第五章 数据库完整性

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
  - 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

5.7 触发器

5.7.1 定义触发器

5.7.2 激活触发器

5.7.3 删除触发器

5.7.1 定义触发器

- **CREATE TRIGGER语法格式**

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>

- 表的拥有者才可以在表上创建触发器

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>

- 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

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>

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>
```

- 触发事件
  - **INSERT、DELETE或UPDATE**
  - 几个事件的组合
  - UPDATE OF<触发列，...>

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>
```

- **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

5.7.1 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>
```

- 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

5.7.1 定义触发器

❖ CREATE TRIGGER语法格式

触发器名和表名必须在同一模式下

```
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

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>

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>

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>

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

5.7.1 定义触发器

在【例5.11】的TEACHER表上创建一个AFTER UPDATE触发器，触发事件是UPDATE语句：

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

假设表TEACHER有1000行

- 如果是语句级触发器，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

5.7.1 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>

- 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

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
REFERENCING NEW|OLD ROW AS
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
- 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

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>
```

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。

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>

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。

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>
```

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。

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>
```

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。

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>
```

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。

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>
```

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。

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>
```

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。

触发器又叫做事件-条件-动作（event-condition-action）规则。

定义触发器（续）

[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下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

修改后的分数

修改前的分数

```
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)
```

定义触发器（续）

[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下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

修改后的分数

修改前的分数

```
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)
```

定义触发器（续）

[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下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

修改后的分数

修改前的分数

```
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)
```

定义触发器（续）

[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下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

修改后的分数

修改前的分数

```
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)
```

定义触发器（续）

[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下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

修改后的分数

修改前的分数

```
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)
```

定义触发器（续）

[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下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

修改后的分数

修改前的分数

```
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)
```

定义触发器（续）

[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下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

修改后的分数

修改前的分数

```
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)
```

定义触发器（续）

[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下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

修改后的分数

修改前的分数

```
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)
```

定义触发器（续）

[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下面表中：

SC_U（Sno,Cno,Oldgrade,Newgrade）

修改后的分数

修改前的分数

```
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)
```

定义触发器（续）

[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StudentInsertLog中。

```
CREATE TRIGGER Student_Count
AFTER INSERT ON Student
REFERENCING
        NEW TABLE AS DELTA
FOR EACH STATEMENT
        INSERT INTO StudentInsertLog (Numbers)
        SELECT COUNT(*) FROM DELTA
```

定义触发器（续）

[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StudentInsertLog中。

```
CREATE TRIGGER Student_Count
AFTER INSERT ON Student
REFERENCING
        NEW TABLE AS DELTA
FOR EACH STATEMENT
        INSERT INTO StudentInsertLog (Numbers)
        SELECT COUNT(*) FROM DELTA
```

定义触发器（续）

[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StudentInsertLog中。

```
CREATE TRIGGER Student_Count
AFTER INSERT ON Student
REFERENCING
        NEW TABLE AS DELTA
FOR EACH STATEMENT
        INSERT INTO StudentInsertLog (Numbers)
        SELECT COUNT(*) FROM DELTA
```

定义触发器（续）

[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StudentInsertLog中。

```
CREATE TRIGGER Student_Count
AFTER INSERT ON Student
REFERENCING
        NEW TABLE AS DELTA
FOR EACH STATEMENT
        INSERT INTO StudentInsertLog (Numbers)
        SELECT COUNT(*) FROM DELTA
```

定义触发器（续）

[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StudentInsertLog中。

```
CREATE TRIGGER Student_Count
AFTER INSERT ON Student
REFERENCING
        NEW TABLE AS DELTA
FOR EACH STATEMENT
        INSERT INTO StudentInsertLog (Numbers)
        SELECT COUNT(*) FROM DELTA
```

定义触发器（续）

[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StudentInsertLog中。

```
CREATE TRIGGER Student_Count
AFTER INSERT ON Student
REFERENCING
        NEW TABLE AS DELTA
FOR EACH STATEMENT
        INSERT INTO StudentInsertLog (Numbers)
        SELECT COUNT(*) FROM DELTA
```

定义触发器（续）

[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4000元，如果低于4000元，自动改为4000元”。

这条规则能否在CREATE TABLE语句中定义?

完整性规则：教授的工资不得低于4000元
违约反应：自动改为4000元

完整性规则：教授的工资不得低于4000元
违约反应：拒绝执行

定义触发器（续）

[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
FOR EACH ROW
 BEGIN
    IF (new.Job='教授') AND (new.Sal < 4000)
    THEN  new.Sal :=4000;
    END IF;
 END;
```

定义触发器（续）

[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
FOR EACH ROW
 BEGIN
    IF (new.Job='教授') AND (new.Sal < 4000)
    THEN  new.Sal :=4000;
    END IF;
 END;
```

定义触发器（续）

[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
FOR EACH ROW
 BEGIN
    IF (new.Job='教授') AND (new.Sal < 4000)
    THEN  new.Sal :=4000;
    END IF;
 END;
```

定义触发器（续）

[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
FOR EACH ROW
 BEGIN
    IF (new.Job='教授') AND (new.Sal < 4000)
    THEN  new.Sal :=4000;
    END IF;
 END;
```

定义触发器（续）

[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
FOR EACH ROW
 BEGIN
    IF (new.Job='教授') AND (new.Sal < 4000)
    THEN  new.Sal :=4000;
    END IF;
 END;
```

5.7 触发器

5.7.1 定义触发器

5.7.2 激活触发器

5.7.3 删除触发器

5.7.2 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:

（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。

5.7 触发器

5.7.1 定义触发器

5.7.2 激活触发器

5.7.3 删除触发器

5.7.3 删除触发器

- 删除触发器的SQL语法：

```
DROP TRIGGER <触发器名> ON <表名>;
```

5.7.3 删除触发器

❖ 删除触发器的SQL语法：

DROP TRIGGER <触发器名> ON <表名>;

5.7.3 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。

第五章 数据库完整性

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
- 关系数据库管理系统完整性实现的机制
  - 完整性约束定义机制
    - 在CREATE TABLE中定义
    - 用断言定义
    - 用触发器定义
  - 完整性检查机制
  - 违背完整性约束条件时RDBMS采取的动作

小结（续）

- 本章目标
  - 掌握什么是数据库的完整性
  - 掌握用SQL语言定义关系模式的完整性约束条件
- 本章重点
  - 牢固掌握DBMS完整性控制的实现机制
  - 举一反三：用SQL语言定义关系模式的完整性约束条件
- 本章难点
  - RDBMS如何进行违约处理，其中比较复杂的是参照完整性的实现机制。