MySQL - Le langage

Description du langage MySQL

Chaînes

Si le serveur est en mode **ANSI_QUOTES**, les chaînes ne peuvent être mises qu'entre guillemets simples

\0 un 0 ASCII(NUL)
\' guillemet simple
\" guillemet double
\b effacement
\n nouvelle ligne
\r retour chariot
\t tabulation
\z ASCII(26)(Control-Z) Peut être encodé pour éviter des problèmes avec windows, puisqu'il équivaut à une fin de fichier
**** un anti-slash
\% un signe pourcentage littéral
_ Un signe souligné littéral

Si vous voulez insérer des données binaires dans un champ chaîne (comme un **BLOB**), les caractères suivants doivent être échappés :

NUL ASCII 0
**** ASCII 92
' ASCII 39
" ASCII 34

Les nombres

Valeurs hexadécimales :
mysql> SELECT x'4D7953514C';
-> 'MySQL'
mysql> SELECT 0xa+0;
-> 10
mysql> SELECT 0x5061756c;
-> 'Paul'
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
-> 'A', 65
mysql> SELECT HEX('cat');
-> '636174'
mysql> SELECT 0x636174;
-> 'cat'
Valeurs booléennes :
TRUE vaut 1 et FALSE vaut 0
champs de bits :

mysql> CREATE TABLE t (b BIT(8)) ;
mysql> INSERT INTO t SET b = b'11111111' ;
mysql> INSERT INTO t SET b = b'1010' ;
Valeurs NULL : signifie "pas de données" et est différent des valeurs comme 0 ou des chaînes vides

Noms de bases, tables, index, colonnes et alias

limité a 64 octets(255 pour les alias). les identifiants sont stockés en utf8 et peuvent être entre guillemets

caractères de protection pour les identifiants contenant des caractères spéciaux ou sont un mot réservé :
‘
mysql> SELECT * FROM ‘select‘ WHERE ‘select‘.id > 100 ;
identifiants. Pour faire référence à une colonne :
col_name
tbl_name.col_name
db_name.tbl_name.col_name
commentaires :
 # ceci est un commentaire
— ceci est un commentaire
/* ceci
est un
commentaire */

Mots clé réservés

ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	ASENSITIVE	BEFORE
BETWEEN	BIGINT	BINARY
BLOB	BOTH	BY
CALL	CASCADE	CASE
CHANGE	CHAR	CHARACTER
CHECK	COLLATE	COLUMN
CONDITION	CONSTRAINT	CONTINUE
CONVERT	CREATE	CROSS
CURRENT_DATE	CURRENT_TIME	CURRENT_TIMESTAMP
CURRENT_USER	CURSOR	DATABASE
DATABASES	DAY_HOUR	DAY_MICROSECOND
DAY_MINUTE	DAY_SECOND	DEC
DECIMAL	DECLARE	DEFAULT
DELAYED	DELETE	DESC
DESCRIBE	DETERMINISTIC	DISTINCT
DISTINCTROW	DIV	DOUBLE
DROP	DUAL	EACH
ELSE	ELSEIF	ENCLOSED
ESCAPED	EXISTS	EXIT
EXPLAIN	FALSE	FETCH
FLOAT	FLOAT4	FLOAT8
FOR	FORCE	FOREIGN
FROM	FULLTEXT	GRANT
GROUP	HAVING	HIGH_PRIORITY
HOUR_MICROSECOND	HOUR_MINUTE	HOUR_SECOND
IF	IGNORE	IN
INDEX	INFILE	INNER

```
INOUT_____________INSENSITIVE_____INSERT
INT_______________INT1____________INT2
INT3______________INT4____________INT8
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
MATCH_____________MEDIUMBLOB______MEDIUMINT
MEDIUMTEXT________MIDDLEINT_______MINUTE_MICROSECOND
MINUTE_SECOND_____MOD_____________MODIFIES
NATURAL___________NOT_____________NO_WRITE_TO_BINLOG
NULL______________NUMERIC_________ON
OPTIMIZE__________OPTION__________OPTIONALLY
OR________________ORDER___________OUT
OUTER_____________OUTFILE_________PRECISION
PRIMARY___________PROCEDURE_______PURGE
READ______________READS___________REAL
REFERENCES________REGEXP__________RELEASE
RENAME____________REPEAT__________REPLACE
REQUIRE___________RESTRICT________RETURN
REVOKE____________RIGHT___________RLIKE
SCHEMA____________SCHEMAS_________SECOND_MICROSECOND
SELECT____________SENSITIVE_______SEPARATOR
SET_______________SHOW____________SMALLINT
SONAME____________SPATIAL_________SPECIFIC
SQL_______________SQLEXCEPTION____SQLSTATE
SQLWARNING________SQL_BIG_RESULT__SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT__SSL_____________STARTING
STRAIGHT_JOIN_____TABLE___________TERMINATED
THEN______________TINYBLOB________TINYINT
TINYTEXT__________TO______________TRAILING
TRIGGER___________TRUE____________UNDO
UNION_____________UNIQUE__________UNLOCK
UNSIGNED__________UPDATE__________USAGE
USE_______________USING___________UTC_DATE
UTC_TIME__________UTC_TIMESTAMP___VALUES
VARBINARY_________VARCHAR_________VARCHARACTER
VARYING___________WHEN____________WHERE
WHILE_____________WITH____________WRITE
XOR_______________YEAR_MONTH______ZEROFILL
```

Voici de nouveaux mots réservés en MySQL 5.0 :

```
ASENSITIVE________CALL____________CONDITION
CONTINUE__________CURSOR__________DECLARE
DETERMINISTIC_____EACH____________ELSEIF
EXIT______________FETCH___________INOUT
INSENSITIVE_______ITERATE_________LEAVE
LOOP______________MODIFIES________OUT
READS_____________RELEASE_________REPEAT
RETURN____________SCHEMA__________SCHEMAS
SENSITIVE_________SPECIFIC________SQL
SQLEXCEPTION______SQLSTATE________SQLWARNING
TRIGGER___________UNDO____________WHILE
```

Types de colonnes - numériques

- si vous spécifiez l'option **ZEROFILL** pour une valeur numérique, mysql ajoute automatiquement l'attribut **UNSIGNED** à la colonne.

TINYINT[(M)] [UNSIGNED] [ZEROFILL] Un très petit entier. L'intervalle de validité pour les entiers signés est de -128 à 127. L'intervalle de validité pour les entiers non-signés est 0 à 255.

BIT, BOOL, BOOLEAN Ce sont des synonymes de TINYINT(1). Un type booléen complet, qui sera introduit pour être en accord avec la norme SQL-99.

SMALLINT[(M)] [UNSIGNED] [ZEROFILL] Un petit entier. L'intervalle de validité pour les entiers signés est de -32768 à 32767. L'intervalle de validité pour les entiers non-signés est 0 à 65535.

MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL] Un entier. L'intervalle de validité pour les entiers signés est de -8388608 à 8388607. L'intervalle de validité pour les entiers non-signés est 0 à 16777215.

INT[(M)] [UNSIGNED] [ZEROFILL] Un grand entier. L'intervalle de validité pour les entiers signés est de -2147483648 à 2147483647. L'intervalle de validité pour les entiers non-signés est 0 à 4294967295.

INTEGER[(M)] [UNSIGNED] [ZEROFILL] Synonyme INT.

BIGINT[(M)] [UNSIGNED] [ZEROFILL] Un très grand entier. L'intervalle de validité pour les entiers signés est de -9223372036854775808 à 9223372036854775807. L'intervalle de validité pour les entiers non-signés est 0 à 18446744073709551615.

Quelques conseils à suivre avec les colonnes de type BIGINT :

- Tous les calculs arithmétiques sont fait en utilisant des BIGINT signés ou des valeurs DOUBLE. Il est donc recommandé de ne pas utiliser de grands entiers non-signés dont la taille dépasse 9223372036854775807 (63 bits), hormis avec les fonctions sur les bits ! Si vous faîtes cela, les derniers chiffres du résultats risquent d'être faux, à cause des erreurs d'arrondis lors de la conversion de BIGINT en DOUBLE.

MySQL peut gérer des BIGINT dans les cas suivants :

- Utiliser des entiers pour stocker des grandes valeurs entières non signées, dans une colonne de type BIGINT.
- Avec MIN(big_int_column) et MAX(big_int_column).
- Avec les opérateurs (+, -, *, etc.) où tous les opérandes sont des entiers.
- Vous pouvez toujours stocker une valeur entière exacte BIGINT dans une colonne de type chaîne. Dans ce cas, MySQL fera des conversions chaîne / nombre, qui n'utilisera pas de représentation intermédiaire en nombre réels.

'-', '+' et '*' utiliseront l'arithmétique entière des BIGINT lorsque les deux arguments sont des entiers. Cela signifie que si vous multipliez deux entiers (ou des résultats de fonctions qui retournent des entiers), vous pourriez rencontrer des résultats inattendus lorsque le résultat est plus grand que 9223372036854775807.

FLOAT(precision) [UNSIGNED] [ZEROFILL] Un nombre à virgule flottante. précision peut valoir <=24 pour une précision simple, et entre 25 et 53 pour une précision double. Ces types sont identiques aux types FLOAT et DOUBLE, décrit ci-dessous. FLOAT(X) a le même intervalle de validité que FLOAT et DOUBLE, mais la taille d'affichage et le nombre de décimales est indéfini.

En MySQL version 3.23, c'est un véritable nombre à virgule flottante. Dans les versions antérieures, FLOAT(precision) avait toujours 2 décimales. Notez qu'utiliser FLOAT peut vous donner des résultats inattendus, car tous les calculs de MySQL sont fait en double précision. Cette syntaxe est fournie pour assurer la compatibilité avec ODBC. Utiliser des FLOAT peut vous donner des résultats inattendus, car les calculs sont fait en précision double.

FLOAT[(M,D)] [UNSIGNED] [ZEROFILL] Un petit nombre à virgule flottante, en précision simple. Les valeurs possibles vont de -3.402823466E+38 à -1.175494351E-38, 0, et 1.175494351E-38 à 3.402823466E+38. Si UNSIGNED est spécifié, les valeurs négatives sont interdites. L'attribut M indique la taille de l'affichage, et D est le nombre de décimales. FLOAT sans argument et FLOAT(X) (où X est dans l'intervalle 0 à 24) représente les nombres à virgule flottante en précision simple.

DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL] Un nombre à virgule flottante, en précision double. Les valeurs possibles vont de -1.7976931348623157E+308 à -2.2250738585072014E-308, 0, et 2.2250738585072014E-308 à 1.7976931348623157E+308. Si UNSIGNED est spécifié, les valeurs négatives sont interdites. L'attribut M indique la taille de l'affichage, et D est le nombre de décimales. DOUBLE sans argument et FLOAT(X) (où X est dans l'intervale 25 to 53) représente les nombres à virgule flottante en précision double.

DOUBLE PRECISION [(M,D)] [UNSIGNED] [ZEROFILL], REAL [(M,D)] [UNSIGNED] [ZEROFILL] Ce sont des synonymes pour DOUBLE. Exception : si le serveur SQL utilise l'option REAL_AS_FLOAT, REAL est alors un synonyme de FLOAT plutôt que DOUBLE.

DECIMAL [(M [,D])] [UNSIGNED] [ZEROFILL] Un nombre à virgule flottante littéral. Il se comporte comme une colonne de type CHAR : "littéral" ("unpacked") signifie que le nombre est stocké sous forme de chaîne : chaque caractère représente un chiffre. La virgule décimale et le signe moins '-' des nombres négatifs ne sont pas comptés dans M (mais de l'espace leur est réservé). Si D vaut 0, les valeurs n'auront pas de virgule décimale ou de partie décimale. L'intervale de validité du type DECIMAL est le même que DOUBLE, mais le vrai intervalle de validité de DECIMAL peut être restreint par le choix de la valeur de M et D. Si UNSIGNED est spécifié, les valeurs négatives sont interdites. 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], FIXED [(M [,D])] [UNSIG Ce sont des synonymes pour DECIMAL

Types de colonnes - dates et heures

DATE Une date. L'intervalle supporté va de '1000-01-01' à '9999-12-31'. MySQL affiche les valeurs de type DATE au format 'YYYY-MM-DD', mais vous permet d'assigner des valeurs DATE en utilisant plusieurs formats de chaînes et nombres.

DATETIME Une combinaison de date et heure. L'intervalle de validité va de '1000-01-01 00 :00 :00' à '9999-12-31 23 :59 :59'. MySQL affiche les valeurs de type DATE au format 'YYYY-MM-DD HH :MM :SS', mais vous permet d'assigner des valeurs DATE en utilisant plusieurs formats de chaînes et nombres.

TIMESTAMP [(M)] Un timestamp. L'intervalle de validité va de '1970-01-01 00 :00 :00' à quelque part durant l'année 2037. les valeurs TIMESTAMP sont affichées au format YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD ou YYMMDD, suivant que la valeur de M est 14 (ou absente), 12, 8 ou 6, respectivement, mais vous permet d'assigner des valeurs aux colonnes TIMESTAMP en utilisant des nombres ou des chaînes. TIMESTAMP est retournée comme une chaîne, au format 'YYYY-MM-DD HH :MM :SS'. Si vous voulez que MySQL vous retourne un nombre, ajoutez +0 à la colonne. Les différentes tailles de timestamp ne sont pas supportées.

Une colonne TIMESTAMP est utile pour enregistrer les dates et heures des opérations INSERT et UPDATE, car elle prend automatiquement date actuellement si vous ne lui assignez pas de valeur par vous-même. Vous pouvez aussi lui donner la valeur courante en lui donnant la valeur NULL. L'argument M affecte l'affichage des colonnes de type TIMESTAMP. ses valeurs sont toujours stockées sur 4 octets. Notez que les colonnes TIMESTAMP(M) où M vaut 8 ou 14 sont indiquée comme étant des nombres, alors que les colonnes TIMESTAMP(M) sont indiquées comme étant des chaînes. Cela est fait pour s'assurer que l'ont peut enregistrer et lire correctement les tables ayant ce type.

TIME Une heure. L'intervalle va de '-838 :59 :59' à '838 :59 :59'. MySQL affiche les valeurs TIME au format 'HH :MM :SS', mais vous permet d'assigner des valeurs TIME en utilisant des nombres ou des chaînes.

YEAR [(2|4)] Une année, au format 2 ou 4 chiffres (par défaut, c'est 4 chiffres). Les valeurs possibles vont de 1901 à 2155 plus 0000 pour le format à 4 chiffres, et de 1970 à 2069 si vous utilisez le format à 2 chiffres. MySQL affiche les valeurs YEAR au format YYYY mais vous permet d'assigner des valeurs en utilisant des nombres ou des chaînes.

Types de colonnes - chaînes

CHAR et VARCHAR Les types CHAR et VARCHAR sont similaires, mais diffèrent dans la manière dont ils sont stockés et récupérés.

La longueur d'une colonne CHAR est fixée à la longueur que vous avez défini lors de la création de la table. La longueur peut être n'importe quelle valeur entre 1 et 255. Quand une valeur CHAR est enregistrée, elle est complété à droite avec des espaces jusqu'à atteindre la valeur fixée. Quand une valeur de CHAR est lue, les espaces en trop sont retirés.

Les valeurs contenues dans les colonnes de type VARCHAR sont de tailles variables. Vous pouvez déclarer une colonne VARCHAR pour que sa taille soit comprise entre 1 et 255, exactement comme pour les colonnes CHAR. Par contre, contrairement à CHAR, les valeurs de VARCHAR sont stockées en utilisant autant de caractères que nécessaire, plus un octet pour mémoriser la longueur. Les valeurs ne sont pas

complétées. Au contraire, les espaces finaux sont supprimés avant stockage. Si vous assignez une chaîne de caractères qui dépasse la capacité de la colonne CHAR ou VARCHAR, celle ci est tronquée jusqu'à la taille maximale du champ.

Les valeurs dans les colonnes CHAR et VARCHAR sont classées et comparées sans tenir compte de la casse, à moins que l'attribut BINARY n'ai été spécifié lors de la création de la table. L'attribut BINARY signifie que les valeurs sont classées et triées en tenant compte de la casse, suivant l'ordre des caractères ASCII de la machine ou est installé le serveur MySQL. BINARY n'affecte pas les méthodes de lecture et de stockage des valeurs. L'attribut BINARY se propage dans une expression : il suffit qu'une seule colonne, utilisée dans une expression, ait l'attribut BINARY pour que toute l'expression ne tienne plus compte de la casse.

BINARY et VARBINARY Les types BINARY et VARBINARY sont similaires à CHAR et VARCHAR, hormis le fait qu'ils contiennent des chaînes binaires, plutôt que des chaînes de texte. C'est à dire, qu'ils contiennent des chaînes d'octets, plutôt que des chaînes de caractères. Cela signifie qu'ils n'ont pas de jeu de caractères associé, et les tris et comparaisons sont basées sur la valeur numérique de l'octet. La taille maximale pour les types BINARY et VARBINARY, est la même que celles de CHAR et VARCHAR, hormis le fait que la taille de BINARY et VARBINARY est une taille en octets, et non pas en caractères.

BLOB et TEXT une valeur TEXT est une valeur BLOB insensible à la casse. Pour les index des colonnes BLOB et TEXT, vous devez spécifier une taille d'index. Pour les colonnes de type CHAR et VARCHAR, la taille du préfixe est optionnelle. Il n'y a pas de suppression des espaces finaux lors du stockage de valeur dans des colonnes de type BLOB et TEXT, ce qui est le cas dans pour les colonnes de type VARCHAR. Les colonnes BLOB et TEXT ne peuvent avoir de valeur par défaut. (DEFAULT). Les quatre types BLOB (TINYBLOB, BLOB, MEDIUMBLOB, et LONGBLOB). Les quatre types TEXT (TINYTEXT, TEXT, MEDIUMTEXT, et LONGTEXT)

ENUM Une énumération ENUM est une chaîne dont la valeur est choisie parmi une liste de valeurs autorisées lors de la création de la table. Cette chaîne peut aussi être la chaîne vide ("") ou NULL dans certaines circonstances. Si une colonne d'énumération est déclarée NULL, NULL devient aussi une valeur autorisée, et la valeur par défaut est alors NULL. Si une colonne d'énumération est déclarée NOT NULL, la valeur par défaut est le premier élément de la liste des valeurs autorisées.

Chaque élément de l'énumération dispose d'un index.

- Les valeurs de la liste des valeurs autorisées sont indexées à partir de 1.
- L'index de la valeur NULL est NULL.
- Une énumération peut avoir un maximum de 65535 éléments.
- Il est déconseillé de stocker des valeurs numériques dans un ENUM car cela engendre des confusions

SET Un SET est une chaîne qui peut avoir zéro ou plusieurs valeurs, chacune doit être choisie dans une liste de valeurs définies lors de la création de la table. Les valeurs des colonnes SET composées de plusieurs membres sont définies en séparant celles-ci avec des virgules (‘,’). Ce qui fait que la valeur d'un membre de SET ne peut contenir lui même de virgule. Un SET peut avoir au plus 64 membres.

Par exemple, une colonne définie en tant que SET("un", "deux") NOT NULL peut avoir l'une de ces valeurs :
""
"un"
"deux"
"un,deux"

Capacité des colonnes numériques

Type de colonne/Espace requis

TINYINT 1 octet

SMALLINT 2 octets

MEDIUMINT 3 octets

INT, INTEGER 4 octets

BIGINT 8 octets

FLOAT(p) 4 if X <= 24 or 8 if 25 <= X <= 53

FLOAT 4 octets

DOUBLE PRECISION, REAL 8 octets

DECIMAL(M,D) M+2 octets si D > 0, M+1 octets si D=0 (D+2, si M<D)

Capacité des colonnes temporelles

Type de colonne / Espace requis
DATE 3 octets
DATETIME 8 octets
TIMESTAMP 4 octets
TIME 3 octets
YEAR 1 octet

Capacité des colonnes texte

Type de colonne / Espace requis
CHAR(M) M octets, 1 <= M 255
VARCHAR(M) L+1 octets, avec L <= M et 1 <= M <= 255
TINYBLOB, TINYTEXT L+1 octets, avec L < 2^8
BLOB, TEXT L+2 octets, avec L < 2^16
MEDIUMBLOB, MEDIUMTEXT L+3 octets, avec L < 2^24
LONGBLOB, LONGTEXT L+4 octets, avec L < 2^32
ENUM('valeur1','valeur2',...) 1 ou 2 octets, suivant le nombre d'éléments de l'énumération (65535 au maximum)
SET('valeur1','valeur2',...) 1, 2, 3, 4 ou 8 octets, suivant le nombre de membres de l'ensemble (64 au maximum)

Fonctions dans SELECT et WHERE - Priorité des opérateurs

:=
||, OR, XOR
&&, AND
BETWEEN, CASE, WHEN, THEN, ELSE
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
|
&
«, »
-, +
***, /, DIV, %, MOD**
^
- (unary minus), (unary bit inversion)
NOT, !
BINARY, COLLATE

Fonctions dans SELECT et WHERE - Opérateurs de comparaison

= 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
```

<=> Comparaison compatible avec NULL. Cet opérateur fait une comparaison d'égalité comme l'opérateur =, mais retourne 1 plutôt que NULL si les deux opérandes sont NULL, et 0 plutôt que NULL si un opérande est NULL

<>, != Différent

```
mysql> SELECT '.01' <> '0.01' ;
-> 1
mysql> SELECT .01 <> '0.01' ;
-> 0
mysql> SELECT 'zapp' <> 'zappp' ;
-> 1
```

<= Inférieur ou égal

```
mysql> SELECT 0.1 <= 2 ;
-> 1
```

< Strictement inférieur

```
mysql> SELECT 2 < 2 ;
-> 0
```

>= Supérieur ou égal

```
mysql> SELECT 2 >= 2 ;
-> 1
```

> Strictement supérieur

```
mysql> SELECT 2 > 2 ;
-> 0
```

IS NULL, IS NOT NULL Tester si une valeur est ou n'est pas NULL

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL ;
-> 0 0 1
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL ;
-> 1 1 0
```

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. Dans tous les autres cas, la conversion de type prends place, selon les règles suivantes, mais appliquée aux trois arguments.

```
mysql> SELECT 1 BETWEEN 2 AND 3 ;
-> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c' ;
-> 1
mysql> SELECT 2 BETWEEN 2 AND '3' ;
-> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3' ;
-> 0
```

expr NOT BETWEEN min AND max Même chose que NOT (expr BETWEEN min AND max). COALESCE(list) Retourne le premier élément non-NULL de la liste

```
mysql> SELECT COALESCE(NULL,1);
-> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
-> NULL
```

GREATEST(value1,value2,...) Avec deux ou plusieurs arguments, retourne la valeur la plus grande. Les arguments sont comparés en utilisant les mêmes règles que pour LEAST()

```
mysql> SELECT GREATEST(2,0);
-> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
-> 767.0
mysql> SELECT GREATEST('B','A','C');
-> 'C'
```

expr IN (valeur,...) Retourne 1 si **expr** est l'une des valeurs dans la liste **IN**, sinon retourne 0. Si toutes les valeurs sont des constantes, toutes les valeurs sont évaluées avec le type de **expr** et triées. La recherche de l'élément est alors faite en utilisant la recherche binaire. Cela signifie que **IN** est très rapide si les valeurs contenues dans la liste **IN** sont toutes des constantes. Si **expr** est une chaîne sensible à la casse, la comparaison est faite dans un contexte sensible à la casse :

```
mysql> SELECT 2 IN (0,3,5,'wefwf');
-> 0
mysql> SELECT 'wefwf' IN (0,3,5,'wefwf');
-> 1
```

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

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

```
mysql> SELECT ISNULL(1+1);
-> 0
mysql> SELECT ISNULL(1/0);
-> 1
```

Notez que la comparaison de deux valeurs NULL en utilisant = donnera toujours false !

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. Cela est due à la recherche binaire utilisée (très rapide)

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
-> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
-> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
-> 0
```

LEAST(value1,value2,...) Avec deux arguments ou plus, retourne la plus petite valeur.

```
mysql> SELECT LEAST(2,0);
-> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
-> 3.0
mysql> SELECT LEAST('B','A','C');
-> 'A'
```

Fonctions dans SELECT et WHERE - Opérateurs logiques

NOT, ! (NON) logique. Évalue à 1 si l'opérande est 0, à 0 si l'opérande est non nulle, et NOT NULL retourne NULL. Le dernier exemple donne 1 car l'expression est évaluée comme (!1)+1

```
mysql> SELECT NOT 10;
-> 0
mysql> SELECT NOT 0;
```

-> 1
mysql> SELECT NOT NULL ;
-> NULL
mysql> SELECT ! (1+1) ;
-> 0
mysql> SELECT ! 1+1 ;
-> 1

AND, && (ET) logique. Évalue à 1 si toutes les opérandes sont différentes de zéro et de NULL, à 0 si l'une des opérandes est 0, dans les autres cas, NULL est retourné.

mysql> SELECT 1 && 1 ;
-> 1
mysql> SELECT 1 && 0 ;
-> 0
mysql> SELECT 1 && NULL ;
-> NULL
mysql> SELECT 0 && NULL ;
-> 0
mysql> SELECT NULL && 0 ;
-> 0

Notez que pour les versions antérieures à la 4.0.5 l'évaluation est interrompue lorsque NULL est rencontré, au lieu de continuer à tester une éventuelle existence de 0. Cela signifie que dans ces versions, SELECT (NULL AND 0) retourne NULL au lieu de 0. En 4.0.5 le code a été revu pour que le résultat réponde toujours au normes ANSI tout en optimisant le plus possible.

OR, || (OU inclusif) logique. Évalue à 1 si aucune opérande n'est nulle, à NULL si l'une des opérandes est NULL, sinon 0 est retourné.

mysql> SELECT 1 || 1 ;
-> 1
mysql> SELECT 1 || 0 ;
-> 1
mysql> SELECT 0 || 0 ;
-> 0
mysql> SELECT 0 || NULL ;
-> NULL
mysql> SELECT 1 || NULL ;
-> 1

XOR (OU exclusif) logique. Retourne NULL si l'une des opérandes est NULL. Pour les opérandes non-NULL, évalue à 1 si un nombre pair d'opérandes est non-nul, sinon 0 est retourné. a XOR b est mathématiquement égal à (a AND (NOT b)) OR ((NOT a) and b).

mysql> SELECT 1 XOR 1 ;
-> 0
mysql> SELECT 1 XOR 0 ;
-> 1
mysql> SELECT 1 XOR NULL ;
-> NULL
mysql> SELECT 1 XOR 1 XOR 1 ;
-> 1

Fonctions dans SELECT et WHERE - Fonctions de contrôle

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

mysql> SELECT IFNULL(1,0) ;
-> 1

```
mysql> SELECT IFNULL(NULL,10);
-> 10
mysql> SELECT IFNULL(1/0,10);
-> 10
mysql> SELECT IFNULL(1/0,'oui');
-> 'oui'
```

NULLIF(expr1,expr2) Si l'expression expr1 = expr2 est vrai, la fonction retourne NULL sinon elle retourne expr1. Cela revient à faire CASE WHEN x = y THEN NULL ELSE x END

```
mysql> SELECT NULLIF(1,1);
-> NULL
mysql> SELECT NULLIF(1,2);
-> 1
```

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

```
mysql> SELECT IF(1>2,2,3);
-> 3
mysql> SELECT IF(1<2,'oui','non');
-> 'oui'
mysql> SELECT IF(STRCMP('test','test1'),'non','oui');
-> 'non'
```

Si l'argument expr2 ou expr3 est explicitement NULL alors le type du résultat de la fonction IF() est le type de la colonne non NULL. L'argument expr1 est évalué comme un entier, cela signifie que si vous testez un nombre à virgule flottante ou une chaîne de caractères, vous devez utiliser une opération de comparaison

```
mysql> SELECT IF(0.1,1,0);
-> 0
mysql> SELECT IF(0.1<>0,1,0);
-> 1
```

CASE valeur WHEN [compare-value] THEN résultat [WHEN [compare-value] THEN résultat ...] [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 réalisé, alors le résultat de la clause ELSE est retourné. Si il n'y a pas de clause ELSE alors NULL est retourné. Le type de la valeur retournée (INTEGER, DOUBLE ou STRING) est de même type que la première valeur retournée (l'expression après le premier THEN).

```
mysql> SELECT CASE 1 WHEN 1 THEN "un"
WHEN 2 THEN "deux" ELSE "plus" END;
-> "un"
mysql> SELECT CASE WHEN 1>0 THEN "vrai" ELSE "faux" END;
-> "vrai"
mysql> SELECT CASE BINARY "B" WHEN "a" THEN 1 WHEN "b" THEN 2 END;
-> NULL
```

Fonctions dans SELECT et WHERE - Fonctions de chaînes de caractères

Les fonctions qui traitent les chaînes de caractères retournent NULL si la longueur du résultat finit par dépasser la taille maximale du paramètre max_allowed_packet, défini dans la configuration du serveur.

ASCII(str) Retourne le code ASCII du premier caractère de la chaîne de caractères str. Retourne 0 si la chaîne de caractère str est vide. Retourne NULL si la chaîne de caractères str est NULL. ASCII() fonctionne avec des valeurs numériques entre 0 et 255. Voir aussi la fonction ORD().

mysql> SELECT ASCII('2');
-> 50
mysql> SELECT ASCII(2);
-> 50
mysql> SELECT ASCII('dx');
-> 100

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.

mysql> SELECT BIN(12);
-> '1100'

BIT_LENGTH(str) Retourne le nombre de bits de la chaîne de caractères str.

mysql> SELECT BIT_LENGTH('text');
-> 32

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

mysql> SELECT CHAR(77,121,83,81,'76');
-> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
-> 'MMM'

CHAR_LENGTH(str) Retourne le nombre de caractères de la chaîne str : Un caractère multi-octets compte comme un seul caractère. Cela signifie que pour une chaîne contenant 5 caractères de 2 octets, LENGTH() retournera 10, alors que CHAR_LENGTH() retournera 5.

CHARACTER_LENGTH(str) est un synonyme de CHAR_LENGTH()

COMPRESS(string_to_compress) Compresse une chaîne. Cette fonction requiert la présence de la bibliothèque zlib. Sinon, la valeur retournée sera toujours NULL.

mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
-> 21
mysql> SELECT LENGTH(COMPRESS(''));
-> 0
mysql> SELECT LENGTH(COMPRESS('a'));
-> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
-> 15

La chaîne compressée est stockée de cette manière : Les chaînes vides sont stockées comme des chaînes vides ; Les chaînes non-vides sont stockées avec 4 octets de plus, indiquant la taille de la chaîne non compressée, suivie de la chaîne compressée. Si la chaîne se termine avec des espaces, un point supplémentaire "." est ajouté, pour éviter que les espaces terminaux soient supprimés de la chaîne. N'utilisez pas les types CHAR ou VARCHAR pour stocker des chaînes compressée. Il est mieux d'utiliser un type BLOB.

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

mysql> SELECT CONCAT_WS('','','Premier nom','Deuxième nom','Dernier nom');
-> 'Premier nom,Deuxième nom,Dernier nom'
mysql> SELECT CONCAT_WS('','','Premier nom',NULL,'Dernier nom');
-> 'Premier nom,Dernier nom'

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. Dans le cas contraire, N sera traité comme un nombre non-signé. La fonction CONV travaille avec une précision de 64 bits

mysql> SELECT CONV('a',16,2);
-> '1010'

```
mysql> SELECT CONV("6E",18,8);
-> '172'
mysql> SELECT CONV(-17,10,-18);
-> '-H'
mysql> SELECT CONV(10+"10"+'10'+0xa,10,10);
-> '40'
```

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()

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
-> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
-> 'foo'
```

EXPORT_SET(bits,on,off,[séparateur,[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

```
mysql> SELECT EXPORT_SET(5,'Y','N',',',4)
-> Y,N,Y,N
```

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()

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
-> 0
```

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 ’,’

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
-> 2
```

HEX(N_or_S) Si l'argument N_OR_S est un nombre, cette fonction retournera une chaîne de caractère 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). Si N_OR_S est une chaîne de caractères, cette fonction retournera une chaîne de caractères hexadécimale de N_OR_S où chaque caractère de N_OR_S est converti en 2 chiffres hexadécimaux. C'est l'inverse de la chaîne 0xff.

```
mysql> SELECT HEX(255);
-> 'FF'
mysql> SELECT HEX("abc");
-> 616263
mysql> SELECT 0x616263;
-> "abc"
```

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. Cette fonction gère les caractères multi-octets.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
-> 'QuWhattic'
```

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. Cette fonction gère les caractères multi-octets. cette fonction sera sensible à la casse si l'argument est une chaîne de caractères binaire.

```
mysql> SELECT INSTR('foobarbar', 'bar');
-> 4
mysql> SELECT INSTR('xbar', 'foobar');
-> 0
```

LCASE(str) est un synonyme de LOWER().

LEFT(str,len) Retourne les len caractères les plus à gauche de la chaîne de caractères str. Cette fonction gère les caractères multi-octets.

```
mysql> SELECT LEFT('foobarbar', 5);
-> 'fooba'
```

LENGTH(str) Retourne la taille de la chaîne str, mesurée en octets. Un caractère multi-octets compte comme un seul caractère. Cela signifie que pour une chaîne contenant 5 caractères de 2 octets, LENGTH() retournera 10, alors que CHAR_LENGTH() retournera 5.

```
mysql> SELECT LENGTH('text');
-> 4
```

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, vous devez spécifier le chemin absolu du fichier et vous devez avoir les droits en lecture sur celui-ci. Le fichier doit pouvoir être lisible par tous et doit être plus petit que max_allowed_packet. Si ce fichier n'existe pas ou ne peut pas être lu pour différentes raisons, la fonction retourne NULL

```
mysql> UPDATE tbl_name
SET blob_column=LOAD_FILE("/tmp/picture")
WHERE id=1
```

LOCATE(substr,str), LOCATE(substr,str,pos) 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. Cette fonction gère les caractères multi-octets.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
-> 4
mysql> SELECT LOCATE('xbar', 'foobar');
-> 0
mysql> SELECT LOCATE('bar', 'foobarbar',5);
-> 7
```

LOWER(str) Retourne la chaîne str avec tous les caractères en minuscules, en fonction du jeu de caractères courant (par défaut, c'est le jeu ISO-8859-1 Latin1). Cette fonction gère les caractères multi-octets.

```
mysql> SELECT LOWER('QUADRATIQUE');
-> 'quadratique'
```

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.

```
mysql> SELECT LPAD('hi',4,'??');
-> '??hi'
```

LTRIM(str) Retourne la chaîne de caractères str sans les espaces initiaux

```
mysql> SELECT LTRIM(' barbar');
-> 'barbar'
```

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

```
mysql> SELECT MAKE_SET(1,'a','b','c');
-> 'A'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
-> 'hello,world'
mysql> SELECT MAKE_SET(0,'a','b','c');
-> ''
```

MID(str,pos,len) est un synonyme de SUBSTRING(str,pos,len).

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

```
mysql> SELECT OCT(12);
-> '14'
```

OCTET_LENGTH(str) est un synonyme de LENGTH().

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

(1st octet * 256)
+ (2nd octet * 256^2)
+ (3rd octet * 256^3) ...
Si le premier caractère n'est pas un caractère multi-octet, la fonction retournera la même valeur que la fonction ASCII()
mysql> SELECT ORD('2');
-> 50

POSITION(substr IN str) est un synonyme de LOCATE(substr,str).

QUOTE(str) Échappe 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 Contrôle-Z. Si l'argument vaut NULL, la valeur retournée sera le mot "NULL" sans les guillemets simples.

mysql> SELECT QUOTE("Don't");
-> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
-> NULL

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

mysql> SELECT REPEAT('MySQL', 3);
-> 'MySQLMySQLMySQL'

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. Cette fonction gère les caractères multi-octets.

mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
-> 'WwWwWw.mysql.com'

REVERSE(str) Retourne une chaîne dont l'ordre des caractères est l'inverse de la chaîne str. Cette fonction gère les caractères multi-octets.

mysql> SELECT REVERSE('abc');
-> 'cba'

RIGHT(str,len) Retourne les len caractères les plus à droite de la chaîne de caractères str. Cette fonction gère les caractères multi-octets.

mysql> SELECT RIGHT('foobarbar', 4);
-> 'rbar'

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.

mysql> SELECT RPAD('hi',5,'?');
-> 'hi???'

RTRIM(str) Retourne la chaîne de caractères str sans les espaces finaux. Cette fonction gère les caractères multi-octets.

mysql> SELECT RTRIM('barbar ');
-> 'barbar'

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 alpha-numérique sera ignoré. Tous les caractères internationaux qui ne font pas partie de l'alphabet de base (A-Z) seront considérés comme des voyelles

mysql> SELECT SOUNDEX('Hello');
-> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
-> 'Q36324'
expr1 SOUNDS LIKE expr2
Identique à
SOUNDEX(expr1)=SOUNDEX(expr2)

SPACE(N) Retourne une chaîne constituée de N espaces

```
mysql> SELECT SPACE(6);
-> ' '
```

SUBSTRING(str,pos), SUBSTRING(str FROM pos), SUBSTRING(str,pos,len), SUBSTRING(str FROM pos FOR len) Retourne une chaîne de len caractères de long de la chaîne str, à partir de la position pos. La syntaxe ANSI SQL92 utilise une variante de la fonction FROM. Cette fonction gère les caractères multi-octets.

```
mysql> SELECT SUBSTRING('Quadratically',5);
-> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
-> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
-> 'ratica'
```

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 suit le délimiteur final sera retourné. Cette fonction gère les caractères multi-octets.

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com&#39;, '.', 2);
-> 'www.mysql&#39;
mysql> SELECT SUBSTRING_INDEX('www.mysql.com&#39;, '.', -2);
-> 'mysql.com'
```

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. Cette fonction gère les caractères multi-octets.

```
mysql> SELECT TRIM(' bar ');
-> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
-> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
-> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
-> 'barx'
```

UCASE(str) UCASE() est un synonyme de UPPER()

UNCOMPRESS(string_to_uncompress) Décompresse une chaîne compressée avec COMPRESS(). Si l'argument n'est pas une valeur compressée, le résultat est NULL. Cette fonction requiert la bibliothèque zlib. Sinon, la valeur retournée est toujours NULL.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
-> 'any string'
mysql> SELECT UNCOMPRESS('any string');
-> NULL
```

UNCOMPRESSED_LENGTH(compressed_string) Retourne la taille de la chaîne avant compression.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
-> 30
```

UNHEX(str) Le contraire de HEX(string). C'est à dire, chaque paire de chiffres hexadécimaux sont interprétées comme des nombres, et sont convertis en un caractère représenté par le nombre. Le résultat est retournée sous forme de chaîne binaire.

```
mysql> SELECT UNHEX('4D7953514C');
-> 'MySQL'
mysql> SELECT 0x4D7953514C;
-> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
-> 'string'
mysql> SELECT HEX(UNHEX('1267'));
-> '1267'
```

UPPER(str) Retourne la chaîne str en majuscules, en fonction du jeu de caractères courant. Par défaut, c'est le jeu ISO-8859-1 Latin1. Cette fonction gère les caractères multi-octets.

mysql> SELECT UPPER('Hey');
-> 'HEY'

Fonctions dans SELECT et WHERE - Opérateurs de comparaison de chaînes de caractères

MySQL convertit automatiquement les nombres en chaînes et et vice-versa. Si vous devez convertir explicitement un nombre en chaîne, passez-le en argument de la fonction **CONCAT()** ou **CAST()**

expr LIKE pat [ESCAPE 'escape-char']

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

% 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

Pour tester la présence littérale d'un joker, précédez-le d'un caractère d'échappement. Si vous ne spécifiez pas le caractère d'échappement **ESCAPE**, le caractère "\" sera utilisé :

\% Remplace le caractère littéral '%'

_ Remplace le caractère littéral '_'

mysql> SELECT 'David!' LIKE 'David_';
-> 0
mysql> SELECT 'David_' LIKE 'David_';
-> 1

Pour spécifier un caractère d'échappement différent, utilisez la clause ESCAPE :

mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
-> 1

Les deux exemples suivants illustrent le fait que les comparaisons de chaînes de caractères ne sont pas sensibles à la casse à moins qu'une des opérandes soit une chaîne binaire.

mysql> SELECT 'abc' LIKE 'ABC';
-> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
-> 0

LIKE est également autorisé pour les expressions numériques. (C'est une extension MySQL à la norme ANSI SQL LIKE.)

mysql> SELECT 10 LIKE '1%';
-> 1

expr NOT LIKE pat [ESCAPE 'escape-char'] Équivalent à **NOT (expr LIKE pat [ESCAPE 'escape-char']**).

expr NOT REGEXP pat, expr NOT RLIKE pat Équivalent à **NOT (expr REGEXP pat)**.

expr REGEXP pat, expr RLIKE pat Effectue une recherche de chaîne avec l'expression régulière **pat**. Le masque peut être une expression régulière étendue. Retourne 1 si **expr** correspond au masque **pat**, sinon, retourne 0. **RLIKE** est un synonyme de **REGEXP**.

mysql> SELECT 'Monty!' REGEXP 'm%y%%';
-> 0
mysql> SELECT 'Monty!' REGEXP '.*';
-> 1

```
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
-> 1
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
-> 1 0
mysql> SELECT 'a' REGEXP '^[a-d]';
-> 1
```

STRCMP(expr1,expr2) STRCMP() retourne 0 si les chaînes sont identiques, -1 si la première chaîne est plus petite que la seconde et 1 dans les autres cas :

```
mysql> SELECT STRCMP('text', 'text2');
-> -1
mysql> SELECT STRCMP('text2', 'text');
-> 1
mysql> SELECT STRCMP('text', 'text');
-> 0
```

Fonctions dans SELECT et WHERE - Fonctions numériques

Opérations arithmétiques

+ Addition

- Soustraction

- Moins unaire.

***** Multiplication

/ Division

DIV Division entière

Fonctions mathématiques

Toutes les fonctions mathématiques retournent **NULL** en cas d'erreur.

ABS(X) Retourne la valeur absolue de X. Cette fonction est utilisable avec les valeurs issues des champs BIGINT.

```
mysql> SELECT ABS(2);
-> 2
mysql> SELECT ABS(-32);
-> 32
```

ACOS(X) Retourne l'arc cosinus de X, c'est à dire, la valeur de l'angle dont X est la cosinus. Retourne NULL si X n'est pas dans l'intervalle -1 - 1.

```
mysql> SELECT ACOS(1);
-> 0.000000
mysql> SELECT ACOS(1.0001);
-> NULL
mysql> SELECT ACOS(0);
-> 1.570796
```

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

```
mysql> SELECT ASIN(0.2);
-> 0.201358
mysql> SELECT ASIN('foo');
-> 0.000000
```

ATAN(X) Retourne l'arc tangente de **X**, c'est à dire, la valeur de l'angle dont la tangente est X.

```
mysql> SELECT ATAN(2);
-> 1.107149
mysql> SELECT ATAN(-2);
-> -1.107149
```

ATAN(Y,X), ATAN2(Y,X) Retourne l'arctangente des variables X et Y. Cela revient à calculer l'arctangente de Y / X, excepté que les signes des deux arguments servent à déterminer le quadrant du résultat

```
mysql> SELECT ATAN(-2,2);
-> -0.785398
mysql> SELECT ATAN2(PI(),0);
-> 1.570796
```

CEILING(X), CEIL(X Retourne la valeur entière supérieure de X. Notez que la valeur retournée sera de type BIGINT

```
mysql> SELECT CEILING(1.23);
-> 2
mysql> SELECT CEILING(-1.23);
-> -1
```

COS(X) Retourne le cosinus de X, où X est donné en radians.

```
mysql> SELECT COS(PI());
-> -1.000000
```

COT(X) Retourne la cotangente de X.

```
mysql> SELECT COT(12);
-> -1.57267341
mysql> SELECT COT(0);
-> NULL
```

CRC32(expr) Calcule la somme de contrôle et retourne un entier 32 bits non-signé. Le résultat est la valeur NULL si l'argument est NULL. L'argument attendu est une chaîne, et sera traité comme une chaîne s'il n'est pas du bon type.

```
mysql> SELECT CRC32('MySQL');
-> 3259397556
```

DEGREES(X) Retourne l'argument X, convertit de radians en degrés.

```
mysql> SELECT DEGREES(PI());
-> 180.000000
```

EXP(X) Retourne la valeur de e (la base des logarithmes naturels) élevé à la puissance X.

```
mysql> SELECT EXP(2);
-> 7.389056
mysql> SELECT EXP(-2);
-> 0.135335
```

FLOOR(X) Retourne la valeur entière inférieure de X. Notez que la valeur retournée sera de type BIGINT

```
mysql> SELECT FLOOR(1.23);
-> 1
mysql> SELECT FLOOR(-1.23);
-> -2
```

LN(X) Retourne le logarithme naturel de X (népérien). C'est un synonyme de la fonction LOG(X).

```
mysql> SELECT LN(2);
-> 0.693147
mysql> SELECT LN(-2);
-> NULL
```

LOG(X), LOG(B,X) Appelée avec un seul paramètre, cette fonction retourne le logarithme naturel (népérien) de X. Appelée avec deux paramètres, cette fonction retourne le logarithme naturel de X pour une base B arbitraire

```
mysql> SELECT LOG(2);
-> 0.693147
mysql> SELECT LOG(-2);
-> NULL
mysql> SELECT LOG(2,65536);
-> 16.000000
mysql> SELECT LOG(1,100);
-> NULL
```

LOG(B,X) est l'équivalent de LOG(X)/LOG(B).

LOG2(X) Retourne le logarithme en base 2 de X. Utile pour trouver combien de bits sont nécessaires pour stocker un nombre

```
mysql> SELECT LOG2(65536);
-> 16.000000
mysql> SELECT LOG2(-100);
-> NULL
```

MOD(N,M), N % M, N MOD M Modulo (équivalent de l'opérateur % dans le langage C). Retourne le reste de la division de N par M. Cette fonction ne pose pas de problèmes avec les BIGINT

```
mysql> SELECT MOD(234, 10);
-> 4
mysql> SELECT 253 % 7;
-> 1
mysql> SELECT MOD(29,9);
-> 2
```

PI() Retourne la valeur de pi. Par défaut, 5 décimales sont retournées, mais MySQL utilise la double précision pour pi.

```
mysql> SELECT PI();
-> 3.141593
mysql> SELECT PI()+0.000000000000000000;
-> 3.141592653589793116
```

POW(X,Y), POWER(X,Y) Retourne la valeur de X élevée à la puissance Y

```
mysql> SELECT POW(2,2);
-> 4.000000
mysql> SELECT POW(2,-2);
-> 0.250000
```

RADIANS(X) Retourne l'argument X, converti de degrés en radians.

```
mysql> SELECT RADIANS(90);
-> 1.570796
```

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. Vous ne pouvez pas utiliser une colonne de valeur RAND() dans une clause ORDER BY, parce que ORDER BY va évaluer la colonne plusieurs fois.

```
mysql> SELECT RAND();
-> 0.9233482386203
mysql> SELECT RAND(20);
-> 0.15888261251047
mysql> SELECT RAND(20);
-> 0.15888261251047
mysql> SELECT RAND();
-> 0.63553050033332
mysql> SELECT RAND();
-> 0.70100469486881
```

ROUND(X), ROUND(X,D) Retourne l'argument X, arrondi à un nombre à D décimales. Avec deux arguments, la valeur est arrondie avec D décimales. Si D vaut 0, le résultat n'aura ni de partie décimale, ni de séparateur de décimal. Notez que le comportement de l'opérateur **ROUND()**, lorsque l'argument est exactement entre deux entiers, dépend de la bibliothèque C active. Certaines arrondissent toujours à l'entier pair le plus proche, toujours vers le haut, toujours vers le bas, ou toujours vers zéro. Si vous avez besoin d'un certain type d'arrondissement, vous devez utiliser une fonction bien définie comme **TRUNCATE()** ou **FLOOR()**.

```
mysql> SELECT ROUND(-1.23);
-> -1
mysql> SELECT ROUND(-1.58);
-> -2
mysql> SELECT ROUND(1.58);
-> 2
mysql> SELECT ROUND(1.298, 1);
-> 1.3
```

mysql> SELECT ROUND(1.298, 0) ;
-> 1
mysql> SELECT ROUND(23.298, -1) ;
-> 20

SIGN(X) Retourne le signe de l'argument sous la forme -1, 0, ou 1, selon que X est négatif, zéro, ou positif.

mysql> SELECT SIGN(-32) ;
-> -1
mysql> SELECT SIGN(0) ;
-> 0
mysql> SELECT SIGN(234) ;
-> 1

SIN(X) Retourne le sinus de X, où X est donné en radians.

mysql> SELECT SIN(PI()) ;
-> 0.000000
SQRT(X) Retourne la racine carrée de X.
mysql> SELECT SQRT(4) ;
-> 2.000000
mysql> SELECT SQRT(20) ;
-> 4.472136

TAN(X) Retourne la tangente de X, où X est donné en radians.

mysql> SELECT TAN(PI()+1) ;
-> 1.557408

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. Notez que les nombres décimaux ne sont pas stockés exactement comme les nombres entiers , mais comme des valeurs doubles. Vous pouvez être dupés par le résultat suivant (Ce résultat est normal car 10.28 est actuellement stocké comme cela 10.2799999999999999) :

mysql> SELECT TRUNCATE(10.28*100,0) ;
-> 1027

Fonctions dans SELECT et WHERE - Dates et heures

ADDDATE(date,INTERVAL expr type), ADDDATE(expr,days) Lorsque utilisé avec la forme **INTERVAL**, **ADDDATE()** est un synonyme de **DATE_ADD()**. La fonction complémentaire **SUBDATE()** est un synonyme **DATE_SUB()**. La seconde syntaxe est utilisée si expr est une expression de type DATE ou DATETIME, et que days est un nombre de jour à ajouter à expr.

mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY) ;
-> '1998-02-02'
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY) ;
-> '1998-02-02'
mysql> SELECT ADDDATE('1998-01-02', 31) ;
-> '1998-02-02'

ADDTIME(expr,expr2) ADDTIME() Ajoute **expr2** à **expr** et retourne le résultat. **expr** est une expression de type **DATE** ou **DATETIME**, et **expr2** est une expression de type **TIME**.

mysql> SELECT ADDTIME("1997-12-31 23 :59 :59.999999", "1 1 :1 :1.000002") ;
-> '1998-01-02 01 :01 :01.000001'
mysql> SELECT ADDTIME("01 :00 :00.999999", "02 :00 :00.999998") ;
-> '03 :00 :01.999997'

CURDATE(), CURRENT_DATE Retourne la date courante au format **'YYYY-MM-DD'** ou **YYYYMMDD**, suivant le contexte numérique ou chaîne :

mysql> SELECT CURDATE() ;
-> '1997-12-15'
mysql> SELECT CURDATE() + 0 ;
-> 19971215

CURRENT_DATE, CURRENT_DATE() sont synonymes de **CURDATE()**.

CURTIME() Retourne l'heure courante au format **'HH :MM :SS'** or **HHMMSS** suivant le contexte numérique ou chaîne

mysql> SELECT CURTIME() ;
-> '23 :50 :26'
mysql> SELECT CURTIME() + 0 ;
-> 235026

CURRENT_TIME, CURRENT_TIME() sont synonymes de **CURTIME()**

CURRENT_TIMESTAMP, CURRENT_TIMESTAMP() sont synonymes de **NOW()**.

DATE(expr) Extrait la partie date de l'expression **expr** de type **DATE** ou **DATETIME**

mysql> SELECT DATE('2003-12-31 01 :02 :03') ;
-> '2003-12-31'

DATEDIFF(expr,expr2) Retourne le nombre de jours entre la date de début **expr** et la date de fin **expr2**. **expr** et **expr2** sont des expressions de type **DATE** ou **DATETIME**. Seule la partie **DATE** est utilisée dans le calcul

mysql> SELECT DATEDIFF('1997-12-31 23 :59 :59','1997-12-30') ;
-> 1
mysql> SELECT DATEDIFF('1997-11-31 23 :59 :59','1997-12-31') ;
-> -30

DATE_ADD(date,INTERVAL expr type), DATE_SUB(date,INTERVAL expr type) Ces fonctions effectuent des calculs arithmétiques sur les dates. La table suivante indique la signification des arguments **type** et **expr** :

```
type Valeur_____________Attendue expr Format
MICROSECOND_____________MICROSECONDS
SECOND__________________SECONDS
MINUTE__________________MINUTES
HOUR____________________HOURS
DAY_____________________DAYS
WEEK____________________WEEKS
MONTH ___________________MONTHS
QUARTER _________________QUARTERS
YEAR____________________YEARS
SECOND_MICROSECOND_____'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND_____'MINUTES.MICROSECONDS'
MINUTE_SECOND__________'MINUTES:SECONDS'
HOUR_MICROSECOND_______'HOURS.MICROSECONDS'
HOUR_SECOND____________'HOURS:MINUTES:SECONDS'
HOUR_MINUTE____________'HOURS:MINUTES'
DAY_MICROSECOND________'DAYS.MICROSECONDS'
DAY_SECOND_____________'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE_____________'DAYS HOURS:MINUTES'
DAY_HOUR_______________'DAYS HOURS'
YEAR_MONTH ____________'YEARS-MONTHS'
```

MySQL autorise tous les signes de ponctuation, comme délimiteur dans le format de **expr**. Ceux qui sont affichés dans la table sont des suggestions. Si l'argument **date** est une valeur **DATE** et que vos calculs impliquent des parties **YEAR**, **MONTH** et **DAY** (c'est à dire, sans partie horaire), le résultat sera de type **DATE**. Sinon, le résultat est de type **DATETIME**

mysql> SELECT '1997-12-31 23 :59 :59' + INTERVAL 1 SECOND ;
-> '1998-01-01 00 :00 :00'
mysql> SELECT INTERVAL 1 DAY + '1997-12-31' ;
-> '1998-01-01'
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND ;
-> '1997-12-31 23 :59 :59'
mysql> SELECT DATE_ADD('1997-12-31 23 :59 :59',
-> INTERVAL 1 SECOND) ;
-> '1998-01-01 00 :00 :00'

mysql> SELECT DATE_ADD('1997-12-31 23 :59 :59',
-> INTERVAL 1 DAY) ;
-> '1998-01-01 23 :59 :59'
mysql> SELECT DATE_ADD('1997-12-31 23 :59 :59',
-> INTERVAL '1 :1' MINUTE_SECOND) ;
-> '1998-01-01 00 :01 :00'
mysql> SELECT DATE_SUB('1998-01-01 00 :00 :00',
-> INTERVAL '1 1 :1 :1' DAY_SECOND) ;
-> '1997-12-30 22 :58 :59'
mysql> SELECT DATE_ADD('1998-01-01 00 :00 :00',
-> INTERVAL '-1 10' DAY_HOUR) ;
-> '1997-12-30 14 :00 :00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY) ;
-> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23 :59 :59.000002',
-> INTERVAL '1.999999' SECOND_MICROSECOND) ;
-> '1993-01-01 00 :00 :01.000001'

Si vous spécifiez un intervalle qui est trop court (il n'inclut pas toutes les parties d'intervalle attendues par type), **MySQL** suppose que vous avez omis les valeurs de gauche. Par exemple, si vous spécifiez un type type de **DAY_SECOND**, la valeur **expr** devrait contenir des jours, heures, minutes et secondes. Si vous fournissez une valeur de la forme **'1 :10'**, MySQL suppose que les jours et heures manquent, et que la valeur représente des minutes et secondes. En d'autres termes, **'1 :10' DAY_SECOND** est interprété comme **'1 :10' MINUTE_SECOND**. C'est similaire au comportement de MySQL avec les valeurs de type TIME, qui représente des durées plutôt que des horaires. Notez que si vous ajoutez ou soustrayez à une valeur de type DATE des horaires, le résultat sera automatiquement au format DATETIME

mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY) ;
-> '1999-01-02'
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR) ;
-> '1999-01-01 01 :00 :00'

Si vous utilisez des dates malformées, le résultat sera **NULL**. Si vous ajoutez des **MONTH**, **YEAR_MONTH** ou **YEAR**, et que le résultat a un jour du mois qui est au-delà de ce qui est possible dans le mois, le jour sera adapté au plus grand jour possible du mois. Par exemple

mysql> SELECT DATE_ADD('1998-01-30', interval 1 month) ;
-> '1998-02-28'

Notez que dans l'exemple précédent, le mot clé **INTERVAL** et le spécificateur type sont insensibles à la casse.

DATE_FORMAT(date,format)

Formate la date avec le format. Les spécificateurs suivants peuvent être utilisé dans la chaîne format

%% Un signe pourcentage littéral '%'.

%a Nom du jour de la semaine, en abrégé et en anglais (Sun..Sat)

%b Nom du mois, en abrégé et en anglais (Jan..Dec)

%c Mois, au format numérique (1..12)

%d Jour du mois, au format numérique (00..31)

%D Jour du mois, avec un suffixe anglais (1st, 2nd, 3rd, etc.)

%e Jour du mois, au format numérique (0..31)

%f Microsecondes (000000..999999)

%H Heure (00..23)

%h Heure (01..12)

%I Heure (01..12)

%i Minutes, au format numérique (00..59)

%j Jour de l'année (001..366)

%k Heure (0..23)

%l Heure (1..12)

%m Mois, au format numérique (01..12)

%M Nom du mois (January..December)

%p AM ou PM

%r Heures, au format 12 heures (hh :mm :ss [AP]M)

%s Secondes (00..59)

%s Secondes (00..59)

%T Heures, au format 24 heures (hh :mm :ss)

%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 '%x'

%W Nom du jour de la semaine (Sunday..Saturday)

%w Numéro du jour de la semaine (0=Sunday..6=Saturday)

%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'

%y Année, au format numérique, sur 2 chiffres

%Y Année, au format numérique, sur 4 chiffres

Tous les autres caractères sont simplement copiés dans le résultat sans interprétation

```
mysql> SELECT DATE_FORMAT('1997-10-04 22 :23 :00', '%W %M %Y');
-> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22 :23 :00', '%H :%i :%s');
-> '22 :23 :00'
mysql> SELECT DATE_FORMAT('1997-10-04 22 :23 :00',
'%D %y %a %d %m %b %j');
-> '4th 97 Sat 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22 :23 :00',
'%H %k %I %r %T %S %w');
-> '22 22 10 10 :23 :00 PM 22 :23 :00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
-> '1998 52'
```

DAY(date) DAY() est un synonyme de **DAYOFMONTH()**.

DAYNAME(date) Retourne le nom du jour de la semaine de date

```
mysql> SELECT DAYNAME('1998-02-05');
-> 'Thursday'
```

DAYOFMONTH(date) Retourne le jour de la date date, dans un intervalle de 1 à 31

```
mysql> SELECT DAYOFMONTH('1998-02-03');
-> 3
```

DAYOFWEEK(date) Retourne l'index du jour de la semaine : pour date (1 = Dimanche, 2 = Lundi, ... 7 = Samedi). Ces index correspondent au standard ODBC

```
mysql> SELECT DAYOFWEEK('1998-02-03');
-> 3
DAYOFYEAR(date) Retourne le jour de la date date, dans un intervalle de 1 à 366 :
mysql> SELECT DAYOFYEAR('1998-02-03');
-> 34
```

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.

```
mysql> SELECT EXTRACT(YEAR FROM "1999-07-02");
-> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM "1999-07-02 01 :02 :03");
-> 199907
```

mysql> SELECT EXTRACT(DAY_MINUTE FROM "1999-07-02 01 :02 :03") ;
-> 20102
mysql> SELECT EXTRACT(MICROSECOND FROM "2003-01-02 10 :30 :00.00123") ;
-> 123

FROM_DAYS(N) Retourne la date correspondant au nombre de jours (N) depuis la date 0. **FROM_DAYS()** n'est pas fait pour travailler avec des dates qui précèdent l'avènement du calendrier Grégorien (1582), car elle ne prend pas en compte les jours perdus lors du changement de calendrier.

mysql> SELECT FROM_DAYS(729669) ;
-> '1997-10-07'

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 utilisé dans un contexte numérique ou de chaîne.

mysql> SELECT FROM_UNIXTIME(875996580) ;
-> '1997-10-04 22 :23 :00'
mysql> SELECT FROM_UNIXTIME(875996580) + 0 ;
-> 19971004222300
Si format est donné, le résultat est formaté en fonction de la chaîne format. format peut contenir les mêmes options de format que celles utilisées par **DATE_FORMAT()**

mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
-> '%Y %D %M %h :%i :%s %x') ;
-> '2003 6th August 06 :22 :58 2003'

GET_FORMAT(DATE | TIME | TIMESTAMP, 'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL') Retourne une chaîne de format. Cette fonction est pratique lorsqu'elle est utilisée avec les fonctions **DATE_FORMAT()** et **STR_TO_DATE()**. Les trois valeurs possibles pour le premier argument, et les cinq valeurs possible pour le second argument donnent 15 formats d'affichage (pour les options utilisées, voyez la table de la fonction **DATE_FORMAT())**

Appel fonction / Résultat

GET_FORMAT(DATE,'USA') '%m.%d.%Y'

GET_FORMAT(DATE,'JIS') '%Y-%m-%d'

GET_FORMAT(DATE,'ISO') '%Y-%m-%d'

GET_FORMAT(DATE,'EUR') '%d.%m.%Y'

GET_FORMAT(DATE,'INTERNAL') '%Y%m%d'

GET_FORMAT(TIMESTAMP,'USA') '%Y-%m-%d-%H.%i.%s'

GET_FORMAT(TIMESTAMP,'JIS') '%Y-%m-%d %H :%i :%s'

GET_FORMAT(TIMESTAMP,'ISO') '%Y-%m-%d %H :%i :%s'

GET_FORMAT(TIMESTAMP,'EUR') '%Y-%m-%d-%H.%i.%s'

GET_FORMAT(TIMESTAMP,'INTERNAL') '%Y%m%d%H%i%s'

GET_FORMAT(TIME,'USA') '%h :%i :%s %p'

GET_FORMAT(TIME,'JIS') '%H :%i :%s'

GET_FORMAT(TIME,'ISO') '%H :%i :%s'

GET_FORMAT(TIME,'EUR') '%H.%i.%S'

GET_FORMAT(TIME,'INTERNAL') '%H%i%s'

mysql> SELECT DATE_FORMAT('2003-10-03', GET_FORMAT(DATE, 'EUR')
-> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003', GET_FORMAT(DATE, 'USA'))
-> 2003-10-31

HOUR(time) Retourne le nombre d'heures pour l'heure time, dans un intervalle de 0 à 23

mysql> SELECT HOUR('10 :05 :03') ;
-> 10
Cependant, l'intervalle des valeurs **TIME** est bien plus grand, et donc, **HOUR** peut retourner des valeurs plus grandes que 23

mysql> SELECT HOUR('272 :59 :59') ;
-> 272

LAST_DAY(date) Prend une valeur de format **DATE** ou **DATETIME**, et retourne le dernier jour du mois correspondant. Retourne NULL si l'argument est invalide.

```
mysql> SELECT LAST_DAY('2003-02-05'), LAST_DAY('2004-02-05');
-> '2003-02-28', '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01 :01 :01');
-> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
-> NULL
```

LOCALTIME, LOCALTIME() sont synonymes de **NOW()**.

LOCALTIMESTAMP, LOCALTIMESTAMP() sont synonymes de **NOW()**.

MAKEDATE(year,dayofyear) Retourne une valeur de format **DATE**, à partir d'une année et du numéro de jour. dayofyear doit être plus grand que 0 ou le résultat sera NULL.

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
-> '2001-01-31', '2001-02-01'
mysql> SELECT MAKEDATE(2001,365), MAKEDATE(2004,365);
-> '2001-12-31', '2004-12-30'
mysql> SELECT MAKEDATE(2001,0);
-> NULL
```

MAKETIME(hour,minute,second) Retourne une valeur de format **TIME**, calculée à partir des arguments hour, minute et second.

```
mysql> SELECT MAKETIME(12,15,30);
-> '12 :15 :30'
```

MICROSECOND(expr) Retourne le nombre de microsecondes dans l'expression de type **TIME** ou **DATETIME expr**, sous la forme d'un nombre entre 0 et 999999.

```
mysql> SELECT MICROSECOND('12 :00 :00.123456');
-> 123456
mysql> SELECT MICROSECOND('1997-12-31 23 :59 :59.000010');
-> 10
```

MINUTE(time) Retourne le nombre de minutes pour l'heure time, dans un intervalle de 0 à 59

```
mysql> SELECT MINUTE('98-02-03 10 :05 :03');
-> 5
```

MONTH(date) Retourne le numéro du mois de la date date, dans un intervalle de 1 à 12

```
mysql> SELECT MONTH('1998-02-03');
-> 2
```

MONTHNAME(date) Retourne le nom du mois de la date

```
mysql> SELECT MONTHNAME("1998-02-05");
-> 'February'
```

NOW() Retourne la date courante au format **'YYYY-MM-DD HH :MM :SS'** ou **YYYYMMDDHHMMSS**, suivant le contexte numérique ou chaîne

```
mysql> SELECT NOW();
-> '1997-12-15 23 :50 :26'
mysql> SELECT NOW() + 0;
-> 19971215235026
```

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

```
mysql> SELECT PERIOD_ADD(9801,2);
-> 199803
```

PERIOD_DIFF(P1,P2) Retourne le nombre de mois entre les périodes P1 et P2. P1 et P2 doivent être au format YYMM ou YYYYMM. Notez que les arguments P1 et P2 ne sont pas de type date

```
mysql> SELECT PERIOD_DIFF(9802,199703);
-> 11
```

QUARTER(date) Retourne le numéro du trimestre de la date date, dans un intervalle de 1 à 4

```
mysql> SELECT QUARTER('98-04-01');
-> 2
```

SECOND(time) Retourne le nombre de secondes pour l'heure time, dans un intervalle de 0 à 59

```
mysql> SELECT SECOND('10 :05 :03');
-> 3
```

SEC_TO_TIME(seconds) Retourne l'argument seconds, convertit en heures, minutes et secondes au format **'HH :MM :SS'** ou **HHMMSS**, suivant le contexte numérique ou chaîne

```
mysql> SELECT SEC_TO_TIME(2378);
-> '00 :39 :38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
-> 3938
```

STR_TO_DATE(str,format) Cette fonction est l'inverse de la fonction **DATE_FORMAT()**. Elle prend la chaîne str, et une chaîne de format format, puis retourne une valeur **DATETIME**. Les valeurs de type **DATE**, **TIME** ou **DATETIME** contenues dans la chaîne str doivent être au format spécifié. Pour les options qui sont utilisables dans la chaîne format, voyez la table dans la description de la fonction **DATE_FORMAT()**. Tous les autres caractères sont utilisés littéralement, et ne seront pas interprétés. Si str contient une valeur illégale, **STR_TO_DATE()** retourne NULL.

```
mysql> SELECT STR_TO_DATE('03.10.2003 09.20', '%d.%m.%Y %H.%i')
-> 2003-10-03 09 :20 :00
mysql> SELECT STR_TO_DATE('10rap', '%crap')
-> 0000-10-00 00 :00 :00
mysql> SELECT STR_TO_DATE('2003-15-10 00 :00 :00', '%Y-%m-%d %H :%i :%s')
-> NULL
```

SUBDATE(date,INTERVAL expr type), SUBDATE(expr,days) Lorsqu'elle est utilisée avec la forme **INTERVAL** du second argument, **SUBDATE()** est synonyme **DATE_SUB()**

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
-> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02 12 :00 :00', 31);
-> '1997-12-02 12 :00 :00'
```

SUBTIME(expr,expr2) SUBTIME() Soustrait **expr2** de **expr** et retourne le résultat. expr est une expression de format **DATE** ou **DATETIME** et **expr2** est une expression de type **TIME**.

```
mysql> SELECT SUBTIME("1997-12-31 23 :59 :59.999999", "1 1 :1 :1.000002");
-> '1997-12-30 22 :58 :58.999997'
mysql> SELECT SUBTIME("01 :00 :00.999999", "02 :00 :00.999998");
-> '-00 :59 :59.999999'
```

SYSDATE() est un synonyme de NOW().

TIME(expr) Extrait la partie horaire de l'expression expr, de type **TIME** ou **DATETIME**.

```
mysql> SELECT TIME('2003-12-31 01 :02 :03');
-> '01 :02 :03'
mysql> SELECT TIME('2003-12-31 01 :02 :03.000123');
-> '01 :02 :03.000123'
```

TIMEDIFF(expr,expr2) TIMEDIFF() retourne la durée entre l'heure de début expr et l'heure de fin expr2. expr et expr2 sont des expressions de type TIME ou DATETIME, et doivent être de même type.

```
mysql> SELECT TIMEDIFF('2000 :01 :01 00 :00 :00', '2000 :01 :01 00 :00 :00.000001');
-> '-00 :00 :00.000001'
mysql> SELECT TIMEDIFF('1997-12-31 23 :59 :59.000001','1997-12-30 01 :01 :01.000002');
-> '46 :58 :57.999999'
```

TIMESTAMP(expr), TIMESTAMP(expr,expr2) Avec un seul argument, retourne l'expression expr de type DATE ou DATETIME sous la forme d'une valeur DATETIME. Avec deux arguments, ajouter l'expression expr2 à l'expression expr et retourne le résultat au format DATETIME.

mysql> SELECT TIMESTAMP('2003-12-31');
-> '2003-12-31 00 :00 :00'
mysql> SELECT TIMESTAMP('2003-12-31 12 :00 :00','12 :00 :00');
-> '2004-01-01 00 :00 :00'

TIMESTAMPADD(interval,int_expr,datetime_expr) Ajoute l'expression entière **int_expr** à l'expression **datetime_expr** au format **DATE** ou **DATETIME**. L'unité de **int_expr** est donnée avec l'argument **interval**, qui peut être l'une des valeurs suivantes : **FRAC_SECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, ou YEAR**. La valeur interval peut être spécifiée, en utilisant un des mots-clé cités, ou avec le préfixe **SQL_TSI_**. Par exemple, **DAY** et **SQL_TSI_DAY** sont tous les deux valides.

mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
-> '2003-01-02 00 :01 :00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
-> '2003-01-09'

TIMESTAMPDIFF(interval,datetime_expr1,datetime_expr2) Retourne la différence entière entre les expressions **datetime_expr1** et **datetime_expr2**, de format **DATE** et **DATETIME**. L'unité du résultat est donné par l'argument interval. Les valeurs légales de interval sont les mêmes que pour la fonction **TIMESTAMPADD()**.

mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
-> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
-> -1

TIME_FORMAT(time,format) Cette fonction est utilisée exactement comme la fonction **DATE_FORMAT()** ci-dessus, mais la chaîne format ne doit utiliser que des spécificateurs d'heures, qui gèrent les heures, minutes et secondes. Les autres spécificateurs génèreront la valeur NULL ou 0. Si la valeur time contient une valeur d'heure qui est plus grande que 23, les formats %H et %k produiront une valeur qui est hors de l'intervalle 0..23. L'autre format d'heure produira une heure modulo 12

mysql> SELECT TIME_FORMAT('100 :00 :00', '%H %k %h %I %l');
-> '100 100 04 04 4'

TIME_TO_SEC(time) Retourne l'argument time, convertit en secondes

mysql> SELECT TIME_TO_SEC('22 :23 :00');
-> 80580
mysql> SELECT TIME_TO_SEC('00 :39 :38');
-> 2378

TO_DAYS(date) Retourne le nombre de jours depuis la date 0 jusqu'à la date date. **TO_DAYS()** n'est pas fait pour travailler avec des dates qui précèdent l'avènement du calendrier Grégorien (1582), car elle ne prend pas en compte les jours perdus lors du changement de calendrier.

mysql> SELECT TO_DAYS(950501);
-> 728779
mysql> SELECT TO_DAYS('1997-10-07');
-> 729669

UNIX_TIMESTAMP(), UNIX_TIMESTAMP(date) Lorsqu'elle est appelé sans argument, cette fonction retourne un timestamp Unix (nombre de secondes depuis '1970-01-01 00 :00 :00' GMT). Si **UNIX_TIMESTAMP()** est appelé 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. Lorsque **UNIX_TIMESTAMP** est utilisé sur une colonne de type TIMESTAMP, la fonction reçoit directement la valeur, sans conversion explicite. Si vous donnez à UNIX_TIMESTAMP() une date hors de son intervalle de validité, elle retourne 0. Si vous voulez soustraire une colonne de type UNIX_TIMESTAMP(), vous devez sûrement vouloir un résultat de type entier signé.

mysql> SELECT UNIX_TIMESTAMP();
-> 882226357
mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22 :23 :00');
-> 875996580

UTC_DATE, UTC_DATE() Retourne la date **UTC** courante au format **'YYYY-MM-DD'** ou **YYYYMMDD** suivant le contexte numérique ou chaîne

mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
-> '2003-08-14', 20030814

UTC_TIME, UTC_TIME() Retourne l'heure UTC courante au format 'HH :MM :SS' or HHMMSS suivant le contexte numérique ou chaîne

mysql> SELECT UTC_TIME(), UTC_TIME() + 0 ;
-> '18 :07 :53', 180753

UTC_TIMESTAMP, UTC_TIMESTAMP() Retourne l'heure et la date UTC courante au format 'YYYY-MM-DD HH :MM :SS' or YYYYMMDDHHMMSS suivant le contexte numérique ou chaîne

mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0 ;
-> '2003-08-14 18 :08 :04', 20030814180804

WEEK(date [,mode]) Avec un seul argument, retourne le numéro de la semaine dans l'année de la date spécifiée, 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. Lorsque l'argument mode est omis, la valeur de la variable default_week_format est utilisé. Voici un tableau explicatif sur le fonctionnement du second argument

Valeur / Signification

0 La semaine commence le Sunday ;l'intervalle de valeur de retour va de 0 à !2 ; la semaine 1 est la première semaine de l'année

1 La semaine commence le Monday ;l'intervalle de valeur de retour va de 0 à !2 ; la semaine 1 est la première semaine de l'année qui a plus de trois jours

2 La semaine commence le Sunday ;l'intervalle de valeur de retour va de 1 à !2 ; la semaine 1 est la première semaine de l'année

3 La semaine commence le Monday ;l'intervalle de valeur de retour va de 1 à !2 ; la semaine 1 est la première semaine de l'année qui a plus de trois jours

4 La semaine commence le Sunday ;l'intervalle de valeur de retour va de 0 à !2 ; la semaine 1 est la première semaine de l'année qui a plus de trois jours

5 La semaine commence le Monday ;l'intervalle de valeur de retour va de 0 à !2 ; la semaine 1 est la première semaine de l'année

6 La semaine commence le Sunday ;l'intervalle de valeur de retour va de 1 à !2 ; la semaine 1 est la première semaine de l'année qui a plus de trois jours

7 La semaine commence le Monday ;l'intervalle de valeur de retour va de 1 à !2 ; la semaine 1 est la première semaine de l'anné

mysql> SELECT WEEK('1998-02-20') ;
-> 7
mysql> SELECT WEEK('1998-02-20',0) ;
-> 7
mysql> SELECT WEEK('1998-02-20',1) ;
-> 8
mysql> SELECT WEEK('1998-12-31',1) ;
-> 53
Si vous préférez que le résultat soit calculé en fonction de l'année qui contient le premier jour de la semaine de la date utilisée en argument, vous devriez utiliser les valeurs 2, 3, 6, or 7 de l'argument mode.

mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0) ;
-> 2000, 0
mysql> SELECT WEEK('2000-01-01',2) ;
-> 52
Alternativement, utilisez la fonction YEARWEEK()
mysql> SELECT YEARWEEK('2000-01-01') ;
-> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2) ;
-> '52'

WEEKDAY(date) Retourne l'index du jour de la semaine, avec la conversion suivante : date (0 = Lundi, 1 = Mardi, ... 6 = Dimanche).

mysql> SELECT WEEKDAY('1997-10-04 22 :23 :00') ;
-> 5
mysql> SELECT WEEKDAY('1997-11-05') ;
-> 2

WEEKOFYEAR(date) Retourne le numéro de semaine dans l'année, sous forme d'un nombre compris entre 1 et 53.

mysql> SELECT WEEKOFYEAR('1998-02-20');
-> 8

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

mysql> SELECT YEAR('98-02-03');
-> 1998
mysql> SELECT YEAR('98-02-03');
-> 1998

YEARWEEK(date), YEARWEEK(date,start) Retourne l'année et la semaine d'une date. L'argument start fonctionne exactement comme l'argument start de la fonction WEEK(). Notez que l'année dans le résultat peut être différente de l'année passée en argument, pour la première et la dernière semaine de l'année. Notez que le numéro de semaine est différent de celui que la fonction WEEK() retourne (0) pour les arguments optionnels 0 ou 1, comme WEEK() puis retourne la semaine dans le contexte de l'année.

mysql> SELECT YEARWEEK('1987-01-01');
-> 198653

Fonctions dans SELECT et WHERE - Fonctions sur les bits

MySQL utilise l'arithmétique des **BIGINT** (64-bits) pour les opérations sur les bits. Ces opérateurs travaillent donc sur 64 bits.

| OU bit-à-bit (OR). Le résultat est un entier de 64 bits non signé.

mysql> SELECT 29 | 15;
-> 31

& ET bit-à-bit (AND). Le résultat est un entier de 64 bits non signé.

mysql> SELECT 29 & 15;
-> 13

^ XOR bit-à-bit. Le résultat est un entier de 64 bits non signé.

mysql> SELECT 1 ^1;
-> 0
mysql> SELECT 1 ^0;
-> 1
mysql> SELECT 11 ^3;
-> 8

« Décale les bits de l'entier (BIGINT) sur la gauche. Le résultat est un entier de 64 bits non signé.

mysql> SELECT 1 « 2;
-> 4

» Décale les bits de l'entier (BIGINT) sur la droite. Le résultat est un entier de 64 bits non signé.

mysql> SELECT 4 » 2;
-> 1

Inverse tous les bits. Le résultat est un entier de 64 bits non signé.

mysql> SELECT 5 & 1;
-> 4

BIT_COUNT(N) Retourne le nombre de bits non nuls de l'argument N

mysql> SELECT BIT_COUNT(29);
-> 4

Fonctions dans SELECT et WHERE - Fonctions de chiffrement

AES_ENCRYPT(str,key_str), AES_DECRYPT(crypt_str,key_str) Ces fonctions permettent le chiffrement/déchiffrement de données utilisant l'algorithme AES. Une clé de 128 bits est utilisé pour le chiffrement. Les arguments peuvent être de n'importe quelle taille. Si l'un des arguments est NULL, le résultat de cette fonction sera NULL.

Vous pouvez utiliser les fonctions AES pour stocker des données sous une forme chiffrées en modifiant vos requêtes
INSERT INTO t VALUES (1,AES_ENCRYPT("text","password"));
Vous pouvez obtenir encore plus de sécurité en évitant de transférer la clé pour chaque requête, en la stockant dans une variable sur le serveur au moment de la connexion
SELECT @password :="my password";
INSERT INTO t VALUES (1,AES_ENCRYPT("text",@password));

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()**.

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.

DES_DECRYPT(crypt_str[,key_str]) Déchiffre une chaîne chiffrée à l'aide de la fonction **DES_ENCRYPT()**. Notez que cette fonction fonctionne uniquement si vous avez configuré MySQL avec le support SSL. Si l'argument **key_string** n'est pas donné, la fonction **DES_DECRYPT()** examine le premier bit de la chaîne chiffrée pour déterminer le numéro de clé DES utilisé pour chiffrer la chaîne originale, alors la clé est lu dans le fichier **des-key-file** pour déchiffrer le message. Pour pouvoir utiliser cela, l'utilisateur doit avoir le privilège **SUPER**. Si vous passé l'argument **key_string** à cette fonction, cette chaîne est utilisée comme clé pour déchiffrer le message. Si la chaîne **string_to_decrypt** ne semble pas être une chaîne chiffrée, MySQL retournera la chaîne **string_to_decrypt**. Si une erreur survient, cette fonction retourne NULL.

DES_ENCRYPT(str[,(key_num|key_str)]) Chiffre la chaîne avec la clé donnée en utilisant l'algorithme DES. Notez que cette fonction fonctionne uniquement si vous avez configuré MySQL avec le support SSL. La clé de hachage utilisée est choisie en suivant les recommandations suivantes

Un seul argument La première clé de des-key-file est utilisée.

Un numéro de clé Le numéro de la clé donnée (0-9) de des-key-file est utilisée.

Une chaîne La chaîne donnée key_string doit être utilisé pour chiffrer string_to_encrypt.

La chaîne retournée doit être une chaîne binaire où le premier caractère doit être **CHAR(128 | key_number)**.

Le nombre 128 a été ajouté pour reconnaître facilement une clé de hachage. Si vous utilisez une chaîne comme clé, **key_number** doit être 127.

Si une erreur survient, la fonction retournera NULL.

La longueur de la chaîne de résultat doit être : **new_length= org_length + (8-(org_length % 8))+1.**

Chaque **key_number** doit être un nombre dans l'intervalle 0 à 9. Les lignes dans le fichier peuvent être dans n'importe quel ordre. **des_key_string** est la chaîne qui permettra le chiffrement du message. Entre le nombre et la clé, il doit y avoir au moins un espace. La première clé est la clé par défaut qui sera utilisé si vous ne spécifiez pas d'autres clés en arguments de la fonction **DES_ENCRYPT()**.

Vous pouvez demander à MySQL de lire de nouvelles valeurs de clé dans le fichier de clés avec la commande **FLUSH DES_KEY_FILE**. Cela requière le privilège **Reload_priv**.

Un des bénéfices d'avoir une liste de clés par défaut est que cela donne aux applications la possibilité de regarder l'existence de la valeur chiffrée de la colonne, sans pour autant donner la possibilité à l'utilisateur final de déchiffrer ces valeurs.

mysql> SELECT customer_address FROM customer_table WHERE
crypted_credit_card = DES_ENCRYPT("credit_card_number");

ENCRYPT(str[,salt]) Chiffre la chaîne str en utilisant la fonction crypt(). L'argument salt doit être une chaîne de deux caractères.

mysql> SELECT ENCRYPT("hello");
-> 'VxuFAJXVARROc'
Si la fonction **crypt()** n'est pas disponible sur votre système, la fonction **ENCRYPT()** retournera toujours NULL. La fonction **ENCRYPT()** conserve uniquement les 8 premiers caractères de la chaîne str, au moins, sur certains système. Le comportement exact est directement déterminé par la fonction système crypt() sous-jacente.

MD5(str) Calcul 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. C'est l'algorithme RSA ("RSA Data Security, Inc. MD5 Message-Digest Algorithm").

mysql> SELECT MD5("testing");
-> 'ae2b1fca515949e5d54fb22b8ed95575'

OLD_PASSWORD(str) retourne la valeur pre-4.1 de PASSWORD()

PASSWORD(str) Calcule un mot de passe chiffré à partir de la chaîne str. C'est cette fonction qui est utilisé pour chiffrer les mots de passes MySQL pour être stockés dans une colonne de type Password de la table user. Le chiffrage par PASSWORD() n'est pas réversible. PASSWORD() n'est pas un chiffrage comparable à la fonction de chiffrage Unix. Voir ENCRYPT(). La fonction PASSWORD() est utilisée durant l'identification au serveur MYSQL. Il est recommandé de ne pas l'utiliser pour vos applications. Utilisez plutôt MD5() ou SHA1().

mysql> SELECT PASSWORD('badpwd');
-> '7f84554057dd964b'

SHA1(str), SHA(str) Calcule la somme de vérification SHA1 160 bits 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. La fonction SHA1() a été ajoutée dans la version 4.0.2 de MySQL et peut être considérée comme une méthode de cryptographie plus sûre que la fonction MD5(). La fonction SHA() est un alias de la fonction SHA1()

Fonctions dans SELECT et WHERE - Fonctions d'information

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'afficher la durée d'exécution de la requête. Le temps affiché est le temps côté client, et non pas les ressources processeurs consommées. il est conseillé d'utiliser **BENCHMARK()** plusieurs fois de suite pour interpréter un résultat, en dehors de charges ponctuelles sur le serveur.

mysql> SELECT BENCHMARK(1000000,ENCODE("bonjour","au revoir"));

CHARSET(str) Retourne le jeu de caractères de la chaîne argument.

mysql> SELECT CHARSET('abc');
-> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
-> 'utf8'
mysql> SELECT CHARSET(USER());
-> 'utf8'

Les valeurs retournées possibles sont (Les valeurs les plus faibles ont la plus haute priorité) :

0 Collation explicite

1 Par de collation

2 Collation implicite

3 Coercible

COLLATION(str) Retourne la collation du jeu de caractères de la chaîne argument

mysql> SELECT COLLATION('abc');
-> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
-> 'utf8_general_ci'

CONNECTION_ID() Retourne l'identifiant de connexion courant (thread_id). Chaque connexion a son propre identifiant unique

mysql> SELECT CONNECTION_ID();
-> 23786

CURRENT_USER() Retourne le nom d'utilisateur et le nom d'hôte de la session courante. Cette valeur correspond au compte qui a été utilisé durant l'identification auprès du serveur. Cela peut être différent des valeurs de USER().

mysql> SELECT USER();
-> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044 : Access denied for user : '@localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();

-> '@localhost'

Cet exemple montre que même si le client a indiqué le nom d'utilisateur davida (comme mentionné par la fonction USER()), le serveur a identifié le client comme un utilisateur anonyme (comme indiqué par la fonction CURRENT_USER()). Une situation qui arrive s'il n'y a aucun compte de listé dans les tables de droits pour davida.

DATABASE() Retourne le nom de la base de données courante. Si aucune base de données n'a été sélectionnée, DATABASE() retourne une chaîne vide. A partir de la version 4.1.1, elle retourne NULL.

mysql> SELECT DATABASE() ;
-> 'test'

FOUND_ROWS() Une commande **SELECT** peut inclure une clause **LIMIT** pour restreindre le nombre de lignes qui sera retourné par le client. Dans certains cas, il est mieux de savoir combien de lignes une commande aurait retourné, sans la clause **LIMIT**, mais sans lancer à nouveau le calcul. Pour cela, ajoutez l'option **SQL_CALC_FOUND_ROWS** dans la commande **SELECT**, puis appelez **FOUND_ROWS()** après.

mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
-> WHERE id > 100 LIMIT 10 ;
mysql> SELECT FOUND_ROWS() ;

Le second **SELECT** retourne un nombre indiquant combien de lignes le premier **SELECT** aurait retourné s'il n'avait pas été écrit avec une clause **LIMIT**. Notez que si vous utilisez **SELECT SQL_CALC_FOUND_ROWS ...**, MySQL calcule toutes les lignes dans la liste des résultats. Ainsi, c'est plus rapide si vous n'utilisez pas de clause **LIMIT** et que la liste des résultats n'a pas besoin d'être envoyée au client. Si la commande **SELECT** précédente n'inclut pas l'option **SQL_CALC_FOUND_ROWS**, alors **FOUND_ROWS()** pourrait retourner une valeur différente suivant que **LIMIT** est utilisé ou pas.

SQL_CALC_FOUND_ROWS et **FOUND_ROWS()** peuvent être pratiques dans des situations où vous devez limiter le nombre de lignes que la requête retourne, mais que vous devez tout de même connaître le nombre de lignes total, sans exécuter une seconde requête. Un exemple classique est un script web qui présente des résultats de recherche. En utilisant **FOUND_ROWS()**, vous connaîtrez facilement le nombre de lignes de résultat. L'utilisation de **SQL_CALC_FOUND_ROWS** et **FOUND_ROWS()** est plus complexe pour les requêtes **UNION** que pour les commandes **SELECT** simples, car **LIMIT** peut intervenir plusieurs fois dans une commande UNION. Elle sera appliquée à différentes commandes **SELECT** de la commande **UNION**, ou globalement à l'UNION.

Le but de **SQL_CALC_FOUND_ROWS** pour **UNION** est de retourner le nombre de lignes qui aurait été retourné sans la clause globale **LIMIT**. Les conditions d'utilisation de **SQL_CALC_FOUND_ROWS** avec **UNION** sont

- Le mot clé **SQL_CALC_FOUND_ROWS** doit apparaître dans le premier **SELECT** de **l'UNION**.
- La valeur de **FOUND_ROWS()** est exactement la même que si **UNION ALL** était utilisé. Si **UNION** sans **ALL** est utilisé, des réductions de doublons surviendront, et la valeur de **FOUND_ROWS()** sera approximative.
- Si aucune clause **LIMIT** n'est présente dans **UNION**, **SQL_CALC_FOUND_ROWS** est ignoré et retourne le nombre de lignes dans la table temporaire créé durant le traitement de l'**UNION**.

LAST_INSERT_ID(), LAST_INSERT_ID(expr) Retourne le dernier identifiant automatiquement généré par une colonne AUTO_INCREMENT.

mysql> SELECT LAST_INSERT_ID() ;
-> 195

Le dernier **ID** généré est conservé par le serveur pour chaque connexion. Un autre client ne la modifiera donc pas, même s'ils génèrent une autre valeur **AUTO_INCREMENT** de leur coté. Ce comportement permet de s'assurer que les actions des autres clients ne perturbe pas les actions du client en cours. La valeur de **LAST_INSERT_ID()** ne sera pas modifiée non plus si vous modifiez directement la valeur d'une colonne **AUTO_INCREMENT** avec une valeur simple (c'est à dire, une valeur qui n'est ni NULL, ni 0).

Si vous insérez plusieurs lignes au même moment avec une requête **INSERT**, **LAST_INSERT_ID()** retourne la valeur de la première ligne insérée. La raison à cela est que cela rend possible la reproduction facilement la même requête **INSERT** sur d'autres serveurs. Si vous utilisez une commande **INSERT IGNORE** et que la ligne est ignorée, le compteur **AUTO_INCREMENT** sera malgré tout incrémenté, et **LAST_INSERT_ID()** retournera une nouvelle valeur. Si **expr** est donnée en argument à la fonction **LAST_INSERT_ID()**, alors la valeur de l'argument sera retourné par la fonction et sera enregistré comme étant la prochaine valeur retournée par **LAST_INSERT_ID()**. Cela peut être utilisé pour simuler des séquences.

Commencez par créer la table suivante

mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
Utilisez cette table pour générer des séquences de nombre comme ceci
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
La commande **UPDATE** incrémente le compteur de séquence, et fait que le prochain appel à **LAST_INSERT_ID()** va retourner une valeur différente. La commande **SELECT** lit cette valeur. La fonction C **mysql_insert_id()** peut aussi être utilisée pour lire la valeur. Vous pouvez générer des séquences sans appeler la fonction **LAST_INSERT_ID()**, mais l'utilité d'utiliser cette fonction cette fois ci est que la valeur ID est gérée par le serveur comme étant la dernière valeur générée automatiquement. (sécurité multi-utilisateur). Vous pouvez retrouver le nouvelle ID tout comme vous pouvez lire n'importe quelle valeur **AUTO_INCREMENT** dans MySQL. Par exemple, la fonction **LAST_INSERT_ID()** (sans argument) devrait retourner la nouvelle ID.

La fonction C de l'API **mysql_insert_id()** peut être également utilisée pour trouver cette valeur. Notez que la fonction **mysql_insert_id()** est incrémentée uniquement après des requêtes **INSERT** et **UPDATE**, donc, vous ne pouvez pas utiliser la fonction C de l'API pour trouver la valeur de **LAST_INSERT_ID(expr)** après avoir exécuté d'autres types de requêtes, comme **SELECT** ou bien **SET**.

SESSION_USER() est un synonyme de USER().

SYSTEM_USER() est un synonyme de USER().

USER() Retourne le nom d'utilisateur et le nom d'hôte courant MySQL. La valeur indique le nom d'utilisateur qui a été spécifié lors de l'identification avec le serveur MySQL, et l'hôte client avec lequel il est connecté

mysql> SELECT USER();
-> 'davida@localhost'

VERSION() Retourne une chaîne indiquant la version courante du serveur MySQL. Notez que si votre version se termine par -log, cela signifie que le système d'historique est actif.

mysql> SELECT VERSION();
-> '4.1.2-alpha-log'

Fonctions dans SELECT et WHERE - Fonctions diverses

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

mysql> SELECT FORMAT(12332.123456, 4);
-> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
-> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
-> '12,332'

GET_LOCK(str,timeout) Tente de poser un verrou nommé **str**, avec un délai d'expiration (**timeout**) exprimé en seconde. 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 survenu (comme par exemple un manque de mémoire, ou la mort du thread lui-même, par mysqladmin kill). Un verrou sera levé lorsque vous exécuterez la commande **RELEASE_LOCK()**, **GET_LOCK()** ou si le thread se termine. Cette fonction peut être utilisée pour implémenter des verrous applicatifs ou pour simuler des verrous de lignes. Les requêtes concurrentes des autres clients de même nom seront bloquées ; les clients qui s'entendent sur un nom de verrou peuvent les utiliser pour effectuer des verrouillages coopératifs.

mysql> SELECT GET_LOCK("lock1",10);
-> 1
mysql> SELECT IS_FREE_LOCK("lock2");
-> 1
mysql> SELECT GET_LOCK("lock2",10);
-> 1
mysql> SELECT RELEASE_LOCK("lock2");
-> 1
mysql> SELECT RELEASE_LOCK("lock1");

-> NULL
Notez que le deuxième appel à **RELEASE_LOCK()** retourne **NULL** car le verrou "lock1" a été automatiquement libéré par le deuxième appel à **GET_LOCK()**.

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.

mysql> SELECT INET_ATON("209.207.224.40");
-> 3520061480
Le nombre généré est toujours dans l'ordre des octets réseau ; par exemple, le nombre précédent est calculé comme ceci : **209*256^3 + 207*256^2 + 224*256 +40**. Depuis MySQL 4.1.2, INET_ATON() comprend aussi les IP courtes

mysql> SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1');
-> 2130706433, 2130706433

INET_NTOA(expr) Retourne l'adresse réseau (4 ou 8 octets), de l'expression numérique exp

mysql> SELECT INET_NTOA(3520061480);
-> "209.207.224.40"

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).

IS_USED_LOCK(str) Vérifie si le verrou appelé str est actuellement posé ou pas. Si c'est le cas, la fonction retourne l'identifiant de connexion qui a le verrou. Sinon, elle retourne NULL.

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. Si l'historique principal n'est pas initialisé, retourne NULL. Si l'esclave n'est pas démarré, le maître restera bloqué jusqu'à ce que l'esclave soit démarré et ai atteint la position demandée. Si l'esclave a déjà dépassé cette position, la fonction se termine immédiatement. La valeur retournée est le nombre d'évènements qui a du être traité pour atteindre la position demandée, ou NULL en cas d'erreur. Cette fonction est très utile pour contrôler la synchronisation maître-esclave, mais elle a été initialement écrite pour faciliter les tests de réplications.

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. Le verrou n'existe pas si il n'a pas été obtenu par la fonction **GET_LOCK()** ou si il a déjà été libéré. La commande **DO** est utilisable avec **RELEASE_LOCK()**.

UUID() Retourne un Universal Unique Identifier (UUID) généré grâce à "DCE 1.1 : Remote Procedure Call" (Appendix A) CAE (Common Applications Environment) Specifications, publié par le The Open Group en octobre 1997 (Document numéro C706).

Un **UUID** est conçu comme un numéro qui est globalement unique dans l'espace, et le temps. Deux appels à **UUID()** sont supposés générer deux valeurs différentes, même si ces appels sont faits sur deux ordinateurs séparés, qui ne sont pas connectés ensembles. Un **UUID** est un nombre de 128 bits, représenté par une chaîne de 5 nombres hexadécimaux, au format **aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee** : Les trois premiers nombres sont générés à partir d'un timestamp. Le quatrième nombre préserver l'unicité temporelle si le timestamp perd sa monotonie (par exemple, à cause du changement d'heure d'hiver/été). Le cinquième nombre est un nombre IEEE 802 qui fournit l'unicité. Un nombre aléatoire est utilisé la si ce dernier n'est pas disponible (par exemple, comme l'hôte n'a pas de carte Ethernet, nous ne savons pas comment trouver une adresse matériel sur le système d'exploitation). Dans ce cas, l'unicité spatiale ne peut être garantie. Néanmoins, une collision aura une très faible propriété.

mysql> SELECT UUID();
-> '6ccd780c-baba-1026-9564-0040f4311e29'

Fonctions dans GROUP BY

AVG(expr) Retourne la moyenne de l'expression expr

mysql> SELECT student_name, AVG(test_score)
-> FROM student
-> GROUP BY student_name;

BIT_AND(expr) Retourne la combinaison AND bit à bit de expr. Le calcul est fait en précision de 64 bits (BIGINT).

BIT_OR(expr) Retourne la combinaison OR bit à bit de expr. Le calcul est fait en précision de 64 bits (BIGINT). Cette fonction retourne 0 s'il n'y a pas de ligne à traiter.

BIT_XOR(expr) Retourne la combinaison XOR bit à bit de expr. Le calcul est fait en précision de 64 bits (BIGINT). Cette fonction retourne 0 s'il n'y a pas de ligne à traiter.

COUNT(expr) Retourne le nombre de valeurs non-NULL dans les lignes lues par la commande SELECT

```
mysql> SELECT student.student_name,COUNT(*)
-> FROM student,course
-> WHERE student.student_id=course.student_id
-> GROUP BY student_name;
```

COUNT(*) est un peu différente dans son action, car elle retourne le nombre de lignes, même si elles contiennent NULL. COUNT(*) est optimisée pour retourner très rapidement un résultat si SELECT travaille sur une table, qu'aucune autre colonne n'est lue, et qu'il n'y a pas de clause WHERE. Par exemple :

```
mysql> SELECT COUNT(*) FROM student;
```

Cette optimisation s'applique uniquement pour les tables MyISAM et ISAM, car un compte exact du nombre de lignes est stocké pour ces types de tables, et il peut être lu très rapidement. Pour les moteurs de tables transactionnels, (InnodB, BDB), le stockage de cette valeur est plus problématique, car plusieurs transactions peuvent survenir en même temps, et affecter ce compte.

COUNT(DISTINCT expr,[expr...]) Retourne le nombre de valeurs non-NULL distinctes :

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

Avec MySQL, vous pouvez lire le nombre d'expression distinctes qui ne contiennent pas NULL, en plaçant ici une liste d'expression. Avec SQL-99, vous devriez faire une concaténation de toutes les expressions dans **COUNT(DISTINCT ...)**

GROUP_CONCAT(expr) Syntaxe complète :

GROUP_CONCAT([DISTINCT] expr [,expr ...]

[ORDER BY unsigned_integer | col_name | formula [ASC | DESC] [,col ...]]

[SEPARATOR str_val])

Retourne la chaîne résultant de la concaténation de toutes les valeurs du groupe

```
mysql> SELECT student_name,
-> GROUP_CONCAT(test_score)
-> FROM student
-> GROUP BY student_name;
```

ou :

```
mysql> SELECT student_name,
-> GROUP_CONCAT(DISTINCT test_score
-> ORDER BY test_score DESC SEPARATOR " ")
-> FROM student
-> GROUP BY student_name;
```

Avec MySQL, vous pouvez obtenir la concaténation d'une série d'expressions. Vous pouvez éliminer les doublons en utilisant **DISTINCT**. Si vous voulez trier les valeurs du résultat, il faut utiliser **ORDER BY**. Pour trier en ordre inverse, ajoutez le mot clé **DESC** (descendant) au nom de la colonne que vous triez dans la clause **ORDER BY**. Par défaut, l'ordre est ascendant. Cela peut être spécifié explicitement avec le mot clé **ASC**. **SEPARATOR** est une chaîne qui sera insérée entre chaque valeur du résultat. La valeur par défaut est une virgule ",". vous pouvez supprimer le séparateur en spécifiant la chaîne vide SEPARATOR "".

Vous pouvez donner une taille maximale à la variable **group_concat_max_len** de votre configuration. La syntaxe pour faire cela durant l'exécution est

```
SET [SESSION | GLOBAL] group_concat_max_len = unsigned_integer;
```

Si une taille maximale a été atteinte, le résultat sera tronqué à cette taille maximale. Note : il y a encore de petites limitations pour **GROUP_CONCAT()** lorsqu'il faut utiliser des valeurs **DISTINCT** avec **ORDER BY** et et en utilisant les valeurs **BLOB**.

MIN(expr), MAX(expr) Retourne le minimum ou le maximum de **expr**. MIN() et MAX() peuvent prendre des chaînes comme argument : dans ce cas, elles retournent la valeur minimale ou maximale de la valeur de la chaîne.

mysql> SELECT student_name, MIN(test_score), MAX(test_score)
-> FROM student
-> GROUP BY student_name ;

STD(expr), STDDEV(expr) Retourne la déviation standard de **expr** la racine carrée de la **VARIANCE()**. Ceci est une extension au standard SQL 99. La forme **STDDEV()** de cette fonction est fournie pour assurer la compatibilité Oracle.

SUM(expr) Retourne la somme de **expr**. Notez que si le résultat ne contient pas de ligne, cette fonction retournera NULL.

VARIANCE(expr) Retourne la variance standard de l'expression **expr** (en considérant que les lignes forment une population totale, et non pas un échantillon. Le nombre de ligne est le dénominateur.

Procédures stockées et fonctions

Une procédure stockées est un jeu de commandes SQL qui réside sur le serveur. Une fois qu'elle sont enregistrées, les clients n'ont pas besoin de soumettre chaque commande individuellement, mais peuvent les lancer d'un seul coup.

Les procédures stockées fournissent un gain de performances, car moins d'informations sont échangées entre le serveur et le client. En échange, cela augmente la charge du serveur, car ce dernier doit réaliser plus de travail. Souvent, il y a de nombreux clients, mais peut de serveurs.

- Les procédures stockées requièrent la table **proc** dans la base mysql.
- Le droit de **CREATE ROUTINE** est nécessaire pour créer une procédure stockée.
- Le droit de **ALTER ROUTINE** est nécessaire pour pouvoir modifier ou effacer une procédure stockée. Le droit est fourni automatiquement au créateur d'une routine.
- Le droit de **EXECUTE** est requis pour exécuter une procédure stockée. Cependant, ce droit est fourni automatiquement au créateur d'une routine. De plus, la caractéristique par défaut **SQL SECURITY** est définie (**DEFINER**), ce qui fait que les utilisateurs qui ont accès à une base de données associée à une routine ont le droit d'exécuter la routine

Déclencheurs

Un déclencheur est un objet de base de données nommé, qui est associé à une table et qui s'active lorsqu'un événement particulier survient dans une table. les commandes suivantes configurent une table, ainsi qu'un déclencheur pour les commandes **INSERT** sur cette table. Le déclencheur va effectuer la somme des valeurs insérées dans une des colonnes

mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2)) ;
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
-> FOR EACH ROW SET @sum = @sum + NEW.amount ;

CREATE TRIGGER

CREATE TRIGGER trigger_name trigger_time trigger_event ON tbl_name FOR EACH ROW trigger_stmt

Le déclencheur est associé à la table appelée **tbl_name**. **tbl_name** doit faire référence à une table permanente. Vous ne pouvez pas associer un déclencheur avec une table **TEMPORARY** ou une vue. **trigger_time** est le moment d'action du déclencheur. Il peut être **BEFORE** (avant) ou **AFTER** (après). **trigger_event** indique le type de commande qui active le déclencheur. Il peut valoir **INSERT**, **UPDATE** ou **DELETE**. Il ne peut pas y avoir deux déclencheurs pour une même table avec les mêmes configurations de moment et de commande.

trigger_stmt est la commande à exécuter lorsque le déclencheur s'active. Si vous voulez utiliser plusieurs commandes, utilisez les agrégateurs **BEGIN ... END**. Cela vous permet aussi d'utiliser les mêmes codes que ceux utilisés dans des procédures stockées. Dans la commande d'activation d'un déclencheur, vous pouvez faire référence aux colonnes dans la table associée au déclencheur en utilisant les mots **OLD** et **NEW**. **OLD.col_name** fait référence à une colonne d'une ligne existante avant sa modification ou son effacement. **NEW.col_name** fait référence à une colonne d'une ligne après insertion ou modification.

L'utilisation de **SET NEW.col_name = value** requiert le droit de **UPDATE** sur la colonne. L'utilisation de **SET value = NEW.col_name** requiert le droit de **SELECT** sur la colonne. La commande **CREATE TRIGGER** requiert le droit de **SUPER**.

DROP TRIGGER

DROP TRIGGER tbl_name.trigger_name Supprime un déclencheur. Le nom du déclencheur doit inclure le nom de la table, car chaque déclencheur est associé à une table particulière. La commande **DROP TRIGGER** requiert le droit de **SUPER**.

Vues

ALTER VIEW

ALTER VIEW view_name [(column_list)] AS select_statement Cette commande modifie la définition d'une vue. **select_statement** est le même que pour **CREATE VIEW**

CREATE VIEW CREATE [OR REPLACE] [ALGORITHM = MERGE | TEMPTABLE] VIEW view_name [(column_list)] AS select_statement [WITH [CASCADED | LOCAL] CHECK OPTION]

Cette commande crée une nouvelle vue, ou remplace une vue existante si la clause OR REPLACE est fournie. La clause select_statem

Par exemple, SELECT peut faire référence à une table seule, une jointure ou une UNION. La commande SELECT peut ne pas faire

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
```

Par défaut, la vue est placée dans la base de données par défaut. Pour créer une vue explicitement dans une base de données, spécifiez le nom de la base de données lors de la création : **db_name.view_name**

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

DROP VIEW

DROP VIEW [IF EXISTS] view_name [, view_name] ... [RESTRICT | CASCADE] DROP VIEW supprime une ou plusieurs vues. Vous devez avoir les droits de **DROP** pour chaque vue.

SHOW CREATE VIEW

SHOW CREATE VIEW view_name Cette commande montre la commande **CREATE VIEW** qui créera la vue spécifiée.

INFORMATION_SCHEMA

Les **métadonnées** sont des informations sur les données, telles que le nom des bases de données, des tables, le type de données des colonnes ou les droits d'accès. On appelle aussi ces données le **dictionnaire de données** ou le **catalogue système**.