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.

Although the 'Q' in SQL stands for 'Query', SQL is a language not only for querying data, but for creating and modifying database structures and their contents, for inserting, updating and deleting database data, for managing database sessions and connections, and for granting and revoking users' rights to all this.

Traditionally, SQL statements specify what a DBMS is to do, not *how* the DBMS is to do it. So SQL is a partial computer language: 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*: you can embed fixed SQL statements for execution within programs written in other languages (Perl, PHP, Java, 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 between specifying what the RDBMS is to do, and how it should do it, is blurring; for example MySQL has syntax for telling the query optimiser how to execute a particular query. Since MySQL is an open-source product, 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 API for user 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 references	FROM, WHERE, WHEN
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, procedures, 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 implementation of SQL is identical to any other. 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 and SQL2003. SQL92 remains a common reference point, with three levels: entry level, intermediate and full. SQL99, in contrast, has no levels but rather core and other features: what was *entry-level* in SQL92 becomes *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

MySQL 5 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 support for modern 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 adds SQL roles, recursive Common Table Expressions (CTEs) and windowing functions.

Table 6-2: Some SQL basics still missing from MySQL

Check Constraint*
Update subqueries
Nested transactions, Queues
Full Outer Join, Assertions
* supported by MariaDB 10.2.1 and later

MySQL AB said its eventual aim was complete ISO SQL compatibility. Oracle has not publicly adopted 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. Here is our rough take. A feature is SQL if it is in one of the five published standards, *or* if it is commonly implemented by vendors. Whether missing SQL functionalities like those listed in Table 6-2 matter to you depends on your database requirement.

Notable MySQL variations from SQL92

Transactions: To enable transactions on a table, create it with a transactional *storage engine*. The MySQL MYISAM database engine is transactionless, but 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, the foreign declaration is *ignored*.

CREATE | DROP VIEW: MySQL Views did not support FROM clause subqueries until version 5.7, a serious limitation. MySQL Views still do not optimise well.

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

Definition of a user: Here MySQL goes its own way. A MySQL user is defined by an authentication ID or *authID* in the form *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.

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

The 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 permits use of reserved words as identifiers if they are quoted. Don't do it! It complicates writing SQL commands, and it compromises portability. Likewise for the underscore. MySQL now has a *page* pointing to version-specific lists of reserved words.

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.

A /*...*/ comment beginning with '!' followed by a version string tells MySQL to execute the string following the version string if the current server is that version or later. Thus CREATE /*!32302 TEMPORARY */ TABLE ... creates a temporary table in MySQL if the server is 3.23.02 or later; the commented text is ignored outside MySQL.

MySQL Operators

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

Logical operators: There are five (Table 6-3). MySQL uses || as a synonym for OR, so under MySQL the ANSI SQL expression "join" || "this" returns zero! To concatenate strings, use *CONCAT()*.

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

Remember that operations on NULLs follow the rules of *three-valued logic* (Table 6-4): NULL is *never* equal to NULL, a known value OR NULL = the value, and a known value AND NULL is NULL.

Arithmetic operators: MySQL has 7 (Table 6-5). There is no exponential operator; to raise a number to a power 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 the context requires, on these rules:

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

2. If both arguments are strings, they are compared as strings; if both are integers, they are compared 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. Now white-space is not ignored.

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

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
null	true	null	null	null	false	null	false	false	true

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

DATE) emulates the earlier behaviour.

5. If one operand is TIMESTAMP or DATETIME and the other is a constant, the constant is converted to a timestamp before the comparison, so in `thisdate > 020930`, `020930` will be converted to a timestamp.

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

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
>=	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>

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