数 据 库 系 统

Database System

主讲: 张仲楠 教授
Email：zhongnan_zhang@xmu.edu.cn
Office：海韵A416

数据库系统

Database System

第五章　数据库完整性

数据库完整性

- 数据库的完整性
  - 数据的正确性
    - 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
    - 指数据库同一对象在不同关系表中的数据是符合逻辑的
    - Ex: 学生所选的课程必须是学校开设的课程，学生所在的院系必须是学校已成立的院系

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(续)

- 为维护数据库的完整性，DBMS必须：

  1.提供**定义**完整性约束条件的**机制**

  - 完整性约束条件也称为**完整性规则**，是数据库中的数据必须满足的**语义约束条件**
  - SQL标准使用了一系列概念来描述完整性，包括关系模型的实体完整性、参照完整性和用户定义完整性
  - 这些完整性一般由SQL的**数据定义语言语句**来实现
  - 作为数据库模式的一部分存入**数据字典**中

数据库完整性(续)

- **2.提供完整性检查的方法**
  - 数据库管理系统中检查数据是否满足完整性约束条件的机制称为完整性检查。
  - 一般在**INSERT、UPDATE、DELETE**语句执行后开始检查，也可以在事务提交时检查

数据库完整性(续)

3.违约处理

- 数据库管理系统若发现用户的操作违背了完整性约束条件，就采取一定的动作
  - 拒绝（NO ACTION）执行该操作
  - 级连（CASCADE）执行其他操作

数据库完整性(续)

- 完整性定义和检查控制机制由DBMS实现
  - 减轻应用程序员的负担
  - 为所有用户和应用提供一致的数据库完整性
  - 由应用程序来实现的完整性控制时有漏洞的

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

- **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**

5.1.1 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(**inline specification**)
  - 定义为表级约束条件(**out-of-line specification**)
- 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

实体完整性定义(续)

［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));
```

实体完整性定义(续)

(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)
);
```

实体完整性定义(续)

［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 实体完整性

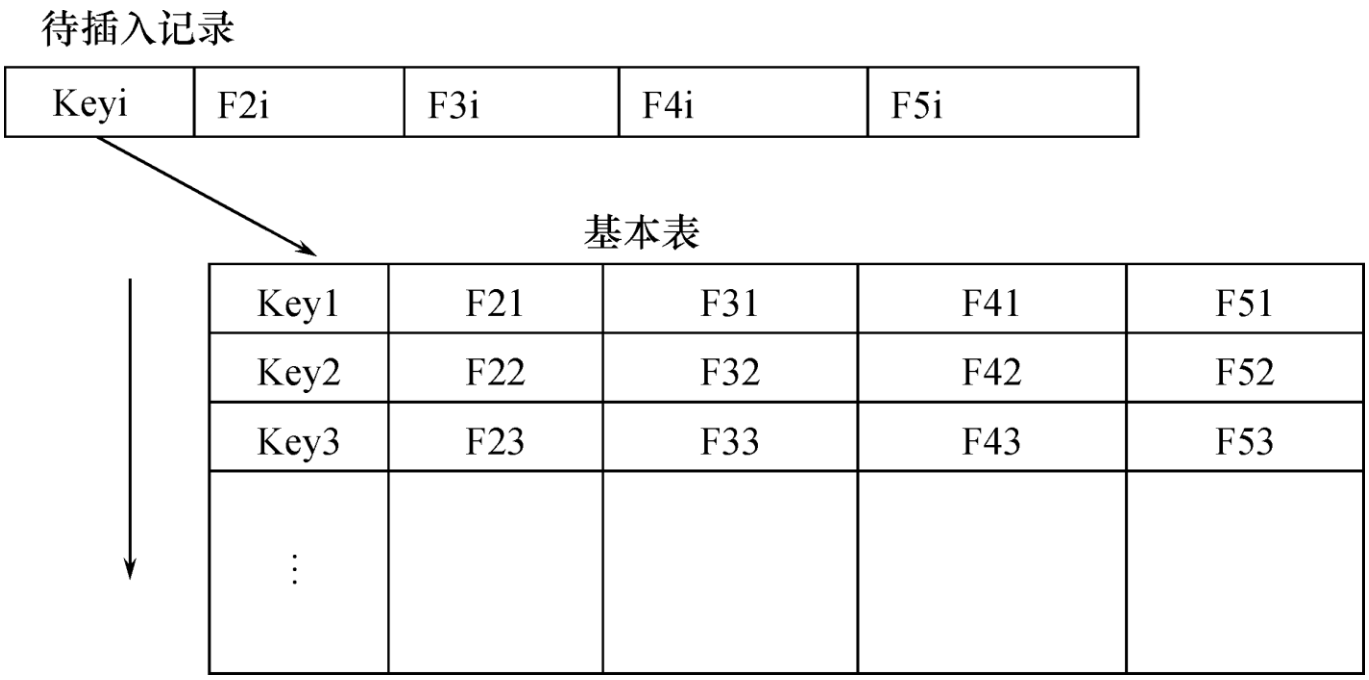
- **5.1.1 实体完整性定义**
- **5.1.2 实体完整性检查和违约处理**

5.1.2 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

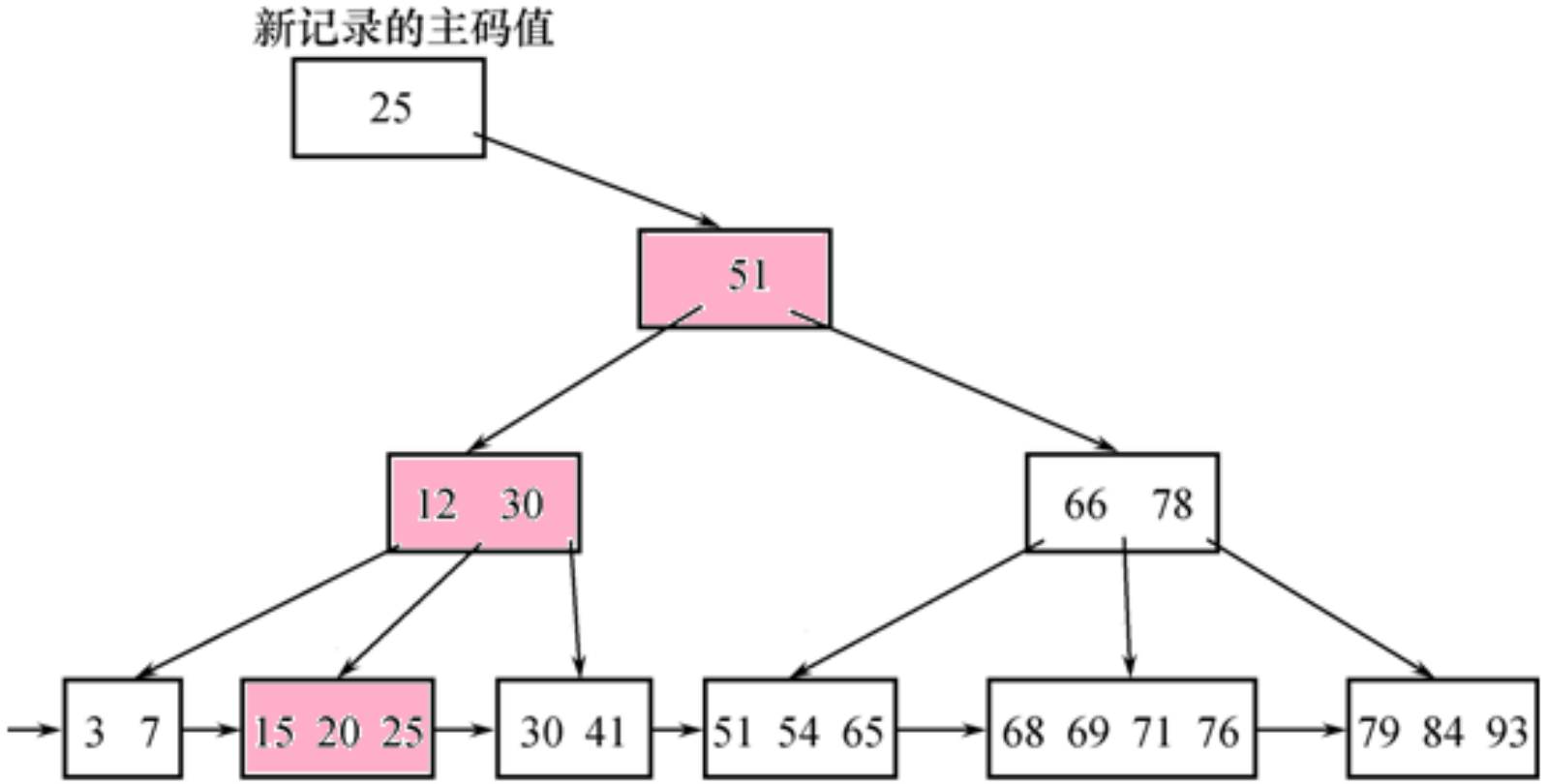
实体完整性检查和违约处理(续)

- 检查记录中主码值是否唯一的一种方法是进行**全表扫描**



实体完整性检查和违约处理(续)

- **索引**：避免全表扫描



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

5.2 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

参照完整性定义(续)

［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*/
);
```

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**

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
    - 当删除/修改被参照表→对应删除/修改参照表中不一致元组
    - 例如：修改Student, 级联修改SC
  - 3. 设置为空值（SET NULL）
    - 例如：专业表中某元组删除，学生表中对应专业号置空
    - 如果删除某一Student, SC表中的对应Sno不能置空
    - 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

违约处理(续)

［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*/
   );
```

ON DELETE CASCADE

删被参照关系 Student中元组

级联删参照关系 SC中元组

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

5.3 用户定义的完整性

- 用户定义的完整性就是针对某一具体应用的数据必须满足的语义要求
- RDBMS提供，而不必由应用程序承担

5.3 用户定义的完整性

- **5.3.1 属性上的约束条件的定义**
- **5.3.2 属性上的约束条件检查和违约处理**
- **5.3.3 元组上的约束条件的定义**
- **5.3.4元组上的约束条件检查和违约处理**

5.3.1 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）

属性上的约束条件的定义(续)

- 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 * /
    );
```

属性上的约束条件的定义(续)

- 2.列值唯一

［例6］ 建立部门表DEPT，要求部门名称Dname列取值唯一且非空，部门编号Deptno列为主码

```
CREATE TABLE DEPT
    (   Deptno  NUMERIC(2),
        Dname  CHAR(9)  UNIQUE NOT NULL,
                        /*要求Dname列值唯一, 并且不能取空值*/
        Location  CHAR(10),
        PRIMARY KEY (Deptno)
    );
```

属性上的约束条件的定义(续)

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

属性上的约束条件的定义(续)

- 3. 用CHECK短语指定列值应该满足的条件

［例8］ SC表的Grade的值在0到100之间。

```
CREATE TABLE SC
(Sno    CHAR(9)，
 Cno     CHAR(4)，
 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

- **5.3.1 属性上的约束条件的定义**
- **5.3.2 属性上的约束条件检查和违约处理**
- **5.3.3 元组上的约束条件的定义**
- **5.3.4元组上的约束条件检查和违约处理**

5.3.2 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**

5.3 用户定义的完整性

- **5.3.1 属性上的约束条件的定义**
- **5.3.2 属性上的约束条件检查和违约处理**
- **5.3.3 元组上的约束条件的定义**
- **5.3.4元组上的约束条件检查和违约处理**

5.3.3 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

元组上的约束条件的定义(续)

［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

5.3 用户定义的完整性

- **5.3.1 属性上的约束条件的定义**
- **5.3.2 属性上的约束条件检查和违约处理**
- **5.3.3 元组上的约束条件的定义**
- **5.3.4 元组上的约束条件检查和违约处理**

5.3.4 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

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

***5.5** 域中的完整性限制

***5.6** 断言

5.7 触发器

5.8 小结

5.4 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

|NOT NULL短语

|UNIQUE短语］

完整性约束命名子句(续)

［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。

完整性约束命名子句(续)

［例11］ 建立教师表Teacher，要每个教师的应发工资不低于10000元。应发工资实际上就是实发工资列Sal与扣除项Deduct之和。

```
CREATE TABLE Teacher
  (Eno  NUMERIC(4) PRIMARY KEY,
   Ename  CHAR(10)，
   Job CHAR(8),
   Sal  NUMERIC(7,2)，
   Deduc NUMERIC(7,2),
   Deptno  NUMERIC(2),
   CONSTRAINT EMPFKey FOREIGN KEY(Deptno) REFERENCES
DEPT(Deptno),
   CONSTRAINT C1 CHECK(Sal + Deduc >= 10000)
 );
```

完整性约束命名子句(续)

- **2. 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

完整性约束命名子句(续)

［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);
```

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

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
  - 任何对表的增删改操作均由服务器自动激活相应的触发器
  - 在DBMS核心层实施完整性控制
  - 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

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>
```

触发器又叫做**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**。

定义触发器(续)

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>

- 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)

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>

（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)

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>

（5）触发器类型

- 行级触发器（FOR EACH ROW）
- 语句级触发器（FOR EACH STATEMENT）⇦ Oracle 默认

例如,在STUDENT表上创建一个AFTER UPDATE触发器，触发事件是UPDATE语句：

UPDATE STUDENT SET SAGE = SAGE+1;

假设表STUDENT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)

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>

（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)

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>

（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
- 如果是**行级**触发器，用户都可以在过程体中**使用NEW和OLD**引用事件之后的新值和事件之前的旧值
- 如果是**语句级**触发器，则**不能**在触发动作体中使用NEW或OLD进行引用
- 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)

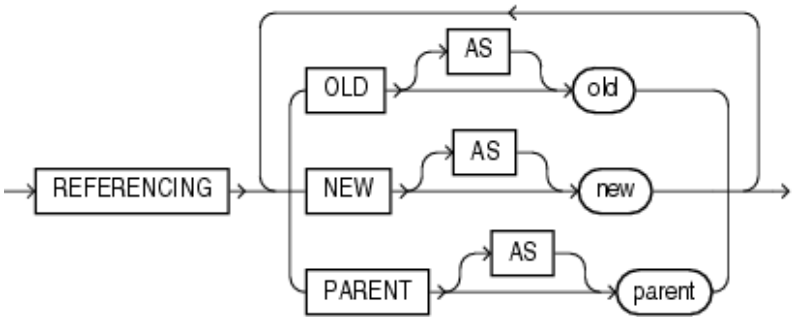
[例5.21]当对表SC的Grade属性进行修改时，若分数增加了10%则将此次操作记录到下面表中：

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 or REPLACE TRIGGER  SC_T
AFTER UPDATE OF Grade ON SC
REFERENCING
OLD AS  oldtuple
NEW AS  newtuple
FOR EACH ROW
WHEN (newtuple.Grade >= 1.1 * oldtuple.Grade)
BEGIN
        INSERT INTO SC_U
        VALUES(:oldtuple.Sno, :oldtuple.Cno, :oldtuple.Grade, :newtuple.Grade);
END;
```

无逗号



在After行级触发器里，触发体只能读OLD和NEW，不能写

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14000元，如果低于14000元，自动改为14000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 < 14000) THEN
       :new.Sal :=14000;
       END IF;
    END;
```

在Before行级触发器里，触发体既能读也能写OLD和NEW

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**

5.7.2 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。

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>**;

  Oracle中不需要

- 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21］ 删除教师表Teacher上的触发器Insert_Sal

```
DROP TRIGGER Insert_Sal ON Teacher;
```

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

5.8 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