第五章 数据库完整性

数据库的完整性是指数据的正确性和相容性

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

数据完整性：防范对象：不合语义的、不正确的数据

数据安全性：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：

1、提供定义安全性约束条件的机制

数据库中数据必须要满足的语义约束条件

2、提供完整性检查的方法

一般在语句执行好开始检查或是在事务提交时检查

3、进行违约处理

拒绝执行（NO ACTION） 级联执行其他操作（CASCADE）

5.1 实体完整性

5.1.1 定义实体完整性

在 CREATE TABLE 中用 PRIMARY KEY 定义

（1）定义为列级约束条件

```
CREATE TABLE Student
   (    Sno   CHAR(9)   PRIMARY KEY,
        Sname   CHAR(20) NOT NULL,
     );
```

（2）定义为表级约束条件

```
CREATE TABLE SC
(   Sno    CHAR(9)   NOT NULL,
    Cno   CHAR(4)   NOT NULL,
    PRIMARY KEY (Sno,Cno)      /*只能在表级定义主码*/
);
```

5.1.2 实体完整性检查和违规处理

检查：1、检查主码值是否唯一，若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，只要有一个为空则拒绝插入或修改

在检查时需要扫描全表，可以在主码上自动建立一个索引，节约时间

5.2 参照完整性

5.2.1 定义参照完整性

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

```
CREATE TABLE SC
(   Sno       CHAR(9)    NOT NULL,
    Cno       CHAR(4)    NOT NULL,
    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*/
```

5.2.2 参照完整性检查和违约处理

问题：对该表的增加、删除、修改操作导致外码 Sno 在 Student 表中找不到与之对应的 Sno

策略：

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

（1）拒绝执行（NO ACTION）

（2）级联操作（CASCADE）

（3）设置为空值（SET-NULL）

将参照表中所有造成不一致的元组的对应属性设置为空值

对于参照完整性，处理应该定义外码，还应该定义外码列是否允许空值

例：

```
CREATE TABLE SC
(   Sno     CHAR(9)   NOT NULL，
    Cno     CHAR(4)   NOT NULL，
    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

定义：针对某一具体应用的数据必须满足的语义要求

5.3.1 属性上的约束

1、属性（列）上约束条件的定义

在 CREATE TABLE 时定义 NOT NULL, UNIQUE, CHECK（检验列值是否满足一个表达式）

```
CREATE TABLE Student
(   Sno   CHAR(9) PRIMARY KEY,
    Sname CHAR(8) NOT NULL,
    Ssex   CHAR(2)   CHECK （Ssex IN （‘男’,’女’)），/*Ssex 只允许取'男'或'女' */
    Sage   SMALLINT,
    Sdept   CHAR(20)   );
```

2、属性上的约束条件检查和违约处理

在插入或是修改时，NBMS 会检查是否满足约束条件，若不满足，则拒绝执行

5.3.2 元组上的约束条件

1、元组上约束条件的定义

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

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

```
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*/ );
```

2、元组上约束条件的检查和违规处理

在插入或者删除时，会检查元组上的约束条件，若不满足则拒绝执行

5.4 完整性约束命名子句

1、完整性约束命名子句

CONSTRAINT<完整性约束条件名><完整性约束条件>

完整性约束条件包括 NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY ,CHECK

```
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(Ssage IN (‘男’, ’女’);
   CONSTRAINT StrudentKey PRIMARY KEY(Sno)
);
```

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

```
ALTER TABLE Student
DROP CONSTRAINT C4;
```

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

5.6 断言

在 SQL 中，可以使用 CREATE ASSAERTION 语句，通过声明性断言来指定更具一般性的约束，可以定义设计多个表或聚集成聚集操作的比较复杂的完整性约束

1、创建断言的语句格式

CERATE ASSERTION<断言名><CHECK 子句>

```
CREATE ASSERTION ASSE_SC_DB_NUM
CHECK( 60>=(SELECT COUNT(*)
                FROM SC, Course
                WHERE SC,Cno = Course.Cno and Course.Cname='数据库')
);
CREATE ASSERTION ASSE_SC_CNUM1
      CHECK(60 >= ALL (SELECT count(*)
                           FROM SC
                           GROUP by cno)
);
```

2、删除断言的语句格式为

DROP ASSERTION<断言名>

5.7 触发器

触发器是用户定义在关系表上的一类由时间驱动的特殊过程

触发器保存在数据库服务器中，进行增、删、改操作均由服务器自动激活相应的触发器

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>

（1） 表的创建者才可以在表上创建触发器

（2） 触发器名，触发器名和表名必须在同一模式下；同一模式下，触发器是唯一的

（3） 表名：触发器只能定义在基本表上，不能在视图上；当基本表的数据发生变化，则激活触发器

（4） 触发事件

可以是 INSERT, UPDATE, DELETE

AFTER/BEFORE 是触发的时机

（5） 触发器类型

FOR EACH ROW 行级触发器，每执行完一行激活一次触发器

FOR EACH STATEMENT 语句级触发器，该语句执行完之后再激活触发器

（6） 触发条件

触发条件为真时触发动作才执行

（7）触发动作体

触发动作体即可以是一个匿名 PL/SQL 块，也可以是对存储过程的调用

若是行级触发器，则可以在过程中国用 OLDROW AS 和 NEWROW AS 对新旧值进行引用

若是语句级触发器，则不能

例：

```
CREATE TRIGGER  SC_T
AFTER UPDATE OF Grade ON SC
REFERENCING
    OLD row  AS  OldTuple,
     NEW row 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)

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

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

触发器激活顺序：BEFORE, SQL, AFTER

5.7.3 删除触发器

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。