

数据库系统概论

An Introduction to Database System

第五章 数据库完整性





第五章 数据库完整性

什么是数据库的完整性

- 1) 数据的正确性和相容性
- 2) 防止不合语义的数据进入数据库。

例：学生的年龄必须是整数，取值范围为 **14--29** ；
学生的性别只能是男或女；
学生的学号一定是唯一的；
学生所在的系必须是学校开设的系；

- 3) 完整性：是否真实地反映现实世界
-



完整性控制机制

1. 完整性约束条件定义机制

2. 完整性检查机制

一般在 **INSERT, UPDATE, DELETE** 执行后开始检查，也可以在事务提交时检查。

3. 违约处理

拒绝或级联执行等。



第五章 数据库完整性

5.1 实体完整性

5.2 参照完整性

5.3 用户定义的完整性

5.4 完整性约束命名子句

5.6 触发器

5.7 小结



5.1 实体完整性

• 5.1.1 实体完整性定义

– PRIMARY KEY 定义，表级完整性约束或列级完整

[例 1] 将 Student 表中的 Sno 属性定义为码。

```
CREATE TABLE Student
```

```
(Sno CHAR(9) PRIMARY KEY,  
 Sname CHAR(20) NOT NULL,  
 Ssex CHAR(2) ,  
 Sage SMALLINT,  
 Sdept CHAR(20)  
);
```

/* 在列级定义主码 */

或者

```
CREATE TABLE Student
```

```
(Sno CHAR(9) ,  
 Sname CHAR(20) NOT NULL,  
 Ssex CHAR(2) ,  
 Sage SMALLINT,  
 Sdept CHAR(20) ,  
 PRIMARY KEY (Sno)  
);
```

/* 在表级定义主码 */



[例 2] 将 SC 表中的 Sno,Cno 属性组定义为码

```
CREATE TABLE SC  
  (Sno CHAR(9) NOT NULL,  
   Cno CHAR(4) NOT NULL,  
   Grade SMALLINT,  
   PRIMARY KEY (Sno,Cno)  
  );
```

/* 只能在表级定义主码 */



• 5.1.2 实体完整性检查和违约处理

每当用户程序对基本表插入一条记录或者对主码列进行更新操作时，按照实体完整性规则，系统将进行检查：

- 主码值是否唯一，如果不唯一则拒绝插入或修改
- 主码各属性是否为空，只要有一个为空就拒绝插入或修改

方法：全表扫描或者索引查找（一般在主码上自动建立一个索引）

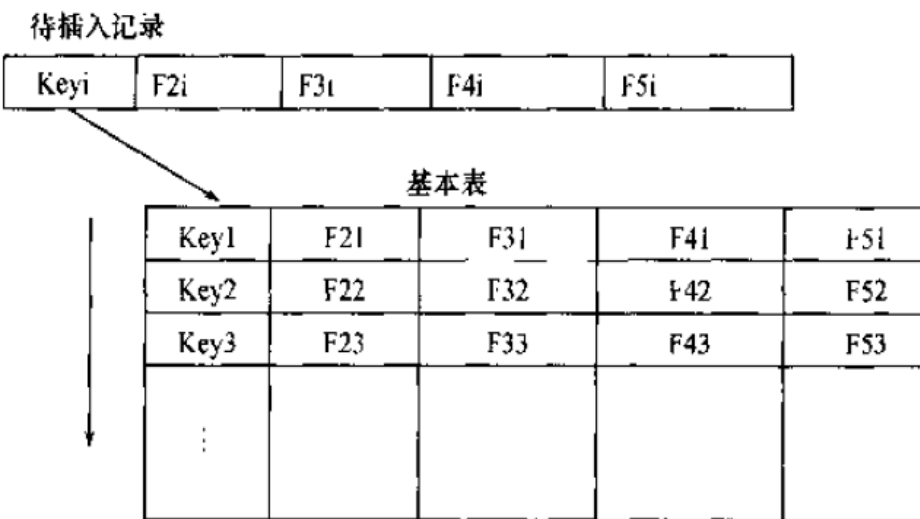


图 5.1 用全表扫描方法检查主码唯一性

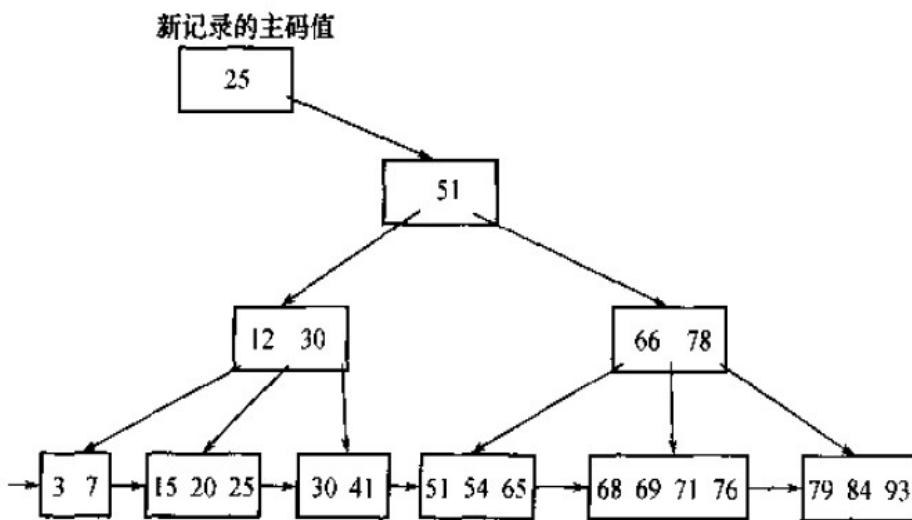


图 5.2 使用索引检查主码唯一性



5.2 参照完整性

5.2.1 参照完整性定义

– 创建表时, FOREIGN KEY 定义

[例 3] 定义 SC 中的参照完整性

```
CREATE TABLE SC
```

```
(Sno CHAR(9) NOT NULL,
```

```
 Cno CHAR(4) NOT NULL,
```

```
 Grade SMALLINT,
```

```
 PRIMARY KEY (Sno, Cno),
```

/* 在表级定义实体完整性 */

```
 FOREIGN KEY (Sno) REFERENCES Student(Sno),
```

/* 在表级定义参照完整性 */

```
 FOREIGN KEY (Cno) REFERENCES Course(Cno)
```

/* 在表级定义参照完整性 */

```
);
```

- SC 中增加一个元组, 而在 student 中找不到相应的元组
- 修改 SC 中的一个元组……
- 从 student 删除一个元组……
- 修改 student 中的一个元组……



5.2.2 参照完整性检查和违约处理

- 当上述的不一致发生时，系统可以采用以下的策略：

1 拒绝（**No Action**）执行

不允许该操作执行。该策略一般设置为默认策略。

2 级联（**Cascade**）操作

当删除或修改被参照表（**Student**）的一个元组造成了与参照表（**SC**）的不一致，则删除或修改参照表中的所有造成不一致的元组。

例如：删除 **Student** 表中的元组，**Sno** 的值为 200215121，则从要 **SC** 表中级联删除 **SC.Sno='200215121'** 的所有元组。



5.2.2 参照完整性检查和违约处理

- **3 设置为空值**

- 当删除或修改被参照表的一个元组时造成了不一致，则将参照表中的所有造成不一致的元组对应属性设置为空值。

例：

学生(学号, 姓名, 性别, 专业号, 年龄)

专业(专业号, 专业名)

学生关系的“专业号”是外码,因为专业号是专业关系的主码。

假设专业表中某个元组被删除,专业号为 12,按照设置为空值的策略,就要把学生表中专业号 = 12 的所有元组的专业号设置为空值。这对应了这样的语义:某个专业删除了,该专业的所有学生专业未定,等待重新分配专业。



5.2.2 参照完整性检查和违约处理

但在学生 - 选课数据库中, Student 关系为被参照关系, 其主码为 Sno。SC 为参照关系, Sno 为外码。若 SC 的 Sno 为空值, 则表明尚不存在的某个学生, 或者某个不知学号的学生, 选修了某门课程, 其成绩记录在 Grade 列中。这与学校的应用环境是不相符的, 因此 SC 的 Sno 列不能取空值。同样, SC 的 Cno 列不能取空值。

因此, 对于参照完整性, 除了应该定义外码, 还应定义外码列是否允许空值。

一般, 当对参照表和被参照表的操作违反了参照完整性, 系统选用默认策略, 即拒绝执行。如果能让系统采用其他的策略则必须**在创建表的时候显示地加以说明。**



显示说明参照完整性的违约处理

CREATE TABLE SC

(Sno CHAR(9) NOT NULL,

Cno CHAR(4) NOT NULL,

Grade SMALLINT,

PRIMARY KEY(Sno, Cno),

/* 在表级定义实体完整性 */

FOREIGN KEY (Sno) REFERENCES Student(Sno)

/* 在表级定义参照完整性 */

ON DELETE CASCADE /* 当删除 student 表中的元组时,级连删除 SC 表中相应的元组 */

ON UPDATE CASCADE, /* 当更新 student 表中的 sno 时,级连更新 SC 表中相应的元组 */

FOREIGN KEY (Cno) REFERENCES Course(Cno)

/* 在表级定义参照完整性 */

ON DELETE NO ACTION /* 当删除 course 表中的元组造成了与 SC 表不一致时拒绝删除 */

ON UPDATE CASCADE /* 当更新 course 表中的 cno 时,级连更新 SC 表中相应的元组 */

);

- [ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }]
- [ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }]



5.3 用户定义的完整性

- 5.3.1 属性上的约束条件的定义

- 列值非空（ **Not Null** ）；
- 列值唯一（ **Unique** ）；
- 检查列值是否满足布尔表达式（ **CHECK** ）；

- 5.3.2 属性上的约束条件检查和违约处理

当往表中插入元组或修改属性的值时，检查属性上的约束是否被满足，如果不满足则操作被拒绝执行。

- 5.3.3 元组上的约束条件的定义

- **CHECK** 短语（同属性值限制相比，元组级的限制可以设置不同属性之间的取值的相互约束条件）

- 5.3.4 元组上的约束条件检查和违约处理

当往表中插入元组或修改属性的值时，检查属性上的约束是否被满足，如果不满足则操作被拒绝执行。



5.3 用户定义的完整性

3. 用 CHECK 短语指定列值应该满足的条件

[例 7] Student 表的 Ssex 只允许取“男”或“女”。

```
CREATE TABLE Student
(Sno CHAR(9) PRIMARY KEY,          /* 在列级定义主码 */
Sname CHAR(8) NOT NULL,           /* Sname 属性不允许取空值 */
Ssex CHAR(2) CHECK (Ssex IN ('男','女')), /* 性别属性 Ssex 只允许取'男'或'女' */
Sage SMALLINT,
Sdept CHAR(20)
);
```

[例 8] SC 表的 Grade 的值应该在 0 和 100 之间。

```
CREATE TABLE SC
(Sno CHAR(9) NOT NULL,
Cno CHAR(4) NOT NULL,
Grade SMALLINT CHECK (Grade >= 0 AND Grade <= 100),
PRIMARY KEY (Sno, Cno),
FOREIGN KEY (Sno) REFERENCES Student(Sno),
FOREIGN KEY (Cno) REFERENCES Course(Cno)
);
```




5.3 用户定义的完整性

[例 9] 当学生的性别是男时,其名字不能以 Ms. 打头。

```
CREATE TABLE Student
```

```
(Sno CHAR(9),
```

```
  Sname CHAR(8) NOT NULL,
```

```
  Ssex CHAR(2),
```

```
  Sage SMALLINT,
```

```
  Sdept CHAR(20),
```

```
  PRIMARY KEY (Sno),
```

```
  CHECK (Ssex = '女' OR Sname NOT LIKE 'Ms. %')
```

```
);          /* 定义了元组中 Sname 和 Ssex 两个属性值之间的约束条件 */
```



5.6 触发器

触发器的基础知识

触发器是一种特殊的存储过程，是 SQL Server 为保证数据完整性、确保系统正常工作而设置的一种高级技术。触发器在特定的表上定义，该表也称为触发器表。当触发器所保护的数据发生变化时，触发器就会自动运行，以保证数据的完整性与正确性。



1. 触发器有如下作用:

- 1) 可以对数据库进行级联修改。
 - 2) 可以完成比 CHECK 更复杂的约束。与 CHECK 约束不同，在触发器中可以引用其他的表。
 - 3) 根据改变前后表中不同的数据进行相应的操作。
 - 4) 对于一个表上的不同的操作 (INSERT、UPDATE 或 DELETE) 可以采用不同的触发器，即使是对相同的
语句也可以调用不同的触发器完成不同的操作。
-



- 在创建数据表时，已经定义了各字段的类型及其他约束条件，比如主键、外键关系等。这些作为预先过滤，在数据写入数据库之前就会被校验，只有当这些校验全都通过后，触发器才会执行。如果前面的这些校验没有全部通过，触发器就不会执行。因为触发器是在操作之后才执行。



2. 触发器具有以下特点：

- 1) 它是在操作有效后才执行的, 即其他约束优先于触发器。
 - 2) 它与存储过程的不同之处在于存储过程可以由用户直接调用, 而触发器不能被直接调用, 是由事件触发的。
 - 3) 一个表可以有多个触发器, 在不同表上同一种类型的触发器也可以有多个。
 - 4) 触发器允许嵌套, 最多为 32 层。
 - 5) 触发器可以提高对表及表行有级联操作的应用程序的性能。
-



触发器定义之后，其名称存储于 `sysobjects` 表中，定义语句存储在 `syscomments` 表中。

定义触发器的 Transact-SQL 语句中不能出现以下语句，否则 SQL Server 将拒绝编译、存储这些语句相关的触发器。

- 所有的 CREATE 命令
- 所有的 DROP 命令
- ALTER TABLE 和 ALTER DATABASE 命令
- TRUNCATE TABLE 命令（删除表中所有行）
- GRANT 和 REVOKE 命令
- UPDATE STATISTICS 命令
- SELECT INTO 命令等。（创建表将结果集填充）



在创建触发器时，还要遵循以下原则：

- (1) 触发器的定义必须是批处理的第一条命令。
 - (2) 触发器只能在表上定义。
 - (3) 触发器不能处理 TEXT 和 IMAGE 数据类型的
大型二进制对象表列。
 - (4) 建议不要使用触发器返回一个结果集。
-



3. 触发器的类型

在 SQL Server 2008 中，根据激活触发器执行的 T-SQL 语句类型，可以把触发器分为两类：

- DML 触发器

当数据库服务器中发生数据操作语言（Data Manipulation Language）事件（Insert, Update, Delete）时执行的存储过程。

- DDL 触发器

- 响应数据定义语言（Data Definition Language）事件（Create, Alter, Drop）时执行的存储过程。



DML 触发器

- **DML 触发器**根据引起触发时间的不同可分为 **After 触发器（后触发器）** 和 **Instead Of 触发器（替代触发器）**
 - **After 触发器（后触发器）**

在记录已经改变完后（执行完 insert, update 或 delete 和处理完约束后）才被激活执行，主要用于记录变更后的处理或检查，一旦发现错误，也可以用 Rollback Transaction 语句来回滚本次操作。
 - **Instead Of 触发器（替代触发器）**

用来取代原本要进行的操作，在记录变更之前发生的，不执行原来 SQL 语句里的操作，而是代替 insert, update, delete 语句去执行触发器本身所定义的操作。
-



DML 触发器

- **DML 触发器与表和视图是不能分开的，触发器定义在表和视图中，当表或视图中执行 `insert, update, delete` 操作时触发器被触发并自动执行。**
 - **当表或视图被删除时与它关联的触发器也一同被删除。**
 - **一个表或视图可以定义多个 `After` 触发器，一个表或视图只可以定义一个 `Instead` 触发器。**
-



创建 DML 触发器的语法格式为：

```
CREATE TRIGGER 触发器名 ON 表名或视图名  
{[FOR|AFTER] | [INSTEAD OF]}  
{[DELETE] [, ] [INSERT] [, ] [UPDATE]}  
AS  
SQL 语句 [ . . . n ]
```

注：不能在视图或临时表上建立触发器，但是在触发器定义中可以引用视图或临时表。当触发器引用视图或临时表时，产生两个特殊的表：**deleted 表和 inserted 表**。这两个表的结构总与激活触发器的表的结构相同，触发器执行完成后，与该触发器相关的这两个临时表也会被自动删除。用户可以用 SELECT 语句查询临时表的内容，但不能对它们进行修改。**可以用于触发器的条件测试。**



例：对 stju 库中 s 表的 DELETE 操作定义触发器。

```
USE stju
```

```
GO
```

```
IF EXISTS( SELECT name FROM sysobjects  
            WHERE name='reader_d' AND type='TR' )
```

```
DROP TRIGGER reader_d
```

```
GO
```

```
CREATE TRIGGER reader_d
```

```
ON s
```

```
FOR DELETE
```

```
AS
```

```
PRINT ' 数据被删除! '
```

```
GO
```



INSERT 触发器和 DELETE 触发器

当向表中插入数据时，所有数据约束都通过之后，INSERT 触发器就会执行。新的记录不但加到触发器表中，而且还会有副本加入 **inserted** 表中。同样，DELETE 触发器会将删除的内容保存在 **deleted** 表中。INSERTED 表与 DELETED 表一样，它们的记录是可读的，可以进行比较，以便确认这些数据是否正确。

UPDATE 触发器

利用 UPDATE 修改一条记录时，相当于删除一条记录然后再增加一条新记录。所以 UPDATE 操作使用 **inserted** 和 **deleted** 两个表。当使用 UPDATE 操作时，触发器表中原来的记录被移到 **deleted** 表中，修改过的记录插入到 **inserted** 表中，触发器可以检查这两个表，以便确定应执行什么样的操作。



例 1

- 例：创建触发器“T_学生删除”，从“学生”表中删除数据时，相应地从“成绩”表中删除数据。

Create trigger T_学生删除

On 学生

After delete

As

Delete from 成绩

Where 学生编号 = (select 学生编号 from deleted)

实验思考题： 1、如果删除了多行，触发器只执行一次还是多次？

2、 **deleted** 表里面的数据多行？取哪行？



例 2

- 例：创建触发器“T_教师添加”，向“辅导员”表中添加数据时，相应地向“教师”表中添加数据。

Create trigger T_教师添加

On 辅导员

After insert

As

Insert into 教师（编号，姓名，出生年月）

Select 编号，姓名，出生年月 from inserted



例 3

- 例：创建触发器“T_教师修改”，向“辅导员”表中修改“姓名”列时，相应地修改“教师”表中的对应数据。

Create trigger T_教师修改

On 辅导员

After update

As

If update(姓名)

Begin

Update 教师

Set 姓名 = (select 姓名 from inserted)

Where 编号 = (select 编号 from deleted)

End



局部变量与全局变量

- 局部变量

局部变量是用户自定义的变量。使用范围是定义它的批、存储过程或触发器。局部变量前面通常加上 @ 标记。

- DECLARE 定义局部变量，并指明此变量的数据类型
- SET 或 SELECT 命令对其赋值。局部变量的数据类型可以是用户自定义的数据类型，也可以是系统数据类型，但不能将其定义为 TEXT 或 IMAGE 数据类型。
- 定义局部变量的语法如下：

```
DECLARE @local_variable data_type  
        [, @local_variable data_type]...
```

DECLARE 命令可以定义多个局部变量，之间用逗号分隔。



用 SELECT 为局部变量赋值的语法如下：

```
SELECT @variable_name=expression select statement  
      [, @variable_name=expression select statement]  
[FROM list of tables]  
[WHERE expression]  
[GROUP BY... ]  
[HAVING... ]  
[ORDER BY]
```

说明：

- (1) SELECT 命令可以将一个表达式的值赋给一个局部变量，也可以将一个 SELECT 查询的结果赋给一个局部变量。
 - (2) SELECT 命令通常返回一个值给局部变量。当返回多个值，
则变量的值为最后一个返回值。
-



【例 4】 多个返回值的赋值。

- **DECLARE @var1 varchar(8)**
- **SELECT @var1=' 学生姓名 '**
- **SELECT @var1 = sname**
- **FROM s**
- **Select @var1 AS ' 学生姓名 '**

执行结果为：

学生姓名

张三

----- 返回**最后一名**学生的姓名（注意顺序）



【例 5】

```
DECLARE @var1 varchar(8)  -- 声明局部变量
SELECT @var1=' 学生姓名 '  -- 为局部变量赋初值
Print @var1                -- 显示局部变量结果
SELECT @var1=sname
FROM s
WHERE sno=200215121
SELECT @var1 as ' 学生姓名 '
```



用 SET 为局部变量赋值

用 SET 为局部变量赋值的常用语法格式为:

`SET @local_variable= expression`



【例6】使用SET 命令赋值的变量。

```
USE stju
```

```
GO
```

```
DECLARE @no varchar(10)
```

```
SET @no='200215122'
```

```
SELECT sno, sname
```

```
FROM s
```

```
WHERE sno= @no
```

```
GO
```

执行结果为：

sno	sname
-----	-----
200215122	李亚茜



全局变量

全局变量是一组特殊的函数，他们的名称以 @@ 开头，而且不需要任何参数，在调用时也无需在函数名后面加上一对（），这些函数又称为“无参函数”



- 使用全局变量时请注意以下规则：
 - 1) 全局变量是由 SQL Server 系统提供并赋值的，是在服务器级定义的变量。用户不能建立全局变量，也不能用 SET 语句修改全局变量的值。但可以将全局变量的值赋给局部变量，以便保存和处理。
 - 2) 用户只能使用系统预定义的全局变量。
 - 3) 引用全局变量时，前面一定加上 @@ 标记。
 - 4) 用户不能定义与系统全局变量同名的局部变量，否则将产生不可预测的结果。



例如：

- **@@ERROR** 保存最近执行操作的错误状态，即返回最后一次执行 **SQL** 语句的错误代码；
 - **@@MAX_CONNECTIONS** 返回 **SQL Server** 上允许用户同时连接的最大数；
 - **@@CONNECTIONS** 返回 **SQL Server** 最近一次启动后连接或尝试连接的次数。
 - **@@ROWCOUNT** ， 是返回最近一次数据库操作所涉及到的行数。
-



【例 7】使用全局变量 @@ROWCOUNT，查询命令

影响的行数。

```
UPDATE Readers
```

```
SET 已借数量 =2
```

```
SELECT @@ROWCOUNT AS ‘行数’
```

```
GO
```

执行结果为

行数

5



例 8

- 例：在“成绩”上创建触发器，检查插入的成绩是否在 0 到 100 之间。

Create trigger check_成绩

On 成绩

For insert, update

As

Declare @score int

Select @score = 成绩 from inserted

If @score<0 or @score>100

Begin

Print '成绩必须在 0 到 100 之间！'

Rollback

End



例 9

- 在“学生”表中创建触发器，当有人试图修改学生表中的数据时，利用下述触发器可以跳过修改数据的 **SQL** 语句（防止数据被修改），并向客户端显示提示信息。

Create trigger T_学生_update

On 学生

Instead of update

As

Begin

 raiserror('对不起，学生表的数据不允许修改' ,16,10)

End (考虑：如何用 **after** 触发器改写？)



例 10

Create trigger T_学生_update2

After update

As

Begin

print ‘对不起，学生表的数据不允许修改’

Rollback

End



例 11

- **Readers** (读者编号, 读者类型, 姓名, 单位, 已借图书数量)
- **Books** (图书编号, 数量, 出版社名)
- **borrowinf** (读者编号, 图书编号, 借书日期, 还书日期)
- 当在表 borrowinf 中插入借阅信息记录时, 得到该书的应还日期。

```
CREATE TRIGGER T_return_date
ON borrowinf
for INSERT
AS
DECLARE @type varchar(8)
SELECT @type= 读者类型
FROM readers
WHERE 编号=(select 读者编号 from INSERTED )
update borrowinf set 还书日期=getdate()+
    case
    when @type=3 then 45
    when @type=2 then 60
    when @type=1 then 90
    end
where 读者编号=(select 读者编号 from
```

触发器创建之后, 用户执行一条命令:
**insert into borrowinf(读者编号 , 图书编号)
values('2004060002','F33.33')**



例 12

```
CREATE TRIGGER s_d
ON sc
FOR DELETE
AS
DECLARE @data_yj int
SELECT @data_yj= grade FROM deleted
IF @data_yj>0
begin
    PRINT '成绩为 '+RTRIM(@data_yj)
    rollback
end
ELSE
PRINT '该学生选课记录已被删除！'
```

触发器创建之后，用户执行一条命令：
DELETE from sc WHERE sno=200215125



- 下面那条命令能使数据库数据发生变化 ()
- **USE CollegeMIS**
- **GO**
- **CREATE TRIGGER Update_Teacher ON Teacher**
- **FOR UPDATE**
- **AS**
- **IF UPDATE(TeaID)**
- **BEGIN**
- **PRINT '不能修改教师的身份证号'**
- **ROLLBACK TRANSACTION**
- **END** ☐ `UPDATE Teacher SET TeaName=' asdd' where TeaID=' 0978'`
`UPDATE Teacher SET TeaID=' 0977' where TeaID=' 0978'`
- **GO** `UPDATE Teacher SET TeaID=' 0977' where age=30`
`UPDATE Teacher SET TeaID=' 0977' where TeaName=' asdd'`



• 下面程序执行结果 (

○ 列出所有年龄60岁以上的教师记录
· 删除所有年龄60岁以上的教师记录
结果为空
非法操作|

• **USE CollegeMIS**

• **GO**

• **CREATE TRIGGER Delete_Teacher ON Teacher**

• **FOR DELETE**

• **AS**

• **ROLLBACK TRANSACTION**

• **GO**

• **DELETE FROM Teacher WHERE age>60**

• **GO**

• **SELECT * FROM Teacher WHERE age>60**



实验题目：

例 1：在 market 数据库中建立一个名为 del_goods 的 DELETE 触发器，存储在 goods 表中。当用户删除 goods 表中的某些货品时，这些货品在 orders 表中的订单全部删除，以实现 goods 表和 orders 表的级联删除。

- 第一步：创建触发器
- USE market
- GO
- CREATE TRIGGER del_goods ON goods
- AFTER DELETE
- AS
- DELETE from orders WHERE 货品名称 IN
- (SELECT 货品名称 FROM DELETED)
- GO



第二步：检验触发器的作用

```
DELETE from goods WHERE 货品名称 ='pen'  
IF NOT EXISTS(SELECT * FROM Orders  
                WHERE 货品名称 ='pen')  
PRINT '相关记录已从 orders 表中删除掉！'
```



例 2：在 **market** 数据库中创建一个名为 **ins_orders** 的 **INSERT** 触发器，存储在 **orders** 表中，当向表 **orders** 中插入一条记录时，检查该订单中的货品是否正在整理中（查看对应货品 **goods** 表中的状态是否为 1 在），如果是在整理中，则不能下定单（该记录不能插入 **goods** 表中）。

第一步：创建触发器

```
USE market
```

```
GO
```

```
CREATE TRIGGER ins_orders ON orders
```

```
    AFTER INSERT
```

```
    AS
```

```
    DECLARE @ x char(20), @ y bit
```



```
SELECT @x= 货品名称 FROM inserted
```

```
SELECT @y= 状态 FROM goods
```

```
WHERE 货品名称 = @x
```

```
IF @y=1
```

```
BEGIN
```

```
    PRINT (' 本货品正在整理中，现在不能下定  
单' )
```

```
    ROLLBACK TRANSACTION
```

```
END
```

```
GO
```



第二步：检验触发器的作用

```
INSERT into orders ( 货品名称 , 客户编号 , 数量 )  
VALUES ('desk',2,5)  
SELECT * FROM orders  
WHERE 货品名称 ='desk'
```



- 例 3： 创建触发器，当向 **CJB** 表中插入一个学生的成绩时，将 **XSB** 表中该学生的总学分加上添加的课程的学分。

Create trigger cjb_insert

On cjb

After insert

As

Begin

declare @num char(6),@kc_num char(3)

Declare @xf int

Select @num= 学号 , @kc_num= 课程号 from inserted

Select @xf = 学分 from kcb where 课程号 =@kc_num

Update xsb set 总学分 = 总学分 + @xf where 学号 =@num

Print ' 修改成功'

End



- 例 4： 向 XSCP 表插入或修改一记录时，通过触发器检查记录 CPBH 字段的值在 CP 表是否存在，如不存在，则取消插入或修改操作

```
if exists(select name from sysobjects where xtype='TR'  
and name = 'xscp_tri')
```

```
drop trigger xscp_tri
```

```
Go
```

```
Create trigger xscp_tri on xscp
```

```
for insert, update
```

```
As
```

```
Begin
```

```
if((select cpbh from inserted) not in (select cpbh from cp))
```

```
rollback
```

```
End
```



禁用 DML 触发器:

Alter table 数据表名

Disable trigger 触发器名 | ALL

(如果要禁用所有触发器, 用 ALL 代替触发器名)

启用 DML 触发器:

Alter table 数据表名

Enable trigger 触发器名 | ALL

删除 DML 触发器:

Drop trigger 触发器名

查看 DML 触发器:

sp_help ‘触发器名’ 或 sp_helptext ‘触发器名’



DDL 触发器

- 格式:

Create trigger 触发器名
on {all server|database}
{for|after}
As
SQL 语句



DDL 触发器

- 例：建立用于保护“实例数据库”中的数据表不被删除的触发器。

Use 实例数据库

Create trigger T_禁止删除表

On database

For drop_table

As

Begin

print ‘对不起，表不允许删除！’

rollback

End
