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

a) 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
```

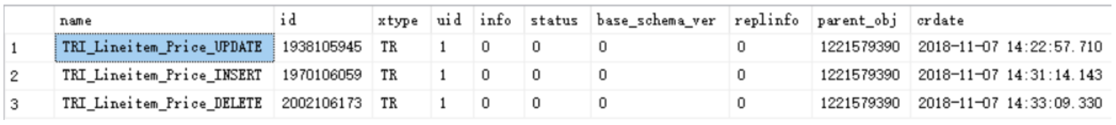
b) 在 Lineitem 表上定义一个 INSERT 触发器，当增加一项订单明细时，自动修改订单 Orders 的 Totalprice，以保持数据一致性。

```
    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;
```

c) 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:



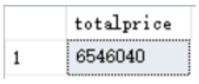
	name	id	xtype	uid	info	status	base_schema_ver	replinfo	parent_obj	crdate
1	TRI_Lineitem_Price_UPDATE	1938105945	TR	1	0	0	0	0	1221579390	2018-11-07 14:22:57.710
2	TRI_Lineitem_Price_INSERT	1970106059	TR	1	0	0	0	0	1221579390	2018-11-07 14:31:14.143
3	TRI_Lineitem_Price_DELETE	2002106173	TR	1	0	0	0	0	1221579390	2018-11-07 14:33:09.330

上述三个触发器均已建立。

d) 验证触发器 TRI_Lineitem_Price_UPDATE。

```
    SELECT totalprice
    FROM Orders
    WHERE orderkey = 1854;
```

结果:

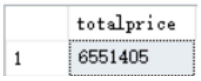


	totalprice
1	6546040

```
    UPDATE Lineitem SET tax = tax + 0.5
    WHERE orderkey = 1854;

    SELECT totalprice
    FROM Orders
    WHERE orderkey = 1854;
```

结果:



	totalprice
1	6551405

显然 TRI_Lineitem_Price_UPDATE 触发器起作用了。

2) BEFORE 触发器

a) 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
```

b) 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。

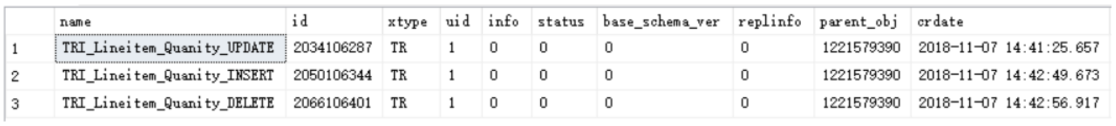
```
    CREATE TRIGGER TRI_Lineitem_Quanity_INSERT
    ON Lineitem
```

```sql
    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);
```

c) 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。

```sql
    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：



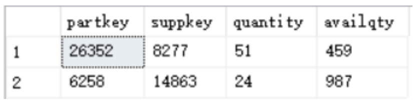
	name	id	xtype	uid	info	status	base_schema_ver	replinfo	parent_obj	crdate
1	TRI_Lineitem_Quanity_UPDATE	2034106287	TR	1	0	0	0	0	1221579390	2018-11-07 14:41:25.657
2	TRI_Lineitem_Quanity_INSERT	2050106344	TR	1	0	0	0	0	1221579390	2018-11-07 14:42:49.673
3	TRI_Lineitem_Quanity_DELETE	2066106401	TR	1	0	0	0	0	1221579390	2018-11-07 14:42:56.917

上述三个触发器均已建立。

d) 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：

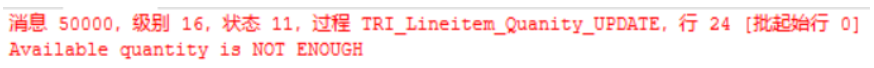


	partkey	suppkey	quantity	availqty
1	26352	8277	51	459
2	6258	14863	24	987

```
    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：



```
消息 50000，级别 16，状态 11，过程 TRI_Lineitem_Quanity_UPDATE，行 24 [批起始行 0]
Available quantity is NOT ENOUGH
```

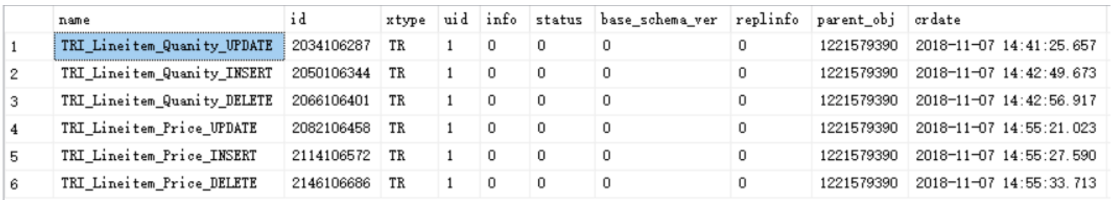
显然触发器 TRI_Lineitem_Quantity_UPDATE 起作用了。

3) 删除触发器

删除触发器 TRI_Lineitem_Price_UPDATE。

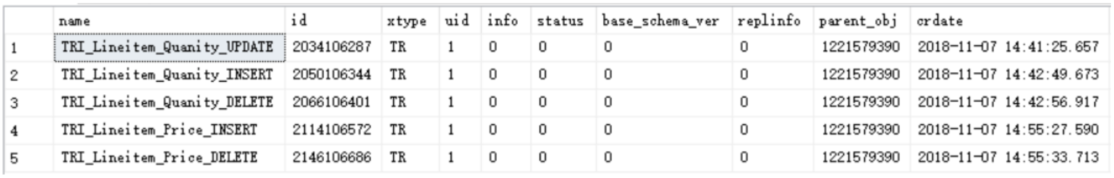
```
    DROP TRIGGER TRI_Lineitem_Price_UPDATE;
```

执行前：



	name	id	xtype	uid	info	status	base_schema_ver	replinfo	parent_obj	crdate
1	TRI_Lineitem_Quanity_UPDATE	2034106287	TR	1	0	0	0	0	1221579390	2018-11-07 14:41:25.657
2	TRI_Lineitem_Quanity_INSERT	2050106344	TR	1	0	0	0	0	1221579390	2018-11-07 14:42:49.673
3	TRI_Lineitem_Quanity_DELETE	2066106401	TR	1	0	0	0	0	1221579390	2018-11-07 14:42:56.917
4	TRI_Lineitem_Price_UPDATE	2082106458	TR	1	0	0	0	0	1221579390	2018-11-07 14:55:21.023
5	TRI_Lineitem_Price_INSERT	2114106572	TR	1	0	0	0	0	1221579390	2018-11-07 14:55:27.590
6	TRI_Lineitem_Price_DELETE	2146106686	TR	1	0	0	0	0	1221579390	2018-11-07 14:55:33.713

执行后:



	name	id	xtype	uid	info	status	base_schema_ver	replinfo	parent_obj	crdate
1	TRI_Lineitem_Quanity_UPDATE	2034106287	TR	1	0	0	0	0	1221579390	2018-11-07 14:41:25.657
2	TRI_Lineitem_Quanity_INSERT	2050106344	TR	1	0	0	0	0	1221579390	2018-11-07 14:42:49.673
3	TRI_Lineitem_Quanity_DELETE	2066106401	TR	1	0	0	0	0	1221579390	2018-11-07 14:42:56.917
4	TRI_Lineitem_Price_INSERT	2114106572	TR	1	0	0	0	0	1221579390	2018-11-07 14:55:27.590
5	TRI_Lineitem_Price_DELETE	2146106686	TR	1	0	0	0	0	1221579390	2018-11-07 14:55:33.713

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。