**函数的应用**

**一、数据处理函数（第11章）**  
mysql存储过程基本函数包括：字符串类型，数值类型，日期类型

**一）字符串类（文本处理函数）**

|  |  |
| --- | --- |
| **函数** | **说明** |
| CHARSET(str) | 返回字串字符集 |
| CONCAT (string2 [,… ]) | 连接字串 |
| INSTR (string ,substring ) | 返回substring首次在string中出现的位置,不存在返回0 |
| LCASE (string2 ) | 转换成小写 |
| LEFT (string2 ,length ) | 从string2中的左边起取length个字符 |
| LENGTH (string ) | string长度 |
| LOAD\_FILE (file\_name ) | 从文件读取内容 |
| LOCATE (substring , string ,start\_position ] ) | 同INSTR,但可指定开始位置 |
| LPAD (string2 ,length ,pad ) | 重复用pad加在string开头,直到字串长度为length |
| LTRIM (string2 ) | 去除前端空格 |
| REPEAT (string2 ,count ) | 重复count次 |
| REPLACE (str ,search\_str ,replace\_str ) | 在str中用replace\_str替换search\_str |
| RPAD (string2 ,length ,pad) | 在str后用pad补充,直到长度为length |
| RTRIM (string2 ) | 去除后端空格 |
| STRCMP (string1 ,string2 ) | 逐字符比较两字串大小, |
| SUBSTRING (str , position [,length ]) | 从str的position开始,取length个字符, |
| TRIM([[BOTH|LEADING|TRAILING] [padding] FROM]string2) | 去除指定位置的指定字符 |
| UCASE (string2 ) | 转换成大写 |
| RIGHT(string2,length) | 取string2最后length个字符 |
| SPACE(count) | 生成count个空格 |
| lower() | 将串转换为小写 |
| upper() | 将串转换为大写 |
| SOUNDEX() | 返回串的SOUNDEX值 |

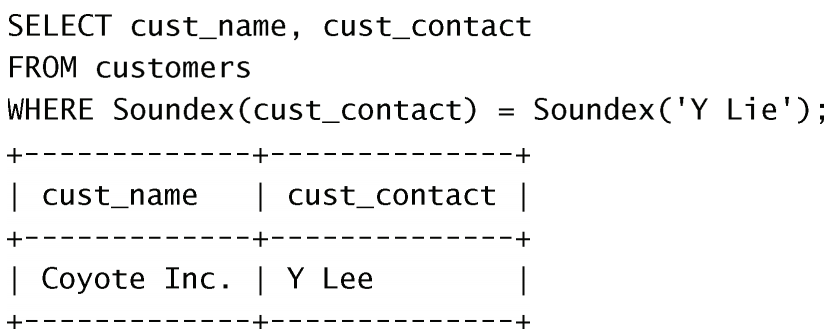
**注：mysql中处理字符串时，默认第一个字符下标为1，即参数position必须大于等于1  
例1：**

mysql> select substring(’abcd’,0,2);  
+———————–+  
| substring(’abcd’,0,2) |  
+———————–+  
|                       |  
+———————–+  
1 row in set (0.00 sec)

**例2：**

mysql> select substring(’abcd’,1,2);  
+———————–+  
| substring(’abcd’,1,2) |  
+———————–+  
| ab                    |  
+———————–+  
1 row in set (0.02 sec)

**例3：匹配发音类似的**



**二）数值类型（数值处理函数）**

|  |  |
| --- | --- |
| **函数** | **说明** |
| ABS (number2 ) | 绝对值 |
| BIN (decimal\_number ) | 十进制转二进制 |
| CEILING (number2 ) | 向上取整 |
| CONV(number2,from\_base,to\_base) | 进制转换 |
| FLOOR (number2 ) | 向下取整 |
| FORMAT (number,decimal\_places ) | 保留小数位数 |
| HEX (DecimalNumber ) | 转十六进制 |
| **注：HEX()中可传入字符串，则返回其ASC-11码，如HEX(’DEF’)返回4142143** | |
| **也可以传入十进制整数，返回其十六进制编码，如HEX(25)返回19** | |
| LEAST (number , number2 [,..]) | 求最小值 |
| MOD (numerator ,denominator ) | 求余 |
| POWER (number ,power ) | 求指数 |
| RAND([seed]) | 随机数 |
| ROUND (number [,decimals ]) | 四舍五入,decimals为小数位数] |
| SIGN (number2 ) | 返回符号,正负或0 |
| SQRT(number2) | 开平方 |
| Cos（） | 返回一个角度的余弦 |
| Sin（） | 返回一个角度的正弦弦 |
| Tan（） | 返回一个角度的正切 |
| Pi（） | 返回圆周率 |
| Exp（） | 返回一个数据的指数值 |
| Mod（） | 返回除操作的余数 |

**注：返回类型并非均为整数**

**例4：**

(1)默认变为整形值  
mysql> select round(1.23);  
+————-+  
| round(1.23) |  
+————-+  
|           1 |  
+————-+  
1 row in set (0.00 sec)

**例5：**

mysql> select round(1.56);  
+————-+  
| round(1.56) |  
+————-+  
|           2 |  
+————-+  
1 row in set (0.00 sec)

**例6：**

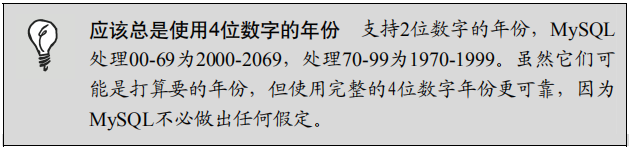
(2)可以设定小数位数，返回浮点型数据

mysql> select round(1.567,2);  
+—————-+  
| round(1.567,2) |  
+—————-+  
|           1.57 |  
+—————-+  
1 row in set (0.00 sec)

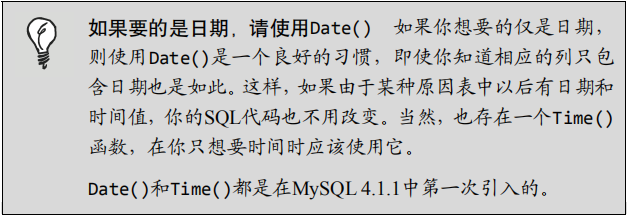
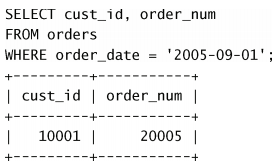
**三）日期类型（日期和时间处理函数）**

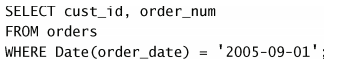
|  |  |
| --- | --- |
| **函数** | **说明** |
| ADDDATE( ) | 增加一个日期（天，周等） |
| ADDTIME (date2 ,time\_interval ) | 将time\_interval加到date2 |
| CONVERT\_TZ (datetime2 ,fromTZ ,toTZ ) | 转换时区 |
| CURRENT\_DATE ( ) | 当前日期 |
| CURRENT\_TIME ( ) | 当前时间 |
| CURRENT\_TIMESTAMP ( ) | 当前时间戳 |
| DATE (datetime ) | 返回datetime的日期部分 |
| DATE\_ADD (date2 , INTERVAL d\_value d\_type ) | 在date2中加上日期或时间（日期运算函数） |
| DATE\_FORMAT (datetime ,FormatCodes ) | 使用formatcodes格式显示datetime |
| DATE\_SUB (date2 , INTERVAL d\_value d\_type ) | 在date2上减去一个时间 |
| DATEDIFF (date1 ,date2 ) | 两个日期差 |
| DAY (date ) | 返回日期的天 |
| DAYNAME (date ) | 英文星期 |
| DAYOFWEEK (date ) | 星期(1-7) ,1为星期天 |
| DAYOFYEAR (date ) | 一年中的第几天 |
| EXTRACT (interval\_name FROM date ) | 从date中提取日期的指定部分 |
| MAKEDATE (year ,day ) | 给出年及年中的第几天,生成日期串 |
| MAKETIME (hour ,minute ,second ) | 生成时间串 |
| MONTHNAME (date ) | 英文月份名 |
| NOW ( ) | 当前时间 |
| SEC\_TO\_TIME (seconds ) | 秒数转成时间 |
| STR\_TO\_DATE (string ,format ) | 字串转成时间,以format格式显示 |
| TIMEDIFF (datetime1 ,datetime2 ) | 两个时间差 |
| TIME\_TO\_SEC (time ) | 时间转秒数 |
| WEEK (date\_time [,start\_of\_week ]) | 第几周 |
| YEAR (datetime ) | 年份 |
| DAYOFMONTH(datetime) | 月的第几天 |
| HOUR(datetime) | 小时 |
| LAST\_DAY(date) | date的月的最后日期 |
| MICROSECOND(datetime) | 微秒 |
| MONTH(datetime) | 月 |
| MINUTE(datetime) | 分 |

**注：可用在INTERVAL中的类型：DAY ,DAY\_HOUR ,DAY\_MINUTE ,DAY\_SECOND ,HOUR ,HOUR\_MINUTE ,HOUR\_SECOND ,MINUTE ,MINUTE\_SECOND,MONTH ,SECOND ,YEAR  
DECLARE variable\_name [,variable\_name...] datatype [DEFAULT value];   
其中，datatype为mysql的数据类型，如:INT, FLOAT, DATE, VARCHAR(length)**

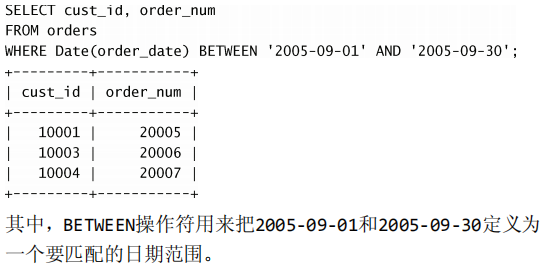


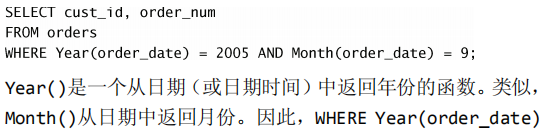
**例7：**





**例8：**





**例9：**

DECLARE l\_int INT unsigned default 4000000;   
DECLARE l\_numeric NUMERIC(8,2) DEFAULT 9.95;   
DECLARE l\_date DATE DEFAULT '1999-12-31';   
DECLARE l\_datetime DATETIME DEFAULT '1999-12-31 23:59:59';  
DECLARE l\_varchar VARCHAR(255) DEFAULT 'This will not be padded';

**四）数据类型转换函数**

| **名称** | **说明** |
| --- | --- |
| BINARY | 将 string 转换为二进制 string |
| CAST() | 将某个值转换为特定类型 |
| CONVERT() | 将某个值转换为特定类型 |

数据类型转换可以通过CAST()和CONVERT()函数来实现。大多数情况下，这两个函数是重叠的，它们反映了SQL语言的演化历史。这两个函 数的功能相似，不过它们的语法不同。虽然并非所有类型的值都能转变为其他数据类型，但总的来说，任何可以转换的值都可以用简单的函数实现转换。

CAST()函数的参数是一个表达式，它包括用AS关键字分隔的源值和目标数据类型。以下例子用于将文本字符串’123’转换为整型:

1 SELECT CAST('123' AS int)

返回值是整型值123。如果试图将一个代表小数的字符串转换为整型值，又会出现什么情况呢？

1 SELECT CAST('123.4' AS int)

CAST()函数和CONVERT()函数都不能执行四舍五入或截断操作。由于123.4不能用int数据类型来表示，所以对这个函数调用将产生一个错误:

1 Server: Msg 245, Level 16, State 1, Line 1 Syntax error converting the varchar value '123.4' to a column of data type int.

要返回一个合法的数值，就必须使用能处理这个值的数据类型。对于这个例子，存在多个可用的数据类型。如果通过CAST()函数将这个值转换为 decimal类型，需要首先定义decimal值的精度与小数位数。在本例中，精度与小数位数分别为9与2。精度是总的数字位数，包括小数点左边和右边 位数的总和。而小数位数是小数点右边的位数。这表示本例能够支持的最大的整数值是9999999，而最小的小数是0.01。

1 SELECT CAST('123.4' AS decimal(9,2))

decimal数据类型在结果网格中将显示有效小数位:

123.40

精度和小数位数的默认值分别是18与0。如果在decimal类型中不提供这两个值，SQL Server将截断数字的小数部分，而不会产生错误。

1 SELECT CAST('123.4' AS decimal)

结果是一个整数值：

123

1、将Int 转为varchar经常用 concat函数，比如concat(8,’0′) 得到字符串 ’80’

2、将varchar 转为Int 用 cast(a as signed) a为varchar类型的字符串

总结：类型转换和SQL Server一样,就是类型参数有点点不同 : CAST(xxx AS 类型) , CONVERT(xxx,类型)

**下面对几种转换进行示例讲解**

**说明：示例中的固定值可以换为实际的查询的表的字段，例如：id**

**1、固定值转为BINARY 二进制型**

**SELECT CAST( 1231 AS BINARY ) AS result**

**运行结果：1231**

**2、int类型值转为CHAR 字符型**

**SELECT CAST(1995 AS CHAR) as result**

**运行结果："1995"**

**3、固定时间字符串转为DATE 日期，格式为 'YYYY-MM-DD’**

**SELECT CAST('2019-08-29 16:50:21' as date) as result**

**运行结果：2019-08-29**

**4、固定时间字符串转为DATETIME 日期加具体的时间，格式为 'YYYY-MM-DD HH:MM:SS’**

**SELECT CAST('2019-08-29 16:50:21' as DATETIME) as result**

**运行结果：2019-08-29 16:50:21**

**5、固定时间字符串转为TIME 时间，格式为 'HH:MM:SS’**

**SELECT CAST('2019-08-29 16:50:21' as TIME) as result**

**运行结果：16:50:21**

**6、float型值通过DECIMAL 获取精度**

**SELECT CAST(220.23211231 AS DECIMAL(10, 3)) AS result**

**运行结果：220.232**

**7、固定字符串转为SIGNED int 型**

**SELECT CAST("12321" AS SIGNED ) AS result**

**运行结果：21312**

**8、固定字符串转为UNSIGNED 无符号int**

**SELECT CAST("12321" AS UNSIGNED ) AS result**

**运行结果：21312**

**五）条件控制函数**

条件判断表达式是为了实现控制流，也就是判断在不同的条件下执行不同的流程。

MySQL中提供了三种条件判断函数：IF()、IFNULL()与CASE。

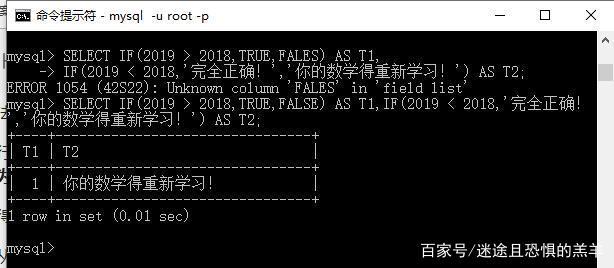
1）IF()

语法格式：IF(expr,v1,v2)；

执行逻辑：若expr是TRUE（即：expr <> 0 and expr <> NULL），则该函数返回值为v1，否则返回值为v2。

所以，我们得出结论是其返回的值是在函数的入参中指定。

接着我们来从简单案例着手，由浅入深来探索其真正的价值。



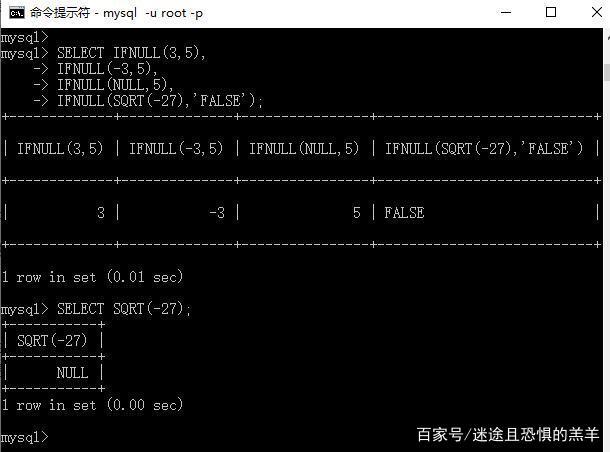
IF(2019 > 2018,TRUE,FALES)执行报错，我们可以得知函数中可以指定的值为MySQL保留词；从IF(2019 > 2018,TRUE,FALSE)返回结果为1而不是TRUE可以得知，MySQL对TRUE再做了一次运算，即视TRUE为1；接着我们尝试是否可以在expr封装其他复杂逻辑（譬如函数）。



函数STRCMP('AEV','UCR')的计算结果为-1，即不等于0也不为NULL；依据函数STRCMP('AEV','UCR')的返回值，IF()返回的结果是v1（即：YES）。

2）IFNULL()

该函数则相对比较简单，从它的语法格式IFNULL(v1,v2)可以看出它只需定义返回值不需定义判断表达式，因为其函数名本身就是判断的含义。



判断两个参数是否为NULL；若两个参数都不为空，则返回第一个参数值；若第一个为空则返回第二个参数值；两个参数也是可以封装更复杂的运算（譬如函数）。

3）CASE

**看语法格式我们会发现此类函数写法上比较麻烦，但其表达的逻辑却最贴近我们日常的思维习惯。**

**CASE expr WHEN v1 THEN r1 [WHEN v2 THEN r2] ... [WHEN vn THEN rn] ... [ELSE r(n+1)] END或CASE WHEN v1 THEN r1 [WHEN v2 THEN r2] ... [WHEN vn THEN rn] ... ELSE r(n+1)] END**

**用通俗的话来读这个判断函数的就是：**在条件expr满足条件v1就返回值r1，满足条件v2就返回值r2……否则就返回一个指定的值r(n+1)。

****

**我们也可以看看第二种写法。**

****

【结论】两种写法的效果是一致的。

1. **系统信息函数**

**系统信息函数用来查询mysql数据库的系统信息**

**1、VERSION()返回数据库版本号**

**mysql> SELECT VERSION();**

**+-------------------------+**

**| VERSION() |**

**+-------------------------+**

**| 5.5.28-0ubuntu0.12.10.2 |**

**+-------------------------+**

**row in set (0.00 sec)**

**2、CONNECTION\_ID()返回数据库的连接次数**

**mysql> SELECT CONNECTION\_ID();**

**+-----------------+**

**| CONNECTION\_ID() |**

**+-----------------+**

**| 36 |**

**+-----------------+**

**row in set (0.00 sec)**

**其实每次连接到mysql的时候就会有显示**

**3、DATABASE()、SCHEMA()返回当前数据库名**

**mysql> SELECT DATABASE(), SCHEMA();**

**+------------+----------+**

**| DATABASE() | SCHEMA() |**

**+------------+----------+**

**| person | person |**

**+------------+----------+**

**row in set (0.00 sec)**

**4、USER()、SYSTEM\_USER()、SESSION\_USER()返回当前用户**

**mysql> SELECT USER(), SYSTEM\_USER(), SESSION\_USER();**

**+----------------+----------------+----------------+**

**| USER() | SYSTEM\_USER() | SESSION\_USER() |**

**+----------------+----------------+----------------+**

**| root@localhost | root@localhost | root@localhost |**

**+----------------+----------------+----------------+**

**row in set (0.00 sec)**

**5、CURRENT\_USER()、CURRENT\_USER返回当前用户**

**mysql> SELECT CURRENT\_USER(), CURRENT\_USER;**

**+----------------+----------------+**

**| CURRENT\_USER() | CURRENT\_USER |**

**+----------------+----------------+**

**| root@localhost | root@localhost |**

**+----------------+----------------+**

**row in set (0.00 sec)**

**6、CHARSET(str)返回字符串str的字符集**

**mysql> SELECT CHARSET('张三');**

**+-------------------+**

**| CHARSET('张三') |**

**+-------------------+**

**| utf8 |**

**+-------------------+**

**row in set (0.00 sec)**

**7、COLLATION(str)返回字符串str的字符排列方式**

**mysql> SELECT COLLATION('张三');**

**+---------------------+**

**| COLLATION('张三') |**

**+---------------------+**

**| utf8\_general\_ci |**

**+---------------------+**

**row in set (0.00 sec)**

**8、LAST\_INSERT\_ID()返回最后生成的AUTO\_INCREMENT值**

**mysql> CREATE TABLE t1(id INT PRIMARY KEY AUTO\_INCREMENT);**

**Query OK, 0 rows affected (0.10 sec)**

**mysql> INSERT INTO t1 VALUES(NULL);**

**Query OK, 1 row affected (0.04 sec)**

**mysql> INSERT INTO t1 VALUES(NULL);**

**Query OK, 1 row affected (0.03 sec)**

**mysql> INSERT INTO t1 VALUES(NULL);**

**Query OK, 1 row affected (0.04 sec)**

**mysql> SELECT \* FROM t1;**

**+----+**

**| id |**

**+----+**

**| 1 |**

**| 2 |**

**| 3 |**

**+----+**

**rows in set (0.00 sec)**

**mysql> SELECT LAST\_INSERT\_ID();**

**+------------------+**

**| LAST\_INSERT\_ID() |**

**+------------------+**

**| 3 |**

**+------------------+**

**row in set (0.00 sec)**

**上面的语句首先创建了一张表t1，其中有一个自增字段id，然后分三次插入NULL，使其自增确认已经存在数据之后，使用LAST\_INSERT\_ID()获取最后自动生成的值。**

1. **用户自定义函数（补充）**

函数可以看作是一个“加工作坊”，这个“加工作坊”接收“调用者”传递过来的“原料”（实际上是函数的参数），然后将这些“原料”“加工处理”成“产品”（实际上是函数的返回值），再把“产品”返回给“调用者”。

一）创建自定义函数的语法格式

**create function 函数名（参数1，参数2，…）returns 返回值的数据类型**

**[函数选项]**

**begin**

**函数体;**

**return 语句;**

**end;**

**函数选项由以下一种或几种选项组合而成。**

**language sql**

**| [not] deterministic**

**| { contains sql | no sql | reads sql data | modifies sql data }**

**| sql security { definer | invoker }**

**| comment '注释'**

**函数选项说明：**

**language sql：默认选项，用于说明函数体使用SQL语言编写。**

**deterministic（确定性）：当函数返回不确定值时，该选项是为了防止“复制”时的不一致性。如果函数总是对同样的输入参数产生同样的结果，则被认为它是“确定的”，否则就是“不确定”的。例如函数返回系统当前的时间，返回值是不确定的。如果既没有给定deterministic也没有给定not deterministic，默认的就是not deterministic。**

**contains sql：表示函数体中不包含读或写数据的语句（例如set命令等）。**

**no sql：表示函数体中不包含SQL语句。**

**reads sql data：表示函数体中包含select查询语句，但不包含更新语句。**

**modifies sql data：表示函数体包含更新语句。如果上述选项没有明确指定，默认是contains sql。**

**sql security：用于指定函数的执行许可。**

**definer：表示该函数只能由创建者调用。**

**invoker：表示该函数可以被其他数据库用户调用。默认值是definer。**

**comment：为函数添加功能说明等注释信息**

二）函数的创建与调用

**应用举例：**

1）单参数

**DELIMITER $$**

**DROP FUNCTION IF EXISTS genPerson$$**

**CREATE FUNCTION genPerson(name varchar(20)) RETURNS varchar(50)**

**BEGIN**

**DECLARE str VARCHAR(50) DEFAULT '';**

**SET @tableName=name;**

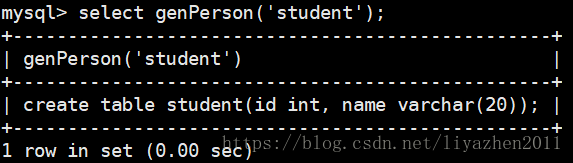
**SET str=CONCAT('create table ', @tableName,'(id int, name varchar(20));');**

**return str;**

**END $$**

**DELIMITER ;**

**select genPerson('student');**



2）无参数

**create FUNCTION f1() returns varchar(30)**

**RETURN DATE\_FORMAT(NOW(),'%Y年:%m月:%d日 %H点:%i分:%s秒');**

**mysql> select f1();**

1. **多参数**

**DELIMITER //**

**CREATE FUNCTION addTwoNumber(x SMALLINT UNSIGNED, Y SMALLINT UNSIGNED)**

**RETURNS SMALLINT**

**BEGIN**

**DECLARE a, b SMALLINT UNSIGNED DEFAULT 10;**

**SET a = x, b = y;**

**RETURN a+b;**

**END//**

**mysql>select addTwoNumber（10,20）;**

三）函数的维护

函数的维护包括查看函数的定义、修改函数的定义以及删除函数的定义等内容。

1．查看函数的定义

（1）查看当前数据库中所有的自定义函数信息，可以使用MySQL命令“show function status;”。如果自定义函数较多，使用MySQL命令“show function status like 模式;”可以进行模糊查询。

（2）查看指定数据库（例如choose数据库）中的所有自定义函数名，可以使用下面的SQL语句，如图6-28所示。

select name from mysql.proc where db = 'choose' and type = 'function' ;

（3）使用MySQL命令“show create function 函数名;”可以查看指定函数名的详细信息。例如查看get\_name\_fn()函数的详细信息，可以使用“show create function get\_name\_fn\G”

（4）函数的信息都保存在information\_schema数据库中的routines表中，可以使用select语句检索routines表，查询函数的相关信息。

select \* from information\_schema.routines where routine\_name='get\_name\_fn'\G

**2．函数定义的修改**

**由于函数保存的仅仅是函数体，而函数体实际上是一些MySQL表达式，因此函数自身不保存任何用户数据。当函数的函数体需要更改时，可以使用drop function语句暂时将函数的定义删除，然后使用create function语句重新创建相同名字的函数即可。这种方法对于存储过程、视图、触发器的修改同样适用。**

**3．函数定义的删除**

**使用MySQL命令“drop function函数名”删除自定义函数。例如删除get\_name\_fn()函数可以使用“drop function get\_name\_fn;”**

1. 条件控制语句

条件控制语句分为两种，一种是if语句，另一种是case语句。

1．if语句

if语句根据条件表达式的值确定执行不同的语句块，if语句的用法格式如下。

**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;**

**说明：end if后必须以“;”结束。**

**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语句的语法格式如下。**

**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;**

**说明：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无需使用“break”语句。**

五）循环语句

MySQL提供了三种循环语句，分别是while、repeat以及loop。除此以外，MySQL还提供了iterate语句以及leave语句用于循环的内部控制。

1．while语句

当条件表达式的值为true时，反复执行循环体，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];**

**说明：end while后必须以“;”结束。**

2．leave语句

leave语句用于跳出当前的循环语句（例如while语句），语法格式如下。

**leave 循环标签;**

**说明：leave 循环标签后必须以“;”结束。**3．iterate语句

iterate语句用于跳出本次循环，继而进行下次循环。iterate语句的语法格式如下。

**iterate 循环标签;**

**说明：iterate循环标签后必须以“;”结束。**

**add\_num: LOOP**

**SET @count=@count+1;**

**IF @count=100 THEN**

**LEAVE add\_num ;**

**ELSE IF MOD(@count,3)=0 THEN**

**ITERATE add\_num;**

**SELECT \* FROM employee ;**

**END LOOP add\_num ;**

4．repeat语句

当条件表达式的值为false时，反复执行循环，直到条件表达式的值为true，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];**

**说明：end repeat后必须以“;”结束。**

5．loop语句的语法格式

由于loop循环语句本身没有停止循环的语句，因此loop通常使用leave语句跳出loop循环，loop的语法格式如下。

**[循环标签:] loop**

**循环体;**

**if 条件表达式 then**

**leave [循环标签];**

**end if;**

**end loop;**

**说明：end loop后必须以“;”结束。**

**附：**

**一、函数和存储过程的区别**

|  | **关键字** | **调用语法** | **返回值** | **应用场景** |
| --- | --- | --- | --- | --- |
| 函数 | FUNCTION | SELECT 函数（） | 有且仅有一个 | 一般用于查询结果为一个值并返回时，当有返回值而且仅仅一个 |
| 存储过程 | PROCEDURE | CALL 存储过程（） | 可以有0个或多个 | 一般用于更新 |

# 二、函数

## 一、创建语法

（1）语法

CREATE FUNCTION 函数名(参数名 参数类型,...) RETURNS 返回类型

BEGIN

函数体

END

（2）注意事项

1. 参数列表包含两部分：参数名和参数类型。
2. 函数体：必须有return语句，否则会报错。
3. 若函数体中仅有一条语句，则可以省略BEGIN END。
4. 使用delimiter语句设置结束标记。

（3）调用函数

SELECT 函数名（实参列表） //执行完函数图中的所有语句，并显示返回值

## 二、无参数有返回值

（1）举例

返回公司的员工数

（2）代码

DELIMITER $

CREATE FUNCTION myf1() RETURNS INT

BEGIN

DECLARE c INT DEFAULT 0;#定义局部变量

SELECT COUNT(\*) INTO c#赋值

FROM employees;

RETURN c;

END $

SELECT myf1() $

（3）运行结果



三、有参数有返回值

（1）举例

根据员工名，返回他的工资

（2）代码

CREATE FUNCTION myf2(empName VARCHAR(20)) RETURNS DOUBLE

BEGIN

SET @sal=0;#定义用户变量

SELECT salary INTO @sal #赋值

FROM employees

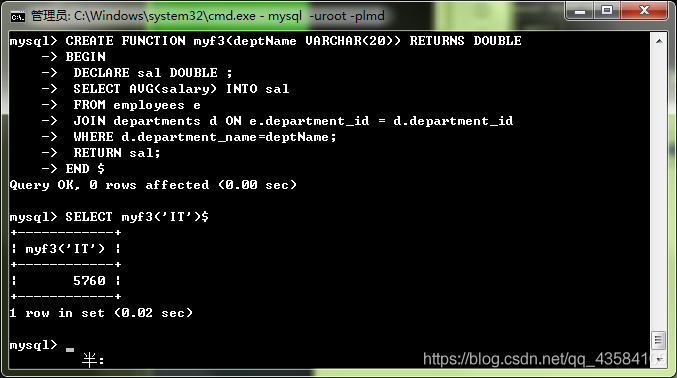
WHERE last\_name = empName;

RETURN @sal;

END $

SELECT myf2('kochhar') $

（3）运行结果

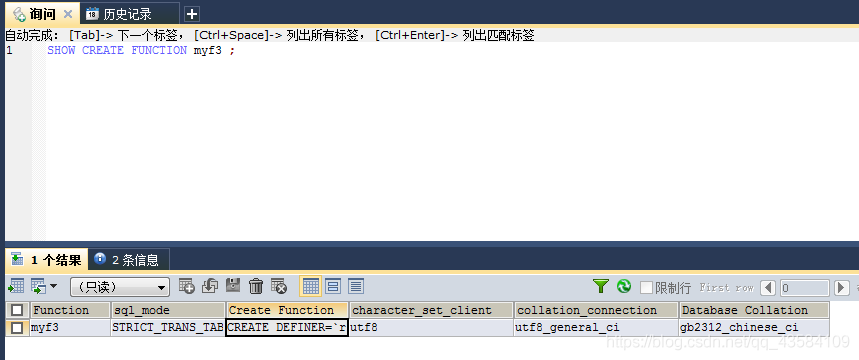


## 四、查看函数

（1）语法

SHOW CREATE FUNCTION 函数名;

（2）举例



## 五、删除函数

DROP FUNCTION 函数名;