TOC ***Previous*** ***Next***

MySQL Command Syntax

Structured Query Language

Structured Query Language (SQL) is a non-procedural computer language, *originally developed* in the late 1970s by IBM at its San Jose Research Laboratory.

Let's begin with how to pronounce it. The American National Standards Institute (ANSI) wants it pronounced *ess-kew-ell*. The International Standards Organisation (ISO) takes no position on pronunciation. Many database professionals and most Microsoft SQL Server developers say *see-kwel*. The makers of MySQL prefer *my-ess-kew-ell*. Take your pick.

The 'Q' in SQL stands for 'Query', which’implies retrieving an answer to a question, but in SQL a query may retireve (SELECT), add (INSERT), UPDATE or DELETE data (the elements referenced by the CRUD acronym: create, read, update, delete). What’s more, such data may be user data, or data structure data, or data management source code, or user privileges. SQL is a famously incomplete language, but it's a powerful language for data-driving software.

Traditionally, SQL's incompleteness lies in specifying *what* a DBMS is to do rather than *how* the DBMS is to do it. You cannot use it to produce a complete computer program, only to interface with a database. This you can do in three ways:

- *interactively*: in a standalone application with a MySQL command interface, or
- *statically*: embed fixed SQL statements to execute in other programs (PHP, etc) or
- *dynamically*: you can *PREPARE* SQL statements, and you can use other languages to build runtime SQL statements based on program logic, user choices, business rules, etc., and to send those SQL statements to MySQL.

With many RDBMS products, the line is blurring between specifying what the RDBMS is to do and how it should do it; for example MySQL has syntax for telling the query optimiser how to execute a particular query. MySQL is an open-source product so you can, in theory, rewrite how MySQL does anything. In practice you are not likely to try that on a large scale. But MySQL has had, traditionally, an interface for writing user-defined functions (UDFs), and with 5.1 MySQL introduced an an API for user-coded plugins.

In general a SQL statement …

- begins with a keyword verb (e.g., SELECT),
- must have a reference to the object of the verb (e.g., `*` meaning all columns), and
- usually has modifiers (e.g., FROM `my_table`, WHERE `conditional_expression`) that scope the verb's action. Modifying clauses may be simple keywords (e.g., DISTINCT), or may be built from expressions (e.g., `WHERE myID < 100`).

Table 6-1: SQL Statement Components

Component	*Type*	*Use*	*Examples*
verbs	keywords	action descriptors	SELECT, JOIN, UPDATE, COMMIT, GRANT
object, type names	keywords	general object references	TABLE, VIEW, DOMAIN, INTEGER, VARCHAR
function/variable names	keywords	function and variable references	MAX, AVG, SESSION_USER
conjoiners	keywords	conjoin verbs & object refs	FROM, WHERE, WHEN, WITH
modifiers	keywords	define scope	ANY, TEMPORARY
constant values	keywords	defined constant values	TRUE, FALSE, NULL
identifiers	string literals	names of schemas, databases, tables, views, cursors, routines, columns, authorization IDs, etc.	tableName.columnName, "columnName"
operators	symbolic	relate variables and values	<, <=, =, >, >=, LIKE, *
literal values	literals	data	1006, 'Smith', 2005-5-20

Clauses, expressions and statements are built according to a set of simple syntactic rules from keywords (verbs, nouns, conjunctions), identifiers, symbolic operators, literal values and (except in dynamic SQL) a statement terminator, ';'. Table 6-1 lists the nine kinds of atoms used in SQL to assemble SQL expressions, clauses and statements.

The set of all SQL statements that define schemas and the objects within them, including tables, comprise the SQL *Data Definition Language* (DDL). The set of SQL statements that control users' rights to database objects comprise the *Data Control Language* (DCL). Often DCL is considered part of DDL. SQL statements that store, alter or retrieve table data comprise *Data Manipulation Language* (DML).

SQL also has:

- *connection statements*, which connect to and disconnect from a database
- *session statements*, which define and manage sessions,
- *diagnostic statements*, which elicit information on the database and its operations,
- *transaction statements*, which define units of work and mark rollback points.

This much, most SQL vendors and users can agree on. But no two SQL implementations are identical. Variation is the exceptionless rule. ANSI and ISO have approved SQL as the official relational query language. ANSI has issued *five* SQL standards: SQL86, SQL89, SQL92, SQL99, SQL2003. SQL92 remains a common reference point, with three levels: entry, intermediate, full. SQL99 has no levels: what was *entry-level* in SQL92 became *core* in SQL99. Several commercial vendors implement an SQL variant known as Transact-SQL (T-SQL). And so it goes.

The MySQL variant of SQL

Since version 5.0, MySQL complies with entry-level SQL92, implements much of SQL99, has some features of T-SQL and SQL 2003, and extends ISO SQL in other ways for performance, ease of use and modern postrelational features (see Markus Winand's excellent *review*). 5.0 brought stored routines, updateable Views, Triggers, information_schema and XA transactions, 5.1 partitions. 5.5 SIGNAL and RESIGNAL and LOAD XML, and 5.6 GET DIAGNOSTICS. Version 8.0 added SQL roles, recursive *Common Table Expressions* (CTEs) and windowing functions; 8.0.16 finally added *CHECK CONSTRAINT*; 8.0.22 added *parenthesised query expressions* and a REPLICA alias for the offensive SQL replication keyword SLAVE.

Table 6-2: Some SQL basics still missing from MySQL
Update subqueries
Nested transactions, Queues
Full Outer Join, Assertions

MySQL AB said its eventual aim was complete ISO SQL compatibility, but Oracle has not announced that goal. In any case, with five official definitions, ISO SQL is *a moving target*, fully implemented by *no* vendor![1-4] Is a SQL feature implemented by no vendor actually SQL? Does SQL consist of the five sets of ISO standards, or the union of all commands implemented by all SQL vendors? Is a feature implemented by many vendors "SQL" before it appears in a subsequent version of the standard? Impossible questions, all. Yet we need a usage that makes sense. Our take is that a feature is SQL if it is in one of the five published standards, *or* commonly implemented by vendors. Do the items in Table 6-2 matter to you? Only you can decide.

Notable MySQL variations from SQL92

Transactions: To enable transactions on a table, create it with a transactional *storage engine*. The MYISAM engine is transactionless. Since version 5.5 the default engine is the mainly ACID-compliant INNODB engine. Since 8.0, MySQL systems tables use this too.

Foreign Keys: MySQL accepts FOREIGN KEY syntax, but implements it only if the table uses a transaction engine, e.g., INNODB; if the table uses a transactionless engine like MYISAM, foreign key declarations are *ignored*.

CREATE | DROP VIEW*:* MySQL Views support FROM clause subqueries since version 5.7. They still do not optimise well.

SELECT INTO TABLE: MySQL supports not SELECT ... INTO TABLE ... but the equivalent INSERT ... SELECT ... .

Definition of a user: On this, MySQL goes its own way, defining a user as a unique *authID* formed as *user@host*, where *user* is the user name, and *host* is the network address from which the user may connect. Since 8.0 (and MariaDB 10.0.5), *user* may also name a *role*, which is thus a named privilege set; users may be assigned roles and vice versa, a huge convenience in privilege management.

Other deviations of note from SQL92 and SQL99. MySQL provides the functionality of DECLARE LOCAL TEMPORARY TABLE via CREATE TEMPORARY TABLE and CREATE TABLE ... ENGINE=HEAP. Other DDL elements awaiting implementation are: schemas within databases, CREATE/DROP DOMAIN, CREATE/DROP CHARACTER SET, CREATE/DROP COLLATION, CREATE/DROP TRANSLATION., INSTEAD OF TRIGGER.

MySQL identifier names

Rules for building MySQL identifier names are simple:

- In the first character position, MySQL accepts an alphanumeric, '_' or '$', but some RDBMSs forbid digits here, so for portability do not use them.
- The first character of the name of a database, table, column or index may be followed by any character allowed in a directory name to a maximum length of 64, or 255 for aliases. For portability, a safe maximum is 30.
- Identifiers can be qualified (`tblname.colname`), and quoted by backticks (`` `tblname` ``) or by double quotes if *`sql_mode`* includes `ansi_quotes`.
- Unquoted identifier names cannot be case-insensitive-identical with any keyword.

MySQL allows reserved words (so labelled in 8.0 in `information_schema.keywords`) as identifiers if they are quoted or backticked. Don't! They complicate writing SQL commands and compromise portability. Likewise for the underscore. For links to pre-8.0 reserved word changes see our *incompatibilities page*.

MySQL comments

MySQL accepts three comment styles:

- `#` marks everything to the right of it as a comment;
- so does `--` ; the dashes must be followed by a space;
- `/*...*/` marks off an in-line or multi-line comment, but if it contains a semi-colon or unmatched quote, MySQL will fail to detect the end of the comment.

Nested comments are not supported. Other SQL engines will ignore a `/*...*/` comment beginning with `'!'`, but MySQL will execute it if no version string follows the ‘!’, or if the version string is not later than the running MySQL version, e.g., CREATE /*!32302 TEMPORARY */ TABLE... executes in MySQL if the server is 3.23.02 or later.

MySQL Operators

MySQL has four kinds of operators: logical, arithmetic, bit, comparison.

Logical operators: There are five (Table 6-3). Before 8.0.17, || is accepted as a synonym for OR,

&& for AND, ! for NOT. When || means OR, the ANSI SQL expression `"join" || "this"` returns zero! Concatenate strings with *CONCAT()*. Operations on NULLs follow the rules of *three-valued logic* (Table 6-4): NULL is *never* equal to anything including itself, so both (NULL=NULL) and (0 || NULL) return NULL.

Arithmetic operators: MySQL has 7 (Table 6-5). There is no exponential operator; use *POWER*, *SQRT*, *LOG10*, *LOG* or *EXP*.

Bit operators (Table 6-6) use 64-bit BIGINT numbers.

Comparison operators (Table 6-7) apply to numbers, strings and dates, returning 1 (TRUE), 0 (FALSE), or NULL. Data conversions are automatic as context requires. Rules:

1. Except with <=>, if any argument is NULL, the result is NULL.

2. Two strings are compared as strings; two integers as integers.

3. MySQL treats hex values as binary strings except in numeric comparisons. Beware that INNODB ignored trailing whitespace in BINARY- VARBINARY comparisons until 5.0.18; since, not.

4. Before 5.0.42 and 5.1.18, DATE-DATETIME comparisons ignored time. Since then, MySQL coerces the TIME portion of DATE to `00:00: 00`; CAST(`datevalue` AS DATE) emulates the earlier behaviour.

5. If one operand is TIMESTAMP or DATETIME and the other constant, the constant converts to timestamp before comparison, so in `d>020930`, `020930` becomes a timestamp.

6. Otherwise operands are compared as floats, so `SELECT 7>'6x'` returns TRUE.

Table 6-3: Logical operators in MySQL

Syntax	*Meaning*
x OR y, x}}y*	1 if either x or y is non-zero
x XOR y	1 if odd no. of operands non-zero
x AND y, x&&y*	1 if x and y are non-zero
NOT x, !x*	1 if x is zero
x IS y	1 if x is y
* *Deprecated 8.0.17*	

Table 6-4: Three-valued logic

	OR			*AND*			*IS*		
	true	*false*	*null*	*true*	*false*	*null*	*true*	*false*	*null*
true	true	true	true	true	false	null	true	false	false
false	true	false	null	false	false	false	false	true	false

Table 6-5: Arithmetic operators in MySQL

Operator	*Syntax*	*Meaning*
+	x+y	addition
-	-x	negative value
-	x-y	subtraction
*	x*y	multiplication
/	x/y	division
DIV	x DIV y	integer division
%, MOD	x%y	modulo, same as MOD(x,y)

Table 6-6: Bit operators in MySQL

Syntax	*Meaning*	*Example*
x | y	bitwise OR	29 | 15 = 31
x & y	bitwise AND	29 & 15 = 13
x ^ y	bitwise XOR	29 ^ 15 = 18
x<<y	shift x left y bits	1<<2=4
x>>y	shift x right y bits	4>>2=1

Table 6-7: Comparison operators in MySQL

Operator	*Syntax*	*Meaning*
=	x=y	true if x equals y
<>, !=	x<>y, x!=y	true if x and y not equal
<	x<y	true if x less than y
<=	x<=y	true if x less than or equal to y
>	x>y	true if x greater than y

Operator	Syntax	Meaning
>=	x>=y	true if x greater than or equal to y
<=>	x<=>y	true if x and y are equal, even if both are NULL (but 5<=>NULL is false, 5=NULL and 5<>NULL are NULL)
[NOT] IN(...)	x [NOT] IN (y1,y2,... | subquery)	true if x (not) in list or subquery result
= ANY | SOME	= ANY | SOME(subquery)	true if any row satisfies subquery
<> ANY | SOME	<> ANY | SOME(subquery)	true if some row does not satisfy subquery
= ALL	= ALL(subquery)	true if every row satisfies subquery
<> ALL	<> ALL(subquery)	same as NOT IN(subquery)
[NOT] BETWEEN ... AND	x [NOT] BETWEEN y1 AND y2	true if x (not) between y1 and y2
x [NOT] LIKE y [ESCAPE 'esc_char']	x [NOT] LIKE y [ESCAPE c]	true if x does [not] match pattern y; if given, use escape_char bounded by single quotes in place of '\'
[NOT] REXEXP | RLIKE	x [NOT] REGEXP | RLIKE y	true if x does (not) match y as extended reg. expression
SOUNDS LIKE	x SOUNDS LIKE y	true if SOUNDEX(x) = SOUNDEX(y)
IS [NOT] NULL	x IS [NOT] NULL	true if x is [not] NULL
BINARY	BINARY x <op> y	Treat x case-sensitively in string comparison <op>

Operator precedence

Operators have a definite precedence hierarchy, as shown from high to low in Table 6-8. Note two variations: the precedence of || rises to just below that of the unary operators if the `sql_mode` setting includes `pipes_as_concat`, and NOT precedence rises to the level of ! when `sql_mode` includes `high_not_precedence`.

To override the precedence hierarchy in complex expressions when you are uncertain about operator binding, or to build in insurance against unwitting precedence errors, use parentheses freely.

Table 6-8: Operator precedence	
1	BINARY, COLLATE
2	!
3	- (unary minus), ~ (unary bit inversion)
4	^
5	*, /, DIV, %, MOD
6	-, +
7	<<, >>
8	&
9	|
10	=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
11	BETWEEN, CASE, WHEN, THEN, ELSE
12	NOT
13	&&, AND
14	||, OR, XOR
15	:=

LIKE, RLIKE | REGEXP

LIKE compares strings case-insensitively unless an argument is a BLOB or BINARY is applied; `'%'` represents any string, `'_'` any character (Table 6-9). To specify an escape character other than `'\'`, use `LIKE … ESCAPE 'char'`. Since 8.0.4 MySQL uses International Unicode components so it is now multibyte-safe and Unicode-compliant, see REGEXP string functions in *Chapter 8*.

Table 6-9: SELECT ... LIKE ... arguments	
LIKE argument	*Retrieves*
'p%'	Values beginning with 'p'
'%t'	values ending with 't'
'%a%'	values containing 'a'
'___'	values containing exactly 3 characters

RLIKE | REGEXP extends LIKE for limited regular expression matching (Table 6-10). Like LIKE, it returns 1 if the pattern on the right matches that on the left, 0 if not:

```
SELECT 'MySQL' LIKE 'M%';   -- returns 1
SELECT 'MySQL' RLIKE '^M';  -- returns 1
```

A `word` is a sequence of `alnum` or `'_'` characters. Apostrophes end words.

Table 6-10: RLIKE | REGEXP Patterns

REGEXP argument	*Matches ...*	*Example of a match*
`.`	any single character	`'abc' RLIKE '.'`
`^`	beginning of string	`'abc' RLIKE '^a'`
`$`	end of string	`'abc' RLIKE 'c$'`
`*`	zero or more occurrences of preceding token	`'abc' RLIKE 'd*'`
`+`	one or more occurrences of preceding token	`'abc' RLIKE 'c+'`
`?`	zero or one occurrences of preceding token	`'abc' RLIKE 'd?'`
`x|y`	token x or token y	`'abc' RLIKE 'c|x'`
`[…]`	any member of the character class	`'abc' RLIKE '[cyz]'`
`[x-y]`	class of characters from x through y	`'abc' RLIKE '[c-g]'`
`[A-Z]`	class of upper case alphabetic characters	`'ABC' RLIKE '[A-Z]'`
`[a-z]`	class of lower case alphabetic characters	`'bcd' RLIKE '[a-z]'`
`[0-9]`	class of digits	`'123' RLIKE ['0-9]'`
`[A-Za-z]`	class of all alphabetic characters	`'abc' RLIKE '[A-Za-z]'`
`[^…]`	class of characters which do *not* occur	`'abc' RLIKE '[^def]'`
`{n}`	at least n instances of preceding token	`'abbc' RLIKE 'b{2}'`
`{m,n}`	m through n instances of preceding token	`'abbbc' RLIKE 'b(1,3}'`
`[.char.]`	character: literal, or as named in *regexp/cname.h* *	`'ab~' RLIKE '[[.tilde.]]'`
`[:class:]`	named character class **	`'ab12' RLIKE '[[:alnum:]]'`
`[=equiv=]`	characters of the same collation value	`'abc' RLIKE '[[=c=]]'`
`[[:<:]]`	beginning of a word	`'a bit' REGEXP '[[:<:]]bit]]'`
`[[:>:]]`	end of a word	`'bit 8' REGEXP 'bit[[:>:]]'`

* NUL,SOH,STX,ETX,EOT,ENQ,ACK,BEL,alert,BS,backspace,HT,tab,LF,newline,VT,vertical-tab,FF,form-feed,CR,carriage-return, SO,SI,DLE,DC1,DC2,DC3,DC4,NAK,SYN,ETB,CAN,EM,SUB,ESC,IS4,FS,IS3,GS,IS2,RS,IS1,US,space,exclamation-mark, quotation-mark,number-sign,dollar-sign,percent-sign,ampersand,apostrophe,left-parenthesis,right-parenthesis,asterisk,plus-sign, comma,hyphen,hyphen-minus,period,full-stop,slash,solidus,zero…nine,colon,semicolon,less-than-sign,equals-sign, greater-than-sign,question-mark,commercial-at,left-square-bracket,backslash,reverse-solidus,right-square-bracket,circumflex, circumflex-accent,underscore,low-line,grave-accent,left-brace,left-curly-bracket,vertical-line,right-brace,right-curly-bracket,tilde,DEL

***alnum*: alphanumerics; *alpha*: alphabetics; *blank*: whitespace; *cntr*l: control; *digit*: digits; *graph*: graphic; *lowe*r: lowercase alphabetics; *print*: graphic or space; *punc*t: punctuation; *spac*e: ' ', tab, cr, or lf; *upper*: uppercase alphabetics; *xdigit*: hexadecimal.

The row constructor

Whether you consider the row constructor an operator or function is a matter of taste:

[ROW](expr1,expr2[,…,[exprN]])

It constructs a logical row from a comma-separated list of scalar expressions, and can be used in comparisons within JOIN and WHERE clauses, for example

```
SELECT … WHERE ROW(col1,col2,col3)=(11,20,25);
```

Connections and Sessions

A session begins with connection, ends with disconnection. Connect to MySQL server

- from an operating system (OS) prompt using the *mysql* client or Workbench, or
- from a custom MySQL front-end application, or
- from another program via its API (e.g., *C/C++*, *Perl*, *PHP*, *Java*, *Visual Studio*)

Connect, disconnect from the OS shell

MySQL has always provided an interactive commandline utility called *mysql*. Version 5.5 introduced the *mysql shell* (*mysqlsh*) enhanced commandline utility (*Chapter 3*).

Some MySQL installations allow users to connect anonymously, so you can connect using the MySQL client program by typing, in a directory that can see the installation *bin* folder …:

```
shell> mysql
```

which loads that program. More usually, you need to provide a host name, a user name, and if the user.password column value for your user name is non-empty, a password. If the MySQL installation you are connecting to is not your own, you may have to ask your administrator for the required connection parameters. Once you know your host name, user name and password, you can issue this command:

```
mysql -hhost -uuser -pPASSWORD
```

with no space between the '`p`' and your password, or one of these:

```
mysql --host=hostname --user=username --password=PASSWORD
mysql -hhostname -uusername -p
```

In the latter case the mysql program will respond with:

```
Enter password:
```

In Linux the client program *mysql* defaults missing connection parameters, hostname to `localhost`, user name to your Linux login. Without `-p` there is no password prompt.

There are two other ways to specify command line connection parameters save them in environment variables `MYSQL_HOST, USER` (Windows) and `MYSQL_PWD`, or save them to the `[client]` section of the *my.cnf* configuration file in your home directory:

```
[client]
host=your_host_name
user=your_user_name
password=your_password
```

Under *Nix you can make this a private file with

```
chmod 800 my.cnf
```

but obviously keeping an unencrypted password in a configuration file, or in a text file which sets environment variables, is not secure.

Once a connection succeeds, you get the `mysql>` prompt:

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 459 to server version: 5.0.9-beta
Type 'help' for help.
mysql>
```

Disconnect by typing \q, `QUIT`, `EXIT` or Ctrl+D at the `mysql>` prompt.

Connecting from other software

From early on, MySQL has provided APIs for Perl, PHP, Java, C and C++ . Later came APIs for Microsoft Access, ODBC, dotNet, then for Python, node.js, Arduino. Developers can provide user-friendly MySQL access via any of these APIs in custom applications, or in generally available database management apps or webapps (*phpMyAdmin*, *HeidiSQL*, *Toad*, *TheUsual* &c). And of course, since version 5.0 MySQL provides the GUI MySQL management tool *MySQL Workbench*.

TOC ***Previous*** ***Next***