CHAPTER 5

第5章 触发器和数据完整性

5.1 触发器

在电力抢修工程数据库中，当某一抢修工程领取了一定数量的物资后，配电物资库存记录表中的库存量就应相应减少。如何自动实现二者的关联呢？触发器可以帮助用户解决这个问题，当用户进行插入、删除、更新等数据操作时，MySQL 就会自动执行触发器所定义的 SQL 语句。

5.1.1 触发器的基本概念

触发器(Trigger)是用户定义在关系表上的一类由事件驱动的特殊过程，也是一种保证数据完整性的方法。触发器实际上就是一类特殊的存储过程，其特殊性表现在一旦定义，无须用户调用，任何对表的修改操作均由服务器自动激活相应的触发器。

触发器的主要作用是能够实现主键和外键不能保证的复杂的参照完整性和数据的一致性。除此之外，触发器还有以下功能。

(1) 强化约束：能够实现比 CHECK 语句更加复杂的约束。

(2) 跟踪变化：侦测数据库内的操作，从而不允许数据库中未经许可的指定更新和变化。

(3) 级联运行：侦测数据库内的操作，并自动地级联影响整个数据库的各项内容。

(4) 存储过程的调用：可以调用一个或多个存储过程。

扫一扫

视频讲解

5.1.2 创建 MySQL 触发器

1. MySQL 触发器的工作原理

触发器是指对某个表执行某种数据操作时(INSERT、UPDATE、DELETE 等)，自动完成的一段程序，用以完成这些操作在引起数据变化后的完善工作。MySQL 为触发器定义了两张特殊的表：一张是 NEW，另一张是 OLD，用来表示触发器所在的表中触发了触发器的那一行数据。这两张表是建立在数据库服务器的内存中的，都是由系统管理的逻辑表，而不是真正存储在数据库中的物理表。这两张表的结构与触发器所在数据表的结构完全一致。当触发器的工作完成之后，这两张表也将会从内存中删除。

对于 INSERT 操作，NEW 表中存放的是将要(BEFORE)或已经(AFTER)插入的新数据；而对于 UPDATE 操作，NEW 表中存放的是将要或已经更新的新数据(即更新后的新值)。使用方法：NEW.columnName (columnName 为相应数据表某一列名)。

对于 DELETE 操作，OLD 表中存放的是将要或已经被删除的数据；而对于 UPDATE 操作，OLD 表中存放的将要或已经更新前的原数据。

注意：OLD 表是只读的，而 NEW 表则可以在触发器中使用 SET 语句赋值，这样不会再次触发触发器，造成循环调用。

只有数据库的所有者才能定义触发器，这是因为给表增加触发器时，将改变表的访问方式，以及与其他对象的关系，实际上是修改了数据库模式。创建一个触发器时必须指定以下几项内容：触发器的名称、在其上定义触发器的表、触发器将何时激活、执行触发操作的编程语句。

定义触发器的语法格式如下。

```
CREATE TRIGGER <触发器名>
BEFORE | AFTER
INSERT | UPDATE | DELETE
ON  表名   FOR EACH ROW
  SQL 语句
```

其中，主要参数的作用如下。

- 触发器名：标识触发器名称。
- BEFORE|AFTER：标识触发时机，表示触发器是在激活它的语句之前或之后触发。MySQL 中触发器分为 BEFORE 触发器和 AFTER 触发器两类。BEFORE 触发器在记录操作之前触发，即先完成触发再作增、删、改操作，触发的语句先于监视的增、删、改操作。BEFORE 触发器中不存在 OLD 表。AFTER 触发器在记录操作之后触发，即先完成数据的增、删、改操作再触发，触发的语句晚于监视的增、删、改操作，无法影响前面的增、删、改动作。所以，如果想要在激活触发器的语句执行之后执行相应的改变，通常使用 AFTER 选项；如果想要验证新数据是否满足使用的限制，则使用 BEFORE 选项。
- INSERT | UPDATE | DELETE：标识触发事件。
- 表名：标识建立触发器的表名，即在哪张表上建立触发器。
- FOR EACH ROW：表示任何一条记录上的操作满足触发事件都会触发该触发器。
- SQL 语句：触发器所要执行的 SQL 语句，它可以是一句 SQL 语句，也可以是用 BEGIN 和 END 包含的多条语句，不同的执行语句之间使用分号进行分隔。

注意：不能同时在一张表上建立两个相同类型的触发器。

2. INSERT 事件触发器

INSERT 事件触发器在每次向基本表插入数据时触发执行，该数据被复制到 NEW 表中。该触发事件可以用来检验要输入的数据是否符合规则、在插入的数据中增加数据、级联改变数据库中其他的数据表。

【例 5.1】 创建一个 INSERT 事件触发器，在向 Stock 表插入一条物资记录前，对新插入的 amount 字段值进行求和计算。

```
DELIMITER &&
CREATE TRIGGER tr1_stock
BEFORE INSERT
ON Stock FOR EACH ROW
  SET @sum = @sum + NEW.amount;
```

若在 Stock 表中执行如下插入语句，则运行结果如图 5.1 所示。

```
SET @sum = 0;
INSERT
INTO Stock(mat_num,mat_name,speci,warehouse,amount,unit)
VALUES ('m030','护套绝缘电线','BVV－120','供电局 1＃仓库',10,100),
       ('m031','护套绝缘电线','BVV－150','供电局 1＃仓库',20,100);
SELECT @sum;
```

	@sum
▶	30

图 5.1　触发器 tr1_stock 执行结果

【例 5.2】　创建一个 INSERT 事件触发器，在向 Salvaging 表插入一条抢修项目前，如果项目开始日期和项目结束日期相同，则将项目结束日期设为开始日期 3 天后。

```
DELIMITER &&
CREATE TRIGGER tr1_salvaging
BEFORE INSERT
ON Salvaging FOR EACH ROW
BEGIN
  IF(NEW.end_date = NEW.start_date) THEN
    SET NEW.end_date = ADDDATE(NEW.start_date,3);
  END IF;
END;
```

若在 Salvaging 表中执行如下插入语句，则运行结果如图 5.2 所示。

```
INSERT INTO Salvaging
VALUES('20190001','抢修项目 1','2019－6－1','2019－6－1',0),
      ('20190002','抢修项目 2','2019－6－1','2019－6－2',0);
```

20190001	抢修项目1	2019-06-01 00:00:00	2019-06-04 00:00:00	0
20190002	抢修项目2	2019-06-01 00:00:00	2019-06-02 00:00:00	0

图 5.2　触发器 tr1_salvaging 执行结果

【例 5.3】　创建一个 INSERT 事件触发器，在向 Out_stock 表插入一条记录后，更改对应物资在 Stock 表中的库存数量，完成级联更改操作。

```
DELIMITER &&
CREATE TRIGGER tr1_outstock
AFTER INSERT
ON Out_stock FOR EACH ROW
BEGIN
   DECLARE m_amout int(11);
   SELECT amount INTO m_amout
   FROM Out_stock
   WHERE prj_num = NEW.prj_num AND mat_num = NEW.mat_num;
                                                    -- 查询新插入记录的物资领取数量
   UPDATE Stock
```

```
    SET amount = amount - m_amout
    WHERE mat_num = NEW.mat_num;                    -- 更改 Stock 表中对应物资的库存数量
END
```

若在 Out_stock 表中执行如下插入语句，则 Stock 表中对应字段前后对比如图 5.3 所示。

```
INSERT INTO Out_stock VALUES('20110006','m003',10,'2011-3-8','工程1部')
```

mat_num	mat_name	speci	warehouse	amount	unit
m003	护套绝缘电线	BVV-35	供电局2#仓库	80	22.80

(a) 执行插入语句之前的Stock表

mat_num	mat_name	speci	warehouse	amount	unit
m003	护套绝缘电线	BVV-35	供电局2#仓库	70	22.80

(b) 执行插入语句之后的Stock表

图 5.3　触发器 tr1_outstock 执行前后对比

3. DELETE 事件触发器

DELETE 触发器在从基本表中删除数据时触发执行，在用户执行了 DELETE 触发器后，将删除的数据行保存在 OLD 表中，即数据行并没有消失，还可在 SQL 语句中引用。DELETE 触发器主要用于以下两种情况：防止删除数据库中的某些数据行、级联删除数据库中其他表中的数据行。

【例 5.4】　创建一个 DELETE 触发器，当用户从 Stock 表中删除数据时，同时将 Out_stock 表中相关物资的出库情况一并删除。

```
DELIMITER &&
CREATE TRIGGER tr2_stock
AFTER DELETE
ON Stock FOR EACH ROW
BEGIN
    DELETE
    FROM Out_stock
    WHERE mat_num = OLD.mat_num;
END
```

注意：使用触发器进行级联删除，前提是 Out_stock 表没有定义和 Stock 表相关的外键。

4. UPDATE 事件触发器

UPDATE 触发器在用户发出 UPDATE 语句后触发执行，即为用户修改数据行增加限制规则。UPDATE 触发器合并了 DELETE 触发器和 INSERT 触发器的作用。在用户执行了 UPDATE 语句后，原来的数据行从基本表中删除，但保存在 OLD 表中，同时基本表更新后的新数据行也在 NEW 表中保存了一个副本。可利用 OLD 表和 NEW 表获取更新前后的数据行，完成比较操作。

【例 5.5】　定义一张数据表 Modify_amount，用于存储 Out_stock 表中领取数量发生变化的情况。

Modify_amount 表的创建语句如下。

```
CREATE TABLE Modify_amount
(   prj_num        char(8),          -- 被修改的工程项目号
    mat_num        char(8),          -- 被修改的抢修物资号
    username       char(50) ,        -- 修改人
```

```
    updatetime      datetime,          -- 修改时间
    amount_old      int,               -- 修改前的领取数量
    amount_new      int                -- 修改后的领取数量
);
```

在 Out_stock 表上创建触发器的语句如下。

```
DELIMITER &&
CREATE TRIGGER tr2_outstock
AFTER UPDATE
ON Out_stock FOR EACH ROW
BEGIN
    INSERT
    INTO Modify_amount
    VALUES(OLD.prj_num,OLD.mat_num,USER(),NOW(),OLD.amount,NEW.amount);
END
```

若在 Out_stock 表上执行如下更新语句，则 Modify_amount 表中增加一条记录，如图 5.4 所示。

```
UPDATE Out_stock
SET amount = 8
WHERE prj_num = '20110005' AND mat_num = 'm006'
```

prj_num	mat_num	username	updatetime	amount_old	amount_new
20110005	m006	root@localhost	2019-06-14 14:31:56	3	8

图 5.4　更新 Out_stock 表领取数量后 Modify_amount 表的数据

扫一扫

视频讲解

5.1.3　创建 SQL Server 触发器

扫一扫

视频讲解

1. SQL Server 触发器的工作原理

在 SQL Server 中，触发器可以分为两大类，即 DML 触发器和 DDL 触发器。

DML 触发器是当数据库服务器中发生数据操纵语言(Data Manipulation Language)事件时执行的存储过程。

DDL 触发器是在响应数据定义语言(Data Definition Language)事件时执行的存储过程，一般用于执行数据库中的管理任务、审核和规范数据库操作、防止数据库表结构被修改等。

下面重点介绍 DML 触发器的工作原理及具体应用。

DML 触发器是在发生数据操纵语言时执行的触发器，主要针对添加、修改、删除进行触发，用于完成这些操作在引起数据变化后的完善工作。

在 SQL Server 中，为每个 DML 触发器都定义了两张特殊的表：一张是插入表(INSERTED)，另一张是删除表(DELETED)。这两张表是建立在数据库服务器的内存中的，都是由系统管理的逻辑表，而不是真正存储在数据库中的物理表。对于这两张表，用户只有读取的权限，没有修改的权限。这两张表的结构与触发器所在数据表的结构完全一致。当触发器的工作完成之后，这两张表也将从内存中删除。

对于 INSERT 操作，INSERTED 表中存放的是要插入的数据；而对于 UPDATE 操作，INSERTED 表中存放的是要更新的记录(即更新后的新值)。

对于 DELETE 操作，DELETED 表中存放的是被删除的记录；而对于 UPDATE 操作，

DELETED 表中存放的是更新前的记录(更新完毕后即被删除)。

DML 触发器又分为 AFTER 触发器和 INSTEAD OF 触发器。

1) AFTER 触发器的工作原理

AFTER 触发器是在记录变更完成后才被激活执行的。以删除操作为例,当接收到一个要执行删除操作的 SQL 语句时,SQL Server 先将要删除的记录存放在 DELETED 表中,然后把数据表中的记录删除,再激活 AFTER 触发器,执行 AFTER 触发器中的 SQL 语句。执行完毕后,删除内存中的 DELETED 表,退出整个操作。

2) INSTEAD OF 触发器的工作原理

INSTEAD OF 触发器与 AFTER 触发器不同。AFTER 触发器是在 INSERT、UPDATE、DELETE 操作完成后才被激活的,而 INSTEAD OF 触发器则是在这些操作进行之前就被激活了,并且不再执行原来的 SQL 操作,而是用触发器内部的 SQL 语句代替执行。

只有数据库的所有者才能定义触发器,这是因为给表增加触发器时将改变表的访问方式,以及与其他对象的关系,实际上是修改了数据库模式。

定义触发器的语法格式如下。

```
CREATE TRIGGER <触发器名>
ON{ 表名 | 视图名 }
[ WITH ENCRYPTION ]
{ AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ NOT FOR REPLICATION ]
AS
SQL 语句
```

其中,主要参数的作用如下。

- 触发器名：给出了触发器的名称。
- 表名|视图名：触发器所依存的表或视图的名称。
- WITH ENCRYPTION：表示加密触发器代码,使其他用户无法查询到触发器的创建语句,可防止 SQL Server 对触发器进行复制。
- AFTER：表示触发器只有在 SQL 语句中指定的所有操作都已成功执行后才激活。注意不能在视图上定义 AFTER 触发器。
- INSTEAD OF：表示在表或视图上执行增、删、改操作时用该触发器中的 SQL 语句代替原语句。在一个表或视图上,每条 INSERT、UPDATE、DELETE 语句只能定义一个 INSTEAD OF 触发器,然而可以在每个具有 INSTEAD OF 触发器的视图上定义视图。注意,INSTEAD OF 触发器不能更新带 WITH CHECK OPTION 的视图。
- [INSERT] [,] [UPDATE] [,] [DELETE]：说明激活触发器的触发条件,可选择多项,用逗号分隔。
- NOT FOR REPLICATION：表示在表的复制过程中对表的修改将不会激活触发器。
- SQL 语句：触发器所要执行的 SQL 语句,它可以是一组 SQL 语句,可以包含流程控制语句等。

可以看出,一个触发器只能应用在一张表上,但一个触发器可以包含很多动作,执行很

多功能，触发器可以建立在基本表上，也可以建立在视图上。

2. INSERT 触发器

INSERT 触发器在每次向基本表插入数据时触发执行，该数据同时复制到基本表和内存中的 INSERTED 表中。INSERT 触发器主要有 3 个作用：检验要输入的数据是否符合规则、在插入的数据中增加数据、级联改变数据库中其他的数据表。

【例 5.6】　创建一个 INSERT 触发器，在对 Stock 表进行插入后验证库存量的大小，若库存量小于 1，则撤销该插入操作。

```
IF EXISTS(SELECT name FROM sysobjects WHERE name = 'tr1_stock' AND type = 'TR')
DROP TRIGGER tr1_stock
GO
CREATE TRIGGER tr1_stock
ON Stock
AFTER INSERT
AS
DECLARE @amount int
SELECT @amount = amount
FROM INSERTED
IF @amount < 1
BEGIN
    ROLLBACK TRAN
    RAISERROR('Amount must be greater than 1!',16,10)
END
GO
```

在创建了该触发器后，一旦在 Stock 表中插入数据行，就会激活触发器 tr1_stock，验证 amount 的值。SQL 语句如下。

```
INSERT  INTO  Stock(mat_num,mat_name,speci,warehouse,amount,unit)
VALUES('m030','护套绝缘电线','BVV－120','供电局 1＃仓库',2,100)
```

上述插入语句由于库存量大于或等于 1，符合规则，可以正常插入执行。

```
INSERT  INTO  Stock(mat_num,mat_name,speci,warehouse,amount,unit)
VALUES('m031','护套绝缘电线','BVV－120','供电局 1＃仓库',0,100)
```

上述插入语句由于库存量小于 1，不符合规则，将撤销表的插入操作，提示如图 5.5 所示。

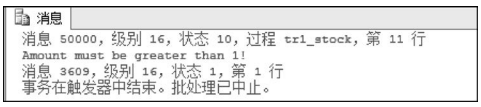

图 5.5　触发器消息提示

本例中使用了 ROLLBACK 语句。当表的修改不符合触发器设定的规则时，触发器认为修改无效，回滚事务，即撤销对表的修改操作。语法格式如下。

```
ROLLBACK  TRAN
```

执行该语句的操作为由触发器执行的所有操作以及修改语句对基本表执行的所有工作都被撤销。若使该语句在撤销表的修改时给出错误信息，可在执行了 ROLLBACK 语句时使用 PRINT 语句显示提示信息或使用 RAISERROR 语句返回错误信息。

【例 5.7】　创建一个 INSERT 触发器，在向 Out_stock 表插入一条记录后，更改对应物

资在 Stock 表中的库存数量，完成级联更改操作。

```
CREATE TRIGGER tr1_outstock
ON Out_stock
AFTER INSERT
AS
BEGIN
DECLARE @m_num char(8), @m_amount   int
SELECT @m_num = mat_num,@m_amount = amount
FROM INSERTED                    -- 查询新插入的物资编号以及领取数量
UPDATE Stock   SET amount = amount - @m_amount
WHERE mat_num = @m_num           -- 更改 Stock 表中对应物资的库存数量
END
GO
```

若在 Out_stock 表中执行以下插入语句，则 Stock 表中对应字段的前后对比如图 5.6 所示。

```
INSERT INTO Out_stock VALUES('20110006','m001',10,'2011-3-8','工程部')
```

	mat_num	amount	unit	total
1	m001	220	89.80	19756.00
2	m002	30	17.00	510.00

(a) 执行插入语句之前的Stock表

	mat_num	amount	unit	total
1	m001	210	89.80	18858.00
2	m002	30	17.00	510.00

(b) 执行插入语句之后的Stock表

图 5.6　触发器 tr1_outstock 执行前后的对比

3. DELETE 触发器

DELETE 触发器在从基本表中删除数据时触发执行，在用户执行了 DELETE 触发器后，SQL Server 将删除的数据行保存在 DELETED 表中，即数据行并没有消失，还可在 SQL 语句中引用。DELETE 触发器主要用于以下两种情况：防止删除数据库中的某些数据行、级联删除数据库中其他表中的数据行。

【例 5.8】　创建一个 DELETE 触发器，当用户从 Stock 表中删除数据时同时将 Out_stock 表中相关物资的出库情况一并删除。

```
CREATE TRIGGER tr2_stock
ON Stock
AFTER   DELETE
AS
BEGIN TRANSACTION
DECLARE @mat_num char(8)
SELECT @mat_num = mat_num    FROM DELETED
DELETE   FROM Out_stock
WHERE mat_num = @mat_num
COMMIT TRANSACTION
GO
```

注意：使用触发器进行级联删除，前提是 Out_stock 表没有定义和 Stock 表相关的外键。

4. UPDATE 触发器

UPDATE 触发器在用户发出 UPDATE 语句后触发执行，即为用户修改数据行增加限制规则。UPDATE 触发器合并了 DELETE 触发器和 INSERT 触发器的作用。在用户执行了 UPDATE 语句后，原来的数据行从基本表中删除，但保存在 DELETED 表中，同时基本表更新后的新数据行也在 INSERTED 表中保存了一个副本。用户可利用 DELETED 表和 INSERTED 表获取更新前后的数据行，完成比较操作。

【例 5.9】　创建一个 UPDATE 触发器，当用户更新 Stock 表中的数据时，从 INSERTED 表中读取修改的新的 amount 值，如果该值小于 1，则撤销更新操作，即触发器从 DELETED 表中查询修改前的值，将其重新更新到 Stock 表中（也可采用事务回滚的方法撤销更新操作）。

```
CREATE TRIGGER tr3_stock
ON Stock
AFTER  UPDATE
AS
DECLARE @amount_new int,@amount_old int,@mat_num char(10)
SELECT @amount_new = amount,@mat_num = mat_num
FROM INSERTED
IF @amount_new < 1
BEGIN
    SELECT @amount_old = amount FROM DELETED
    UPDATE Stock SET amount = @amount_old
    WHERE mat_num = @mat_num
    PRINT 'The row can not be updated!'
END
GO
```

UPDATE 语句可以检测到一个列的更新。因为有时用户并不关心表中所有列的更新，只关心一些重要列的更新，而且用户在触发器中设置数据行更新规则时往往只针对个别列。此时，可以使用 UPDATE 语句检测这些列的更新。

【例 5.10】　修改前面创建的 UPDATE 触发器，使其先检测更新的列，当更新 warehouse 列时，禁止更新；当更新 amount 列时，设置更新规则，若更新后的值小于 1，则撤销该更新操作。

```
CREATE TRIGGER tr4_stock
ON Stock
AFTER  UPDATE
AS
DECLARE @amount int
IF UPDATE(warehouse)
BEGIN
    ROLLBACK TRAN
    PRINT '不允许修改物资存放仓库!'
END
IF UPDATE(amount)
BEGIN
    SELECT @amount = amount
    FROM  INSERTED
    IF @amount < 1
    BEGIN
      ROLLBACK TRAN
      PRINT '库存量小于 1,不允许更新!'
    END
END
GO
```

【例 5.11】　定义一张数据表 Modify_amount，用于存储 Out_stock 表中领取数量发生变化的情况。

创建 Modify_amount 表的语句如下。

```
CREATE TABLE Modify_amount
(   prj_num char(8),              --被修改的工程项目号
    mat_num char(8),              --被修改的抢修物资号
    username char(6) ,            --修改人
    updatetime datetime,          --修改时间
    amount_old int,               --修改前的领取数量
    amount_newint                 --修改后的领取数量
);
```

在 Out_stock 表上创建触发器的语句如下。

```
CREATE TRIGGER  tr2_outstock
ON Out_stock
AFTER  UPDATE
AS
IF UPDATE(amount)
BEGIN
    DECLARE @amount_old int,@amount_new  int
    DECLARE @prj_no char(8),@mat_no char(8)
    SELECT @prj_no = (SELECT prj_num FROM DELETED)              --被修改的项目号
    SELECT @mat_no = (SELECT mat_num FROM DELETED)              --被修改的物资号
    SELECT @amount_old = (SELECT amount FROM DELETED)           --修改前的领取数量
    SELECT @amount_new = (SELECT amount FROM INSERTED)          --修改后的领取数量
    INSERT  INTO Modify_amount
    VALUES(@prj_no,@mat_no,USER_NAME(),GETDATE(),
           @amount_old,@amount_new)
END
GO
```

若在 Out_stock 表上执行以下更新语句，则 Modify_amount 表中增加了一条记录，如图 5.7 所示。

```
UPDATE Out_stock
SET amount = 8
WHERE prj_num = '20110005' AND mat_num = 'm006'
```

	prj_num	mat_num	username	updatetime	amount_old	amount_new
▶	20110005	m006	dbo	2015-04-21 14:36:32.940	4	8
*	NULL	NULL	NULL	NULL	NULL	NULL

图 5.7　更新 Out_stock 表领取数量后 Modify_amount 表的数据

5. INSTEAD OF 触发器

INSTEAD OF 触发器为替代操作触发器，可用于视图操作。因为视图有时显示的是表中的部分列，所以用视图修改基本表中的数据行时有可能导致失败。解决方法之一就是针对视图建立 INSTEAD OF 触发器，通过触发器插入所缺的列值完成更新。当视图执行到对基本表的插入、删除和更新操作时，用触发器的操作替代视图的操作。INSTEAD OF 触发器也可以实现级联删除的操作。注意，视图只能使用 INSTEAD OF 触发器，不能使用 AFTER 触发器。

【例 5.12】 在 Out_stock 表上创建一个 INSTEAD OF 触发器，确保插入的抢修工程项目号在 Salvaging 表中存在（需要注意的是，在插入数据时，系统先将数据插入 INSERTED 表中，再由所建的 INSTEAD OF 触发器执行实际的插入）。

```
CREATE TRIGGER  tr3_outstock
ON Out_stock
```

```
INSTEAD OF INSERT
AS
IF EXISTS ( SELECT * FROM  INSERTED
           WHERE prj_num NOT IN(SELECT prj_num FROM Salvaging))
    PRINT'对不起,有抢修工程项目号不在工程项目表中,不能正确插入!'
ELSE
    INSERT INTO Out_stock
    SELECT * FROM  INSERTED
```

当执行以下插入语句时,由于其中一条插入语句的项目号 20110007 在 Salvaging 表中并不存在,因此系统会输出对应的提示信息并拒绝执行。

```
INSERT INTO Out_stock
VALUES ('20110006','m001',2,'2011-3-9','工程 4 部'),
       ('20110007','m002',3,'2011-3-9','工程 4 部');
```

【例 5.13】　利用 INSTEAD OF 触发器实现级联删除,即若在 Salvaging 表中删除一条工程项目记录,则在 Out_stock 表中应同时删除相关项目领取物资的信息。

```
CREATE TRIGGER  tr1_salvaging
ON Salvaging
INSTEAD OF DELETE
AS
BEGIN TRANSACTION
    DELETE FROM Out_stock
    WHERE prj_num IN (SELECT prj_num FROM DELETED)
    DELETE FROM Salvaging
    WHERE prj_num IN (SELECT prj_num FROM DELETED)
COMMIT TRANSACTION
```

此时,无论 Out_stock 表与 Salvaging 表有无参照完整性约束,当执行以下删除语句时,对应的项目信息及物资领取信息均被删除。

```
DELETE FROM Salvaging WHERE prj_num = '20110005'
```

6. 复合触发器

多个触发器可以组合在一起形成复合触发器,能够使数据库的管理工作变得更加简便。

【例 5.14】　在 Salvaging 表中添加一个新列 sumcost,记录每个工程项目的抢修总成本。编写一个复合触发器,当对 Out_stock 表进行增加、删除和修改操作使抢修物资领取数量发生变化时,Salvaging 表中该项目的 sumcost 字段值能够自动更新。

```
CREATE TRIGGER  tr3_outstock
ON Out_stock
AFTER INSERT,DELETE,UPDATE
AS
BEGIN TRANSACTION
    IF UPDATE(amount)                -- 对 Out_stock 表进行增加、修改操作时更新 sumcost
    BEGIN
        UPDATE Salvaging
        SET sumcost = (SELECT SUM(Out_stock.amount * unit)
                       FROM Out_stock,Stock
                       WHERE Out_stock.mat_num = Stock.mat_num
                             AND Out_stock.prj_num = Salvaging.prj_num)
        WHERE prj_num IN (SELECT prj_num FROM INSERTED)
    END
    ELSE
```

```
    BEGIN
        UPDATE Salvaging          -- 对 Out_stock 表进行删除操作时更新 sumcost
        SET sumcost = (SELECT SUM(Out_stock.amount * unit)
                       FROM Out_stock,Stock
                       WHERE Out_stock.mat_num = Stock.mat_num
                             AND Out_stock.prj_num = Salvaging.prj_num)
        WHERE prj_num IN (SELECT prj_num FROMDELETED)
    END
COMMIT TRANSACTION
GO
```

5.1.4 删除触发器

删除触发器的语法格式如下。

```
DROP TRIGGER 触发器名
```

注意：在删除表时，依存于该表的触发器也将同时被删除。

触发器的修改操作可以理解为先删除原有的触发器，然后在同样的基本表上创建一个同名的新触发器。

扫一扫

视频讲解

5.2 数据库完整性

数据库完整性是数据的正确性和相容性。例如，配电物资库存表中的物资编号必须是唯一的；配电物资库存表中的数量必须是正数；配电物资领料出库表中的物资必须是配电物资库存表中的物资。凡是已经失真的数据都可以说其完整性受到了破坏。为了维护数据库的完整性，DBMS 必须提供一种机制检查数据库的完整性。现代数据库技术采用对数据完整性予以约束和检查的方式保护数据库的完整性。实现的方式主要有两种：一种是定义和使用完整性约束规则；另一种是通过触发器和存储过程等来实现。

前面章节曾经介绍过关系数据模型中数据完整性的概念和规则，第 3 章介绍了 CREATE TABLE 语句中实现的一些完整性约束，主要是实体完整性约束和参照完整性约束的实现。其他与数据完整性有关的内容都是用户定义的数据完整性范畴，而实现用户定义的完整性规则，除了 CREATE TABLE 命令中的 CHECK 约束，更多的是使用触发器实现灵活、复杂的数据完整性要求。

在电力抢修工程数据库中，Out_stock 表中的 prj_num 属性是外键，参照属性是 Salvaging 表中 prj_num 属性，并且要求对于某一项抢修工程，其 Out_stock 表中的领料日期 get_date 的值必须介于 Salvaging 表中该工程的 start_date 和 end_date 值之间。例如，Salvaging 表中 prj_num 为 20110006 的抢修工程的开始日期为 2011-03-08，结束日期为 2011-03-10，则 Out_stock 表中 prj_num 为 20110006 的记录的 get_date 值必须介于 2011-03-08 和 2011-03-10 之间。

这样的表和表之间的约束可以通过如下触发器来实现。

```
DELIMITER &&
CREATE TRIGGER tr3_outstock
BEFORE INSERT
ON Out_stock FOR EACH ROW
BEGIN
```

```
    DECLARE s_date datetime;
    DECLARE e_date datetime;
    DECLARE msg varchar(50);
    SELECT start_date,end_date INTO s_date,e_date
    FROM Salvaging
    WHERE prj_num = NEW.prj_num;
    IF(NEW.get_date < s_date)OR(NEW.get_date > e_date) THEN
     SET msg = CONCAT(NEW.prj_num,'项目领取的',NEW.mat_num,'物资领料日期有误!');
     SIGNAL sqlstate 'HY000' SET message_text = msg;
    END IF;
END
```

触发器创建后，如果执行以下语句，系统出错提示如图 5.8 所示，从而保证了 Out_stock 表和 Salvaging 表中数据的一致性。

```
INSERT INTO Out_stock
VALUES('20110006','m005',10,'2019-6-8','工程 1 部');
```

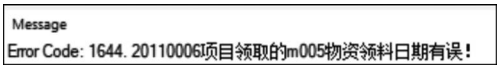

图 5.8　触发器消息提示

小结

本章介绍了触发器。存储过程和触发器都是独立的数据库对象和存储在数据库上的特殊的程序。存储过程由用户调用，完成指定的数据处理任务；触发器则是一种特殊的存储过程，由特定的操作触发，从而自动完成相关的处理任务。触发器的实现离不开两张特定的表：NEW 和 OLD，通过它们检查哪些行被修改，正确理解这两张表就可以理解触发器的本质。本章还介绍了数据库完整性相关的内容，包括规则、默认对象等，并说明了触发器在实现数据库完整性方面的重大作用。

扫一扫

自测题

习题 5

一、简答题

1. 试述触发器的概念和作用。

2. 什么是 NEW 表和 OLD 表？

二、综合题

针对第 2 章习题中的 4 张表：客户表(Customers)、代理人表(Agents)、产品表(Products)和订单表(Orders)，请编写触发器，分别实现以下操作。

(1) 向产品表(Products)插入数据前，检查产品单价 price 的值，若低于 0.50 元，则统一调整为 0.50 元。

(2) 向订单表(Orders)插入一条订货记录后，触发修改该产品在产品表(Products)中的产品销售数量 quantity 的值。

(3) 当修改订单表(Orders)的订货数量 qty 后，触发修改该项订单的订货总金额 amount 的值。