atelier freelance
http://www.atelier-freelance.ch
pierre.calame@atelier-freelance.ch
Formation Linux

Introduction à MySQL

Vous pouvez trouver ce document sur http://www.atelier-freelance.ch

atelier freelance
S'intègrer à votre projet

Introduction à MySQL

Atelier Freelance
S'intègrer à votre projet

Introduction à MySQL

Au sujet de ce support

Ce support est utilisé dans le cadre de cours dont les participants suivent une formation complète. Des sujets complémentaires comme l'installation de MySQL sont traités dans un autre module. Vous devriez trouver les supports concernant ces autres modules également sur http://www.atelier-freelance.ch/.

Conventions utilisées dans ce support
Les caractères en *courrier italique* identifient les commandes passées au shell sous Linux.
Ce symbole « ↵ » représente l'appui sur la touche Return ou Enter.
Les éléments contenus dans des crochets [TEMPORY | AUTRES] sont facultatifs.

Présentation générale

Le logiciel MySQL (TM) est un serveur de base de données SQL. MySQL est une marque déposée de MySQL AB. Le logiciel MySQL dispose de deux licences. Les utilisateurs peuvent choisir entre utiliser MySQL comme un logiciel Open Source/Logiciel libre, sous les termes de la licence GNU General Public License (http://www.gnu.org/licenses/) ou bien, ils peuvent acheter une licence commerciale auprès de MySQL AB.
Consultez http://www.mysql.com/ pour obtenir les dernières informations sur le serveur MySQL.

Définition officielle de MySQL :
MySQL est un système de gestion de bases de données relationnelles. Le SQL dans "MySQL" signifie "Structured Query Language" : le langage standard pour les traitements de bases de données.
MySQL est Open Source. Open Source (Standard Ouvert) signifie qu'il est possible à chacun d'utiliser et de modifier le logiciel. Tout le monde peut le télécharger sur Internet et l'utiliser sans payer aucun droit. Toute personne en ayant la volonté peut étudier et modifier le code source pour l'adapter à ses besoins propres. Toutefois, si vous devez intégrer MySQL dans une application commerciale, vous devez vous procurer une licence auprès de MySQL AB.

Ce qui rend MySQL très intéressant pour les Webmasters est le nombre d'API (application program interface) dont il dispose. Vous pouvez en effet l'intégrer dans des applications écrites en : C, C++, Eiffel, Java, Perl, PHP, Python, Ruby et Tcl.

Découverte du langage SQL

Que vous utilisiez PHP ou un autre langage de développement, vous aurez besoin de connaître les bases du langage SQL. Pour ce faire, nous allons élaborer, étape par étape, la base de données « MySample ».

Pour le moment, elle sera composée d'une seule table, la table "**films**"

Introduction à MySQL

Pour les commandes, j'utilise le client MySQL en mode texte sous Linux. Il me semble être le meilleur pour une approche qui se veut didactique. Toutefois, vous trouverez nombre de produits qui vous permettront de travailler à l'aide de la souris.

Pour des raisons de clarté, les commandes SQL seront écrites en capitales. Le langage n'est pas sensible à la casse.

Utilisation de base du client MySQL

Après l'installation du serveur, l'utilisateur root est créé et possède tous les droits à l'exception de la connexion distante. Il est donc important de créer un utilisateur possédant un accès protégé par un mot de passe, afin d'administrer votre serveur. Reportez-vous à la section « Les droits sous MySQL » si vous devez mettre votre serveur en production.

Sous Linux en mode texte appelez le client MySQL :
mysql ↵
(ou si vous devez fournir un mot de passe mysql –u *user* –p)
Le prompt de mysql client a la forme suivante :
mysql>
C'est depuis cette invite que nous entrerons les commandes SQL. Pour quitter le client entrez la commande quit ou utilisez la combinaison de touches Ctrl+D

Vous pouvez interroger le serveur sur la version courante de MySQL de la manière suivante :
mysql> SELECT VERSION() ;

Les commandes de base :

help	(\h)	Affiche un message d'aide.
?	(\?)	Idem
clear	(\c)	Stoppe la commande en cours de construction.
connect	(\r)	Rouvre la connexion.
edit	(\e)	Edite la commande dans l'éditeur spécifié par $EDITOR.
ego	(\G)	Envoie la commande au serveur. L'affichage se fait sous forme de fiche
exit	(\q)	Quitte.
go	(\g)	Envoie la commande au serveur.
nopager	(\n)	Stoppe le mode pager.
notee	(\t)	Don't write into outfile.
pager	(\P)	Affiche les résultats dans un fichier plutôt qu'à l'écran. Pager cat >> file
print	(\p)	Affiche la commande en cours.
quit	(\q)	Quitte.
rehash	(\#)	Rebuild completion hash.
source	(\.)	Exécute une commande SQL depuis un fichier.
status	(\s)	Retourne des infos sur le statut du serveur.
tee	(\T)	Set outfile [to_outfile]. Append everything into given outfile.
use	(\u)	Pour changer la base de données utilisée par défaut.

Par défaut le client mysql retourne les résultats à l'écran. A des fins de débogage, vous pourriez préférer une sortie dans un fichier. La commande pager vous le permet :
Page cat >> mon_file.txt
Le double >> permet de ne pas écraser les infos qui se trouvent dans le file, mais de les ajouter au fur et à mesure.

Introduction à MySQL

Création d'une base de données

Commençons par la création de la base de données :
mysql>CREATE DATABASE MySample ; ↵

Remarquez le point-virgule qui termine la commande, si vous ne le mettez pas, la commande ne sera pas exécutée. Dans ce cas, vous pouvez lancer son exécution à l'aide de la commande \g.

Les commandes SQL ne sont pas sensibles à la casse, par contre votre OS peut l'être. Les tables et les bases de données étant stockées dans des fichiers, vous devez respecter les règles utilisées par votre OS. Si vous travaillez sous Linux qui, lui, est sensible à la casse, vous devrez respecter la casse utilisée pour vos noms de tables. Sous Windows, ce ne sera pas nécessaire.

La base de données est bien créée, vous devriez pouvoir la voir à l'aide de la commande :
mysql>SHOW DATABASES ;

La base est créée, nous allons en faire la base de données par défaut. Ceci nous évitera d'entrer le nom de la base de données à chaque commande SQL.

mysql>use MySample ;

Création de la table

La table **films :**

```
mysql>CREATE TABLE films(id_film INT UNSIGNED NOT NULL AUTO_INCREMENT
        PRIMARY KEY,
        realisateur     VARCHAR(200),
        titre           VARCHAR(100)
        );
```

Vous remarquerez que tant que vous n'avez pas entré les ; la commande peut facilement s'étendre sur plusieurs lignes, vous pouvez également, pour en améliorer la lisibilité, ajouter des espaces.

A l'aide de la commande suivante, vous pourrez visualiser la structure de votre table :
myslq>describe MySample ;

Pour plus de détail sur la création de table, vous pouvez vous référer à l'annexe « Les commandes en détail ».

Dans un ordre logique des choses, maintenant que notre table est créée, il nous reste à la remplir.

Insertion de data

Nous allons voir deux techniques permettant d'insérer des datas. La première, directement à l'aide de la commande SQL INSERT.

Nourrissons la table films :

```
INSERT INTO films(titre,realisateur)
                values("Beau travail","Claire Denis");
```

La première parenthèse contient les champs concernés par l'insertion, les autres champs recevront la valeur par défaut définie lors de la création de la table. La deuxième parenthèse contient les valeurs qui seront mises dans les différents champs. Les " permettent d'identifier les différentes chaînes de caractères. Si une valeur devait contenir explicitement un ", vous devez le faire suivre du caractère de protection \: Exemple ;
INSERT INTO films(titre,resume,pays)
values("Terminator","Un robot venu du futur (\"2025\")…", "USA");

L'autre méthode consiste à aller lire les différents datas dans un fichier texte.

Importation depuis un fichier texte :
Le contenu du fichier correspond à ceci :

"titre 1","resume 1","pays 1"
"titre 2","resume 2","pays 2"
"titre 3","resume 3","pays 3"
"titre 4","resume 4","pays 4"
…

LOAD DATA LOCAL INFILE 'films.txt' INTO TABLE films FIELDS TERMINATED BY "," ENCLOSED BY "\"" (titre,resume,pays);

Le caractère \ devant le " permet de spécifier qu'il ne faut pas utiliser le " comme fin de chaîne.

Pour plus de détail sur la commande LOAD DATA, consultez "Les commandes en détail"

Sélectionner des enregistrements

La commande clé pour ce genre de requête est SELECT. C'est assurément la commande que nous utiliserons le plus.

Syntaxe de la commande SELECT :
SELECT [champ1,champ2, … | *]
FROM [table 1, table 2, …]
WHERE [condition or, and ou like]

Pour bien comprendre cette commande, vous trouverez ci-dessous une série d'exemples documentés :

SELECT titre FROM films ORDER BY titre;
Tous les titres de la table films triés de manière croissante par titre

SELECT titre FROM films ORDER BY titre DESC;
Tous les titres de la table films triés par titre de manière décroissante

SELECT * from films where pays = "USA";
Tous les champs de la table films dont le pays est égal à "USA"

Passons à la pratique I

- Créez tout d'abord une nouvelle base de données que nous appellerons **cours**.
- Créez ensuite une table se composant des champs suivants :
  id_film -> valeur entière, supérieure à 0, pas de valeur nulle autorisée, auto

incrément et clé primaire
realisateur -> Chaîne de caractères maximum 200
titre -> Chaîne de caractères maximum 200

Vous trouverez sur les données qui vous ont été remises (ou sur le site web de votre serviteur : http://www.atelier-freelance.ch/….) un fichier texte « films.txt ». Ce fichier contient une liste de titre de films et le nom du réalisateur. Les champs sont entourés de " et séparés par des ;. Chaque enregistrement est séparé par un retour de paragraphe.

- Pour importer les datas contenus dans ce fichier, utilisez la commande LOAD INFILE
  En effectuant une simple commande ***SELECT * FROM films;*** vous devriez pouvoir contrôler si l'importation s'est bien passée. Si ça ne devait pas être le cas, utilisez la commande ***DELETE FROM films;*** pour supprimer toutes les entrées.

Maintenant que nous avons une table contenant plusieurs enregistrements, vous allez pouvoir expérimenter la commande ***SELECT*** pour répondre aux questions suivantes :

- Tous les films et les réalisateurs
- Idem, mais triés par réalisateurs
- Idem, mais triés par films de manière décroissante
- Tous les films réalisés par Jean-Luc Godard

Note : *Les exercices pratiques sont corrigés dans l'annexe "Exercices corrigés". Je vous conseille vivement de ne consulter les corrections proposées qu'en dernier recours ou pour contrôle après votre pratique.*

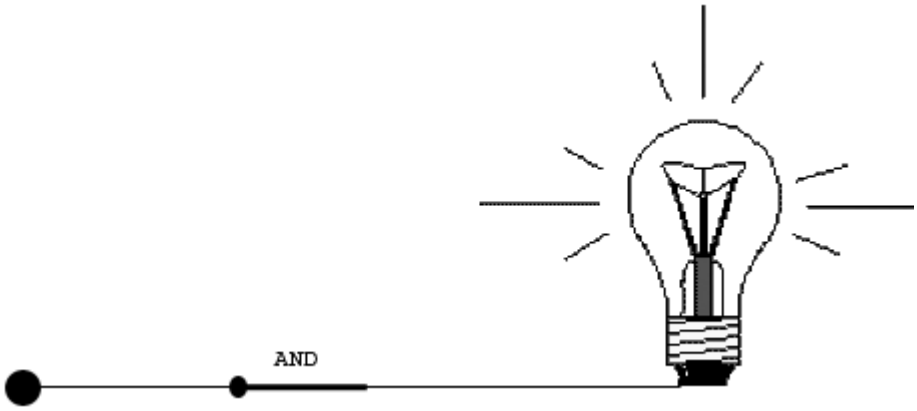
La commande WHERE

Nous l'avons vue dans les exercices précédents, la commande WHERE permet d'imposer une condition à une requête. Les quelques exemples ci-dessus vous donneront une idée de l'étendue des possibilités de cette commande :

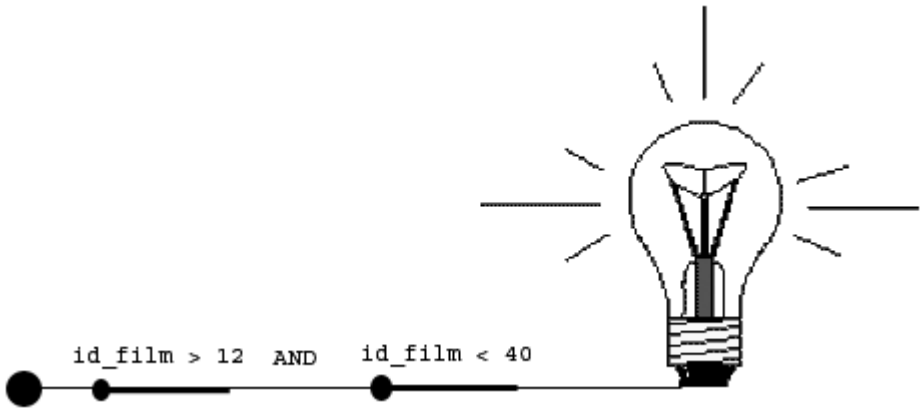
SELECT * FROM films WHERE id_film > 12;
Tous les films dont le id_film est plus grand que 12

SELECT * FROM films WHERE id_film > 12 AND id_film < 40;
Tous les films dont le id_film est plus grand que 12 et plus petit que 40

L'opérateur AND peut être comparé à un interrupteur qui laisse ou coupe le passage du courant électrique. Lorsque le courant passe, la condition est satisfaite, sinon le courant est coupé.



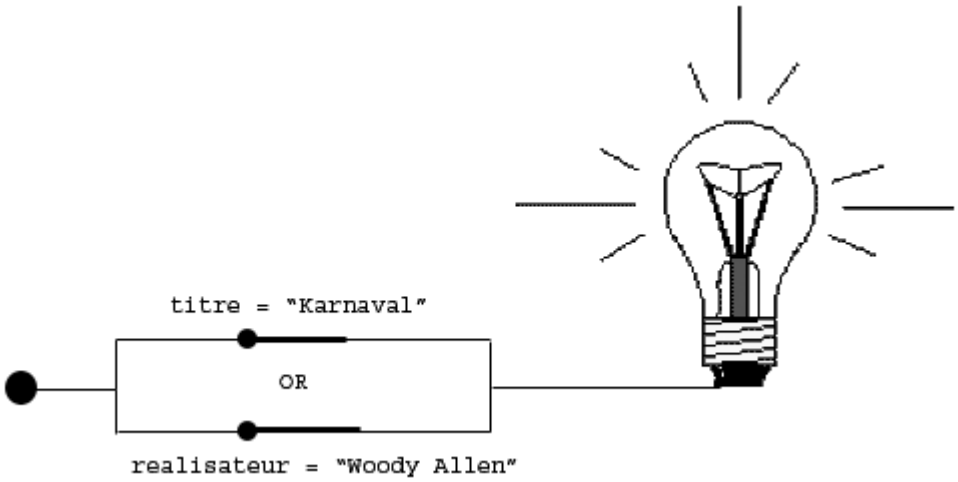
Si vous multipliez les opérateurs AND, vous ajoutez des interrupteurs en reprenant la requête ci-dessus :



Pour que l'ampoule s'allume, les deux interrupteurs doivent être fermés. Autrement dit, la ligne ne sera affichée que si les deux conditions retournent vrai, soit que le id_film soit plus grand que 12, mais également plus petit que 40.

L'opérateur OR peut aussi être représenté de cette manière. Il laissera passer le courant pour autant que l'un des interrupteurs soit fermé. Exemple :

SELECT * FROM films WHERE titre = "Karnaval" OR realisateur = "Woody Allen";



Pour que l'ampoule soit allumée, titre doit être égal à Karnaval ou le réalisateur doit être Woody Allen. La requête retournera tous les films réalisés par Woody Allen et sortira également le film dont le titre est "Karnaval"

On peut combiner les AND et les OR en les entourant de parenthèses
SELECT * FROM films WHERE (id_film > 12 AND id_film < 40) OR (titre = "KARNAVAL" or realisateur = "Woody Allen");

D'autres opérateurs :

L'opérateur LIKE permet de filtrer selon un masque

SELECT * FROM films WHERE titre LIKE "V%";
Tous les films dont le titre commence par "V"

SELECT * FROM films WHERE realisateur LIKE ="C_____ %";
Tous les films dont le nom du réalisateur commence par un C suivi de 5 caractères quelconques, suivi d'un espace et d'une chaîne.

Recherche selon une expression régulière
Les expressions régulières permettent d'aller beaucoup plus loin que les modèles de LIKE

SELECT * FROM films WHERE titre REGEXP "^M";
Tous les films qui commencent pas un M. Le symbole ^ signifie début de la chaîne

SELECT * FROM films WHERE titre REGEXP "^M[er]";
Tous les films qui commencent pas un M suivi d'un e ou d'un r

SELECT * FROM films WHERE realisateur REGEXP "o{2}";
Tous les films dont le réalisateur a un nom contenant deux o qui se suivent

Les expressions régulières sont très puissantes, mais aussi très complexes, nous nous arrêterons à ces quelques exemples.

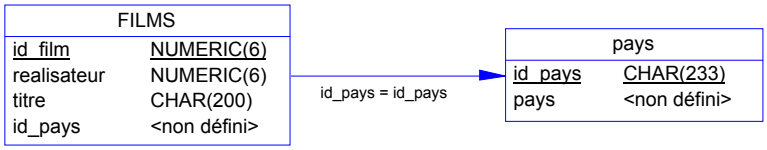
Les regroupements et les fonctions

Une requête permet également de répondre à une question comme "Combien de films a réalisés Jean-Luc Godard

SELECT COUNT(id_film), realisateur FROM films WHERE realisateur = "Jean-Luc Godard" GROUP BY realisateur;

Requête utilisant plus d'une table

Reprenons l'exemple de notre table films en y ajoutant une table qui contient les pays.



La table FILMS contient le champ id_pays qui représente la clé étrangère permettant de faire le lien avec le champ id_pays de la table PAYS.

Si nous voulons sélectionner tous les titres de films, ainsi que le nom du pays correspondant, nous aurions :

SELECT titre, pays FROM films, pays WHERE films.id_pays = pays.id_pays;

Ou encore

SELECT titre, pays FROM films INNER JOIN pays ON pays.id_pays = films.id_pays;

Sous MySQL, ces deux requêtes sont identiques au niveau performance et au niveau résultat

Mettre à jour le contenu d'une table

La commande qui permet la mise à jour d'un contenu est UPDATE. Utilisons-la pour modifier le film 3 de notre table :

mysql> UPDATE films SET titre = "Nouveau titre" WHERE id_film = 3;

La commande UPDATE permet de mettre à jour plusieurs champs (séparés par des virgules). Comme par exemple :

Mysql> UPDATE films SET titre ="Nouveau titre", realisateur = "Woody Allen" WHERE id_film = 3;

Effacer des enregistrements

DELETE permet d'effacer les lignes d'une table, mais attention, ça ne pardonne pas :

mysql> DELETE FROM films;

La commande ci-dessus a nettoyé votre table sans vous demander si vous êtiez sûr de vouloir supprimer tous les enregistrements. Notez que si vous oubliez de spécifier un critère à une requête UPDATE, vous risquez aussi de causer pas mal de dommages à vos données. Pour éviter cette erreur fort désagréable, ajoutez la ligne suivante au fichier /etc/my-cnf :

[mysql]
safe-updates

Avec cette directive, vous recevrez un message d'erreur lorsque vous aurez omis le critère.
Pour supprimer le film ayant le id_film 3

mysql> DELETE FROM films WHERE id_film = 3;

Administration

L'administration d'une base de données MySQL sous-entend au minimum les opérations suivantes :

- Backup régulier de la base
- Paramétrer le serveur

Introduction à MySQL

Sauvegarder les datas de vos bases de données

Il existe un utilitaire livré avec MySQL qui se nomme mysqldump. Il transcrit l'intégralité d'une ou plusieurs bases en commande SQL, ce qui permet de restaurer ou transférer des datas sur un autre serveur. (Y compris un serveur SQL autre que MySQL)

Utilisation de mysqldump en ligne de commande

mysqldump –A

retourne toutes les commandes, pour les récupérer redirigez la sortie dans un fichier :

mysqldump –A > MonBackup.sql

Pour sauvegarder uniquement une base :

mysqldump –B maBase > MonBackup.sql

Vous pouvez également ne sauver que quelques tables

mysqldump –B maBase –tables table_1 table_2 > MonBackup.sql

Les options suivantes sont également très pratiques :
--add-drop-table Supprime les tables qui existent déjà, permet de restaurer sans être obligé de supprimer la base
--force Continue malgré des messages d'erreur du serveur

Il en existe beaucoup d'autres, pour obtenir la liste de ces options :

mysqldump –help

Un autre utilitaire mysqlhotcopy permet de copier, après avoir vidé les caches, toute la base de données passée en argument :

mysqlhotcopy mysql /chemin/de/la/sauvegarde

Réparer une base de données

Si votre serveur semble ne pas être en mesure de lire certaines tables ou a un comportement suspect, vous pouvez tester l'intégrité des datas avec la commande CHECK TABLES

mysql>CHECK TABLES employes;

Si MySQL vous retourne en effet des erreurs, vous pouvez effectuer la réparation avec REPAIR TABLES employes;

Dans la documentation de MySQL, ils prétendent que la réussite est de 99.9%. Dans le cas où l'erreur persisterait, vous pouvez vous tourner vers l'utilitaire en ligne de commande myisamchk /path/to/file

Introduction à MySQL

Traduire un code d'erreur

Autre utilitaire intéressant livré avec MySQL est perror. Il vous retourne le texte relatif à un numéro d'erreur :

perror 13

Modifier le comportement du serveur

(Repris de la documentation officielle)
MySQL, depuis la version 3.22, lit les options de démarrage dans un fichier :

Sous Unix : /etc/my.cnf pour les options globales. DATADIR/my.cnf pour les options spécifiques au serveur et ~/.my.cnf pour celles spécifiques à l'utilisateur.

MySQL essaie de lire les fichiers d'options dans l'ordre dans lequel ils sont présentés ci-dessus. Si des options sont spécifiées plusieurs fois, la dernière occurrence utilisée prend la préséance sur les options précédentes. Les options de ligne de commande ont la priorité sur les options spécifiées dans les fichiers. Certaines options peuvent être spécifiées en utilisant des variables d'environnement.

Les options spécifiées en ligne de commande ou en fichier ont la priorité sur les options qui le sont via une variable d'environnement.

Les programmes suivants utilisent les fichiers d'options : mysql, mysqladmin, mysqld, mysqld_safe, mysql.server, mysqldump, mysqlimport, mysqlshow, mysqlcheck, myisamchk et myisampack.

Toute option longue qui doit être spécifiée en ligne de commande lorsque MySQL fonctionne, peut aussi être configurée dans le fichier d'options (sans les doubles tirets).

Un fichier d'options contient des lignes ayant la forme suivante :

Les lignes de commentaires commencent avec `#
[group]
group est le nom du programme ou du groupe pour lequel vous souhaitez configurer des options. Après une ligne de groupe, toutes les options et set-variable s'appliqueront au groupe nommé, jusqu'à la fin du fichier d'option ou du démarrage d'un autre groupe.

option=value
Ceci est équivalent à --option=value sur la ligne de commande.

set-variable = variable=value
Ceci est équivalent à --set-variable variable=value sur la ligne de commande.
Cette syntaxe doit être utilisée pour spécifier la valeur d'une variable mysqld.
Notez que --set-variable est obsolète depuis MySQL 4.0, utilisez simplement --variable=value comme tel.
Le groupe client vous permet de spécifier des options qui ne s'appliquent qu'aux clients MySQL et non pas au serveur mysqld. C'est le groupe idéal pour spécifier des mots de passe de connexion au serveur (mais assurez-vous que vous êtes le seul à accéder à ce fichier !!).

Introduction à MySQL

Notez que pour les options et les valeurs, tous les caractères blancs de début et de fin seront automatiquement effacés. Vous pouvez utiliser les séquences d'échappement '\b', '\t', '\n', '\r', '\\' et '\s' dans votre chaîne à la place ('\s' == espace).

Voici un exemple typique de fichier d'options globales :

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
set-variable = key_buffer_size=16M
set-variable = max_allowed_packet=1M

[mysqldump]
quick
```

Voici un exemple typique de fichier d'options utilisateur :

```
[client]
# Le mot de passe suivant va être utilisé avec le serveur
password=mon_mot_de_passe

[mysql]
no-auto-rehash
set-variable = connect_timeout=2
safe-updates
prompt=(\u@\h) [\d] >\_s

[mysqlhotcopy]
interactive-timeout
```

Si vous avez une distribution source, vous trouverez des exemples de configuration dans les fichiers nommés 'my-xxxx.cnf' dans le dossier 'support-files'. Si vous avez une distribution binaire, regardez dans le dossier 'DIR/support-files', où DIR est le chemin de l'installation MySQL (typiquement '/usr/local/mysql'). Actuellement, il y a des exemples de configuration pour des systèmes petits, moyens, grands et très grands. Vous pouvez copier l'un des fichiers 'my-xxxx.cnf' dans votre dossier utilisateur (renommez le fichier en '.my.cnf') pour le tester.

Tous les clients MySQL qui supportent les fichiers d'options, acceptent les options suivantes :

–no-defaults
: Ne lire aucun fichier d'options.

–print-defaults
: Affiche le nom du programme et toutes les options qui s'y trouvent.

–defaults-file=full-path-to-default-file
: Utilise uniquement le fichier de configuration donné.

–defaults-extra-file=full-path-to-default-file
: Lit ce fichier de configuration après le fichier de configuration global, mais avant le fichier de configuration utilisateur.

Notez que les options ci-dessus doivent être en ligne de commande pour être utilisées. En scripts shell, vous pouvez utiliser la commande 'my_print_defaults' pour analyser les fichiers de configuration :

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

La ligne ci-dessus affiche toutes les options pour les groupes 'client' et 'mysql'.

Mettre les messages d'erreur en français

Dans le groupe mysqld ajoutez :
language = french;

Ou si le fichier de description de langue se trouve à un endroit particulier :
language = /usr/share/mysql/french/

Quelques options relatives à la sécurité

--local-infile[=(0|1)]
: Si cette option vaut 0, les utilisateurs ne peuvent pas lire un file avec LOAD DATA LOCAL INFILE.

--safe-show-database
: Avec cette option, la commande SHOW DATABASES ne retourne que les bases pour lesquelles l'utilisateur courant a des droits. Depuis la version 4.0.2, cette option est abandonnée et ne sert plus à rien (elle est activée par défaut), car désormais, il y a le droit de SHOW DATABASES.

--safe-user-create
: Si cette option est activée, tout utilisateur ne peut créer d'autres utilisateurs avec les droits de GRANT, s'il ne dispose pas des droits d'insertion dans la table mysql.user. Si vous voulez donner un accès à un utilisateur pour qu'il puisse créer des utilisateurs avec les droits dont il dispose, vous pouvez lui donner les droits suivants :

```
mysql> GRANT INSERT(user) ON mysql.user TO 'user'@'hostname';
```

Cela va s'assurer que l'utilisateur ne peut pars modifier une colonne directement, mais qu'il peut exécuter la commande GRANT sur d'autres utilisateurs.

--skip-grant-tables
: Cette option force le serveur à ne pas utiliser les tables de droits. Cette option donne donc tous les droits à tout le monde sur le serveur !

--skip-name-resolve
: Les noms d'hôtes ne sont pas résolus. Toutes les valeurs de la colonne Host dans les tables de droits doivent être des adresses IP, ou bien localhost.

--skip-networking
: Ne pas accepter les connexions TCP/IP venant du réseau. Toutes les connexions au serveur mysqld doivent être faites avec les sockets Unix. Cette option n'existe pas pour les versions antérieures à la 3.23.27, avec les MIT-pthread, car les sockets Unix n'étaient pas supportés par les MIT-pthreads à cette époque.

Scripts serveurs et des utilitaires

Tous les programmes MySQL prennent des options différentes. Toutefois, tous les programmes MySQL disposent de l'option --help qui vous aidera à connaître la liste complète des différentes options. Essayez par exemple mysql --help.
Vous pouvez modifier toutes les valeurs par défaut des programmes en les plaçant dans le fichier de configuration /etc/my.cnf.

Voici la liste des programmes côté serveur de MySQL :

myisamchk
Un utilitaire pour décrire, vérifier, optimiser et réparer les tables MySQL.
Mysqld
Le démon SQL.
mysql_install_db
Crée les tables de droits MySQL, avec les droits par défaut. Il est généralement exécuté une fois, lors de la première installation de MySQL.
safe_mysqld
Est la méthode recommandée pour démarrer un démon mysqld sous Unix. safe_mysqld ajoute des fonctionnalités de sécurité, telles que le redémarrage automatique lorsqu'une erreur survient et l'enregistrement d'informations d'exécution dans un fichier de log. Normalement, vous ne devriez jamais éditer le script safe_mysqld, mais plutôt utiliser les options de safe_mysqld dans la section [safe_mysqld] du fichier 'my.cnf'.

Gestion des droits

Après l'installation de MySQL, un utilisateur root a été créé. Il a le droit de se connecter en local sur n'importe quelle base de données et possède les droits sur toutes les tables. **Le problème est qu'il n'a pas besoin de mot de passe**. Attention ne le confondez pas avec l'utilisateur system root, qui lui, du moins je l'espère pour votre système, possède un mot de passe extrêmement solide.

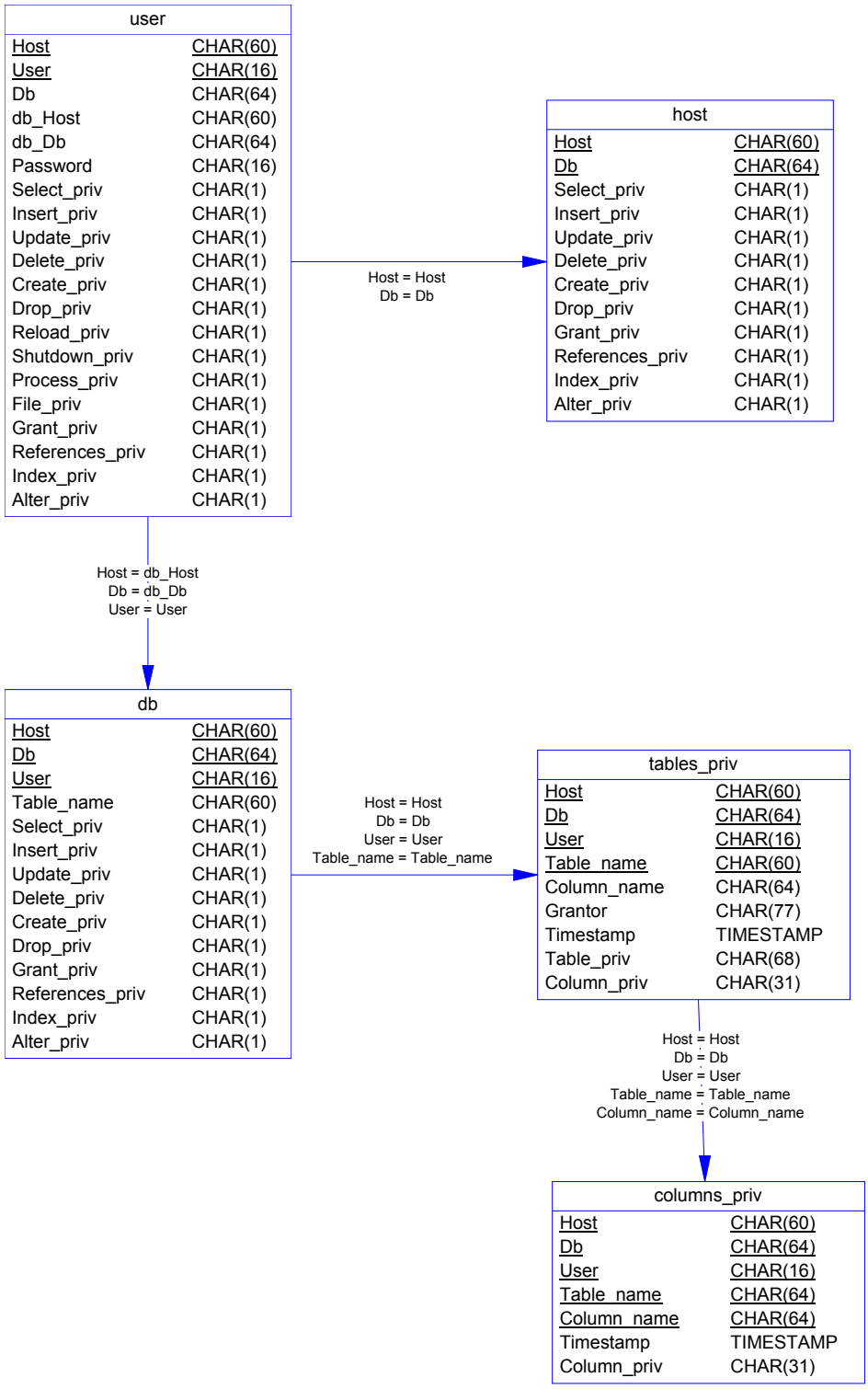
Commençons donc pas le plus urgent : donner un mot de passe à l'utilisateur root :

```
mysql>use mysql
mysql>UPDATE user SET password = password("mot_de_passé_quilestbien") WHERE user="root";
mysql>FLUSH PRIVILEGES
```

Ouf, une bonne chose de faite. Vous remarquerez que nous avons utilisé une base de données appelée mysql et que nous avons modifié le contenu d'un enregistrement à l'aide de la fonction PASSWORD(). De cette manière, les mots de passe seront cryptés à l'aide de l'algorithme MD5.

MySQL authentifie une connexion en prenant deux éléments, le login et l'hôte depuis lequel la connexion est demandée. Pour MySQL le login seul n'est pas déterminant, vous pouvez en effet avoir un user philippe depuis 212.147.125.80 et un autre depuis localhost.

Le système de contrôle des droits sous MySQL est basé sur le modèle suivant :



Lors d'une connexion, le serveur commence à contrôler que le user ait les droits de connexion dans la table user.et ensuite, pour chaque requête, il contrôle les droits d'exécution dans les tables db, tables_priv et columns_priv. C'est dans cette structure que toute la gestion des droits est stockée. Il est donc important de ne laisser que **l'administrateur MySQL avoir les droits sur cette base.**

Attribuer et modifier les droits

Deux commandes vous permettront de gérer les droits **GRANT** et **REVOKE**.

Syntaxe de GRANT :

```
GRANT priv_type [(liste_colonnes)] [, priv_type [(liste_colonnes)] ...]
ON {nom_de_table | * | *.* | nom_base.*}
TO nom_utilisateur [IDENTIFIED BY [PASSWORD] 'password']
[, nom_utilisateur [IDENTIFIED BY 'password'] ...]
[REQUIRE
NONE |
[{SSL| X509}]
[CIPHER cipher [AND]]
[ISSUER issuer [AND]]
[SUBJECT subject]]
[WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR # |
MAX_UPDATES_PER_HOUR # |
MAX_CONNECTIONS_PER_HOUR #]]
```

Syntaxe de REVOKE

```
REVOKE priv_type [(liste_colonnes)] [, priv_type [(liste_colonnes)] ...]
ON {nom_de_table | * | *.* | nom_base.*}
FROM nom_utilisateur [, nom_utilisateur ...]
```

Valeurs possibles

priv_type

ALL [PRIVILEGES]
: Tous les droits sauf WITH GRANT OPTION.

ALTER
: Autorise l'utilisation de ALTER TABLE.

CREATE
: Autorise l'utilisation de CREATE TABLE.

CREATE TEMPORARY TABLES
: Autorise l'utilisation de CREATE TEMPORARY TABLE.

DELETE
: Autorise l'utilisation de DELETE.

DROP
: Autorise l'utilisation de DROP TABLE.

EXECUTE
: Autorise l'utilisateur à exécuter des procédures stockées (pour MySQL 5.0).

FILE
: Autorise l'utilisation de SELECT ... INTO OUTFILE et LOAD DATA INFILE.

INDEX
: Autorise l'utilisation de CREATE INDEX et DROP INDEX.

INSERT
: Autorise l'utilisation de INSERT.

LOCK TABLES
: Autorise l'utilisation de LOCK TABLES sur les tables pour lesquelles l'utilisateur a les droits de SELECT.

PROCESS
Autorise l'utilisation de SHOW FULL PROCESSLIST.
RELOAD
Autorise l'utilisation de FLUSH.
REPLICATION CLIENT
Donne le droit à l'utilisateur de savoir où sont les maîtres et esclaves.
REPLICATION SLAVE
Nécessaire pour les esclaves de réplication (pour lire les historiques binaires du maître).
SELECT
Autorise l'utilisation de SELECT.
SHOW DATABASES
Affiche toutes les bases de données.
SHUTDOWN
Autorise l'utilisation de mysqladmin shutdown.
SUPER
Autorise une connexion unique, même si max connections est atteint, et l'exécution des commandes CHANGE MASTER,KILL thread, mysqladmin debug, PURGE MASTER LOGS et SET GLOBAL.
UPDATE
Autorise l'utilisation de UPDATE.
USAGE
Synonyme de "pas de droits".

Quelques exemples d'attributions de droits

Création d'un user et affectation de tous les droits sur la base nestle :
mysql> **GRANT ALL PRIVILEGES ON nestle.* TO Dupont@localhost IDENTIFIED BY "secret";**

Affectation des droits sur la table alusuisse.employes
mysql> **GRANT ALL PRIVILEGES ON alusuisse.employes TO Dupont@localhost;**

Ajouter les droits de SELECT à un user sur toutes les tables de la base nestle
mysql> **GRANT SELECT ON nestle.* TO Dupont@localhost;**

Supprimer les droits à l'utilisateur Dupont@localhost
Mysql> **REVOKE ALL PRIVILEGES ON nestle.* FROM Dupont@localhost;**

Optimisation des requêtes

Nous ne parlerons ici que de l'optimisation au niveau du code, il est évident que les performances globales de votre serveur dépendent grandement du type de matériel sur lequel il est installé. Les disques, le cpu et la taille de la RAM sont autant d'éléments déterminant la vélocité de votre serveur.

Optimisation des SELECTs

La requête suivante vous donne une première indication sur les performances de votre système :

mysql> SELECT BENCHMARK(1000000,1+1);

Le temps d'exécution de cette requête est de 0.32 sec sur un Pentium II à 400 Mhz de 0.12 secondes pour un Pentium III à 600 Mhz et de 0.07 sur un Pentium IV à 1.6Mhz

Les index

La première chose à faire pour accélérer les requêtes de selection est de créer les bons INDEX. En effet, dans la mesure du possible MySQL va les utiliser pour joindre les tables lors de requête sur plusieurs tables.

Pour déterminer quels sont les index à créer, vous pouvez utiliser la commande EXPLAIN devant un SELECT. Elle vous permettra de voir comment MySQL a lié les tables et surtout de savoir comment MySQL exploite les index.

Pour les jointures complexes, EXPLAIN retourne une ligne d'information pour chaque table utilisée dans la commande SELECT. Les tables sont listées dans l'ordre dans lequel elles seront lues. MySQL résout toutes les jointures avec une seule passe multi-jointure. Cela signifie que MySQL lit une ligne dans la première table, puis recherche les lignes qui correspondent dans la seconde, puis dans la troisième, etc. Lorsque toutes les tables ont été traitées, MySQL attache les colonnes demandées, et il remonte dans les tables jusqu'à la dernière qui avait encore des lignes à traiter. La prochaine ligne est alors traitée de la même façon.

Le résultat de la commande EXPLAIN est constitué des colonnes suivantes :
(Cette partie du support est tirée de la documentation officielle de MySQL)

select_type Type de clause SELECT, qui peut être :
table : La table à laquelle la ligne fait référence.

type : Type de clause SELECT, qui peut-être :

SIMPLE Simple SELECT (sans UNIONs ou sous-requêtes).
PRIMARY SELECT extérieur
UNION Second et autres UNION SELECTs.
DEPENDENT UNION
Second et autres UNION SELECTSs, dépend de la commande extérieure.
SUBSELECT
Premier SELECT de la sous-requête.
DEPENDENT SUBSELECT
Premier SELECT, dépendant de la requête extérieure.
DERIVED Table dérivée SELECT.

type : Le type de jointure. Les différents types de jointures sont les suivants, dans
l'ordre du plus efficace au plus lent :

system La table a une seule ligne (c'est une table système). C'est un cas spécial du type de jointure const.
const La table a au plus une ligne correspondante, qui sera lue dès le début de la requête. Comme il n'y a qu'une seule ligne, les valeurs des colonnes de cette ligne peuvent être considérées comme des constantes pour le reste de l'optimisateur. Les tables const sont très rapides, car elles ne sont lues qu'une fois.

eq_ref Une ligne de cette table sera lue pour chaque combinaison de ligne des tables précédentes. C'est le meilleur type de jointure possible, à l'exception des précédents. Il est utilisé lorsque toutes les parties d'un index sont utilisées par la jointure, et que l'index est UNIQUE ou PRIMARY KEY.

ref Toutes les lignes avec des valeurs d'index correspondantes seront lues dans cette table, pour chaque combinaison des lignes précédentes. ref est utilisé si la jointure n'utilise que le préfixe de gauche de la clé, ou si la clé n'est pas UNIQUE ou PRIMARY KEY (en d'autres termes, si la jointure ne peut pas sélectionner qu'une seule ligne en fonction de la clé). Si la clé qui est utilisée n'identifie que quelques lignes à chaque fois, la jointure est bonne.

range Seules les lignes qui sont dans un intervalle donné seront lues, en utilisant l'index pour sélectionner les lignes. La colonne key indique quel est l'index utilisé. key_len contient la taille de la partie de la clé qui est utilisée. La colonne ref contiendra la valeur NULL pour ce type.

index C'est la même chose que ALL, sauf que seul l'arbre d'index sera lu et scanné. C'est généralement plus rapide que ALL, car le fichier d'index est généralement plus petit que le fichier de données.

ALL Une analyse complète de la table sera faite pour chaque combinaison de lignes issues des premières tables. Ce n'est pas bon si la première table n'est pas une jointure de type const et c'est très mauvais dans les autres cas. Normalement, vous pouvez éviter ces situations de ALL en ajoutant des index basés sur des parties de colonnes.

possible_keys : La colonne possible_keys indique quels index MySQL va pouvoir utiliser pour trouver les lignes dans cette table. Notez que cette colonne est totalement dépendante de l'ordre des tables. Cela signifie que certaines clés de la colonne possible_keys pourraient ne pas être utilisées dans d'autres cas d'ordre de tables. Si cette colonne est vide, il n'y a pas d'index pertinent. Dans ce cas, vous pourrez améliorer les performances en examinant votre clause WHERE pour voir si des colonnes sont susceptibles d'être indexées. Si c'est le cas, créez un index ad hoc et examinez le résultat avec la commande EXPLAIN. Pour connaître tous les index d'une table, utilisez le code SHOW INDEX FROM nom_de_table.

key La colonne key indique l'index que MySQL va décider d'utiliser. Si la clé vaut NULL, aucun index n'a été choisi. Pour forcer MySQL à utiliser un index listé dans la colonne possible_keys, utilisez USE KEY/IGNORE KEY dans votre requête.

key_len La colonne key_len indique la taille de la clé que MySQL a décidé d'utiliser. La taille est NULL si la colonne key vaut NULL. Notez que cela vous indique combien de partie d'une clé multiple MySQL va réellement utiliser.

ref La colonne ref indique quelle colonne ou quelles constantes sont utilisées avec la clé key, pour sélectionner les lignes de la table.

rows La colonne rows indique le nombre de lignes que MySQL estime devoir examiner pour exécuter la requête.

Extra Cette colonne contient des informations additionnelles sur comment MySQL va résoudre la requête. Voici une explication des différentes chaînes que vous pourriez trouver dans cette colonne :

Distinct MySQL ne va pas continuer à chercher d'autres lignes que la ligne courante, après en avoir trouvé une.

Not exists MySQL a été capable d'appliquer une optimisation de type LEFT JOIN sur la requête et ne va pas examiner d'autres lignes de cette table pour la combinaison de lignes précédentes, une fois qu'il a trouvé une ligne qui satisfait le critère de LEFT JOIN.
Voici un exemple de cela :
SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULLSupposons que t2.id est défini comme NOT NULL.
Dans ce cas, MySQL va scanner t1 et rechercher des lignes dans t2 via t1.id. Si MySQL trouve une ligne dans t2, il sait que t2.id ne peut pas être NULL, et il ne va pas scanner le reste des lignes de t2 qui ont le même id. En d'autres termes, pour chaque ligne de t1, MySQL n'a besoin que de faire une recherche dans t2, indépendamment du nombre de lignes qui sont trouvées dans t2.

range checked for each record (index map: #) MySQL n'a pas trouvé d'index satisfaisant à utiliser. Il va, à la place, pour chaque combinaison de lignes des tables précédentes, faire une vérification de quel index utiliser (si il en existe), et utiliser cet index pour continuer la recherche. Ce n'est pas très rapide, mais c'est plus rapide que de faire une recherche sans aucun index.

Using filesort MySQL va avoir besoin d'un autre passage pour lire les lignes dans l'ordre. Le tri est fait en passant en revue toutes les lignes, suivant le type de jointure est stocker la clé de tri et le pointeur de la ligne pour chaque ligne qui satisfont la clause WHERE. Alors, les clés sont triées. Finalement, les lignes sont triées dans l'ordre.

Using index Les informations de la colonne sont lues de la table, en utilisant uniquement les informations contenues dans l'index, sans avoir à faire d'autres lectures. Cela peut arriver lorsque toutes les colonnes utilisées dans une table font partie de l'index.

Using temporary Pour résoudre la requête, MySQL va avoir besoin de créer une table temporaire pour contenir le résultat. C'est typiquement ce qui arrive si vous utilisez une clause ORDER BY sur une colonne différente de celles qui font partie de GROUP BY.

Using where Une clause WHERE sera utilisée pour restreindre les lignes qui seront trouvées dans la table suivante, ou envoyées au client. Si vous n'avez pas cette information, et que la table est de type ALL ou index, vous avez un problème dans votre requête (si vous ne vous attendiez pas à tester toutes les lignes de la table). Si vous voulez rendre vos requêtes aussi rapides que possible, vous devriez examiner les lignes qui utilisent Using filesort et Using temporary.

(Fin de la partie importée de la documentation officielle)

Optimisation de la clause WHERE

Quelques "trucs" :

- Eviter les parenthèses inutiles. Elles facilitent la lecture des conditions, mais ralentissent le serveur.
- La fonction COUNT(*) sans la clause WHERE lit directement les valeurs dans les infos de la table
- HAVING à la place de la clause WHERE. Si vous utilisez GROUP BY ou les fonctions de groupe COUNT(), MIN(), …
  Exemple :
  SELECT AVG(CONDITIONNEMENT.PRIX) AS MOYENNE FROM CONDITIONNEMENTS INNER JOIN VINS ON CONDITIONNEMENT.ID_VIN = VINS.ID_VIN GROUP BY VINS.TYPE **HAVING** MOYENNE>12
- Les requêtes "constantes" sont lues en premier. Une requête constante est un résultat vide ou ne contenant qu'une ligne
  Exemple :
  SELECT * FROM MA_TABLE **WHERE PRIMARY_KEY=23**;
- Vous pouvez aussi améliorer vos requêtes si elles sont triées sur un index et utilisées avec la clause LIMIT.
  Exemple :
  SELECT * FROM MA_TABLE **ORDER BY KEY_1 DESC LIMIT 20**;
- La clause DISTINCT est convertie en GROUP BY par MySQL, ainsi d'ailleurs que les INNER JOIN sont convertis en WHERE

Optimisation des INSERTs

Le coût en temps de ce type de requête est dans l'ordre d'importance :

- La connexion
- Envoi de la requête
- Analyse de la requête
- Insertion de la ligne

Par conséquent, si vous devez faire des insertions multiples n'utilisez pas :
INSERT INTO CLIENST VALUES("Pierre","Pott","021 907 58 45");
INSERT INTO CLIENST VALUES("Paul","Durand","021 902 58 65");
INSERT INTO CLIENST VALUES("Marc","Lepointu","021 807 58 48");

Mais de préférence :
INSERT INTO CLIENST VALUES("Pierre","Pott","021 907 58 45"),("Paul","Durand","021 902 58 65"),("Marc","Lepointu","021 807 58 48");

Vous pouvez également utiliser la clause DELAYED (voir les détails de la commande dans l'annexe)

En verrouillant les tables avant les INSERTs, vous serez également plus performant.
Exemple :
LOCK TABLES ma_table WRITE;
INSERT INTO CLIENST VALUES("Pierre","Pott","021 907 58 45");
INSERT INTO CLIENST VALUES("Paul","Durand","021 902 58 65");
INSERT INTO CLIENST VALUES("Marc","Lepointu","021 807 58 48");
ou de preference INSERT INTO CLIENST VALUES("Pierre","Pott","021 907 58 45"),("Paul","Durand","021 902 58 65"),("Marc","Lepointu","021 807 58 48");
et
UNLOCK TABLES;

Dans tous les cas de figure, la commande LOAD DATA INFILE reste de loin la plus performante pour de grandes insertions de datas

Autres possibilités d'optimisation

- Utilisez de préférence les connexions persistantes (Voir le cours PHP)
- Contrôlez que vos requêtes utilisent des index
- Utilisez régulièrement OPTIMIZE TABLE sur les tables où les données sont souvent ajoutées ou effacées
- Dans la mesure du possible, mettez des valeurs par défaut. Vous y gagnerez en insertion.

Optimiser dès la conception

- Mysql vous fournit une gamme étendue de type de données, utilisez-les au mieux. Evitez de donner des valeurs trop grandes. Par exemple, utilisez MEDIUMINT au lieu de INT. Sur l'ensemble de la base, les effets se feront sentir.
- Prenez également l'habitude, bien évidemment dans la mesure du possible, de déclarer vos champs NOT NULL.
- Le type CHAR est plus rapide que VARCHAR, mais coûte en espace disque

Maintenance et exploitation de la base

Lorsqu'une base de données est souvent sollicitée en ajout et en suppression, il se produit un phénomène de fragmentation comparable à celui d'un disque dur. Il faut donc régulièrement "optimiser" les tables. Cette opération se fait à l'aide de la commande OPTIMIZE TABLE table1.[table2,table3,..]

PHPMyAdmin : Interface Web de gestion de MySQL

Annexes

Correction des exercices pratiques

Passons à la pratique I :

Création de la nouvelle base de données

```
mysql>  create database cours;
mysql>  use cours
```

Création de la table films

```
mysql>  CREATE TABLE films(
        id_film INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
        realisateur VARCHAR(200), titre VARCHAR(200));
```

Importation des datas

```
LOAD DATA INFILE "films.txt" INTO TABLE films FIELDS TERMINATED BY ";"
ENCLOSED BY "\"" (realisateur,titre);
```

Requêtes select

```
SELECT * FROM films;
SELECT * FROM films ORDER BY realisateur;
SELECT * FORM films ORDER BY titre DESC;
```

SELECT titre FROM films WHERE realisateur = "Jean-Luc Godard";

Les commandes en détail

La commande CREATE TABLE

La syntaxe :

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nom_de_table
[(**definition_de_create**,...)]
[**options_de_table**] [select_statement]

definition_de_create :
nom_de_colonne **type** [NOT NULL | NULL] [DEFAULT valeur_par_defaut]
[AUTO_INCREMENT]
[PRIMARY KEY] [definition_de_reference]
ou PRIMARY KEY (index_col_name,...)
ou KEY [nom_index] (index_col_name,...)
ou INDEX [nom_index] (index_col_name,...)
ou UNIQUE [INDEX] [nom_index] (index_col_name,...)
ou FULLTEXT [INDEX] [nom_index] (index_col_name,...)
ou [CONSTRAINT symbol] FOREIGN KEY [nom_index] (index_col_name,...)
[reference_definition]
ou CHECK (expr)

type	**Valeurs possibles**	
	Signé	**Non signé**
TINYINT[(longueur)] [UNSIGNED] [ZEROFILL]	-128 à 127	0 – 255
ou SMALLINT[(longueur)] [UNSIGNED] [ZEROFILL]	- 32768 à 32767	0- 65 535
ou MEDIUMINT[(longueur)] [UNSIGNED] [ZEROFILL]	-8388608 à 8388607	0 à 16777215
ou INT[(longueur)] [UNSIGNED] [ZEROFILL]	-2147483648 à 2147483647	0 à 4294967295
ou INTEGER[(longueur)] [UNSIGNED] [ZEROFILL]	Synonyme de INT	
ou BIGINT[(longueur)] [UNSIGNED] [ZEROFILL]	-9223372036854775808 et 9223372036854775807	0 et 18446744073709551615
ou REAL[(longueur,dµecimales)] [UNSIGNED] [ZEROFILL]	-1.7976931348623157E+308 à -2.2250738585072014E-308	0 à 1.7976931348623157E+308
ou DOUBLE[(longueur,dµecimales)] [UNSIGNED] [ZEROFILL]	Synonyme de REAL	
ou FLOAT[(longueur,dµecimales)] [UNSIGNED] [ZEROFILL]	-3.402823466E+38 à -1.175494351E-38	0 à de 1.175494351E-38
ou DECIMAL(longueur,dµecimales) [UNSIGNED] [ZEROFILL]	FLOAT signé	

ou NUMERIC(longueur,dµecimales) [UNSIGNED] [ZEROFILL]	Synonyme de DECIMAL	
ou CHAR(longueur) [BINARY]	De 0 à 255 caractères, la valeur est complétée par des espaces	
ou VARCHAR(longueur) [BINARY]	De 0 à 255, la valeur est dynamique, mais ne peut excéder 255 caractères	
ou DATE	De '1000-01-01' à '9999-12-31'	
ou TIME	De '-838:59:59' à '838:59:59'.	
ou TIMESTAMP	De '1970-01-01 00:00:00' à quelque part durant l'année 2037	
ou DATETIME	De '1000-01-01 00:00:00' à '9999-12-31 23:59:59'	
ou TINYBLOB	Une colonne TINYBLOB où TINYTEXT a une longueur maximale de 255 (2^8 - 1) caractères.	
ou BLOB	Une colonne BLOB où TEXT a une longueur maximale de 65535 (2^16 - 1) caractères.	
ou MEDIUMBLOB	Une colonne MEDIUMBLOB où MEDIUMTEXT a une longueur maximale de 16777215 (2^24 -1) caractères.	
ou LONGBLOB	Une colonne LONGBLOB où LONGTEXT a une longueur maximale de 4294967295 (2^32 - 1) caractères.	
ou TINYTEXT	Une colonne BLOB où TINYTEXT a une longueur maximale de 255 (2^8 - 1) caractères	
ou TEXT	Une colonne TINYBLOB où TEXT a une longueur maximale de 65535 (2^16 - 1) caractères.	
ou MEDIUMTEXT	Une colonne MEDIUMBLOB où MEDIUMTEXT a une longueur maximale de 16777215 (2^24 -1) caractères.	
ou LONGTEXT	Une colonne LONGBLOB où LONGTEXT a une longueur maximale de 4294967295 (2^32 - 1) caractères.	
ou ENUM(valeur1,valeur2,valeur3,...)	Liste de valeurs possibles, maximum 65535	
ou SET(valeur1,valeur2,valeur3,...)	Liste de valeurs allant de 0 à toutes les valeurs mentionnées avec un maximum de 60	

options_de_table
TYPE = {BDB | HEAP | ISAM | InnoDB | MERGE | MRG_MYISAM | MYISAM }
BDB : Le format très utilisé dans le monde UNIX (Berkley DataBase)
HEAP : Les données de ces tables ne sont stockées qu'en mémoire.
ISAM : Format standardisé. Utilisé par défaut sur les premières versions de MySQL. Utilisé également par d'autres produits, notamment MS-Acess 2.0
InnoDB : Format récent des tables MySQL. Permet la gestion de plus grandes tables, ainsi que les transactions et l'intégrité référentielle.
MERGE : Un ensemble de tables MyISAM utilisées comme une seule et même table.
MRG_MYISQM : Un synonyme pour MERGE les tables.
MYISAM : La valeur par défaut sur les installations standards

Quelques explications sur les options :
NOT NULL où **NULL** spécifie si une valeur null peut être stockée dans le champ
DEFAULT valeur_par_defaut spécifie une valeur à entrer lorsqu'aucune valeur n'est spécifiée
AUTO_INCREMENT, concerne uniquement les valeurs numériques entières, la valeur sera incrémentée de 1 unité à chaque insertion d'un nouvel enregistrement. Il ne peut y

avoir qu'une seule valeur AUTO_INCREMENT par table. On l'utilise généralement comme clé primaire.
PRIMARY KEY définit la clé primaire.
KEY définit un index multiple, les index permettent d'accélérer les recherches
UNIQUE définit un index unique, il ne peux pas y avoir deux fois la même valeur dans le champ
FULLTEXT permet de définir les champs qui seront utilisés pour une recherche full text. Une recherche full texte effectue, selon un modèle, une recherche multi-champs.
[CONSTRAINT symbol] FOREIGN KEY permet de spécifier des contraintes d'intégrité sous MySQL fonctionne uniquement si vous utilisez le format de table InnoDB

La commande LOAD DATA

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'nom_de_fichier.txt'
[REPLACE | IGNORE]
INTO TABLE nom_de_table
[FIELDS
[TERMINATED BY '\t']
[[OPTIONALLY] ENCLOSED BY '']
[ESCAPED BY '\\' ]
]
[LINES TERMINATED BY '\n']
[IGNORE nombre LINES]
[(nom_de_colonne,...)]
```

La commande LOAD DATA INFILE lit les lignes dans un fichier texte et les insère dans une table

Explication sur les différents éléments de la commande
LOW_PRIORITY
MySQL attend qu'aucun client ne le sollicite pour exécuter la requête, évite de surcharger le serveur

CONCURRENT
Les clients pourront accéder à la table durant l'exécution de la commande LOAD DATA. Cette option ralentit un peu les performances de LOAD DATA, même si aucune autre requête n'interroge la table

LOCAL
Le mot clé LOCAL permet d'indiquer que le fichier se trouve sur la machine cliente.

REPLACE
Permet de spécifier la méthode utilisée pour insérer les datas lors de doublon, avec REPLACE les nouvelles données prendront la place des anciennes

IGNORE
Permet de spécifier la méthode utilisée pour insérer les datas lors de doublon, avec IGNORE les nouvelles données ne seront pas prises en compte

FIELDS
Lorsque cette option est spécifiée, vous devez renseigner une des options suivantes : TERMINATED BY ou ENCLOSED

TERMINATED BY
Spécifie le séparateur de champ

ENCLOSED
Spécifie le caractère qui entoure chaque champ

LINES TERMINATED BY
Permet de spécifier le séparateur d'enregistrement

IGNORE nombre LINES
Permet de ne pas prendre en compte "nombre" de lignes

Replication de bases

Vous aurez besoin de deux serveurs MySQL, le maître et un esclave sur lequel s'effectuera la duplication.

Configuration du maître

Créez un compte SQL ayant les droits nécessaires. Exemple :

GRANT FILE ON *.* TO BACKUPC@<IP_SLAVE> IDENTIFIED BY 'SECRET';

Dans le fichier /etc/my.cnf, modifiez la section [mysqld] :

log-bin
server-id=1

Ensuite, redémarrez le serveur

Configuration de l'esclave

Modifier /etc/my.cnf, en ajoutant dans la section [mysqld] :

master-host=<ip_master>
master-user=backup
master-password=secret
master-port=3306
server-id=2

A l'aide de la commande mysqldump répliquez manuellement votre maître sur l'esclave.

Restartez votre serveur.

Si tout s'est bien passé, vous devriez voir dans le fichier /var/mysql/<host>.err des lignes comme celles-ci :

020224 4:38:15 Slave: connected to master 'backup@<master.host>:3306', replication started in log 'master-bin.001' at position 73

Si vous souhaitez ne répliquer qu'une seule base et que vous en possédez bien d'autres sur le serveur master, ajoutez au my.cnf master :

binlog-do-db=<la base que vous souhaitez répliquer>

et dans le my.cnf slave :

replicate-do-db=<la base que vous souhaitez répliquer>

(Reprise résumée de la documentation officielle)

Référence du langage MySQL

Structure du langage

Les chaînes

Une chaîne est une séquence de caractères, entourée de guillemets simples (‘’) ou doubles (‘"’) (simple seulement si vous êtes en mode ANSI).

Exemples: 'une chaîne' "une autre chaîne"

A l’intérieur d’une chaîne, certaines séquences de caractères ont une signification spéciale. Chacune d’elle commence par un anti-slash (‘\’), connu comme le *caractère d’échappement*.

Par exemple pour utiliser un " en tant que composant de la chaîne "Ma chaîne possède un **\"**". On protège le " à l'aide d'un anti-slash

Les nombres

Les entiers sont représentés comme une séquence de chiffres. Les décimaux utilisent ‘.’ comme séparateur. Tous les types de nombres peuvent être précédés d’un ‘-’ pour indiquer une valeur négative.

Un entier peut être utilisé dans un contexte décimal, il sera interprété comme le nombre décimal équivalent.

Valeurs hexadécimales

MySQL supporte les valeurs hexadécimales. Dans un contexte numérique, elles agissent comme des entiers (précision 64-bit). Dans un contexte de chaîne, elles agissent comme une chaîne binaire où chaque paire de caractères hexadécimaux est convertie en caractères :

Exemple : 0xFF pour 255 en hexadécimal

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 NULL peut être représenté par \N lors de la récupération ou écriture avec des fichiers (LOAD DATA INFILE, SELECT ... INTO OUTFILE).

Variables utilisateur

MySQL supporte les variables utilisateur spécifiques à la connexion avec la syntaxe @variablename. Les variables n’ont pas besoin d’être initialisées. Elles sont à NULL par défaut et peuvent contenir un entier, un réel ou une chaîne. Toutes les variables d’un thread sont automatiquement libérées lorsque le thread se termine. Vous pouvez déclarer une variable avec la syntaxe de **SET** :

SET @variable= { expression entier | expression reel | expression chaîne } [,@variable= ...].

Vous pouvez aussi assigner une valeur à une variable avec d'autres commandes que SET. Par contre, dans ce cas-là, l'opérateur d'assignation est := au lieu de =, parce que = est réservé aux comparaisons dans les requêtes autres que SET :

SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;

Variables système

A partir de la version 4.0.3 de MySQL, fourni un meilleur accès à beaucoup de variables système et variables de connexion. On peut changer la plupart d'entre elles, sans avoir à stopper le serveur.

Syntaxe des commentaires

Le serveur MySQL supporte les commentaires de la forme
jusqu'à la fin de la ligne
jusqu'à la fin de la ligne et
/* dans la ligne ou sur plusieurs lignes */ :
Exemple :

```
mysql> SELECT 1+1; # Ce commentaire se continue jusqu'à la fin de la ligne
mysql> SELECT 1+1; -- Ce commentaire se continue jusqu'à la fin de la ligne
mysql> SELECT 1 /* Ceci est un commentaire dans la ligne */ + 1;
mysql> SELECT 1+
/*
Ceci est un commentaire
sur plusieurs lignes
*/
1;
```

Les mots réservés

Evitez d'utiliser les mots suivants, pour désigner des objets MySQL :

ADD ALL ALTER
ANALYZE AND AS
ASC ASENSITIVE AUTO_INCREMENT
BDB BEFORE BERKELEYDB
BETWEEN BIGINT BINARY
BLOB BOTH BTREE
BY CALL CASCADE
CASE CHANGE CHAR
CHARACTER CHECK COLLATE
COLUMN COLUMNS CONNECTION
CONSTRAINT CREATE CROSS
CURRENT_DATE CURRENT_TIME CURRENT_TIMESTAMP
CURSOR DATABASE DATABASES
DAY_HOUR DAY_MINUTE DAY_SECOND
DEC DECIMAL DECLARE
DEFAULT DELAYED DELETE
DESC DESCRIBE DISTINCT
DISTINCTROW DIV DOUBLE
DROP ELSE ELSEIF
ENCLOSED ERRORS ESCAPED
EXISTS EXPLAIN FALSE
FIELDS FLOAT FOR
FORCE FOREIGN FROM
FULLTEXT GRANT GROUP

HASH HAVING HIGH_PRIORITY
HOUR_MINUTE HOUR_SECOND IF
IGNORE IN INDEX
INFILE INNER INNODB
INOUT INSENSITIVE INSERT
INT INTEGER INTERVAL
INTO IS ITERATE
JOIN KEY KEYS
KILL LEADING LEAVE
LEFT LIKE LIMIT
LINES LOAD LOCALTIME
LOCALTIMESTAMP LOCK LONG
LONGBLOB LONGTEXT LOOP
LOW_PRIORITY MASTER_SERVER_ID MATCH
MEDIUMBLOB MEDIUMINT MEDIUMTEXT
MIDDLEINT MINUTE_SECOND MOD
MRG_MYISAM NATURAL NOT
NULL NUMERIC ON
OPTIMIZE OPTION OPTIONALLY
OR ORDER OUT
OUTER OUTFILE PRECISION
PRIMARY PRIVILEGES PROCEDURE
PURGE READ REAL
REFERENCES REGEXP RENAME
REPEAT REPLACE REQUIRE
RESTRICT RETURN RETURNS
REVOKE RIGHT RLIKE
RTREE SELECT SENSITIVE
SEPARATOR SET SHOW
SMALLINT SOME SONAME
SPATIAL SPECIFIC SQL_BIG_RESULT
SQL_CALC_FOUND_ROWS SQL_SMALL_RESULT SSL
STARTING STRAIGHT_JOIN STRIPED
TABLE TABLES TERMINATED
THEN TINYBLOB TINYINT
TINYTEXT TO TRAILING
TRUE TYPES UNION
UNIQUE UNLOCK UNSIGNED
UNTIL UPDATE USAGE
USE USER_RESOURCES USING
VALUES VARBINARY VARCHAR
VARCHARACTER VARYING WARNINGS
WHEN WHERE WHILE
WITH WRITE XOR
YEAR_MONTH ZEROFILL

Les symboles suivants (issus de la table ci-dessus) sont interdits par ANSI SQL, mais permis par MySQL en tant que noms de colonnes ou de tables. Cela est dû au fait que ces noms sont très courants, et de nombreux programmeurs les ont déjà utilisés.

ACTION
BIT
DATE

ENUM
NO
TEXT
TIME
TIMESTAMP
Types de colonnes
MySQL supporte un grand nombre de types de colonnes, qui peuvent être rassemblés en trois groupes : les nombres, les dates et les chaînes de caractères. Cette section présente les types disponibles et leurs tailles de stockage

Les codes suivants sont utilisés dans les descriptions :
M Indique la taille maximale d'affichage. Le maximum légal est 255.
D s'applique aux nombres à virgule flottante, et indique le nombre de décimales qui suivent la virgule. Le nombre maximum est de 30, mais ne doit pas être plus grand que M-2.
Les crochets ('[' et ']') indiquent les spécifications optionnelles.

TINYINT[(M)] [UNSIGNED] [ZEROFILL]
Un très petit entier. S'il est signé, sa valeur varie entre -128 et 127, sinon elle varie de 0 à 255.
BIT
BOOL Ce sont des synonymes de TINYINT(1).
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]
Un petit entier. S'il est signé, sa valeur varie entre -32768 et 32767, sinon elle varie de 0 à 65535.
MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]
Un entier de taille moyenne. S'il est signé, sa valeur varie entre -8388608 et 8388607, sinon elle varie entre 0 et 16777215.
INT[(M)] [UNSIGNED] [ZEROFILL]
Un entier. S'il est signé, sa valeur varie entre -2147483648 et 2147483647, sinon elle varie entre 0 et 4294967295.
INTEGER[(M)] [UNSIGNED] [ZEROFILL]
C'est un synonyme de INT.

Un grand entier. S'il est signé, sa valeur varie entre -9223372036854775808 et 9223372036854775807, sinon elle varie entre 0 et 18446744073709551615.
FLOAT(précision) [UNSIGNED] [ZEROFILL]
Un nombre à virgule flottante. précision<=24 pour un nombre à virgule flottante de précision simple, entre 25 et 53 pour une précision double. Ces types correspondent aux types FLOAT et DOUBLE décrits ci-dessus. FLOAT(X) a le même intervalle de validité que FLOAT et DOUBLE, mais la taille d'affichage et le nombre de décimale sont indéfinis.
FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]
Un petit nombre à virgule flottante (précision simple). L'intervalle de validité va de -3.402823466E+38 à-1.175494351E-38, 0 et de 1.175494351E-38 à 3.402823466E+38. M représente la taille d'affichage et D est le nombre de décimales.
DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]
Un nombre à virgule flottante (précision double). L'intervalle de validité va de -1.7976931348623157E+308 à-2.2250738585072014E-308, 0 et de 2.2250738585072014E-308 à1.7976931348623157E+308. M représente la taille d'affichage et D est le nombre de décimales.
REAL[(M,D)] [UNSIGNED] [ZEROFILL]
C'est un synonyme de DOUBLE.

DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]
Un nombre à virgule flottante. Il doit être signé. Ce type se comporte comme une colonne de type CHAR : la valeur est stockée comme une chaîne, chaque caractère représentant un chiffre de la valeur. La virgule et le signe '-' des nombres négatifs ne sont pas comptés dans l'option M (mais de l'espace de stockage est réservé pour eux). Si D vaut 0, les valeurs n'auront pas de valeur décimale. L'intervalle de validité des valeurs DECIMAL est le même que DOUBLE, mais il peut être limité par les valeurs choisies pour M et D. Si D est omis, la valeur par défaut est 0. Si M est omis, la valeur par défaut est 10.
DEC[(M[,D])] [UNSIGNED] [ZEROFILL]
NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]
Ce sont des synonymes de DECIMAL.
DATE
Une date. L'intervalle de validité va de '1000-01-01' à'9999-12-31'. MySQL attache les valeurs DATE au format 'AAAA-MM-JJ' (année-mois-jour), mais vous pouvez assigner des valeurs aux colonnes DATE en utilisant différents formats numériques ou de chaînes de caractères.
DATETIME
Une combinaison de date et d'heure. L'intervalle de validité va de '1000-01-01 00:00:00' à'9999-12-31 23:59:59'. MySQL attache les valeurs DATETIME au format 'AAAA-MM-JJ HH:MM:SS' (année-mois-jour heure:minutes:secondes), mais vous pouvez assigner des valeurs aux colonnes DATETIME en utilisant différents formats numériques ou de chaînes de caractères.
TIMESTAMP[(M)]
Un timestamp. L'intervalle de validité va de '1970-01-01 00:00:00' à quelque part durant l'année 2037. MySQL attache les valeurs de type TIMESTAMP au format AAAAMMJJHHMMSS, AAMMJJHHMMSS, AAAAMMJJ, ou AAMMJJ, suivant que le paramètre M vaut 14 (ou omis), 12, 8, ou 6, mais vous pouvez assigner des valeurs aux colonnes TIMESTAMP sous forme de nombre ou de chaînes. Une colonne de type TIMESTAMP est pratique pour enregistrer une date, lors d'une commande INSERT ou UPDATE, car elle est automatiquement mise à la date et l'heure du moment de la commande, si vous ne fournissez pas de valeur vous-même. Vous pouvez aussi lui donner la date et l'heure courante en lui assignant la valeur NULL.
TIME
Une heure. L'intervalle de validité va de '-838:59:59' à'838:59:59'.MySQL attache les colonnes de type TIME au format 'HH:MM:SS', mais vous pouvez assigner une valeur de type TIME en lui passant des chaînes ou des entiers.
YEAR[(2|4)]
Une année au format 2 ou 4 chiffres (par défaut, c'est 4 chiffres). L'intervalle de validité va de 1901 à 2155, 0000 pour le format à 4 chiffres, et de 1970 à 2069 pour le format à deux chiffres. MySQL attache les valeurs de type YEAR au format AAAA, mais vous pouvez leur assigner des chaînes ou des nombres.
[NATIONAL] CHAR(M) [BINARY]
Une chaîne de caractères de taille fixe, qui est toujours complétée à droite, par des espaces, lors du stockage. Le paramètre M peut valoir de 0 à255 caractères. Les espaces terminaux sont supprimés lorsque la chaîne est lue dans la base. Les valeurs de type CHAR sont triées et comparées sans tenir compte de la casse et en utilisant le jeu de caractères par défaut. Toutefois, vous pouvez utiliser l'opérateur BINARY pour effectuer des recherches sensibles à la casse. NATIONAL CHAR (forme courte : NCHAR) est la dénomination ANSI SQL pour les colonnes de type CHAR qui doivent utiliser le jeu de caractères par défaut. C'est la configuration par défaut de MySQL.
[NATIONAL] VARCHAR(M) [BINARY]
Une chaîne de caractères de longueur variable. NOTE : les espaces terminaux sont

supprimés lors du stockage des valeurs. L'intervalle de taille de M va de 1 à255 caractères. Les valeurs de type VARCHAR sont triées et comparées sans tenir compte de la casse et en utilisant le jeu de caractères par défaut. Toutefois, vous pouvez utiliser l'opérateur BINARY pour effectuer des recherches sensibles à la casse

TINYBLOB ou TINYTEXT
Une colonne BLOB ou TEXT avec une longueur maximale de 255 (2^8 - 1) caractères.

BLOB ou TEXT
Une colonne BLOB ou TEXT avec une longueur maximale de 65535 (2^16 - 1) caractères.

MEDIUMBLOB ou MEDIUMTEXT
Une colonne BLOB ou TEXT avec une longueur maximale de 16777215 (2^24 -1) caractères.

LONGBLOB ou LONGTEXT
Une colonne BLOB ou TEXT avec une longueur maximale de 4294967295 (2^32 - 1) caractères.

ENUM('valeur1','valeur2',...)
Une énumération. Une chaîne de caractères qui ne peut prendre qu'une valeur, issue d'une liste de valeurs 'valeur1', 'valeur2', ..., NULL ou la valeur spéciale d'erreur "". Un ENUM peut avoir un maximum de 65535 valeurs distinctes.

SET('valeur1','valeur2',...)
Un ensemble. Une chaîne de caractères qui a zéro ou plusieurs valeurs issues d'une liste : 'valeur1', 'valeur2', ... Un SET peut avoir au maximum 60 éléments.

Fonctions à utiliser dans les clauses SELECT et WHERE

Une sélection ou une clause WHERE dans une requête SQL peut se former d'expressions utilisant les fonctions décrites ci-dessous.

Opérateurs de comparaison et fonctions

> Plus grand
< Plus petit
>= Plus grand ou égale
<= Plus petit ou égale
<>Différent
!= Différent :
<=> Egalité sure avec NULL
IS NULL test si valeur null

expression BETWEEN min AND max Si expression est supérieure ou égale à min et expression est inférieure ou égale à max, BETWEEN retourne 1, sinon 0. Ceci est équivalent à l'expression (min <= expression AND expression <= max) si tous les arguments sont du même type. cal Reference for Version 4.1.1-alpha

expr IN (valeur,...) Retourne 1 si expr est l'une des valeurs dans la liste IN, sinon retourne 0.

expr NOT IN (value,...) Même chose que NOT (expr IN (valeur,...)).

ISNULL(expr) Si expr est NULL, ISNULL() retourne 1, sinon il retourne 0:

COALESCE(list) Retourne le premier élément non-NULL de la liste :

INTERVAL(N,N1,N2,N3,...) Retourne 0 si N < N1, 1 si N < N2 etc... Tous les arguments sont traités en tant qu'entiers. Il est requis que N1 < N2 < N3 < ... < Nn pour que cette fonction fonctionne correctement.

NOT ! NOT (NON) logique. Evalue à1 si l'opérande est 0, à0 si l'opérande est non nulle, et NOT NULL retourne NULL.

AND && AND (ET) logique. Evalue à1 si toutes les opérandes sont différentes de zéro et de NULL, à0 si l'une des opérandes est 0, dans les autres cas, NULL est retourné.

OR || OR (OU inclusif) logique. Evalue à1 si aucune opérande n'est nulle, à NULL si l'une des opérandes est NULL, sinon 0 est retourné.

XOR XOR (OU exclusif) logique. Retourne NULL si l'une des opérandes est NULL. Pour les opérandes non-NULL, évalue à1 si un nombre pair d'opérandes est non-null, sinon 0 est retourné.
IFNULL(expr1,expr2) Si l'argument expr1 n'est pas NULL, la fonction IFNULL() retournera l'argument expr1, sinon elle retournera l'argument expr2. La fonction IFNULL() retourne une valeur numérique ou une chaîne de caractères, suivant le contexte d'utilisation.
NULLIF(expr1,expr2) Si l'expression expr1 = expr2 est vrai, la fonction retourne NULL sinon elle retourne expr1.
IF(expr1,expr2,expr3) Si l'argument expr1 vaut TRUE (expr1 <> 0 et expr1 <> NULL) alors la fonction IF() retourne l'argument expr2, sinon, elle retourne l'argument expr3. La fonction IF() retourne une valeur numérique ou une chaîne de caractères, suivant le contexte d'utilisation :
CASE valeur **WHEN** [compare-value] **THEN** résultat [WHEN [compare-value] THEN resultat ...] [ELSE résultat] END
CASE WHEN [condition] THEN résultat [WHEN [condition] THEN résultat …] [ELSE résultat] END La première version retourne résultat si valeur=compare-value. La seconde version retourne le résultat de la première condition qui se réalise. Si aucune des conditions n'est réalisée, alors le résultat de la clause ELSE est retourné. Si il n'y a pas de clause ELSE, alors NULL est retourné
ASCII(str) Retourne le code ASCII du premier caractère de la chaîne de caractères str. Retourne 0 si la chaîne de caractères str est vide. Retourne NULL si la chaîne de caractères str est NULL :
ORD(str) Si le premier caractère de la chaîne str est un caractère multi-octets, la fonction retourne le code de ce caractère, calculé à partir du code ASCII retourné par cette formule : ((first byte ASCII code)*256+(second byte ASCII code))[*256+third byte ASCII code...]. Si le premier caractère n'est pas un caractère multi-octet, la fonction retournera la même valeur que la fonction ASCII() :
CONV(N,from_base,to_base) Convertit des nombres entre différentes bases. Retourne une chaîne de caractères représentant le nombre N, convertit de la base from_base vers la base to_base. La fonction retourne NULL si un des arguments est NULL. L'argument N est interprété comme un entier, mais peut être spécifié comme un entier ou une chaîne de caractères. Le minimum pour la base est 2 et son maximum est 36. Si to_base est un nombre négatif, N sera considéré comme un nombre signé. CONV travaille avec une précision de 64 bits :
BIN(N) Retourne une chaîne de caractères représentant la valeur binaire de l'argument N, où l'argument N est un nombre de type BIGINT. Cette fonction est un équivalent de CONV(N,10,2). Retourne NULL si l'argument N est NULL
OCT(N) Retourne une chaîne de caractères représentant la valeur octal de l'argument N, où l'argument N est un nombre de type BIGINT. Cette fonction est un équivalent de CONV(N,10,8). Retourne NULL si l'argument N est NULL
HEX(N_or_S) Si l'argument N OR S est un nombre, cette fonction retournera une chaîne de caractères représentant la valeur hexadécimale de l'argument N, où l'argument N est de type BIGINT. Cette fonction est un équivalent de CONV(N,10,16).
CHAR(N,...) La fonction CHAR() interprète les arguments comme des entiers et retourne une chaîne de caractères, constituée des caractères, identifiés par leur code ASCII. Les valeurs NULL sont ignorées
CONCAT(str1,str2,...) Retourne une chaîne représentant la concaténation des arguments.
CONCAT_WS(separator, str1, str2,...) La fonction CONCAT_WS() signifie CONCAT With Separator, c'est-à-dire "concaténation avec séparateur. Le premier argument est le séparateur utilisé pour le reste des arguments. Le séparateur peut être une chaîne de caractères, tout comme le reste des arguments. Si le séparateur est NULL, le résultat

sera NULL. Cette fonction ignorera tous les arguments de valeur NULL et vides, hormis le séparateur. Le séparateur sera ajouté entre tous les arguments à concaténer
LENGTH(str) Retourne le nombre de bits de la chaîne de caractères str
LOCATE(substr,str) POSITION(substr IN str) Retourne la position de la première occurrence de la chaîne substr dans la chaîne de caractères str. Retourne 0 si substr ne se trouve pas dans la chaîne de caractères str:
LOCATE(substr,str,pos) Retourne la position de la première occurrence de la chaîne substr dans la chaîne de caractères str, à partir de la position pos. Retourne 0 si substr ne se trouve pas dans la chaîne de caractères str:
INSTR(str,substr) Retourne la position de la première occurrence de la chaîne substr dans la chaîne de caractères str. Cette fonction est exactement la même que la fonction LOCATE(), à la différence que ces arguments sont inversés
LPAD(str,len,padstr) Retourne la chaîne de caractères str, complétée à gauche par la chaîne de caractères padstr jusqu'à ce que la chaîne de caractères str atteigne len caractères de long. Si la chaîne de caractères str est plus longue que len' caractères, elle sera raccourcie de len caractères.
RPAD(str,len,padstr) Retourne la chaîne de caractères str, complétée à droite par la chaîne de caractères padstr jusqu'à ce que la chaîne de caractères str atteigne len caractères de long. Si la chaîne de caractères str est plus longue que len' caractères, elle sera raccourcie de len caractères.
LEFT(str,len) Retourne les len caractères les plus à gauche de la chaîne de caractères str :
RIGHT(str,len) Retourne les len caractères les plus à droite de la chaîne de caractères str :
SUBSTRING(str,pos,len) SUBSTRING(str FROM pos FOR len) MID(str,pos,len) Retourne une chaîne de len caractères de long de la chaîne str, à partir de la position pos.
SUBSTRING(str,pos) SUBSTRING(str FROM pos) Retourne une portion de la chaîne de caractères str à partir de la position pos
SUBSTRING_INDEX(str,delim,count) Retourne une portion de la chaîne de caractères str, située avant count occurrences du délimiteur delim. Si l'argument count est positif, tout ce qui précède le délimiteur final sera retourné. Si l'argument count est négatif, tout ce qui précède le délimiteur final sera retourné
LTRIM(str) Retourne la chaîne de caractères str sans les espaces initiaux
RTRIM(str) Retourne la chaîne de caractères str sans les espaces finaux
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str) Retourne la chaîne de caractères str dont tous les préfixes et/ou suffixes remstr ont été supprimés. Si aucun des spécificateurs BOTH, LEADING ou TRAILING sont fournis, BOTH est utilisé comme valeur par défaut. Si remstr n'est pas spécifié, les espaces sont supprimés
SOUNDEX(str) Retourne la valeur Soundex de la chaîne de caractères str. Deux chaînes qui ont des sonorités proches auront des valeurs soundex proches. Une chaîne Soundex standard possède 4 caractères, mais la fonction SOUNDEX() retourne une chaîne de longueur arbitraire. Vous pouvez utiliser la fonction SUBSTRING() sur ce résultat pour obtenir une chaîne Soundex standard. Tout caractère non alphanumérique sera ignoré. Tous les caractères internationaux qui ne font pas partie de l'alphabet de base (A-Z) seront considérés comme des voyelles
REPLACE(str,from_str,to_str) Retourne une chaîne de caractères str dont toutes les occurrences de la chaîne from_str sont remplacées par la chaîne to_str
REPEAT(str,count) Retourne une chaîne de caractères constituée de la répétition de count fois la chaîne str. Si count <= 0, retourne une chaîne vide. Retourne NULL si str ou count sont NULL
REVERSE(str) Retourne une chaîne dont l'ordre des caractères est l'inverse de la chaîne str

Introduction à MySQL

INSERT(str,pos,len,newstr) Retourne une chaîne de caractères str, après avoir remplacé la portion de chaîne commençant à la position pos et de longueur len caractères, par la chaîne newstr
ELT(N,str1,str2,str3,...) 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()
FIELD(str,str1,str2,str3,...) Retourne l'index de la chaîne str dans la liste str1, str2, str3, ... Retourne 0 si str n'est pas trouvé. La fonction FIELD() est un complément de la fonction ELT():
FIND_IN_SET(str,strlist) Retourne une valeur de 1 à N si la chaîne str se trouve dans la liste strlist constituée de N chaînes. Une liste de chaîne 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 ‘,’
MAKE_SET(bits,str1,str2,...) Retourne une liste (une chaîne contenant des sous-chaînes séparées par une virgule ‘,’) constituée de chaînes qui ont le bit correspondant dans la liste bits. str1 correspond au bit 0, str2 au bit 1, etc... Les chaînes NULL dans les listes str1, str2, ... sont ignorées
EXPORT_SET(bits,on,off,[separateur,[nombre_de_bits]]) Retourne une chaîne dont tous les bits à 1 dans 'bit' sont représentés par la chaîne 'on', et dont tous les bits à 0 sont représentés par la chaîne 'off'. Chaque chaîne est séparée par 'séparateur' (par défaut, une virgule ',') et seul 'nombre de bits' (par défaut, 64) 'bits' est utilisé
LCASE(str) LOWER(str) Retourne une chaîne str dont tous les caractères ont été mis en minuscules, en accord avec le charset courant (le charset par défaut est ISO-8859-1 Latin1)
UCASE(str) UPPER(str) Retourne une chaîne str dont tous les caractères ont été mis en majuscules, en accord avec le charset courant (le charset par défaut est ISO-8859-1 Latin1)
LOAD_FILE(file_name) Lit le fichier file_name et retourne son contenu sous la forme d'une chaîne de caractères. Le fichier doit se trouver sur le serveur qui exécute MySQL.
QUOTE(str) Echappe les caractères d'une chaîne pour produire un résultat qui sera exploitable dans une requête SQL. Les caractères suivants seront précédés d'un anti-slash dans la chaîne retournée : le guillemet simple (‘'’), l'anti-slash (‘\’),ASCII NUL, et le Control-Z. Si l'argument vaut NULL, la valeur retournée sera le mot "NULL" sans les guillemets simples.
MATCH (col1,col2,...) AGAINST (expr) MATCH (col1,col2,...) AGAINST (expr IN BOOLEAN MODE) MATCH ... AGAINST() est utilisé pour les recherches en texte plein et retourne une information de pertinence - pertinence mesurée entre le texte des colonnes (col1,col2,...) et la valeur expr. La pertinence est un nombre à virgule flottante positif. La pertinence zéro signifie qu'il n'y a aucune similitude.
BINARY L'opérateur BINARY modifie la chaîne qui le suit en une chaîne binaire. C'est une solution simple pour forcer la comparaison de colonnes à être sensible à la casse même si la colonne n'est pas définie comme étant de type BINARY ou BLOB
MOD(N,M) % Modulo (équivalent de l'opérateur % dans le langage C). Retourne le reste de la division de N par M
CEILING(X) Retourne la valeur entière supérieure de X
ROUND(X) Retourne l'entier le plus proche de X
TRUNCATE() ou FLOOR(). ROUND(X,D) Retourne l'argument X, arrondi à un nombre à D décimales. Si D vaut 0, le résultat n'aura ni de partie décimale, ni de séparateur de décimale

Introduction à MySQL

EXP(X) Retourne la valeur de e (la base des logarithmes naturels) élevé à la puissance X
LN(X) Retourne le logarithme naturel de X (népérien)
LOG(X) LOG(B,X) Appelée avec un seul paramètre, cette fonction retourne le logarithme naturel (népérien) de X
POW(X,Y) POWER(X,Y) Retourne la valeur de X élevée à la puissance Y
SQRT(X) Retourne la racine carrée de X
PI() Retourne la valeur de PI. Par défaut, 5 décimales sont retournées
COS(X) Retourne le cosinus de X, où X est donné en radians
SIN(X) Retourne le sinus de X, où X est donné en radians
TAN(X) Retourne la tangente de X, où X est donné en radians
ACOS(X) Retourne l'arc cosinus de X, c'est-à-dire, la valeur de l'angle dont X est le cosinus
ASIN(X) Retourne l'arc sinus de X, c'est-à-dire, la valeur de l'angle dont le sinus est X. Retourne NULL si X n'est pas dans l'intervalle -1 – 1
ATAN(X) Retourne l'arc tangente de X, c'est-à-dire, la valeur de l'angle dont la tangente est X
ATAN(Y,X) ATAN2(Y,X) Retourne l'arc tangente des variables X et Y
COT(X) Retourne la cotangente de X
RAND() RAND(N) Retourne un nombre aléatoire à virgule flottante compris dans l'intervalle 0 -1.0. Si l'argument entier N est spécifié, il est utilisé comme initialisation du générateur de nombres aléatoires
LEAST(X,Y,...) Avec deux arguments ou plus, cette fonction retourne la plus petite des valeurs des différents arguments.
GREATEST(X,Y,...) Retourne le plus grand des arguments
DEGREES(X) Retourne l'argument X, converti de radians en degrés
RADIANS(X) Retourne l'argument X, converti de degrés en radians
TRUNCATE(X,D) Retourne l'argument X, tronqué à D décimales. Si D vaut 0, le résultat n'aura ni séparateur décimal, ni partie décimale
WHERE TO_DAYS(NOW()) - TO_DAYS(date_col) <= 30; DAYOFWEEK(date) Retourne l'index du jour de la semaine : pour date (1 = Dimanche, 2 = Lundi, ... 7 = Samedi). Ces index correspondent au standard ODBC
WEEKDAY(date) Retourne l'index du jour de la semaine, avec la conversion suivante : date (0 = Lundi, 1 = Mardi, ... 6 = Dimanche)
DAYOFMONTH(date) Retourne le jour de la date date, dans un intervalle de 1 à 31
DAYOFYEAR(date) Retourne le jour de la date date, dans un intervalle de 1 à 366
MONTH(date) Retourne le numéro du mois de la date date, dans un intervalle de 1 à 12
DAYNAME(date) Retourne le nom du jour de la semaine, en anglais, de la date date
MONTHNAME(date) Retourne le nom du mois de la date date
QUARTER(date) Retourne le numéro du trimestre de la date date, dans un intervalle de 1 à 4
WEEK(date) WEEK(date,first) Avec un seul argument, retourne le numéro de la semaine dans l'année de la date date, dans un intervalle de 0 à 53 (oui, il peut y avoir un début de semaine numéro 53), en considérant que Dimanche est le premier jour de la semaine. Avec deux arguments, la fonction WEEK() vous permet de spécifier si les semaines commencent le Dimanche ou le Lundi et la valeur retournée sera dans l'intervalle 0-53 ou bien 1-52. Voici un tableau explicatif sur le fonctionnement du second argument :

Value	Meaning
0	La semaine commence le Dimanche et retourne une valeur dans l'intervalle 0-53
1	La semaine commence le Lundi et retourne une valeur dans l'intervalle 0-53
2	La semaine commence le Dimanche et retourne une valeur dans l'intervalle 1-53
3	La semaine commence le Lundi et retourne une valeur dans l'intervalle 1-53

YEAR(date) Retourne l'année de la date date, dans un intervalle de 1000 à 9999

Introduction à MySQL

YEARWEEK(date) YEARWEEK(date,first) Retourne l'année et la semaine pour une date. Le second argument fonctionne exactement comme le second argument de la fonction WEEK(). Notez que l'année peut être différente de l'année de la date pour la première et la dernière semaine de l'année
WEEK() retourne une semaine dans une année donnée.
HOUR(time) Retourne le nombre d'heures pour l'heure time, dans un intervalle de 0 à 23 :
MINUTE(time) Retourne le nombre de minutes pour l'heure for time, dans un intervalle de 0 à 59
SECOND(time) Retourne le nombre de secondes pour l'heure time, dans un intervalle de 0 à 59
PERIOD_ADD(P,N) Ajoute N mois à la période P (au format YYMM ou YYYYMM). Retourne une valeur dans le format YYYYMM. Notez que l'argument P n'est pas de type date
PERIOD_DIFF(P1,P2) Retourne le nombre de mois entre les périodes P1 et P2. P1 et P2 doivent être dans au format YYMM ou YYYYMM. Notez que les arguments P1 et P2 ne sont pas de type date
DATE_ADD(date,INTERVAL expr type) DATE_SUB(date,INTERVAL expr type) ADDDATE(date,INTERVAL expr type) SUBDATE(date,INTERVAL expr type) Ces fonctions effectuent des calculs sur les dates.
EXTRACT(type FROM date) La fonction EXTRACT() utilise les mêmes types d'intervalles que la fonction DATE_ADD() ou la fonction DATE_SUB(), mais extrait des parties de date plutôt que des opérations de date.
TO_DAYS(date) Retourne le nombre de jours depuis la date 0 jusqu'à la date date
FROM_DAYS(N) Retourne la date correspondant au nombre de jours (N) depuis la date 0
DATE_FORMAT(date,format) Formate la date date avec le format format. Les spécificateurs suivants peuvent être utilisés dans la chaîne format :

Spécifier	Description
%M	Nom du mois (January..December)
%W	Nom du jour de la semaine (Sunday..Saturday)
%D	Jour du mois, avec un suffixe anglais (1st, 2nd, 3rd, etc.)
%Y	Année, au format numérique, sur 4 chiffres
%y	Année, au format numérique, sur 2 chiffres
%X	Année, pour les semaines qui commencent le Dimanche, au format numérique, sur 4 chiffres, utilisé avec '%V'
%x	Année, pour les semaines qui commencent le Lundi, au format numérique, sur 4 chiffres, utilisé avec '%v'
%a	Nom du jour de la semaine, en abrégé et en anglais (Sun..Sat)
%d	Jour du mois, au format numérique (00..31)
%e	Jour du mois, au format numérique (0..31)
%m	Mois, au format numérique (01..12)
%c	Mois, au format numérique (1..12)
%b	Nom du mois, en abrégé et en anglais (Jan..Dec)
%j	Jour de l'année (001..366)
%H	Heure (00..23)
%k	Heure (0..23)
%h	Heure (01..12)
%I	Heure (01..12)
%l	Heure (1..12)
%i	Minutes, au format numérique (00..59)
%r	Heures, au format 12-heures (hh:mm:ss [AP]M)

%T	Heures, au format 24-heures (hh:mm:ss)
%S	Secondes (00..59)
%s	Secondes (00..59)
%p	AM ou PM
%w	Numéro du jour de la semaine (0=Sunday..6=Saturday)
%U	Numéro de la semaine (00..53), où Dimanche est le premier jour de la semaine
%u	Numéro de la semaine (00..53), où Lundi est le premier jour de la semaine
%V	Numéro de la semaine (01..53), où Dimanche est le premier jour de la semaine, utilisé avec '%X'
%v	Numéro de la semaine (01..53), où Lundi est le premier jour de la semaine, utilisé avec

CURTIME() CURRENT_TIME Retourne l'heure courante au format 'HH:MM:SS' ou HHMMSS, suivant le contexte numérique ou chaîne
NOW() SYSDATE() CURRENT_TIMESTAMP Retourne la date courante au format 'YYYY-MM-DD HH:MM:SS' ou YYYYMMDDHHMMSS, suivant le contexte numérique ou chaîne
UNIX_TIMESTAMP() UNIX_TIMESTAMP(date) Lorsqu'elle est appelée sans argument, cette fonction retourne un timestamp Unix (nombre de secondes depuis '1970-01-01 00:00:00' GMT). Si UNIX_TIMESTAMP() est appelée avec un argument date , elle retourne le timestamp correspondant à cette date. date peut être une chaîne de type DATE, DATETIME, TIMESTAMP, ou un nombre au format YYMMDD ou YYYYMMDD, en horaire local
FROM_UNIXTIME(unix_timestamp) Retourne une représentation de l'argument unix_timestamp sous la forme 'YYYY-MM-DD HH:MM:SS' ou YYYYMMDDHHMMSS, suivant si la fonction est utilisée dans un contexte numérique ou de chaîne.
FROM_UNIXTIME(unix_timestamp,format) Retourne une chaîne représentant le timestamp Unix, formaté en accord avec la chaîne format. format doit contenir les mêmes spécificateurs que ceux utilisés par la fonction DATE_FORMAT()
TIME_TO_SEC(time) Retourne l'argument time, converti en secondes
CONVERT(expression,type) Où type est l'un des suivants :
BINARY
DATE
DATETIME
SIGNED {INTEGER}
TIME
UNSIGNED {INTEGER}
BIT_COUNT(N) Retourne le nombre de bits non nuls de l'argument N :
DATABASE() Retourne le nom de la base de données courante :
USER() SYSTEM_USER() SESSION_USER() Retourne l'utilisateur MySQL courant
PASSWORD(str) Calcule un mot de passe chiffré à partir de la chaîne str. C'est cette fonction qui est utilisée pour encrypter les mots de passe MySQL pour être stockés dans une colonne de type Password de la table user
ENCRYPT(str[,salt]) Chiffre la chaîne str en utilisant la fonction crypt(). L'argument salt doit être une chaîne de deux caractères
ENCRYPT() retournera toujours NULL. La fonction ENCRYPT() conserve uniquement les 8 premiers caractères de la chaîne str, au moins, sur certains systèmes. Le comportement exact est directement déterminé par la fonction système crypt() sous-jacente
ENCODE(str,pass_str) Chiffre la chaîne str en utilisant la clé pass_str. Pour déchiffrer le résultat, utilisez la fonction DECODE(). Le résultat est une chaîne binaire de la même longueur que string. Si vous voulez sauvegarder le résultat dans une colonne, utilisez une colonne de type BLOB.

DECODE(crypt_str,pass_str) Déchiffre la chaîne chiffrée crypt_str en utilisant la clé pass_str. crypt_str doit être une chaîne qui a été renvoyée par la fonction ENCODE()
MD5(string) Calcule la somme de vérification MD5 de la chaîne string. La valeur retournée est un entier hexadécimal de 32 caractères qui peut être utilisé, par exemple, comme clé de hachage
SHA1(string) SHA(string) Calcule la somme de vérification SHA1 160 bit de la chaîne string, comme décrit dans la RFC 3174 (Secure Hash Algorithm). La valeur retournée est un entier hexadécimal de 40 caractères, ou bien NULL dans le cas où l'argument vaut NULL. Une des possibilités d'utilisation de cette fonction est le hachage de clé. Vous pouvez aussi l'utiliser comme fonction de cryptographie sûre pour stocker les mots de passe.
AES_ENCRYPT(string,key_string) AES_DECRYPT(string,key_string) Ces fonctions permettent le chiffrement/déchiffrement de données utilisant l'algorithme AES (Advanced Encryption Standard
DES_DECRYPT(string_to_decrypt [, key_string]) Déchiffre une chaîne chiffrée à l'aide de la fonction ENCRYPT().
FORMAT(X,D) Formate l'argument X en un format comme '#,###,###.##', arrondi à D décimales. Si D vaut 0, le résultat n'aura ni séparateur décimal, ni partie décimale
VERSION() Retourne une chaîne indiquant la version courante du serveur MySQL
CONNECTION_ID() Retourne l'identifiant de connexion courant (thread_id). Chaque connexion a son propre identifiant unique
GET_LOCK(str,timeout) Tente de poser un verrou nommé str, avec un délai d'expiration (timeout) exprimé en secondes. Retourne 1 si le verrou a été posé avec succès, 0 si il n'a pas pu être posé avant l'expiration du délai et NULL si une erreur est survenue (comme par exemple un manque de mémoire, ou la mort du thread lui-même, par mysqladmin kill
RELEASE_LOCK(str) Libère le verrou nommé str, obtenu par la fonction GET_LOCK(). Retourne 1 si le verrou a bien été libéré, 0 si le verrou n'a pas été libéré par le thread (dans ce cas, le verrou reste posé) et NULL si le nom du verrou n'existe pas.
IS_FREE_LOCK(str) Regarde si le verrou nommé str peut être librement utilisé (i.e., non verrouillé). Retourne 1 si le verrou est libre (personne ne l'utilise), 0 si le verrou est actuellement utilisé et NULL si une erreur survient (comme un argument incorrect).
BENCHMARK(count,expr) La fonction BENCHMARK() exécute l'expression expr de manière répétée count fois. Elle permet de tester la vélocité de MySQL lors du traitement d'une requête. Le résultat est toujours 0. L'objectif de cette fonction ne se voit que du côté client, qui permet à ce dernier d'attacher la durée d'exécution de la requêtes sur le serveur.
INET_NTOA(expr) Retourne l'adresse réseau (4 ou 8 octets), de l'expression numérique exp
INET_ATON(expr) Retourne un entier qui représente l'expression numérique de l'adresse réseau. Les adresses peuvent être des entiers de 4 ou 8 octets.
MASTER_POS_WAIT(log_name, log_pos) Bloque le maître jusqu'à ce que l'esclave atteigne une position donnée dans le fichier d'historique principal, durant une réplication.
FOUND_ROWS() Retourne le nombre de lignes que la dernière commande SELECT SQL_CALC_FOUND_ROWS ... a retourné, si aucune limite n'a été définie par LIMIT.
COUNT(expr) Retourne le nombre de valeurs non-nulles (NULL) dans les lignes lues par la commande SELECT
COUNT(*) est légèrement différent, car il retourne le nombre de lignes lues, qu'elles contiennent ou pas la valeur NULL. COUNT(*) est optimisé pour retourner très rapidement le résultat
COUNT(DISTINCT expr,[expr...]) Retourne le nombre de lignes ayant des valeurs distinctes non-nulles (NULL)
AVG(expr) Retourne la valeur moyenne de la colonne expr

Introduction à MySQL

MIN(expr) MAX(expr) Retourne le minimum ou le maximum de l'expression expr. MIN() et MAX()
SUM(expr) Retourne la somme de l'expression expr. Notez que si aucune ligne n'est sélectionnée, la fonction retournera NULL !
STD(expr) STDDEV(expr) Retourne la déviation standard de l'expression expr. avec Oracle.
BIT_OR(expr) Retourne le OR bit-à-bit de l'expression expr. Les calculs sont effectués avec une précision de 64 bits (BIGINT).
BIT_AND(expr) Retourne le AND bit-à-bit de l'expression expr. Les calculs sont effectués avec une précision de 64 bits (BIGINT).