# 实验四 触发器实验

16337341 朱志儒

## 实验4 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。

### 实验步骤

1. **AFTER触发器**
2. 在Lineitem表上定义一个UPDATE触发器，当修改订单明细时，自动修改订单Orders的Totalprice，以保持数据一致性。

CREATE TRIGGER TRI\_Lineitem\_Price\_UPDATE

ON Lineitem

AFTER UPDATE

AS

IF (UPDATE(extendedprice) OR UPDATE(discount) OR UPDATE(tax))

BEGIN

DECLARE @L\_valuediff REAL, @new\_extendedprice REAL, @new\_discount REAL, @new\_tax REAL, @new\_orderkey INT, @old\_extendedprice REAL, @old\_discount REAL, @old\_tax REAL;

SELECT @new\_extendedprice = extendedprice, @new\_discount = discount, @new\_tax = tax, @new\_orderkey = orderkey

FROM inserted;

SELECT @old\_discount = discount, @old\_extendedprice = extendedprice, @old\_tax = tax

FROM deleted;

SELECT @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

1. 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。

CREATE TRIGGER TRI\_Lineitem\_Price\_INSERT

ON Lineitem

AFTER INSERT

AS

DECLARE @L\_valuediff REAL, @new\_extendedprice REAL, @new\_discount REAL, @new\_tax REAL, @new\_orderkey INT;

SELECT @new\_discount = discount, @new\_extendedprice = extendedprice, @new\_tax = tax, @new\_orderkey = orderkey

FROM inserted;

SELECT @L\_valuediff = @new\_extendedprice \* (1 - @new\_discount) \* (1 + @new\_tax);

UPDATE Orders SET totalprice = totalprice + @L\_valuediff

WHERE orderkey = @new\_orderkey;

1. 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，以保持数据一致性。

CREATE TRIGGER TRI\_Lineitem\_Price\_DELETE

ON Lineitem

AFTER DELETE

AS

DECLARE @L\_valuediff REAL, @old\_extendedprice REAL, @old\_discount REAL, @old\_tax REAL, @old\_orderkey INT;

SELECT @old\_discount = discount, @old\_extendedprice = extendedprice, @old\_tax = tax, @old\_orderkey = orderkey

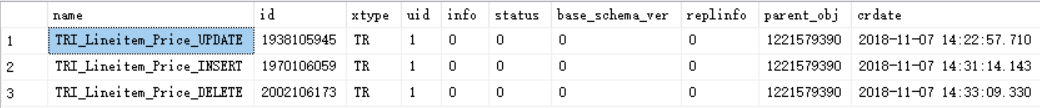
FROM deleted;

SELECT @L\_valuediff = - @old\_extendedprice \* (1 - @old\_discount) \* (1 + @old\_tax);

UPDATE Orders SET totalprice = totalprice + @L\_valuediff

WHERE orderkey = @old\_orderkey;

结果：



上述三个触发器均已建立。

1. 验证触发器TRI\_Lineitem\_Price\_UPDATE。

SELECT totalprice

FROM Orders

WHERE orderkey = 1854;

结果：



UPDATE Lineitem SET tax = tax + 0.5

WHERE orderkey = 1854;

SELECT totalprice

FROM Orders

WHERE orderkey = 1854;

结果：



显然TRI\_Lineitem\_Price\_UPDATE触发器起作用了。

1. **BEFORE触发器**
2. 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。

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

1. 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);

1. 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;

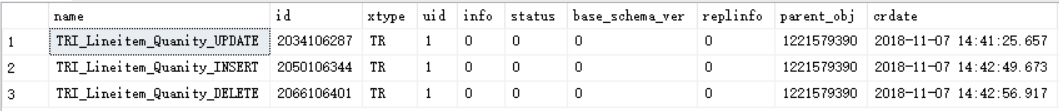
SELECT @L\_valuediff = - @old\_quantity;

UPDATE PartSupp

SET availqty = availqty - @L\_valuediff

WHERE partkey = @old\_partkey AND suppkey = @old\_suppkey;

结果：



上述三个触发器均已建立。

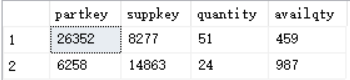
1. 验证触发器TRI\_Lineitem\_Quantity\_UPDATE。

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;

结果：



UPDATE Lineitem

SET quantity = quantity + 1000

WHERE orderkey = 1854;

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;

结果：



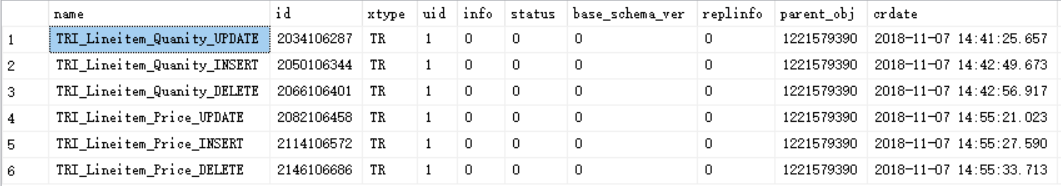
显然触发器TRI\_Lineitem\_Quantity\_UPDATE起作用了。

1. **删除触发器**

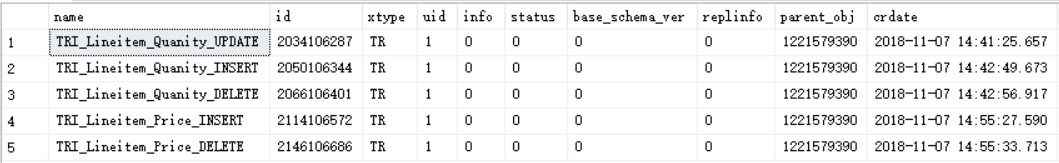
删除触发器TRI\_Lineitem\_Price\_UPDATE。

DROP TRIGGER TRI\_Lineitem\_Price\_UPDATE;

执行前：



执行后：



### 实验总结

通过本次实验，我掌握了SQL Server触发器的设计和使用方法，与书上不同的是，在SQL Server中不支持before语句，而是使用instead of代替before；SQL Server也不支持referencing new row as语句和referencing old row as语句，而是使用inserted表示在插入完成后存储所插入行的值，使用deleted存储已经更新或删除行的旧值。