1. call语句

CALL *sp\_name*([*parameter*[,...]])

CALL *sp\_name*[()]

可以用call语句调用之前定义过的存储过程create procedure

没有参数的存储过程可以在没有括号的情况下调用。也就是说，CALL p()而且 CALL p是等价的。

CREATE PROCEDURE p (OUT ver\_param VARCHAR(25), INOUT incr\_param INT)

BEGIN

SELECT VERSION() INTO ver\_param;

SET incr\_param = incr\_param + 1;

END;

mysql> SET @increment = 10;

mysql> CALL p(@version, @increment);

mysql> SELECT @version, @increment;

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

| @version | @increment |

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

| 5.7.20-debug-log | 11 |

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

在[CALL](https://dev.mysql.com/doc/refman/5.7/en/call.html" \o "13.2.1 CALL语法)用于[PREPARE](https://dev.mysql.com/doc/refman/5.7/en/prepare.html)和的 预处理语句中[EXECUTE](https://dev.mysql.com/doc/refman/5.7/en/execute.html)，占位符可用于IN参数OUT和 INOUT参数。这些类型的参数可以使用如下：

mysql> SET @increment = 10;

mysql> PREPARE s FROM 'CALL p(?, ?)';

mysql> EXECUTE s USING @version, @increment;

mysql> SELECT @version, @increment;

处理[CALL](https://dev.mysql.com/doc/refman/5.7/en/call.html) 使用[mysql\_query()](https://dev.mysql.com/doc/refman/5.7/en/mysql-query.html)or 执行的语句 的结果[mysql\_real\_query()](https://dev.mysql.com/doc/refman/5.7/en/mysql-real-query.html)，请使用调用的循环[mysql\_next\_result()](https://dev.mysql.com/doc/refman/5.7/en/mysql-next-result.html)来确定是否有更多的结果

C程序可以使用prepared-statement接口来执行 [CALL](https://dev.mysql.com/doc/refman/5.7/en/call.html)语句，访问 OUT和INOUT参数。这是通过[CALL](https://dev.mysql.com/doc/refman/5.7/en/call.html)使用调用[mysql\_stmt\_next\_result()](https://dev.mysql.com/doc/refman/5.7/en/mysql-stmt-next-result.html)来确定是否有更多结果的循环来处理语句的结果来完成的

1. delete语句

[DELETE](https://dev.mysql.com/doc/refman/5.7/en/delete.html) 是从表中删除行的DML语句。

DELETE [LOW\_PRIORITY] [QUICK] [IGNORE] FROM *tbl\_name*

[PARTITION (*partition\_name* [, *partition\_name*] ...)] //支持分区删除

[WHERE *where\_condition*]

[ORDER BY ...]

[LIMIT *row\_count*]

可选WHERE子句中的条件标识要删除的行。没有WHERE 子句，所有行都被删除。

***where\_condition***是一个表达式，每一行被删除的结果都是true。

为了确保给定的[DELETE](https://dev.mysql.com/doc/refman/5.7/en/delete.html) 语句不占用太多时间，MySQL特定的 子句用于指定要删除的最大行数。如果要删除的行数大于限制，请重复该 语句，直到受影响的行数小于该值。 LIMIT ***row\_count*** [DELETE](https://dev.mysql.com/doc/refman/5.7/en/delete.html) DELETE LIMIT

多表删除：

DELETE [LOW\_PRIORITY] [QUICK] [IGNORE]

*tbl\_name*[.\*] [, *tbl\_name*[.\*]] ...

FROM *table\_references*

[WHERE *where\_condition*]

DELETE [LOW\_PRIORITY] [QUICK] [IGNORE]

FROM *tbl\_name*[.\*] [, *tbl\_name*[.\*]] ...

USING *table\_references*

[WHERE *where\_condition*]

您可以在[DELETE](https://dev.mysql.com/doc/refman/5.7/en/delete.html)语句中指定多个表，以 根据WHERE子句中的条件从一个或多个表中删除行 。你不能使用ORDER BY或LIMIT在一个多表 DELETE。

对于第一个多表语法，只FROM删除子句前列出的表中的匹配行。对于第二个多表语法，只删除FROM子句（子句之前USING）中列出的表中的匹配行。其效果是可以同时从多个表中删除行，并具有仅用于搜索的附加表：

DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3

WHERE t1.id=t2.id AND t2.id=t3.id;

分区删除：

DELETE FROM t PARTITION (p0)具有相同的效果

[ALTER TABLE t TRUNCATE PARTITION (p0)](https://dev.mysql.com/doc/refman/5.7/en/alter-table.html); 在这两种情况下，t表的分区p0中的所有行都被删除。

PARTITION可以与WHERE条件一起使用 ，在这种情况下，条件仅在列出的分区中的行上进行测试。例如， DELETE FROM t PARTITION (p0) WHERE c < 5 仅从p0条件c < 5为true的分区中删除行; 任何其他分区中的行不会被检查，因此不会受到影响 DELETE。

该PARTITION选项也可以用在多表DELETE语句中。在选项中，每个表格最多可以使用一个这样的 FROM选项。

修饰符：

该[DELETE](https://dev.mysql.com/doc/refman/5.7/en/delete.html" \o "13.2.2 DELETE语法)语句支持以下修饰符：

* 如果指定LOW\_PRIORITY，服务器会延迟执行，[DELETE](https://dev.mysql.com/doc/refman/5.7/en/delete.html" \o "13.2.2 DELETE语法) 直到没有其他客户端正在从表中读取。这会影响只使用表级锁只存储引擎（例如MyISAM，MEMORY和MERGE）。
* 对于MyISAM表，如果使用 QUICK修饰符，存储引擎在删除过程中不会合并索引叶，这可能会加速某些类型的删除操作。
* 该IGNORE修饰符使MySQL在删除行的过程中忽略错误。（在解析阶段遇到的错误以通常的方式进行处理。）由于使用而IGNORE被忽略的错误 将作为警告返回。

按顺序删除：

如果[DELETE](https://dev.mysql.com/doc/refman/5.7/en/delete.html)语句包含一个 ORDER BY子句，则按照子句指定的顺序删除行。这主要是有用的LIMIT。例如，以下语句查找与该WHERE子句匹配的行，对它们进行排序timestamp\_column，并删除第一个（最早的）一个：

DELETE FROM somelog WHERE user = 'jcole'

ORDER BY timestamp\_column LIMIT 1;

InnoDb表：

如果从大表中删除多行，则可能会超出表的锁表大小InnoDB。为了避免这个问题，或者只是为了尽量减少表保持锁定的时间，以下策略（根本不使用 [DELETE](https://dev.mysql.com/doc/refman/5.7/en/delete.html)）可能会有所帮助：

1. 选择不被删除的行到与原始表具有相同结构的空表中：

INSERT INTO t\_copy SELECT \* FROM t WHERE ... ;

1. 用[RENAME TABLE](https://dev.mysql.com/doc/refman/5.7/en/rename-table.html" \o "13.1.33 RENAME TABLE语法)原子的方式移动原始表格，并将副本重命名为原始名称：

RENAME TABLE t TO t\_old, t\_copy TO t;

1. 删除原来的表：

DROP TABLE t\_old;

1. Do 语句

[DO](https://dev.mysql.com/doc/refman/5.7/en/do.html)执行表达式但不返回任何结果。

[DO](https://dev.mysql.com/doc/refman/5.7/en/do.html)主要用于有副作用的功能，比如 [RELEASE\_LOCK()](https://dev.mysql.com/doc/refman/5.7/en/miscellaneous-functions.html#function_release-lock)。

例子：这个[SELECT](https://dev.mysql.com/doc/refman/5.7/en/select.html)语句暂停，但也产生一个结果集：

mysql> SELECT SLEEP(5);

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

| SLEEP(5) |

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

| 0 |

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

1 row in set (5.02 sec)

[DO](https://dev.mysql.com/doc/refman/5.7/en/do.html)另一方面，暂停而不产生结果集。mysql> DO SLEEP(5);

这可能是有用的，例如在一个存储的函数或触发器中，它禁止生成结果集的语句。

[DO](https://dev.mysql.com/doc/refman/5.7/en/do.html)只执行表达式。它不能在所有可以使用的情况下SELECT使用。例如，DO id FROM t1它是无效的，因为它引用了一个表。

1. Handler语句

该HANDLER语句提供对表存储引擎接口的直接访问。它可用于 InnoDB和MyISAM表格。

HANDLER ... OPEN语句打开一个表

用后续使它访问HANDLER ... READ语句

HANDLER ... CLOSE关闭了一个用来打开的表格HANDLER ... OPEN。。

第一种HANDLER ... READ语法提取所指定索引满足给定值并满足 WHERE条件的行。如果您有多列索引，请将索引列值指定为逗号分隔列表。

HANDLER ... READ my\_idx = (col\_a\_val,col\_b\_val,col\_c\_val) ...

第二种HANDLER ... READ语法按照与WHERE条件匹配的索引顺序从表中提取一行 。

第三种HANDLER ... READ语法按自然行顺序从表中获取与WHERE条件匹配的行 。它比想要全表扫描更快 。

有几个原因使用HANDLER 接口，而不是正常的[SELECT](https://dev.mysql.com/doc/refman/5.7/en/select.html) 语句：

1.HANDLER快于 [SELECT](https://dev.mysql.com/doc/refman/5.7/en/select.html)：

2.指定的存储引擎处理程序对象被分配给HANDLER ... OPEN。该对象被重HANDLER 用3.于该表的后续语句; 它不需要为每一个重新初始化。

4.涉及的解析较少。

5.没有优化器或查询检查开销。

6.处理程序接口不必提供一致的数据外观，因此存储引擎可以使用[SELECT](https://dev.mysql.com/doc/refman/5.7/en/select.html)通常不允许的优化。

7. HANDLER使您能够以难以（甚至不可能）完成的方式遍历数据库[SELECT](https://dev.mysql.com/doc/refman/5.7/en/select.html)。在 HANDLER使用向数据库提供交互式用户界面的应用程序时，界面是更自然地查看数据的方式。

8. HANDLER使您能够以难以（甚至不可能）完成的方式遍历数据库[SELECT](https://dev.mysql.com/doc/refman/5.7/en/select.html)。在 HANDLER使用向数据库提供交互式用户界面的应用程序时，界面是更自然地查看数据的方式。

5.inset 语句

1.insert …&&insert…on duplicate key update

INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]

[INTO] *tbl\_name*

[PARTITION (*partition\_name* [, *partition\_name*] ...)]

[(*col\_name* [, *col\_name*] ...)]

{VALUES | VALUE} (*value\_list*) [, (*value\_list*)] ...

[ON DUPLICATE KEY UPDATE *assignment\_list*]

INSERT [LOW\_PRIORITY | DELAYED | HIGH\_PRIORITY] [IGNORE]

[INTO] *tbl\_name*

[PARTITION (*partition\_name* [, *partition\_name*] ...)]

SET *assignment\_list*

[ON DUPLICATE KEY UPDATE *assignment\_list*]

*value*:

{*expr* | DEFAULT}

*value\_list*:

*value* [, *value*] ...

*assignment*:

*col\_name* = *value*

*assignment\_list*:

*assignment* [, *assignment*] ...

[INSERT](https://dev.mysql.com/doc/refman/5.7/en/insert.html)将新行插入到现有表中。该[INSERT ... VALUES](https://dev.mysql.com/doc/refman/5.7/en/insert.html)和 [INSERT ... SET](https://dev.mysql.com/doc/refman/5.7/en/insert.html) 语句的形式插入基于明确指定的值的行

如果要插入的行将导致UNIQUE索引或重复值PRIMARY KEY。[INSERT](https://dev.mysql.com/doc/refman/5.7/en/insert.html" \o "13.2.5 INSERT语法)使用 ON DUPLICATE KEY UPDATE子句可以使现有行更新，

Eg:

INSERT INTO tbl\_name (col1,col2) VALUES(15,col1\*2);

INSERT INTO tbl\_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);

2.insert…select

[INSERT ... SELECT](https://dev.mysql.com/doc/refman/5.7/en/insert-select.html)形式插入从另一个或多个表中选择行

INSERT [LOW\_PRIORITY | HIGH\_PRIORITY] [IGNORE]

[INTO] *tbl\_name*

[PARTITION (*partition\_name* [, *partition\_name*] ...)]

[(*col\_name* [, *col\_name*] ...)]

SELECT ...

[ON DUPLICATE KEY UPDATE *assignment\_list*]

Eg:

INSERT INTO tbl\_temp2 (fld\_id)

SELECT tbl\_temp1.fld\_order\_id

FROM tbl\_temp1 WHERE tbl\_temp1.fld\_order\_id > 100;

1. Load Data Infile

**要将表中的数据写入文件，请使用**[**SELECT ... INTO OUTFILE**](https://dev.mysql.com/doc/refman/5.7/en/select-into.html)**。**

LOAD DATA [LOW\_PRIORITY | CONCURRENT] [LOCAL] INFILE '*file\_name*'

[REPLACE | IGNORE]

INTO TABLE *tbl\_name*

[PARTITION (*partition\_name* [, *partition\_name*] ...)] //分区

[CHARACTER SET *charset\_name*]

[{FIELDS | COLUMNS}

[TERMINATED BY '*string*']

[[OPTIONALLY] ENCLOSED BY '*char*'] //如果指定OPTIONALLY的 ENCLOSED BY字符仅用于从具有字符串数据类型

[ESCAPED BY '*char*']

]

[LINES

[STARTING BY '*string*']

[TERMINATED BY '*string*']

]

[IGNORE *number* {LINES | ROWS}]

[(*col\_name\_or\_user\_var*

[, *col\_name\_or\_user\_var*] ...)]

[SET *col\_name*={*expr* | DEFAULT},

[, *col\_name*={*expr* | DEFAULT}] ...]

| **字符** | **转义序列** |
| --- | --- |
| \0 | 一个ASCII NUL（X'00'）字符 |
| \b | 退格字符 |
| \n | 换行符（换行符） |
| \r | 回车符 |
| \t | 制表符。 |
| \Z | ASCII 26（Control + Z） |
| \N | 空值 |

该[LOAD DATA INFILE](https://dev.mysql.com/doc/refman/5.7/en/load-data.html" \o "13.2.6 LOAD DATA INFILE语法)语句以非常高的速度从文本文件读取表格到表格中。

FIELDS和 LINES语句是相同的。两个子句都是可选的，但是如果两者都被指定，则FIELDS必须先行LINES。

LOW\_PRIORITY延迟执行 语句，直到没有其他客户端正在读取表格

CONCURRENT 使用基于行的复制，CONCURRENT无论MySQL版本如何，都会被复制。使用基于语句的复制CONCURRENT不会在MySQL 5.5.1之前复制

LOCAL 影响的文件和错误处理预期的位置LOCAL只有当你的服务器和你的客户端都被配置为允许时才有效。

LOAD DATA INFILE 'data.txt' INTO TABLE db2.my\_table;

如果指定了一个**FIELDS**子句，则每个子句（TERMINATED BY，， [OPTIONALLY] ENCLOSED BY和ESCAPED BY）也是可选的，除了必须至少指定其中的一个。

如果您指定no FIELDS或 LINESclause，则默认值与您写下此内容相同：

FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'

LINES TERMINATED BY '\n' STARTING BY ''

换句话说，[LOAD DATA INFILE](https://dev.mysql.com/doc/refman/5.7/en/load-data.html" \o "13.2.6 LOAD DATA INFILE语法)在读取输入时，默认值会导致 如下操作：

* 在换行符处查找行边界。
* 不要跳过任何行前缀。
* 将选项卡中的行分成几个字段。
* 不要期望字段被引用到任何引用字符中。
* 将转义字符前面的字符解释 \为转义序列。例如， \t，\n，和 \\分别表示选项卡，换行，反斜线。

相反，[SELECT ... INTO OUTFILE](https://dev.mysql.com/doc/refman/5.7/en/select-into.html" \o "13.2.9.1 SELECT ... INTO语法)写入输出时，默认设置的 作用如下：

* 写字段之间的标签。
* 不要将字段放在任何引号内。
* 使用\逃脱制表符，换行符的情况下，或\字段值内出现。
* 在行尾写新行。

**注意**

如果您在Windows系统上生成了该文本文件，则可能需要使用LINES TERMINATED BY '\r\n' 该文件才能正确读取该文件，因为Windows程序通常使用两个字符作为行终止符。有些程序，比如 **写字板**，\r在写文件时可能会用作行终止符。要阅读这些文件，请使用 LINES TERMINATED BY '\r'。

如果您想要读取的所有行都有一个要忽略的公共前缀，则可以使用跳过前缀以及前面的任何内容。如果一行不包含前缀，则整行将被跳过。假设您发出以下声明： LINES STARTING BY '***prefix\_string***'

LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test

FIELDS TERMINATED BY ',' LINES STARTING BY 'xxx';

如果数据文件如下所示：

xxx"abc",1

something xxx"def",2

"ghi",3

结果行将("abc",1)和 ("def",2)。文件中的第三行被跳过，因为它不包含前缀。

该选项可用于忽略文件开始处的行。例如，您可以使用跳过包含列名的初始标题行： IGNORE ***number*** LINESIGNORE 1 LINES

LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;

LOAD DATA INFILE 'file.txt'

INTO TABLE t1

(column1, column2)

SET column3 = CURRENT\_TIMESTAMP;