MySQL Executable Objects

- 目的:
-     嵌入式SQL
-     过程化SQL
- 参考课本第8章和5.7节
- 内容:
  - 存储函数，存储过程，触发器，事件
  - 变量范围
  - 流程控制语句IF Then， Loop，Case
  - 游标

MySQL Executable Objects (1)

- 存放在MySQL服务器端，供重复使用的对象叫做存储程序。存储程序分为以下四种：
- (1)存储过程（ Stored procedures ）：不直接返回一个计算结果，但可以用来完成一般的运算或是生成一个结果集并传递回客户端。

- 一条SQL语句如果比作一行java代码，存储过程就相当于一个java方法，可以包含许多SQL语句，进行更复杂的操作。

- (2)存储函数（ Stored functions ）：返回一个计算结果，该结果可以用在表达式里。
- 就相当于自定义MySQL函数一样，它的作用和MySQL函数类似，只不过需要我们自己去定义。

- (3)触发器（ Triggers ）：与数据表相关联，当那个数据表被INSERT、DELETE或UPDATE语句修改时，触发器将自动执行。
- 如果表关联了触发器，当表数据有修改操作时，触发器将自动执行，至于做什么是自己定义的。

- (4)事件（ Events ）：根据时间表在预定时刻自动执行。
- 比如，可以自己定一个开始时间点，然后让它每隔指定的时间段重复做某些事情。

MySQL Executable Objects (2)

- 存储程序优点
  - 重复使用
  - 运行效率高
  - 存储程序都保存在服务端，降低了客户机和服务器之间的通信量
  - 可以提高数据库安全性，可以限制存储程序的访问权限。

A Simple MySQL Stored Procedure (存储过程)

- Get all employee details.

```
mysql > delimiter //   -- replace default statement delimiter ; by //
mysql >  create procedure GetAllEmployees()
          begin
              select  * from employees;
          end //
mysql > delimiter ;  -- make ; the statement delimiter again
```

- Calling Stored Procedure :

```
mysql > call GetAllEmployees();
```

DELIMITER 定好结束符为“\\", 然后最后又定义为";", MYSQL的默认结束符为";".

变量Variables

- 存储程序可以有他们自己的变量.
- 变量声明 declare :

  declare variable_name datatype [default default_value];

  – Datatype can be any type supported by MySQL.

  Examples: declare total_sale int default 0;

  declare x, y int;

- Assign values to variables:

  Examples: declare total_count int;

  set total_count = 10;

  declare total_counts int;

  select count(*) into total_counts from employees;

变量作用范围Variable Scope

- If you declare a variable inside a stored procedure, it will be out of scope when the end statement of stored procedure is reached.
- If you declare a variable inside a begin-end block, it will be out of scope if the end is reached.
- A variable that begins with the @ sign at the beginning is a session variable（会话变量）. It is available and accessible until the end of the session.

@variable

- MySQL has the concept of session variables. They are loosely typed variables that may be initialized somewhere in a session and keep their value until the session ends.
- They are with an @ sign, like this: @var
- You can initialize @ variables like

```
mysql> set @x = 1;
mysql> select @x;
```

Another Example (1)

```
mysql> delimiter //
mysql> create procedure prc_test ( )
        begin
          declare var2 int default 1;
          set var2 := var2 + 1;
          set @var2 := @var2 + 1;
          select var2, @var2;
        end //
mysql> delimiter ;
```

Another Example (2)

```
mysql > set @var2 = 1;
mysql > call prc_test();
output:
             var2    @var2
             -----   --------
               2        2

 mysql > call prc_test();
             var2    @var2
             -----   --------
               2        3

mysql > call prc_test();
             var2    @var2
             -----   --------
               2        4
```

- @ variables can carry their values across different procedure calls.

Basic Syntax of Stored Procedures/Functions 存储过程/函数基本语法

- Procedures

  create procedure proc_name ([parameters])

  routine_body

- Functions

  create function func_name ([parameters])

  returns data_type routine_body

  存储函数必须指定返回的类型

- Parameter definition:

  [in | out | inout] param_name data_type（数据类型）

  – “in” is the default

  – Only in parameter can be used for function

Parameter Modes

- IN: Default mode.
  - The calling program must pass an argument to each in parameter.
- OUT
  - The value of an out parameter can be changed inside the stored procedure and its new value is passed back to the calling program.
- INOUT
  - Combination of in & out

Example 3

- Create a procedure to return the number of students in any given department.

```
mysql> delimiter //
mysql> create procedure GetStudentSizeByDept(in
          deptname varchar(50))
       begin
          select count(*) from Students
          where dept_name = deptname;
       end //
mysql> delimiter ;
mysql> call GetStudentSizeByDept('CS');
```

Example 3: Another Way

- Another way to write the procedure in the previous slide.

```
mysql> delimiter //
mysql> create procedure GetStudentSizeByDept1(in
             deptname varchar(50), out num_of_students int)
           begin
             select count(*) into num_of_students
             from Students
             where dept_name = deptname;
           end //
mysql> delimiter ;
mysql> call GetStudentSizeByDept1(‘CS’, @num_of_students);
mysql> select @num_of_students;
```

存储过程可以有返回值，也可以没有返回值

Example 4

- Create a function to return the number of students in any given department.

```
mysql> delimiter //
mysql> create function GetStudentSizeByDept2(in
          deptname varchar(50)) returns int
        begin
          declare num_of_students int;
          select count(*) into num_of_students from Students
          where dept_name = deptname;
          return (num_of_students);
        end //
mysql> delimiter ;
mysql> select GetStudentSizeByDept2( ‘CS’ );
```

存储函数必须指定返回的类型

Example 4: Another Way

- Here is another way to write the function in the previous slide.

```
mysql> delimiter //
mysql> create function GetStudentSizeByDept2(in
          deptname varchar(50)) returns int
       begin
          return (select count(*) from Students
                      where dept_name = deptname);
       end //
mysql> delimiter ;
```

Example 5

```
mysql> delimiter //
mysql> create procedure set_counter(inout count int(4),
           in increment int(4))
         begin
           set count = count + increment;
         end //
mysql> delimiter ;
mysql> set @counter = 1;
mysql> call set_counter(@counter, 1); -- 2
mysql> call set_counter(@counter, 1); -- 3
mysql> call set_counter(@counter, 5); -- 8
mysql> select @counter; -- 8
```

流程控制语句
IF Statement

- Syntax:

```
IF if_expression THEN commands
[ELSEIF elseif_expression THEN commands]
[ELSE commands]
END IF;
```

Example 6

根据银行账户存钱的数量，判断卡的级别

过程/参数定义

变量声明

选择数据

判断

```
mysql> delimiter //
mysql> create procedure GetCustomerLevel(
        in p_customerNumber int(11),
        out p_customerLevel  varchar(10))
  begin
    declare creditlim double;
    select creditlimit into creditlim from customers
    where customerNumber = p_customerNumber;
    if creditlim > 50000 then
      set p_customerLevel = 'PLATINUM';
    elseif (creditlim <= 50000 and creditlim >= 10000) then
      set p_customerLevel = 'GOLD';
    elseif creditlim < 10000 then
      set p_customerLevel = 'SILVER';
    end if;
  end//
mysql> delimiter ;
```

Case Statement

- Syntax:

```
CASE  case_expression
      WHEN when_expression_1 THEN
      commands
      WHEN when_expression_2 THEN
       commands
      …
      ELSE commands
END CASE;
```

Example 7

```
mysql> delimiter //
mysql> create procedure GetCustomerShipping(
           in p_customerNumber int(11),
           out p_shiping varchar(50))
        begin
           declare customerCountry varchar(50);
           select country into customerCountry from customers
           where customerNumber = p_customerNumber;
           CASE customerCountry
               WHEN 'USA' THEN
                    set p_shiping = '2-day Shipping';
               WHEN 'Canada' THEN
                    set p_shiping = '3-day Shipping';
               ELSE
                    set p_shiping = '5-day Shipping';
          END CASE;
        end //
mysql> delimiter ;
```

循环语句Loop Statements

- While Loop

```
WHILE expression DO
      Statements
END WHILE;
```

- Repeat Loop

```
REPEAT
     Statements;
UNTIL expression
END REPEAT;
```

Example 8

```
mysql>  delimiter //
mysql>  drop procedure if exists WhileLoopProc //
mysql>  create procedure WhileLoopProc()
        begin
            declare x int;  declare str  varchar(255);
            set x = 1;  set str = '';
            WHILE x <= 5 DO
                set  str = concat(str, x, ',');  set  x = x + 1;
            END WHILE;
            select str;
        end //
mysql> delimiter ;
mysql> call WhileLoopProc();
```

Output :

```
str
----
1,2,3,4,5,
```

Example 8 (Continued)

- We can replace the WHILE loop in the example in the previous slide by the following REPEAT loop

```
REPEAT
    set  str = concat(str, x, ',');
    set  x = x + 1;
UNTIL x > 5       -- no semicolon here
END REPEAT;
```

Loop, Leave & Iterate

- LEAVE : Equivalent to *break* in JAVA, C/C++, PHP
- ITERATE : Equivalent to *continue*
- They can be used to create another loop mechanism.

Example 9

```
mysql> delimiter //
mysql> create procedure LOOPLoopProc()
mysql> begin
              declare  x  int;  declare  str  VARCHAR(255);
              set x = 1;   set str =  '';
              loop_label:  LOOP
                  IF  x > 10 THEN
                    LEAVE  loop_label;
                  END  IF;
                  set  x = x + 1;
                  IF  (x mod 2) THEN
                    ITERATE  loop_label;
                  ELSE
                    set  str = concat(str, x, ',');
                  END  IF;
             END LOOP;
             select str;
               end //
mysql> delimiter ;
Output :
   str
   ----
   2,4,6,8,10,
```

MySQL Cursor (游标)

- SQL语句查询结果是多条记录，如何存储这些结果？
- 将一条条记录存储起来，使用Cursor (游标)
- 游标是系统为用户开设的一个数据缓冲区

MySQL Cursor (2)

- Cursor is used to iterate through a result set returned by a select statement one row at a time.
- MySQL Cursors are:
  - Read Only 只读: Can not update data in the underlying table through cursor.
  - Non-Scrollable 无法滚动: Can only go through rows in the result set in forward order. You can not travel backward or skip rows.

MySQL Cursor (3)

- Syntax for defining a cursor:

  declare cursor_name cursor for select_statement;

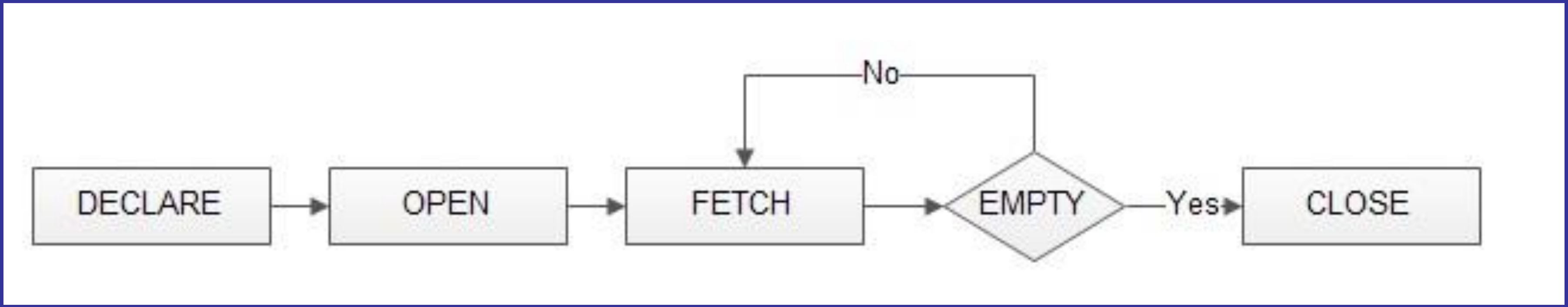
Example:

```
    declare c1 cursor for
            select cid, cname, city from customers;
```

- The query defining the cursor is not executed at the declaration time.
- Cursor declaration comes after variable declaration.
- Question: What's the difference between cursor and a view?

MySQL Cursor (4)

- Statements for working with cursors:
  - open cursor_name;
  - fetch cursor_name
    into record_or_variable_list;
  - close cursor_name;
- Open causes the query defining the cursor to be executed.



MySQL Cursor (5)

- A cursor always points to the prior row. Each fetch statement will first advance the cursor position by 1 and then retrieve the values in the pointed row.
- A cursor can be re-opened after it is closed. Re-opened cursor points to the beginning of the table.
- A cursor is not needed if you are sure that the result can have at most one row.
  - In this case, you can use “select … into …”

游标充当**指针**的作用。尽管游标能遍历结果中的所有行，但他一次只指向一行。

MySQL Cursor (6)

- When working with MYSQL cursor, also need to declare a NOT FOUND Handler to handle the situation when the cursor could not find any row when the next FETCH statement is executed.
- Declare a NOT FOUND Handler

```
declare continue handler for not found set finished
    = 1;
```

Example 10

```
mysql> delimiter //
mysql> create procedure build_email_list (inout email_list  varchar(4000))
      begin
          declare v_finished int default 0;
          declare v_email varchar(100) default '';
          -- declare cursor for employee email
          declare email_cursor cursor for select email from employees;
          -- declare NOT FOUND handler
          declare continue handler  for not found set v_finished = 1;
          open email_cursor;
          get_email: loop
              fetch email_cursor into v_email;
              if v_finished = 1 then
                  leave get_email;  -- exit the loop
              end if;
              -- build email list with “;” as the delimiter
              set email_list = concat(v_email, ';', email_list);
          end loop get_email;
          close email_cursor;
      end //
```

Example 10 (Continued)

```
 mysql> delimiter ;
Test:
mysql> set @email_list = '';
mysql> call build_email_list(@email_list);
mysql> select @email_list;
Output :
       @email_list
       ------------------
       a@gmail.com;b@gmail.com;c@gmail.com; (List of all email id separated by ;)
```

Stored Procedure/Function Information (1) 查看存储程序

- To find information about a stored procedure or function, use the following statement:

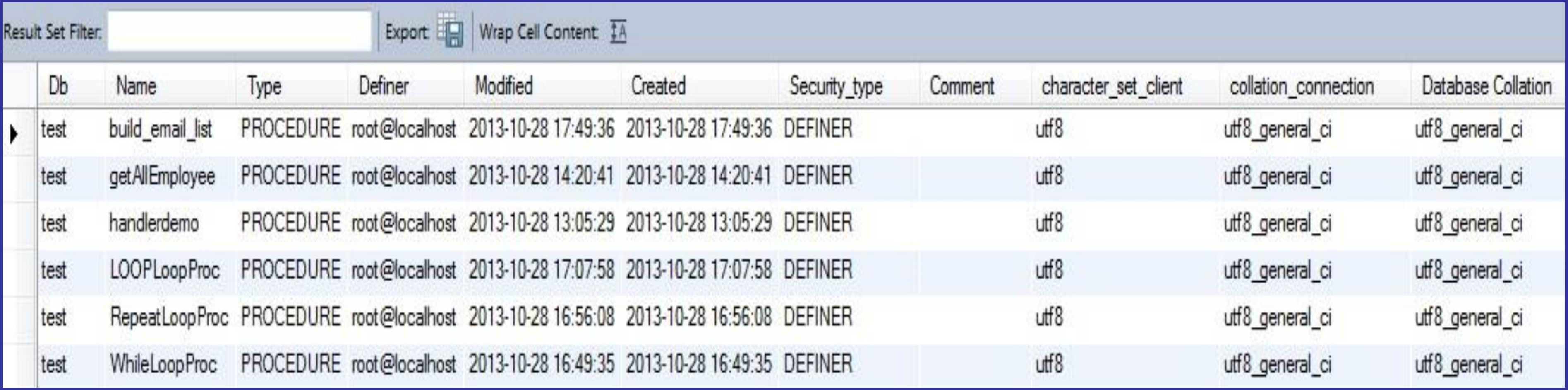
Syntax:

show procedure | function status [where expression];

- Examples:

```
mysql> show procedure status;
mysql> show function status where name like '%product%‘;
```

Sample Output of “show procedure status” in Workbench



Db	Name	Type	Definer	Modified	Created	Security_type	Comment	character_set_client	collation_connection	Database Collation
test	build_email_list	PROCEDURE	root@localhost	2013-10-28 17:49:36	2013-10-28 17:49:36	DEFINER		utf8	utf8_general_ci	utf8_general_ci
test	getAllEmployee	PROCEDURE	root@localhost	2013-10-28 14:20:41	2013-10-28 14:20:41	DEFINER		utf8	utf8_general_ci	utf8_general_ci
test	handlerdemo	PROCEDURE	root@localhost	2013-10-28 13:05:29	2013-10-28 13:05:29	DEFINER		utf8	utf8_general_ci	utf8_general_ci
test	LOOPLoopProc	PROCEDURE	root@localhost	2013-10-28 17:07:58	2013-10-28 17:07:58	DEFINER		utf8	utf8_general_ci	utf8_general_ci
test	RepeatLoopProc	PROCEDURE	root@localhost	2013-10-28 16:56:08	2013-10-28 16:56:08	DEFINER		utf8	utf8_general_ci	utf8_general_ci
test	WhileLoopProc	PROCEDURE	root@localhost	2013-10-28 16:49:35	2013-10-28 16:49:35	DEFINER		utf8	utf8_general_ci	utf8_general_ci

Stored Procedure/Function Information (2)

- To see a stored procedure's or function's source code, use

  show create procedure | function stored_procedure_name;

Examples:

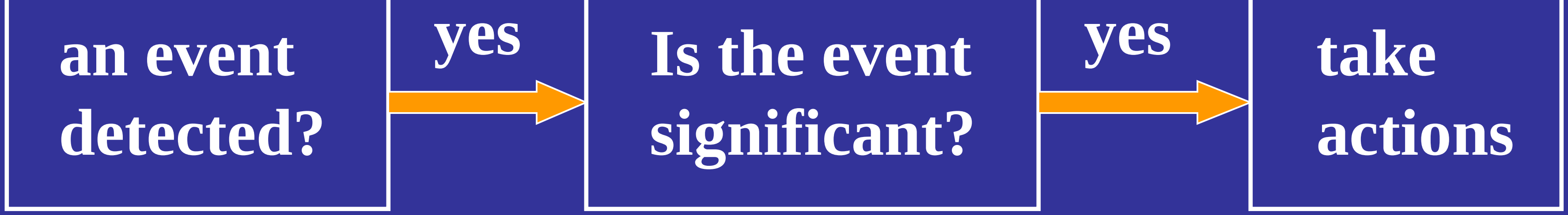
```
mysql> show create procedure build_email_list;
mysql> show create function GetStudentSizeByDept2;
```

Trigger (触发器)

- 触发器(Trigger)是用户定义在关系表上的一类由事件驱动的特殊过程
  - 触发器保存在数据库服务器中
  - 任何用户对表的增、删、改操作均由服务器自动激活相应的触发器
  - 触发器可以实施更为复杂的检查和操作，具有更精细和更强大的数据控制能力

Trigger (2)

ECA Model:



insert delete update	need to satisfy additional condition	execute MySQL block
Event	Condition	Action

- MySQL currently only supports the event-action part.

Trigger (3)

- Syntax for creating trigger:

```
      create trigger trigger_name  <触发器名>
       { before | after }
       { insert | update | delete }  <触发事件> ON <表名>
       on table_name
       for each row
       <trigger body>
```

- Syntax for deleting trigger:

```
      drop trigger [if exists] trigger_name
```

Trigger (4)

- Trigger event examples
  - update on employees
  - insert on employees
  - delete on employees
- trigger timing
  - before: execute the trigger body before executing the triggering statement
  - after: execute the trigger body after executing the triggering statement

Trigger (5)

- row trigger
  - "for each row" is specified
  - Execute the trigger body once for each row that is affected by the event
- statement trigger
  - "for each row" is not specified
  - fire the trigger once for the entire trigger event
  - Statement trigger is not supported by MySQL.

Trigger (6)

- Within the trigger body, the OLD and NEW keywords enable you to access columns in the rows affected by a trigger.
  - In an INSERT trigger, only NEW.col_name can be used; there is no old row.
  - In a DELETE trigger, only OLD.col_name can be used; there is no new row.
  - In an UPDATE trigger, you can use OLD.col_name to refer to the columns of a row before it is updated and NEW.col_name to refer to the columns of the row after it is updated.

●用户都可以在过程体中使用NEW和OLD引用事件之后的新值和事件之前的旧值

Trigger (7)

- A column named with OLD is read only.
- In a BEFORE trigger, you can also change its value with “set NEW.col_name = value”.
  - This makes it possible to use a trigger to modify the values to be inserted into a new row or used to update a row.
  - Such a set statement has no effect in an AFTER trigger because the row change has already occurred.

Trigger Example 1

- We want to insert tuples into table Sale(Percent, Sale_Date), where Percent must be between 5 and 80. Any attempt to insert a Percent below 5 will be replaced by 5 and that above 80 by 80. The Sale_Date value will be the current date.

```
mysql> create trigger before_insert_sale
          before insert on Sale for each row
          begin
             set new.sale_date = curdate();
             if new.percent < 5 then
                set new.percent = 5;
             elseif new.percent > 80 then
                set new.percent = 80;
             end if;
         end //
```

Trigger Example 2

- Use trigger for monitoring changes

Example: Add a log entry each time the price of a product is changed.

log table for product:

```
create table product_log
  (pid varchar(4),
    username varchar(20),
    update_time datetime,
    old_price decimal(6, 2),
    new_price decimal(6, 2));
```

Sample Example 2 (Continued)

```
mysql> create trigger update_product_price
            after update on products
            for each row
            begin
                if old.price <> new.price then
                insert into product_log values
                    (old.pid, user(), current_timestamp,
                     old.price, new.price);
           end //
```

Trigger Example 3

- Use trigger to enforce integrity constraints

Example: If a student is removed, delete all enrollments by the student.

```
mysql> create trigger stud_enroll
       after delete on students
       for each row
       begin
         delete from enrollments where sid = old.sid;
       end //
```

- If there is just one statement in the trigger body, “begin … end” is optional.

Trigger Example 4

- Use trigger to maintain data consistency

Example: If an order is made for a product with certain quantity, then the quantity on hand of the product should be reduced accordingly.

```
mysql> create trigger prod_qoh_on_order
       after insert on orders for each row
       begin
            update products
            set quantity = quantity – new.qty
            where pid = new.pid;
       end //
```

Trigger Restrictions

- Triggers can not:
  - Modify a table being used by the DML without NEW or OLD aliases.
  - Use SELECT without INTO variable_name.
  - Use SHOW commands.
  - Use ALTER VIEW.
  - Use RETURN in stored programs.
  - Use statements that explicitly or implicitly begin or end a transaction, such as “start transaction”, “commit”, or ”rollback”.

Handling Trigger Errors in MySQL

- If a BEFORE trigger fails, the operation on the corresponding row is not performed.
- A BEFORE trigger is activated by the *attempt* to insert or modify the row, regardless of whether the attempt subsequently succeeds.
- An AFTER trigger is executed only if any BEFORE triggers and the row operation execute successfully.
- An error during either a BEFORE or AFTER trigger results in failure of the entire statement that caused trigger invocation.
- For transactional tables, failure of a statement causes rollback of all changes performed by the statement.
- For non-transactional tables, rollback cannot be done, any changes performed prior to the error point remain in effect.

Events事件

- In MySQL, an event is a stored program that executes according to a schedule. 根据时间设置启动一个事件
- Basic create event syntax:

  create event *event_name*

  on schedule *schedule_spec*

  do *event_body*;

  – *schedule_spec*: at *timestamp* [+ interval *interval_spec*] | every *interval* [starts *timestamp* [+ interval *interval_spec*]] [ends *timestamp* [+ interval *interval_spec*]]
  – *interval_spec*: *quantity* {year | quarter | month | day | hour | minute | week | second | year_month | day_hour | day_minute | day_second | hour_minute | hour_second | minute_second}

Event Example 1

- Suppose we have table MySchedule(event_name, event_time, event_place).
- Create an event that will add a tuple about a meeting into MySchedule 3 hours from now:

```
mysql> create event my_schedule
            on schedule at current_timestamp
                        + interval 3 hours
            do insert into MySchedule values (‘faculty
       meeting’, ‘2013-11-14 16:30:00’, ‘G11’);
```

Event Example 2

- Create an event that removes activity tuples that have expired from MySchedule every day:

```
mysql> create event my_schedule
            on schedule every 1 day
           do delete from MySchedule
                where event_time < current_timestamp;
```

  - The event runs immediately after its creation and then once each day.