# ****触发器原理：****

**http://www.cnblogs.com/nerxious/archive/2013/01/04/2844354.html**

**http://www.cnblogs.com/diegodu/p/5474019.html**      触发器与存储过程非常相似，触发器也是SQL语句集，

两者唯一的区别是触发器不能用EXECUTE语句调用，而是在用户执行Transact-SQL语句时自动触发（激活）执行。

触发器是在一个修改了指定表中的数据时执行的存储过程。通常通过创建触发器来强制实现不同表中的逻辑相关数据的引用完整性和一致性。由于用户不能绕过触发器，所以可以用它来强制实施复杂的业务规则，以确保数据的完整性。触发器不同于存储过程，触发器主要是通过事件执行触发而被执行的，而存储过程可以通过存储过程名称名字而直接调用。当对某一表进行诸如UPDATE、INSERT、DELETE这些操作时，SQLSERVER就会自动执行触发器所定义的SQL语句，从而确保对数据的处理必须符合这些SQL语句所定义的规则。

## 触发器的作用：

触发器的主要作用是其能够实现由主键和外键所不能保证的复杂的参照完整性和数据的一致性。它能够对数据库中的相关表进行级联修改，强制比CHECK约束更复杂的数据完整性，并自定义操作消息，维护非规范化数据以及比较数据修改前后的状态。与CHECK约束不同，触发器可以引用其它表中的列。在下列情况下使用触发器实现复杂的引用完整性；强制数据间的完整性。创建多行触发器，当插入，更新、删除多行数据时，必须编写一个处理多行数据的触发器。执行级联更新或级联删除这样的动作。级联修改数据库中所有相关表。撤销或者回滚违反引用完整性的操作，防止非法修改数据。

# **触发器与存储过程的区别：**

         触发器与存储过程的主要区别在于触发器的运行方式。存储过程必须有用户、应用程序或者触发器来显示的调用并执行，而触发器是当特定时间出现的时候，自动执行或者激活的，与连接用数据库中的用户、或者应用程序无关。当一行被插入、更新或者删除时触发器才执行，同时还取决于触发器是怎样创建的，当UPDATE发生时使用一个更新触发器，当INSERT发生时使用一个插入触发器，当DELETE发生时使用一个删除触发器。

# 1.创建简单触发器

CREATE TRIGGER 触发器名 BEFORE|AFTER 触发事件

ON 表名 FOR EACH ROW 执行语句

其中，触发器名参数指要创建的触发器的名字

BEFORE和AFTER参数指定了触发执行的时间，在事件之前或是之后

FOR EACH ROW表示任何一条记录上的操作满足触发事件都会触发该触发器

## 1.创建单条语句

mysql> CREATE TRIGGER trig1 AFTER INSERT

-> ON work FOR EACH ROW

-> INSERT INTO time VALUES(NOW());

Query OK, **0** rows affected (**0.09** sec)

## 2.创建多条

BEGIN与END之间的执行语句列表参数表示需要执行的多个语句，不同语句用分号隔开

CREATE TRIGGER 触发器名 BEFORE|AFTER 触发事件

ON 表名 FOR EACH ROW

BEGIN

执行语句列表

END

## 3.MySQL解决冲突

BEGIN与END之间的执行语句列表参数表示需要执行的多个语句，不同语句用分号隔开

**tips：**一般情况下，mysql默认是以 ; 作为结束执行语句，与触发器中需要的分行起冲突

　　   为解决此问题可用DELIMITER，如：DELIMITER ||，可以将结束符号变成||

　　   当触发器创建完成后，可以用DELIMITER ;来将结束符号变成;

上面的语句中，开头将结束符号定义为||，中间定义一个触发器，一旦有满足条件的删除操作

就会执行BEGIN和END中的语句，接着使用||结束

最后使用DELIMITER ; 将结束符号还原

mysql> DELIMITER ||

mysql> CREATE TRIGGER trig2 BEFORE DELETE

-> ON work FOR EACH ROW

-> BEGIN

-> INSERT INTO time VALUES(NOW());

-> INSERT INTO time VALUES(NOW());

-> END

-> ||

Query OK, **0** rows affected (**0.06** sec)

mysql> DELIMITER

## 3.1简单真实测试

1. **MySQL> create table userinfo(userid int,username varchar(10),userbirthday date);**
2. **MySQL> create table userinfolog(logtime datetime,loginfo varchar(100));**
3. MySQL> create trigger beforeinsertuserinfo
4. -> before insert on userinfo
5. -> for each row begin
6. -> insert into userinfolog values(now(),CONCAT(new.userid,new.username));
7. -> end;

## 4.查看触发器

### 1.SHOW TRIGGERS语句查看触发器信息结果会显示所有触发器的基本信息

tips：**SHOW TRIGGERS语句无法查询指定的触发器**

mysql> SHOW TRIGGERS\G;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **1**. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Trigger: trig1

Event: INSERT

Table: work

Statement: INSERT INTO time VALUES(NOW())

Timing: AFTER

Created: NULL

sql\_mode:

Definer: root@localhost

character\_set\_client: utf8

collation\_connection: utf8\_general\_ci

Database Collation: latin1\_swedish\_ci

### 2. 在triggers表中查看触发器信息

mysql> SELECT \* FROM information\_schema.triggers\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **1**. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TRIGGER\_CATALOG: def

TRIGGER\_SCHEMA: person

TRIGGER\_NAME: trig1

EVENT\_MANIPULATION: INSERT

EVENT\_OBJECT\_CATALOG: def

EVENT\_OBJECT\_SCHEMA: person

EVENT\_OBJECT\_TABLE: work

ACTION\_ORDER: **0**

ACTION\_CONDITION: NULL

ACTION\_STATEMENT: INSERT INTO time VALUES(NOW())

### 2.1该方法可以查询制定触发器的详细信息

**mysql> SELECT \* FROM information\_schema.triggers WHERE TRIGGER\_NAME='trig1'\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* **1**. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TRIGGER\_CATALOG: def

TRIGGER\_SCHEMA: person

TRIGGER\_NAME: trig1

EVENT\_MANIPULATION: INSERT

EVENT\_OBJECT\_CATALOG: def

EVENT\_OBJECT\_SCHEMA: person

EVENT\_OBJECT\_TABLE: work

### 3.解释

所有触发器信息都存储在information\_schema数据库下的triggers表中

　　   可以使用SELECT语句查询，如果触发器信息过多，最好通过TRIGGER\_NAME字段指定查询

## 5.删除触发器

**tips：**如果不需要某个触发器时一定要将这个触发器删除，以免造成意外操作

mysql> DROP TRIGGER trig1;

Query OK, **0** rows affected (**0.04** sec)

# 2.存储过程

## 1.简单创建一个存储过程

1. MySQL> delimiter //
2. MySQL> create procedure spinsertuserinfo(
3. -> puserid int,pusername varchar(10)
4. -> ,puserbirthday date
5. -> )
6. -> begin
7. -> insert into userinfo values(puserid,pusername,puserbirthday);
8. -> end;
9. -> //
10. MySQL> show procedure status like 'spinsertuserinfo';
11. MySQL> call spinsertuserinfo(1,'zhangsan',current\_date);
12. MySQL> select \* from userinfo;

# 3.自定义函数

1. MySQL> update userinfo
2. -> set userbirthday='2000.01.01'
3. -> where userid='1';
4. MySQL> drop function if exists fngetage;
5. MySQL> delimiter //
6. MySQL> create function fngetage(pbirthday date)
7. -> returns integer
8. -> begin
9. -> return year(now()) - year(pbirthday);
10. -> end
11. -> //

# 4.视图

## 1.解释

视图是从一个或多个表中导出来的表，是一种虚拟存在的表。

视图就像一个窗口，通过这个窗口可以看到系统专门提供的数据。

这样，用户可以不用看到整个数据库中的数据，而之关心对自己有用的数据。

数据库中只存放了视图的定义，而没有存放视图中的数据，这些数据存放在原来的表中。

使用视图查询数据时，数据库系统会从原来的表中取出对应的数据。

视图中的数据依赖于原来表中的数据，一旦表中数据发生改变，显示在视图中的数据也会发生改变。

## 2.视图的作用

1.使操作简单化，可以对经常使用的查询定义一个视图，使用户不必为同样的查询操作指定条件

2.增加数据的安全性，通过视图，用户只能查询和修改指定的数据。

3.提高表的逻辑独立性，视图可以屏蔽原有表结构变化带来的影响。

## 3. 在单表上创建视图

此处work\_view为视图名，后面括号内的参数代表视图中的列

AS表示将后面SELECT 语句中的查询结果赋给前面的视图中

mysql> CREATE VIEW work\_view(ID,NAME,ADDR) AS SELECT id,name,address FROM work;

Query OK, **0** rows affected (**0.05** sec)

## 4. 在多表上创建视图

在多表中创建视图需要两表有指定联系，如上面的work.id=salary.id

mysql> CREATE ALGORITHM=MERGE VIEW work\_view2(ID,NAME,SALARY)

-> AS SELECT work.id,name,salary FROM work,salary

-> WHERE work.id=salary.id

-> WITH LOCAL CHECK OPTION;

Query OK, **0** rows affected (**0.02** sec)

# 5。细节

## 1. declare 定义数据类型

declare v\_number number(10);  
 v\_char varchar2(20);  
这个一般是用在函数和存储过程中的.

## 2. loop 循环

create or replace procedure pro\_test\_loop is  
i number;  
begin  
i:=0;  
loop  
  i:=i+1;  
  dbms\_output.put\_line(i);  
  if i>5 then  
    exit;  
  end if;  
end loop;  
end pro\_test\_loop;