第5章

约束和触发器

编写数据库的应用程序会面临一个非常重要的问题，即当更新数据库时数据可能存在各种各样的错误，如手工录入数据时不小心录错等，因此应用程序需要对每个插入、删除或修改命令都编写相应的检查以确保数据正确。如果将这些检查保存在数据库中并由 DBMS 管理检查，则可以保证这些检查不会被遗忘执行，也避免了大量的重复工作。

SQL 将完整性约束作为数据库模式的一部分，数据库的完整性指数据的正确性和相容性。数据的正确性指数据是否符合现实世界语义、反映当前实际状况；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。SQL 允许在属性上、元组上和关系之间定义约束，DBMS 提供完整性约束的定义，提供完整性检查的方法并进行违约处理。

本章主要介绍关系间的参照约束、属性和元组上的约束及特定事件用触发器表示的约束。


约束

5.1 主码和外码约束

第 2 章关系模型的完整性中介绍了实体完整性和参照完整性。实体完整性利用主码约束来实现；参照完整性则通过外码约束来实现，保证出现在某个属性子集的值必须是同一个表或另一个表的主码的值。

5.1.1 主码约束

SQL 的 CREATE TABLE 语句中用 PRIMARY KEY 来定义主码约束，定义某个属性子集是主码，定义为主码的属性子集的值必须是非空且唯一的。

【例 5.1】 创建基本表 Student，用学号 Sno 作为主码。

```
CREATE TABLE Student
( Sno CHAR(12) PRIMARY KEY,
Sname VARCHAR(8),
Ssex CHAR(2)
Sage INT,
Sdept VARCHAR(20));
```

在上面的建表定义中，主码定义在属性上，即关键字 PRIMARY KEY 写在属

性的后面。这种主码约束定义称为基于属性的主码约束。

```
CREATE TABLE Student
( Sno CHAR(12) ,
Sname VARCHAR(8),
Ssex CHAR(2)
Sage INT,
Sdept VARCHAR(20),
PRIMARY KEY(Sno));
```

主码约束也可以定义在元组上,即主码定义单列一行写在属性列表后面。这种主码约束定义称为基于元组的主码约束。

用 PRIMARY KEY 定义关系的主码后,每当用户程序对基本表插入记录或对主码进行更新操作时,DBMS 将按照实体完整性规则自动进行检查。

(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的值是否为空,只要有一个为空就拒绝插入或修改。

通过检查 DBMS 保证了实体完整性。

【例 5.2】 创建成绩表 SC,用学号 Sno 和课程号 Cno 作为主码。

```
CREATE TABLE SC
(Sno  CHAR(9),
Cno  CHAR(4),
Grade  SMALLINT,
PRIMARY KEY (Sno, Cno));
```

对于多属性组成的主码,声明主码时必须用基于元组的主码约束方式。

5.1.2 主外码约束

关系之间的约束称为断言,外码约束是一个断言。外码约束要求定义为外码的属性子集的值是有意义的,即外码的值必须是其他某个属性子集的值。例如,选修表 SC 中的学号 Sno 被定义为外码,其引用了学生表 Student 中的学号 Sno,表示选修表 SC 中的学号值一定是学生表 Student 中的某个学生。SQL 用 FOREIGN KEY 声明基表的外码,并用 REFERENCE 声明其引用了哪个表的属性子集。声明外码时需要注意以下两点。

(1) 被引用的属性子集必须被声明为 PRIMARY KEY 或者 UNIQUE 约束,否则该属性子集不能作为外码。

(2) 定义为外码的属性子集的值要么在被引用的属性子集中出现过,要么为空。

SQL 提供了以下两种声明外码的方法。

(1) 在 CREATE TABLE 语句的属性类型之后声明其引用某个关系的某个属性,其语法格式如下。

```
REFERENCES <表名>(<属性名>)
```

【例 5.3】 在课程表 Course 中,属性 Cpno 表示本课程的先修课程的课程号,先修课程

本身也是一门课程，所以将 Cpno 作为外码，并引用 Course 表的主码 Cno。

```
CREATE TABLE Course
( Cno CHAR(4) PRIMARY KEY,
Cname CHAR(40),
Cpno CHAR(4) REFERENCES Course(Cno),
Ccredit  SMALLINT);
```

从课程表 Course 的定义中可以看出，外码引用的属性不一定是另外一个基表的主码，也可以是与外码同一个表的主码，例如，基本表 Course 的外码 Cpno 引用的就是自身表的主码 Cno。

(2) 可以在 CREATE TABLE 语句的属性列表中追加多个声明，用来说明这组属性是该基表的外码。SQL 外码的格式如下。

```
FOREIGN KEY (<属性名>) REFERENCES <表名>(<属性名>)
```

【例 5.4】 学生选课表 SC 中的两个属性：学号 Sno 和课程号 Cno，学号 Sno 必须是在学生表 Student 中出现，课程号 Cno 必须是在课程表 Course 中出现，故需要将这两个属性声明为外码。

```
CREATE TABLE SC
( Sno  CHAR(9),
Cno  CHAR(4),
Grade  SMALLINT,
PRIMARY KEY (Sno,Cno),
FOREIGN KEY (Sno) REFERENCES Student(Sno),
FOREIGN KEY (Cno) REFERENCES Course(Cno));
```

考虑基本表 SC，对其插入一个新的元组，其中，Sno 值非空，且其值没有出现在表 Student 的 Sno 属性中，这时按照主外码约束的定义，这个操作将被数据库拒绝。

如果修改基本表 SC 的 Cno 属性，修改后的 Cno 属性值非空，且其值没有出现在 Course 表的 Cno 属性中，这时按照主外码约束的定义，这个操作也将被数据库拒绝。

考虑基本表 Student，删除其中一个元组，且被删除的 Sno 值出现在基本表 SC 中，这时也违反了主外码约束，是否也会被数据库拒绝呢？

同样地，对基本表 Student 做更新操作，且旧的 Sno 值出现在基本表 SC 中，这时也违反了主外码约束，是否同样会被数据库拒绝呢？

对于上面两种对被引用关系上的更新操作，SQL 提供了以下三种处理策略。

(1) 默认原则：拒绝任何违反主外码约束的更新操作，该操作一般设置为默认原则。

(2) 级联原则：被引用属性子集的修改同样移到外码上，这样就保证了主外码约束。例如，在 Student 表中删除了某个学号，在级联原则下，就会从 SC 表中删除该学号的元组。如果将 Student 表中某个学生的学号 Sno1 修改为 Sno2，在级联原则下，数据库系统就会将 SC 表中 sno1 的学号修改为 Sno2。

(3) 置空值原则：当被引用关系的更新影响外码时，将外码的值置为空值(NULL)。例

如,在 Course 表中删除了某个课程号 Cno,在置空值原则下,就会将 Course 表中先修课程 Cpno 属性中值为 Cno 的改为 NULL。如果将 Course 表中某个课程的课程号 Cno1 修改为 Cno2,在置空值原则下,数据库系统就会将 Course 表中先修课程 Cpno 属性中值为 Cno1 的改为 NULL。

因此,表 SC 的插入、修改操作和表 Student 上的删除、修改操作都可能违反外码约束,从而破坏参照完整性规则,其违约处理策略归纳为表 5.1。

表 5.1 可能破坏参照完整性的情况及违约处理

外码引用的表(如 Student)	定义了外码的表(如 SC)	违 约 处 理
SC 插入的 Sno 值不存在	插入元组	拒绝
SC 更新后的 Sno 值不存在	修改外码值	拒绝
删除元组	存在与删除操作相关的元组	拒绝/级联删除/设置为空值
修改主码值	存在与修改操作相关的元组	拒绝/级联修改/设置为空值

上述三种策略中默认原则不需要做任何处理,级联原则通过 CASCADE 选项来定义,置空置原则用 SET NULL 来定义。SQL 针对 DELETE 和 UPDATE 操作可以定义不同的选项。

【例 5.5】 将课程表 Course 中外码 Cpno 对 DELETE 操作设置置空值原则,对 UPDATE 操作设置级联原则。

```
CREATE TABLE Course
(Cno CHAR(4) PRIMARY KEY,
Cname CHAR(40),
Cpno CHAR(4) REFERENCES Course(Cno)
ON DELETE SET NULL
ON UPDATE CASCADE,
Ccredit SMALLINT);
```

理论上,DELETE 操作和 UPDATE 操作都可以有两种选择,但一般情况下对于删除操作采用置空原则,对于更新操作采用级联原则更有实用意义。例如,假设某门课程 A 不适合当前教学,需要删除该门课程 A,则采用 A 作为先修课程的课程 B,采用置空原则,只需要将 B 的先修课程设为 NULL,而不是采用级联原则同样删除课程 B。同样,如果 A 课程的课程号 Cno1 改为 Cno2,则根据级联原则,将课程 B 的先修课程改为 Cno2,而不是采用置空原则,将课程 B 的先修课程改为 NULL。

5.2 基于属性和元组的约束

用户自定义完整性主要针对某个具体应用的数据所需满足的语义要求。DBMS 也同样提供了定义和检验这类完整性的机制。SQL 在 CREATE TABLE 语句中可以采用两种约束形式来定义用户自定义完整性。

(1) 在某个属性上的约束。

(2) 在整个元组上的约束。

如果约束涉及基本表的多个属性,那么必须采用基于元组的约束。如果约束只涉及一个属性,那么既可以采用基于元组的约束,也可以采用基于属性的约束。但是基于元组的约束比基于属性的约束更频繁地被数据库检查,因为只要元组的任意一个属性被改变都要被检查,无论被改变的属性是不是约束中涉及的属性。

5.2.1　非空值约束

在 SQL 中用 NOT NULL 来表示非空值约束,其作用是要求任何元组该属性的值不能为空。非空值约束在 CREATE TABLE 语句的属性声明后用 NOT NULL 声明。

【例 5.6】　将学生表 Student 中的性别属性设为非空值约束。

```
CREATE TABLE Student
(Sno CHAR(12) PRIMARY KEY,
Sname VARCHAR(8),
Ssex CHAR(2) NOT NULL
Sage INT,
Sdept VARCHAR(20));
```

在例 5.6 中,学生的性别不能为空,所以在插入元组时,必须要给定学生的性别,否则会导致插入失败。

如果声明非空值约束的属性同时是外码,这时对破坏参照完整性的违约处理不能用置空值原则,因为该原则与非空值约束冲突。

5.2.2　UNIQUE 约束

数据库规定 UNIQUE 约束要求该属性值是唯一的,与 PRIMARY KEY 不同的是,UNIQUE 约束允许属性值为空。

【例 5.7】　创建院系表 College,要求院系名称唯一,院系编码为主码。

```
CREATE TABLE College
(Cno CHAR(12) PRIMARY KEY,
Cname VARCHAR(8) UNIQUE,
Location VARCHAR(50));
```

对基表 College 执行插入或更新操作时,都必须先判断基表中 Cname 属性是不是唯一的,如果不是唯一的,数据库将拒绝执行该操作。

5.2.3　CHECK 子句

无论是非空值约束还是 UNIQUE 约束都是数据库上的简单约束,对于更复杂的要求是无法实现的,例如,性别只能是“男”或“女”,成绩不能高于 100 等。这些更高级、更复杂的约束用 CHECK 子句来完成。

CHECK 约束可以定义在属性上,也可以定义在元组上。基于属性的 CHECK 约束是对属性值的简单约束,如算术表达式或合法值的枚举等。

【例 5.8】 在学生表 Student 中,对 Ssex 和 Sage 两个属性增加 CHECK 约束,属性 Ssex 要求其值只能是“男”或“女”,属性 Sage 要求其值大于 0。

```
CREATE TABLE Student
(Sno CHAR(12) PRIMARY KEY,
Sname VARCHAR(8),
Ssex CHAR(2) NOT NULL CHECK(Ssex IN ('男' , '女')),
Sage INT CHECK(Sage>0),
Sdept VARCHAR(20));
```

基于属性的 CHECK 约束只有在插入元组或更新元组时对其属性值进行检查。例如,向 Student 表中插入一个元组,这时会检查插入元组的 Ssex 值是不是“男”或“女”中的一个,检查 Sage 的值是否大于 0,只有都满足时才能将该元组插入表中。如果修改 Student 表 Sage 的值,DBMS 会检查新值是否大于 0。

如果对学生表 Student 的元组进行了更新,但更新的属性不包括与 CHECK 约束相关的属性如 Sdept 属性,则 DBMS 不调用基于属性的 CHECK 约束检查。

如果一个约束条件同时涉及多个属性,那么就需要使用基于元组的 CHECK 约束。SQL 在 CREATE TABLE 语句的属性列表后追加 CHECK 约束,约束条件用括号括起来,约束条件可以是 WHERE 子句中出现的表达式。

【例 5.9】 假设学校规定护理系只能招收女同学,请在 Student 表中添加约束。

```
CREATE TABLE Student
( Sno CHAR(12) PRIMARY KEY,
Sname VARCHAR(8),
Ssex CHAR(2) NOT NULL CHECK(Ssex IN ('男' , '女')),
Sage INT CHECK(Sage>0),
Sdept VARCHAR(20)
CHECK (Sdept<>'护理系' or Ssex='女'));
```

例 5.9 中对于非护理系的同学和女同学的元组都满足 CHECK 约束条件,如果插入一个学生信息是护理系的男同学,则这次插入操作将被拒绝。

5.2.4 用户自定义域

域本质上是一种带有可选约束(对值集合上的限制)的数据类型。SQL 定义表时会为每个列指派一种数据类型(如字符型、整型等),这些数据类型提供了一个广泛域。

如果域还需要满足某些约束时,SQL 提供了用户自定义域。使用用户自定义域时,必须先创建该自定义域,其语法规则如下。

```
CREATE DOMAIN <域名> <数据类型> [CONSTRAINT <约束名>] CHECK (约束表达式)
```

【例 5.10】 创建一个分数域,类型为短整型且数据取值范围是[0,100]。

```
CREATE DOMAIN GradeDomain SMALLINT
CONSTRAINT GradeCheck CHECK( VALUE BETWEEN 0 AND 100);
```

这样，选课表 SC 的定义就可以改为

```
CREATE TABLE SC
(Sno CHAR(9),
Cno CHAR(4),
Grade GradeDomain,
PRIMARY KEY (Sno, Cno),
FOREIGN KEY (Sno) REFERENCES Student(Sno),
FOREIGN KEY (Cno) REFERENCES Course(Cno));
```

对于存在的域定义，用户可以用 DROP DOMAIN 删除，其语法格式为

```
DROP DOMAIN <域名> [CASCADE | RESTRICT];
```

关键字 CASCADE 表示级联删除，自动删除依赖于该域的对象（例如表的属性），然后删除所有依赖于那些对象的对象。

关键字 RESTRICT 表示如果有任何对象依赖于该域，则拒绝删除它，该选项是默认值。

【例 5.11】　删除域 GradeDomain。

```
DROP DOMAIN GradeDomain;
```

5.3　约束的修改

用户任何时候都可以添加和删除约束，但前面的主外码约束、基于属性和元组约束的定义中，约束都是匿名的，这样就难以对它们进行修改操作。SQL 提供了 CONSTRAINT 子句来对约束命名。

例如：

```
CREATE TABLE Student
( Sno CHAR(12) CONSTRAINT SnoKey PRIMARY KEY,
…
);
```

通过 CONSTRAINT 子句将主码约束命名为 SnoKey，后面就可以根据 SnoKey 对该约束进行删除、修改等操作。

SQL 中也可通过 ALTER TABLE 语句来增加或删除约束。用关键字 DROP 删除约束。例如：

```
ALTER TABLE Student DROP SnoKey;
```

这样就删除了 Student 表中的主码约束。

SQL 在 ALTER TABLE 语句中用关键字 ADD 来添加约束。例如：

```
ALTER TABLE Student ADD SnoKey RPIMARY KEY (Sno);
```

对于域中的约束,可以用 ALTER DOMAIN 来添加和删除约束。

【例 5.12】 将 GradeDomain 的取值范围改为[0,150]。

先删除 GradeDomain 的 CHECK 约束:

```
ALTER DOMAIN GradeDomain
DROP GradeCheck;
```

再新加 CHECK 约束,取值范围为[0,150]:

```
ALTER DOMAIN GradeDomain
ADD CONSTRAINT GC CHECK(VALUE>=0 AND VALUE<=150);
```

需要注意的是,通过 ADD 添加的约束必须是基于元组的约束,不能将其恢复到基于属性的约束。

5.4 断　　言

断言和触发器是 DBMS 中最强的约束形式,与特定的元组或元组分量无关,属于数据库模式的一部分,同基本表一样。断言是 SQL 逻辑表达式,只需要用户说明什么是真。

5.4.1 创建断言

每个断言(Assertion)都是一个谓词,其表达了数据库需要满足的一个条件,是具更一般性的约束,基于属性和元组的约束是断言的特殊形式。

SQL 提供了一种简单的断言形式,其语法格式如下。

```
CREATE ASSERTION <断言名> CHECK (<谓词>);
```

CHECK 子句中的约束条件与 WHERE 子句的条件表达式类似。断言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。断言的谓词必须为真,且要保持永远为真,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。

5.4.2 使用断言

断言与基于元组的 CHECK 约束在书写上是不同的,基于元组的 CHECK 约束能直接引用它在声明中出现的关系的属性,断言没有这些特权。断言条件中引用的任何属性都必须介绍,特别是要提及在 SELECT-FROM-WHERE 表达式中的关系。

在数据库中创建断言时,系统会检测该断言的有效性,如果断言是有效的,则以后对数据库的修改必须在满足断言的情况下才能生效。

断言的条件必须是逻辑值,因此采用某种方式聚集条件的结果,以获得 True 或 False 值。如 SELECT 操作产生的是一个集合,用 NOT EXISTS 来决断集合是否为空。另外,也

可以用 COUNT、SUM、AVG 之类的聚集函数，将其结果与某个常数比较来获得逻辑值。

【例 5.13】　护理系的学生必须是女生。

```
CREATE ASSERTION Nurse CHECK (
NOT EXISTS (SELECT Student.sno
FROM Student
WHERE Sdept='护理系' AND Ssex='男')
);
```

【例 5.14】　每个系的学生人数不能少于 100 人。

```
CREATE ASSERTION StuNum CHECK (100<= ALL
(SELECT COUNT(Sno) FROM Students GROUP BY Sdept));
```

CHECK 约束发生在对关系插入元组或属性修改时，并且若有子查询的话就不能确保成立(检查的是一条元组)，断言发生在对任何提及的关系做改变时，必须以某种方式聚集条件的结果(检查的是整个关系)。

该约束只涉及单个关系，似乎也可以用基于元组的 CHECK 约束而不用断言，如在表 Student 创建时增加基于元组的 CHECK 约束：

```
CHECK (100<= ALL (SELECT COUNT(Sno) FROM Students GROUP BY Sdept));
```

如果采用基于元组的 CHECK 约束，则对表做删除操作时将不做任何检查，但删除操作可能导致关系中的数据违反该约束，但如果用断言定义的约束，就不会出现违反约束的情况，因为断言检查的是整个表。不同约束类型的主要区别如表 5.2 所示。

表 5.2　基于属性的 CHECK、基于元组的 CHECK、断言的区别

约 束 类 型	声明的位置	动作的时间	确保不违反约束
基于属性的 CHECK	属性	对关系插入元组或属性修改时	如果 CHECK 包含子查询，则不能确保
基于元组的 CHECK	关系模式元素	对关系插入元组或属性修改时	如果 CHECK 包含子查询，则不能确保
断言	数据库模式元素	对任何涉及的关系做改变时	确保

断言可以被删除，删除断言与删除数据库模式的元素格式类似，使用 DROP ASSERTION 来删除断言，格式如下。

```
DROP ASSERTION <断言名>
```

【例 5.15】　删除断言 StuNum。

```
DROP ASSERTION StuNum;
```

如果创建的断言很复杂，则系统对该断言的检测会带来很大的开销。因此在数据库中使用断言应该谨慎，建议使用易于检测的简单断言。

5.5 触 发 器

触发器(Trigger)也称作事件-条件-动作(Event-Condition-Action,ECA)规则。触发器与前面介绍的约束有以下几点不同。


PG 编程逻辑

(1) 仅当用户设置的事件(Event)发生时,触发器才被激活。这些事件通常是对某个关系的插入、修改或删除操作。

(2) 当触发器被激活后,触发器测试触发的条件(Condition),如果条件不满足,则响应该事件的触发器不做任何动作。

(3) 如果触发器的条件满足,则由数据库执行该触发器相关的动作(Action)。这些动作可以是任何数据库的操作序列,例如,修改事件的结果,撤销相关事务等,也可以是 PL/SQL 程序。


触发器介绍

5.5.1 定义触发器

SQL 中使用 CREATE TRIGGER 来创建触发器,其语法规则如下。

```
CREATE TRIGGER <触发器名称>
{BEFORE|AFTER|INSTEAD OF}<触发事件> ON <表名|视图名>
REFERENCING {NEW|OLD} {ROW|TABLE} AS <变量名>
FOR EACH {ROW|STATEMENT}
[WHEN <条件>] <动作体>;
```


PG 触发器

对触发器的语法规则做如下说明。

(1) 触发器名称。

触发器名称可以包含模式名,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或视图必须在同一模式下。

(2) 触发事件。

触发事件可以是对基本表或视图的插入 INSERT、删除 DELETE 和更新 UPDATE 操作,也可以是它们的组合,如 INSERT OR DELETE 等,更新操作可以指定相关属性如 UPDATE OF <属性列>。

(3) 触发器的目标。

触发器可以定义在基本表上,也可以定义在视图上,称为触发器的目标。

(4) 触发时机。

BEFORE/AFTER 是触发的时机。BEFORE 表示在触发事件的操作执行之前激活触发器,AFTER 表示在触发事件的操作执行之后激活触发器。INSTEAD OF 用来定义视图的触发器。

(5) 触发器类型。

触发器类型分为行级触发器(FOR EACH ROW)和语句级触发器(FOR EACH STATEMENT)。行级触发器是触发语句每操作一行,触发器就被执行一次;而语句级类型触发器只被执行一次。