数据库系统实验

实验四：触发器

16 级计科教务 2 班

16337327

郑映雪

实验题目

触发器实验

实验目的

掌握数据库触发器的设计和使用方法。

实验重点和难点

实验重点：触发器的定义。

实验难点：利用触发器实现较为复杂的用户自定义完整性。

实验内容

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。

实验操作和结果

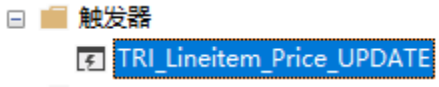
AFTER 触发器

①在 Lineitem 表上定义一个 UPDATE 触发器，当修改列项目 extendedprice、discount、tax 时，要把 orders 表的 totalprice 一起修改，以保证数据一致性。

这一步 SQL SERVER 与书上的又很不一样，在设置触发器之后，新的值会放在 inserted 临时表中，旧的值会放在 deleted 临时表中，查阅资料可知，可以采用游标对 inserted 表遍历并赋给新值，如下所示。（这一步我也复习了一下游标的概念）

```
CREATE TRIGGER TRI_Lineitem_Price_UPDATE
ON Lineitem
FOR UPDATE
AS
BEGIN
IF (UPDATE(extendedprice) or UPDATE(tax) or UPDATE(discount))
BEGIN
    --inserted 表
    DECLARE cursor_inserted cursor
    FOR SELECT orderkey,linenumber,extendedprice,discount,tax from inserted
    --DECLARE 变量需要以@开头
    DECLARE @orderkey2 int,@linenumber2 int,@extendedprice2 real,@discount2
real,@tax2 real
    OPEN cursor_inserted
    --把游标当前的值赋给变量
    FETCH NEXT FROM cursor_inserted INTO
@orderkey2,@linenumber2,@extendedprice2,@discount2,@tax2
    --赋值成功
    WHILE @@FETCH_STATUS=0
    BEGIN
        --计算新的价格
        DECLARE @new_totalprice REAL
        SELECT @new_totalprice = @extendedprice2*(1-@discount2)*(1+@tax2)
        --新的价格赋值给 totalprice
        UPDATE Orders
        SET totalprice=@new_totalprice
        WHERE orderkey=@orderkey2
        --游标移动到下一个，并赋值给变量
        FETCH NEXT FROM cursor_inserted INTO
@orderkey2,@linenumber2,@extendedprice2,@discount2,@tax2
    END
    --释放游标
    DEALLOCATE cursor_inserted
END
END
```

结果如下，触发器创建成功，验证触发器见后文：

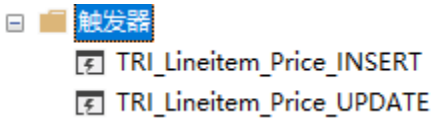


②在 lineitem 表上定义一个 after 触发器，当增加一项订单明细时，自动修改 orders 表的 totalprice，以保证数据一致性。

这一步与上一步做法相似，不同之处是需要修改新的 totalprice 为和的形式。

```
CREATE TRIGGER TRI_Lineitem_Price_INSERT
ON Lineitem
FOR INSERT
AS
BEGIN
    --insert 表
    DECLARE cursor_inserted cursor
    FOR SELECT orderkey,linenumber,extendedprice,discount,tax from inserted
    --DECLAER 变量需要以@开头
    DECLARE @orderkey2 int,@linenumber2 int,@extendedprice2 real,@discount2
real,@tax2 real
    OPEN cursor_inserted
    --把游标当前的值赋给变量
    FETCH NEXT FROM cursor_inserted INTO
@orderkey2,@linenumber2,@extendedprice2,@discount2,@tax2
    --赋值成功
    WHILE @@FETCH_STATUS=0
    BEGIN
        --计算新的价格
        DECLARE @new_totalprice REAL
        SELECT @new_totalprice = @extendedprice2*(1-@discount2)*(1+@tax2)
        --新的价格赋值给 totalprice
        UPDATE Orders
        SET totalprice=@new_totalprice+totalprice
        WHERE orderkey=@orderkey2
        --游标移动到下一个，并赋值给变量
        FETCH NEXT FROM cursor_inserted INTO
@orderkey2,@linenumber2,@extendedprice2,@discount2,@tax2
    END
    --释放游标
    DEALLOCATE cursor_inserted
END
```

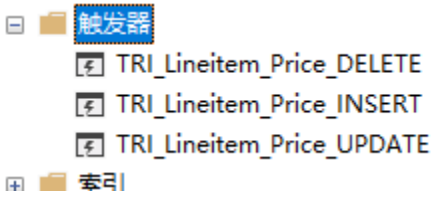
结果如下，触发器创建成功，验证触发器见后文：



(3) 在 lineitem 表上定义一个 after 触发器，当删除一项订单明细记录时，自动修改 orders 表的 totalprice，以保证数据一致性。

```
CREATE TRIGGER TRI_Lineitem_Price_DELETE
ON Lineitem
FOR DELETE
AS
BEGIN
    --insert 表
    DECLARE cursor_inserted cursor
    FOR SELECT orderkey,linenumber,extendedprice,discount,tax from inserted
    --DECLAER 变量需要以@开头
    DECLARE @orderkey2 int,@linenumber2 int,@extendedprice2 real,@discount2
real,@tax2 real
    OPEN cursor_inserted
    --把游标当前的值赋给变量
    FETCH NEXT FROM cursor_inserted INTO
@orderkey2,@linenumber2,@extendedprice2,@discount2,@tax2
    --赋值成功
    WHILE @@FETCH_STATUS=0
    BEGIN
        --计算新的价格
        DECLARE @new_totalprice REAL
        SELECT @new_totalprice = @extendedprice2*(1-@discount2)*(1+@tax2)
        --新的价格赋值给 totalprice
        UPDATE Orders
        SET totalprice=totalprice-@new_totalprice
        WHERE orderkey=@orderkey2
        --游标移动到下一个，并赋值给变量
        FETCH NEXT FROM cursor_inserted INTO
@orderkey2,@linenumber2,@extendedprice2,@discount2,@tax2
    END
    --释放游标
    DEALLOCATE cursor_inserted
END
```

结果如下，触发器创建成功，验证触发器见后文：



④验证触发器 UPDATE

```
SELECT totalprice
```

```
FROM Orders
WHERE orderkey=1854;
UPDATE Lineitem SET tax=tax+0.005
WHERE orderkey=1854 AND linenumber=1;
SELECT totalprice
FROM Orders
WHERE orderkey=1854;
```

在修改之前，totalprice 如下：

	totalprice
1	1.979697E+07

在修改之后，税率增加，totalprice 如下：

totalprice
1.898305E+07

触发器成功起了作用。

BEFORE 触发器（在 SQL SERVER 里是 INSTEAD OF 触发器）

①在 Lineitem 表上定义一个 BEFORE UPDATE 触发器，当修改订单明细中的数量时，先检查供应表 PartSupp 中的可用数量 availqty 是否足够。

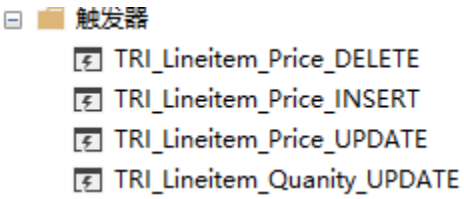
SQL SERVER 里没有 before 触发器，只有 instead of 触发器，作用与 before 触发器相同。改写结果如下：

```
   CREATE TRIGGER TRI_Lineitem_Quanity_UPDATE
    ON Lineitem
    INSTEAD OF UPDATE
    AS
        IF (UPDATE(quantity))
        BEGIN
            DECLARE @L_valuediff int, @L_availqty int, @new_quantity int,
@old_quantity int, @new_partkey int, @new_suppkey int;
            SELECT @new_quantity = quantity, @new_partkey = partkey, @new_suppkey
= suppkey FROM inserted;
            SELECT @old_quantity = quantity FROM deleted;
            SELECT @L_valuediff = @new_quantity - @old_quantity;

            SELECT @L_availqty = availqty
            FROM PartSupp
            WHERE partkey = @new_partkey AND suppkey = @new_suppkey;
```

```
        IF (@L_availqty - @L_valuediff >= 0)
        BEGIN
            PRINT 'Available quantity is ENOUGH';
            UPDATE PartSupp
            SET availqty = availqty - @L_valuediff
            WHERE partkey = @new_partkey AND suppkey = @new_suppkey;
        END
        ELSE
            RAISERROR('Available quantity is NOT ENOUGH', 16, 11);
    END
```

结果如下：



②在 Lineitem 表上定义一个 BEFORE INSERT 触发器，当插入订单明细项时，先检查供应表 PartSupp 中的可用数量 availqty 是否足够。

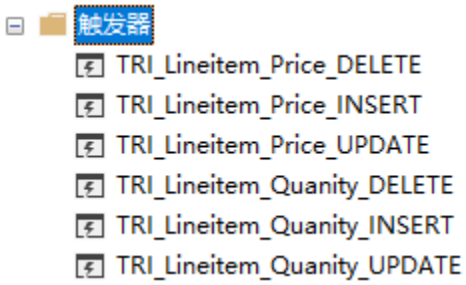
```
CREATE TRIGGER TRI_Lineitem_Quanity_INSERT
 ON Lineitem
 INSTEAD OF INSERT
 AS
    DECLARE @L_valuediff int, @L_availqty int, @new_quantity int,
@new_partkey int, @new_suppkey int;
    SELECT @new_quantity = quantity, @new_partkey = partkey, @new_suppkey =
suppkey
    FROM inserted;
    SELECT @L_valuediff = @new_quantity;
    SELECT @L_availqty = availqty
    FROM PartSupp
    WHERE partkey = @new_partkey AND suppkey = @new_suppkey;
    IF (@L_availqty - @L_valuediff >= 0)
    BEGIN
        PRINT 'Available quantity is ENOUGH';
        UPDATE PartSupp
        SET availqty = availqty - @L_valuediff
        WHERE partkey = @new_partkey AND suppkey = @new_suppkey;
    END
    ELSE
        RAISERROR('Available quantity is NOT ENOUGH', 16, 11);
```

结果如下：

③在 Lineitem 表上定义一个 BEFORE DELETE 触发器，当删除订单明细时，该订单明细项的数量要归还对应的零件供应记录。

```
CREATE TRIGGER TRI_Lineitem_Quanity_DELETE
    ON Lineitem
    INSTEAD OF DELETE
    AS
        DECLARE @L_valuediff int, @old_quantity int, @old_partkey int,
@old_suppkey int;
        SELECT @old_quantity = quantity, @old_partkey = partkey, @old_suppkey =
suppkey
        FROM deleted;
        SELECT @L_valuediff = - @old_quantity;
        UPDATE PartSupp
        SET availqty = availqty - @L_valuediff
        WHERE partkey = @old_partkey AND suppkey = @old_suppkey;
```

结果：



④验证触发器 UPDATE

在更新前：

```
SELECT L.partkey,L.suppkey,L.quantity,PS.availqty
FROM Lineitem L,PartSupp PS
WHERE L.partkey=PS.partkey AND L.suppkey=PS.suppkey AND
    L.orderkey=1854 AND L.linenumber=1
```

结果如下：

	partkey	suppkey	quantity	availqty
1	19966	14762	73	516

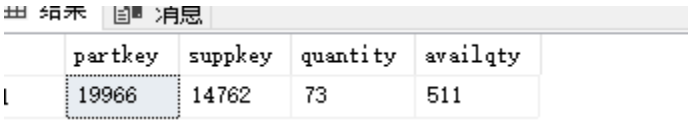
更新：

```
UPDATE Lineitem SET quantity=quantity+5
WHERE orderkey=1854 AND linenumber=1;
```

验证触发器：

```
SELECT L.partkey,L.suppkey,L.quantity,PS.availqty
FROM Lineitem L,PartSupp PS
WHERE L.partkey=PS.partkey AND L.suppkey=PS.suppkey AND
    L.orderkey=1854 AND L.linenumber=1
```

结果如下：



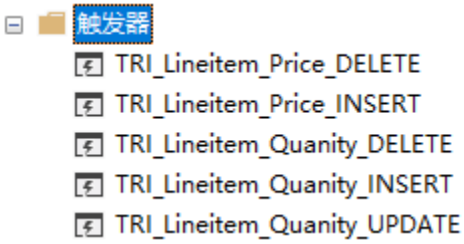
	partkey	suppkey	quantity	availqty
1	19966	14762	73	511

availqty 发生变化。

删除触发器

```
DROP TRIGGER TRI_Lineitem_Price_UPDATE;
```

如图所示，触发器由原来的六个变成了五个，PRICEUPDATE 的触发器已经删除。



实验总结

这次的实验是触发器相关的实验。通过这次实验，我了解到 SQL SERVER 的触发器种类与写法（虽然因为 DBMS 的不同并不是来源于书本而是自己去找了资料……）。SQL SERVER 与我们实验课本和理论课本都不同，它有 AFTER 触发器和 INSTEAD OF

触发器，INSTEAD OF 触发器本质上是 BEFORE 触发器，但是写法与书上有挺多不同的，这一点需要格外注意。

另外，在写 AFTER 触发器的时候我看到网上有使用游标遍历 inserted 和 deleted 表的操作，所以就仿照写了一遍，复习了一下理论课上的知识，也学到了一些 SQL SERVER 的新语句。虽然这次实验看上去内容多，但实际上只是打字比较多，本身的内容是简单的设置、验证。但是，这次的实验还是让我学到了许多。