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
    - 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
- 为维护数据库的完整性，数据库管理系统必须实现如下功能
  - 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
  - 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
  - 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，就采取一定的动作，如拒绝（NO ACTION）执行该操作或级连（CASCADE）执行其他操作

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。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：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)
    );
```

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，关系数据库管理系统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

5.2 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：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。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),
     FOREIGN KEY (Sno) REFERENCES Student(Sno),
     FOREIGN KEY (Cno) REFERENCES Course(Cno)
    );
```

5.2.2 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

> 例：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) /*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

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）

> 例：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')),
     Sage SMALLINT,
     Sdept CHAR(20)
);
```

2. 属性上约束条件的检查和违规处理

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

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：当学生的性别是男时，其名字不能以Ms.打头

```sql
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.%')
);
```

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

1. 完整性约束命名子句

```sql
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` 短语、`FOREIGN KEY` 短语、`CHECK` 短语等

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

> 例：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

```sql
/*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 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：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(*)
        FROM Course,SC
        WHERE SC.Cno = Course.Cno AND Course.Cname = '数据库')
    );
```

2. 删除断言的语句格式

```
DROP ASSERTION <断言名>;
```

5.6 触发器

- 触发器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
  - 触发器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
- 不同的关系数据库管理系统触发器语法各不相同

5.6.1 定义触发器

```
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
- 表的拥有者才可以在表上创建触发器
- 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
- 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
- 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
- 触发器类型
    - 行级触发器（`FOR EACH ROW`）
    - 语句级触发器（`FOR EACH STATEMENT`）
- 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
- 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

> 例：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          /*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是触发时机，表示当对SC的Grade属性修改完后再触发下面的规则*/
REFERENCING
    OLD row  AS  OldTuple,
    NEW row 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)
```

5.6.2 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:
    - 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

5.6.3 删除触发器

```
DROP TRIGGER <触发器名> ON <表名>;
```