第5章 数据库完整性

数据库的完整性(integrity)是指数据的正确性(correctness)和相容性(compat- ability)。数据的正确性是指数据是符合现实世界语义、反映当前实际状况的；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，本科学生年龄的取值范围为 14～50 的整数，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等。

数据的完整性和安全性是两个既有联系又不尽相同的概念。数据的完整性是为了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。数据的安全性是保护数据库防止恶意破坏和非法存取。因此，完整性检查和控制的防范对象是不合语义的、不正确的数据，防止它们进入数据库。安全性控制的防范对象是非法用户和非法操作，防止他们对数据库数据的非法存取。

为维护数据库的完整性，数据库管理系统必须能够实现如下功能。

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。它表达了给定的数据模型中数据及其联系所具有的制约和依存规则，用以限定符合数据模型的数据库状态以及状态的变化，以保证数据的正确、有效和相容。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，它们作为数据库模式的一部分存入数据字典中。

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。检查这些操作执行后数据库中的数据是否违背了完整性约束条件。

3. 进行违约处理

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将采取一定的动作，如拒绝

(NO ACTION) 执行该操作或级联 (CASCADE)执行其他操作，进行违约处理以保证数据的完整性。

早期的数据库管理系统不支持完整性检查，因为完整性检查费时费资源。现在商用的关系数据库管理系统产品都支持完整性控制，即完整性定义和检查控制由关系数据库管理系统实现，不必由应用程序来完成，从而减轻了应用程序员的负担。更重要的是，关系数据库管理系统使得完整性控制成为其核心支持的功能，从而能够为所有用户和应用提供一致的数据库完整性。因为由应用程序来实现完整性控制是有漏洞的，有的应用程序定义的完整性约束条件可能被其他应用程序破坏，数据库数据的正确性仍然无法保障。

第 2 章 2.3 节关系的完整性已经讲解了关系数据库三类完整性约束的基本概念，下面将介绍 SQL 语言中实现这些完整性控制功能的方法。

5.1 实体完整性

5.1.1 定义实体完整性

关系模型的实体完整性在 CREATE TABLE 中用 PRIMARY KEY 定义。对单属性构成的码有两种说明方法，一种是定义为列级约束条件，另一种是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，即定义为表级约束条件。

[**例 5.1**] 将 Student 表中的 Sno 属性定义为码。

```
CREATE TABLE Student
      (Sno CHAR(9) PRIMARY KEY,                    /*在列级定义主码*/
       Sname CHAR(20) NOT NULL,
       Ssex CHAR(2),
       Sage SMALLINT,
       Sdept CHAR(20)
      );
```

或者

```
CREATE TABLE Student
       (Sno CHAR(9),
        Sname CHAR(20) NOT NULL,
        Ssex CHAR(2) ,
        Sage SMALLINT,
        Sdept CHAR(20),
        PRIMARY KEY (Sno)                          /*在表级定义主码*/
```

```
);
```

［例 5.2］ 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*/
    );
```

5.1.2 实体完整性检查和违约处理

用 PRIMARY KEY 短语定义了关系的主码后，每当用户程序对基本表插入一条记录或对主码列进行更新操作时，关系数据库管理系统将按照第 2 章 2.3.1 小节中讲解的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。

从而保证了实体完整性。

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的主码值（或者修改的新主码值）是否相同，如图 5.1 所示。

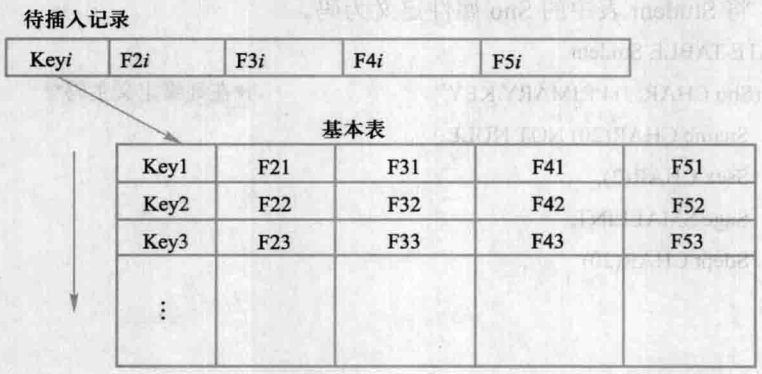

图 5.1 用全表扫描方法检查主码唯一性

全表扫描是十分耗时的。为了避免对基本表进行全表扫描，关系数据库管理系统一般都在主码上自动建立一个索引，如图 5.2 的 *B*+树索引，通过索引查找基本表中是否已经存在新的主码值将大大提高效率。例如，如果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，只要读取三个结点就可以知道该主码值已经存在，所以不能插入这条记录。这三个结点是根结点（51）、中间结点（12 30）和叶结点（15 20 25）。如果新插入记录

的主码值是 86，也只要查找三个结点就可以知道该主码值不存在，所以可以插入该记录。

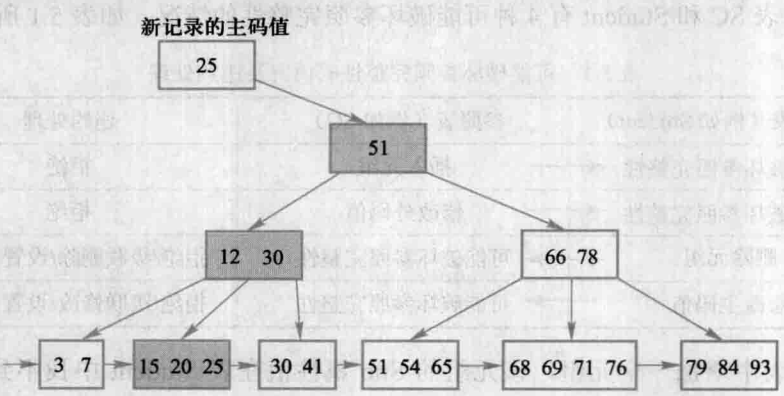

图 5.2 使用索引检查主码唯一

5.2 参照完整性

5.2.1 定义参照完整性

关系模型的参照完整性在 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。

例如，关系 SC 中一个元组表示一个学生选修的某门课程的成绩，（Sno，Cno）是主码。Sno、Cno 分别参照引用 Student 表的主码和 Course 表的主码。

［**例 5.3**］ 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*/
    );
```

5.2.2 参照完整性检查和违约处理

参照完整性将两个表中的相应元组联系起来了。因此，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，必须进行检查以保证这两个表的相容性。

例如，对表 SC 和 Student 有 4 种可能破坏参照完整性的情况，如表 5.1 所示。

表 5.1 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/级联删除/设置为空值
修改主码值 →	可能破坏参照完整性	拒绝/级联修改/设置为空值

（1）SC 表中增加一个元组，该元组的 Sno 属性值在表 Student 中找不到一个元组，其 Sno 属性值与之相等。

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

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

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

当上述的不一致发生时，系统可以采用以下策略加以处理。

（1）拒绝（NO ACTION）执行

不允许该操作执行。该策略一般设置为**默认策略**。

（2）级联（CASCADE）操作

当删除或修改被参照表（Student）的一个元组导致与参照表（SC）的不一致时，删除或修改参照表中的所有导致不一致的元组。

例如，删除 Student 表中 Sno 值为“201215121”的元组，则从要 SC 表中级联删除 SC.Sno='201215121'的所有元组。

（3）设置为空值

当删除或修改被参照表的一个元组时造成了不一致，则将参照表中的所有造成不一致的元组的对应属性设置为空值。例如，有下面两个关系：

学生(学号,姓名,性别,专业号,年龄)

专业(专业号,专业名)

其中学生关系的“专业号”是外码，因为专业号是专业关系的主码。

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。这对应了这样的语义：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。

这里讲解一下外码能否接受空值的问题。

例如，学生表中“专业号”是外码，按照应用的实际情况可以取空值，表示这个学生的专业尚未确定。但在学生－选课数据库中，关系 Student 为被参照关系，其主码为 Sno；SC 为参照关系，Sno 为外码，它能否取空值呢？答案是否定的。因为 Sno 为 SC 的主属性，按照实体完整性 Sno 不能为空值。若 SC 的 Sno 为空值，则表明尚不存在的某个学生，或者某个不知学号的学生，选修了某门课程，其成绩记录在 Grade 列中。这与学校的应用环境是不相符的，因此 SC 的 Sno 列不能取空值。同样，SC 的 Cno 是外码，也是 SC 的主属性，也不能取空值。

因此对于参照完整性，除了应该定义外码，还应定义外码列是否允许空值。

一般地，当对参照表和被参照表的操作违反了参照完整性时，系统选用默认策略，即拒绝执行。如果想让系统采用其他策略则必须在创建参照表时显式地加以说明。

［**例 5.4**］ 显式说明参照完整性的违约处理示例。

```
CREATE TABLE SC
( Sno CHAR(9),
Cno CHAR(4),
Grade SMALLINT,
PRIMARY KEY(Sno,Cno), /*在表级定义实体完整性，Sno、Cno 都不能取空值*/
FOREIGN KEY (Sno) REFERENCES Student(Sno)          /*在表级定义参照完整性*/
    ON DELETE CASCADE
                          /*当删除 Student 表中的元组时，级联删除 SC 表中相应的元组*/
    ON UPDATE CASCADE,
                          /*当更新 Student 表中的 sno 时，级联更新 SC 表中相应的元组*/
FOREIGN KEY (Cno) REFERENCES Course(Cno)          /*在表级定义参照完整性*/
    ON DELETE NO ACTION
                          /*当删除 Course 表中的元组造成与 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*/
);
```

可以对 DELETE 和 UPDATE 采用不同的策略。例如，例 5.4 中当删除被参照表 Course 表中的元组，造成与参照表（SC 表）不一致时，拒绝删除被参照表的元组；对更新操作则采取级联更新的策略。

从上面的讨论可以看到，关系数据库管理系统在实现参照完整性时，除了要提供定义主码、外码的机制外，还需要提供不同的策略供用户选择。具体选择哪种策略，要根据应

用环境的要求确定。

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。目前的关系数据库管理系统都提供了定义和检验这类完整性的机制，使用了和实体完整性、参照完整性相同的技术和方法来处理它们，而不必由应用程序承担这一功能。

5.3.1 属性上的约束条件

1. 属性上约束条件的定义

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，包括：

- 列值非空（NOT NULL）。
- 列值唯一（UNIQUE）。
- 检查列值是否满足一个条件表达式（CHECK 短语）。

（1）不允许取空值

[例 5.5] 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,          /*Sno 属性不允许取空值*/
      Cno CHAR(4) NOT NULL,          /* Cno 属性不允许取空值*/
      Grade SMALLINT NOT NULL,       /* Grade 属性不允许取空值*/
      PRIMARY KEY (Sno, Cno),        /*在表级定义实体完整性，隐含了 Sno、Cno 不允
                                       许取空值，在列级不允许取空值的定义可不写*/
      ⋮
    );
```

（2）列值唯一

[例 5.6] 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 NOT NULL,  /*要求 Dname 列值唯一，且不能取空值*/
      Location CHAR(10),
      PRIMARY KEY (Deptno)
    );
```

（3）用 CHECK 短语指定列值应该满足的条件

［例 5.7］ Student 表的 Ssex 只允许取“男”或“女”。

```
CREATE TABLE Student
    ( Sno CHAR(9) PRIMARY KEY,                          /*在列级定义主码*/
    Sname CHAR(8) NOT NULL,                             /* Sname 属性不允许取空值*/
    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)
    );
```

［例 5.8］ SC 表的 Grade 的值应该在 0 和 100 之间。

```
CREATE TABLE SC
    ( Sno CHAR(9),
    Cno CHAR(4),
    Grade SMALLINT CHECK (Grade>=0 AND Grade <=100),   /*Grade 取值范围是 0 到 100*/
    PRIMARY KEY (Sno, Cno),
    FOREIGN KEY (Sno) REFERENCES Student(Sno),
    FOREIGN KEY (Cno) REFERENCES Course(Cno)
    );
```

2. 属性上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，关系数据库管理系统将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。

5.3.2 元组上的约束条件

1. 元组上约束条件的定义

与属性上约束条件的定义类似，在 CREATE TABLE 语句中可以用 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 5.9］ 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 CHECK 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，因为 Ssex='男'时，条件要想为真值，Sname NOT LIKE 'Ms.%' 必须为真值。

2. 元组上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，关系数据库管理系统将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。

5.4 完整性约束命名子句

以上讲解的完整性约束条件都在 CREATE TABLE 语句中定义，SQL 还在 CREATE TABLE 语句中提供了完整性约束命名子句 CONSTRAINT，用来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。

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 短语等。

[**例 5.10**] 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 这 4 个列级约束。

[例 5.11] 建立教师表 TEACHER，要求每个教师的应发工资不低于 3 000 元。应发工资是工资列 Sal 与扣除项 Deduct 之和。

```
CREATE TABLE TEACHER
    ( Eno NUMERIC(4) PRIMARY KEY,                    /*在列级定义主码*/
      Ename CHAR(10),
      Job CHAR(8),
      Sal NUMERIC(7,2),
      Deduct NUMERIC(7,2),
      Deptno NUMERIC(2),
      CONSTRAINT TEACHERFKey FOREIGN KEY (Deptno)
      REFERENCES DEPT (Deptno),
      CONSTRAINT C1 CHECK (Sal + Deduct >= 3000)
    );
```

2. 修改表中的完整性限制

可以使用 ALTER TABLE 语句修改表中的完整性限制。

[例 5.12] 去掉例 5.10 Student 表中对性别的限制。

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

[例 5.13] 修改表 Student 中的约束条件，要求学号改为在 900 000～999 999 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);
```

*5.5 域中的完整性限制

在第 1、2 章中已经讲到，域是数据库中一个重要概念。一般地，域是一组具有相同

数据类型的值的集合。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，然后就可以用域来定义属性。这样定义的优点是，数据库中不同的属性可以来自同一个域，当域上的完整性约束条件改变时只要修改域的定义即可，而不必一一修改域上的各个属性。

［例 5.14］ 建立一个性别域，并声明性别域的取值范围。

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

这样例 5.10 中对 Ssex 的说明可以改写为：

```
Ssex GenderDomain
```

［例 5.15］ 建立一个性别域 GenderDomain，并对其中的限制命名。

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

［例 5.16］ 删除域 GenderDomain 的限制条件 GD。

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

［例 5.17］ 在域 GenderDomain 上增加性别的限制条件 GDD。

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

这样，通过例 5.16 和例 5.17，就把性别的取值范围由('男', '女')改为 ('1', '0')。

5.6 断　　言

在 SQL 中可以使用数据定义语言中的 CREATE ASSERTION 语句，通过声明性断言（declarative assertions）来指定更具一般性的约束。可以定义涉及多个表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')
    );
```

每当学生选修课程时，将在 SC 表中插入一条元组（Sno，Cno，NULL），ASSE_SC_DB_NUM 断言被触发检查。如果选修数据库课程的人数已经超过 60 人，CHECK 子句返回值为“假”，对 SC 表的插入操作被拒绝。

［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 (*)      /*此断言的谓词，涉及聚集操作 count */
                     FROM SC               /*和分组函数 group by 的 SQL 语句*/
                     GROUP by cno )
                    );
```

［例 5.20］ 限制每个学期每一门课程最多 60 名学生选修。

首先修改 SC 表的模式，增加一个“学期（TERM）”的属性。

```
ALTER TABLE SC ADD TERM DATE; /*先修改 SC 表，增加 TERM 属性，它的类型是 DATE*/
```

然后定义断言：

```
CREATE ASSERTION ASSE_SC_CNUM2
    CHECK (60>=ALL ( select count (*) from SC group by cno,TERM ) );
```

2. 删除断言的语句格式

```
DROP ASSERTION <断言名>;
```

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

5.7 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。一旦定义，触发器将被保存在数据库服务器中。任何用户对表的增、删、改操作均由服务器自动激活相应的触发器，在关系数据库管理系统核心层进行集中的完整性控制。触发器类似于约束，但是比约束更加灵活，可以实施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。

触发器在 SQL 99 之后才写入 SQL 标准，但是很多关系数据库管理系统很早就支持触发器，因此不同的关系数据库管理系统实现的触发器语法各不相同、互不兼容。请读者在上机实验时注意阅读所用系统的使用说明。

5.7.1 定义触发器

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（如对一个表的增、删、改操作，事务的结束等）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，可以涉及其他表和其他数据库对象，通常是一段 SQL 存储过程。

SQL 使用 CREATE TRIGGER 命令建立触发器，其一般格式为

```
CREATE TRIGGER <触发器名>                    /*每当触发事件发生时，该触发器被激活*/
{BEFORE | AFTER} <触发事件> ON <表名>
                                             /*指明触发器激活的时间是在执行触发事件前或后*/
REFERENCING NEW|OLD ROW AS<变量>             /*REFERENCING 指出引用的变量*/
FOR EACH{ROW | STATEMENT}                    /*定义触发器的类型，指明动作体执行的频率*/
[WHEN <触发条件>] <触发动作体>               /*仅当触发条件为真时才执行触发动作体*/
```

下面对定义触发器的各部分语法进行详细说明。

（1）只有表的拥有者，即创建表的用户才可以在表上创建触发器，并且一个表上只能创建一定数量的触发器。触发器的具体数量由具体的关系数据库管理系统在设计时确定。

（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，并且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，因此该表也称为触发器的目标表。

（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，如 INSERT OR DELETE 等，还可以是 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。

（5）触发器类型

触发器按照所触发动作的间隔尺寸可以分为行级触发器（FOR EACH ROW）和语句级触发器（FOR EACH STATEMENT）。

例如，假设在例 5.11 的 TEACHER 表上创建了一个 AFTER UPDATE 触发器，触发事件是 UPDATE 语句：

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

假设表 TEACHER 有 1 000 行，如果定义的触发器为语句级触发器，那么执行完

UPDATE 语句后触发动作体执行一次；如果是行级触发器，触发动作体将执行 1 000 次。

（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。

（7）触发动作体

触发动作体既可以是一个匿名 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，用户可以在过程体中使用 NEW 和 OLD 引用 UPDATE/INSERT 事件之后的新值和 UPDATE/DELETE 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（即对数据库的增、删、改操作）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。

［**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另一个表 SC_U（Sno、Cno、Oldgrade、Newgrade）中，其中 Oldgrade 是修改前的分数，Newgrade 是修改后的分数。

```
CREATE TRIGGER SC_T                          /*SC_T 是触发器的名字*/
AFTER UPDATE OF Grade ON SC                  /*UPDATE OF Grade ON SC 是触发事件，*/
        /* AFTER 是触发的时机，表示当对 SC 的 Grade 属性修改完后再触发下面的规则*/
REFERENCING
    OLDROW AS OldTuple,
    NEWROW AS NewTuple
FOR EACH ROW      /*行级触发器，即每执行一次 Grade 的更新，下面的规则就执行一次*/
WHEN (NewTuple.Grade >= 1.1 * OldTuple.Grade)     /*触发条件，只有该条件为真时才执行*/
    INSERT INTO SC_U (Sno,Cno,OldGrade,NewGrade)          /*下面的 insert 操作*/
    VALUES(OldTuple.Sno,OldTuple.Cno,OldTuple.Grade,NewTuple.Grade)
```

在本例中 REFERENCING 指出引用的变量，如果触发事件是 UPDATE 操作并且有 FOR EACH ROW 子句，则可以引用的变量有 OLDROW 和 NEWROW，分别表示修改之前的元组和修改之后的元组。若没有 FOR EACH ROW 子句，则可以引用的变量有 OLDTABLE 和 NEWTABLE，OLDTABLE 表示表中原来的内容，NEWTABLE 表示表中变化后的部分。

［**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 Student-InsertLog 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
```

在本例中出现的 FOR EACH STATEMENT，表示触发事件 INSERT 语句执行完成后才执行一次触发器中的动作，这种触发器叫做语句级触发器。而例 5.21 中的触发器是行级触发器。默认的触发器是语句级触发器。DELTA 是一个关系名，其模式与 Student 相同，包含的元组是 INSERT 语句增加的元组。

［**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 4 000 元，如果低于 4 000 元，自动改为 4 000 元”。

```
CREATE TRIGGER Insert_Or_Update_Sal        /*对教师表插入或更新时激活触发器*/
BEFORE INSERT OR UPDATE ON Teacher         /*BEFORE 触发事件*/
REFERENCING NEW row AS newTuple
FOR EACH ROW                       /*这是行级触发器*/
BEGIN                              /*定义触发动作体，这是一个 PL/SQL 过程块*/
    IF (newtuple.Job='教授') AND (newtuple.Sal < 4000)
                                   /*因为是行级触发器，可在过程体中*/
        THEN newtuple.Sal :=4000;  /*使用插入或更新操作后的新值*/
    END IF;
END;                               /*触发动作体结束*/
```

因为定义的是 BEFORE 触发器，在插入和更新教师记录前就可以按照触发器的规则调整教授的工资，不必等插入后再检查再调整。

5.7.2 激活触发器

触发器的执行是由触发事件激活，并由数据库服务器自动执行的。一个数据表上可能定义了多个触发器，如多个 BEFORE 触发器、多个 AFTER 触发器等，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。

对于同一个表上的多个 BEFORE(AFTER)触发器，遵循“谁先创建谁先执行”的原则，即按照触发器创建的时间先后顺序执行。有些关系数据库管理系统是按照触发器名称的字母排序顺序执行触发器。

5.7.3 删除触发器

删除触发器的 SQL 语法如下：

```
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。

触发器是一种功能强大的工具，但在使用时要慎重，因为在每次访问一个表时都可能触发一个触发器，这样会影响系统的性能。

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。所谓正确是指符合现实世界语义的。本章讲解了关系数据库管理系统完整性实现的机制，包括完整性约束定义机制、完整性检查机制和违背完整性约束条件时关系数据库管理系统应采取的动作等。

在关系系统中，最重要的完整性约束是实体完整性和参照完整性，其他完整性约束条件则可以归入用户定义的完整性。

数据库完整性的定义一般由 SQL 的数据定义语言来实现。它们作为数据库模式的一部分存入数据字典中，在数据库数据修改时关系数据库管理系统的完整性检查机制将按照数据字典中定义的这些约束进行检查。

完整性机制的实施会影响系统性能。因此，许多数据库管理系统对完整性机制的支持比对安全性的支持要晚得多，也弱得多。随着硬件性能的提高以及数据库技术的发展，目前的关系数据库管理系统都提供了定义和检查实体完整性、参照完整性和用户定义的完整性的功能。

对于违反完整性的操作一般的处理是采用默认方式，如拒绝执行。对于违反参照完整性的操作，本书讲解了不同的处理策略。用户要根据应用语义来定义合适的处理策略，以保证数据库的正确性。

实现数据库完整性的一个重要方法是触发器。触发器和前面介绍的各种完整性约束不同之处是，完整性控制是当被限制的对象发生变化时系统就去检查该对象变化后能否满足完整性约束条件，如果不能满足就进行违约处理，违约处理通常比较简单。而触发器功能就要强得多，因为触发器规则中的动作体可以很复杂，通常是一段 SQL 存储过程。触发器不仅可以用于数据库完整性检查，也可以用来实现数据库系统的其他功能，包括数据库安全性，以及更加广泛的应用系统的一些业务流程和控制流程、基于规则的数据和业务控制功能等。不过也要特别注意，一个触发器的动作可能激活另一个触发器，最坏的情况是导致一个触发链，从而造成难以预见的错误。

习 题

1. 什么是数据库的完整性？

2. 数据库的完整性概念与数据库的安全性概念有什么区别和联系？

3. 什么是数据库的完整性约束条件？

4. 关系数据库管理系统的完整性控制机制应具有哪三方面的功能？

5. 关系数据库管理系统在实现参照完整性时需要考虑哪些方面？

6. 假设有下面两个关系模式：

职工（职工号，姓名，年龄，职务，工资，部门号），其中职工号为主码；

部门（部门号，名称，经理名，电话），其中部门号为主码。

用 SQL 语言定义这两个关系模式，要求在模式中完成以下完整性约束条件的定义：

（1）定义每个模式的主码；（2）定义参照完整性；（3）定义职工年龄不得超过 60 岁。

7. 在关系系统中，当操作违反实体完整性、参照完整性和用户定义的完整性约束条件时，一般是如何分别进行处理的？

8. 某单位想举行一个小型的联谊会，关系 Male 记录注册的男宾信息，关系 Female 记录注册的女宾信息。建立一个断言，将来宾的人数限制在 50 人以内。（提示，先创建关系 Female 和关系 Male。）

实 验

实验 3 数据库完整性语言

理解和掌握数据库完整性设计以及完整性语言的使用方法。掌握实体完整性、参照完整性和用户自定义完整性的定义和维护方法。掌握单属性和多属性的实体完整性和参照完整性的定义、修改、删除等各种基本功能；掌握列级完整性约束和表级完整性约束的定义方法；掌握创建表时定义完整性和创建表后定义实体完整性两种方法，并能够设计 SQL 语句验证完整性约束是否起作用。

实验 4 触发器

理解和掌握利用触发器实现较为复杂的用户自定义完整性约束的机制和方法。理解和掌握数据库触发器的分类，了解和掌握各类数据库触发器的设计和使用方法，包括创建、使用、删除、激活等各种基本功能，并能设计和执行相应的 SQL 语句验证触发器的有效性。

本章参考文献

[1] HAMMER M, MCLEOD D. Semantic Integrity in a Relational Data Base System. in Proceedings of

VLDB，1975.

（文献［1］讨论数据库管理系统中语义完整性的概念。）

[2] SCHMID J，SWENSON J. On the Semantics of the Relational Model. in SIGMOD，1975.

[3] CODD E F. Extending the Database Relational Model to Capture More Meaning. TODS，1979(4):4.

（文献［3］扩展了关系模型的语义表达能力。）

[4] STONEBRAKER M，WONG E. Access Control in a Relational Database Management System by Query Modification. Proceedings of the ACM Annual Conference，1974.

（文献［4］讨论 INGRES 数据库管理系统的完整性约束机制。）

[5] STONEBRAKER M R. Implementation of Integrity Constrains and Views by Query Modification. in Proceedings of SIGMOD，1975.

（文献［5］讨论 INGRES 数据库管理系统的完整性约束机制。）

[6] CHAMBERLIN D，et al. SEQUEL 2：A Unified Approach to Data Definition，Manipulation，and Control. IBM Journal of Research and Development，1976(20):6.

（文献［6］提出了 SYSTEM R 中 SQL 的 Integrity Assert（完整性断言）和触发器语句。）

[7] HAMMER M，SARIN S. Efficient Monitoring of Database Assertions. in SIGMOD，1978.

（文献［7］介绍了完整性断言的有效验证方法。）

[8] BERNSTEIN P，BLAUSTEIN B，CLARKE E. Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data. in Proceedings of VLDB，1980.

[9] HSU A，IMIELINSKY T. Integrity Checking for Multiple Updates. in SIGMOD，1985.