第五章 数据库完整性

学习要点:

- 了解数据库完整性概念
- 掌握关系数据库完整定义、检查和处理方法
- 了解约束子句、域完整性定义及触发器概念

一个数据库的服务质量首先应当是其所提供的数据质量。

» 数据库完整性的定义

– 数据的正确性

● 是指数据是符合现实世界语义，反映了当前实际状况的。

– 数据的相容性

● 是指数据库同一对象在不同关系表中的数据是符合逻辑的。

例如，

● 学生的学号必须唯一

● 性别只能是男或女

DBMS为维护数据库完整性必须提供以下三种功能：

①定义功能：提供定义完整性约束条件的机制。

②检查功能：检查用户发出的操作请求是否违背了完整性约束条件。

③违约处理功能：如果发现用户的操作请求使数据违背了完整性约束条件，则采取一定的动作(如拒绝等)来保证数据的完整性。

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 完整性的约束命名子句

5.5 域中的完整性限制

5.6 断言

5.7 触发器

学生-课程数据库

学生表

Student(Sno,Sname,Sgender,Sage,Sdept)

课程表

Course(Cno,Cname,Cpno,Ccredit)

学生选课表

SC(Sno,Cno,Grade)

5.1 实体完整性

5.1.1 实体完整性定义

5.1.2 实体完整性检查和违约处理

5.1 实体完整性定义

回顾

- 规则： 主属性非空，主码取值惟一。
- 实体完整性规则：若属性*A*是基本关系*R*的主属性，则属性*A*不能取空值。

» 关系模型的实体完整性

– CREATE TABLE中用PRIMARY KEY定义

```
CREATE TABLE Student
  (Sno          CHAR (5)  PRIMARY KEY,
 Sname       CHAR (20)   UNIQUE,
 Sgender     CHAR (1) ,
 Sage          INT,
 Sdept        CHAR (15)
  ) ;

 CREATE TABLE SC
(Sno     CHAR(5),
 Cno     CHAR(3),
 Grade   INT,
  Primary Key (Sno, Cno)
 );
```

5.1.2实体完整性检查和违约处理

当用户对基本表插入一条记录或对主码进行修改操作时，RDBMS将进行如下自动检查：

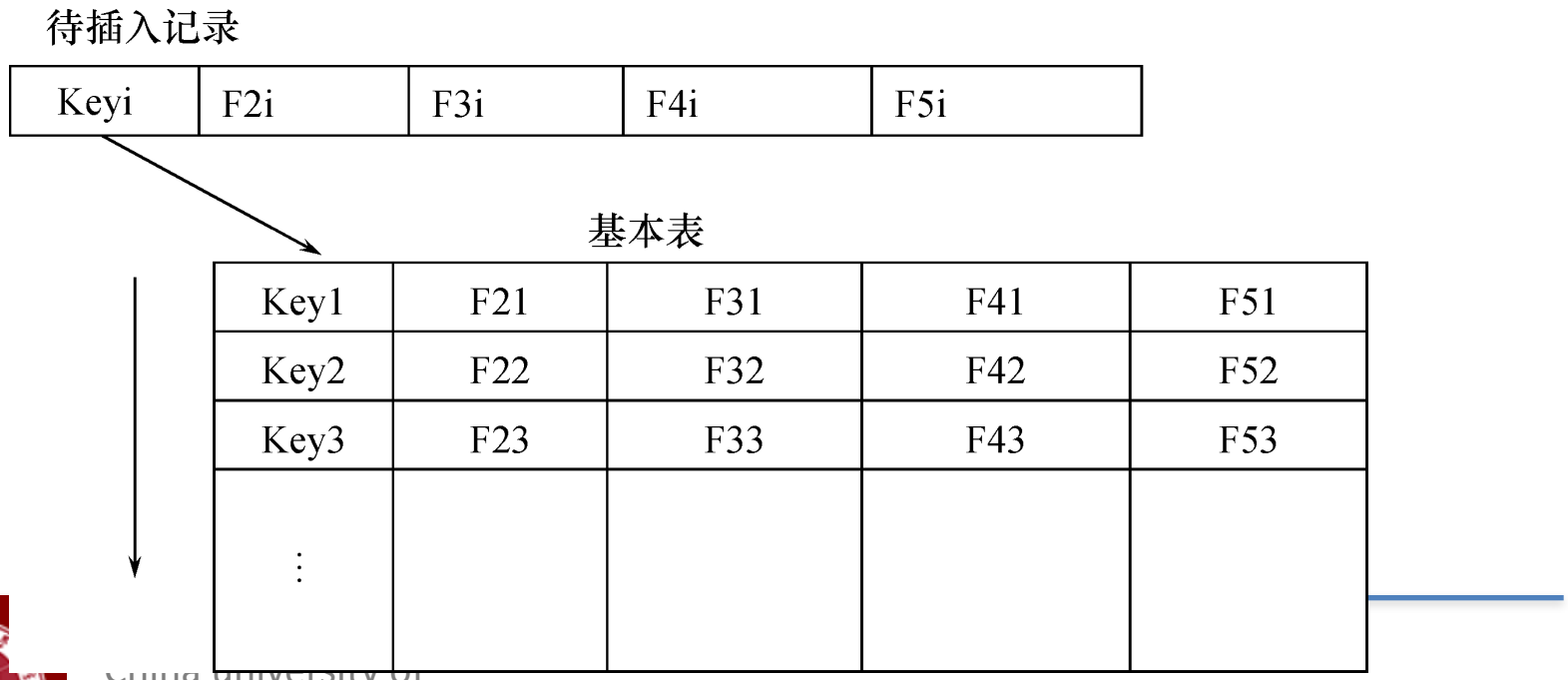
① 检查主码是否唯一，如果不唯一则拒绝插入或修改。

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。

实体完整性检查和违约处理（续）

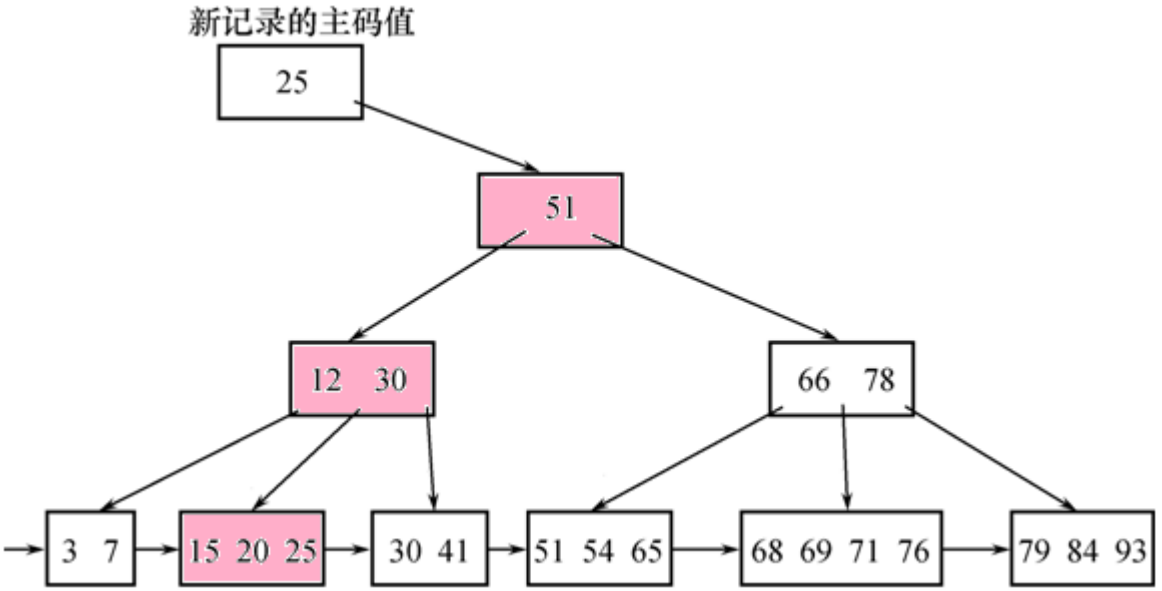
» 检查记录中主码值是否唯一的一种方法是进行全表扫描

■ 依次判断表中每一条记录的主码值与将插入记录上的主码值（或者修改的新主码值）是否相同



实体完整性检查和违约处理（续）

» B+树索引



例如，

– 新插入记录的主码值是25

- ●通过主码索引，从B+树的根结点开始查找
- ●读取3个结点：根结点（51）、中间结点（12 30）、叶结点（15 20 25）
- ●该主码值已经存在，不能插入这条记录

5.2 参照完整性

5.2.1 参照完整性定义

5.2.2 参照完整性检查和违约处理

5.2.1 参照完整性的定义

回顾

参照完整性规则

- 若属性（或属性组）*F*是基本关系*R*的外码，它与基本关系*S*的主码K_s相对应（基本关系*R*和*S*不一定是不同的关系），则对于*R*中每个元组在*F*上的值必须为：
  - 或者取空值
  - 或者等于*S*中某个元组的主码值

» 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

5.2 参照完整性

5.2.1 参照完整性定义

例6：定义SC表的参照完整性。

```
CREATE  TABLE  SC
(Sno CHAR(5)  NOT NULL,
 Cno CHAR(3)  NOT NULL,
 Grade   INT,
 Primary Key(Sno,Cno),
 Foreign Key(Sno) Refrences Student(Sno),
 Foreign Key(Cno) References Course(Cno)
);
```

5.2.2参照完整性检查和违约处理

» 一个参照完整性将两个表中的相应元组联系起来

» 对被参照表和参照表进行增删改操作时有可能破坏参照完整性，必须进行检查

» 例如，对表SC和Student有四种可能破坏参照完整性的情况 ：

– SC表中增加一个元组，该元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。

– 修改SC表中的一个元组，修改后该元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。

» 例如，对表SC和Student有四种可能破坏参照完整性的情况 （续）：

– 从Student表中删除一个元组，造成SC表中某些元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。

– 修改Student表中一个元组的Sno属性，造成SC表中某些元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等 。

5.2.2 参照完整性检查和违约处理

被参照表	参照表	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

5.2.2 参照完整性检查和违约处理

- 拒绝(no action)：不允许执行该操作。常为默认策略。
- 级连(cascade)操作：当删除或修改被参照表元组造成了与参照表不一致，则删除或修改参照表中的所有不一致的元组。
- 设置为空值(set null)：当删除或修改被参照表元组造成了与参照表不一致，则将参照表中的所有不一致的元组的对应属性设置为空值(不破坏实体完整性的情况下)。

例如，有下面2个关系

学生（学号，姓名，性别，专业号，年龄）

专业（专业号，专业名）

- 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
- 对应语义：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

例8：显式说明参照完整性的违约处理。

```
CREATE TABLE SC
(Sno CHAR(5),
 Cno CHAR(1),
 Grade   INT,
 Primary Key(Sno,Cno),
 Foreign Key(Sno) References Student(Sno)
    on delete cascade on update cascade,
 Foreign Key(Cno) References Course(Cno)
    on delete no action on update cascade);
```

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 完整性的约束命名子句

5.5 域中的完整性限制

5.6 触发器

5.3 用户定义的完整性

- 用户定义的完整性是针对某一具体关系数据库的约束条件，反映某一具体应用所涉及的数据必须满足的语义要求。

5.3 用户定义的完整性

5.3.1 属性上的约束条件的定义

- 在CREATE TABLE中定义属性的同时可以根据应用要求，定义属性上的约束条件，即属性值限制：
  - 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 ( )

5.3 用户定义的完整性

5.3.1 属性上的约束条件的定义

例10：在定义SC表时，说明Grade属性不允许取空值，缺省值0。

```
CREATE TABLE SC
(Sno CHAR(5),
 Cno CHAR(3),
 Grade   INT NOT NULL DEFAULT 0,
 Primary Key(Sno,Cno)
);
```

UNIQUE(唯一约束)

用于指明基本表在某一列或多个列的组合上的取值必须唯一。定义了UNIQUE约束的那些列称为唯一键，系统自动为唯一键建立唯一索引，从而保证了唯一键的唯一性。唯一键允许为空，但系统为保证其唯一性，最多只可以出现一个NULL值。

在建立UNIQUE约束时，要考虑以下因素：

- 使用UNIQUE约束的字段允许为NULL值；
- 一个表中可以允许有多个UNIQUE约束；
- 可以把UNIQUE约束定义在多个字段上；
- UNIQUE约束用于强制在指定字段上创建一个UNIQUE索引。

PRIMARY KEY约束与UNIQUE约束类似，通过建立唯一索引来保证基本表在主键列取值的唯一性，二者区别：

- 在一个基本表中只能定义一个PRIMARY KEY约束，但可以定义多个UNIQUE约束。
- 对于指定为PRIMARY KEY列的一个列或多个列的组合，其中任何一个列都不能出现NULL值，而对于UNIQUE所约束的唯一键，允许NULL。
- 不能为同一个列或一组列既定义UNIQUE约束，又定义PRIMARY KEY约束。

CHECK约束

CHECK约束用来检查属性值所允许的范围，以此来保证域的完整性。

在建立CHECK时，需要考虑以下几个因素：

- 一个表中可以定义多个CHECK约束；
- 每个属性只能定义一个CHECK约束；
- 在多个字段上定义的CHECK约束必须为表约束；
- 当执行INSERT，UPDATE语句时，CHECK约束将验证数据。

CHECK约束既可用于列约束，也可用于表约束，其语法：

[CONSTRAINT <约束名>] CHECK(<条件>)

5.3 用户定义的完整性

5.3.1 属性上的约束条件的定义

例11：Student表的Sgender只许取“男”或“女”。

```
CREATE TABLE Student
(Sno      CHAR(9)   Primary Key,
 Sname    CHAR(8)  Unique,
 Sgender CHAR(2)
            CHECK(Sgender IN ('男','女')),
 Sage SMALLINT,
 Sdept CHAR(20)
);
```

5.3 用户定义的完整性

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，RDBMS就检查是否满足属性上的约束条件，如果不满足则拒绝执行该操作。

5.3 用户定义的完整性

5.3.3 元组上的约束条件定义

- 用CHECK短语定义元组上的约束条件，即元组级的限制，可以设置不同属性之间的取值的相互约束条件。

例12：男学生的名字不能以Ms.开头。

```
ALTER TABLE Student
ADD  CHECK(Sgender='女' OR Sname NOT LIKE 'Ms.%');
```

5.3 用户定义的完整性

5.3.4 元组上的约束条件的检查和违约处理

- 当往表中插入元组或修改属性的值时，RDBMS就检查是否满足属性上的约束条件，如果不满足则拒绝执行该操作。

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 完整性的约束命名子句

5.5 域中的完整性限制

5.6 断言

5.7 触发器

5.4 完整性的约束命名子句

5.4.1 完整性约束命名子句

- CONSTRAINT <完整性约束条件名> [PRIMARY KEY短语 | FOREIGN KEY短语|CHECK短语|NOT NULL|UNIQUE]
- 用于对完整性约束条件命名，从而可以灵活地增加、删除一个完整性约束条件。

5.4 完整性的约束命名子句

5.4.1 完整性约束命名子句

例13：建立Student表，学号在9000～9999之间，姓名不能取空，年龄小于30，性别只能取男或女

```
CREATE TABLE Student
(Sno NUMERIC(5) CONSTRAINT C1 CHECK(Sno BETWEEN 9000
    AND 9999),
 Sname CHAR(20)  CONSTRAINT C2 not null,
 Sgender  CHAR(2) CONSTRAINT C3  CHECK(Sgender IN('男','女')),
 Sage SMALLINT CONSTRAINT C4 CHECK(Sage<30)
);
```

5.4 完整性的约束命名子句

5.4.2 修改表中的完整性限制

例14：去掉对Student表中性别取值的限制

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

思考：恢复对Student表中性别取值的限制?

```
alter table student add CHECK(Sgender IN ('男','女'));
```

❖ 注意：增加约束时可以不命名，但只有命名的约束可以删除。

5.4 完整性的约束命名子句

5.4.2 修改表中的完整性限制

例15：修改Student表中的约束条件，要求年龄大于17。

- ALTER TABLE Student
  DROP CONSTRAINT C4;
- ALTER TABLE Student
  ADD CONSTRAINT C4 CHECK(Sage>17 and Sage <30);

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 完整性的约束命名子句

5.5 域中的完整性限制(自学)

5.6 断言

5.7 触发器

5.5 域中的完整性限制

❖ SQL中可以用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16：建立一个性别域，并声明性别域的取值范围。

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

例17：将student表中的性别属性说明为取自性别域。

```
ALTER TABLE Student
    ALTER COLUMN Sgender  GenderDomain;
```

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 完整性的约束命名子句

5.5 域中的完整性限制

5.6 断言

5.7 触发器

断言

- SQL中，可以使用 CREATE ASSERTION语句，通过声明性断言来指定更具一般性的约束。
- 可以定义涉及多个表的或聚集操作的比较复杂的完整性约束。
- 断言创建以后，任何对断言中所涉及的关系的操作都会触发关系数据库管理系统对断言的检查，任何使断言不为真值的操作都会被拒绝执行

1. 创建断言的语句格式

- **CREATE ASSERTION<断言名><CHECK 子句>**
- 每个断言都被赋予一个名字，**<CHECK 子句>**中的约束条件与**WHERE**子句的条件表达式类似。

[例**5.18**] 限制数据库课程最多**60**名学生选修

```
CREATE ASSERTION ASSE_SC_DB_NUM
CHECK (60 >= (select count(*)
        /*此断言的谓词涉及聚集操作count的SQL语句*/
 From Course,SC
 Where SC.Cno=Course.Cno and
       Course.Cname ='数据库')
                    );
```

断言（续）

[例5.19]限制每一门课程最多60名学生选修

```
CREATE ASSERTION ASSE_SC_CNUM1
    CHECK(60 >= ALL (SELECT count(*)
   FROM    SC
                    GROUP by cno)
                );
```

/*此断言的谓词，涉及聚集操作count 和分组函数group by 的SQL语句*/

断言（续）

[例5.20]限制每个学期每一门课程最多60名学生选修

首先，需要修改SC表的模式，增加一个“学期（TERM）”属性

```
ALTER TABLE SC ADD TERM DATE;
```

然后，定义断言：

```
CREATE ASSERTION ASSE_SC_CNUM2
CHECK(60 >= ALL (SELECT count(*)
                 FROM SC
                 GROUP by cno,TERM)
                );
```

断言（续）

2. 删除断言的语句格式为

– DROP ASSERTION <断言名>;

– 如果断言很复杂，则系统在检测和维护断言的开销较高，这是在使用断言时应该注意的

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 完整性的约束命名子句

5.5 域中的完整性限制

5.6 断言

5.7 触发器

5.6 触发器

触发器概念

- 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。
- 触发器（trigger）的执行不是由程序调用，也不是手工启动，而是由事件来触发(激活)，比如当对一个表进行操作(insert，delete，update)时就会激活它执行。也称为主动规则（Active Rule）或事件－条件－动作规则（Event－Condition－Action）。
- 触发器可以查询其他表，而且可以包含复杂的 SQL 语句。它们主要用于强制服从复杂的业务规则或要求。经常用于加强数据的完整性约束和业务规则等。

5.7 触发器

5.7.1 定义触发器

5.7.2 激活触发器

5.7.3 删除触发器

5.7.1 定义触发器

» CREATE TRIGGER语法格式

CREATE TRIGGER <触发器名>

{BEFORE | AFTER} <触发事件> ON <表名>

REFERENCING NEW|OLD ROW AS<变量>

FOR EACH {ROW | STATEMENT}

[WHEN <触发条件>]<触发动作体>

事件-条件-动作（event-condition-action）规则：

当特定的系统事件发生时，对规则的条件进行检查，如果条件成立则执行规则中的动作，否则不执行该动作。规则中的动作体可以很复杂，通常是一段SQL存储过程。

[例5.21]当对表SC的Grade属性进行修改时，若分数增加了10%则将此次操作记录到下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

其中Oldgrade是修改前的分数，Newgrade是修改后的分数。

```
CREATE TRIGGER  SC_T
AFTER UPDATE OF Grade ON SC
 REFERENCING
      OLDrow  AS  OldTuple,
      NEWrow AS  NewTuple
FOR EACH ROW
WHEN (NewTuple.Grade >= 1.1*OldTuple.Grade)
    INSERT INTO SC_U(Sno,Cno,OldGrade,NewGrade)
  VALUES(OldTuple.Sno,OldTuple.Cno,OldTuple.Grade,NewTuple.Grade)
```

定义触发器（续）

» 定义触发器的语法说明

（1）表的拥有者才可以在表上创建触发器

（2）触发器名

- 触发器名可以包含模式名，也可以不包含模式名
- 同一模式下，触发器名必须是唯一的
- 触发器名和表名必须在同一模式下

（3）表名

- 触发器只能定义在基本表上，不能定义在视图上
- 当基本表的数据发生变化时，将激活定义在该表上相应触发事件的触发器

定义触发器（续）

（4）触发事件

- 触发事件可以是INSERT、DELETE或UPDATE

  也可以是这几个事件的组合

- 还可以UPDATE OF<触发列，...>，即进一步指明修改哪些列时激活触发器

- AFTER/BEFORE是触发的时机

  ➢AFTER表示在触发事件的操作执行之后激活触发器

  ➢BEFORE表示在触发事件的操作执行之前激活触发器

定义触发器（续）

（5）触发器类型

➢行级触发器（FOR EACH ROW）

➢语句级触发器（FOR EACH STATEMENT）

例如,在例5.11的TEACHER表上创建一个AFTER UPDATE触发器，触发事件是UPDATE语句：

```
UPDATE TEACHER SET Deptno=5;
```

假设表TEACHER有1000行

- 如果是语句级触发器，那么执行完该语句后，触发动作只发生一次
- 如果是行级触发器，触发动作将执行1000次

定义触发器（续）

[例5.22] 将每次对表Student的插入操作所增加的学生个数记录到表StudentInsertLog中。

```
CREATE TRIGGER Student_Count
AFTER INSERT ON Student
    /*指明触发器激活的时间是在执行INSERT后*/
REFERENCING
 NEW TABLE AS DELTA
FOR EACH STATEMENT
    /*语句级触发器, 即执行完INSERT语句后下面的触发动作体才执行一次*/
 INSERT INTO StudentInsertLog (Numbers)
 SELECT COUNT(*) FROM DELTA
```

定义触发器（续）

（6）触发条件

- 触发器被激活时，只有当触发条件为真时触发动作体才执行;否则触发动作体不执行。
- 如果省略WHEN触发条件，则触发动作体在触发器激活后立即执行

定义触发器（续）

（7）触发动作体

- 触发动作体可以是一个匿名PL/SQL过程块

  也可以是对已创建存储过程的调用

- 如果触发动作体执行失败，激活触发器的事件就会终止执行，触发器的目标表或触发器可能影响的其他对象不发生任何变化

注意：不同的RDBMS产品触发器语法各不相同

定义触发器（续）

[例5.23] 定义一个BEFORE行级触发器，为教师表Teacher定义完整性规则“教授的工资不得低于4000元，如果低于4000元，自动改为4000元”。

```
CREATE TRIGGER Insert_Or_Update_Sal
 BEFORE INSERT OR UPDATE ON Teacher
                    /*触发事件是插入或更新操作*/
FOR EACH ROW        /*行级触发器*/
BEGIN                          /*定义触发动作体，是PL/SQL过程块*/
        IF (new.Job='教授') AND (new.Sal < 4000)
      THEN  new.Sal :=4000;
      END IF;
END;
```

5.7.2 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，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。

5.7 触发器

5.7.1 定义触发器

5.7.2 激活触发器

5.7.3 删除触发器

5.7.3 删除触发器

» 删除触发器的SQL语法：

DROP TRIGGER <触发器名> ON <表名>;

» 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。

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

小 结

- 数据库的完整性是为了保护数据库中数据的正确性和相容性。
- SQL Server提供的数据完整性机制主要包括：约束（Constraint）、默认（Default）、规则（Rule）、触发器（Trigger）、存储过程（Stored Procedure）等。
- 关系系统中最重要的完整性约束包括实体完整性、参照完整性和用户定义的完整性。这些完整性的定义一般由SQL的DDL语句来实现。
- DBMS提供完整性的定义、检查和违约处理。
- 实现数据库完整性的一个重要方法是使用触发器即事件－条件－动作结构。

测验：

employee(employee_name, street, city)
works(employee_name, company_name, salary)
company(company_name, city)
manages(employee_name, manager_name)

1.给出上述雇员数据库的SQL DDL定义（包括完整性约束）
2.查询没有经理的雇员（没有列在manages表中，或者有null经理）