# MySQL语言结构

本章讨论了在使用MySQL中编写[SQL](https://dev.mysql.com/doc/refman/5.7/en/glossary.html" \l "glos_sql" \o "SQL)语句时要用到的以下元素的规则：

* 字面值，如字符串和数字
* 标识符，如数据库，表和列名等
* 关键字和保留字
* 用户定义变量和系统变量
* 注释

## 1、字面值

本节描述了如何在MySQL中写字面值。包括字符串、数值、十六进制值、位值、布尔值和NULL。本节还包含MySQL在处理这些基本类型时的各种细微差别。

（1）字符串字面值

字符串是用单引号或者双引号引起来的字符序列，彼此相邻的字符串链接成单个字符串，例如下列是等价的。

'a string'和'a' ' ' 'string'是等价的

如果启用了SQL模式ANSI\_QUOTES，则只能在单引号内引用字符串，因为双引号内引用的字符串将被解释为标识符。

字符串分为二进制字符串和非二进制字符串，二进制字符串的单元为字节，非二进制字符串的单元为字符（有些编码，一个字符用多个字节表示，因此不一定为一个字节），二进制字符串排序时比较的为字符串中各个字节的数字值，非二进制字符串比较的为各个字符对应的数值码值（可见，对于非二进制字符串，不同字符集下，同一个字符串底层存储可以不同，同一个码值也可以不同，因此排序大小也就不同了）。二进制字符串可以理解为字节流。字符串文字可以包含一个可选的字符集引导器和COLLATE子句，以将其指定为特定的字符集和排序规则

[\_*charset\_name*]'*string*' [COLLATE *collation\_name*]

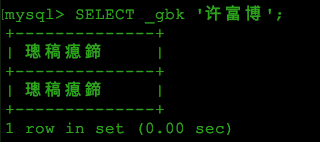
例如：

SELECT \_latin1'*string*';

SELECT \_binary'*string*';

SELECT \_utf8'*string*' COLLATE utf8\_danish\_ci;

这时候MySQL就将字符串按指定的字符集编码，例子如下：

为什么作图第一个为乱码呢？因为MySQL将字符按照latin1编码，而我的屏幕是UTF8编码，这样，第一个就是乱码了。再比如，你用gbk，依然是乱码，如右图所示。

你也可以使用N'literal'(or n'literal') 创建国家字符集的字符串。例如，如果国家字符集为utf8，则以下等价：

SELECT N'some text';

SELECT n'some text';

SELECT \_utf8'some text';

有关这些字符串语法形式的信息，请参见 [第10.1.3.7节“国家字符集”](https://dev.mysql.com/doc/refman/5.7/en/charset-national.html" \o "10.1.3.7国家字符集)和 [第10.1.3.8节“字符集引用”](https://dev.mysql.com/doc/refman/5.7/en/charset-introducer.html" \o "10.1.3.8字符集引导器)。

在字符串中，某些序列具有特殊的含义，除非启用了SQL模式NO\_BACKSLASH\_ESCAPES。这些序列都以反斜杠（\）开头，称为转义字符。MySQL识别下边的转义字符。对不在表中的其他转义序列，会忽略反斜线，像没有被转义一样。例如：\x表示x。转义序列区分大小写。例如， \b被解释为退格，但 \B被解释为B。MySQL依据系统变量 [character\_set\_connection](https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html" \l "sysvar_character_set_connection)指定的字符集对转义进行处理。即使在前面指定了不同字符集导引器的字符串也是如此。

| 转义序列 | 含义 |
| --- | --- |
| \0 | ASCII 0（NUL）字符 |
| \' | 单引号（'） |
| \" | 双引号（"） |
| \b | 退格 |
| \n | 换行符（换行） |
| \r | 回车符 |
| \t | 制表符（tab） |
| \Z | ASCII 26（ctrl+ Z）; |
| \\ | 反斜杠（\） |
| \% | %字符 |
| \\_ | \_字符 |

ASCII 26（Ctrl-Z）。该字符用‘\Z’表示，以允许你解决在Windows中ASCII 26代表文件结尾这一问题。如果你试图使用mysql db\_name < file\_name，ASCII 26会带来问题。

在模式匹配的上下文中，例如：like，\%和\\_代表%和\_，在非模式匹配上下文中，\%和\\_就代表\%和\\_，而非%和\_。

有如下几种方式在字符串中包含引号：

* 用’引用的字符串中，’可以表示为’’。
* 用"引用的字符串中， "可以写成 ""。
* 引号前加转义字符\。
* 单引号引用的字符串内，双引号无需特殊处理，同样，双引号引用的字符串内，单引号无需特殊处理。

例如：

mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello'; +-------+---------+-----------+--------+--------+

| hello | "hello" | ""hello"" | hel'lo | 'hello | +-------+---------+-----------+--------+--------+

mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello"; +-------+---------+-----------+--------+--------+

| hello | 'hello' | ''hello'' | hel"lo | "hello | +-------+---------+-----------+--------+--------+

mysql> SELECT 'This\nIs\nFour\nLines';

+--------------------+

| This Is Four Lines |

+--------------------+

mysql> SELECT 'disappearing\ backslash';

+------------------------+

| disappearing backslash |

+------------------------+

如果你想要在字符串列内插入二进制数据（例如BLOB），必须通过转义序列表示这些字符,反斜线（"\"）和引号必须被转义.在一些客户端环境中，NUL和Control+Z也需要去转义.mysql客户端会截取被引号包含的字符串中的NUL字符（如果NUL字符没被转义的话）.在windows下如果Control+Z没被转义的话,会当成END-OF-FILE。

当编写应用程序时，在将包含这些特殊字符的字符串发送到MySQL服务器之前，必须对它们进行正确的转义。可以用两种方法来完成：

* 用转义特殊字符的函数处理字符串。例如，在C程序中，可以使用mysql\_real\_escape\_string() C API函数来转义字符。参见 [第27.8.7.56节“mysql\_real\_escape\_string\_quote（）”](https://dev.mysql.com/doc/refman/5.7/en/mysql-real-escape-string-quote.html" \o "27.8.7.56 mysql_real_escape_string_quote（）)。Perl DBI接口提供一个quote方法来将特殊字符转换为正确的转义序列。其他语言也有提供相应的接口API。
* 显式转义特殊字符，许多MySQL API提供了占位符功能，允许你在查询字符串中插入特殊标记，然后当你发出查询时将数据值同它们绑定起来。在这种情况下，API关注转义值中的特殊字符.

（2）数值字面值

数字字面值包括精确值（整数和 DECIMAL）和近似值（浮点数）。这个不做详细解释，因为很简单，需要说明的是：

精确值具有整数部分或小数部分，或两者兼有。例如： 1，.2， 3.4，-5， -6.78，+9.10。

近似值数字字面值用科学符号表示，带有尾数和指数。示例：1.2E3,1.2E-3，-1.2E3，-1.2E-3。

看起来相似的两个数字可能会被MySQL以不同的方式处理。例如，2.34是精确值（定点数），而2.34E0是近似值（浮点数）。

（3）日期和时间

日期和时间值可以以多种格式表示，例如引用的字符串或数字，具体取决于值的确切类型和其他因素。例如，在MySQL需要日期的上下文中，，它会将 '2015-07-21'，'20150721'以及20150721的任何一种作为一个日期来解释。

本节介绍日期和时间的可接受格式。有关时间数据类型（例如允许的值的范围）的更多信息，请参阅以下部分：

* [第11.1.2节“日期和时间类型概述”](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-type-overview.html" \o "11.1.2日期和时间类型概述)
* [第11.3节“日期和时间类型”](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-types.html" \o "11.3日期和时间类型)

标准的SQL和ODBC的日期和时间字面值：

标准SQL使用type关键字和字符串指定时间文字。关键字和字符串之间的空格是可选的。

DATE '*str*'

TIME '*str*'

TIMESTAMP '*str*'

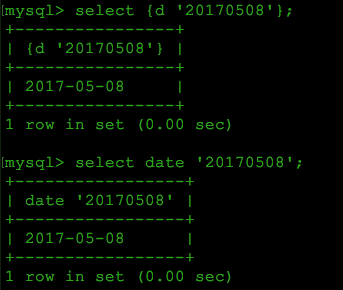
MySQL识别如上结构以及相应的ODBC语法：

{ d '*str*' }

{ t '*str*' }

{ ts '*str*' }

例如：



MySQL会用如上结构的值生成DATA、TIME和DATETIME 类型的值，如果指定的话，它还包括秒的小数部分。在MySQL 中TIMESTAMP语法会产生一个DATETIME类型的值，因为MySQL中DATETIME 范围更贴近标准的SQL 的TIMESTAMP类型，TIMESTAMP的年的范围是0001 到 9999（MySQL 的TIMESTAMP 年的范围是1970 到2038）。

**日期和时间上下文中MySQL如下处理字符串和数值。**

**MySQL可以识别下面三种格式的DATE值:**

* 对于标准的'YYYY-MM-DD'或者'YY-MM-DD'格式的字符串，宽松的语法是允许的: 任何符号都可以放在日期间作为分隔符，如: '2012-12-31'、 '2012/12/31'、'2012^12^31'、'2012@12@31' 是等价的.
* 对于没有分隔符的字符串，如'YYYYMMDD'和 'YYMMDD'，如果日期值是合理的， 如'20070523'、 '070523'会被解释成'2007-05-23'， 但'071332'的月份和日期值就是不合法的，会被取成'0000-00-00'。
* 对于数字YYYYMMDD和 YYMMDD如果日期值是合理的，也会被解释成正确的日期格式，如19830905 和 830905被解释成 '1983-09-05'。

**MySQL可以识别下面的三种格式的DATETIME和TIMESTAMP 值：**

* 对于标准的'YYYY-MM-DD HH:MM:SS'和'YY-MM-DD HH:MM:SS' 格式的字符串。宽松的语法是允许的，可以用任何符号作为日期和时间中的分隔符，如：'2012-12-31 11:30:45'、'2012^12^31 11+30+45'、'2012/12/31 11\*30\*45'和 '2012@12@31 11^30^45'是等价的。

不仅仅空格可以作为日期和时间部分之间的分隔符，其他字符也可以，如T，'2012-12-31 11:30:45' 和'2012-12-31T11:30:45'是等价的

* 对于没有分隔符的字符串'YYYYMMDDHHMMSS'or 'YYMMDDHHMMSS'，如果日期和时间值是合理，如'20070523091528'和 '070523091528'会被解释成 '2007-05-23 09:15:28'，但是'071122129015'是不合法的 （它的分钟不分不合法），会被取成 '0000-00-00 00:00:00'。
* 对于数字格式的YYYYMMDDHHMMSS和 YYMMDDHHMMSS，如日期时间值合理的话，如19830905132800和 830905132800会被解释成'1983-09-05 13:28:00'。

DATETIME和TIMESTAMP 可以包括达到微妙的6位数小数，秒的小数部分与秒之间使用“.”来分割，不可以使用别的符号分割秒和秒的小数部分。更多关于MySQL微妙的信息，参见[第11.3.6节“时间分数秒”](https://dev.mysql.com/doc/refman/5.7/en/fractional-seconds.html" \o "11.3.6时间分数秒)。

仅用两位数表示年的日期类型是不确定的，因为世纪的值不知道。MySQL 用下面的规则来解释两位数表示的年：

* 年的范围在70-99 转换成 1970-1999
* 年的范围在00-69 转换成 2000-2069

对于拥有日期部分分隔符的字符串的值，对小于10的月和天的部分不需要写成两位数。'2015-6-9'与 '2015-06-09'是等价的，对于时间部分也是这样，'2015-10-30 1:2:3'与'2015-10-30 01:02:03'等价。

指定为数字的值应为6,8,12或14位数字。如果是8或14位的数字，被认为是YYYYMMDD或 YYYYMMDDHHMMSS格式，前四个数字表示年。如果是6或12位的数字长度，则认为是YYMMDD或YYMMDDHHMMSS格式，前两位数字表示年，不是这些长度之一的数字被解释为用前导零填充到最接近的长度。

对于没有分隔符的字符串依据他们的长度来处理。对于8或14个字符的字符串，前四位表示年，其他长度的前两位表示年。字符串从左到右被解释为年月日时分秒，尽可能的找到年月日时分秒这六个部分。这意味着表示日期的字符串不能少于六位。如果你指定'9903'想表示1999年03月，然而MySQL会把她转换成00-00-00，这是因为你指定的字符串9903中，99被解释成1999年，03被解释成3月，但是缺失天的部分。不过你可以通过显示指定0值来表示缺少的天数部分。如：990300，会被解释成1999-03-00来处理只需要精确到月份的需求。

**MySQL可以识别下面格式的TIME值:**

* 对于标准的'D HH:MM:SS'格式的字符串，也可以用宽松的语法格式：'HH:MM:SS'、'HH:MM'、'D HH:MM'、'D HH'或者'SS'。这里 D表示天数，可以有0到34的值。
* 对于没有分隔符的'HHMMSS'格式的字符串，只要提供合理的时间值，如'101112'表示10:11:12，但是'109712'是不合法的，它有不合理的分钟值，而被MySQL取成00:00:00。
* 对于HHMMSS格式的数字，只要提供合理的时间值，如101112表示'10:11:12'。以下备选格式也可以被识别：SS， MMSS，或HHMMSS。

对于具有秒的小数部分的格式中，这些格式可以被识别：'D HH:MM:SS.fraction'、'HH:MM:SS.fraction'、'HHMMSS.fraction', 和 HHMMSS.fraction，小数部分可以达到6位数字精度的微秒值。更多MySQL微秒的信息，可参阅[第11.3.6节“时间分数秒”](https://dev.mysql.com/doc/refman/5.7/en/fractional-seconds.html" \o "11.3.6时间分数秒)。

同上边所说，'8:3:2'与'08:03:02'等价。

（4）十六进制字面值

MySQL中十六进制格式为：

（5）bit-value字面值

（6）布尔字面值

（7）NULL值

NULL值意味着没有数据，在SQL语句中用NULL（不区分大小写）或者\N（区分大小写）表示，在SQL语句中\N在MySQL 5.7.18中被弃用在MySQL 8.0中移除。NULL与数值中的0和空字符串不同，不要混淆，以下例子来自MySQL 5.7手册， [Section B.5.4.3, “Problems with NULL Values”](https://dev.mysql.com/doc/refman/5.7/en/problems-with-null.html" \o "B.5.4.3 Problems with NULL Values).

NULL值的概念对于SQL新手来说是一个常见的混淆来源，他们通常认为NULL与空字符串“相同”。实际不是这样。例如，以下语句完全不同：

mysql> INSERT INTO my\_table (phone) VALUES (NULL);

mysql> INSERT INTO my\_table (phone) VALUES ('');

两个语句都将一个值插入到phone列中，但第一个插入一个NULL值，第二个插入一个空字符串。第一个的意思可以认为是“电话号码不知道”（**无法确定**这个人有没有电话以及号码），第二个的含义可以被认为是“确定该人没有电话，因此没有电话号码”。

为了处理NULL，在SQL语句中可以使用IS NULL和IS NOT NULL运算符和IFNULL（）函数。在SQL中，NULL与任何其他值相比，结果值绝对不为true，甚至为NULL（可以理解为与一个不确定的值做比较或者计算，结果当然不确定了）。在表达式中包含NULL值，那么表达式的结果往往为NULL值。除非在表达式中使用的运算符和函数有特殊说明，例如：以下示例结果均为NULL：

mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);

要查询某列为NULL的行，不要使用=，因为刚说过，与不知道的东西对比没有意义，结果还是不知道。可以理解为不知道的东西无法比较，只能说某个值是不是不确定。因此，要使用IS NULL来测定。即：

不要使用：

mysql> SELECT \* FROM my\_table WHERE phone = NULL;

要使用：

mysql> SELECT \* FROM my\_table WHERE phone IS NULL;

查询空电话号码的语句如下：

mysql> SELECT \* FROM my\_table WHERE phone = '';

更多示例：

mysql> SELECT 1 IS NULL, 1 IS NOT NULL;

+-----------+---------------+

| 1 IS NULL | 1 IS NOT NULL |

+-----------+---------------+

| 0 | 1 |

+-----------+---------------+

mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;

+----------+-----------+----------+----------+

| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |

+----------+-----------+----------+----------+

| NULL | NULL | NULL | NULL |

+----------+-----------+----------+----------+

mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL; +-----------+---------------+------------+----------------+

| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |

+-----------+---------------+------------+----------------+

| 0 | 1 | 0 | 1 |

+-----------+---------------+------------+----------------+

更多的信息，请参见[第3.3.4.6节“使用NULL值”](https://dev.mysql.com/doc/refman/5.7/en/working-with-null.html" \o "3.3.4.6使用NULL值)

如果使用 MyISAM，InnoDB或 MEMORY存储引擎，你可以为NULL列建立索引，否则，必须将列限制为NOT NULL。

当使用LOAD DATA INFILE读取数据时，使用’’更新空的或缺少的列。要将NULL值加载到列中，请在数据文件中使用\ N。在某些情况下也可以使用字面值NULL。参见[13.2.6, “LOAD DATA INFILE Syntax”](https://dev.mysql.com/doc/refman/5.7/en/load-data.html" \o "13.2.6 LOAD DATA INFILE Syntax).

当使用DISTINCT，GROUP BY或，或ORDER BY时所有 NULL值被认为是相等的。

聚合函数，例如 [COUNT()](https://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html" \l "function_count)， [MIN()](https://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html" \l "function_min)，和 [SUM()](https://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html" \l "function_sum)忽略 NULL的值。 [COUNT(\*)](https://dev.mysql.com/doc/refman/5.7/en/group-by-functions.html" \l "function_count)是个例外，它计数行而不是单个列值的个数。例如，以下语句产生两个计数。第一个是表中行数的计数，第二个是列age中非NULL值的计数：

mysql> SELECT COUNT(\*), COUNT(age) FROM person;

对于某些数据类型，MySQL将特别处理NULL值。如果插入NULL值到一个 TIMESTAMP列，则当前日期和时间会被插入。如果插入 NULL值到具有AUTO\_INCREMENT 属性的整数或浮点列，则插入序列中的下一个数字。

对于使用LOAD DATA INFILE或SELECT ... INTO OUTFILE执行的文本文件导入或导出操作，NULL由\N表示。参见[Section 13.2.6, “LOAD DATA INFILE Syntax”](https://dev.mysql.com/doc/refman/5.7/en/load-data.html" \o "13.2.6 LOAD DATA INFILE Syntax)。在文本文件中使用\N不受SQL语句中\N被弃用的影响。

## 2、Schema对象名称

MySQL中的某些对象（包括数据库，表，索引，列，别名，视图，存储过程，分区，表空间和其他对象）的名称称为标识符。本节介绍MySQL中标识符的语法（包括大小写敏感等）。

标识符可以被引号包起来，也可以不包，如果标识符包含特殊字符或是保留字，则在使用时必须使用引号包起来。

标识符在内部被转换为Unicode。他们可能包含这些字符：

* 不需要引号包的字符：
  + ASCII：[0-9，a-z，A-Z $ \_ ]
  + 扩展：U + 0080 .. U + FFFF
* 需要引号包的字符包括完整的Unicode Basic多语言平面（BMP），但U + 0000除外：
  + ASCII：U + 0001 .. U + 007F
  + 扩展：U + 0080 .. U + FFFF
* ASCII NUL（U + 0000）和补充字符（U + 10000及更高）不允许在标识符中使用。
* 标识符可以以数字开头，但要求标识符不能仅由数字组成。
* 数据库，表和列名称不能以空格字符结尾。

标识符引号字符是backtick（`）。注意：不是双引号也不是单引号。

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

如果启用S​​QL模式[ANSI\_QUOTES](https://dev.mysql.com/doc/refman/5.7/en/sql-mode.html" \l "sqlmode_ansi_quotes)，也可以在双引号内引用标识符：

mysql> CREATE TABLE "test" (col INT);

ERROR 1064: You have an error in your SQL syntax...

mysql> SET sql\_mode='ANSI\_QUOTES';

mysql> CREATE TABLE "test" (col INT);

Query OK, 0 rows affected (0.00 sec)

ANSI\_QUOTES模式使服务器将双引号字符串解释为标识符。因此，当启用此模式时，字符串字面值必须包含在单引号内。它们不能包含在双引号内。

如果引用标识符（将标识符包起来的意思），标识符引用字符（包标识符的字符，即反引号）可以包含在标识符中。如果标识符中包含的字符与用于引用（包含）标识符本身的字符**相同**，则需要将字符加倍。下面的语句创建一个名为a`b 的表包含列c"d：

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

在查询的选择列表中，可以使用标识符或字符串引用字符指定引用的列**别名**：

mysql> SELECT 1 AS `one`, 2 AS 'two';

+-----+-----+

| one | two |

+-----+-----+

| 1 | 2 |

+-----+-----+

在其他地方，别名也必须使用反引号。否则将作为字符串对待。

选取标识符还会以有意义来选，不要选择一些奇奇怪怪的标识符。特别是在不同的上下文中有不同含义的情况。标识符就使用反引号引用。这样免得出错。

使用它[MD5()](https://dev.mysql.com/doc/refman/5.7/en/encryption-functions.html" \l "function_md5)来生成表名时要小心，因为它可以产生非法或不明确格式的名称。

**用户变量**不能直接在SQL语句中用作标识符或标识符的一部分。

数据库和表名中的特殊字符以相应的文件系统名称进行编码。每种标识符的限制如下表：

| **标识符** | **最大长度（字符）** |
| --- | --- |
| 数据库 | 64（NDB存储引擎：63） |
| 表 | 64（NDB存储引擎：63） |
| 列 | 64 |
| 索引 | 64 |
| 约束 | 64 |
| 存储程序 | 64 |
| 视图 | 64 |
| 表空间 | 64 |
| 服务器 | 64 |
| 日志文件组（Log File Group） | 64 |
| 别号 | 256（见下面的例外） |
| 复合语句标识 | 16 |
| 用户定义的变量 | 64 |

[CREATE VIEW](https://dev.mysql.com/doc/refman/5.7/en/create-view.html" \o "13.1.21 CREATE VIEW语法)根据最大列长度为64个字符（不是最大别名长度为256个字符）的限制来检查语句中列名称的别名。

使用Unicode（UTF-8）存储标识符。这适用于存储在.frm文件中的表定义中的标识符以及存储在mysql数据库中的授权表中的标识符。授权表中的标识符字符串列的大小以为单位进行测量。您可以使用多字节字符，而不会减少存储在这些列中的值所允许的字符数。如前所述，允许的Unicode字符是基本多语言平面（BMP）中的Unicode字符。不允许附加字符。

**1）标识符限定符**

有时候依赖于上下文，需要对标识符进行限定，用以指出这个标识符代表的对象在哪个数据库，哪个表中。例如：

CREATE TABLE t1 (i INT);

如果没有选择数据库，将会出错，因此必须先选择数据库或者指定一个数据库名作为t1的限定符。如：

CREATE TABLE db1.t1 (i INT);

将在db1数据库中创建一个表t1。明确指定的比默认选择的优先级高。

限定词具有以下特点：

* 不限定的名称由单个标识符组成。限定名称由多个标识符组成。
* 多个名称之间必须用period（.）字符分隔。
* 限定符是一个单独的标记，不需要与相关联的标识符相邻。

例如， ***tbl\_name.col\_name***和 ***tbl\_name . col\_name***是相等的。

* 如果多个名称的任何一个名称需要引用（包起来），请单独引用它们，而不是引用整个名称。例如， `my-table`.`my-column`，而不是 `my-table.my-column`。
* 在限定名称中的句点之后（例如DB名的句点之后）的保留字一定表示一个标识符，因此在该上下文中不需要引用，其他情况下，保留字必须用反引号引起来。
* 语法 ***.tbl\_name*** 表示***tbl\_name***默认数据库中的表。

注意：MySQL 5.7.20中不推荐使用此语法，并将在未来的版本中删除此语法。

对象名称允许的限定符取决于对象的类型：

* 数据库名称是完全限定的，不需要限定符：

CREATE DATABASE db1;

* 表，视图或存储的程序（包括存储过程和函数和事件等）名称可以被赋予数据库名限定符。CREATE声明中限定和不限定名称的示例：

CREATE TABLE mytable ...;

CREATE VIEW myview ...;

CREATE PROCEDURE myproc ...;

CREATE FUNCTION myfunc ...;

CREATE EVENT myevent ...;

CREATE TABLE mydb.mytable ...;

CREATE VIEW mydb.myview ...;

CREATE PROCEDURE mydb.myproc ...;

CREATE FUNCTION mydb.myfunc ...;

CREATE EVENT mydb.myevent ...;

* 触发器与表相关联，因此任何限定符都适用于表名称：

CREATE TRIGGER mytrigger ... ON mytable ...;

CREATE TRIGGER mytrigger ... ON mydb.mytable ...;

* 列名可以被赋予多个限定符，如下表所示。

| 列引用方式 | 含义 |
| --- | --- |
| *col\_name* | 包含*col\_name列名的表中的列* |
| *tbl\_name.col\_name* | 默认数据库下表*tbl\_name的col\_name列* |
| *db\_name.tbl\_name.col\_name* | *tbl\_name*数据库的*db\_name表的col\_name列* |

如果在语句中的列式明确的，那么你不需要为列名称指定限定符。不明确的情况下则必须指出，例如：列c1只在表 t1中出现，c2只在 t2中出现，c在 t1和t2中。则引用时就需要进行限定，表示在哪个表中的列c：

SELECT c1, c2, t1.c FROM t1 INNER JOIN t2 WHERE t2.c > 100;

又如：

SELECT c1, c2, db1.t.c FROM db1.t INNER JOIN db2.t WHERE db2.t.c > 100;

还可以使用别名：

SELECT c1, c2, t1.c FROM db1.t AS t1 INNER JOIN db2.t AS t2 WHERE t2.c > 100;

2）大小写敏感性

3）标识符和文件系统中文件的映射

4）函数名解析

## 3、关键字和保留字

## 4、用户定义变量

## 5、表达式语法

以下规则定义MySQL中的表达式语法。这里显示的语法是基于MySQL源代码分发文件中的sql/sql\_yacc.yy给出的 。

*expr*:

*expr* OR *expr*

| *expr* || *expr*

| *expr* XOR *expr*

| *expr* AND *expr*

| *expr* && *expr*

| NOT *expr*

| ! *expr*

| *boolean\_primary* IS [NOT] {TRUE | FALSE | UNKNOWN}

| *boolean\_primary*

*boolean\_primary*:

*boolean\_primary* IS [NOT] NULL

| *boolean\_primary* <=> *predicate*

| *boolean\_primary* *comparison\_operator* *predicate*

| *boolean\_primary* *comparison\_operator* {ALL | ANY} (*subquery*)

| *predicate*

*comparison\_operator*: = | >= | > | <= | < | <> | !=

*predicate*:

*bit\_expr* [NOT] IN (*subquery*)

| *bit\_expr* [NOT] IN (*expr* [, *expr*] ...)

| *bit\_expr* [NOT] BETWEEN *bit\_expr* AND *predicate*

| *bit\_expr* SOUNDS LIKE *bit\_expr*

| *bit\_expr* [NOT] LIKE *simple\_expr* [ESCAPE *simple\_expr*]

| *bit\_expr* [NOT] REGEXP *bit\_expr*

| *bit\_expr*

*bit\_expr*:

*bit\_expr* | *bit\_expr*

| *bit\_expr* & *bit\_expr*

| *bit\_expr* << *bit\_expr*

| *bit\_expr* >> *bit\_expr*

| *bit\_expr* + *bit\_expr*

| *bit\_expr* - *bit\_expr*

| *bit\_expr* \* *bit\_expr*

| *bit\_expr* / *bit\_expr*

| *bit\_expr* DIV *bit\_expr*

| *bit\_expr* MOD *bit\_expr*

| *bit\_expr* % *bit\_expr*

| *bit\_expr* ^ *bit\_expr*

| *bit\_expr* + *interval\_expr*

| *bit\_expr* - *interval\_expr*

| *simple\_expr*

*simple\_expr*:

*literal*

| *identifier*

| *function\_call*

| *simple\_expr* COLLATE *collation\_name*

| *param\_marker*

| *variable*

| *simple\_expr* || *simple\_expr*

| + *simple\_expr*

| - *simple\_expr*

| ~ *simple\_expr*

| ! *simple\_expr*

| BINARY *simple\_expr*

| (*expr* [, *expr*] ...)

| ROW (*expr*, *expr* [, *expr*] ...)

| (*subquery*)

| EXISTS (*subquery*)

| {*identifier* *expr*}

| *match\_expr*

| *case\_expr*

| *interval\_expr*

有关运算符优先级，参见 [第12.3.1节“运算符优先级”](https://dev.mysql.com/doc/refman/5.7/en/operator-precedence.html" \o "12.3.1运算符优先级)。

对于字面值语法，参见[第9.1节“字面值”](https://dev.mysql.com/doc/refman/5.7/en/literals.html" \o "9.1字面值)。

有关标识符语法，参见[第9.2节“Schema对象名称”](https://dev.mysql.com/doc/refman/5.7/en/identifiers.html" \o "9.2模式对象名称)。

变量可以是用户定义变量，系统变量或存储程序局部变量或参数：

* 用户变量：[第9.4节“用户自定义变量”](https://dev.mysql.com/doc/refman/5.7/en/user-variables.html" \o "9.4用户定义的变量)
* 系统变量：[第5.1.6节“使用系统变量”](https://dev.mysql.com/doc/refman/5.7/en/using-system-variables.html" \o "5.1.6使用系统变量)
* 局部变量：[第13.6.4.1节“局部变量DECLARE语法”](https://dev.mysql.com/doc/refman/5.7/en/declare-local-variable.html" \o "13.6.4.1局部变量DECLARE语法)
* 参数：[第13.1.16节“CREATE PROCEDURE和CREATE FUNCTION语法”](https://dev.mysql.com/doc/refman/5.7/en/create-procedure.html" \o "13.1.16 CREATE PROCEDURE和CREATE FUNCTION语法)

***param\_marker***是预处理语句的占位符即：?。参见 [第13.5.1节“预处理语句语法”](https://dev.mysql.com/doc/refman/5.7/en/prepare.html" \o "13.5.1 PREPARE语法)。

(***subquery***)表示返回单个值的子查询; 也就是标量子查询。见[第13.2.10.1节“子查询作为标量操作数”](https://dev.mysql.com/doc/refman/5.7/en/scalar-subqueries.html" \o "13.2.10.1子查询作为标量操作数)。

{***identifier*** ***expr***}是ODBC转义语法，被用于ODBC兼容性。值是 ***expr***。

***match\_expr***表示一个 [MATCH](https://dev.mysql.com/doc/refman/5.7/en/fulltext-search.html" \l "function_match)表达式。参见 [第12.9节“全文搜索功能”](https://dev.mysql.com/doc/refman/5.7/en/fulltext-search.html" \o "12.9全文搜索功能)。

***case\_expr***表示一个 [CASE](https://dev.mysql.com/doc/refman/5.7/en/control-flow-functions.html" \l "operator_case)表达式。参见 [第12.4节“控制流程功能”](https://dev.mysql.com/doc/refman/5.7/en/control-flow-functions.html" \o "12.4控制流程功能)。

***interval\_expr***表示时间间隔。语法是INTERVAL ***expr*** ***unit***，其中 unit是一个说明符表示一个时间段即一个时间单位，如 HOUR, DAY, 或者WEEK。有关unit可以取值的完整列表 ，参见 [12.7节，“日期和时间函数”](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html" \o "12.7日期和时间函数)中对 [DATE\_ADD()](https://dev.mysql.com/doc/refman/5.7/en/date-and-time-functions.html" \l "function_date-add)的描述。

一些运算符的含义取决于服务器SQL模式：

* 默认情况下，||是一个逻辑OR运算符。 [PIPES\_AS\_CONCAT](https://dev.mysql.com/doc/refman/5.7/en/sql-mode.html" \l "sqlmode_pipes_as_concat)启用后， ||是字符串连接，优先级在^与一元运算符之间 。
* 默认情况下，!的优先级高于NOT。[HIGH\_NOT\_PRECEDENCE](https://dev.mysql.com/doc/refman/5.7/en/sql-mode.html" \l "sqlmode_high_not_precedence)启用之后， !和 NOT具有相同的优先级。

参见[第5.1.8节“服务器SQL模式”](https://dev.mysql.com/doc/refman/5.7/en/sql-mode.html" \o "5.1.8服务器SQL模式)。

## 6、注释语法

MySQL服务器支持三种注释格式：

* 从#字符到行尾。
* 从-- 序列到行尾。在MySQL中，-- 注释样式需要第二个虚线后跟至少一个空格或控制字符（如空格，制表符，换行符等）。该语法与标准SQL注释语法略有不同。
* 如C编程语言一样，从/\*到 \*/之间的内容。这种语法使得注释可以扩展到多行。

以下示例演示了所有三种注释样式：

mysql> SELECT 1+1; # This comment continues to the end of line

mysql> SELECT 1+1; -- This comment continues to the end of line

mysql> SELECT 1 /\* this is an in-line comment \*/ + 1;

mysql> SELECT 1+

/\*

this is a

multiple-line comment

\*/

1;

MySQL服务器不支持嵌套注释。

MySQL服务器也支持C风格注释的一些变体。这使你能够编写包含特定于MySQL扩展的代码，但仍可移植，使用以下的注释方式：

/\*! *MySQL-specific code* \*/

在这种情况下，MySQL Server解析并执行注释中的SQL代码，就像其他SQL语句一样，但其他SQL服务器将忽略这些扩展。例如，MySQL Server 在以下语句中识别关键字STRAIGHT\_JOIN，但其他服务器将不会：

SELECT /\*! STRAIGHT\_JOIN \*/ col1 FROM table1,table2 WHERE ...

如果在! 字符后添加版本号，则仅当MySQL版本大于或等于指定版本号时，才会执行注释中的代码。以下注释中的关键字KEY\_BLOCK\_SIZE仅由MySQL 5.1.10或更高版本的服务器执行：

CREATE TABLE t1(a INT, KEY (a)) /\*!50110 KEY\_BLOCK\_SIZE=1024 \*/;

上述的注释语法适用于**[mysqld](https://dev.mysql.com/doc/refman/5.7/en/mysqld.html" \o "4.3.1 mysqld  -  MySQL服务器)**服务器如何解析SQL语句。在 **[MySQL的](https://dev.mysql.com/doc/refman/5.7/en/mysql.html" \o "4.5.1 mysql  -  MySQL命令行工具)**客户端程序将它们发送到服务器之前还执行语句的一些分析。（这是为了确定多语句输入行中的语句边界，这个在MySQL 5.7手册第三章中有介绍）

这种格式的注释/\*!12345 ... \*/不会存储在服务器上。如果此格式用于对存储的例程进行注释，则注释将不会保留在服务器上。

使用C风格注释语法的另一个变体是来指定优化器提示。即在/\*后紧跟一个+。例：

SELECT /\*+ BKA(t1) \*/ FROM ... ;