

# 第五章 数据库完整性

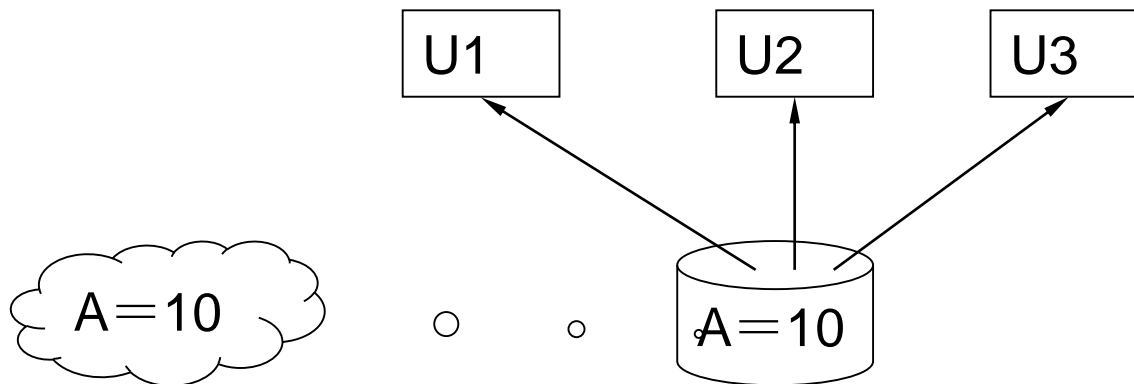
*Principles of Database Systems*



## ■ 数据库的完整性:

数据的**正确性**和**相容性**。

- **正确性**指数据库中数据与现实世界的实际情况是相符的。
- **相容性**指数据库中数据自身不存在自相矛盾的现象。



若现实世界的A与DB中的A保持一致，且U1、U2、U3等所有用户从DB中查询的结果均为10，则称A具有完整性，否则就是不完整的。

# 数据库完整性

- 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。

# 数据库完整性

## DBMS的完整性控制功能:

### 1. 为用户提供定义完整性约束条件的机制

DBMS将这些约束条件作为模式的一部分存入数据库中。

### 2. 监督系统中执行的操作是否违反完整性限制条件

应进行完整性检查的操作有**INSERT/DELETE/UPDATE**。

### 3. 对违反完整性约束的情况进行相应处理

**拒绝执行 / 级联操作**

**思考：**完整性控制由DBMS实现和由应用程序实现有什么区别？  
哪种方式更好？



# 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 完整性约束命名字句

## \*5.5 域中的完整性限制

## 5.6 断言

## 5.7 触发器

## 5.8 小结

## 5.1 实体完整性

实体是客观存在且**相互区别**的事物。

### 1. 如何定义实体完整性?

实体完整性在关系模型中通过关系的主码 (**PRIMARY KEY**) 来体现。

PRIMARY KEY在CREATE TABLE语句中定义。

- **方法一**：在**列定义**子句中说明（列级约束）。

```
CREATE TABLE COURSE (  
    cno CHAR(8) PRIMARY KEY,  
    cname CHAR(30),  
    credit SMALLINT  
);
```

## 5.1 实体完整性

- **方法二**：用单独的**表约束**子句说明。（表级约束）

```
CREATE TABLE COURSE (  
    cno CHAR(8),  
    cname CHAR(30),  
    credit SMALLINT,  
    PRIMARY KEY (cno)  
);
```

如果主码由多个列组成，则**只能**用第二种方法定义主码。

```
CREATE TABLE SC (  
    sno CHAR(8),  
    cno CHAR(8),  
    grade SMALLINT,  
    PRIMARY KEY(sno, cno)  
);
```

# 5.1 实体完整性

## 2. 如何检查实体完整性

对基表**插入**数据或对**主码**列进行**更新**时，DBMS将自动对该操作进行实体完整性检查。包括：

- ❖ 主码值是否唯一，否则拒绝执行该操作；
- ❖ 各主码列的值是否为空，是则拒绝执行该操作。

检查记录中主码值是否唯一的一种方法是进行**全表扫描**。

索引是整表的“目录”，有提高数据访问速度的作用：

- ❖ 索引比整表小得多，所以查索引比在整表中查询要快；
- ❖ 随机查询时利用索引可以快速定位整表中的元组；
- ❖ 有些查询如果只涉及索引项，则可避免读整表。



- 检查记录中主码值是否唯一的一种方法是进行**全表扫描**

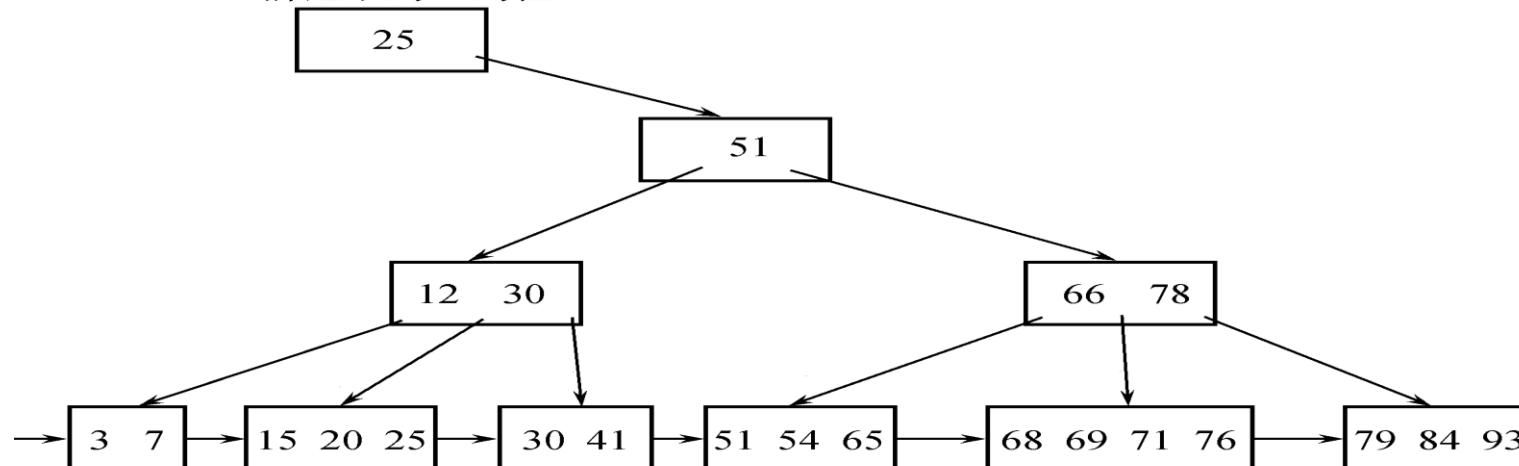
待插入记录

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
⋮				

新记录的主码值



## 5.2 参照完整性

引用数据与被引用数据应该是一致的。

### ❖ 如何定义参照完整性

参照完整性在关系模型中通过关系的外码 (**FOREIGN KEY**) 来体现。

**FOREIGN KEY**同样在CREATE TABLE语句中定义。

#### ■ 方法一：在列定义子句中说明（列级约束）。

```
CREATE TABLE SC (  
    sno CHAR(8) REFERENCES STUDENT(sno),  
    cno CHAR(8) REFERENCES COURSE(cno),  
    grade SMALLINT,  
    PRIMARY KEY(sno, cno)  
);
```

## 5.2 参照完整性

- **方法二**：用单独的表约束子句说明。（**表级约束**）

```
CREATE TABLE SC (  
    sno CHAR(8),  
    cno CHAR(8),  
    grade SMALLINT,  
    PRIMARY KEY(sno, cno),  
    FOREIGN KEY(sno) REFERENCES STUDENT(sno),  
    FOREIGN KEY(cno) REFERENCES COURSE(cno)  
);
```

在参照完整性约束中，参照关系（即外码所在的关系，如SC），称为**子表**；被参照关系（如STUDENT和COURSE）称为**父表**。

## 5.2 参照完整性

### 2. 如何检查参照完整性

#### 外码的取值限制:

- NULL (思考: 如果该外码同时又是子表的主码呢? )
- 引用父表中的某值

#### 可能破坏参照完整性的情况及违约处理:

被参照表 (如Stu)	参照表 (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

## 5.2 参照完整性

### 1) 在子表中插入元组

仅当父表中存在与插入元组相应的元组时，系统执行在子表中插入元组的操作，否则拒绝此插入操作。

**例：**假设STUDENTS表的cno引用了CLASS表的cno。

在STUDENTS表中插入一条cno = ' 01' 的学生记录，如果CLASS表中存在cno = ' 01' 的记录，则执行；如果不存在，则拒绝执行。

### 2) 修改子表外码

如果父表中存在待修改值，则执行；否则不允许执行此类修改操作。

**例：**Sno = ' 991001' 的学生的cno = ' 05' ，将其cno改为' 01' ，已知CLASS表中存在cno = ' 01' 但不存在cno = ' AA' 的记录  
则执行；如果改为 'AA' ，则拒绝执行。

## 5.2 参照完整性

### 3) 修改父表主码 (若子表中有相关参照记录)

- **拒绝修改**: 拒绝执行此类操作。
- **级联修改**: 将子表中相关记录在外码上的值一起自动修改。

**例**: 要将CLASS表中的Clno = ' 01' 改为' 101' ,  
则由DBMS自动将STUDENTS表中的所有Clno = ' 01' 的记录的Clno都修改为' 101' 。

- **置空修改**: 将子表中相关记录在外码上的值全部置为NULL。

**例**: 要将CLASS表中的Clno = ' 01' 改为' 101' ,  
则由DBMS自动将STUDENTS表中的所有Clno = ' 01' 的记录的Clno属性置NULL。

## 5.2 参照完整性

### 4) 删除父表元组（若子表中有相关参照记录）

- **拒绝删除:** 发出警告，拒绝执行此类操作。
- **级联删除:** 删除父表中元组的同时，自动删除子表中的相关元组。

**例:** 删除CLASS表cIno = ' 01' 的元组时，  
DBMS自动将STUDENTS表中所有cIno = ' 01' 的元组一起删除。

- **置空删除:** 删除父表中元组的同时，自动将子表中的相关元组的外码置NULL值。

**例:** 删除CLASS表cIno = ' 01' 的元组时，  
DBMS自动将STUDENTS表中所有cIno = ' 01' 的元组的cIno改为NULL。

## 5.2 参照完整性

如何指定修改或删除父表时的违约处理动作？

在定义参照完整性约束时，可同时指定删除或修改时的违约处理动作，如**NOT ACTION**、**SET NULL**或**CASCADE**。缺省动作为**NOT ACTION**

。

形如：

```
CREATE TABLE SC (
```

```
.....
```

```
FOREIGN KEY(sno) REFERENCES STUDENT(sno) ON DELETE  
SET NULL
```

```
ON UPDATE CASCADE,
```

```
.....
```

```
);
```



[例] 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*/
```

```
);
```

## 5.3 用户定义的完整性约束

在一个具体应用中，数据需满足一些特定的语义要求。

### 1) 空值约束 (列级)

指定属性的值不允许为空值 (NOT NULL)

```
CREATE TABLE COURSE (  
    cno CHAR(8),  
    cname CHAR(30) NOT NULL,  
    credit SMALLINT,  
    PRIMARY KEY (cno)  
);
```

## 5.3 用户定义的完整性约束

### 2) 唯一性约束 (列级/表级)

指定属性或属性组的值必须唯一 (UNIQUE)

方法一: CREATE TABLE COURSE (  
    cno CHAR(8),  
    cname CHAR(30) NOT NULL **UNIQUE**,  
    credit SMALLINT,  
    PRIMARY KEY (cno)  
); /\* 列级约束 \*/

方法二: CREATE TABLE COURSE (  
    cno CHAR(8),  
    cname CHAR(30) NOT NULL,  
    credit SMALLINT,  
    PRIMARY KEY (cno),  
    **UNIQUE(cname)**  
); /\* 表级约束 \*/

## 5.3 用户定义的完整性约束

### 3) CHECK约束 (列级/表级)

检查一个属性或多个属性的取值是否满足一个布尔表达式。

方法一：CREATE TABLE SC (  
    sno CHAR(8),  
    cno CHAR(8),  
    grade SMALLINT CHECK(grade <=100),  
    .....  
); /\* 列级约束 \*/

方法二：CREATE TABLE EMPLOYEE(  
    .....  
    age SMALLINT,  
    work\_year SMALLINT,  
    CHECK(age>work\_year),  
    .....  
);/\* 表级约束 \*/

## 5.4 完整性约束命名子句

- ❖ 完整性约束与列一样是关系模式的构成元素。
- ❖ 关系中的列是必须命名的，但是约束却不一定要命名。

思考：为什么列必须命名而约束可以不命名？约束不命名可能带来什么问题？

- ❖ 如何给完整性约束命名

约束命名子句

**CONSTRAINT** <约束名> <约束说明>

其中，<约束说明> 可以是**PRIMARY KEY**子句，**FOREING KEY** 子句，**NOT NULL**子句，**UNIQUE**子句和**CHECK**子句。

## 5.4 完整性约束命名子句

在CREATE TABLE语句中使用约束命名子句为约束命名：

```
CREATE TABLE SC (  
    sno CHAR(8),  
    cno CHAR(8),  
    grade SMALLINT  
    CONSTRAINT C1 CHECK(grade<=100),  
    CONSTRAINT PK PRIMARY KEY(sno, cno),  
    CONSTRAINT FK1 FOREIGN KEY(sno)  
        REFERENCES STUDENT(sno),  
    CONSTRAINT FK2 FOREIGN KEY(cno) REFERENCES COURSE(cno)  
);
```

## 5.4 完整性约束命名子句

- 在ALTER TABLE语句中通过约束名删除约束：  
**ALTER TABLE SC DROP CONSTRAINT C1;**
- 在ALTER TABLE语句中使用约束命名子句增加新的约束：  
**ALTER TABLE S**  
**ADD CONSTRAINT C2**  
**CHECK(age >= 15 AND age <= 30) ;**

# 例：SQLServer命名约束检测

```
CREATE TABLE Parent
  (pkey1 INT NOT NULL
  CONSTRAINT pk_Parent PRIMARY KEY (pkey1))
GO
CREATE TABLE ConstraintName
  (Pkey INT NOT NULL
  CONSTRAINT pk_CnstNm primary key,
  Parent_pkey1 INT NOT NULL,
  col1 INT NULL
  CONSTRAINT ck_CnstNm_col1 CHECK (col1 IN ( 'a','b' ) )
  CONSTRAINT df_CnstNm_col1 DEFAULT 1,
  CONSTRAINT fk_Parent_CnstNm FOREIGN KEY (Parent_pkey1)
  REFERENCES Parent (pkey1)
  )
GO
exec sp_helpconstraint ConstraintName //存储过程查看表ConstraintName上的约束
GO
DROP TABLE ConstraintName
GO
```

请自行上机检查，  
带命名约束 和  
非命名约束  
的差异



# SQLServer命名约束检测 (续)



```
CREATE TABLE ConstraintName  
  (Pkey INT NOT NULL primary key,  
   Parent_pkey1 INT NOT NULL  
   FOREIGN KEY (Parent_pkey1) REFERENCES PARENT(pkey1),  
   col1 INT NULL  
   CHECK (col1 IN ( 'a','b' ) )  
   DEFAULT 1  
  )  
GO
```

```
exec sp_helpconstraint ConstraintName  
GO
```

```
DROP TABLE ConstraintName  
GO
```

```
CREATE TABLE ConstraintName  
  (Pkey INT NOT NULL primary key,  
   Parent_pkey1 INT NOT NULL  
   FOREIGN KEY (Parent_pkey1) REFERENCES PARENT(pkey1),  
   col1 INT NULL  
   CHECK (col1 IN ( 'a','b' ) )  
   DEFAULT 1  
  )  
GO
```

```
exec sp_helpconstraint ConstraintName  
GO  
DROP TABLE ConstraintName  
GO  
DROP TABLE Parent  
GO
```

## 5.5 域中的完整性限制

- SQL支持域的概念，并可以用CREATE DOMAIN语句建立一个域以及该域应该满足的完整性约束条件。

[例] 创建一个名为GenderDomain的域，它是值为 '男' '女' 字符

```
CREATE DOMAIN GenderDomain CHAR(2)  
CHECK (VALUE IN ('男', '女'));
```

对Ssex的说明可以改写为: Ssex GenderDomain,

[例] 建立一个性别域GenderDomain，并对其中的限制命名为GD

```
CREATE DOMAIN GenderDomain CHAR(2)  
CONSTRAINT GD CHECK (VALUE IN ('男', '女'));
```

[例] 删除域GenderDomain的限制条件GD。

```
ALTER DOMAIN GenderDomain DROP CONSTRAINT GD;
```

[例] 在域GenderDomain上增加限制条件GDD。

```
ALTER DOMAIN GenderDomain  
ADD CONSTRAINT GDD CHECK (VALUE IN ('1', '0'));
```

- 上两步将性别的取值范围由('男', '女')改为 ('1', '0')

## 5.7 触发器

### 什么是触发器？

触发器是定义在基表上的一种数据库对象，它指定：在执行对表的某些操作的时候，另一些操作也同时被执行。

### 定义一个触发器应包含哪些要素？

- 触发器的名字
- 触发器关联的表名
- 哪些操作执行时将引发其它操作被执行（触发事件）
- 被触发后执行的操作（触发动作）
- 执行触发动作的时机

## 5.7.1 定义触发器

**CREATE TRIGGER** <触发器名>  
[**BEFORE** | **AFTER**] <触发事件> **ON** <表名>  
**FOR EACH** {**ROW** | **STATEMENT**}  
[**WHEN** <触发条件>]  
<触发动作体>;

### 说明:

- <触发事件>可以是**INSERT**、**DELETE**或**UPDATE**，或这些事件的组合，**UPDATE**后面可以带**OF** <触发列, ...>，指明只当哪些列被修改时才激活触发器。**BEFORE**或**AFTER**表示触发器是在触发事件之前还是之后被激活。
- **FOR EACH ROW**表示该触发器为行级触发器，触发事件每影响一条元组都将激活一次触发器。
- **FOR EACH STATEMENT**表示该触发器为语句级触发器，触发事件只激活一次触发器。

## 5.7.1 定义触发器

DBMS如何执行触发器:

- 触发器被激活后, 先检查<触发条件>, 当其为真时, <触发动作体>才执行; 如果没有指定<触发条件>, 则<触发动作体> 在触发器被激活后立即执行。
- <触发动作体>是一段程序, 如果触发器是行级触发器, 则这段程序中还可以使用 NEW和OLD分别引用触发事件发生前后的元组值。

例: 保留表BOOKS中被删除的书的记录。

```
CREATE TRIGGER BOOKS_DELETE  
AFTER DELETE ON BOOKS  
FOR EACH ROW  
INSERT INTO BOOKS_DELETED_LOG  
VALUES(OLD.TITLE);
```

## 5.7.1 定义触发器

**例：** 教授的工资不得低于3000元，否则自动改为3000元。

```
CREATE TRIGGER Update_Sal  
  BEFORE INSERT OR UPDATE OF Sal, Pos  
  ON Teacher  
  FOR EACH ROW  
  WHEN ( NEW.Pos = '教授' AND NEW.Sal < 3000 )  
    NEW.Sal := 3000;
```

下列语句执行时将产生什么结果？

- A. INSERT INTO Teacher(Sal, Pos) VALUES(2000, '讲师' )
- B. INSERT INTO Teacher(Sal, Pos) VALUES(2500, '教授' )
- C. UPDATE 教师A的职称为教授，且教师A的工资为2000
- D. UPDATE 教师A的工资为3500，且该教师的职称为教授

## 5.7.1 定义触发器示例

**[例]** 定义AFTER行级触发器，当教师表Teacher的工资发生变化后就自动在工资变化表Sal\_log中增加一条相应记录

首先建立工资变化表Sal\_log

```
CREATE TABLE Sal_log
```

```
(Eno  NUMERIC(4) references teacher(eno),
```

```
Sal  NUMERIC(7, 2),
```

```
Username char(10),
```

```
Date  TIMESTAMP
```

```
);
```

## 5.7.1 定义触发器示例(续)

```
2) CREATE TRIGGER Insert_Sal
    AFTER INSERT ON Teacher    /*触发事件是INSERT*/
    FOR EACH ROW
    AS BEGIN
        INSERT INTO Sal_log VALUES(
            new.Eno, new.Sal, CURRENT_USER,
            CURRENT_TIMESTAMP);

    END;

3) CREATE TRIGGER Update_Sal
    AFTER UPDATE ON Teacher    /*触发事件是UPDATE */
    FOR EACH ROW
    AS BEGIN
        IF (new.Sal <> old.Sal) THEN INSERT INTO Sal_log
            VALUES(new.Eno, new.Sal, CURRENT_USER,
                CURRENT_TIMESTAMP);

        END IF;
    END;
```

计算机学院数据库所 Zuo



## 5.6.2 激活触发器

- 触发器的执行，是由触发事件激活的，并由数据库服务器自动执行
- 一个数据表上可能定义了多个触发器
  - 同一个表上的多个触发器激活时遵循如下的执行顺序：
    - (1) 执行该表上的BEFORE触发器；
    - (2) 激活触发器的SQL语句；
    - (3) 执行该表上的AFTER触发器。
  - 有多个触发器时，这些触发器按创建时间顺序依次执行。
  - 任一触发器的执行失败都将中止整个操作。
- [例] 执行修改某个教师工资的SQL语句，激活上述定义的触发器。

UPDATE Teacher SET Sal=800 WHERE Ename='陈平';

执行顺序是：

执行触发器Insert\_Or\_Update\_Sal

执行SQL语句 “UPDATE Teacher SET Sal=800 WHERE  
Ename='陈平';”

执行触发器Insert\_Sal;

执行触发器Update\_Sal

### 3. 触发器的撤销

**DROP TRIGGER <触发器名> ON <表名>;**

**说明：**触发器必须是一个已经创建的触发器，并且只能由具有相应权限的用户删除。

**例：**DROP TRIGGER Insert\_Sal ON Teacher;

**注：**触发器是与表相关联的，因此，表的撤销将引起该表上的所有触发器同时被撤销。

不同的DBMS对于触发器的支持方式和语法是有所区别的。这是因为SQL标准直到SQL99才将触发器纳入标准，而在此之前几乎所有主流的商用DBMS都已经实现了对触发器的支持。

- **DML触发器：**
  - Insert 触发器
  - Update触发器
  - Delete触发器
- **DDL触发器：**
  - 当DD事件被激活，使用DDL触发器防治对数据架构进行的某些更改或记录数据中的更改或事件操作。
- **登录触发器：**
  - 响应LOGIN事件而激发存储过程。与SQL Server实例建立用户会话时引发此事件。

- 触发器触发时：
  - 系统自动在内存中创建 **deleted** 表或 **inserted** 表；
  - 只读，不允许修改，触发器执行完成后，自动删除。
- Inserted和deleted表对照：

修改操作记录	inserted表	deleted表
增加( <b>insert</b> )记录	存放新增的记录	.....
删除( <b>deleted</b> )记录	.....	存放被删除的记录
修改( <b>update</b> )记录	存放更新后的记录	存放更新前的记录

# SQL Server触发器



- 触发器语法:

**CREATE TRIGGER trigger\_name**

**ON table\_name**

**[WITH ENCRYPTION]**

**FOR | AFTER | INSTEAD OF [DELETE, INSERT, UPDATE]**

**AS**

**T-SQL语句**

**GO**

- --查看数据库中所有的触发器

**use 数据库名**

**go**

**select \* from sysobjects where xtype='TR'**

- --查看触发器内容: **exec sp\_helptext '触发器名称'**

# 例：创建insert触发器

```
create trigger trig_insert
on student
after insert
as
begin
    if object_id(N'student_sum',N'U') is null--判断student_sum表是否存在
        create table student_sum(stuCount int default(0));--创建存储学生人数的
        student_sum表
    declare @stuNumber int;
    select @stuNumber = count(*)from student;
    if not exists (select * from student_sum)--判断表中是否有记录
        insert into student_sum values(0);
    update student_sum set stuCount = @stuNumber;
        --把更新后总的学生数插入到student_sum表中
end
```

## 例：带游标的触发器

Create trigger tri\_wk\_CSVHead\_History on wk\_CSVHead\_History  
instead of insert as

begin

**declare YB cursor** --声明一个游标

for

select NoteNO from inserted--这里的NoteNO要和下面的匹配

**open YB**

**declare @NoteNO** varchar(50)--定义一个游标变量@NoteNO,用来操作insered表中的NoteNo.

**fetch next from YB into @NoteNO**--移动游标

**while(@@fetch\_status=0)**--0操作成功, -1 FETCH 语句失败或此行不在结果集中, -2 被提取的行不存在

begin

delete from wk\_CSVDetail\_History where NoteNO=@NoteNO

delete from wk\_CSVHead\_History where NoteNO=@NoteNO

fetch next from YB into @NoteNO --继续移动游标, 直到@@fetch\_status不等于0时.

**end**

**close YB** --关闭游标

**deallocate YB** --释放游标

insert into wk\_CSVHead\_History select \* from inserted

end

# 触发器的作用

- 强制实现由主键和外键所不能保证的**参照完整性和数据的一致性**；
- 实现比CHECK语句更为复杂的约束（涉及多表）；
- **找到**数据修改前后表状态的**差异**，并基于此差异采取行动；
- 级联运行修改数据库中相关表；
- 跟踪变化，禁止不合规则的操作；
- 返回自定义的错误消息（约束做不到）；
- 自动调用存储过程。



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。  
为此，RDBMS提供了一套完整性机制，即：

- 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。
- 关系数据库中的三类完整性约束为：
  - 实体完整性约束（由主键约束来实现）
  - 参照完整性约束（由外键约束来实现）
  - 用户定义的完整性约束（NOT NULL, UNIQUE, CHECK, 触发器）
- 触发器是由事件驱动的特殊过程，它的功能非常强大，并不仅限于完整性控制，目前被广泛应用于数据库应用中。