第5章 MySQL编程语言

CHAPTER 5

MySQL 程序设计结构是在 SQL 标准的基础上增加了一些程序设计语言的元素，其中包括变量、运算符、表达式、流程控制以及函数等内容。为了便于 MySQL 代码维护，以及提高 MySQL 代码的重用性，MySQL 开发人员将频繁使用的业务逻辑封装成诸如触发器、存储过程、函数等存储程序。

本章首先介绍了 MySQL 编程的基础知识，然后讲解了存储过程和存储函数的实现方法，最后介绍了 MySQL 触发器的使用。

5.1 运算符

运算符用于运算 MySQL 的表达式，它可以针对一个或以上操作数进行运算。当 MySQL 数据库中的表结构确立后，表中的数据所代表的意义就已经确定。而通过 MySQL 运算符进行运算，可以获得表结构以外的另一种数据。根据运算符功能的不同，可将 MySQL 的运算符分为 4 大类：算术运算符、比较运算符、逻辑运算符、位运算符。本节即将介绍各种运算符的特点和使用方法。

5.1.1 算术运算符

算术运算符是 SQL 中最基本的运算符，用于两个操作数之间执行算术运算。MySQL 中的算术运算符有＋(加)、－(减)、*(乘)、/(除)、%(求余)以及 div(求商)6 种运算符，如表 5-1 所示。

表 5-1 MySQL 中的算术运算符

算术运算符	说　明	算术运算符	说　明
＋	加法运算	/	除法运算，返回商
－	减法运算	%	求余运算，返回余数
*	乘法运算	div	求商运算

【例 5.1】 在当前数据库中创建表 tmp，定义一个数据类型为 INT 的字段 num，插入值 64，对 num 值进行算术运算。

首先，创建表 tmp，输入如下命令。

```
CREATE TABLE tmp (num INT);
```

向字段 num 插入数据 64,输入如下命令。

```
INSERT INTO tmp value(64);
```

接下来,对 num 值进行加法和减法运算,输入如下命令。

```
mysql> select num, num + 10, num - 3 + 5, num + 5 - 3, num + 36.7 FROM tmp;
+------+--------+---------+---------+----------+
| num  | num + 10 | num - 3 + 5 | num + 5 - 3 | num + 36.7 |
+------+--------+---------+---------+----------+
|   64 |     74 |      66 |      66 |    100.7 |
+------+--------+---------+---------+----------+
```

【例 5.2】 对 num 进行乘法和除法运算,输入如下命令。

```
mysql> select num, num * 2, num /2, num/3, num % 3 FROM tmp;
+------+-------+---------+---------+-------+
| num  | num * 2 | num /2  | num/3   | num % 3 |
+------+-------+---------+---------+-------+
|   64 |   128 | 32.0000 | 21.3333 |       1 |
+------+-------+---------+---------+-------+
1 row in set (0.00 sec)
```

通过观察计算结果可以看到,对 num 进行除法运算的时候,由于 64 无法被 3 整除,因此 MySQL 对 num/3 求商的结果保存到了小数点后面 4 位,结果为 21.3333;64 除以 3 的余数为 1,因此取余运算 num % 3 的结果为 1。

在数学运算时,除数为 0 的除法是没有意义的,因此除法运算中除数不能为 0,如果被 0 整除,则返回结果为 NULL。

【例 5.3】 若用 0 除 num,输入如下命令。

```
mysql> SELECT num, num / 0, num % 0 FROM tmp;
+------+---------+---------+
| num  | num / 0 | num % 0 |
+------+---------+---------+
|   64 |    NULL |    NULL |
+------+---------+---------+
1 row in set (0.00 sec)
```

可见,对 num 进行除法求商、求余运算的结果均为 NULL。

5.1.2 比较运算符

比较运算符(又称关系运算符)用于比较操作数之间的大小关系,其运算结果要么为 TRUE,要么为 FALSE,否则为 NULL(不确定)。MySQL 中比较运算符经常在 SELECT 的查询条件子句中使用,用来查询满足指定条件的记录。比较结果为真,则返回 1,为假则返回 0,比较结果不确定则返回 NULL。MySQL 中的比较运算符如表 5-2 所示。

1. 等于运算符“=”

“=”用来比较两边的操作数是否相等,若相等返回 1,不相等返回 0。具体的语法规则如下。

表 5-2　MySQL 中的比较运算符

运　算　符	作　　用
=	等于
<=>	安全等于
<>(!=)	不等于
<=	小于或等于
>=	大于或等于
>	大于
IS NULL	判断一个值是否为 NULL
IS NOT NULL	判断一个值是否不为 NULL
LEAST	在有两个或多个参数时，返回最小值
GREATEST	当有两个或多个参数时，返回最大值
BETWEEN AND	判断一个值是否落在两个值之间
ISNULL	与 IS NULL 作用相同
IN	判断一个值是 IN 列表中的任意一个值
NOT IN	判断一个值不是 IN 列表中的任意一个值
LIKE	通配符匹配
REGEXP	正则表达式匹配

- 若有一个或两个操作数为 NULL，则比较运算的结果为 NULL。
- 若两个操作数都是字符串，则按照字符串进行比较。
- 若两个操作数均为整数，则按照整数进行比较。
- 若一个操作数为字符串，另一个操作数为数字，MySQL 自动将字符串转换为数字后再进行比较。

【例 5.4】　使用“=”进行相等判断，输入如下命令。

```
mysql> SELECT 1 = 0, '2' = 2, 2 = 2, '0.02' = 0, 'b' = 'b', (1 + 3) = (2 + 2), NULL = NULL;
+-----+-------+-----+----------+---------+---------------+------------+
| 1 = 0 | '2' = 2 | 2 = 2 | '0.02' = 0 | 'b' = 'b' | (1 + 3) = (2 + 2) | NULL = NULL |
+-----+-------+-----+----------+---------+---------------+------------+
|   0 |     1 |   1 |        0 |       1 |             1 |       NULL |
+-----+-------+-----+----------+---------+---------------+------------+
1 row in set (0.01 sec)
```

可以看出，在进行比较之前，MySQL 自动进行了转换，把字符'2'转换成了数字 2，则 2=2 和'2'=2 的返回值相同，都为 1；表达式'b'='b'为相同的字符比较，返回值为 1；表达式 1+3 和表达式 2+2 的结果都为 4，结果相等，返回值为 1；由于“=”不能用于空值 NULL 的判断，因此返回值为 NULL。

2. 安全等于运算符“<=>”

该运算符“<=>”和“=”运算符都是执行比较操作，与“=”的区别是它可以判断 NULL 值，具体的语法规则如下。

- 当两个操作数均为 NULL 时，返回值为 1。
- 若一个操作数不为 NULL，另一个操作码为 NULL 时，返回值为 0 而不是 NULL。

【例 5.5】 用运算符“<=>”进行安全等于判断,输入如下命令。

```
mysql> SELECT 1<=>0, '2'<=>2, 2<=>2, 'b'<=>'b', (1+3)<=>(2+2), NULL<=>NULL;
+-------+---------+-------+-----------+---------------+--------------+
| 1<=>0 | '2'<=>2 | 2<=>2 | 'b'<=>'b' | (1+3)<=>(2+2) | NULL<=>NULL  |
+-------+---------+-------+-----------+---------------+--------------+
|     0 |       1 |     1 |         1 |             1 |            1 |
+-------+---------+-------+-----------+---------------+--------------+
1 row in set (0.00 sec)
mysql> select NULL<=>'1234', NULL<=>123, '456'<=>NULL;
+----------------+------------+--------------+
| NULL<=>'1234'  | NULL<=>123 | '456'<=>NULL |
+----------------+------------+--------------+
|              0 |          0 |            0 |
+----------------+------------+--------------+
1 row in set (0.00 sec)
```

3. 不等于运算符“<>”或者“!=”

“<>”或者“!=”用于判断数字、字符串、表达式不相等的判断。如果不相等,返回值为1;否则返回值为0。这两个运算符不能用于判断空值NULL。

【例 5.6】 用“<>”和“!=”进行不相等判断,输入如下命令。

```
mysql>  SELECT 'good'<>'god', 1<>2, 4!=4, 5.5!=5, (1+3)!=(2+1), NULL<>NULL;
+----------------+------+------+--------+--------------+-------------+
| 'good'<>'god'  | 1<>2 | 4!=4 | 5.5!=5 | (1+3)!=(2+1) | NULL<>NULL  |
+----------------+------+------+--------+--------------+-------------+
|              1 |    1 |    0 |      1 |            1 |        NULL |
+----------------+------+------+--------+--------------+-------------+
1 row in set (0.00 sec)
```

由结果可以看到,两个不等于运算符作用相同,都可以进行数字、字符串、表达式的比较判断。

4. 小于运算符“<”和小于或等于运算符“<=”

“<”(或“<=”)运算符用来判断左边的操作数是否小于(或小于或等于)右边的操作数,如果小于(或小于或等于),返回值为1;否则,返回值为0;但这两个运算符不能用于判断空值NULL。

5. 大于运算符“>”和大于或等于运算符“>=”

“>”(或“>=”)运算符用于判断左边的操作数是否大于(或大于或等于)右边的操作数,如果大于(或大于或等于),返回值为1;否则返回值为0,但是不能用于判断空值NULL。

【例 5.7】 分别使用“<”“<=”“>”和“>=”进行判断,输入如下命令。

```
mysql> SELECT 'good'<'god',1<=2, 4>4,5>=5,(1+3)>(2+1),NULL<NULL;
+----------------+------+-----+------+-------------+------------+
| 'good'<'god'   | 1<=2 | 4>4 | 5>=5 | (1+3)>(2+1) | NULL<NULL  |
+----------------+------+-----+------+-------------+------------+
|              0 |    1 |   0 |    1 |           1 |       NULL |
+----------------+------+-----+------+-------------+------------+
1 row in set (0.00 sec)
```

6. IN、NOT IN 运算符

IN 运算符用于判断操作数是否为 IN 列表的其中一个值，如果是，返回值为 1；否则返回值为 0。

NOT IN 运算符用于判断表达式是否为 IN 列表中的其中一个值，如果不是，返回值为 1；否则返回值为 0。

【例 5.8】 分别使用 IN 和 NOT IN 运算符进行判断，输入如下命令。

```
mysql> SELECT 2 IN (1, 3, 5,'thks'), 'thinks' NOT IN (1, 3, 5,'thks');
+-------------------------+-------------------------------------+
| 2 IN (1, 3, 5,'thks')   | 'thinks' NOT IN (1, 3, 5,'thks')    |
+-------------------------+-------------------------------------+
|                       0 |                                   1 |
+-------------------------+-------------------------------------+
1 row in set, 2 warnings (0.01 sec)
```

7. IS NULL、IS NOT NULL 运算符

IS NULL 检验一个值是否为 NULL，如果为 NULL，返回值为 1，否则返回值为 0；IS NOT NULL 检验一个值是否为非 NULL，如果是，那么返回值为 1；否则返回值为 0。

【例 5.9】 使用 IS NULL 和 IS NOT NULL 判断 NULL 值和非 NULL 值，输入如下命令。

```
mysql> SELECT NULL IS NULL, ISNULL(NULL), ISNULL(10), 10 IS NOT NULL;
+---------------+---------------+-------------+-----------------+
| NULL IS NULL  | ISNULL(NULL)  | ISNULL(10)  | 10 IS NOT NULL  |
+---------------+---------------+-------------+-----------------+
|             1 |             1 |           0 |               1 |
+---------------+---------------+-------------+-----------------+
1 row in set (0.00 sec)
```

由结果可以看到，IS NULL 和 ISNULL 的作用相同，只是格式不同，ISNULL 和 IS NOT NULL 的返回值正好相反。

8. BETWEEN AND 运算符

语法格式为

```
expr BETWEEN min AND max
```

假如 expr 大于或等于 min 且小于或等于 max，则 BETWEEN 的返回值为 1，否则返回值为 0。

【例 5.10】 使用 BETWEEN AND 进行区间判断，输入如下命令。

```
mysql> SELECT 4 BETWEEN 3 AND 5, 12 BETWEEN 7 AND 10;
+--------------------+----------------------+
| 4 BETWEEN 3 AND 5  | 12 BETWEEN 7 AND 10  |
+--------------------+----------------------+
|                  1 |                    0 |
+--------------------+----------------------+
1 row in set (0.00 sec)
```

由结果可以看到，4 在端点值区间内时，BETWEEN AND 表达式返回值为 1；12 并不在指定区间内，因此返回值为 0。注意，对于字符串类型的比较，按字母表中字母顺序进行比较。

9. LEAST(值 1，值 2，…，值 n)函数

其中，值 n 表示参数列表中有 n 个值，在有两个或多个参数的情况下，返回最小值。假如任意一个自变量为 NULL，则 LEAST 函数的返回值为 NULL。

10. GREATEST(值 1，值 2，…，值 n)函数

与 LEAST 函数相反，用来获得多个值中的最大值，其中，n 表示参数列表中有 n 个值。当有两个或多个参数时，返回值为最大值，假如任意一个自变量为 NULL，则 GREATEST 的返回值为 NULL。

【例 5.11】 使用 LEAST 和 GREATEST 函数进行大小判断，SQL 语句如下。

```
mysql > SELECT LEAST (13, 3.0, 103.5), LEAST ('a', 'd', 'f'), LEAST (20, NULL);
+-----------------------+-----------------------+------------------+
| LEAST (13, 3.0,103.5)  | LEAST ('a', 'd', 'f')  | LEAST (20, NULL)  |
+-----------------------+-----------------------+------------------+
|                  3.0  |                     a  |            NULL  |
+-----------------------+-----------------------+------------------+
1 row in set (0.00 sec)
mysql > SELECT GREATEST (13, 3.0, 103.5), GREATEST ('a', 'd', 'f'), GREATEST(20, NULL);
+-------------------------+-------------------------+--------------------+
| GREATEST (13, 3.0, 103.5)  | GREATEST ('a', 'd', 'f')  | GREATEST(20, NULL)  |
+-------------------------+-------------------------+--------------------+
|                    103.5  |                     f  |               NULL  |
+-------------------------+-------------------------+--------------------+
1 row in set (0.00 sec)
```

由结果可以看到，当参数中是整数或者浮点数时，LEAST 将返回其中最小的值，而 GREATEST 返回其中的最大值；当参数为字符串时，LEAST 返回字母表中顺序最靠前的字符，而 GREATEST 返回字母表中顺序最靠后的字符；当比较值列表中有 NULL 时，不能判断大小，返回值为 NULL。

11. LIKE 运算符

LIKE 运算符用来匹配字符串，语法格式为

```
expr LIKE 匹配条件
```

如果 expr 满足匹配条件，则返回值为 1(TRUE)；如果不匹配，则返回值为 0(FALSE)。若 expr 或匹配条件中任何一个为 NULL，则结果为 NULL。LIKE 运算符在进行匹配时，可以使用下面两种通配符。

- %：匹配任何数目的字符，甚至包括零字符。
- _：只能匹配一个字符。

【例 5.12】 使用运算符 LIKE 进行字符串匹配运算，输入命令如下。

```
mysql > SELECT 'mylib' LIKE 'myli_','mylib' LIKE '%b', 'tear' LIKE 't_ _', 's' LIKE NULL;
+--------------------+----------------+-----------------+----------------+
| 'mylib' LIKE 'myli_'  | 'mylib' LIKE '%b'  | 'tear' LIKE 't_ _'  | 's' LIKE NULL  |
```

```
+--------------------+----------------+-----------------+----------------+
|                  1 |              1 |               0 |           NULL |
+--------------------+----------------+-----------------+----------------+
1 row in set (0.00 sec)
```

由结果可以看出，'mylib' 表示匹配以 mylib 开头的长度为 5 个字符的字符串，'mylib' 正好是 5 个字符，满足匹配条件返回 1；'%b' 表示匹配以字母 b 结尾的字符串，'mylib' 满足匹配条件，匹配成功返回 1；'t_ _' 表示匹配以 t 开头的长度为 3 个字符的字符串，而 'tear' 长度为 4，不满足匹配条件，因此返回 0；字符 's' 与 NULL 匹配时，结果为 NULL。

12. REGEXP 正则表达式

REGEXP 正则表达式用来匹配字符串，语法格式为

```
expr REGEXP 匹配条件
```

如果 expr 满足匹配条件，返回 1；如果不满足，则返回 0；若 expr 或匹配条件任意一个为 NULL，则结果为 NULL。

REGEXP 正则表达式在进行匹配时，常用的有下面几种通配符。

- ^：匹配字符串的开始位置，如 '^a' 表示以字母 a 开头的字符串。
- $：匹配字符串的结束位置，如 'X$' 表示以字母 X 结尾的字符串。
- .：这个字符就是英文半角的句号，它匹配任何一个字符，包括回车、换行等。
- [...]：匹配在方括号内的任何一个字符。如 '[abc]' 匹配 a、b 或 c 任意单个字符。
- -：使用'-'可以指定命名字符的范围，如 '[a-z]' 匹配 a～z 的任何一个字母，而 '[0-9]' 匹配 0～9 任何一位数字。
- *：匹配零个或多个在它前面的字符，在它之前必须有内容。如 'x*' 匹配任何长度的 x 字符，'[0-9]*' 匹配任何长度的数字，而 '*' 匹配任何长度的任何字符。

【例 5.13】　使用正则表达式 REGEXP 进行字符串匹配运算，输入如下命令。

```
mysql> SELECT 'mylib' REGEXP '^m', 'mylib' REGEXP 'b$', 'tear' REGEXP '[ab]';
+---------------------+---------------------+---------------------+
| 'mylib' REGEXP '^m' | 'mylib' REGEXP 'b$' | 'tear' REGEXP '[ab]' |
+---------------------+---------------------+---------------------+
|                   1 |                   1 |                   1 |
+---------------------+---------------------+---------------------+
1 row in set (0.02 sec)
```

由结果可以看出，匹配字符串为'mylib'，'^m' 表示匹配任何以字母 m 开头的字符串，'mylib' 满足条件返回 1；'b$' 表示任何以字母 b 结尾的字符串，'mylib' 以 b 结尾满足匹配条件返回 1；'[ab]' 匹配任何包含字母 a 或者 b 的字符串，字符串 'tear' 中虽然没有字母 b 但有字母 a，因此满足匹配条件返回 1。

5.1.3　逻辑运算符

逻辑运算符（又称布尔运算符）对布尔类型操作数进行运算，在 SQL 中逻辑运算结果为 TRUE、FALSE 或 NULL。在 MySQL 中它们体现为 1（TRUE）、0（FALSE）和空（NULL），

其中大多数都可以和 SQL 通用。MySQL 中的逻辑运算符如表 5-3 所示。

表 5-3 MySQL 中的逻辑运算符

运 算 符	作 用	运 算 符	作 用
NOT 或 !	逻辑非	OR 或 ||	逻辑或
AND 或 &&	逻辑与	XOR	逻辑异或

1. 与运算

"&&"或者"AND"是"与"运算的两种表达方式。如果所有操作数不为 0 且不为空值(NULL),则结果返回 1;如果存在任何一个操作数为 0,则结果返回 0;如果存在一个操作数为 NULL 且没有操作数为 0,则结果返回 NULL。"与"运算符支持多个操作数同时进行运算,按照运算优先级顺序执行。

2. 或运算

"||"或者"OR"表示"或"运算。所有操作数中存在任何一个非 0 数据时,运算结果返回 1;如果操作数中不包含非 0 的数据,但包含 NULL 时,运算结果返回 NULL;如果操作数中只有 0 时,运算结果返回 0。"或"运算符支持多个操作数同时进行运算,按照运算优先级顺序执行。

【例 5.14】 分别使用与运算符和或运算符进行逻辑运算,输入命令如下。

```
mysql > SELECT 1 AND - 1, 1 && 0, 1 && NULL, 0 && NULL, 1 || 0, 1 || NULL, 0 || NULL;
+---------+------+-----------+----------+--------+-----------+----------+
| 1 AND - 1 | 1 && 0  | 1 && NULL     | 0 && NULL   | 1 || 0    | 1 || NULL     | 0 || NULL   |
+---------+------+-----------+----------+--------+-----------+----------+
|         1 |       0|       NULL    |          0 |       1   |          1 |       NULL |
+---------+------+-----------+----------+--------+-----------+----------+
1 row in set (0.00 sec)
```

由结果可以看到,表达式"1 AND -1"中没有 0 或者 NULL,因此结果为 1;表达式"1 && 0"中有操作数 0,因此结果为 0;表达式"1 && NULL"中虽然有 NULL,但是没有操作数 0,返回结果为 NULL;表达式 0 && NULL 有 0,所以结果为 0;表达式"1 || 0"中包含非 0 的值 1,返回结果为 1;"1 || NULL"中虽然有 NULL,但是有操作数 1,返回结果为 1;"0 || NULL"中有 0 值,并且有 NULL,返回结果为 NULL。

3. 非运算

"!"或者 NOT 表示"非"运算。通过"非"运算,将返回与操作数相反的结果。如果操作数是非 0 的数字,结果返回 0;如果操作数是 0,结果返回 1;如果任何一个操作数是 NULL,结果返回 NULL。

【例 5.15】 利用非运算符进行逻辑判断,输入如下命令。

```
mysql > SELECT NOT 12, NOT (3 - 3), NOT NULL, NOT (NOT NULL);
+--------+-----------+-----------+---------------+
| NOT 12    | NOT (3 - 3)    | NOT NULL     |NOT (NOT NULL)      |
+--------+-----------+-----------+---------------+
|       0   |          1    |     NULL    |           NULL     |
+--------+-----------+-----------+---------------+
1 row in set (0.00 sec)1
```

4. 异或运算

XOR 表示“异或”运算。当其中一个操作数是真而另外一个操作数是假时，运算返回结果为真；当两个操作数都是真或都是假时，运算返回结果为假。即两个操作数不同时，结果为真；操作数相同时，结果为假。有任意一个操作数为 NULL 时，结果返回 NULL。

【例 5.16】　使用异或运算符“XOR”进行逻辑判断，输入如下命令。

```
mysql> SELECT 1 XOR 1, 0 XOR 0, 1 XOR 0, 1 XOR NULL, NULL XOR NULL;
+---------+---------+---------+-------------+----------------+
| 1 XOR 1 | 0 XOR 0 | 1 XOR 0 | 1 XOR NULL  | NULL XOR NULL  |
+---------+---------+---------+-------------+----------------+
|       0 |       0 |       1 |        NULL |           NULL |
+---------+---------+---------+-------------+----------------+
1 row in set (0.00 sec)
```

由结果可以看到，表达式“1 XOR 1”和“0 XOR 0”中运算符两边的操作数相同，返回结果为 0；表达式“1 XOR 0”中两边的操作数不同，返回结果为 1；表达式“1 XOR NULL”和“NULL XOR NULL”中有操作数为 NULL，则返回结果为 NULL。

5.1.4　位运算符

位运算符对二进制类型数据进行运算，如果不是二进制类型的数，系统将自动进行类型转换为二进制之后再运算，运算结果为二进制类型数据。使用 SELECT 语句显示运算结果时，系统将其自动转换为十进制数显示。MySQL 中的位运算符如表 5-4 所示。

表 5-4　MySQL 中的位运算符

运　算　符	作　　用	运　算　符	作　　用
&	按位与	~	按位取反
|	按位或	>>	位右移
^	按位异或	<<	位左移

1. 位与运算符 &

位与运算“&”是对两个操作数的二进制数，按对应数位做逻辑“与”操作。例如，表达式 2&3，由于 2 的二进制数为 10，3 的二进制数为 11，10&11 的结果为 10，即结果为十进制整数 2。表达式 2&3&4，其中 2 的二进制为 010，3 的二进制为 011，4 的二进制为 100，010&&011&100 的结果为 0。

2. 位或运算符 |

位或运算“|”是对两个操作数的二进制数，按对应数位做逻辑“或”操作。即对应的二进制位只要有一个或全为 1，则该位的运算结果为 1，否则为 0。

3. 位异或运算符 ^

位异或运算符“^”是对两个操作数的二进制数，按对应数位进行“异或”操作。即当两个操作数对应位的二进制数不同时，位结果为 1；当对应位相同时，位结果为 0。

4. 位取反运算符~

位取反运算符“~”是对两个操作数的二进制数，按对应数位做“非”操作，这里的操作数只能是一位，1 的位取反得值为 0，而 0 的位取反得值为 1。

【例 5.17】 使用位与、位或和位异或运算符运算，输入如下命令。

```
mysql > SELECT 10 | 15, 10 & 15, 10 ^ 15, 5 & ~1, ~1;
+---------+---------+---------+--------+----------------------+
| 10 | 15 | 10 & 15 | 10 ^ 15 | 5 & ~1 | ~1                  |
+---------+---------+---------+--------+----------------------+
|      15 |      10 |       5 |      4 |18446744073709551614  |
+---------+---------+---------+--------+----------------------+
1 row in set (0.00 sec)
```

十进制数据 10 的二进制数值为 1010，15 的二进制数值为 1111，按位或运算之后，结果为 1111，即十进制整数 15；同理，这两个数按位与运算之后，结果为 1010，即十进制整数 10；按位异或运算之后，结果为 0101，即十进制整数 5；表达式 5&~1 中，由于取反运算符“~”的优先级高于位与运算符“&”，因此先对 1 取反操作，取反之后，除了最低位为 0 其他位都为 1，然后再与十进制数值 5 进行与运算，结果为 0100，即十进制整数 4。

在 MySQL 中，常量数字默认会以 8B 来表示，8B 就是 64b，常量 1 的二进制表示为 63 个“0”加“1”，按位取反的值为 63 个“1”加一个“0”，转换为二进制后就是 18446744073709551614。

5. 位右移运算符>>

位右移运算符“>>”是对操作数向右移动指定的位数，右移指定位数之后，右边低位的数值将被移出并丢弃，左边高位空出的位置用 0 补齐。

语法格式为：expr >> *n*

这里的 *n* 为指定 expr 要移位的位数。

6. 位左移运算符<<

位左移运算符“<<”是操作数向左移动指定的位数。左移指定位数之后，左边高位的数值将被移出并丢弃，右边低位空出的位置用 0 补齐。

语法格式为：expr << *n*

这里的 *n* 为指定 expr 要移位的位数。

【例 5.18】 使用位左移和位右移运算符进行运算，输入如下命令。

```
mysql > SELECT  1 << 2, 4 << 2, 1 >> 1, 16 >> 2;
+------+------+------+-------+
| 1 << 2 | 4 << 2 | 1 >> 1 | 16 >> 2 |
+------+------+------+-------+
|    4 |   16 |    0 |     4 |
+------+------+------+-------+
1 row in set (0.00 sec)
```

从结果可以看出，1 的二进制值为 0001，左移两位之后变成 0100，即十进制整数 4；十进制 4 左移两位之后变成 0001 0000，即变成十进制的 16；十进制整数 1 的二进制值为 0001，右移 1 位之后变成 0000，即十进制整数 0；十进制整数 16 的二进制值为 0001 0000，右移两位之后变成 0000 0100，即变成十进制整数 4。

5.1.5　运算符的优先级

运算符的优先级决定了不同的运算符在表达式中计算的先后顺序，表 5-5 列出了 MySQL 中的各类运算符及其优先级。

表 5-5　运算符按优先级由低到高排列

优先级顺序（由低到高）	运　算　符
1	=（赋值运算），:=
2	||，OR
3	XOR
4	&&，AND
5	NOT
6	BETWEEN，CASE，WHEN，THEN，ELSE
7	=（比较运算），<=>，>=，>，<=，<，<>，!=，IS，LIKE，REGEXP，IN
8	|
9	&
10	<<，>>
11	-，+
12	*，/（DIV），%（MOD）
13	^
14	-（负号），~（按位取反）
15	!

可以看出，不同运算符的优先级是不同的。可用如下顺口溜记忆：先单目，后双目；先算术，后关系，再逻辑；同等级别从左向右顺序算，括号最优先。

一般情况下，级别高的运算符先进行计算，如果级别相同，MySQL 表达式按从左到右的顺序依次计算。当然，在无法确定优先级的情况下，可以使用半角圆括号（ ）来改变优先级，并且这样会使计算过程更加清晰。

5.1.6　MySQL 中的变量

MySQL 中的变量分为全局变量、会话变量、用户变量和局部变量。

1. 全局变量

MySQL 服务器维护了许多系统变量来控制其运行的行为，在服务器启动时会使用这些内置的变量和配置文件中的变量来初始化整个 MySQL 服务器的运行环境，这些变量通常就是人们所说的全局变量，全局变量的名称前加@@。

这些变量大部分是可以由 root 用户通过 SET 命令直接在运行时来修改的，一旦 MySQL 服务器重新启动，所有修改都被还原。如果修改了配置文件，想恢复最初的设置，只需要将配置文件还原，重新启动 MySQL 服务器，一切都可以恢复原始状态。

查询所有的全局变量指令为

```
show global variables;
```

但一般不会这么用，因为全局变量太多了，大概有 500 多个，通常使用 like 控制过滤条

件,输入如下命令。

```
mysql > show global variables like 'sql % ';
+---------------------------+-----------------------+
| Variable_name             | Value                 |
+---------------------------+-----------------------+
| sql_auto_is_null          | OFF                   |
| sql_big_selects           | ON                    |
| sql_buffer_result         | OFF                   |
| sql_log_off               | OFF                   |
| sql_mode                  |                       |
| sql_notes                 | ON                    |
| sql_quote_show_create     | ON                    |
| sql_require_primary_key   | OFF                   |
| sql_safe_updates          | OFF                   |
| sql_select_limit          | 18446744073709551615  |
| sql_slave_skip_counter    | 0                     |
| sql_warnings              | OFF                   |
+---------------------------+-----------------------+
12 rows in set, 1 warning (0.00 sec)
```

使用以下 SQL 指令,可以查询指定名称的全局变量。

```
select  @@全局变量名;
```

例如,查询 sql_mode 和 sql_notes 两个全局变量的值,输入如下命令。

```
mysql > select  @@sql_mode;
+--------------------------------------------+
| @@sql_mode                                 |
+--------------------------------------------+
| STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION |
+--------------------------------------------+
1 row in set (0.00 sec)
mysql > select  @@sql_notes;
+--------------+
| @@sql_notes  |
+--------------+
|            1 |
+--------------+
1 row in set (0.00 sec)
```

2. 会话变量

当有客户机连接到 MySQL 服务器的时候,MySQL 服务器会将这些全局变量的大部分复制一份作为这个连接客户机的会话变量,这些会话变量与客户机连接绑定,连接的客户机可以修改其中允许修改的变量,但是当连接断开时这些会话变量将全部消失,重新连接时会从全局变量中重新复制一份。

查询所有会话变量的指令是:

```
show session variables;
```

也可以添加过滤条件，查询部分会话变量，输入如下命令。

```
mysql > show session variables like 'sql % ';
+---------------------------+---------------------------------------------+
| Variable_name             | Value                                       |
+---------------------------+---------------------------------------------+
| sql_auto_is_null          | OFF                                         |
| sql_big_selects           | ON                                          |
| sql_buffer_result         | OFF                                         |
| sql_log_bin               | ON                                          |
| sql_log_off               | OFF                                         |
| sql_mode                  | STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION  |
| sql_notes                 | ON                                          |
| sql_quote_show_create     | ON                                          |
| sql_require_primary_key   | OFF                                         |
| sql_safe_updates          | OFF                                         |
| sql_select_limit          | 18446744073709551615                        |
| sql_slave_skip_counter    | 0                                           |
| sql_warnings              | OFF                                         |
+---------------------------+---------------------------------------------+
13 rows in set, 1 warning (0.00 sec)
```

由于会话变量复制于全局变量，所以它也是以@@开头的。它与客户机连接有关，无论怎样修改，当连接断开后，一切都会还原，下次连接时又是一次新的开始。查询特定的会话变量，可以使用以下三种形式的 SQL 指令，输入如下命令。

```
mysql > select @@session.sql_mode; # 形式一，查询指定全局会话变量
+-------------------------------------------+
| @@session.sql_mode                        |
+-------------------------------------------+
| STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION|
+-------------------------------------------+
1 row in set (0.00 sec)

mysql > select @@local.sql_mode; # 形式二，查询本地连接会话变量
+-------------------------------------------+
| @@local.sql_mode                          |
+-------------------------------------------+
| STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION|
+-------------------------------------------+
1 row in set (0.00 sec)

 mysql > select @@sql_mode; # 形式三，查询指定全局变量
+-------------------------------------------+
| @@sql_mode                                |
+-------------------------------------------+
| STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION|
+-------------------------------------------+
1 row in set (0.00 sec)
```

3. 用户变量

MySQL 中用户是通过建立与服务器连接的方式来使用数据库服务资源的，一个连接

就是一个会话,因此,除了会话变量外,用户还可以自己定义的变量。用户变量名用@开头,当连接断开时用户变量失效,用户变量的定义和使用相比会话变量来说简单许多。

查询用户变量的指令是:

```
select @用户变量名;
```

可以使用 set 命令对用户变量赋值,使用 select 查询用户变量,输入如下命令。

```
mysql > set @count = 1; # 创建用户变量 count,并赋初值 1
Query OK, 0 rows affected (0.00 sec)
mysql > set @sum = 0; # 创建用户变量 sum,并赋初值 0
Query OK, 0 rows affected (0.00 sec)
mysql > select @count,@sum; # 查询用户变量 count 和 sum
+--------+------+
| @count | @sum |
+--------+------+
|      1 |    0 |
+--------+------+
1 row in set (0.00 sec)
```

4. 局部变量

MySQL 中使用 DECLARE 关键字定义局部变量,局部变量通常出现在存储过程和存储函数中,用于计算中间结果、交换数据等,当存储过程或函数执行完,局部变量的生命周期也就结束了。

5. MySQL 中变量的区别

全局变量的有效范围是在整个服务器上,MySQL 就是通过维护这些全局变量来提供相应的数据库服务,其影响最大。用户不能定义全局变量而只能查看,用户可以通过 SET 命令修改全局变量,但必须具有相应的用户权限才能修改,当服务器重启时系统恢复默认值。MySQL 的变量类型及用法如表 5-6 所示。

表 5-6 MySQL 的变量类型及用法

操作类型	全局变量	会话变量	用户变量	局部变量(参数)
变量名称	global variables	session variables	user-defined variables	local variables
出现位置	命令行、函数、存储过程	命令行、函数、存储过程	命令行、函数、存储过程	函数、存储过程
定义方式	只能查看修改,不能定义	只能查看修改,不能定义	@用户变量	declare 局部变量 类型;
有效生命周期	服务器重启时恢复默认值	断开连接时,变量释放	断开连接时,变量释放	调用函数或存储过程的结束,变量无效
查看所有变量	show global variables;	show session variables;		
查看部分变量	show global variables like 'sql%';	show session variables like 'sql%';		
查看指定变量	select @global. sql_mode; select @@ sql_mode;	show session variables like 'sql%';	select @var;	select aa;

续表

操作类型	全局变量	会话变量	用户变量	局部变量(参数)

设置指定变量	set global sql_mode='';
	set@@global.sql_mode=''';
		set session sql_mode = '';
		set local sql_mode = '';
		set session.sql_mode='';
		set @@local.sql_mode = '';
		set @@sql_mode = '';
		set sql_mode = '';
			set @v=1;
			set @v=2;
			select 8 into @v;
				set a=10;
				set a=101;
				select 5 into a;

会话变量的修改通常只会影响当前连接，但是有个别一些变量是例外的，修改它们也需要较高的权限，如 binlog_format 和 sql_log_bin，设置这些变量的值将影响当前会话的二进制日志记录，也有可能对服务器复制和备份的完整性产生更广泛的影响。

对于用户变量和局部变量，可以根据实际应用的需求直接进行修改，它的定义和使用全都由用户自己掌握。

5.2　流程控制语句

流程控制语句用来根据条件控制语句的执行，通常在存储过程和函数中使用。在 MySQL 中，常用控制流程的语句有 IF 语句、CASE 语句、LOOP 语句、WHILE 语句、REPEAT 语句、LEAVE 语句和 ITERATE 语句。它们可以进行流程控制，每个流程中可能包含一个单独语句，或者是使用 BEGIN…END 构造的复合语句，并且可以嵌套。

5.2.1　IF 语句

IF 语句包含多个条件判断，根据条件表达式的值，确定执行不同的语句块，IF 语句的语法格式如下。

```
IF 条件表达式 1 THEN 语句块 1;
   [ELSEIF 条件表达式 2 THEN 语句块 2] …
   [ELSE 语句块 n]
END IF;
```

IF 语句用来进行条件判断，根据不同的条件执行不同的操作。该语句在执行时首先判断 IF 后的条件是否为真，如果为真，则执行 THEN 后的语句，如果为假则继续判断 ELSEIF 语句直到为真为止，当以上都不满足时则执行 ELSE 语句后的内容，IF 语句需要使用 END IF 来结束。

【例 5.19】 IF 语句的示例，输入如下命令。

```
CREATE PROCEDURE 'test_if'(val int)
BEGIN
  IF val IS NULL THEN SELECT 'val is NULL';
    ELSE SELECT 'val is not NULL';
  END IF;
END
```

该例判断 val 值是否为空，如果 val 值为空，输出字符串“val is NULL”；否则输出字符

串“val is not NULL”。

5.2.2 CASE 语句

CASE 语句用于实现比 IF 语句分支更为复杂的条件判断，CASE 语句为多分支语句结构，该语句首先从 WHEN 后的 VALUE 中查找与 CASE 后的 VALUE 相等的值，如果查找到则执行该分支的内容，否则执行 ELSE 后的内容。CASE 有两种语句格式，第一种格式如下。

```
CASE 表达式
    WHEN value1 THEN 语句块 1;
    [WHEN value2 THEN 语句块 2;]
    …
    [ELSE 语句块 n;]
END CASE;
```

CASE 的另一种语法表示形式为

```
CASE
    WHEN 表达式 1 THEN 语句块 1;
    [WHEN 表达式 2  THEN 语句块 2;]
    …
    [ELSE 语句块 n;]
END CASE;
```

其中，表达式参数表示条件判断语句；语句块参数表示不同条件的执行语句。在语句中，WHEN 语句将被逐个执行，直到某个表达式为真，则执行对应 THEN 关键字后面的语句块。如果没有条件匹配，ELSE 子句里的语句块将被执行。

注意：MySQL 中的 CASE 语句与 C 语言、Java 语言等高级程序设计语言不同，在高级程序设计语言中，每个 CASE 的分支需使用“break”跳出，而 MySQL 无须使用“break”语句。

【例 5.20】 使用 CASE 流程控制语句的第二种格式，判断 number 是否小于 0、大于 0 或者等于 0，输入如下命令。

```
CREATE PROCEDURE 'test_case'( number int)
BEGIN
  CASE
    WHEN number < 0 THEN SELECT 'number is less than 0';
    WHEN number > 0 THEN SELECT 'number is greater than 0';
   ELSE SELECT 'number is 0';
  END CASE;
END
```

5.2.3 WHILE 语句

WHILE 循环语句执行时首先判断指定的表达式是否为真，当条件表达式的值为 true 时，则执行循环体内的语句，直到条件表达式的值为 false。WHILE 语句的语法格式如下。

```
[循环标签:] WHILE 条件表达式 DO
    循环体;
END WHILE [循环标签];
```

【例 5.21】 使用 WHILE 循环语句,求解前 n 项的和。输入如下命令。

```
CREATE PROCEDURE 'test_while'(n int)
BEGIN
  DECLARE i INT DEFAULT 1;
  DECLARE s INT DEFAULT 0;
  WHILE i <= n DO
    SET s = s + i;
    SET i = i + 1;
  END WHILE;
  SELECT i,s;
END
```

5.2.4 LEAVE 和 ITERATE 跳转语句

MySQL 提供了 LEAVE 和 ITERATE 两个跳转语句。

LEAVE 语句:主要用于跳出循环控制。其语法格式如下。

```
LEAVE label;
```

其中,参数 label 表示循环的标志,可以用在循环语句内,或者以 BEGIN 和 END 包裹起来的程序体内,表示跳出循环或者跳出程序体的操作,类似于 C 语言中的 break 语句。

ITEATE 语句:ITERATE 只能用在 LOOP、REPEAT 和 WHILE 循环语句内,表示重新开始循环,即将执行转到循环开始处,类似于 C 语言中的 continue 语句。其语句格式如下。

```
ITERATE label;
```

该语句的格式与 LEAVE 相同,功能区别在于:LEAVE 语句是离开一个循环,而 ITERATE 语句是重新开始该循环。

5.2.5 LOOP 语句

LOOP 语句用来循环执行某些语句,与 IF 和 CASE 语句相比,LOOP 只是创建一个循环操作的过程,并不进行条件判断。LOOP 内的语句一直重复执行直到循环被退出。由于 LOOP 循环语句本身没有停止循环的语句,可以使用 LEAVE 或 ITERATE 子句改变程序流向。LOOP 语句的基本格式如下。

```
[循环标签:] LOOP
      循环体;
      IF 条件表达式 THEN LEAVE [循环标签];
      END IF;
END LOOP;
```

【例 5.22】 使用 LOOP 循环结合 LEAVE 和 ITERATE 语句,改变程序流程。输入如下命令。

```
CREATE PROCEDURE 'test_loop'()
BEGIN
   DECLARE num INT DEFAULT 0;
   my_loop:LOOP
       SET num = num + 1;
       IF num < 10
          THEN ITERATE my_loop;
       ELSEIF num > 15
          THEN LEAVE my_loop;
       END IF;
       SELECT num;
   END LOOP my_loop;
END
```

可以看出,num 的初值赋为 0,my_loop 循环执行 num 加 1 的操作。当 num 小于 10 时,使用 ITERATE 语句直接回到 my_loop 循环开始处,执行加 1 操作;当 num 大于或等于 10 时,输出 num 的值,继续 my_loop 循环,因此屏幕上输出 10～15;当 num 等于 16 时,使用 LEAVE 语句结束 my_loop 循环。

5.2.6 REPEAT 语句

REPEAT 语句创建一个有条件判断的循环过程,每次循环体执行完毕之后,将会对条件表达式进行判断,如果表达式为真,则循环结束;否则重复执行循环体中的语句。REPEAT 循环都以 END REPEAT 结束。REPEAT 语句的语法格式如下。

```
[循环标签:] REPEAT
     循环体;
UNTIL 条件表达式
END REPEAT [循环标签];
```

【例 5.23】 REPEAT 语句示例,输入如下命令。

```
CREATE PROCEDURE 'test_repeat'()
BEGIN
   DECLARE id INT DEFAULT 0;
   REPEAT
     SET id = id +1;
     SELECT id;
     UNTIL id>=16
   END REPEAT;
END
```

可以看出,循环执行 id 加 1 的操作。当 id 值小于 16 时,循环重复执行;当 id 值大于或等于 16 时,退出循环。

5.3 存储过程和存储函数

如果在实现用户的某些需求时，需要编写一组复杂的 SQL 语句来实现，并且用户经常调用这些需求时，可以将这组复杂的 SQL 语句集提前编写在数据库中，通过用户调用来执行。这些提前编写好的 SQL 语句集称为存储程序，MySQL 的存储程序有存储过程和函数。

1. 存储过程

存储过程(PROCEDURE)是为了完成特定功能的一组 SQL 语句集，存储在数据库中，经过第一次编译后，以后将不需要再次编译而直接可调用。用户通过指定存储过程的名字和参数(如果该存储过程有参数)来执行它。调用存储过程可以简化应用开发人员的很多工作，减少数据在数据库服务器和应用之间的传输，提高数据处理的效率。存储过程参数包括 in、out、inout 三种模式。

2. 存储函数

存储函数(FUNCTION)是一组 SQL 语句集，带函数名、参数。存储函数和存储过程的结构类似，但必须有一个 RETURN 子句来返回结果。使用函数可以减少很多工作量，降低数据在数据库服务器和应用之间的传输，提高数据处理的效率。函数的参数类型只有 in 一种模式，函数必须有返回值。

3. 存储过程和存储函数的区别

1) 形式上的不同

存储过程和存储函数统称为存储程序，两者的语法很相似，但却是不同的内容。存储函数限制比较多，如不能用临时表，只能用表变量；而存储过程的限制就相对比较少，所实现的功能更复杂一些。

2) 返回值不同

存储函数将向调用者返回一个且仅有一个结果值；而存储过程将返回一个或多个结果集，或者仅实现某种效果或动作而无须返回结果。

3) 调用方式不同

存储函数嵌入在 SQL 语句中使用，可以在 SELECT 中调用，就像内置函数一样，如 cos()、sin()；而存储过程使用 call 进行调用。

4) 参数的模式不同

存储函数的参数模式只有 IN 一种；而存储过程的参数有 IN、OUT、INOUT 三种模式。

5.3.1 创建存储过程和用户自定义函数

1. 创建存储过程

创建存储过程是通过 CREATE PROCEDURE 语句来创建的，其基本语法格式为

```
CREATE PROCEDURE sp_name ( [proc_parameter] ) [characteristics … ] routine_body
```

各选项说明：

- CREATE PROCEDURE：创建存储过程的命令。

- sp_name：存储过程名称。
- proc_parameter：指定存储过程的参数列表，列表中每项参数的形式为

```
[ IN | OUT | INOUT ]  param_name  type
```

其中，IN 模式参数只是从外部传入存储过程内部，可以是常数或变量，是值传递方式；OUT 模式的参数是在存储过程内部使用，输出给外部的，只能是变量，是引用传递方式。即使外部传入有值的参数，该参数的值也会被先清空才会进入内部，因此该模式的参数只能输出无法输入；INOUT 模式参数既可以输入也可以输出。param_name 表示参数名称。type 表示参数的类型，该类型可以是 MySQL 数据库中的任意类型。

- routine_body：代码的内容，使用 BEGIN…END 来表示 SQL 代码的开始和结束。
- characteristics：用于指定存储过程的特性，有以下取值。

LANGUAGE SQL：说明 routine_body 部分是由 SQL 语句组成的，当前系统支持的语言为 SQL，SQL 是 LANGUAGE 特性的唯一值。

[NOT] DETERMINISTIC：指明存储过程执行的结果是否确定。DETERMINISTIC 表示结果是确定的，每次执行存储过程时，相同的输入会得到相同的输出；NOT DETERMINISTIC 表示结果是不确定的，相同的输入可能得到不同的输出；如果没有指定任意一个值，系统默认为 NOT DETERMINISTIC。

{CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}：该短语指明存储过程能否通过 SQL 语句修改数据的特性，其中，CONTAINS SQL 表明子程序包含 SQL 语句；NO SQL 表明子程序不包含 SQL 语句；READS SQL DATA 说明子程序包含读数据的语句；MODIFIES SQL DATA 表明子程序包含写数据的语句。默认情况下，系统会指定为 CONTAINS SQL。

SQL SECURITY {DEFINER | INVOKER}：该短语指明谁有权限来执行，其中，DEFINER 表示只有定义者才能执行；INVOKER 表示拥有权限的调用者可以执行。默认情况下，系统指定为 DEFINER。

COMMENT 'string'：字符串 string 表示注释信息，用来描述存储过程或函数。

【例 5.24】 创建查看 course 表的存储过程，输入如下命令。

```
mysql > USE tb_test;
Database changed
mysql > DELIMITER //
mysql > CREATE PROCEDURE proc( )
      - > BEGIN
      - > SELECT * FROM  tb_courses;
      - > END //
Query OK, 0 rows affected (0.01 sec)
mysql > DELIMITER ;
```

可以看出，“DELIMITER //”语句的作用是将 MySQL 的结束符设置为“//”，因为 MySQL 默认的语句结束符号为分号“;”，为了避免与存储过程中 SQL 语句结束符相冲突，需要使用 DELIMITER 改变存储过程的结束符，并以“END //”结束存储过程。存储过程定义完毕之后再使用“DELIMITER ;”恢复默认结束符。DELIMITER 也可以指定其他符

号为结束符，但应该避免使用反斜杠“\”字符，因为反斜线是 MySQL 的转义字符。

【例 5.25】　创建名称为 CountProc 的存储过程，输入如下命令。

```
CREAT PROCEDURE CountProc(OUT param INT )
    BEGIN
      SELECT COUNT( * ) INTO param FROM tb_courses;
    END;
```

上述代码的作用是创建一个获取 tb_courses 表记录条数的存储过程，存储过程的名称是 CountProc，COUNT(*)计算后把结果放入传出参数 param 中，调用者可以从该参数中得到处理结果。

2. 存储函数

创建存储函数使用“CREATE FUNCTION”语句，语法格式为

```
CREATE FUNCTION func_name ([param_name   type ])
RETURNS type
[characteristic … ] routine_body
```

- CREATE　FUNCTION 为用来创建存储函数的关键字。
- func_name 表示存储函数的名称。
- func_parameter 短语为存储过程的参数列表，type 表示参数的类型，该类型可以是 MySQL 数据库中的任意类型。
- RETURNS type 短语表示函数返回数据的类型。
- characteristic 短语指定存储函数的特性，取值与存储过程相同，这里不再赘述。

使用 MySQL 创建、调用存储过程、函数以及触发器的时候会有错误符号为 1418 错误。[Errors] 1418-This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you * might * want to use the less safe log_bin_trust_function_creators variable)

通过查阅相关资料，可知当 binlog 启用时，创建的函数必须声明类型。因为 binlog 需要知道这个函数创建的类型，否则同步数据时将会产生不一致现象。所以如果 MySQL 开启了 binlog，用户就必须指定如下函数类型。

- DETERMINISTIC：表示该函数运行结果是不确定的。
- NO SQL：表示该函数没有 SQL 语句，当然也不会修改数据。
- READS SQL DATA：表示该函数只是读取数据，当然也不会修改数据。
- MODIFIES SQL DATA：表示该函数将要修改数据。
- CONTAINS SQL：表示该函数包含 SQL 语句。

为了解决这个问题，MySQL 强制要求：在主服务器上，除非子程序被声明为确定性的或者不更改数据，否则创建或者替换子程序将被服务器拒绝。这意味着当创建一个子程序的时候，必须要么声明它是确定性的，要么声明是否会改变数据。

声明方式有以下三种。

1）声明是否是确定性的

DETERMINISTIC 和 NOT DETERMINISTIC 指出一个子程序是否对给定的输入总

是产生同样的结果。如果没有给定该声明选项,则系统默认是 NOT DETERMINISTIC,因此必须明确使用 DETERMINISTIC 来声明一个存储函数是确定性的。

2) 声明是否会改变数据

CONTAINS SQL, NO SQL, READS SQL DATA, MODIFIES SQL 用来指出子程序是读数据还是写数据的。无论是 NO SQL 还是 READS SQL DATA 都指出,子程序没有改变数据,但是必须明确地指定其中一个,如果没有任何指定,默认的指定是 CONTAINS SQL。

3) 是否信任子程序的创建者

禁止创建、修改子程序时对 SUPER 权限的要求,设置 log_bin_trust_routine_creators 全局系统变量为 1。具体设置方法有如下三种。

- 在客户机上执行 SET GLOBAL log_bin_trust_function_creators=1。
- MySQL 启动时,加上 log-bin-trust-function-creators 选项,参数设置为 1。
- 在 MySQL 配置文件 my.ini 的[mysqld]段加上 log-bin-trust-function-creators=1。

【例 5.26】 创建一个存储函数 get_num,该函数插入一条值为(6,'Newcourse',10)的新记录后,返回 tb_courses 表的记录数,输入命令如下。

```
1.    CREATE FUNCTION 'get_num'()
2.      RETURNS int
3.    --   READS SQL DATA            # 如果没有此短语,将产生 1418 错误
4.    --   DETERMINISTIC             # 如果没有此短语,将产生 1418 错误
5.    BEGIN
6.        declare a int;
7.          INSERT INTO tb_courses (course_id,course_name,course_grade)
8.          VALUES(6,'Newcourse',10);
9.        select count( * ) into a from tb_courses;
10.   RETURN a;
11.   END
```

可以看出,如果注释掉第 3、4 行代码,将会产生 1418 错误,必须从这两行代码选任一行,才能消除这个错误,这就是根据上述的方式 1)和方式 2)做的对应修改。

也可以直接使用方式 3),先做设置,再按如下代码创建函数,就能消除错误 1418,输入如下命令。

```
SET GLOBAL log_bin_trust_function_creators = 1;        # 先执行此行代码,信任子程序创建者
CREATE FUNCTION 'test_fun3'()
  RETURNS int
BEGIN
    declare a int;
    INSERT INTO tb_courses (course_id,course_name,course_grade)
     VALUES(6,'Newcourse',10);
    select count( * ) into a from tb_courses;
    RETURN a;
END
```

3. 存储过程或函数中使用变量

变量可以在子程序中声明并使用,这些变量的作用范围是在 BEGIN…END 程序中。

1）定义变量

在存储过程中通过 DECLARE 语句定义变量，定义变量的格式如下。

```
DECLARE var_name[,varname]  date_type  [DEFAULT value];
```

- var_name 为局部变量的名称。
- DEFAULT value 子句给变量提供一个默认值，默认值可以是一个常数，还可以为一个表达式。如果没有 DEFAULT 子句，初始值为 NULL。

2）为变量赋值

定义后的变量可以通过 SELECT…INTO var_list 进行赋值，语法如下。

```
SELECT col_name [, … ] INTO  var_name [, … ] table_expr;
```

- SELECT 指令把选定的列直接存储到对应位置指定的变量中。
- col_name 表示字段名称。
- var_name 表示定义的变量名称。
- table_expr 表示查询条件表达式包括表名称和 WHERE 子句。

【例 5.27】 MySQL 存储过程中使用变量，输入命令如下。

```
mysql> use tb_test;                    #打开 tb_test t 数据库，在 tb_test 数据库中创建存储过程
Database changed
mysql> delimiter //                                    #声明使用//作为命令结束标记
mysql> create procedure sp1(v_sid int)                 #创建存储过程
    -> begin
    -> declare xname varchar(10) default 'dayi123';  #局部变量 xname
    -> declare xsex int;                               #局部变量 xsex
    -> select xname, xsex;
    -> end //
Query OK, 0 rows affected (0.01 sec)
```

MySQL 中还可以使用 SET 语句对多个局部变量进行赋值，具体格式如下。

```
SET var_name1 = expr1 [, var_name2 = expr2] … ;
```

- 这些参数变量可以是子程序内部声明的变量、系统变量或者用户变量。

【例 5.28】 声明三个会话变量，分别为 var1、var2 和 var3，数据类型为 INT，使用 SET 为变量赋值，输入如下命令。

```
mysql> SET @var1 = 10, @var2 = 20;
Query OK, 0 rows affected (0.00 sec)
mysql> SET @var3 = @var1 + @var2;
Query OK, 0 rows affected (0.01 sec)
mysql> select @var1,@var2,@var3;
+-------+-------+-------+
| @var1 | @var2 | @var3 |
+-------+-------+-------+
|    10 |    20 |    30 |
+-------+-------+-------+
1 row in set (0.00 sec)1 row in set (0.00 sec)
```

5.3.2 调用存储过程和存储函数

存储过程和函数有多种调用方法。存储过程使用 CALL 语句调用,而存储函数的调用与 MySQL 中预定义的函数的调用方式相同,可以直接使用 SELECT 语句进行调用。

1. 调用存储过程

存储过程是通过 CALL 语句进行调用的,语法格式如下。

```
CALL [dbname.]sp_name ([parameter[,…]]);
```

- CALL 语句调用一个已经存在于当前数据库中的存储过程。
- sp_name 为存储过程名称。
- parameter 为存储过程的参数,如果调用其他数据库的存储过程,则必须在存储过程前加上[dbname.]的前缀。

【例 5.29】 调用当前数据库中名为 sp1 的存储过程,输入如下命令。

```
mysql > call sp1(1);
+----------+------+
| xname    | xsex |
+----------+------+
| dayi123  | NULL |
+----------+------+
1 row in set (0.01 sec)
Query OK, 0 rows affected (0.01 sec)
```

2. 调用存储函数

在 MySQL 中,用户自定义的存储函数与 MySQL 内部函数是同一性质的。区别在于,存储函数是用户自己定义并存储于当前数据库中,而内部函数是 MySQL 的开发者定义的,存储在服务器上。函数是用 SELECT 语句调用的,语法格式如下。

```
SELECT [dbname.]fn_name ([parameter[,…]]) ;
```

- [dbname.]fn_name( ):为调用的函数名,若是调用其他数据库中的函数,则在函数名前添加[dbname.]前缀。
- [parameter[,…]]:为调用的实参列表,实参个数与类型必须与定义一致,各实参之间用逗号分隔。

【例 5.30】 创建并调用当前数据库中的 get_area 函数,并传入两个参数,输入如下命令。

```
mysql > USE 'test_db';
Database changed
mysql > DELIMITER $$
mysql > USE 'test_db'$$
Database changed
mysql > CREATE FUNCTION 'get_area'(a int,b int)
    -> RETURNS INTEGER
    -> DETERMINISTIC
    -> BEGIN
```

```
    -> RETURN a * b;
    -> END $$
Query OK, 0 rows affected (0.01 sec)
mysql> DELIMITER ;
mysql> select get_area(3,4);
+---------------------+
| get_area(3,4)       |
+---------------------+
|                  12 |
+---------------------+
1 row in set (0.00 sec)
```

5.3.3 查看存储过程和存储函数

MySQL 存储了存储过程和函数的状态信息，用户可以使用 SHOW STATUS 语句或 SHOW CREATE 语句来查看，也可直接从系统的 information_schema 数据库中查询。

1. 使用 SHOW STATUS 语句查看存储过程和函数的状态

SHOW STATUS 语句可以查看存储过程和函数的状态，其基本语法结构如下。

```
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

- PROCEDURE 参数标识查询存储过程，FUNCTION 表示查询存储函数。
- LIKE 'pattern'参数可以过滤出与通配符 'pattern' 相匹配的存储过程或存储函数。
- 该语句返回子程序的特征，如数据库、名字、类型、创建者及创建和修改日期。如果没有指定样式，将列出所有存储过程和函数的信息。

2. 使用 SHOW CREATE 语句查看存储过程和函数的定义

除了 SHOW STATUS 之外，MySQL 还可以使用 SHOW CREATE 语句查看存储过程和函数的状态。其语法如下。

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

- PROCEDURE 和 FUNCTION 分别表示查看存储过程和函数。
- sp_name 参数表示匹配存储过程或函数的名称。

该语句类似于 SHOW CREATE TABLE，返回一个已命名子程序的创建 SQL 代码字符串。

3. 从 information_schema. Routines 表中查看存储过程和函数的信息

MySQL 中存储过程和函数的信息存储在 information_schema 数据库下的 Routines 表中。因此，可以通过查询该表的记录来查询存储过程和函数的信息。其基本语法形式如下。

```
SELECT * FROM information_schema.Routines WHERE ROUTINE_NAME = 'sp_name';
```

ROUTINE_NAME 字段中存储的是存储过程和函数的名称，可以是字符串常量或变量。

5.3.4 修改存储过程和存储函数

可以使用 ALTER 语句修改存储过程或函数的某些特征。其语法格式如下。

```
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic … ]
```

其中,sp_name 参数表示存储过程或函数的名称;characteristic 参数指定存储函数的特性,可能的取值如下。

- CONTAINS SQL:表示子程序包含 SQL 语句,但不包含读或写数据的语句。
- NO SQL:表示子程序中不包含 SQL 语句。
- READS SQL DATA:表示子程序中包含读数据的语句。
- MODIFIES SQL DATA:表示子程序中包含写数据的语句。
- SQL SECURITY { DEFINER | INVOKER }:指明是定义者或调用者有权限来执行。
- DEFINER:表示只有定义者自己才能够执行。
- INVOKER:表示调用者可以执行。
- COMMENT:'注释内容'表示注释信息。

【例 5.31】 修改存储过程 CountProc 的定义。将读写权限改为 MODIFIES SQL DATA,并指明调用者可以执行,输入如下命令。

```
ALTER PROCEDURE CountProc MODIFIES SQL DATA SQL SECURITY INVOKER;
```

5.3.5 删除存储过程和存储函数

删除存储过程、存储函数主要使用 DROP 语句,其语法结构如下。

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

其中,sp_name 为要删除的存储过程或函数的名称。IF EXISTS 子句是一个 MySQL 的扩展,表示如果当前数据库已经存在该存储过程或函数,则进行删除。

【例 5.32】 删除存储过程 CountProc,输入如下命令。

```
mysql > DROP PROCEDURE CountProc;
Query OK, 0 rows affected (0.00 sec)
```

5.3.6 综合实例——存储过程和存储函数的使用

实例 1:分别使用存储过程和存储函数实现根据输入的年份、月份和当前系统的年份比较,不满 1 年按 1 年计算,多出 1 年 11 个月也按 1 年计算。

创建存储过程,输入如下命令。

```
DELIMITER $$
USE 'tb_test' $$
DROP PROCEDURE IF EXISTS 'sp_calc_year' $$
CREATE PROCEDURE 'sp_calc_year'(IN Y INT,IN M INT,OUT Diff INT)
BEGIN
    # declare current date default now();
    DECLARE c_y INT DEFAULT 0;
    DECLARE c_m INT DEFAULT 0;
```

```
    SET c_y = YEAR(NOW());
    SET c_m = MONTH(NOW());
    IF c_y > Y THEN
         IF c_m < M THEN
           SET c_y = c_y - 1;
         END IF;
         SET Diff = c_y - Y;
         IF Diff = 0 THEN
           SET Diff = 1;
         END IF;
    ELSE
         SET Diff = 1;
    END IF;
    END $$
DELIMITER ;
```

调用存储过程，输入如下命令。

```
mysql > SET @p_inY = 2011;
Query OK, 0 rows affected (0.00 sec)
mysql > SET @p_inM = 10;
Query OK, 0 rows affected (0.00 sec)
mysql > SET @p_out = 0;
Query OK, 0 rows affected (0.00 sec)
mysql > CALL sp_calc_year(@p_inY,@p_inM,@p_out);
Query OK, 0 rows affected (0.00 sec)
mysql > SELECT @p_out;
+---------+
| @p_out  |
+---------+
|     10  |
+---------+
1 row in set (0.00 sec)
```

创建函数的代码如下。

```
DELIMITER $$
CREATE FUNCTION 'tb_test','sp_calc_ym'(Y INT, M INT)
   RETURNS INT
 DETERMINISTIC
  BEGIN
     DECLARE c_y INT DEFAULT 0;
     DECLARE c_m INT DEFAULT 0;
     DECLARE Diff INT DEFAULT 1;
     SET c_y = YEAR(NOW());
     SET c_m = MONTH(NOW());
     IF c_y > Y THEN
          IF c_m < M THEN
            SET c_y = c_y - 1;
          END IF;
          SET Diff = c_y - Y;
          IF Diff = 0 THEN
            SET Diff = 1;
```

```
            END IF;
        ELSE
            SET Diff = 1;
        END IF;
        RETURN Diff;
    END $$
DELIMITER ;
```

调用函数的 SQL 代码及运行结果如下。

```
mysql > SELECT 'sp_calc_ym'(2011,10);
+-------------------------+
| 'sp_calc_ym'(2011,10)   |
+-------------------------+
|                      10 |
+-------------------------+
1 row in set (0.00 sec)
```

实例 2：求 sum 以内所有奇数之和。

(1) 创建存储过程。

```
DROP procedure IF EXISTS 'example_iterate';
DELIMITER $$
USE 'tb_test' $$
CREATE DEFINER = 'root'@'localhost' PROCEDURE 'example_iterate'(inout sum INT)
BEGIN
    DECLARE i INT DEFAULT 0;
    DECLARE s INT DEFAULT 0;
        loop_label: LOOP
        SET i = i + 1;
        IF i > sum THEN
            LEAVE loop_label;                    #退出整个循环
        END IF;
        IF (i mod 2) THEN
            SET s = s + i;
        ELSE
            ITERATE loop_label;                  #结束本次循环,转到循环开始处
        END IF;
    END LOOP;
    SET sum = s;
END $$
DELIMITER ;
```

(2) 调用存储过程。

```
set @sum = 10;                                   #给参数赋初值
call example_iterate(@sum);                      #调用存储过程
```

(3) 执行效果如下。

```
mysql > select @sum;
+-------+
| @sum  |
```

```
+-------+
|   25  |
+-------+
1 row in set (0.00 sec)
```

5.4　MySQL 触发器

MySQL 从 5.0.2 版本开始支持触发器(TRIGGER),触发器是数据库对象之一,该对象类似于函数,需要声明才能使用。但是触发器的执行不是由程序调用,也不是由手工启动,而是由事件来触发、激活从而实现执行的。

那么为什么要使用触发器呢?下面以教学管理中的一个实例进行说明。

1. 统计更新

在学生表中拥有学生姓名字段、学生总数字段,每当添加一条学生信息时,学生的总数就必须同时更改。

2. 数值校验

在学生表中还会有学生姓名的缩写、学生住址等字段,添加学生信息时,往往需要检查电话、邮箱等格式是否正确。

3. 参照完整性

当学生选修了某门课程后,若该门课程被取消了,则该学生选修该门课程的记录也应随之取消;那么应该何时删除此条选修记录,又由谁来发出删除此条记录的指令呢?

上面的例子具有这样的特点,即在表内数据发生改变时,需要自动处理其他相关联的一系列活动,这就是数据库的完整性。可以使用第 4 章所讲的“外键”约束来实现,而本节主要介绍的是使用触发器来实现数据库的完整性。

可以把要处理的这一系列活动封装为触发器,当数据库执行这些语句的时候就会激发触发器执行相应的操作。触发器是表内的独立命名对象,当表上出现特定事件时,将激活该对象,可以在创建表时定义,也可以在表创建后再独立定义。

本节的示例数据库采用第 4 章习题中的 Teaching 数据库,为了使用触发器实现数据完整性,必须将所有表外键去除,可重新创建 5 个表,输入如下命令。

```
Create database Teaching;                          # 创建数据库 Teaching
Use Teaching;                                      # 打开数据库 Teaching
CREATE TABLE 'class'                               # 创建 class 表,主键为 id
   ( 'id' smallint(6) NOT NULL,
     'name' varchar(50) DEFAULT NULL,
     PRIMARY KEY ('id')
) ;
CREATE TABLE 'student'                             # 创建 student 表,主键为 id
 ( 'id' smallint(6) NOT NULL,
  'name' varchar(20) DEFAULT NULL,
  'gender' char(2) DEFAULT NULL,
  'class_id' smallint(6) DEFAULT NULL,
  PRIMARY KEY ('id')
) ;
```

```
CREATE TABLE 'teacher'                                  #创建 teacher 表,主键为 id
  ( 'id' smallint(6) NOT NULL,
    'name' varchar(100) DEFAULT NULL,
  PRIMARY KEY ('id')
) ;
CREATE TABLE 'course'                                   #创建 course 表,主键为 id
( 'id' smallint(6) NOT NULL,
    'name' varchar(100) DEFAULT NULL,
    'teacher_id' smallint(6) DEFAULT NULL,
    PRIMARY KEY ('id')
) ;
CREATE TABLE 'score'                                    #创建 score 表,主键为 id
  ( 'id' int(11) DEFAULT NULL,
    'student_id' smallint(6) NOT NULL,
    'course_id' smallint(6) NOT NULL,
    'mark' tinyint(4) DEFAULT NULL,
    PRIMARY KEY ('id')
);
```

为 Teaching 数据库添加示例数据,输入如下指令。

```
Use Teaching;                #打开数据库 Teaching,添加如下示例数据
INSERT INTO 'class' ('id', 'name') VALUES ('1', '软件工程 1 班');
INSERT INTO 'class' ('id', 'name') VALUES ('2', '计算机科学技术 1 班');
INSERT INTO 'class' ('id', 'name') VALUES ('3', '网络工程 1 班');

INSERT INTO 'teacher' ('id', 'name') VALUES ('1', '老虎');
INSERT INTO 'teacher' ('id', 'name') VALUES ('2', '小马');
INSERT INTO 'teacher' ('id', 'name') VALUES ('3', '大牛');

INSERT INTO 'course' ('id', 'name', 'teacher_id') VALUES ('1', '数据结构', '1');
INSERT INTO 'course' ('id', 'name', 'teacher_id') VALUES ('2', 'Java 语言', '2');
INSERT INTO 'course' ('id', 'name', 'teacher_id') VALUES ('3', '数据库原理', '3');
INSERT INTO 'course' ('id', 'name', 'teacher_id') VALUES ('4', 'C 语言', '1');
INSERT INTO 'student' ('id', 'name', 'gender', 'class_id') VALUES ('1', '牡丹', '女', '1');
INSERT INTO 'student' ('id', 'name', 'gender', 'class_id') VALUES ('2', '柳树', '男', '2');
INSERT INTO 'student' ('id', 'name', 'gender', 'class_id') VALUES ('3', '玫瑰', '女', '3');
INSERT INTO 'student' ('id', 'name', 'gender', 'class_id') VALUES ('4', '月季', '女', '1');
INSERT INTO 'student' ('id', 'name', 'gender', 'class_id') VALUES ('5', '小草', '男', '2');
INSERT INTO 'student' ('id', 'name', 'gender', 'class_id') VALUES ('6', '风清', '男', '3');
INSERT INTO 'student' ('id', 'name', 'gender', 'class_id') VALUES ('7', '月明', '女', '1');

INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('1', '1', '2', '79');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('2', '2', '1', '58');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('3', '2', '3', '66');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('4', '2', '4', '80');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('5', '3', '1', '63');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('6', '3', '4', '95');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('7', '4', '2', '88');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('8', '4', '3', '62');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('9', '5', '2', '59');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('10', '5', '4', '100');
INSERT INTO 'score' ('id', 'student_id', 'course_id', 'mark') VALUES ('11', '1', '1', '55');
```

```
INSERT INTO 'score'('id', 'student_id', 'course_id', 'mark') VALUES ('12', '3', '2', '81');
INSERT INTO 'score'('id', 'student_id', 'course_id', 'mark') VALUES ('13', '4', '4', '50');
INSERT INTO 'score'('id', 'student_id', 'course_id', 'mark') VALUES ('14', '5', '3', '77');
INSERT INTO 'score'('id', 'student_id', 'course_id', 'mark') VALUES ('15', '1', '4', '58');
INSERT INTO 'score'('id', 'student_id', 'course_id', 'mark') VALUES ('16', '1', '3', '91');
INSERT INTO 'score'('id', 'student_id', 'course_id', 'mark') VALUES ('17', '6', '2', '75');
INSERT INTO 'score'('id', 'student_id', 'course_id', 'mark') VALUES ('18', '4', '1', '80');
INSERT INTO 'score'('id', 'student_id', 'course_id', 'mark') VALUES ('19', '2', '2', '75');
```

5.4.1 创建触发器

触发器是针对表的INSERT、DELETE和UPDATE这三种操作进行的，触发器可以建立在表上，也可以建立在视图上。触发器的执行不需要使用CALL语句来调用，也不需要手动启动，只要当预定义的INSERT、DELETE和UPDATE事件发生的时候，就会被MySQL自动调用。

建立在表上的触发器会在表数据发生改变时触发，而建立在视图上的触发器会在视图数据改变时触发。**注意**：视图触发器仅在该视图进行SQL语句执行时才会触发，当其所依赖的基本表的数据发生改变时会引起视图数据的变化，但不会触发该视图触发器。

1. 创建简单触发器

当触发器程序只有一条执行语句时，可以创建如下语法格式的简单触发器。

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON tb_name FOR EACH ROW trigger_stmt;
```

- trigger_name标识触发器名称，可以由用户自行指定。
- trigger_time为触发时机，可以指定为before或after；trigger_event为触发事件，包括INSERT、UPDATE和DELETE。
- tb_name为建立触发器的表名，即在哪张表上建立触发器。
- trigger_stmt是触发器执行语句。

2. 创建包含多条执行语句的触发器

创建多条执行语句的触发器的语法如下。

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON tb_name FOR EACH ROW
BEGIN
  trigger_statement;
END;
```

当触发器程序需要执行一条以上的语句时，可以使用BEGIN和END作为语句体的开始和结束，中间包含多条语句。

【例5.33】 为teacher表创建一个名为teacher_AFTER_UPDATE的触发器，当更改teacher中某位教师的id时，将course表中的对应teacher_id全部更新。SQL代码和运行结果如下。

```
mysql > DROP TRIGGER IF EXISTS teacher_AFTER_UPDATE;
mysql > DELIMITER $$
mysql > USE 'teaching' $$                    # 打开 teaching 数据库
Database changed
mysql > CREATE DEFINER = CURRENT_USER TRIGGER 'teacher_AFTER_UPDATE'
  AFTER UPDATE ON 'teacher' FOR EACH ROW
      -> BEGIN
      -> update course set teacher_id = new.id where teacher_id = old.id;    # 更新
      -> END $$
Query OK, 0 rows affected (0.01 sec)
mysql > DELIMITER ;
```

验证触发器 teacher_AFTER_UPDATE 的功能，SQL 代码和执行结果如下。

```
mysql > SELECT * FROM teaching.teacher;
+----+--------+
| id | name   |
+----+--------+
|  1 | 老虎   |
|  2 | 小马   |
|  3 | 大牛   |
+----+--------+
3 rows in set (0.01 sec)
mysql > UPDATE 'teaching'.'teacher' SET 'id' = '4' WHERE ('id' = '3');
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql > SELECT * FROM teaching.teacher;
+----+--------+
| id | name   |
+----+--------+
|  1 | 老虎   |
|  2 | 小马   |
|  4 | 大牛   |
+----+--------+
3 rows in set (0.00 sec)
mysql > SELECT * FROM teaching.course;
+----+------------------+--------------+
| id | name             | teacher_id   |
+----+------------------+--------------+
|  1 | 数据结构         |            1 |
|  2 | Java 语言        |            2 |
|  3 | 数据库原理       |            4 |
|  4 | C 语言           |            1 |
+----+------------------+--------------+
4 rows in set (0.00 sec)
```

从运行结果可以看出：

(1) 在本例中，UPDATE 是触发事件，AFTER 是触发程序的动作时间。当把 teacher 表中的 3 号老师更新为 4 号后，激活触发器更新了 course 表的相应记录。可以使用 SELECT 语句查看 teacher 和 course 表中的数据，发现所有原值为 3 的 teacher_id 的记录确实已被更新。

（2）在本例中，new 和 old 是和 teacher 结构相同的两个表，其中，new 存放着更新后的新记录，而 old 中存放着更新前的旧记录。当在 teacher 表更新 id 时，course 中原 teacher_id 的记录将随之更新为新的教师号，这样就实现了同步更新，维护了数据的完整性。

5.4.2　查看触发器

查看触发器是指查看数据库中已存在的触发器的定义、状态和语法信息等。可以通过命令查看已经创建的触发器。有两种查看触发器的方法，分别是 SHOW TRIGGERS 和在 triggers 表中查看触发器信息。

1. SHOW TRIGGERS

通过 SHOW TRIGGERS 查看触发器的语句如下。

```
SHOW TRIGGERS;
```

【例 5.34】　在 teaching 数据库中查看触发器情况，输入如下命令。

```
USE teaching;
SHOW TRIGGERS;
```

如果在 SHOW TRIGGERS 命令的后面添加上‘\G’，显示信息如下。

```
mysql > SHOW TRIGGERS \G;
*************************** 1. row ***************************
             Trigger: cno_update
               Event: UPDATE
               Table: course
           Statement: begin
update score set course_id = new.id where course_id = old.id;
end
              Timing: AFTER
             Created: 2020 - 04 - 24 00:14:13.50
            sql_mode: STRICT_TRANS_TABLES, NO_ENGINE_SUBSTITUTION
             Definer: root@localhost
character_set_client: gbk
collation_connection: gbk_chinese_ci
   Database Collation: utf8mb4_0900_ai_ci
1 row in set (0.00 sec)
```

其中，各主要参数含义如下。

Trigger 表示触发器的名称，在这里触发器的名称为 cno_update。

Event 表示激活触发器的事件，这里的触发事件为更新操作 UPDATE。

Table 表示激活触发器的操作对象表，这里为 course 表。

Timing 表示触发器触发的时间，为更新操作之后 AFTER。

Statement 表示触发器执行的操作。

还有一些其他信息，如 SQL 的模式触发器的定义账户和字符集等，这里不再一一介绍了。

2. 在 TRIGGERS 表中查看触发器信息

通过执行 SHOW TRIGGERS 命令查看触发器，由于没有指定查询的触发器，所以每

次都返回所有触发器的信息。在触发器较少的情况下,使用该语句会很方便,当查看多个触发器时会发生格式混乱的现象。

MySQL 中所有触发器的定义都存在 INFORMATION_SCHEMA 数据库的 TRIGGERS 表中,如果要查看特定触发器的信息,可以直接从 INFORMATION_SCHEMA 数据库的 TRIGGERS 表中,通过查询命令 SELECT 来查看。

【例 5.35】 查看 teaching 数据库中 teacher 表的触发器情况,输入如下命令。

```
  mysql > SELECT * FROM INFORMATION_SCHEMA.TRIGGERS WHERE TRIGGER_NAME = 'teacher_AFTER_
UPDATE'\G;
  *************************** 1. row ***************************
             TRIGGER_CATALOG: def
              TRIGGER_SCHEMA: teaching
                TRIGGER_NAME: teacher_AFTER_UPDATE
          EVENT_MANIPULATION: UPDATE
        EVENT_OBJECT_CATALOG: def
         EVENT_OBJECT_SCHEMA: teaching
          EVENT_OBJECT_TABLE: teacher
              ACTION_ORDER: 1
          ACTION_CONDITION: NULL
          ACTION_STATEMENT: BEGIN
 update course set teacher_id = new.id where teacher_id = old.id;
END
        ACTION_ORIENTATION: ROW
            ACTION_TIMING: AFTER
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
             CREATED: 2022-09-19 16:54:06.38
             SQL_MODE: STRICT_TRANS_TABLES, NO_ENGINE_SUBSTITUTION
             DEFINER: root@%
    CHARACTER_SET_CLIENT: utf8mb4
    COLLATION_CONNECTION: utf8mb4_0900_ai_ci
      DATABASE_COLLATION: utf8mb4_0900_ai_ci
1 row in set (0.00 sec)
mysql >
```

从结果可以看出:

- TRIGGER_NAME 表示触发器的名称,在这里名称为 teacher_AFTER_UPDATE。
- EVENT_MANIPULATION 表示激活触发器的事件,这里为更新事件 UPDATE。
- EVENT_OBJECT_TABLE 表示激活触发器的操作对象表,这里为 teacher 表。
- ACTION_TIMING 表示触发器触发的时间,此处为 AFTER。
- ACTION_STATEMENT 表示触发器执行的操作,其他类似于 SQL 的模式触发器的定义账户和字符集等的信息,此处就不详细解释了。

5.4.3 触发器的使用

当对表执行 INSERT、DELETE 或 UPDATE 语句时,将激活触发器程序。可以将触发器设置为在执行语句之前或之后激活。例如,可以在表中删除每一行之前,或在更新每一行

之后激活触发器。

【例 5.36】　创建在 student 表中插入记录之后，同步更新 student_bak 表的触发器。

```
mysql> use teaching;                                          # 切换到 teaching 数据库
Database changed
mysql> CREATE TABLE student_bak AS select * from student;     # 创建备份表 student_bak
Query OK, 7 rows affected (0.03 sec)
Records: 7  Duplicates: 0  Warnings: 0                        # 当前 student 有 7 条记录
```

为 student 表创建了名称为 trig_insert 的触发器，当 student 表中插入记录之后被触发激活，执行的操作是在表 student_bak 中插入同样内容的一条记录，实现了同步备份功能，输入如下命令。

```
mysql> CREATE TRIGGER trig_insert AFTER INSERT ON student FOR EACH ROW
  INSERT INTO student_bak VALUES (new.id, new.name,new.gender,new.class_id);
Query OK, 0 rows affected (0.01 sec)
mysql> INSERT INTO 'student'('id', 'name', 'gender', 'class_id') VALUES ('8', 'ccc', '男', '1');
Query OK, 1 row affected (0.01 sec)
mysql> select * from student_bak;
+----+--------+--------+----------+
| id | name   | gender | class_id |
+----+--------+--------+----------+
|  1 | 牡丹   | 女     |        1 |
|  2 | 柳树   | 男     |        2 |
|  3 | 玫瑰   | 女     |        3 |
|  4 | 月季   | 女     |        1 |
|  5 | 小草   | 男     |        2 |
|  6 | 风清   | 男     |        3 |
|  7 | 月明   | 女     |        1 |
|  8 | ccc    | 男     |        1 |
+----+--------+--------+----------+
8 rows in set (0.00 sec)mysql>
```

从测试结果可看出，向 student 表中插入值为('8', 'ccc', '男', '1')的新记录之后触发激活，执行的操作是在表 student_bak 中插入同样内容的一条记录，实现了同步备份功能。

此外，触发器的使用限制有以下几点。

(1) 触发器只能创建在永久表上，不能对临时表创建触发器。

(2) 由于触发器中不能包含带有返回结果集的 SQL 语句，所以不能使用 CALL 语句调用具有返回值或使用了动态 SQL 的存储过程。

(3) 触发器中不能使用开启或结束事务的语句段，例如，开始事务(START TRANSACTION)、提交事务(COMMIT)或是回滚事务(ROLLBACK)，但是回滚到一个保存点(SAVEPOINT)是允许的，因为回滚到保存点不会结束事务。

(4) 如果实现同一功能的数据库完整性约束，外键和触发器不能同时使用，两者只能择一而用，否则将会产生错误代码为 1415 的错误。

(5) 由于触发器中不允许返回值，因此触发器中不能有返回语句，如果要立即停止一个触发器，应该使用 LEAVE 语句。

5.4.4 删除触发器

使用 DROP TRIGGER 语句可以删除 MySQL 中已经定义的触发器，删除触发器指令语法格式如下。

```
DROP TRIGGER [IF EXISTS] [database_name] trigger_name
```

- trigger_name 表示要删除的触发器名称。
- database_name 指定触发器所在的数据库的名称，是可选的。若没有指定，则为当前默认的数据库。
- IF EXISTS 是可选项，表示如果该触发器存在，则删除。

注意：删除一个表的同时，也会自动删除该表上的触发器。另外，触发器不能更新或覆盖，当修改一个触发器时，必须先删除它，再重新创建。此外，执行 DROP TRIGGER 语句需要 SUPER 权限。

【例 5.37】 删除 teacher_AFTER_UPDATE 触发器，输入如下命令。

```
mysql > DROP TRIGGER teacher_AFTER_UPDATE;
Query OK, 0 rows affected (0.01 sec)
```

5.4.5 综合实例——触发器的使用

实例 1：创建简单触发器，在向学生表插入数据时，学生数增加，删除学生时，学生数减少。

(1) 首先创建学生表 student_info 和学生数目统计表 student_count，student_info 表中有两个字段，分别为 stu_no 字段(定义为 INT 类型)和 stu_name 字段(定义为 VARCHAR 类型)；student_count 表中有一个 amount 字段(定义为 INT 类型)。输入如下命令。

```
CREATE TABLE student_info (
   stu_no INT(11) NOT NULL AUTO_INCREMENT,
   stu_name VARCHAR(255) DEFAULT NULL,
   PRIMARY KEY (stu_no));
CREATE TABLE student_count (
   student_count INT(11) DEFAULT 0);
INSERT INTO student_count VALUES(0);
```

(2) 在向学生表插入数据时，学生数增加，删除学生时，学生数减少。

```
CREATE TRIGGER trigger_student_count_insert AFTER INSERT
ON student_info FOR EACH ROW
UPDATE student_count SET student_count = student_count + 1;
CREATE TRIGGER trigger_student_count_delete AFTER DELETE
ON student_info FOR EACH ROW
UPDATE student_count SET student_count = student_count - 1;
```

(3) 插入、删除数据，查看触发器是否正常工作。

```
mysql > INSERT INTO student_info VALUES(NULL,'张明'),(NULL,'李明'),(NULL,'王明');
Query OK, 3 rows affected (0.02 sec)
```

```
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM student_info;
+--------+----------+
| stu_no | stu_name |
+--------+----------+
|      1 | 张明     |
|      2 | 李明     |
|      3 | 王明     |
+--------+----------+
3 rows in set (0.00 sec)
mysql> SELECT * FROM student_count;
+---------------+
| student_count |
+---------------+
|             3 |
+---------------+
1 row in set (0.00 sec)
mysql> DELETE FROM student_info WHERE stu_name IN('张明','李明');
Query OK, 2 rows affected (0.00 sec)
mysql> SELECT * FROM student_info;
+--------+----------+
| stu_no | stu_name |
+--------+----------+
|      3 | 王明     |
+--------+----------+
1 row in set (0.00 sec)
mysql> SELECT * FROM student_count;
+---------------+
| student_count |
+---------------+
|             1 |
+---------------+
1 row in set (0.00 sec)
```

可以看到，无论是插入还是删除学生，学生数目都是跟随着变化的。

实例 2：创建包含多条执行语句的触发器。

依然沿用上面例子中的表，对 student_count 表做如下变更：增加 student_class 字段表示具体年级的学生数，其中，0 表示全年级，1 代表 1 年级，……；同样，学生表中也增加该字段。清空两个表中的所有数据。

(1) 删除上例中的两个触发器，初始化 student_count 表中数据，插入三条数据(0,0)(1,0)(2,0)表示全年级、一年级、二年级的初始人数都是 0。

(2) 创建触发器，在插入时首先增加学生总人数，然后判断新增的学生是几年级的，再增加对应年级的学生总数。输入如下命令。

```
DELIMITER $$
    CREATE TRIGGER trigger_student_count_insert
    AFTER INSERT
    ON student_info FOR EACH ROW
    BEGIN
    UPDATE student_count SET student_count = student_count + 1 WHERE student_class = 0;
```

```
    UPDATE student_count SET student_count = student_count + 1 WHERE student_class =  NEW.
student_class;
    END
    $$
    DELIMITER ;
```

(3) 创建触发器,在删除时首先减少学生总人数,然后判断删除的学生是几年级的,再减少对应年级的学生总数。输入如下命令。

```
    DELIMITER $$
    CREATE TRIGGER trigger_student_count_delete
    AFTER DELETE
    ON student_info FOR EACH ROW
    BEGIN
    UPDATE student_count SET student_count = student_count - 1 WHERE student_class = 0;
    UPDATE student_count SET student_count = student_count - 1 WHERE student_class =  OLD.
student_class;
    END
    $$
    DELIMITER ;
```

(4) 向学生表中分别插入多条不同年级的学生信息,观察触发器是否起作用。输入如下命令。

```
  mysql > INSERT INTO student_info
VALUES(NULL,'AAA',1),(NULL,'BBB',1),(NULL,'CCC',2),(NULL,'DDD',2),(NULL,'ABB',1),(NULL,'ACC',1);
  Query OK, 6 rows affected (0.02 sec)
  Records: 6  Duplicates: 0  Warnings: 0
  mysql > SELECT * FROM student_info;
  +-----------+------------+----------------+
  | student_no  | student_name | student_class     |
  +-----------+------------+----------------+
  |       4     | AAA          |             1     |
  |       5     | BBB          |             1     |
  |       6     | CCC          |             2     |
  |       7     | DDD          |             2     |
  |       8     | ABB          |             1     |
  |       9     | ACC          |             1     |
  +-----------+------------+----------------+
  6 rows in set (0.00 sec)
  mysql > SELECT * FROM student_count;
  +----------------+---------------+
  | student_count     | student_class     |
  +----------------+---------------+
  |              6    |             0     |
  |              4    |             1     |
  |              2    |             2     |
  +----------------+---------------+
  3 rows in set (0.00 sec)
```

可以看到,总共插入了 6 条数据,学生总数是 6,1 年级 4 个,2 年级 2 个,trigger 正确执行。

（5）从学生表中分别删除多条不同年级的学生信息，观察触发器是否起作用。输入如下命令。

```
mysql > DELETE FROM student_info WHERE stu_name LIKE 'A % ';
Query OK, 3 rows affected (0.02 sec)

mysql > SELECT * FROM student_info;
+-----------+------------+----------------+
| student_no | student_name | student_class |
+-----------+------------+----------------+
|       5    | BBB        |              1 |
|       6    | CCC        |              2 |
|       7    | DDD        |              2 |
+-----------+------------+----------------+
3 rows in set (0.00 sec)
mysql > SELECT * FROM student_count;
+----------------+----------------+
| student_count  | student_class  |
+----------------+----------------+
|              3 |              0 |
|              1 |              1 |
|              2 |              2 |
+----------------+----------------+
3 rows in set (0.00 sec)
```

从结果可以看出，在学生表中将姓名以 A 开头的学生信息删除，当学生信息删除的同时，数量表也跟随变化。

小结

本章主要讲解了 MySQL 中常用的运算符、流程控制语句，触发器的创建、查看、使用和删除，存储过程和函数的创建、调用、查看、修改和删除。用户可以在 SQL 执行窗口以指令操作数据库，还可以使用存储过程、用户自定义函数和触发器的形式将指令存储起来，以调用或触发激活的方式来操作数据库。

习题

一、单选题

1. 存储过程是 MySQL 服务器中定义并（　　）的 SQL 语句集合。

A. 保存　　B. 执行　　C. 解释　　D. 编写

2. 下面有关存储过程的叙述错误的是（　　）。

A. MySQL 允许在存储过程创建时引用一个不存在的对象

B. 存储过程可以带多个输入参数，也可以带多个输出参数

C. 使用存储过程可以减少网络流量

D. 在一个存储过程中不可以调用其他存储过程

3. MySQL 所支持的触发器不包括(　　)。

A. INSERT 触发器　　B. DELETE 触发器

C. CHECK 触发器　　D. UPDATE 触发器

4. 下面有关触发器的叙述错误的是(　　)。

A. 触发器是一个特殊的存储过程

B. 触发器不可以引用所在数据库以外的对象

C. 在一个表上可以定义多个触发器

D. 触发器在 CHECK 约束之前执行

5. MySQL 为每个触发器创建了(　　)两个临时表。

A. max 和 min　　B. avg 和 sum　　C. int 和 char　　D. old 和 new

6. 下列说法中错误的是(　　)。

A. 常用触发器有 INSERT、UPDATE、DELETE 三种

B. 对于同一个数据表,可以同时有两个 BEFORE UPDATE 触发器

C. new 临时表在 INSERT 触发器中用来访问被插入的行

D. old 临时表中的值只能读不能被更新

二、简答题

1. MySQL 在创建多条执行语句的存储过程或触发器时,为何总是遇到分号就结束创建,然后报错? 如何解决这个问题?

2. MySQL 的触发器的触发顺序是什么?

3. 简述 MySQL 的存储过程与存储函数的区别。

三、上机练习题(以下题目所用数据库为第 4 章习题中的 teaching 数据库)

1. 创建存储过程 selectscore(),用指定的学号查询学生成绩。

2. 编程在表 course 中创建一个触发器 course_detrigger,用于每次当删除表 course 中一行数据时,将会话变量 perl 的值设置为“old course deleted!”。

3. 创建一个存储过程,用于实现给定表 student 中一个学生的姓名即可修改表 student 中该人的电子邮件地址为一个给定的值。

4. 创建一个存储过程 scoreInfo,完成的功能是在表 student、表 course 和表 score 中查询学号、姓名、性别、课程名称、期末分数字段。

5. 假设之前创建的 course 表没有设置外键级联策略,设置触发器,实现在 course 表中删除课程信息时,可自动删除该课程在 score 表上的成绩信息。