Avec la même méthode, on peut enlever un champ (drop)
mysql> ALTER TABLE ...DROP ...;
Il faut noter que les champs de type **CHAR** et **VARCHAR** ne sont pas sensibles à la casse. Autrement dit, lors d'une recherche, *"texte"* sera identique à *"TexTe"* . En effet, pour rendre ces types de colonnes sensibles à la différence entre majuscule et minuscule, il faut ajouter l'argument

BINARY dans la définition du champ (ex : **VARCHAR** *(30)* **BINARY**).

```
mysql> ALTER TABLE carnet ADD prenom  VARCHAR( 30 ) ;
Query OK, 2 rows affected (0,07 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> select * from carnet;
+--------+----------+---------------------+--------+
| numero | nom      | email               | prenom |
+--------+----------+---------------------+--------+
|      1 | Plomteux | michel@plomteux.net | NULL   |
|      2 | Nemard   | nemard@savon.net    | NULL   |
+--------+----------+---------------------+--------+
2 rows in set (0,00 sec)
```

Supprimer une colonne au sein d'une table

```
ALTER TABLE Nom_table DROP COLUMN nom_colonne ;
```

Et ajouter mon prénom ?

Mettre un champ à jour

UPDATE "nom de table"
SET colonne 1 = [valeur 1], colonne 2 = [valeur 2]
WHERE {condition}

```
update carnet set prenom='michel' where numero=1;
Query OK, 1 row affected (0,00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from carnet;
+--------+----------+---------------------+--------+
| numero | nom      | email               | prenom |
+--------+----------+---------------------+--------+
|      1 | Plomteux | michel@plomteux.net | michel |
|      2 | Nemard   | nemard@savon.net    | NULL   |
+--------+----------+---------------------+--------+
2 rows in set (0,00 sec)
```

Une petite variante qui met en évidence l'insensibilité à la casse (majuscule)?

```
 update carnet set prenom='jean' where nom='nemard';
Query OK, 1 row affected (0,00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from carnet;
+--------+----------+---------------------+--------+
| numero | nom      | email               | prenom |
+--------+----------+---------------------+--------+
|      1 | Plomteux | michel@plomteux.net | michel |
|      2 | Nemard   | nemard@savon.net    | jean   |
+--------+----------+---------------------+--------+
```

Définir des clés primaires

```
PRIMARY KEY (colonne1, colonne2,…) ;
```

Si on y pense dès le départ, cela donne quelque chose comme :

```
CREATE TABLE Customer
(SID integer,
Last_Name varchar(30),
First_Name varchar(30),
PRIMARY KEY (SID));
```

Sinon, il reste à l'ajouter:

ALTER TABLE Customer ADD PRIMARY KEY (SID);

Dans notre table carnet, cela donne :

```
mysql> alter table carnet add primary key (numero);
Query OK, 9 rows affected (0,06 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> describe carnet;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| numero | int(11)     | NO   | PRI | 0       |       |
| nom    | varchar(60) | YES  |     | NULL    |       |
| email  | varchar(80) | YES  |     | NULL    |       |
| prenom | varchar(30) | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0,00 sec)
```

Créer un index

Les index permettent d'accéder à certaines informations d'une base de donnée de manière plus efficace.

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 !

Ainsi, d'une recherche à coût prohibitif, on passe à une recherche sur un ensemble déjà trié. 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. Par contre, on épargnera les attributs qui contiennent peu de valeurs différentes les unes des autres et ceux dont les valeurs sont très fréquemment modifiées.

Syntaxe

```
CREATE INDEX nom_index
ON nom_table (colonne [ASC/DESC], colonne2,…);
```

```
create index nom_prenom on carnet(nom,prenom);
```

Ajouter un grand nombre d'enregistrements.

Vous pouvez créer un fichier carnet.txt contenant un enregistrement par ligne, avec les valeurs séparés par des tabulations, et ordonnées comme les champs l'étaient dans la requête CREATE TABLE. Pour les données manquantes (comme un sexe inconnu ou la date de mort d'un animal toujours en vie), vous pouvez utiliser les valeurs NULL. Pour les représenter dans votre fichier texte, utilisez \N. Par exemple, l'enregistrement de Gwen ressemblera à (l'espace entre les valeurs est une tabulation):

```
3	Gwen	gwen@skynet.be	\N
```

numero	nom	email	prenom
3	Gwen	gwen@skynet.be	\N

Remarque: n'ajoutez pas de ligne vide !!

Pour charger le fichier carnet.txt dans la table carnet, utilisez cette commande:

```
mysql> LOAD DATA LOCAL INFILE "carnet.txt" INTO TABLE carnet;
Query OK, 1 row affected (0,00 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
On vérifie
mysql> select * from carnet;
+--------+----------+--------------------+--------+
| numero | nom      | email              | prenom |
+--------+----------+--------------------+--------+
|      1 | Plomteux | michel@plomteux.net | michel |
|      2 | Nemard   | nemard@savon.net    | jean   |
|      3 | Gwen     | gwen@skynet.be      | NULL   |
+--------+----------+--------------------+--------+
3 rows in set (0,00 sec)
```

Vous pouvez spécifier la valeur du séparateur de colonnes et le marqueur de fin de lignes explicitement dans la commande LOAD DATA si vous le voulez, mais les valeurs par défaut sont la tabulation et le retour à la ligne.

Fichier csv

Un fichier CSV n'est rien d'autre qu'un fichier texte. tous les champs de la base sont séparés par un séparateur (habituellement ;).

La commande LOAD DATA INFILE lit les lignes dans un fichier texte et les insère à très grande vitesse. Si le mot clé LOCAL est spécifié, il est interprété en suivant les règles suivantes :

- Si LOCAL est spécifié, le fichier est lu par le programme client, et envoyé vers l'hôte.
- Si LOCAL n'est pas spécifiée, le fichier doit être sur le serveur hôte, et sera lu directement par le serveur.

Pour des raisons de sécurité, lorsque les fichiers sont lus sur le serveur, ils doivent se trouver dans le répertoire de la base de données courante, ou bien être lisible par tous. Pour utiliser la commande LOAD DATA INFILE sur des fichiers du serveur, vous devez avoir le droit de FILE sur le serveur.

Utiliser LOCAL est plus lent que de laisser le serveur accéder directement aux fichiers, car le contenu du fichier doit être envoyé via le réseau au serveur. D'un autre coté, vous n'aurez pas besoin

de droits de FILE pour faire un chargement local.

```
LOAD DATA LOCAL INFILE '/importfile.csv'
INTO TABLE test_table
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
(field1, filed2, field3);
```

Le fichier de données à importer doit être structuré avec des caractères séparateurs de champs et de lignes. La plupart des tableurs permettent de générer de tels fichiers (format CSV : séparateur point-virgule par exemple). L'importation se fait automatiquement grâce à l'instruction SQL

```
LOAD DATA [LOCAL] INFILE 'nom_fichier' INTO TABLE nom_table
```

yntaxe de LOAD DATA INFILE

```
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,...)]
```

La commande LOAD DATA INFILE lit les lignes dans un fichier texte et les insère à très grande vitesse.

Si LOCAL n'est pas spécifiée, le fichier doit être sur le serveur hôte, et sera lu directement par le serveur.

Si vous ne spécifiez par de clause FIELDS, les valeurs par défaut sont :

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

Si vous ne spécifiez par de clause LINES, les valeurs par défaut sont :

```
LINES TERMINATED BY '\n'
Attention,
      sous windows :LINES TERMINATED BY '\r\n' ;
      OS X, on préfèrera  LINES TERMINATED BY '\r'
```

L'option IGNORE nombre LINES sert à ignorer une en-tête de fichier, telle que des noms de colonnes, qui débutent parfois un fichier à charger :

```
mysql> LOAD DATA INFILE "/tmp/nom_fichier" INTO TABLE test IGNORE 1 LINES ;
```

exemple

carnet2.txt

```
4;Handrin;Ahandrin@free.fr;alex
```

```
mysql> LOAD DATA LOCAL INFILE "carnet2.txt" INTO TABLE carnet
```

```
    -> FIELDS TERMINATED BY ';' LINES TERMINATED BY '\r\n' ;
Query OK, 1 row affected (0.01 sec)
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from carnet;
+--------+----------+--------------------+--------+
| numero | nom      | email              | prenom |
+--------+----------+--------------------+--------+
|      1 | Plomteux | michel@plomteux.net | michel |
|      2 | Nemard   | nemard@savon.net    | Jean   |
|      3 | Gwen     | gwen@skynet.be      | NULL   |
|      4 | Handrin  | Ahandrin@free.fr    | alex   |
```

Supprimer un enregistrement:

DELETE FROM la_table WHERE champ1='valeur1;

```
mysql> Delete from carnet where numero=3;
Query OK, 1 row affected (0,00 sec)

mysql> select * from carnet;
+--------+----------+--------------------+--------+
| numero | nom      | email              | prenom |
+--------+----------+--------------------+--------+
|      1 | Plomteux | michel@plomteux.net | michel |
|      2 | Nemard   | nemard@savon.net    | jean   |
+--------+----------+--------------------+--------+
2 rows in set (0,00 sec)
```

Un exemple plus complet

Soit le fichier:

```
5	Dhame	lagrosse@swing.be	jeanne
6	Therieur	ater@blegacom.be	alex
7	Therieur	alainter@free.fr	alain
8	Porte	dufric@radin.com	shara
9	Konnu	\N	alain
```

```
mysql> LOAD DATA LOCAL INFILE "carnet.txt" INTO TABLE carnet;
Query OK, 7 rows affected (0,00 sec)
Records: 7  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from carnet;
+--------+----------+---------------------+--------+
| numero | nom      | email               | prenom |
+--------+----------+---------------------+--------+
|      1 | Plomteux | michel@plomteux.net | michel |
|      2 | Nemard   | nemard@savon.net    | jean   |
|      3 | Gwen     | gwen@skynet.be      | NULL   |
|      4 | Handrin  | Ahandrin@free.fr    | alex   |
|      5 | Dhame    | lagrosse@swing.be   | jeanne |
|      6 | Therieur | ater@blegacom.be    | alex   |
|      7 | Therieur | alainter@free.fr    | alain  |
|      8 | Porte    | dufric@radin.com    | shara  |
|      9 | Konnu    | NULL                | alain  |
+--------+----------+---------------------+--------+
9 rows in set (0,00 sec)
```

Récupérer des informations à partir d'une table

La commande `SELECT` est utilisée pour récupérer des informations à partir d'une table. La forme usuelle est:

```
SELECT quoi_selectionner (les champs)
FROM quelle_table
WHERE conditions_a_satisfaire
```

Sélectionner toutes les données *

La plus simple forme de `SELECT` récupère toutes les données d'une table:

```
mysql> SELECT * FROM carnet;
```

Cette forme de `SELECT` est utile si vous voulez récupérer la table entière. Par exemple, après l'avoir juste remplie avec vos données d'origine.

```
mysql> select * from carnet;
+--------+----------+---------------------+--------+
| numero | nom      | email               | prenom |
+--------+----------+---------------------+--------+
|      1 | Plomteux | michel@plomteux.net | michel |
|      2 | Nemard   | nemard@savon.net    | jean   |
|      3 | Gwen     | gwen@skynet.be      | NULL   |
|      4 | Handrin  | Ahandrin@free.fr    | alex   |
|      5 | Dhame    | lagrosse@swing.be   | jeanne |
|      6 | Therieur | ater@blegacom.be    | alex   |
|      7 | Therieur | alainter@free.fr    | alain  |
|      8 | Porte    | dufric@radin.com    | shara  |
|      9 | Konnu    | NULL                | alain  |
+--------+----------+---------------------+--------+
```

Sélectionner des colonnes particulières

Si vous ne voulez pas voir les lignes entières de votre table, nommez les colonnes qui vous intéressent, en les séparant par des virgules.

```
mysql> select nom,prenom from carnet;
+----------+--------+
| nom      | prenom |
+----------+--------+
| Dhame    | jeanne |
| Gwen     | NULL   |
| Handrin  | alex   |
| Konnu    | alain  |
| Nemard   | jean   |
| Plomteux | michel |
| Porte    | shara  |
| Therieur | alain  |
| Therieur | alex   |
+----------+--------+
```

Trier les enregistrements

Vous avez sûrement noté dans les exemples précédents que les lignes de résultat sont affichées sans ordre particulier. Cependant, il est souvent plus facile d'examiner les résultats lorsqu'ils sont triés d'une manière significative. Pour trier un résultat, vous devez utiliser une clause `ORDER BY`.

Le tri, comme toutes les opérations de comparaison, est normalement exécuté sans tenir compte de la casse. Cela signifie que l'ordre sera indéfini pour les colonnes qui sont identiques, excepté leur casse. Vous pouvez forcer le tri sensible à la casse en utilisant la clause BINARY: `ORDER BY BINARY(champ)`.

```
 select nom,prenom from carnet order by nom,prenom;
+----------+--------+
| nom      | prenom |
+----------+--------+
| Dhame    | jeanne |
| Gwen     | NULL   |
| Handrin  | alex   |
| Konnu    | alain  |
| Nemard   | jean   |
| Plomteux | michel |
| Porte    | shara  |
| Therieur | alain  |
| Therieur | alex   |
+----------+--------+
```

Pour trier dans l'ordre inverse, ajoutez le mot-clé `DESC` (décroissant) au nom de la colonne à trier.

Sélectionner des lignes particulières "where"

Vous pouvez sélectionner des lignes particulières de votre table.

Trouver la famille "Therieur":

```
select nom,prenom from carnet where nom='therieur';
+----------+--------+
| nom      | prenom |
+----------+--------+
| Therieur | alain  |
| Therieur | alex   |
+----------+--------+
```

On a appliqué l'operateur logique "equivalent à" =.

Trouver tous les prénoms comme "Al ...quelque chose"

```
select nom,prenom from carnet where prenom like 'al%';
+----------+--------+
| nom      | prenom |
+----------+--------+
| Handrin  | alex   |
| Konnu    | alain  |
| Therieur | alain  |
| Therieur | alex   |
+----------+--------+
```

LIKE

La réalisation d'expression utilisant les expressions régulières simples de comparaison de SQL. Retourne 1 (TRUE) ou 0 (FALSE). Avec LIKE, vous pouvez utiliser les deux jokers suivants :

Char	Description
%	Remplace n'importe quel nombre de caractères, y compris aucun
_	Remplace exactement **un** caractère

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
        -> 1
```

Liste des noms contennant la lettre T

```
select nom,prenom from carnet where nom like '%t%';
+----------+--------+
| nom      | prenom |
+----------+--------+
| Plomteux | michel |
| Porte    | shara  |
| Therieur | alain  |
| Therieur | alex   |
+----------+--------+
```

Liste des clients de free:

```
select nom,prenom from carnet where email like '%free%';
+----------+--------+
| nom      | prenom |
+----------+--------+
| Handrin  | alex   |
| Therieur | alain  |
+----------+--------+
```

Problèmes avec les valeurs NULL

Le concept de la valeur NULL est une source de confusions pour les débutants en SQL, qui pensent

souvent que NULL est la même chose qu'une chaîne de caractères vide "". Ce n'est pas le cas !

Pour trouver les valeurs NULL, vous devez utiliser le test **IS NULL**.

```
select nom,prenom from carnet where email is null;
+-------+--------+
| nom   | prenom |
+-------+--------+
| Konnu | alain  |
+-------+--------+
```

et **IS NOT NULL** pour les champs non "null".

```
select nom,prenom from carnet where email is not null;
+----------+--------+
| nom      | prenom |
+----------+--------+
| Plomteux | michel |
| Nemard   | jean   |
| Gwen     | NULL   |
| Handrin  | alex   |
| Dhame    | jeanne |
| Therieur | alex   |
| Therieur | alain  |
| Porte    | shara  |
+----------+--------+
8 rows in set (0,00 sec)
```

rappel:Lors de la lecture de données avec LOAD DATA INFILE, les colonnes vides sont interprétées en tant que ". Si vous voulez une valeur NULL dans une colonne, vous devez utiliser \N dans le fichier.

Lors de l'utilisation de ORDER BY, les valeurs NULL sont présentées en premier. Si vous triez dans l'ordre décroissant en utilisant DESC, les valeurs NULL sont présentées en dernier. Lors de l'utilisation de GROUP BY, toutes les valeurs NULL sont considérées comme égales.

conversion des chaînes en nombres pour les opérations de comparaison

Les exemples suivants, montrent la conversion des chaînes en nombres pour les opérations de comparaison :

```
mysql> SELECT 1 > '6x';
         -> 0
mysql> SELECT 7 > '6x';
         -> 1
mysql> SELECT 0 > 'x6';
         -> 0
mysql> SELECT 0 = 'x6';
         -> 1
```

Egal :

```
mysql> SELECT 1 = 0;
        -> 0
mysql> SELECT '0' = 0;
        -> 1
mysql> SELECT '0.0' = 0;
        -> 1
mysql> SELECT '0.01' = 0;
        -> 0
mysql> SELECT '.01' = 0.01;
        -> 1
```

Opération	Opérateur SQL	Exemple	Commentaire
est égal à	=	…**where** `âge=25`	
est différent de	!= ou <>	…**where** `nombreEnfants!=0`	Selon les implémentations de SQL, les veux versions sont acceptées ou seulement une.
est strictement supérieur à	>	…**where** `âge>18`	
est strictement inférieur à	<	…**where** `numéro<1024`	
est supérieur ou égal à	>=	…**where** `âge>=10`	
est inférieur ou égal à	<=	…**where** `salaire<=1000`	
est compris entre … et …	**between** … **and**	…**where** `numéro` **between** `12` **and** `20`	Les bornes de l'intervalle sont généralement incluses.
est conforme à	**like**	…**where** `nom` **like** `"D%"`	**like** permet de vérifier qu'une chaîne de caractères est conforme à un modèle contenant _ et/ou `%`.
n'est pas conforme à	**not like**	…**where** `nom` **not like** `"l%e"`	**not like** est l'inverse de **like**.
est vide	**is null**	…**where** `salaire` **is null**	**is null** teste l'absence de valeur qui peut être assimilée à une valeur inconnue.
n'est pas vide	**is not null**	…**where** `nom` **is not null**	**is not null** est l'inverse de **is null**.
et	**and**	…**where** `âge>25` **and** `âge!=20`	Le **and** de SQL a généralement le comportement suivant : si le premier opérande permet de déterminer la réponse, le second n'est pas évalué.

Grouper

Exemple: liste des noms différents (il y a deux "Therieur").

```
select nom from carnet group by nom;
+----------+
| nom      |
+----------+
| Dhame    |
| Gwen     |
| Handrin  |
| Konnu    |
| Nemard   |
| Plomteux |
| Porte    |
| Therieur |
+----------+
8 rows in set (0,00 sec)
```

Fonctions avec GROUP BY

Si vous utilisez les fonctions de groupement avec une requête ne contenant pas de clause GROUP BY, cela revient à grouper toutes les lignes.

Compter les enregistrements count(champ)

COUNT(expr)

Retourne le nombre de valeurs non-NULL dans les lignes lues par la commande SELECT :

Exemple: nombre de clients de free:

```
 select count(*) from carnet where email like '%free%';
+----------+
| count(*) |
+----------+
|        2 |
+----------+
```

Petit exemple qui montre que les valeurs NULL ne sont pas prises en compte

```
mysql> select count(*) from carnet;
+----------+
| count(*) |
+----------+
|        9 |
+----------+
1 row in set (0,00 sec)

mysql> select count(email) from carnet;
+--------------+
| count(email) |
+--------------+
|            8 | (un email null sur les 9 enregistements)
+--------------+
1 row in set (0,00 sec)
```

Compter le nombre de personnes d'une même famille:

```
mysql> select nom ,count(*) from carnet group by nom;
+----------+----------+
| nom      | count(*) |
+----------+----------+
| Dhame    |        1 |
| Gwen     |        1 |
| Handrin  |        1 |
| Konnu    |        1 |
| Nemard   |        1 |
| Plomteux |        1 |
| Porte    |        1 |
| Therieur |        2 |
+----------+----------+
```

COUNT(DISTINCT expr,[expr...])

Retourne le nombre de valeurs non-NULL distinctes :

```
select count(distinct nom) from carnet;
+---------------------+
| count(distinct nom) |
+---------------------+
|                   8 |
+---------------------+
```

1 row in set (0,00 sec)

Tables liées

Imaginons que nous voulons gérer une vidéothèque avec une base de données. Cette base devra alors contenir des noms de films et pour chacun la catégorie de film à laquel ils appartient. En première réflexion, nous serions tentés de créer une table de la façon suivante:

```
CREATE TABLE films (nom varchar(64), categorie varchar(64))
```

et que l'on pourrait remplir comme suit:

Nom	Catégorie
La cité de la peur	Horreur
Matrix	Science-fiction
West side story	Comédie musicale
Men in black	Science-fiction

en utilisant les requêtes:

```
INSERT INTO films (nom,categorie) VALUES ('La cité de la peur','Horreur')
INSERT INTO films (nom,categorie) VALUES ('Matrix','Science-fiction')
INSERT INTO films (nom,categorie) VALUES ('West side story','Comédie musicale')
INSERT INTO films (nom,categorie) VALUES ('Men in black','Science-fiction')
```

Si l'on désire rechercher tous les enregistrements ayant pour catégorie 'Science-fiction' alors on pourra utiliser la requête:

```
SELECT * FROM films WHERE categorie='Science-fiction'
```

Mais voilà, la façon dont a été créé la table précédente est la plus basique mais ce n'est pas la meilleure façon de faire. Par exemple, cela implique d'être très rigoureux à chaque saisie de catégorie afin de toujours choisir les mêmes termes pour désigner le même genre et de toujours l'orthographier de la même façon (majucule/minuscule) si l'on désire faire un tri sur la catégorie.

Dans la pratique cela implique de suggérer à l'utilisateur de choisir la catégorie parmi une liste donnée. Et pourquoi alors, ne pas stocker dans la BD la liste des catégories?

Id	Categorie
1	Comédie musicale
2	Horreur
3	Humour
4	Policier
5	Science-fiction

```
CREATE TABLE categories (id int4, categorie varchar(64));
```

Remarquez qu'au passage, j'ai ajouté un identifiant (id de type entier) à chaque catégorie, ce qui pourra donner une table du type:

Dans ce cas, associer à un nom de film, une catégorie revient à associer à un nom de film un identifiant de catégorie. La table initialement imaginée devient donc

```
CREATE TABLE films (nom varchar(64), categorieid int4)
```

et que l'on pourrait remplir comme suit:

Nom	CategorieId
La cité de la peur	2
Matrix	5
West side story	1
Men in black	5

Certes, à première vue c'est moins lisible et plus compliqué que la première solution retenue. Mais qu'avons nous gagné ?

1. La liste des catégories est prédéfinie (et possède sa propre table) donc pour chaque film, on ne saisira pas un "nom" de catégorie mais on en selectionnera une dans une liste (limitant

ainsi tout risque de confusion).
2. Si l'on désire changer l'intitulé d'un catégorie (ex "Epouvante" au lieu de "Horreur") il suffit de faire la correction dans la liste des catégories sans avoir à passer en revue la table des films
3. On a gagné en place disque occupée. Si l'on considère qu'un nom de catégorie c'est en moyenne 10 caractères, un nom de film 25 caractères et que l'on a 1000 films dans 20 catégories. Dans le cas 1 la table film occupe en gros 1000*(25+10)=35000 octets alors que dans le cas 2 les tables films et catégories occupent à peu près 1000*(25+4)+20*(4+10)=29280 (sachant que l'identifiant de catégorie de type int4 occupe 4 octets) soit un gain de 16% (le gain serait bien plus important si la taille moyenne des noms de catégorie était plus grande)

Comme vous le constatez il y a de nombreux avantages à faire de la sorte. Encore faut-il pouvoir faire le lien entre les 2 tables me direz-vous... Pour cela, il suffit de faire référence aux 2 tables lors des requêtes SELECT et de les associer avec la condition WHERE adéquate, comme suit:

```
SELECT * FROM films,categories WHERE film.categorieid=categories.id
```

Indiquant ainsi que le champ id de la table categories doit coïncider avec le champ categorieid de la table film.

Remarques:

Comme, dans notre cas, il ne peut y avoir de confusion (categorieid ne peut qu'être un champ de la table film et id un champ de la table categorie) nous aurions également pu écrire "SELECT * FROM films,categories WHERE categorieid=id"

Liens 1 à plusieurs

Dans le chapitre "Tables liées" nous avons vu comment associé à un élément d'une table un élément d'une autre table (lien 1 à 1). Dans ce chapitre, nous allons étendre le principe au lien 1 à plusieurs (autrement dit associer à un enregistrement d'une table plusieurs enregistrements d'une autre table). [REM dans ce cas, il s'agit même d'un lien plusieurs à plusieurs puisque plusieurs films peuvent être associé à un même acteur]

Car, pour reprendre l'exemple de ce précédent chapitre, si l'on peut admettre qu'à un film donné ne correspond qu'une catégorie, il en va tout autrement des acteurs. Et il n'est évidemment pas question de créer une table contenant par exemple les champs film,acteur1,acteur2,acteur3 (dans ce cas à combien limiter le nombre d'acteur? quel requête executer pour retrouver le film dans lequel joue "louis de funes"?).

Non, la solution passe par la création d'une table intermediaire entre la table contenant les films et la table contenant les acteurs.

Table Films

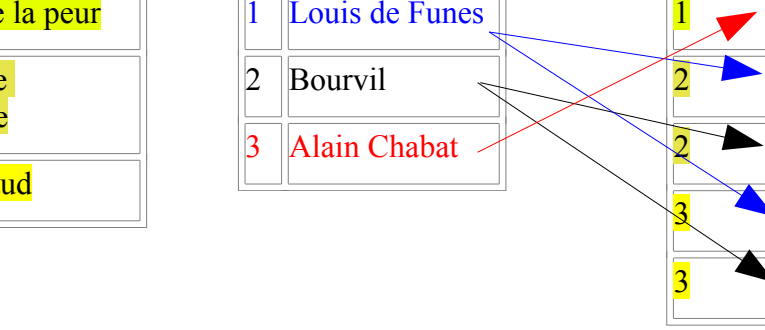
Id	Film
1	La cité de la peur
2	La grande vadrouille
3	Le corniaud

Table Acteurs

Id	Acteur
1	Louis de Funes
2	Bourvil
3	Alain Chabat

Table Liens_Films_Acteurs

FilmId	ActeurId
1	3
2	1
2	2
3	1
3	2



A partir de là, il suffit d'appliquer le même principe que ce nous avions vu précédemment.

```
SELECT * FROM films,acteurs,liens_films_acteurs WHERE films.id=filmid AND acteurs.id=acteurid
```