

The logo for SQL Server 2005 features a blue and green wave-like background. On the left, there are two spheres: a blue one and a green one. The text "SQL SERVER 2005" is displayed in a stylized, metallic font across the center of the wave.

SQL SERVER 2005

# 数据库系统概论

第五章 数据库完整性

# 本节内容

- 第一节 概述
- 第二节 实体完整性
- 第三节 参照完整性
- 第四节 用户定义的完整性
- 第五节 完整性约束命名字句
- 第六节 域中的完整性限制(了解)
- 第七节 触发器



# 第一节 概述

## ❖ 数据库的完整性

- 数据的正确性和相容性

## ❖ 数据的完整性和安全性是两个不同概念

### ■ 数据的完整性

- 防止数据库中存在不符合语义的数据，也就是防止数据库中存在不正确的数据
- 防范对象：不合语义的、不正确的数据

### ■ 数据的安全性

- 保护数据库防止恶意的破坏和非法的存取
- 防范对象：非法用户和非法操作





❖ 为维护数据库的完整性，DBMS必须：

- 提供定义完整性约束条件的机制
- 提供完整性检查的方法
- 违约处理

# 本节内容

- 第一节 概述
- 第二节 实体完整性
- 第三节 参照完整性
- 第四节 用户定义的完整性
- 第五节 完整性约束命名字句
- 第六节 域中的完整性限制(了解)
- 第七节 触发器



# 第二节 实体完整性

## ❖ 实体完整性定义

## ❖ 实体完整性检查和违约处理

# 实体完整性定义

## ❖ 关系模型的实体完整性

- CREATE TABLE中用PRIMARY KEY定义

## ❖ 单属性构成的码有两种说明方法

- 定义为列级约束条件
- 定义为表级约束条件

## ❖ 对多个属性构成的码只有一种说明方法

- 定义为表级约束条件

# 示例

**[例1]** 将Student表中的Sno属性定义为码。

(1)在列级定义主码

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

(2)在表级定义主码

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



# 示例

**[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*/
);
```

## 第二节 实体完整性

❖ 实体完整性定义

❖ 实体完整性检查和违约处理

# 实体完整性检查和违约处理

❖ 插入或对主码列进行更新操作时，RDBMS按照实体完整性规则自动进行检查。包括：

- 1. 检查主码值是否唯一，如果不唯一则拒绝插入或修改
- 2. 检查主码的各个属性是否为空，只要有一个为空就拒绝插入或修改

❖ 检查记录中主码值是否唯一的一种方法是进行**全表扫描**

待插入记录

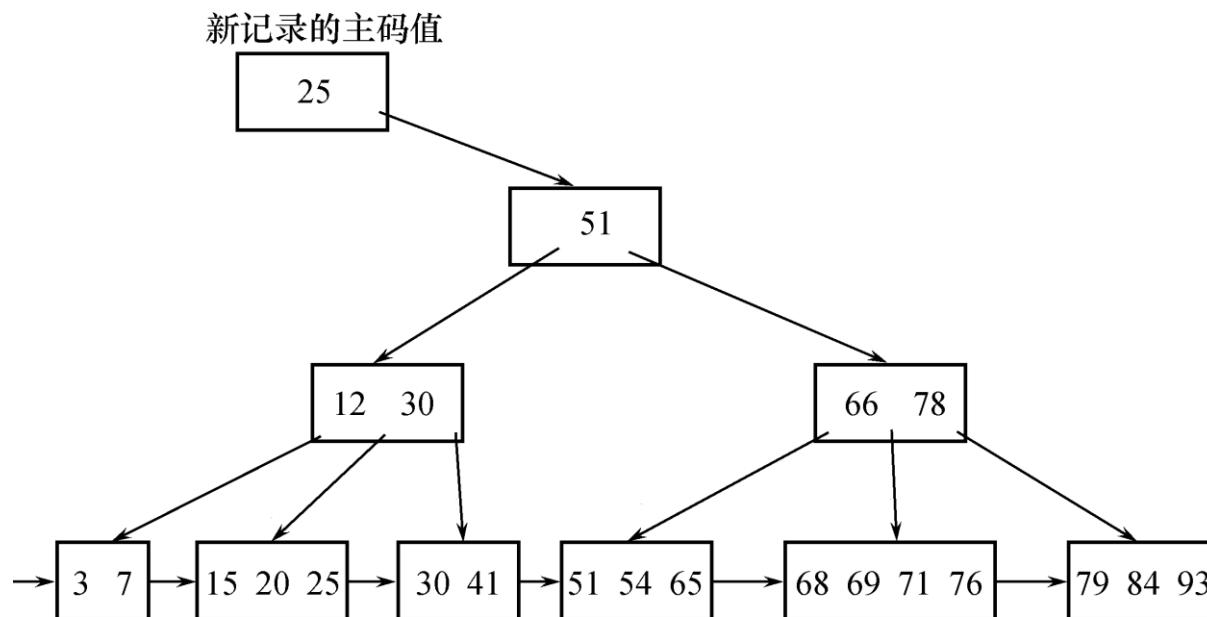
Key <sub>i</sub>	F2 <sub>i</sub>	F3 <sub>i</sub>	F4 <sub>i</sub>	F5 <sub>i</sub>
------------------	-----------------	-----------------	-----------------	-----------------

基本表

Key1	F21	F31	F41	F51
Key2	F22	F32	F42	F52
Key3	F23	F33	F43	F53
⋮				

## ❖ 索引

- 为了避免全表扫描，一般会在主码上建立索引



# 本节内容

- 第一节 概述
- 第二节 实体完整性
- 第三节 参照完整性
- 第四节 用户定义的完整性
- 第五节 完整性约束命名字句
- 第六节 域中的完整性限制(了解)
- 第七节 触发器



# 第三节 参照完整性

❖ 参照完整性定义

❖ 参照完整性检查和违约处理

# 参照完整性定义

## ❖ 关系模型的参照完整性定义

- 在CREATE TABLE中用FOREIGN KEY短语定义哪些列为外码
- 用REFERENCES短语指明这些外码参照哪些表的主码



例如 关系SC中一个元组表示一个学生选修的某门课程的成绩，（Sno，Cno）是主码。Sno，Cno分别参照引用Student表的主码和Course表的主码

**[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),
```

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

```
);
```

# 第三节 参照完整性

❖ 参照完整性定义

❖ 参照完整性检查和违约处理

# 参照完整性检查和违约处理

## 可能破坏参照完整性的情况及违约处理

被参照表（例如Student）	参照表（例如SC）	违约处理
可能破坏参照完整性 ←	插入元组	拒绝
可能破坏参照完整性 ←	修改外码值	拒绝
删除元组 →	可能破坏参照完整性	拒绝/级连删除/设置为空值
修改主码值 →	可能破坏参照完整性	拒绝/级连修改/设置为空值

# 违约处理

## ❖ 参照完整性违约处理

- 1. 拒绝(NO ACTION)执行
  - 默认策略
- 2. 级联(CASCADE)操作
- 3. 设置为空值 (SET-NULL)
  - 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

# 示例

**[例4]** 显式说明参照完整性的违约处理示例。

```
CREATE TABLE SC
(Sno CHAR(9) NOT NULL,
Cno CHAR(4) NOT NULL,
Grade SMALLINT,
PRIMARY KEY (Sno, Cno) ,
FOREIGN KEY (Sno) REFERENCES Student(Sno)
ON DELETE CASCADE /*级联删除SC表中相应的元组*/
ON UPDATE CASCADE, /*级联更新SC表中相应的元组*/
FOREIGN KEY (Cno) REFERENCES Course(Cno)
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，级联更新SC表中相应的元组*/
);
```

# 本节内容

- 第一节 概述
- 第二节 实体完整性
- 第三节 参照完整性
- 第四节 用户定义的完整性
- 第五节 完整性约束命名字句
- 第六节 域中的完整性限制(了解)
- 第七节 触发器



# 第四节 用户定义的完整性

❖ 属性上的约束条件的定义

❖ 属性上的约束条件检查和违约处理

❖ 元组上的约束条件的定义

❖ 元组上的约束条件检查和违约处理

# 属性上的约束条件的定义

## ❖ CREATE TABLE时定义

- 列值非空（NOT NULL）
- 列值唯一（UNIQUE）
- 检查列值是否满足一个布尔表达式（CHECK）





## ❖ 1. 不允许取空值

**[例5]** 在定义SC表时，说明Sno、Cno、Grade属性不允许取空值。

```
CREATE TABLE SC
```

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

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

```
Grade SMALLINT NOT NULL,
```

```
PRIMARY KEY (Sno, Cno), /* 如果在表级定义实体完整性，隐含了Sno,  
                          Cno不允许取空值，则在列级不允许取空值  
                          的定义就不必写了 */  
) ;
```

## ❖ 2.列值唯一

**[例6]** 建立部门表DEPT，要求部门名称Dname列取值唯一，部门编号Deptno列为主码。

```
CREATE TABLE DEPT
(Deptno NUMERIC(2),
  Dname CHAR(9) UNIQUE, /*要求Dname列值唯一*/
  Location CHAR(10),
  PRIMARY KEY (Deptno)
);
```



**[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 (Sno) REFERENCES Student(sno),
FOREIGN (Cno) REFERENCES Course(cno)
);
```

# 第四节 用户定义的完整性

❖ 属性上的约束条件的定义

❖ 属性上的约束条件检查和违约处理

❖ 元组上的约束条件的定义

❖ 元组上的约束条件检查和违约处理

# 属性上的约束条件检查和违约处理

❖ 插入元组或修改属性的值时，RDBMS检查

属性上的约束条件是否被满足

❖ 如果不满足则操作被拒绝执行

# 第四节 用户定义的完整性

❖ 属性上的约束条件的定义

❖ 属性上的约束条件检查和违约处理

❖ 元组上的约束条件的定义

❖ 元组上的约束条件检查和违约处理

# 元组上的约束条件的定义

- ❖ 在CREATE TABLE时可以用CHECK短语定义元组上的约束条件，即元组级的限制
- ❖ 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



**[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*/
);
```

- ✓性别是女性的元组都能通过该项检查，因为Ssex='女' 成立；
- ✓当性别是男性时，要通过检查则名字一定不能以Ms.打头

# 第四节 用户定义的完整性

❖ 属性上的约束条件的定义

❖ 属性上的约束条件检查和违约处理

❖ 元组上的约束条件的定义

❖ 元组上的约束条件检查和违约处理

# 元组上的约束条件检查和违约处理

❖ 插入元组或修改属性的值时，RDBMS检查

元组上的约束条件是否被满足

❖ 如果不满足则操作被拒绝执行



# 本节内容

- 第一节 概述
- 第二节 实体完整性
- 第三节 参照完整性
- 第四节 用户定义的完整性
- 第五节 完整性约束命名子句
- 第六节 域中的完整性限制(了解)
- 第七节 触发器



# 第五节 完整性约束命名字句

❖ 完整性约束命名字句

❖ 修改表中的完整性限制

# 完整性约束命名子句

## ❖ CONSTRAINT 约束

CONSTRAINT <完整性约束条件名>

[PRIMARY KEY短语

|FOREIGN KEY短语

|CHECK短语]

**[例10]** 建立学生登记表Student，要求学号在90000~99999之间，姓名不能取空值，年龄小于30，性别只能是“男”或“女”。

```
CREATE TABLE Student
(Sno NUMERIC(6)
  CONSTRAINT C1 CHECK (Sno BETWEEN 90000 AND 99999),
Sname CHAR(20)
  CONSTRAINT C2 NOT NULL,
Sage NUMERIC(3)
  CONSTRAINT C3 CHECK (Sage < 30),
Ssex CHAR(2)
  CONSTRAINT C4 CHECK (Ssex IN ('男', '女')),
  CONSTRAINT StudentKey PRIMARY KEY(Sno)
);
```

- ✓ 在Student表上建立了5个约束条件，包括主码约束（命名为StudentKey）以及C1、C2、C3、C4四个列级约束。

**[例10]** 建立教师表Teacher，要求每个教师的应发工资不低于3000元。  
(应发工资等于实发工资Sal和扣除项Deduct之和)

**CREATE TABLE TEACHER**

(Eno NUMERIC(4) **PRIMARY KEY**,

Ename CHAR(10),

Job char(8),

Sal NUMERIC(7,2),

Deduct NUMERIC(7,2),

Deptno NUMERIC(7,2),

**CONSTRAINT EMPFKEY FOREIGN KEY (Deptno) REFERENCES DEPT(Deptno),**

**CONSTRAINT C1 CHECK(Sal + Deduct >= 3000)**

);

✓ 在Teacher表上建立了外键约束。



# 第五节 完整性约束命名字句

❖ 完整性约束命名子句

❖ 修改表中的完整性限制

# 修改表中的完整性限制

## ❖ 使用ALTER TABLE语句修改表中的完整性限制

**[例13]** 修改表Student中的约束条件，要求学号改为在900000~999999之间，年龄由小于30改为小于40。

- 可以先删除原来的约束条件，再增加新的约束条件

```
ALTER TABLE Student  
DROP CONSTRAINT C1;
```

```
ALTER TABLE Student  
ADD CONSTRAINT C1 CHECK (Sno BETWEEN 900000 AND 999999),
```

```
ALTER TABLE Student  
DROP CONSTRAINT C3;
```

```
ALTER TABLE Student  
ADD CONSTRAINT C3 CHECK (Sage < 40);
```

# 本节内容

- 第一节 概述
- 第二节 实体完整性
- 第三节 参照完整性
- 第四节 用户定义的完整性
- 第五节 完整性约束命名子句
- 第六节 域中的完整性限制(了解)
- 第七节 触发器



## 第七节 触发器

❖ 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

- 由服务器自动激活
- 可以进行更为复杂的检查和操作，具有更精细和更强大的数据控制能力

# 第七节 触发器

❖ SQL SERVER 2005 触发器

❖ DML 触发器

❖ DDL 触发器

# SQL SERVER2005触发器

❖ 触发器是一种特殊的存储过程，它在执行事件时自动生效。SQL Server2005 包括两大类触发器：DML 触发器和 DDL 触发器。

- DML 触发器在数据库中发生数据操作语言 (DML) 事件时将启用。DML 事件包括在指定表或视图中修改数据的 INSERT 语句、UPDATE 语句或 DELETE 语句。DML 触发器可以查询其他表，还可以包含复杂的 Transact-SQL 语句。将触发器和触发它的语句作为可在触发器内回滚的单个事务对待。如果检测到错误（例如，磁盘空间不足），则整个事务即自动回滚。
- DDL 触发器是 SQL Server 2005 的新增功能。当服务器或数据库中发生数据定义语言 (DDL) 事件时将调用这些触发器。

# 触发器的作用

- ❖ (1) 触发器可以对数据库进行级联修改。
- ❖ (2) 实现比CHECK约束更为复杂的限制。
- ❖ (3) 比较数据修改前后的差别。
- ❖ (4) 强制表的修改要合乎业务规则。

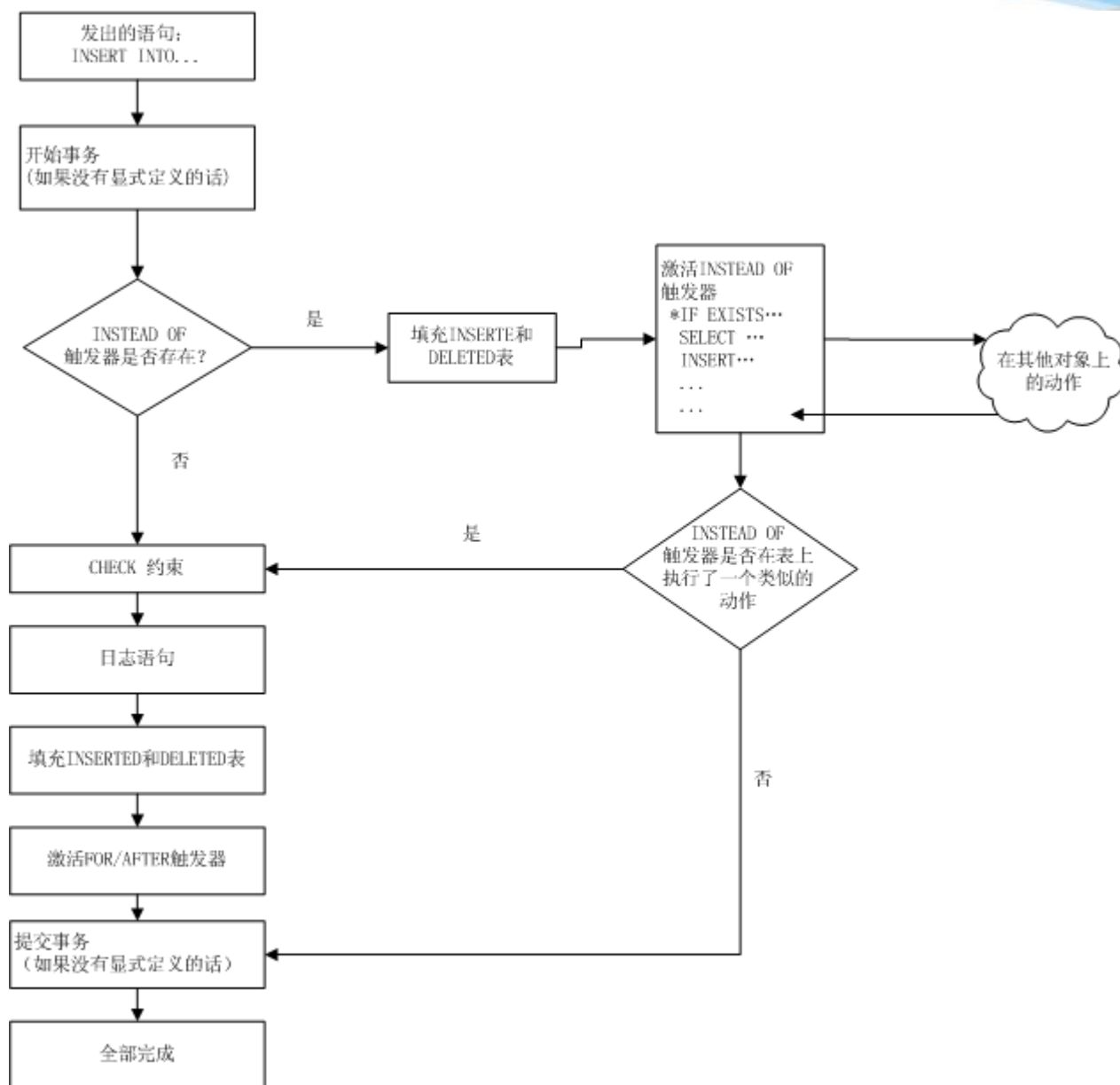
# Inserted表和Deleted表

- ❖ SQL Server 2005为每个触发器都创建了两个专用临时表：Inserted表和Deleted表。这两个表的结构总是与被该触发器作用的表的结构相同，触发器执行完成后，与该触发器相关的这两个表也会被删除。

激活触发器的动作	Inserted表	Deleted表
Insert	存放要插入的记录	
Update	存放要更新的记录	存放更新前的旧记录
Delete		存放要除的旧记录



# 触发器执行过程



# 第八节 SQL SERVER2008触发器

❖ SQL SERVER2008触发器

❖ DML触发器

❖ DDL触发器

# 创建DML触发器

❖ 创建DML触发器的语法格式为：

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ NOT FOR REPLICATION ]
AS
begin sql_statement [ ; ] end
```

# DML触发器示例

❖ 例1：在sc表上创建一个触发器，当插入、删除记录时给出提示信息

```
CREATE TRIGGER tr_sc
ON sc
FOR INSERT, DELETE
AS
begin
PRINT 'inserted表: '
Select * from inserted
PRINT 'deleted表: '
Select * from deleted
end
Go
```

```
--
CREATE TRIGGER tr_sc
ON students.sc
FOR INSERT, UPDATE, DELETE
AS
PRINT 'inserted表: '
Select * from inserted
PRINT 'deleted表: '
Select * from deleted
Go
insert into students.sc
values('200215135','5',90)
delete students.sc
where sno = '200215135' and
      cno = '5'
```

结果 消息

	sno	c...	grade
1	200215135	5	90

	sno	c...	grade
--	-----	------	-------

	sno	c...	grade
--	-----	------	-------

	sno	c...	grade
1	200215135	5	90

# 使用INSERT触发器

❖ INSERT触发器通常被用来更新时间标记字段，或者验证被触发器监控的字段中数据满足要求的标准，以确保数据的完整性

- 例：建立一个触发器，当向sc表中添加数据时，如果添加的数据与student表中的数据不匹配（没有对应的学号），则将此数据删除。

```
CREATE TRIGGER tr_sc_insert  ON students.sc
FOR INSERT
AS
BEGIN
    DECLARE @bh char(10)
    Select @bh=Inserted.sno from Inserted
    If not exists(select sno from students.student where student.sno=@bh)
        Delete students.sc where sno=@bh
END
```

❖ 例：创建一个触发器，当插入或更新成绩列时，该触发器检查插入的数据是否处于设定的范围内。

```
CREATE TRIGGER students.tr_sc_grade
  ON students.sc
  AFTER INSERT,UPDATE
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    DECLARE @score int;
    SELECT @score=inserted.grade from inserted
    IF (@score<0 or @score > 100)
    BEGIN
        RAISERROR ('成绩的取值必须在0到100之间', 16, 1)
        ROLLBACK TRANSACTION
    END
END
```

# 使用UPDATE触发器

❖ 当在一个有UPDATE触发器的表中修改记录时，表中原来的记录被移动到删除表中，修改过的记录插入到了插入表中，触发器可以参考删除表和插入表以及被修改的表，以确定如何完成数据库操作。

- 创建一个修改触发器，该触发器防止用户修改表student的学号。

```
CREATE TRIGGER students.tr_student_sno  ON students.student
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    if update(sno)
    begin
        raiserror('不能修改学号',16,10)
        rollback transaction
    end
END
```

# 使用DELETE触发器

❖ DELETE触发器通常用于两种情况，第一种情况是为了防止那些确实需要删除但会引起数据一致性问题的记录的删除，第二种情况是执行可删除主记录的子记录的级联删除操作。

- 例 建立一个与student表结构一样的表s1，当删除表student中的记录时，自动将删除掉的记录存放到s1表中。

```
CREATE TRIGGER [students].[tr_student_delete]
ON [students].[student]
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;
    insert into students.s1 select * from deleted
END
```



❖ 例 当删除表student中的记录时，自动删除表sc中对应学号的记录。

```
CREATE TRIGGER students.tr_student_sc_delete
  ON students.student
  AFTER DELETE
  AS
  BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    DECLARE @sno char(10)
    Select @sno=deleted.sno from deleted
    Delete students.sc where sno=@sno
  END
```

# INSTEAD OF触发器实例

## ❖ 在student表删除学生记录的同时删除学生的选课记录

```
ALTER TRIGGER [students].[tr_student_instead]
ON [students].[student]
instead of DELETE
AS
BEGIN
    SET NOCOUNT ON;
    delete students.sc
    where sno in ( select deleted.sno      from deleted  )

    delete students.student
    where sno in ( select deleted.sno      from deleted  )

END
```

# 第八节 SQL SERVER2008触发器

❖ SQL SERVER2008触发器

❖ DML触发器

❖ DDL触发器

# DDL触发器

- ❖ DDL 触发器会为响应多种数据定义语言 (DDL) 语句而激发。这些语句主要是以 CREATE、ALTER 和 DROP 开头的语句。DDL 触发器可用于管理任务，例如审核和控制数据库操作。
- ❖ DDL 触发器一般用于以下目的：
  - (1) 防止对数据库架构进行某些更改；
  - (2) 希望数据库中发生某种情况以响应数据库架构中的更改；
  - (3) 要记录数据库架构中的更改或事件。
- ❖ 仅在运行触发 DDL 触发器的 DDL 语句后，DDL 触发器才会激发。DDL 触发器无法作为 INSTEAD OF 触发器使用。

## ❖ 使用CREATE TRIGGER命令创建DDL触发器的语法形式如下

```
CREATE TRIGGER trigger_name  
ON {ALL SERVER|DATABASE}[WITH <ddl_trigger_option> [ ,...n ]]  
{FOR|AFTER} {event_type|event_group}[,...n]  
AS {sql_statement[;] [...n]|EXTERNAL NAME <method specifier>[;]}
```

其中：

<ddl\_trigger\_option>::=[ENCRYPTION] EXECUTE AS Clause]

<method\_specifier> ::= assembly\_name.class\_name.method\_name

# 例 使用 DDL 触发器来防止数据库中的任一表被修改或删除。

```
CREATE TRIGGER tr_edu_safety
ON DATABASE
FOR DROP_TABLE, ALTER_TABLE
AS
begin
    PRINT 'You must disable Trigger "safety" to drop or alter
tables!'
    ROLLBACK
end
```

# 管理SQL SERVER2008触发器

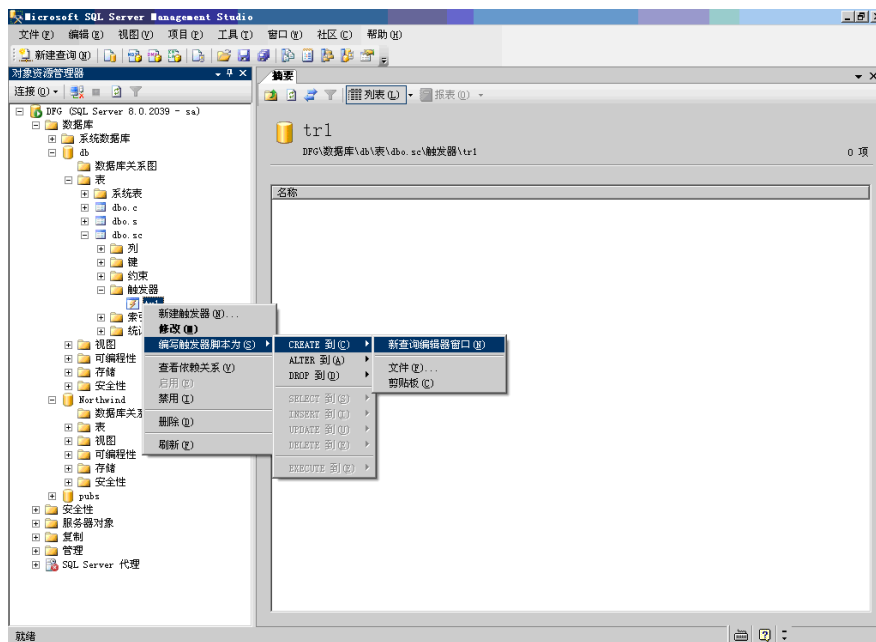
- ❖ 查看触发器
- ❖ 修改触发器
- ❖ 删除触发器

# 查看触发器

❖ 如果要显示作用于表上的触发器究竟对表有哪些操作，必须查看触发器信息。在SQL Server中，有多种方法可以查看触发器信息，其中最常用的有如下两种：

- （1）使用SQL Server管理平台查看触发器信息；

在SQL Server管理平台中，展开服务器和数据库，选择并展开表，然后展开触发器选项，右击需要查看的触发器名称，如右图所示，从弹出的快捷菜单中，选择“编写触发器脚本为→create到→新查询编辑器窗口”，则可以看到触发器的源代码。





# 使用系统存储过程查看触发器

❖ 系统存储过程sp\_help、sp\_helptext和sp\_depends分别提供有关触发器的不同信息。其具体用途和语法形式如下。

- sp\_help: 用于查看触发器的一般信息，如触发器的名称、属性、类型和创建时间。

  - sp\_help ‘触发器名称’

- sp\_helptext: 用于查看触发器的正文信息。

  - sp\_helptext ‘触发器名称’

- sp\_depends: 用于查看指定触发器所引用的表或者指定的表涉及到的所有触发器。

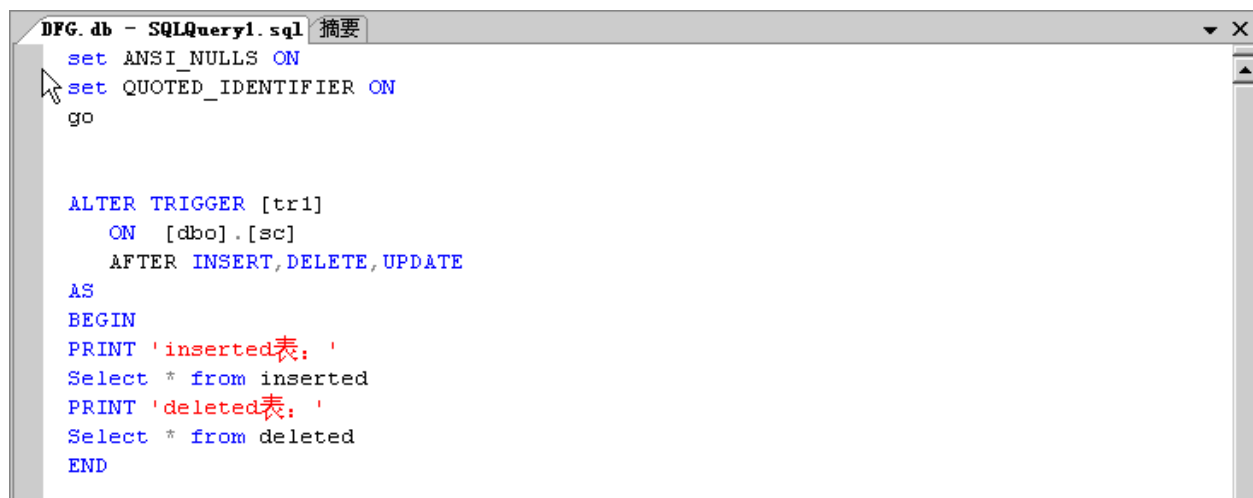
  - sp\_depends ‘触发器名称’

  - sp\_depends ‘表名’

# 修改触发器

❖ 通过SQL Server管理平台、存储过程，可以修改触发器的正文和名称。

- 1. 使用SQL Server管理平台修改触发器正文。
- 在管理平台中，展开指定的表，右击要修改的触发器，从弹出的快捷菜单中选择“修改”选项，则会出现触发器修改窗口，如下图所示。在文本框中修改触发器的SQL语句，单击“语法检查”按钮，可以检查语法是否正确，单击“执行”按钮，可以成功修改此触发器。



```
DFG.db - SQLQuery1.sql 摘要
set ANSI_NULLS ON
set QUOTED_IDENTIFIER ON
go

ALTER TRIGGER [trl]
ON [dbo].[sc]
AFTER INSERT,DELETE,UPDATE
AS
BEGIN
PRINT 'inserted表: '
Select * from inserted
PRINT 'deleted表: '
Select * from deleted
END
```

### ❖修改DML触发器的语法形式如下：

```
ALTER TRIGGER schema_name.trigger_name
ON (table|view)
[WITH <dml_trigger_option>[,...n]]
(FOR|AFTER|INSTEAD OF)
{[DELETE][,][INSERT][,][UPDATE]}
[NOT FOR REPLICATION]
AS {sql_statement[;][...n]|EXTERNAL NAME <method specifier>[;]}
<dml_trigger_option>::=[ENCRYPTION][&lEXECUTE AS Clause >]
<method_specifier> ::=assembly_name.class_name.method_name
```

### ❖修改DDL触发器的语法形式如下：

```
ALTER TRIGGER trigger_name
ON {DATABASE|ALL SERVER}[WITH <ddl_trigger_option> [,...n]]
{FOR|AFTER}{event_type[,...n]|event_group}
AS {sql_statement[;]|EXTERNAL NAME <method specifier> [;]}
<ddl_trigger_option>::=[ENCRYPTION][&lEXECUTE AS Clause > ]
<method_specifier> ::=assembly_name.class_name.method_name
```

# 删除触发器

❖ 由于某种原因，需要从表中删除触发器或者需要使用新的触发器，这就必须首先删除旧的触发器。只有触发器所有者才有权删除触发器。删除已创建的触发器有三种方法：

- （1）使用系统命令DROP TRIGGER删除指定的触发器。其语法形式如下：

DROP TRIGGER { trigger } [ ,...n ]

- （2）删除触发器所在的表。删除表时，SQL Server将会自动删除与该表相关的触发器。
- （3）在SQL Server管理平台中，展开指定的服务器和数据库，选择并展开指定的表，右击要删除的触发器，从弹出的快捷菜单中选择“删除”选项，即可删除该触发器。

# 小结

❖ 数据库的完整性是为了保证数据库中存储的数据是正确的

❖ RDBMS完整性实现的机制

- 完整性约束定义机制
- 完整性检查机制
- 违背完整性约束条件时RDBMS应采取的动作

# Q & A

❖ 约束和触发器各有什么优缺点？



# 这次课我们学到了…

- 数据库实体完整性定义、违约处理
- 参照完整性定义、违约处理
- 用户自定义完整性
  - 属性
  - 元组
- 完整性命名子句
- 触发器
  - 定义、激活、删除

# 作业

❖ 本章实验

❖ 课后练习题 6





# 休息…

吾日三省吾身。为  
人谋而不忠乎？与朋  
友交而不信乎？传不  
习乎？



红梅  
丁巳年  
小鎮  
畫