第11章 触 发 器

11.1 触发器的概念和工作原理

触发器是一种实施复杂的完整性约束的特殊存储过程 ,当对它所保护的数据进行修改时自动激活,防止对数据进行不正确、未授权或不一致的修改。触发器不像一般的存储过程，不可以使用存储过程的名字来调用或执行。当用户对指定的表进行修改 (包括插入、删除或更新)时，SQL Server将自动执行在相应触发器中的SQL语句。

我们在这里只讲在触发器中使用的两个专用表 —— insterted表和deleted表。具体的执行步骤将在讲解每种触发器时讲述。

SQL server为每个触发器都创建了两个专用表： inserted表和deleted表。这是两个逻辑表，由系统来维护，不允许用户直接对这两个表进行修改。它们存放于内存中，不存放在数据库中。这两个表的结构总是与被该触发器作用的表的结构相同。触发器工作完成后，与该触发器相关的这两个表也会被删除。

- insterted表：存放由于insert或UPDATE语句的执行而导致要加到该触发器作用的表中去的所有新行。即用于插入或更新表的新行值，在插入或更新表的同时，也将其副本存入insterted表中。因此，在insterted表中的行总是与作用表中的新行相同。
- deleted表：存放由于DELETE或UPDATE语句的执行而导致要从被该触发器作用的表中删除的所有行。也就是说，把被作用表中要删除或要更新的旧行移到 deleted表中。因此，deleted表和被作用表不会有相同的行。

对INSERT操作，只在insterted表中保存所插入的新行，而 deleted表中无行数据。对于DELETE操作，只在deleted表中保存被删除的旧行，而 insterted表中无行数据。对于UPDATE操作，可以将它考虑为 delete操作和insert操作的结果，所以在 inserted表中存放着更新后的新行值，deleted表中存放着更新前的旧行值。

具体触发器的执行步骤我们将在下面结合具体例子详细讲解。

11.2 创建触发器

11.2.1 一般语法

以下是创建触发器的一般语法：

```
CREATE TRIGGER[OWNER.]trigger_name
ON[OWNER.]table_name
FOR[INSERT，UPDATE，DELETE]
[WITH ENCRYPTION]
AS
sql_statements
```

在这里，

• trigger_name：要创建的触发器名称。

• ON table_name：指定所创建的触发器与之相关联的表的名字。必须是一个现存的表。

• FOR[INSERT，UPDATE，DELETE]：指定所创建的触发器将在发生哪些数据修改事件时被触发。如：INSERT，表示发生插入数据时触发。

• sql_statements：指定触发器执行的SQL语句。

我们从定义看出，每张表只有 3种触发器动作 :UPDATE触发器、 INSERT触发器和DELETE触发器。值得注意的是，触发器不能创建在临时表或视图上，而且一个触发器只能作用在一个表上。从事务角度来说，触发器与触发它的语句 (如：INSERT语句)作为同一个事务的一部分来执行。

下面我们来看看这3种触发器。

11.2.2 插入触发器

例 为loan表创建一个INSERT触发器，当在loan表上插入一行数据时(表明有人借了一本书)，自动更新copy表的on_loan(是否借出的状态信息， Y表示借出， N表示可借)列为Y。

```
USE library
GO
/* 如果存在同名的触发器，则删除之 */
IF EXISTS ( SELECT name FROM sysobjects
              WHERE type = 'TR' AND name = 'loan_insert' )
   DROP TRIGGER loan_insert
GO

CREATE TRIGGER loan_insert
  ON loan FOR INSERT
AS
 UPDATE copy SET on_loan = 'Y'
    FROM copy c INNER JOIN inserted i
    ON c.isbn = i.isbn and c.copy_no = i.copy_no
GO
```

结果显示如下：

```
The command(s) completed successfully.
```

现在，我们已经成功地创建了 loan_insert触发器。现在我们来测试这个触发器。

1) 我们来查看一本ISBN为100、COPY_NO为7的书是否借出。

```
SELECT * FROM library..copy WHERE isbn=100 and copy_no=7
```

结果显示如下：表明未借出 (on_loan是N)

```
isbn      copy_no     title_no    on_loan
--------- ----------- ----------- ---------------
100       7           10          N
(1 row(s) affected)
```

2) 下面假设123号读者借了这本书。执行如下语句：

```
INSERT loan (isbn,copy_no,title_no,member_no,out_date,due_date)
     VALUES(100,7,10,123,GETDATE(),(GETDATE()+15))
```

结果显示如下，表示借书成功。

```
(1 row(s) affected)
```

3) 下面我们再看看这本书的状态。发现 on_loan是Y(表示借出)。

```
SELECT * FROM library..copy WHERE isbn=100 and copy_no=7
isbn    copy_no     title_no       on_loan
------- ----------- -------------- -----------------
100         7             10             Y
(1 row(s) affected)
```

在copy表中on_loan列的值由N改变为Y的原因是：在执行INSERT loan的操作时，触发了在loan表上的插入触发器，是该触发器将 N改变为Y。从上述例子可以看出，通过触发器可以维护数据完整性。在上述定义触发器时，我们使用了 inserted逻辑表。inseted表是一个逻辑表，它保持所插入行的拷贝。即 (100,7,10,123,GETDATE(),(GETDATE()+15))。下面我们具体看看整个的执行步骤：

1) 首先执行INSERT语句执行插入。系统检查被插入新值的正确性 (如：约束等)，如果正确，将新行插入到loan表和inserted表中。

2) 执行触发器中的相应语句。修改copy表中相应列的信息。如果执行到ROLLBACK操作，则系统将回滚整个操作(删除第一步插入的新值，对触发器中已经执行的操作做反操作)。

从事务角度来说，触发器与触发它的语句 (如：INSERT语句)作为同一个事务的一部分来执行。在这里，一定要弄清楚的是，触发器与触发它的语句仅仅是同一个事务的一部分。请看下面一个例子：

```
BEGIN transaction aa
INSERT loan (isbn,copy_no,title_no,member_no,out_date,due_date)
     VALUES(100,6,10,123,GETDATE(),(GETDATE()+15))
ROLLBACK transaction aa
```

如果执行了上述语句，那么虽然执行 INSERT语句成功，但是INSERT语句仅仅是事务的一部分，所以在执行 rollback transaction aa后，会回滚所有的操作：包括插入 loan表和更新copy表。

11.2.3 删除触发器

例 为loan表创建一个DELETE触发器，当在loan表上删除一行数据时(表明有人归还了一本书)，自动更新copy表的on_loan(是否借出的状态信息， Y表示借出， N表示可借)列为N。

```
USE library
GO
/* 如果存在同名的触发器，则删除之 */
IF EXISTS ( SELECT name FROM sysobjects
          WHERE type = 'TR' AND name = 'loan_delete' )
   DROP TRIGGER loan_delete
GO
/*  Create the Trigger */
CREATE TRIGGER loan_delete
  ON loan FOR DELETE
AS
  UPDATE copy SET on_loan = 'N'
    FROM copy c INNER JOIN deleted d
    ON c.isbn = d.isbn and c.copy_no = d.copy_no
GO
```

结果显示如下：

```
The command(s) completed successfully.
```

下面我们来测试这个触发器。

1) 首先查看一本ISBN为100、COPY_NO为7的书是否借出。

```
USE library
GO
SELECT * FROM library..copy WHERE isbn=100 and copy_no=7
isbn     copy_no     title_no     on_loan
-------- ---------- ------------ ------------------
100          7            10           Y
(1 row(s) affected)
```

结果显示这本书已经借出。

2) 下面假设某读者归还了这本书。执行如下语句：

```
DELETE FROM library..loan  WHERE isbn=100 and copy_no=7
```

结果显示如下，表示归还成功。

```
(1 row(s) affected)
```

3) 我们再次查看这本ISBN为100、COPY_NO为7的书是否借出。

```
SELECT * FROM library..copy WHERE isbn=100 and copy_no=7
isbn      copy_no      title_no      on_loan
--------- ---------- ------------- ------------------
100          7              10             N
 (1 row(s) affected)
```

结果显示这本书可借。

在copy表中on_loan列的值由Y改变为N的原因是：在执行DELETE loan的操作时，触发了在loan表上的删除触发器，是该触发器将 Y改变为N。从上述例子可以看出，通过触发器可以维护数据完整性。在上述定义触发器时，我们使用了 deleted逻辑表。deleted表是一个逻辑表，在执行删除触发器语句时 ,从表中删除的行存放在 deleted表中。下面我们看看整个的执行步骤：

1) 首先执行DELETE语句执行删除。系统检查被删除的正确性 (如：约束等)，如果正确，将从loan表中删除该行，并将删除的旧行存放在 deleted表中。

2) 执行触发器中的相应语句。修改copy表中相应列的信息。如果执行到ROLLBACK操作，则系统将回滚整个操作(删除第一步插入的新值，对触发器中已经执行的操作做反操作)。

11.2.4 更新触发器

从触发器的定义可以看出，如果定义的是更新触发器，那么只要对指定表中的数据进行更新，这种更新无论是对表中的一行或多行，还是一列或多列，都将执行触发器操作。但是，在实际需求中，可能只关心对特定列是否被更新。如果特定的列被更新，则执行触发器操作，否则不执行触发器操作。这可以通过列上的 UPDATE语法：IF UPDATE <COLUMN_NAME> 来实现。在同一个触发器的定义语句中，可以使用多个 IF UPDATE语句来对不同的列的修改执行不同的触发器操作。

例 为copy表定义一个更新触发器，当有用户对 copy表的on_loan列进行更新时，通过raiserror函数显示提示信息。

```
USE library
GO
IF EXISTS(SELECT * FROM sysobjects
```

```
        WHERE name = 'copy_update' AND type = 'TR')
  DROP TRIGGER copy_update
GO
CREATE TRIGGER copy_update
   ON copy FOR UPDATE
AS
  IF UPDATE(on_loan)
     RAISERROR ('Library..copy.on_loan has been updated.', 10, 1)
RETURN
GO
```

结果显示如下，表示创建成功。

```
The command(s) completed successfully.
```

下面我们测试一下这个更新触发器，将一本isbn为100、copy_no为7的书的on_loan修改为Y。

```
update library..copy set on_loan = 'Y' where isbn=100 and copy_no=7
```

结果显示如下，表示触发了copy_update触发器。

```
Library..copy.on_loan has been updated.
(1 row(s) affected)
```

例 下面我们为member表创建MEMBER_UPDATE触发器，以防止用户修改成员号。

```
USE LIBRARY
GO
CREATE TRIGGER MEMBER_UPDATE
ON MEMBER FOR UPDATE
AS
IF UPDATE(MEMBER_NO)
BEGIN
   RAISERROR('TRANSACTION CANNOT BY PROCESSED.MEMBER NUMBER CANNOT BE
MODIFIED.',10,1)
ROLLBACK TRANSACTION
END
```

结果显示如下：(100,7,10,Y)

```
The command(s) completed successfully.
```

在上述定义触发器时，我们使用了 deleted逻辑表和inserted逻辑表。在执行UPDATE触发器语句时，在inserted表中存放着被更新后的新行值， deleted表中存放着被更新前的旧行值。下面我们看看整个的执行步骤(第一个例子)：

1) 首先执行UPDATE语句执行更新操作。系统检查被更新的正确性 (如：约束等)，如果正确，在表中修改该行的信息，将修改前的旧行存放在 deleted表中，并将修改后的新行存放在inserted表中。

2) 执行触发器中的相应语句。显示相应信息。如果修改了某些表中相应列的信息并执行到ROLLBACK操作，则系统将回滚整个操作 (将新值改为旧值。对触发器中已经执行的操作做反操作)。

11.2.5 触发器的组合

我们可以将在11.2.2节和11.2.3节中讲述的INSERT触发器例子(loan_insert)和DELETE触发器例子(loan_delete)组合为一个触发器。请看下面的例子。

```
USE library
```

```
GO
/*  如果存在同名的触发器，则删除之 */
IF EXISTS ( SELECT name FROM sysobjects
          WHERE type = 'TR' AND name = 'loan_insert_delete' )
   DROP TRIGGER loan_insert_delete
IF EXISTS ( SELECT name FROM sysobjects
          WHERE type = 'TR' AND name = 'loan_insert' )
   DROP TRIGGER loan_insert
IF EXISTS ( SELECT name FROM sysobjects
          WHERE type = 'TR' AND name = 'loan_delete' )
   DROP TRIGGER loan_delete
GO
/*  Create the Trigger */
CREATE TRIGGER loan_insert_delete
  ON loan FOR INSERT, DELETE
AS
 IF EXISTS (SELECT * FROM inserted)
   BEGIN
     UPDATE copy SET on_loan = 'Y'
FROM copy c INNER JOIN inserted i
ON c.isbn = i.isbn and c.copy_no = i.copy_no
   END
 ELSE
   BEGIN
     UPDATE copy SET on_loan = 'N'
FROM copy c INNER JOIN deleted d
ON c.isbn = d.isbn and c.copy_no = d.copy_no
   END
GO
```

上面这个例子实现了：当删除或增加 loan表中的行时，对copy表的on_loan列做相应修改，来反映出所借书的状态。

11.3 触发器实施数据完整性的实例

11.3.1 实现参照完整性

触发器是实现复杂的完整性约束的有效方法。在前面关于参照完整性的阐述中，通过主键和外键的关联，实现了以下两种情况的参照完整性：

1) 不能删除或更新主表中正在被相应子表中外键参照的主键值。

2) 不能向子表插入或更新在相应主表的主键列中不存在的外键值。

现在，我们以图11-1stores表和sales表为例，如果需要删除或更新主表 (stores)中正在被相应子表(sales)中外键参照的主键值时，有哪些选择呢？

1) 当对stores表的stor_id的值(如：1)更新时，同时也对子表(sales)中相应的stor_id用新值更新。即做级联更新。这可以通过触发器实现 (前提是没有建立主/外键关系)。

2) 当删除stores表中的stor_id的值(如：1)时，也同时删除子表(sales)中相应的stor_id的所

有行。即做级联删除。这可以通过触发器实现 (前提是没有建立主/外键关系)。

3) 如果在stores表中stor_id是正在被相应子表中外键参照的主键值，则不允许用户去更新或删除stor_id。这可以通过触发器实现或通过建立主 /外键关系实现。

4) 当删除stores表中的stor_id的值(如：1)时，同时将sales表中相应的stor_id值替换为空值(NULL)。这可以通过触发器实现 (前提是没有建立主/外键关系)。

stores表：

stor_id	stor_name	stor_address	city	state	zip
1	王府井新华书店	王府井大街3号	北京	北京	100001
2	西单科技书店	西单大街4号	北京	北京	100020

sales表：

stor_id	ord_num	ord_date	qty	payterms	Title_id
1	6871	1999-9-14	8	Net 60	BU1023
2	X999	1998-2-12	2	Net 30	PS3333

图11-1 数据库中的表

下面这个例子是实施参照完整性。

例 为library数据库的 adult表建立 INSERT触发器，以检查插入的人员 (外键，参照member表的主键)的有效性，即必须满足所插入的人员存在于 member表中(只针对插入单行数据，不包含对多行数据的判断)。

```
USE library
GO
CREATE TRIGGER adutt_insert
ON adult
FOR INSERT
AS
IF  (SELECT COUNT(*)
FROM member,inserted
WHERE member.member_no=inserted.member_no)=0
BEGIN
PRINT 'Transaction cannot be processed.'
PRINT 'No entry in member for this adult'.
ROLLBACK  TRANSACTION
END
```

结果显示如下：

```
The command(s) completed successfully.
```

11.3.2 实施数据完整性

我们在设计数据库时，有时为了以后编程的简单或去掉表间的连接 (提高查询速度)，考虑了一部分冗余。如 copy表中的on_loan列就是一个冗余列，目的是方便读者直接获知该书是否可借(如果不冗余，则 on_loan值应该是copy表和loan表做连接来获得)。通过触发器可以使得这些冗余数据保持最新。下面是我们在前面创建的 loan_delete触发器，可完成上述功能。

```
CREATE TRIGGER loan_delete
  ON loan FOR DELETE
AS
```

```
UPDATE copy SET on_loan = 'N'
  FROM copy c INNER JOIN deleted d
  ON c.isbn = d.isbn and c.copy_no = d.copy_no
```

11.3.3 实施业务规则

在实际应用中，有一些特定的业务规则，这也是触发器使用的一个主要方面。如：在图书管理系统中，总是规定未还图书的成员不能被删除。那么我们可以通过下述触发器实现。

```
USE library
GO
CREATE TRIGGER member_delete
ON member FOR DELETE
AS
IF (SELECT COUNT(*)
  FROM loan,deleted
  WHERE loan.member_no=deleted.member_no)>0
BEGIN
PRINT 'Transaction cannot be processed.'
PRINT 'This member shilll has books on loan.'
ROLLBACK TRANSACTION
END
ELSE
DELETE RESERVATION
FROM reservation,deleted
WHERE reservation.member_no=deleted.member_no
```

结果显示如下：The command(s) completed successfully.表示创建成功。

11.4 查看、修改和删除触发器

11.4.1 查看触发器信息

触发器也是存储过程，所以触发器被创建以后，它的名字存放在系统表 sysobjects中，它的创建源代码存放在syscomments系统表中。可以通过SQL server提供的系统存储过程来查看关于用户创建的存储过程信息。

1. 查看创建触发器的源代码

sp_helptext 触发器名

例如，查看member_delete触发器的源代码，可执行sp_helptext MEMBER_DELETE。结果显示如下：

```
Text
----------------------------------------------------------
CREATE TRIGGER member_delete
ON MEMBER FOR DELETE
AS
IF (SELECT COUNT(*)
  FROM loan,deleted
  WHERE loan.member_no=deleted.member_no)>0
BEGIN
PRINT 'Transaction cannot be processed.'
PRINT 'This member still has books on loan.'
```

```
ROLLBACK TRANSACTION
END
ELSE
DELETE RESERVATION
FROM reservation,deleted
WHERE resrvation.member_no=deleted.member_no
```

2. 查看系统表

如：查看member_delete存储过程的对象ID号。

```
  SELECT OBJECT_ID('dbo.member_delete')
```

结果显示如下：

```
-------------------------
1957582012
(1 row(s) affected)
```

又如：查看所有定义的触发器名称。

```
SELECT name FROM sysobjects
  WHERE type = 'TR'
  ORDER BY type, name
GO
```

结果显示如下：

```
name
-------------------------------------
copy_update
loan_delete
loan_insert
member_delete
member_update
(6 row(s) affected)
```

11.4.2 修改触发器

为了满足用户的新需求，触发器可能要求被修改。我们可以使用 ALTER TRIGGER语句来实现。SQL Server可以在保留原来名称的同时修改触发器的内容。

具体语法如下：

```
ALTER TRIGGER        [owner.]trigger_name
                     ON[owner.]table_name
                     FOR[INSERT，UPDATE，DELETE]
                     [with encryption]
                     AS
                     sql_statements
```

在这里，

- trigger_name:要修改的触发器名称。
- ON table_name:指定所修改的触发器与之相关联的表的名字。必须是一个现存的表。
- FOR[INSERT，UPDATE，DELETE]：指定所修改的触发器将在发生哪些数据修改事件时被触发。如：INSERT，表示发生插入数据时触发。

• sql_statements:指定触发器执行的SQL语句。

11.5 触发器的限制和注意事项

1. 触发器的嵌套

如果设置得当，触发器可以嵌套或者递归。所谓触发器的嵌套，就是指：如果一个触发器的执行，会修改“建有另一个触发器”的表，则将激活那个表上的触发器。即：当一个触发器执行插入、删除和更新时，引起另一个触发器的触发，称为触发器嵌套 (Nested)。

例 我们已经在loan表上创建了loan_insert触发器，loan_insert触发器将更新copy表的on_loan列。我们也在copy表上创建了copy_update触发器，每当在copy表上做更新操作时，显示“Library..copy.on_loan has been updated.”信息。下面我们来看看触发器的嵌套。

假设执行如下操作，在loan表上插入一行数据。

```
INSERT loan (isbn,copy_no,title_no,member_no,out_date,due_date)
    VALUES(100,8,10,123,GETDATE(),(GETDATE()+15))
```

则结果将显示如下信息：

```
Library..copy.on_loan has been updated.
(1 row(s) affected)
```

整个处理过程为(见图11-2)：首先执行INSERT操作，触发loan表的插入触发器，该触发器更新copy表，并触发copy表上的更新触发器。

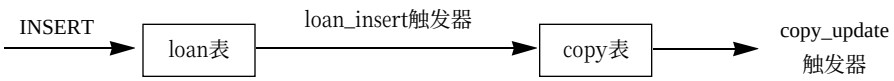

图11-2 触发器的嵌套操作

在SQL Server 7中，触发器最多可嵌套32级。

2. 性能

首先，触发器使用的两个专用表——inserted表和delted表——是创建在内存中的。所以触发器能够快速执行。其次，SQL Server 7规定触发器至多嵌套32级，从而保证触发器不会处于一个无限循环中。还有，若在表或列上存在约束，则系统优先检查约束。

3. 触发器中不允许的语句

SQL Server 7不允许以下SQL语句包含在触发器的定义中。

• 所有的CREATE语句。
• 所有的DROP语句。
• 所有的ALTER语句。
• TRUNCATE TABLE
• GRANT、REVOKE和DENY
• UPDATE STATISTICS
• RECONFIGURE
• LOAD DATABASE/LOG、RESTORE DATABASE/LOG