第25章：使用触发器

1. **什么是触发器**

触发器是MySQL响应某个表发生更改时而自动执行的一条MySQL语句（或于BEGIN和END语句之间的一组语句）。（更改主要是三个更新表的命令:INSERT\UPDATE\DELETE。）

触发器是一种特殊的定义在表上的MYSQL语句的集合（存储过程），它在插入，删除或改动特定表中的数据时触发运行，执行一条或一组语句，它比数据库本身标准的功能有更精细和更复杂的数据控制能力。

1. **触发器的作用**
2. **安全性**

能够基于数据库的值使用户具有操作数据库的某种权利。

  # 能够基于时间限制用户的操作，比如不同意下班后和节假日改动数据库数据。

  # 能够基于数据库中的数据限制用户的操作，比如不同意股票的价格的升幅一次超过10%。

1. **审计跟踪**

能够跟踪用户对数据库的操作。

  # 审计用户操作数据库的语句。

  # 把用户对数据库的更新写入审计表。

**3.实现复杂的数据完整性规则**

  # 实现非标准的数据完整性检查和约束。触发器可产生比规则更为复杂的限制。与规则不同，触发器能够引用列或数据库对象。

  # 提供可变的缺省值。

**4.实现复杂的非标准的数据库相关完整性规则**

触发器能够对数据库中相关的表进行连环更新。比如，在auths表author\_code列上的删除触发器可导致对应删除在其他表中的与之匹配的行。

  # 在改动或删除时级联改动或删除其他表中的与之匹配的行。

  # 在改动或删除时把其他表中的与之匹配的行设成NULL值。

  # 在改动或删除时把其他表中的与之匹配的行级联设成缺省值。

  # 触发器可以拒绝或回退那些破坏相关完整性的变化，取消试图进行数据更新的事务。当插入一个与其主健不匹配的外部键时，这样的触发器会起作用。比如，可以在books.author\_code 列上生成一个插入触发器，假设新值与auths.author\_code列中的某值不匹配时，插入被回退。

**5.同步实时地复制表中的数据**

**6.自己主动计算数据值（统计数据，需要计算并存储的数据）**

假设数据的值达到了一定的要求，则进行特定的处理。比如，假设公司的帐号上的资金低于5万元则马上给財务人员发送警告数据。

1. **创建触发器**

**1）CREATE TRIGGER语法**

CREATE TRIGGER trigger\_name trigger\_time trigger\_event

ON tbl\_name FOR EACH ROW trigger\_stmt

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。

（1）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TEMPORARY表或视图关联起来。**

（2）trigger\_time是触发程序的动作时间。它能够是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。

（3）trigger\_event指明了激活触发程序的语句的类型。trigger\_event能够是下述值之中的一个：

INSERT：将新行插入表时激活触发程序，比如，通过INSERT、LOAD DATA和REPLACE语句。

UPDATE：更改某一行时激活触发程序，比如，通过UPDATE语句。

DELETE：从表中删除某一行时激活触发程序，比如，通过DELETE和REPLACE语句。

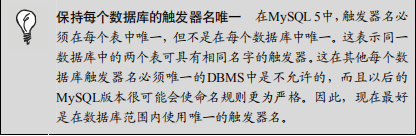
**对于具有同样触发程序动作时间和事件的给定表，不能有两个触发程序。比如，对于某一表，不能有两个BEFORE UPDATE触发程序。但能够有1个BEFORE UPDATE触发程序和1个BEFORE INSERT触发程序，或1个BEFORE UPDATE触发程序和1个AFTER UPDATE触发程序。**

（4）trigger\_stmt是当触发程序激活时运行的语句。假设你打算运行多个语句，可使用BEGIN ... END复合语句结构。**这样，就能使用存储子程序中同意的同样语句，但不能直接通过CALL来调用存储过程。**

**2）创建触发器的要点**

1. 唯一的触发器名称（保持唯一，仅支持表）；
2. 触发器关联的表（每个表最多支持6个触发器）；
3. 触发器应该响应的活动（DELETE\INSERT\UPDATE）;
4. 触发器何时执行（处理之前或之后，单一触发器不能与多个事件或多个表关联）
5. 单一触发器不能与多个事件或多个表关联，如果你需要一个对INSERT和UPDATE操作执行的触发器，则应该定义两个触发器。

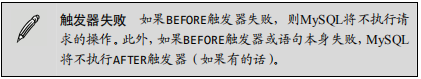
**命名规则：**



**适用对象：**

2~6YH$%1C`CLNVJHA{Z98(V

**异常出错：**



**3）创建触发器举例**

Create trigger newproduct after insert on products

For each row select ‘product added’

1. **删除触发器（不能更新或覆盖，只能创建-删除-再创建）**

**1）DROP TRIGGER语法**

DROP TRIGGER [schema\_name.]trigger\_name

舍弃触发程序。方案名称（schema\_name）是可选的。假设省略了schema（方案），将从当前方案中舍弃触发程序。

**从MySQL 5.0.10之前的MySQL版本号升级到5.0.10或更高版本号时（包含全部的MySQL 5.1版本），必须在升级之前舍弃全部的触发程序，并在随后又一次创建它们，否则，在升级之后DROPTRIGGER不工作。DROP TRIGGER语句须要SUPER权限。**

**触发器不能更新或覆盖。为了修改一个触发器，必须先删除它，然后再重新创建。**

**2）删除触发器举例：**

DROP TRIGGER NEWPRODUCT;

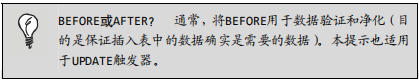
1. **使用触发器**

**1）INSERT 触发器**

**\*在INSERT触发器代码内，可引用一个名为NEW的虚拟表，访问被插入的行；**

**\*在BEFORE INSERT 触发器中，NEW中的值也可被更新（允许更改被插入的值）；**

**\*对于AUTO\_INCREMENT列，NEW在INSERT执行前包含0，在INSERT执行之后包含新的自动生成值。**



**应用举例：**

**创建一个事务日志表，以记录对表的更新提示信息：**

CREATE TABLE orders\_log

(

change\_id int NOT NULL AUTO\_INCREMENT,

changed\_on datetime NOT NULL ,

change\_type char(1) NOT NULL ,

order\_num int NOT NULL ,

PRIMARY KEY (change\_id)

) ENGINE=INNODB;

CREATE TRIGGER neworder AFTER INSERT ON orders

FOR EACH ROW

BEGIN

INSERT INTO orders\_log(changed\_on, change\_type, order\_num)

VALUES(Now(),'A', NEW.order\_num);

END;

**测试：**

INSERT INTO orders(order\_date, cust\_id)

VALUES(Now(), 10001);

SELECT \* FROM orders\_log;

+---------------+-------------------------------+------------------+--------------------+

| change\_id | changed\_on | change\_type | order\_num |

+---------------+-------------------------------+------------------+--------------------+

| 1 | 2011-04-12 10:49:59 | A | 20010 |

+--------------+--------------------------------+------------------+--------------------+

**样例库举例：**

CREATE TRIGGER neworder AFTER INSERT ON orders

FOR EACH ROW select NEW.order\_num;

**2）DELETE触发器**

**\*在DELETE触发器代码内，可引用一个名为OLD的虚拟表，访问被删除的行；**

**\*在DELETE INSERT 触发器中，OLD中的值全都是只读的，不能更新；**

CREATE TRIGGER deleteorder BEFORE DELETE ON orders

FOR EACH ROW

BEGIN

INSERT INTO orders\_log(changed\_on, change\_type, order\_num)

VALUES(Now(),'D', OLD.order\_num);

END;

**应用举例：（多语句触发器）**

CREATE TABLE archive\_orders

(

order\_num int NOT NULL,

order\_date datetime NOT NULL ,

cust\_id int NOT NULL

) ENGINE=InnoDB;

CREATE TRIGGER deleteorder BEFORE DELETE ON orders

FOR EACH ROW

BEGIN

INSERT INTO orders\_log(changed\_on, change\_type, order\_num)

VALUES(Now(),'D', OLD.order\_num);

INSERT INTO archive\_ orders(order\_num, order\_date, cust\_id)

VALUES(OLD.order\_num, OLD.order\_date, OLD.cust\_id);

END;

测试：

INSERT INTO orders(order\_date, cust\_id) VALUES(Now(), 10001);

SELECT \* FROM orders\_log;

SELECT \* FROM archive\_ orders;

**3）UPDATE触发器**

**\*在UPDATE触发器代码内，可引用一个名为OLD的虚拟表访问以前的值，引用一个名为NEW的虚拟表访问新更新后的值；**

**\*在BEFORE UPDATE触发器中，NEW中的值可能也被更新（允许更改将要用于UPDATE语句中的值）；**

**\* OLD中的值全都是只读的，不能更新；**

**应用举例：**

**1)**

**CREATE TRIGGER updatevendor BEFORE UPDATE ON vendors**

**FOR EACH ROW SET NEW.vend\_state = Upper(NEW.vend\_state);**

**2)**

**CREATE TRIGGER updateORDER BEFORE UPDATE ON ORDERS**

**FOR EACH ROW**

**BEGIN**

**INSERT INTO orders\_log(changed\_on, change\_type, order\_num)**

**VALUES(Now(),'U', OLD.order\_num);**

**SET NEW.ORDER\_DATE = NOW();**

**END;**

**测试：**

UPDATE orders SET cust\_id=’10006’ WHERE cust\_id=’10005’;

SELECT \* FROM orders\_log;

SELECT \* FROM orders;

**附样例库关系图：**

