

触发器实验

16337113

劳马东

计算机科学与技术（超算方向）

一、 实验目的

掌握数据库触发器的设计和使用方法。

二、 实验内容和要求

定义 BEFORE 触发器和 AFTER 触发器。能够理解不同类型触发器的作用和执行原理，验证触发器的有效性。

三、 实验环境

系统	Windows 10
SQL	MySQL 8.0
工具	MySQL workbench

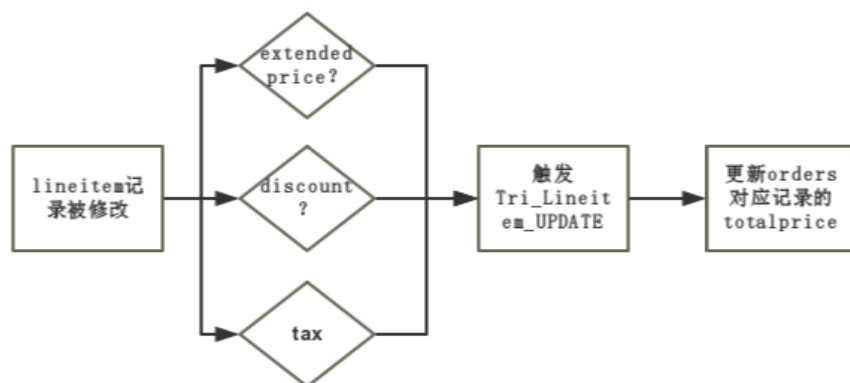
四、 实验过程

1. AFTER 触发器

1) 在 lineitem 表上定义一个 UPDATE 触发器

当修改订单明细（即修改订单明细价格 extendedprice、折扣 discount、税率 tax）时，自动修改订单 order 的 totalprice，以保持数据一致性。

其中, $totalprice = totalprice + extendedprice + (1 - discount) * (1 + tax)$ 。



```

DELIMITER //
CREATE TRIGGER TRI_Lineitem_Price_UPDATE
AFTER UPDATE ON lineitem
FOR EACH ROW
BEGIN
    DECLARE L_valuediff REAL;

    SET L_valuediff = NEW.extendedprice * (1 - NEW.discount) * (1
+ NEW.tax) - OLD.extendedprice * (1 - OLD.discount) * (1 +
OLD.tax);
    UPDATE orders SET totalprice = totalprice + L_valuediff
        WHERE orderkey = NEW.orderkey;
END//

```

MySQL 语句默认以分号结束，它告诉 MySQL 解释器一段命令已经结束，可以开始执行。但有时候，不希望 MySQL 这么做，例如在函数和触发器定义中通常含有较多的语句，而这些语句以分号结束。因此，需要默认把结束符换成其他符号（如 //或\$\$），MySQL 提供了 DELIMITER 语句来实现这个目的。

上面的代码首先计算新的记录（NEW）的 totalprice 和旧记录（OLD）的 totalprice 的差值，然后更新 orders 表对应 orderkey 的 totalprice。

2) 在 lineitem 表上定义一个 INSERT 触发器

当增加一项订单明细时，自动修改订单 orders 的 totalprice，以保持数据一致性。

INSERT 触发器相当于 extendedprice、discount、tax 均改变的情况下的 UPDATE 触发器，所不同的是，**INSERT 触发器中没有 OLD 变量**，因为插入一条新的记录之前不存在一条与其对应的旧记录。

```

CREATE TRIGGER TRI_Lineitem_Price_INSERT
AFTER INSERT ON lineitem
FOR EACH ROW
BEGIN
    DECLARE L_valuediff REAL;

    SET L_valuediff = NEW.extendedprice * (1 - NEW.discount) * (1
+ NEW.tax);
    UPDATE orders SET totalprice = totalprice + L_valuediff
        WHERE orderkey = NEW.orderkey;
END//

```

3) 在 lineitem 表上定义一个 DELETE 触发器

当删除一项订单明细时，自动修改订单 orders 的 totalprice，保持数据一致性。

DELETE 触发器相当于 extendedprice、discount、tax 全部变成 0 情况下的 UPDATE 触发器，所不同的是，**DELETE 触发器中没有 NEW 变量**，因为删除操作实际上不是更新记录，而是去掉原来的记录，因此也只能引用原来的旧记录 OLD。

```

CREATE TRIGGER TRI_Lineitem_Price_INSERT
AFTER INSERT ON lineitem

```




```

FOR EACH ROW
BEGIN
    DECLARE L_valuediff REAL;

    SET L_valuediff = -OLD.extendedprice * (1 - OLD.discount) *
(1 + OLD.tax);
    UPDATE orders SET totalprice = totalprice + L_valuediff
        WHERE orderkey = OLD.orderkey;
END//

```

三个触发器创建成功后，在 MySQL workbench 中查看 lineitem 表的 Triggers，就能发现多出了三个触发器，分别为 TRI_Lineitem_Price_INSERT、TRI_Lineitem_Price_UPDATE 和 TRI_Lineitem_Price_DELETE，分别对应 INSERT、UPDATE 和 DELETE 事件，都属于 AFTER 类型。

Name	Event	Timing	Created
 TRI_Lineitem_Price_INSERT	INSERT	AFTER	2018-11-14 23:1...
 TRI_Lineitem_Price_UPDATE	UPDATE	AFTER	2018-11-14 23:1...
 TRI_Lineitem_Price_DELETE	DELETE	AFTER	2018-11-14 23:2...

4) 验证触发器

查看 1 号订单的含税折扣总价 totalprice。

```
SELECT totalprice FROM orders WHERE orderkey=1;
```

查询结果表明 orderkey 为 1 的订单目前的总价为 172799.49。

	totalprice
▶	172799.49

A. TRI_LineItem_Price_UPDATE

要验证 UPDATE 触发器，就需要修改 lineitem 表中的某个记录的 extendedprice、discount 或 tax。下面的代码修改了订单号为 1 的第一个明细项的税率，增加 0.5%，修改其他属性同理，观察 1 号订单的总价变化。

```

UPDATE lineitem SET tax = tax + 0.005
WHERE orderkey = 1 AND linenum = 1;

```

查询结果表明税率增加后，1 号订单的总价提高到 172918.09968，符合 UPDATE 触发器的定义，说明 UPDATE 触发器起作用了。

	totalprice
▶	172918.09967999998

B. TRI_LineItem_Price_INSERT

在 lineitem 表中插入一条新的记录，由于 orderkey 和 linenum 是主键，新记录不同这两个属性不能与原有记录都相同。下面的代码在 1 号订单的第一个明细的基础上，将 linenum 修改为 100，即第 100 个明细。

```

INSERT INTO lineitem VALUES(1, 1552, 93, 100, 17, 24710.35, 0.04,
0.025, 'N', 'O', '1996-03-13', '1996-02-12', '1996-03-22',
'DELIVER IN PERSON', 'TRUCK', 'egular courts above the');

```

查询结果表明，1 号订单的总价再次提高到 197114.4744，符合 INSERT 触发器定义，说明 INSERT 触发器起作用了。

	totalprice
▶	197114.47439999998

C. TRI_LineItem_Price_DELETE

删除 1 号订单的第一个明细，查看它的总价变化。

```
DELETE FROM lineitem WHERE orderkey=1 AND linenumber=1;
```

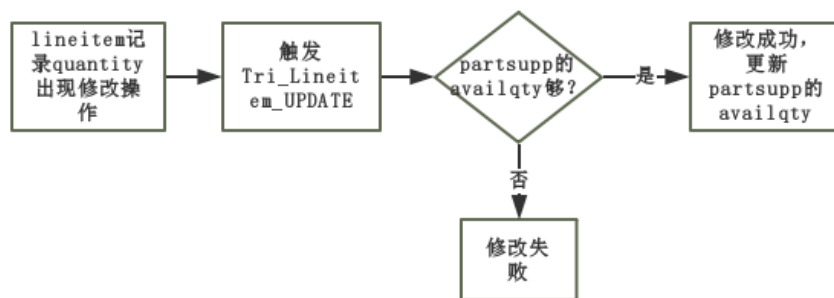
查询结果表明，1 号订单的 totalprice 从 197114.4744 减少到 148603.11528，符合 DELETE 触发器定义，说明 DELETE 触发器起作用。

	totalprice
▶	148603.11528

2. BEFORE 触发器

1) 在 lineitem 表上定义一个 UPDATE 触发器

当修改订单明细中的数量 (quantity) 时，先检查供应表 partsupp 中的可用数量 availqty 是否足够。



```

DELIMITER //
CREATE TRIGGER TRI_Lineitem_Quantity_UPDATE
BEFORE UPDATE ON lineitem
FOR EACH ROW
BEGIN
    DECLARE L_valuediff, L_availqty INTEGER;
    SET L_valuediff = NEW.quantity - OLD.quantity;
    SELECT availqty INTO L_availqty FROM partsupp
        WHERE partkey = NEW.partkey AND suppkey = NEW.suppkey;
    IF (L_availqty - L_valuediff >= 0) THEN
        BEGIN
            UPDATE partsupp SET availqty = availqty - L_valuediff
                WHERE partkey = NEW.partkey AND suppkey = NEW.suppkey;
        END;
    ELSE
        SIGNAL SQLSTATE '45000' SET message_text='Available quantity is NOT
ENOUGH';
    END IF;
END//
  
```

在上面的代码中，首先计算新记录（NEW）的 quantity 和旧记录（OLD）的 quantity 的差值，即所需的供应数。如果 partsupp 中现有的供应数足够，就更新 partsupp 表对应 partkey 和 suppkey 的供应数，否则更新失败，向用户跑出错误信息——Available quantity is not enough。

MySQL 提供了 SIGNAL 语句来抛出异常或错误，其中 SQLSTATE 后的字符串是错误代码，“00”开头代表完全成功执行，“01”开头是警告等等。“45000”是一种错误，发生这个错误时设置错误消息，就能在 MySQL workbench 中看到报错信息。

2) 在 lineitem 表上定义一个 INSERT 触发器

当插入订单明细时，先检查供应表 partsupp 中的可用数量 availqty 是否足够。

定义 INSERT 触发器与 UPDATE 触发器类似，只不过没有 OLD 变量，或者可将 OLD 变量的 quantity 视为 0。

```
CREATE TRIGGER TRI_Quantity_INSERT
BEFORE INSERT ON lineitem
FOR EACH ROW
BEGIN
    DECLARE L_valuediff, L_availqty INTEGER;

    SET L_valuediff = NEW.quantity;
    SELECT availqty INTO L_availqty FROM partsupp
        WHERE partkey = NEW.partkey AND suppkey = NEW.suppkey;
    IF (L_availqty - L_valuediff >= 0) THEN
        BEGIN
            UPDATE partsupp SET availqty = availqty - L_valuediff
                WHERE partkey = NEW.partkey AND suppkey = NEW.suppkey;
        END;
    ELSE
        SIGNAL SQLSTATE '45000' SET message_text='Available quantity is NOT
ENOUGH';
    END IF;
END//
```

3) 在 lineitem 表上定义一个 DELETE 触发器

当删除订单明细时，该订单明细订购的数量要归还对应的零件供应记录。

DELETE 触发器的定义省去了判断是否足够的过程，因为回收订单时 availqty 必然是增加的。

```
CREATE TRIGGER TRI_Lineitem_Quantity_DELETE
BEFORE DELETE ON lineitem
FOR EACH ROW
BEGIN
    DECLARE L_valuediff, L_availqty INTEGER;

    SET L_valuediff = -OLD.quantity;
    UPDATE partsupp SET availqty = availqty - L_valuediff
        WHERE partkey = OLD.partkey AND suppkey = OLD.suppkey;
END//
```

在 MySQL workbench 中查看 lineitem 的 Triggers, 发现多出了 3 个 BEFORE 触发器, 分别是 TRI_Lineitem_Quantity_UPDATE、TRI_Lineitem_Quantity_INSERT 和 TRI_Lineitem_Quantity_DELETE, 分别对应 UPDATE、INSERT 和 DELETE 操作。

Name	Event	Timing	Created
TRI_Lineitem_Quantity_UPDATE	UPDATE	BEFORE	2018-11-15 22:1...
TRI_Lineitem_Quantity_INSERT	INSERT	BEFORE	2018-11-15 22:1...
TRI_Lineitem_Quantity_DELETE	DELETE	BEFORE	2018-11-15 20:4...

4) 验证触发器

查看 1 号订单第一个明细项的零件和供应商编号、订购数量、可用数量。

```
SELECT L.partkey, L.supkey, L.quantity, PS.availqty
FROM lineitem L, partsupp PS
WHERE L.partkey = PS.partkey AND L.supkey = PS.supkey
AND L.orderkey = 1 AND L.linenum=1;
```

下图表明 1 号订单还有 7030 个可用, 其中第一个明细项订购了 17 个,。

	partkey	supkey	quantity	availqty
▶	1552	93	17	7030

A. TRI_LineItem_Quantity_UPDATE

修改 1 号订单第一个明细项的订购数量, 增加 5 个。

```
UPDATE lineitem SET quantity = quantity + 5
WHERE orderkey = 1 AND linenum = 1;
```

再次查询, 结果显示 1 号订单只剩 7025 个可用, 第一个明细项的订购数量也变成了 22 个。

	partkey	supkey	quantity	availqty
▶	1552	93	22	7025

假如我们增加 10000 个订购, 会发现更新失败, MySQL workbench 显示报错, 信息为“Available quantity is NOT ENOUGH”, 符合 UPDATE 触发器定义。

```
UPDATE lineitem SET quantity = quantity + 10000
WHERE orderkey = 1 AND linenum = 1;
```

42 22:27:57 UPDATE lineitem SET quantity = quantity + 10000 WHERE orderkey = 1 A... Error Code: 1644. Available quantity is NOT ENOUGH

再次查询, 发现订购数量依然是 22, 可用数量也没变。

B. TRI_LineItem_Quantity_INSERT

在 lineitem 中插入一条 orderkey 为 1、linenum 为 100、订购数为 17 的订单。

```
INSERT INTO lineitem VALUES(1, 1552, 93, 100, 17, 24710.35, 0.04, 0.025,
'N', 'O', '1996-03-13', '1996-02-12', '1996-03-22', 'DELIVER IN PERSON',
'TRUCK', 'egular courts above the');
```

	partkey	supkey	quantity	availqty
▶	1552	93	17	7008

同理, 插入一条订购数为 10000 的记录, 报错显示可用数量不够, 查看 lineitem 表没有这条记录, partsupp 的 1 号订单可用数量依然为 7008。

```
INSERT INTO lineitem VALUES(1, 1552, 93, 101, 10000, 24710.35, 0.04, 0.025,
'N', 'O', '1996-03-13', '1996-02-12', '1996-03-22', 'DELIVER IN PERSON',
'TRUCK', 'egular courts above the');
```

✖ 45 22:32:18 INSERT INTO lineitem VALUES(1, 1552, 93, 101, 10000, 24710.35, 0.04, ... Error Code: 1644. Available quantity is NOT ENOUGH

C. TRI_LineItem_Quantity_DELETE

删除 lineitem 中 1 号订单的第 100 个明细, 查询 partsupp 中的可用数量, 又变回了 7025, 同时第 100 个明细不再存在于 lineitem 中。

```
DELETE FROM lineitem WHERE orderkey = 1 AND linenumber = 100;
```

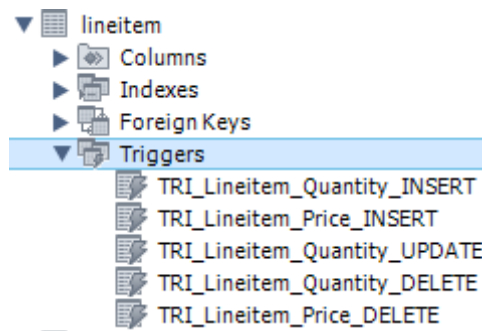
	partkey	suppkey	quantity	availqty
▶	1552	93	22	7025

3. 删除触发器

删除触发器 TRI_Lineitem_Price_UPDATE。

```
DROP TRIGGER TRI_Lineitem_Price_UPDATE;
```

查看 lineitem 表所有的 Trigger, 发现没有了 TRI_Lineitem_Price_UPDATE。



五、 实验总结

触发器实验让我想起了基于事件的编程方法（例如图形界面设计中的按钮事件），当一件事情发生时，就回调一个函数。触发器正是这样的一种回调机制，当表中发生 UPDATE、INSERT、DELETE 操作时，对应的函数被执行。函数中的逻辑包括信息打印、数据同步等。在 AFTER 和 BEFORE 触发器中出现了数据同步的操作；在 BEFORE 触发器抛出了异常，以向用户反馈错误信息。

这个实验同样也遇到了一些问题。

首先，MySQL 的触发器并不针对表的某些属性。即定义触发器时不需用 OF 语句，MySQL 会自动将触发器定义中涉及到的属性作为触发器关联的属性。为了证明这个猜想，我专门在 AFTER UPDATE 触发器上做了测试，修改除 extendedprice、discount、tax 外的其他属性，观察是否输出信息。结果没有输出任何信息，说明触发器没有被触发。

其次，触发器定义前需要修改语句的结束符。一开始，我按照书上代码输入时，MySQL workbench 编辑器给我的代码加上了红色波浪线，我就知道问题来了。原来，默认的语句块结束符是分号，而函数体中有许多以分号结尾的语句。于是我查看 MySQL 8.0 官方文档，发现要更改 DELIMITER 为//，这样整个 BEGIN...END 语句块的结束符就变成了//，也就可以在语句块中输入以分号结尾的语句，而不会让 MySQL 解释器认为

语句块以 **BEGIN** 开始而以一个不匹配的语句结束。

最后，书上的代码有一些逻辑上的错误。例如，在定义 **INSERT** 触发器时，是不存在 **OLD** 变量的，因为这是一个全新的记录，没有与之对应的原记录；定义 **DELETE** 触发器时，不存在 **NEW** 变量，因为删除一条记录不涉及产生新记录。