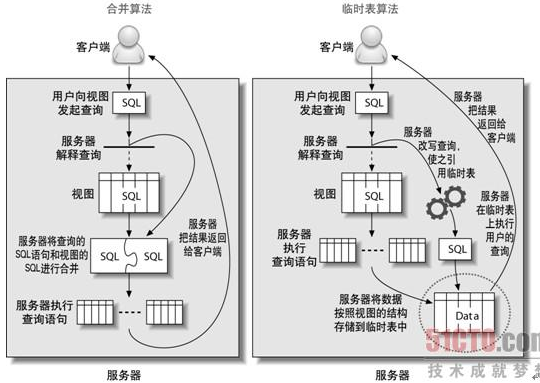
视图和存储程序

# 使用视图

视图本身是一张虚拟表（可以通过show tables显示），不存放任何数据。不能对视图创建触发器，也不能使用drop table 命令删除视图。MySQL使用两种办法来处理视图。这两种算法是合并算法（MERGE）和临时表算法(TEMPTABLE)，MySQL会尽量使用合并算法。如果采用临时表，explain会显示为派生表（DERIVED）。



注意：在视图包含GROUP BY、DISTINCT 、聚集函数、UNION 、子查询或其他无法保持视图返回的行和待查询的行之间一对一关系的结构的时候，MySQL就会使用临时表算法。

## 创建视图

|  |
| --- |
| CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  VIEW view\_name [(column\_list)]  AS select\_statement  [WITH [CASCADED | LOCAL] CHECK OPTION]   * CREATE：表示新建视图； * REPLACE：表示替换已有视图 * ALGORITHM ：表示视图选择算法 * view\_name ：视图名 * column\_list：属性列 * select\_statement：表示select语句 * [WITH [CASCADED | LOCAL] CHECK OPTION]参数表示视图在更新时保证在视图的权限范围之内 * 可选的ALGORITHM子句是对标准SQL的MySQL扩展。ALGORITHM可取三个值：MERGE、TEMPTABLE或UNDEFINED。如果没有ALGORITHM子句，默认算法是UNDEFINED（未定义的）。算法会影响MySQL处理视图的方式。对于MERGE，会将引用视图的语句的文本与视图定义合并起来，使得视图定义的某一部分取代语句的对应部分。对于TEMPTABLE，视图的结果将被置于临时表中，然后使用它执行语句。对于UNDEFINED，MySQL自己选择所要使用的算法。如果可能，它倾向于MERGE而不是TEMPTABLE，这是因为MERGE通常更有效，而且如果使用了临时表，视图是不可更新的。 * LOCAL和CASCADED为可选参数，决定了检查测试的范围，默认值为CASCADED。 |
| --创建视图例子  CREATE VIEW stu\_class (  id  ,NAME  ,glass  )  AS  SELECT student.`stuno`  ,student.`stuname`  ,stuinfo.`class`  FROM student  ,stuinfo  WHERE student.`stuno` = stuinfo.`stuno` |

## 查看视图

|  |
| --- |
| 查看视图必须要有SHOW VIEW权限  查看视图的方法包括：DESCRIBE、SHOW TABLE STATUS、SHOW CREATE VIEW |
| #结果显示了视图的字段定义、字段的数据类型、是否为空、是否为主/外键、默认值和额外信息  desc host\_summary\_by\_file\_io;  #COMMENT的值为VIEW说明该表为视图，其他的信息为NULL说明这是一个虚表，如果是基表那么会基表的信息，这是基表和视图的区别  SHOW TABLE STATUS LIKE 'host\_summary\_by\_file\_io'  #SHOW CREATE VIEW语句查看视图详细信息  SHOW CREATE VIEW host\_summary\_by\_file\_io;  CREATE ALGORITHM=TEMPTABLE DEFINER=`root`@`localhost` SQL SECURITY INVOKER VIEW `host\_summary\_by\_file\_io` AS select if(isnull(`performance\_schema`.`events\_waits\_summary\_by\_host\_by\_event\_name`.`HOST`),'background',`performance\_schema`.`events\_waits\_summary\_by\_host\_by\_event\_name`.`HOST`) AS `host`,sum(`performance\_schema`.`events\_waits\_summary\_by\_host\_by\_event\_name`.`COUNT\_STAR`) AS `ios`,`sys`.`format\_time`(sum(`performance\_schema`.`events\_waits\_summary\_by\_host\_by\_event\_name`.`SUM\_TIMER\_WAIT`)) AS `io\_latency` from `performance\_schema`.`events\_waits\_summary\_by\_host\_by\_event\_name` where (`performance\_schema`.`events\_waits\_summary\_by\_host\_by\_event\_name`.`EVENT\_NAME` like 'wait/io/file/%') group by if(isnull(`performance\_schema`.`events\_waits\_summary\_by\_host\_by\_event\_name`.`HOST`),'background',`performance\_schema`.`events\_waits\_summary\_by\_host\_by\_event\_name`.`HOST`) order by sum(`performance\_schema`.`events\_waits\_summary\_by\_host\_by\_event\_name`.`SUM\_TIMER\_WAIT`) desc  #查看所有的视图  SELECT \* FROM `information\_schema`.`VIEWS` |

## 修改视图

|  |
| --- |
| 修改视图是指修改数据库中存在的视图，当基本表的某些字段发生变化时，可以通过修改视图来保持与基本表的一致性。MYSQL中通过CREATE OR REPLACE VIEW 语句和ALTER语句来修改视图。  ALTER OR REPLACE [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  VIEW view\_name [(column\_list)]  AS select\_statement  [WITH [CASCADED | LOCAL] CHECK OPTION]  或者  ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]  VIEW view\_name [(column\_list)]  AS select\_statement  [WITH [CASCADED | LOCAL] CHECK OPTION] |
| #举例  ALTER VIEW stu\_class AS SELECT stuno FROM student; |

## 更新视图

更新视图是指通过视图来插入、更新、删除表数据，因为视图是虚表，其中没有数据。通过视图更新的时候都是转到基表进行更新，如果对视图增加或者删除记录，实际上是对基表增加或删除记录。当视图中包含如下内容的时候，视图的更新操作将不能被执行：

* 视图中包含基本中被定义为非空的列
* 定义视图的SELECT语句后的字段列表中使用了数学表达式
* 定义视图的SELECT语句后的字段列表中使用聚合函数
* 定义视图的SELECT语句中使用了DISTINCT、UNION、TOP、GROUP BY 、HAVING子句

注意：视图中包含WITH CHECK OPTION，更新的时候，必须满足视图条件。

## 删除视图

|  |
| --- |
| DROP VIEW [IF EXISTS]  view\_name [, view\_name] ...  [RESTRICT | CASCADE] |

注意：物化视图可以使用触发器来实现。

# 使用存储程序

## 存储过程和函数

MYSQL中创建存储过程和函数分别使用CREATE PROCEDURE和CREATE FUNCTION。使用CALL语句来调用存储过程，存储过程也可以调用其他存储过程函数可以从语句外调用，能返回标量值。

### 创建存储过程

|  |
| --- |
| CREATE PROCEDURE sp\_name ([ proc\_parameter ]) [ characteristics..] routine\_body  roc\_parameter指定存储过程的参数列表，列表形式如下：  [IN|OUT|INOUT] param\_name type  其中in表示输入参数，out表示输出参数，inout表示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  characteristic:  LANGUAGE SQL  | [NOT] DETERMINISTIC  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }  | SQL SECURITY { DEFINER | INVOKER }  | COMMENT 'string'  routine\_body:  Valid SQL procedure statement or statements  **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正确。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，默认为[NOT] DETERMINISTIC  **CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA**：指明子程序使用SQL语句的限制。CONTAINS SQL表明子程序包含SQL语句，但是不包含读写数据的语句；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 }** ：指明谁有权限来执行。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’ ：注释信息，可以用来描述存储过程或函数  routine\_body是SQL代码的内容，可以用BEGIN…END来表示SQL代码的开始和结束。 |
| **注意：“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！当使用DELIMITER命令时，应该避免使用反斜杠（\）字符，因为反斜杠是MYSQL的转义字符！！！**  delimiter //  CREATE PROCEDURE optimize\_table(db\_name VARCHAR(64))  BEGIN  DECLARE t VARCHAR(64);  DECLARE done INT DEFAULT 0;  DECLARE c CURSOR FOR  SELECT table\_name  FROM INFORMATION\_SCHEMA.TABLES  WHERE TABLE\_SCHEMA=db\_name AND TABLE\_TYPE='BASE TABLE';  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done=1;  OPEN c;  tables\_loop: LOOP  FETCH c INTO t;  IF done THEN  LEAVE tables\_loop;  END IF;  SET @stmt\_text:= CONCAT("OPTIMIZE TABLE ",db\_name,'.',t);  PREPARE stmt FROM @stmt\_text;  EXECUTE stmt;  DEALLOCATE PREPARE stmt;  END LOOP;  CLOSE c;  END//  delimiter ; |

### 存储函数

|  |
| --- |
| 创建存储函数，需要使用CREATE FUNCTION语句，基本语法如下：  CREATE FUNCTION func\_name([func\_parameter])  RETURNS TYPE  [characteristics...] 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：为存储函数的参数列表，[IN|OUT|INOUT]PARAM\_NAME TYPE其中，IN表示输入参数，OUT表示输出参数，INOUT表示既可以输入也可以输出；param\_name表示参数名称；type表示参数类型，该类型可以是MYSQL数据库中的任意类型  注意：指定参数为IN、OUT、INOUT只对PROCEDURE是合法的。（FUNCTION中总是默认是IN参数）RETURNS子句对FUNCTION做指定，对函数而言这是强制的。他用来指定函数的返回类型，而且函数体必须包含一个RETURN value语句  RETURNS TYPE语句表示函数返回数据的类型；  characteristics：指定存储函数的特性，取值与创建存储过程时相同  #举例  DELIMITER //  CREATE FUNCTION NameByT()  RETURNS CHAR(50)  RETURN (SELECT NAME FROM t3 WHERE id=2);  //  DELIMITER ;  注意：RETURNS CHAR(50)数据类型的时候，RETURNS 是有S的，而RETURN (SELECT NAME FROM t3 WHERE id=2)的时候RETURN是没有S的  如果在存储函数中的RETURN语句返回一个类型不同于函数的RETURNS子句中指定类型的值，返回值将被强制转换为恰当的类型。 |

### 变量的使用

|  |
| --- |
| #定义变量，在存储过程中定义变量  DECLARE var\_name[,varname]...date\_type[DEFAULT VALUE];  说明：var\_name为局部变量的名称。DEFAULT VALUE子句给变量提供一个默认值。值除了可以被声明为一个常数外，还可以被指定为一个表达式。如果没有DEFAULT子句，初始值为NULL  举例：DECLARE MYPARAM INT DEFAULT 100; |
| #为变量赋值  第一种方式：定义变量之后，为变量赋值可以改变变量的默认值，MYSQL中使用SET语句为变量赋值，在select中使用:=代替=为变量赋值  SET var\_name=expr[,var\_name=expr]...  DECLARE var1,var2,var3 INT;  SET var1=10,var2=20;  SET var3=var1+var2;  第二种方式：MYSQL中还可以通过SELECT…INTO为一个或多个变量赋值  DECLARE NAME CHAR(50);  DECLARE id DECIMAL(8,2);  SELECT id,NAME INTO id ,NAME FROM t3 WHERE id=2; |

### 定义条件和处理程序

特定条件需要特定处理。这些条件可以联系到错误，以及子程序中的一般流程控制。定义条件是事先定义程序执行过程中遇到的问题，处理程序定义了在遇到这些问题时候应当采取的处理方式，并且保证存储过程或函数在遇到警告或错误时能继续执行。这样可以增强存储程序处理问题的能力，避免程序异常停止运行。

|  |
| --- |
| #定义条件  DECLARE condition\_name CONDITION FOR[condition\_type]  [condition\_type]:  SQLSTATE[VALUE] sqlstate\_value |mysql\_error\_code  说明：  condition\_name：表示条件名称  condition\_type：表示条件的类型  sqlstate\_value和mysql\_error\_code：都可以表示mysql错误  sqlstate\_value：长度5的字符串错误代码  mysql\_error\_code为数值类型错误代码，例如：ERROR1142(42000)中，sqlstate\_value的值是42000，mysql\_error\_code的值是1142  这个语句指定需要特殊处理条件。他将一个名字和指定的错误条件关联起来。  这个名字随后被用在定义处理程序的DECLARE HANDLER语句中  定义ERROR1148(42000)错误，名称为command\_not\_allowed。  #方法一：使用sqlstate\_value  DECLARE command\_not\_allowed CONDITION FOR SQLSTATE '42000'  #方法二：使用mysql\_error\_code  DECLARE command\_not\_allowed CONDITION FOR SQLSTATE 1148 |
| #定义处理程序  DECLARE handler\_type HANDLER FOR  condition\_value[,...] sp\_statement  handler\_type:  CONTINUE | EXIT | UNDO  condition\_value:  SQLSTATE [VALUE] sqlstate\_value |  condition\_name | SQLWARNING  | NOT FOUND | SQLEXCEPTION | mysql\_error\_code  说明：  handler\_type参数：指明错误的处理方式，该参数有3个取值。这3个取值分别是CONTINUE、EXIT和UNDO。CONTINUE表示遇到错误不进行处理，继续向下执行；EXIT表示遇到错误后马上退出；UNDO表示遇到错误后撤回之前的操作，MySQL中暂时还不支持这种处理方式。  注意：通常情况下，执行过程中遇到错误应该立刻停止执行下面的语句，并且撤回前面的操作。但是，MySQL中现在还不能支持UNDO操作。因此，遇到错误时最好执行EXIT操作。如果事先能够预测错误类型，并且进行相应的处理，那么可以执行CONTINUE操作。  condition\_value参数指明：错误类型，该参数有6个取值。sqlstate\_value和mysql\_error\_code与条件定义中的是同一个意思。condition\_name是DECLARE定义的条件名称；SQLWARNING：表示所有以01开头的sqlstate\_value值；NOT FOUND表示所有以02开头的sqlstate\_value值；SQLEXCEPTION表示所有没有被SQLWARNING或NOT FOUND捕获的sqlstate\_value值；sp\_statement表示一些存储过程或函数的执行语句。  #举例，遇到条件，执行continue操作  #方法一：捕获sqlstate\_value  DECLARE CONTINUE HANDLER FOR SQLSTATE '42000'  SET @info='CAN NOT FIND';  #方法二：捕获mysql\_error\_code  DECLARE CONTINUE HANDLER FOR <strong>1148</strong>SET @info='CAN NOT FIND';  #方法三：先定义条件，然后调用  DECLARE can\_not\_find CONDITION FOR 1146 ;  DECLARE CONTINUE HANDLER FOR can\_not\_find  SET @info='CAN NOT FIND';  #方法四：使用SQLWARNING  DECLARE EXIT HANDLER FOR SQLWARNING  SET @info='ERROR';  #方法五：使用NOT FOUND  DECLARE EXIT HANDLER FOR NOT FOUND  SET @info='CAN NOT FIND';  #方法六：使用SQLEXCEPTION  DECLARE EXIT HANDLER FOR SQLEXCEPTION  SET @info='ERROR'; |

### 游标

查询语句可能查询出多条记录，在存储过程和函数中使用游标来逐条读取查询结果集中的记录。游标的使用包括声明游标、打开游标、使用游标和关闭游标。游标必须声明在处理程序之前，并且声明在变量和条件之后。

|  |
| --- |
| #声明光标  **DECLARE cursor\_name CURSOR FOR select\_statement ;**  **说明：cursor\_name参数表示光标的名称；select\_statement参数表示SELECT语句的内容，返回一个用于创建光标的结果集**  **举例：DECLARE cur\_employee CURSOR FOR SELECT name, age FROM employee ;** |
| #打开光标  OPEN cursor\_name ;  说明：cursor\_name参数表示光标的名称。  举例：OPEN cur\_employee ; |
| #使用光标  FETCH cur\_employee INTO var\_name[,var\_name…] ;  说明：cursor\_name参数表示光标的名称；var\_name参数表示将光标中的SELECT语句查询出来的信息存入该参数中。var\_name必须在声明光标之前就定义好。  举例：FETCH cur\_employee INTO emp\_name, emp\_age ; |
| #关闭光标  CLOSE cursor\_name ;  说明：cursor\_name参数表示光标的名称。  举例：CLOSE cur\_employee ; |

注意：MYSQL中，光标只能在存储过程和函数中使用！

### 流程控制的使用

存储过程和函数中可以使用流程控制来控制语句的执行。MySQL中可以使用IF语句、CASE语句、LOOP语句、LEAVE语句、ITERATE语句、REPEAT语句和WHILE语句来进行流程控制。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，构造可以被嵌套

|  |
| --- |
| #IIF语句用来进行条件判断。根据是否满足条件，将执行不同的语句。其语法的基本形式如下：  IF search\_condition THEN statement\_list  [ELSEIF search\_condition THEN statement\_list] ...  [ELSE statement\_list]  END IF  说明：search\_condition参数表示条件判断语句；statement\_list参数表示不同条件的执行语句。  举例：  IF age>20 THEN SET @count1=@count1+1;  ELSEIF age=20 THEN SET @count2=@count2+1;  ELSE SET @count3=@count3+1;  END IF; |
| #CASE语句也用来进行条件判断，其可以实现比IF语句更复杂的条件判断。CASE语句的基本形式如下：  CASE case\_value  WHEN when\_value THEN statement\_list  [WHEN when\_value THEN statement\_list] ...  [ELSE statement\_list]  END CASE  说明：case\_value参数表示条件判断的变量；when\_value参数表示变量的取值；  statement\_list参数表示不同when\_value值的执行语句。  或者  CASE  WHEN search\_condition THEN statement\_list  [WHEN search\_condition THEN statement\_list] ...  [ELSE statement\_list]  END CASE  说明：其中，search\_condition参数表示条件判断语句；statement\_list参数表示不同条件的执行语句。  举例：  CASE age  WHEN 20 THEN SET @count1=@count1+1;  ELSE SET @count2=@count2+1;  END CASE ;  注意：这里的CASE语句和“控制流程函数”里描述的SQL CASE表达式的CASE语句有轻微不同。这里的CASE语句不能有ELSE NULL子句，并且用END CASE替代END来终止！！ |
| LOOP语句可以使某些特定的语句重复执行，实现一个简单的循环。但是LOOP语句本身没有停止循环的语句，必须是遇到LEAVE语句等才能停止循环。LOOP语句的语法的基本形式如下：  [begin\_label:] LOOP  statement\_list  END LOOP [end\_label]  说明：begin\_label参数和end\_label参数分别表示循环开始和结束的标志，这两个标志必须相同，而且都可以省略；statement\_list参数表示需要循环执行的语句。  举例：  add\_num: LOOP  SET @count=@count+1;  END LOOP add\_num ; |
| LEAVE语句主要用于跳出循环控制。其语法形式如下：  LEAVE label  说明：label参数表示循环的标志。  举例：  add\_num: LOOP  SET @count=@count+1;  IF @count=100 THEN  LEAVE add\_num ;  END LOOP add\_num ; |
| ITERATE语句也是用来跳出循环的语句。但是，ITERATE语句是跳出本次循环，然后直接进入下一次循环。ITERATE语句只可以出现在LOOP、REPEAT、WHILE语句内。ITERATE语句的基本语法形式如下：  ITERATE label  说明：label参数表示循环的标志。  举例：  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 ; |
| REPEAT语句是有条件控制的循环语句。当满足特定条件时，就会跳出循环语句。REPEAT语句的基本语法形式如下：  [begin\_label:] REPEAT  statement\_list  UNTIL search\_condition  END REPEAT [end\_label]  说明：statement\_list参数表示循环的执行语句；search\_condition参数表示结束循环的条件，满足该条件时循环结束。  举例：  REPEAT  SET @count=@count+1;  UNTIL @count=100  END REPEAT ; |
| WHILE语句也是有条件控制的循环语句。但WHILE语句和REPEAT语句是不一样的。WHILE语句是当满足条件时，执行循环内的语句。WHILE语句的基本语法形式如下：  [begin\_label:] WHILE search\_condition DO  statement\_list  END WHILE [end\_label]  说明：search\_condition参数表示循环执行的条件，满足该条件时循环执行；  statement\_list参数表示循环的执行语句。  举例：  WHILE @count<100 DO  SET @count=@count+1;  END WHILE ; |

### 调用存储过程

存储过程和存储函数都是存储在服务器端的SQL语句的集合，要使用这些已经定义好的存储过程和存储函数就必须要通过调用的方式来实现。存储过程是通过CALL语句来调用的。而存储函数的使用方法与MySQL内部函数的使用方法是一样的。执行存储过程和存储函数需要拥有EXECUTE权限EXECUTE权限信息存储在information\_schema数据库下面的**USER\_PRIVILEGES**表中。

CALL语句的基本语法形式如下：

|  |
| --- |
| CALL sp\_name([parameter[,…]]) ;  说明：sp\_name是存储过程的名称；parameter是指存储过程的参数。 |

### 查看存储过程和函数

|  |
| --- |
| **#SHOW STATUS语句只能查看存储过程或函数是操作哪一个数据库、存储过程或函数的名称、类型、谁定义的、创建和修改时间、字符编码等信息。**  SHOW { PROCEDURE | FUNCTION } STATUS [ LIKE ' pattern ' ] ;  # SHOW CREATE语句查看存储过程和函数的定义  SHOW CREATE { PROCEDURE | FUNCTION } sp\_name ;  #从information\_schema.Routines表中查看存储过程和函数的信息  SELECT \* FROM information\_schema.Routines  WHERE ROUTINE\_NAME=' sp\_name ' ; |

### 修改存储过程和函数

|  |
| --- |
| 修改存储过程和函数是指修改已经定义好的存储过程和函数。MySQL中通过ALTER PROCEDURE语句来修改存储过程。通过ALTER FUNCTION语句来修改存储函数。MySQL中修改存储过程和函数的语句的语法形式如下：  ALTER {PROCEDURE | FUNCTION} sp\_name [characteristic ...]  characteristic:  { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }  | SQL SECURITY { DEFINER | INVOKER }  | COMMENT 'string'  说明：  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。  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 }指明谁有权限来执行。  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 ‘string’是注释信息。  注意：修改存储过程和函数，只能修改他们的权限，目前MYSQL还不提供对已存在的存储过程和函数代码的修改。如果要修改，只能通过先DROP掉，然后重新建立新的存储过程和函数来实现。 |

### 删除存储过程和函数

删除存储过程和函数指删除数据库中已经存在的存储过程和函数。MySQL中使用DROP PROCEDURE语句来删除存储过程。通过DROP FUNCTION语句来删除存储函数。

|  |
| --- |
| DROP { PROCEDURE| FUNCTION } sp\_name;  说明：sp\_name参数表示存储过程或函数的名称  举例：DROP PROCEDURE Proc; |

## 触发器

触发器是一个特殊的存储过程，不同的是存储过程要用CALL来调用，而触发器不需要使用CALL也不需要手工启动，只要当一个预定义的事件发生的时候，就会被MYSQL自动调用。

### 创建触发器

|  |
| --- |
| CREATE TRIGGER trigger\_name trigger\_time trigger\_event  ON tbl\_name FOR EACH ROW trigger\_stmt  trigger\_name：触发程序是与表有关的命名数据库对象，当表上出现特定事件时，将激活该对象。触发程序与命名为tbl\_name的表相关。tbl\_name必须引用永久性表。不能将触发程序与临时表表或视图关联起来。  trigger\_time：是触发程序的动作时间。它可以是BEFORE或AFTER，以指明触发程序是在激活它的语句之前或之后触发。  trigger\_event：指明了激活触发程序的语句的类型。trigger\_event可以是下述值之一：   * INSERT：将新行插入表时激活触发程序，例如，通过INSERT、LOAD DATA和REPLACE语句。 * UPDATE：更改某一行时激活触发程序，例如，通过UPDATE语句。 * DELETE：从表中删除某一行时激活触发程序，例如，通过DELETE和REPLACE语句。   trigger\_stmt：是当触发程序激活时执行的语句。如果你打算执行多个语句，可使用BEGIN … END复合语句结构。这样，就能使用存储子程序中允许的相同语句。可以使用NEW.COL\_NAME来引用在insert或update触发器里将被插入或修改的那个新行中的列（新的）。类似的，可以使用OLD.COL\_NAME来引用DELETE或update删除或者更新的原行的数据（老的） |
| 举例：  CREATE TABLE account(acct\_num INT ,amount DECIMAL(10,2));  CREATE TRIGGER ins\_sum BEFORE INSERT ON account  FOR EACH ROW SET @SUM=@SUM+new.amount; |

### 查看触发器

|  |
| --- |
| show triggers;  SELECT \* FROM `information\_schema`.`TRIGGERS` WHERE `TRIGGER\_NAME`='ins\_sum';  #TRIGGER\_SCHEMA 表示触发器所在的数据库  #TRIGGER\_NAME表示触发器的名称  #EVENT\_OBJECT\_TABLE表示在哪个表上触发  #ACTION\_STATEMENT 表示触发器触发的时候执行的具体操作  #ACTION\_ORIENTATION是ROW，表示在每条记录上都触发  #ACTION\_TIMING表示触发的时刻是BEFORE |

### 删除触发器

|  |
| --- |
| DROP TRIGGER [schema\_name.]trigger\_name  其中（schema\_name）是可选的，如果省略了schema（方案），将从当前方案中舍弃触发程序。 |

## 事件

|  |
| --- |
| MySQL默认不支持事件，MySQL开启事件的方式：  配置文件中加入EVENT\_SCHEDULER=ON或者  set global event\_scheduler=on/ 1  注意：当配置文件设置事件的类型是disabled的时候，不能更改其状态 |
| #创建事件  CREATE [DEFINER = { user | CURRENT\_USER }] EVENT [IF NOT EXISTS] e\_name  ON SCHEDULE  {AT datetime | EVERY expr interval [starttime] [endtime]  ON COMPLETION [NOT] PRESERVE]  DO event\_body;  说明：  DEFINER: 定义事件执行的时候检查权限的用户。  ON SCHEDULE schedule: 定义执行的时间和时间间隔。  AT:一次执行，EVERY定时持续执行  ON COMPLETION [NOT] PRESERVE: 定义事件是一次执行还是永久执行，默认为一次执行，即NOT PRESERVE。  DO：事件主体内容，可以使用begin…end复合语句  举例：  CREATE EVENT event\_test  ON SCHEDULE EVERY 1 day STARTS  date\_add(concat(current\_date(), ' 00:00:00'), interval 0 second)  ON COMPLETION PRESERVE ENABLE  DO  delete from v9\_admin where UNIX\_TIMESTAMP()-lastlogintime>100 |
| #禁用和开启事件  ALTER EVENT event\_name DISABLE;  ALTER EVENT event\_name ENABLE; |

# 存储过程动态拼接sql

|  |
| --- |
| delimiter //  CREATE PROCEDURE optimize\_table(db\_name VARCHAR(64))  BEGIN  DECLARE t VARCHAR(64);  DECLARE done INT DEFAULT 0;  DECLARE c CURSOR FOR  SELECT table\_name  FROM INFORMATION\_SCHEMA.TABLES  WHERE TABLE\_SCHEMA=db\_name AND TABLE\_TYPE='BASE TABLE';  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done=1;  OPEN c;  tables\_loop: LOOP  FETCH c INTO t;  IF done THEN  LEAVE tables\_loop;  END IF;  SET @stmt\_text:= CONCAT("OPTIMIZE TABLE ",db\_name,'.',t);  PREPARE stmt FROM @stmt\_text;  EXECUTE stmt;  DEALLOCATE PREPARE stmt;  END LOOP;  CLOSE c;  END//  delimiter ; |
| #prepare预处理语句  PREPARE stmt\_name FROM {‘str’ | @var\_name }  #经过预处理的语句可以通过execute语句来执行  EXECUTE stmt\_name  #最后使用deallocate来释放  DEALLOCATE PREPARE stmt\_name |