第五章　数据库完整性

(2) User 通过DML语言操纵数据进出数据库

(1) User 通过DDL语言定义数据格式

(3) User 通过DCL语言定义数据安全性

(4) User 通过DDL语言定义数据 完整性

用户定义的数据安全性完整性

数据格式

班级	课程	教师	学期	学号	姓名	成绩
981101	数据库	李四	98秋	01	张三	100
981101	数据库	李四	98秋	02	张四	90
981101	数据库	李四	98秋	03	张五	80
981101	计算机	李五	98秋	01	张三	89
981101	计算机	李五	98秋	02	张四	98
981101	计算机	李五	98秋	03	张五	72
981102	数据库	李四	99秋	01	王三	30
981102	数据库	李四	99秋	02	王四	90
981102	数据库	李四	99秋	03	王武	78

数据

981102 数据库 李四 99秋 02 王四 90

检查用户合法性
检查数据合法性
按用户要求进行操作

DBMS

User

(DBMS依据各种定义对数据库进行管理和使用)

SQL

- 集数据定义语言（DDL），数据操纵语言（DML），数据控制语言（DCL）功能于一体

表 3.1　SQL 语言的动词

SQL 功能	动　词
数据查询	SELECT
数据定义	CREATE，DROP，ALTER
数据操纵	INSERT，UPDATE，DELETE
数据控制	GRANT，REVOKE

数据库完整性

- 数据库的完整性
  - 数据的*正确性*和*相容性*

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

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

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 触发器

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 触发器

5.1 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
  - 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

[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)
 );
```

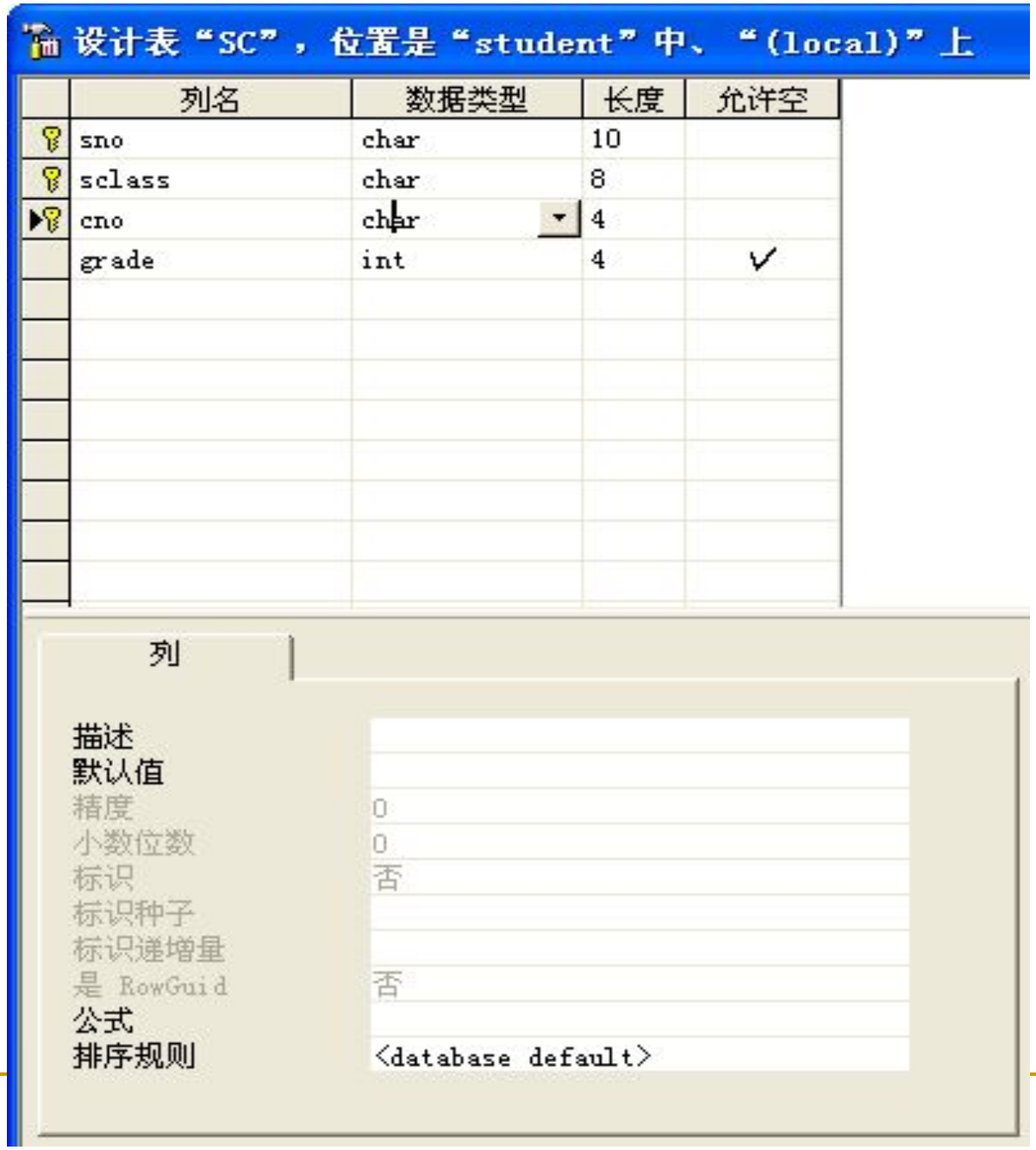
[例1] 将Student表中的Sno属性定义为码

(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)
    );
```

[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*/
 );
```

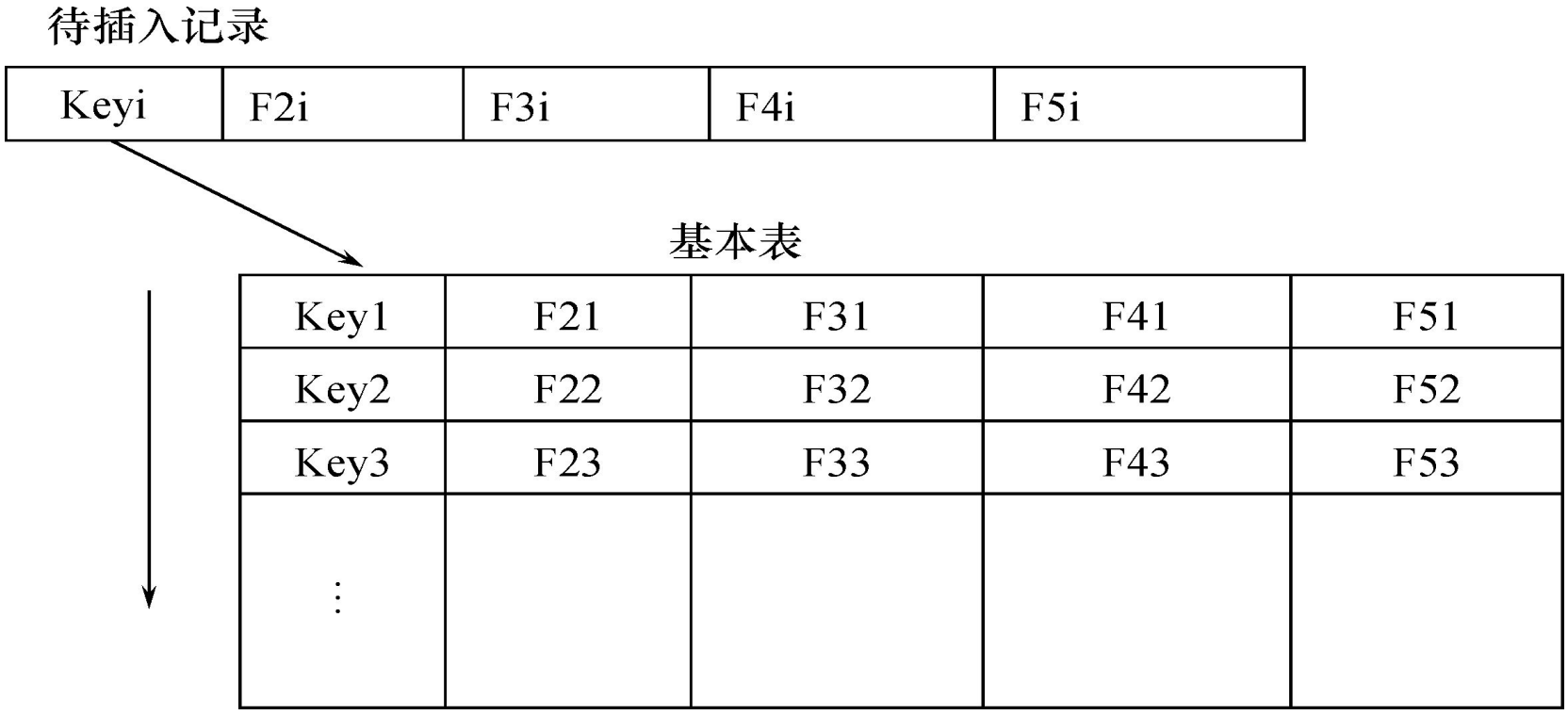

设计表“SC”，位置是“student”中、“(local)”上
列名
数据类型
长度
允许空
sno
char
10
sclass
char
8
cno
char
4
grade
int
4
列
描述
默认值
精度
0
小数位数
0
标识
否
标识种子
标识递增量
是 RowGuid
否
公式
排序规则
<database default>

5.1 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：
  - 检查主码值是否唯一，如果不唯一则拒绝插入或修改
  - 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

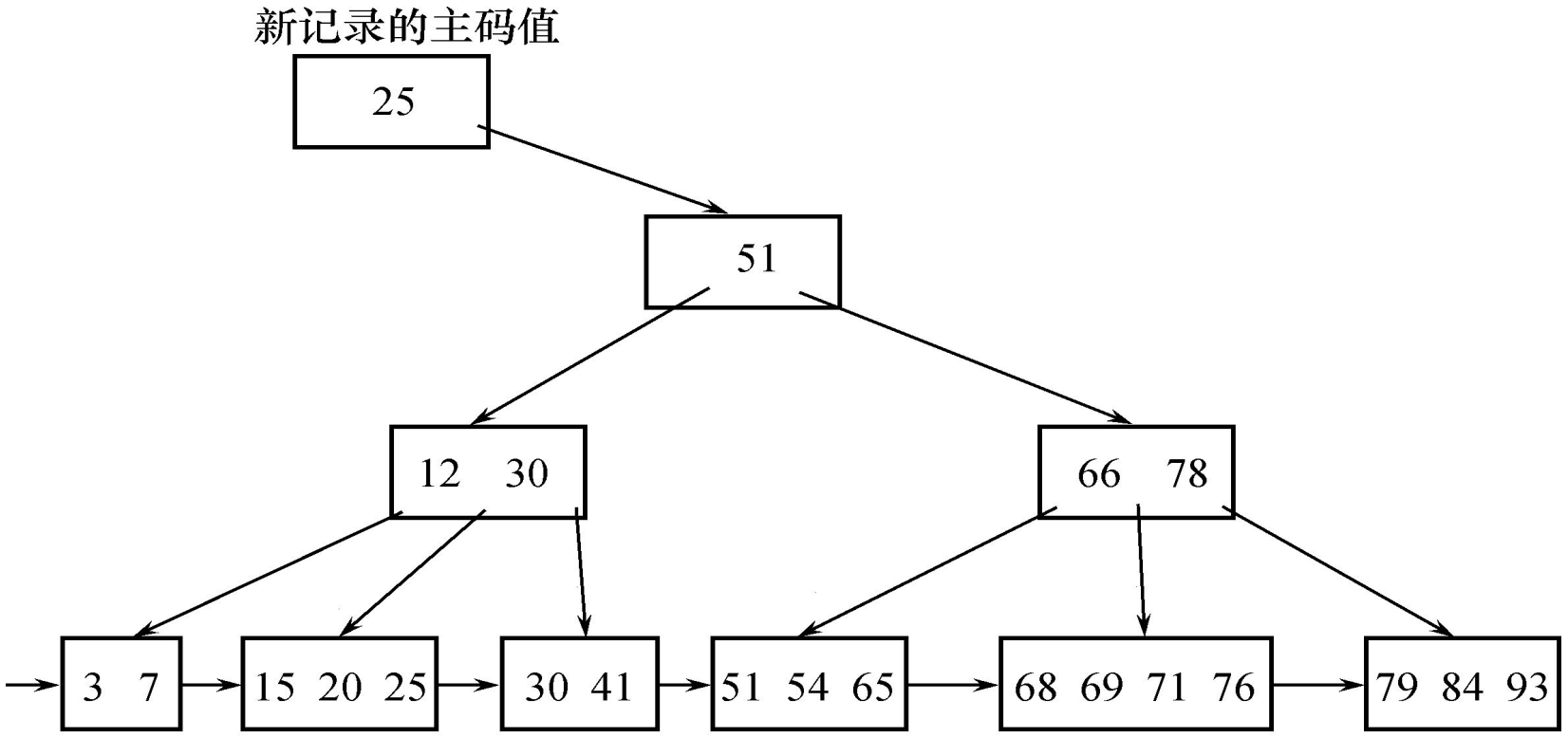
5.1 实体完整性-检查和违约处理

- 检查记录中主码值是否唯一的一种方法是进行*全表扫描*



5.1 实体完整性-检查和违约处理

- 另一个是*索引*



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 触发器

5.2 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-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

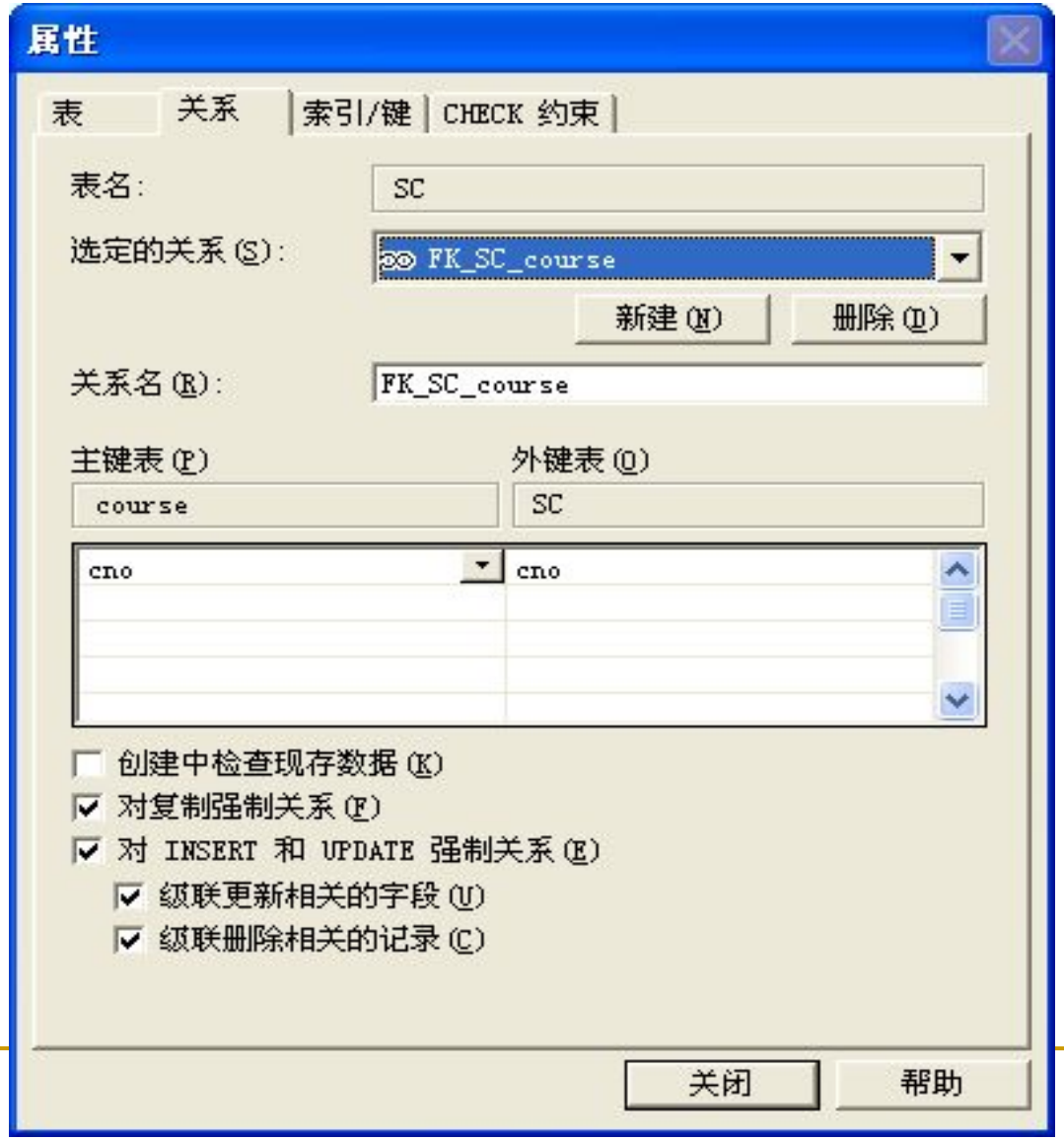
5.2 参照完整性-检查和违约处理

- 参照完整性违约处理
  - 拒绝(NO ACTION)执行
    - 默认策略
  - 级联(CASCADE)操作
  - 设置为空值（SET-NULL）

［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*/);
```

5.2 参照完整性-检查和违约处理



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 触发器

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

- 属性上的约束条件
- 元组上的约束条件

5.3 用户定义的完整性-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）

[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 * / ) ;
```

[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)
    );
```

5.3 用户定义的完整性-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-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8]　当学生的性别是男时，其名字不能以Ms.打头。

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

[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*/
    );
```

5.3 用户定义的完整性-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 触发器

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]

[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)
 )；
```

5.4 完整性约束命名子句

- 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

5.4 完整性约束命名子句

ALTER TABLE <表名>

[ADD <新列名> <数据类型> [完整性约束]]

[DROP <完整性约束名>]

[ALTER COLUMN<列名> <数据类型>];

[例11] 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 完整性约束命名子句

5.5 触发器

5.5 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
  - 只要对它所保护的数据进行修改，它就会自动触发，包括对表进行insert、update和delete操作。
  - 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

5.5 触发器-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>

  FOR EACH { ROW | STATEMENT }

  [WHEN <触发条件>]

  <触发动作体>

5.5 触发器-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>

  FOR EACH { ROW | STATEMENT }

  [WHEN <触发条件>]

  <触发动作体>

  - *创建者：表的拥有者*

5.5 触发器-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>

  FOR EACH { ROW | STATEMENT }

  [WHEN <触发条件>]

  <触发动作体>

  - *触发器的目标表*

5.5 触发器-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>

  FOR EACH { ROW | STATEMENT }

  [WHEN <触发条件>]

  <触发动作体>

  - *触发事件：INSERT、DELETE、UPDATE*

5.5 触发器-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>

  FOR EACH { ROW | STATEMENT }

  [WHEN <触发条件>]

  <触发动作体>

  - *行级触发器（FOR EACH ROW）*
  - *语句级触发器（FOR EACH STATEMENT）*

5.5 触发器-定义触发器

- 例如,假设在［例11］的TEACHER表上创建了一个AFTER UPDATE触发器。如果表TEACHER有1000行，执行如下语句：

  UPDATE TEACHER SET Deptno=5;

  - 如果该触发器为语句级触发器，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

5.5 触发器-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>

  FOR EACH { ROW | STATEMENT }

  [WHEN <触发条件>]

  <触发动作体>

  - ***触发条件：当触发器被激活时，只有触发条件为真时触发动作体才执行***

5.5 触发器-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>

  FOR EACH { ROW | STATEMENT }

  ［WHEN <触发条件>］

  <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***

[例12] 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*/
  AS  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;
```

[例13] 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) reference teacher(eno)，
  Sal    NUMERIC(7，2)，
  Username  char(10)，
  Date   TIMESTAMP
 )；
```

```
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;
```

```
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;
```

5.5 触发器-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：
    - 执行该表上的BEFORE触发器；
    - 激活触发器的SQL语句；
    - 执行该表上的AFTER触发器。

[例14] 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

5.5 触发器-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。

[例21]　删除教师表Teacher上的触发器Insert_Sal

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

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