第五章　数据库完整性

数据库完整性

- 数据库的完整性
  - 数据的正确性和相容性

- 数据的完整性和安全性是两个不同概念
  - 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

数据库完整性(续)

为维护数据库的完整性，DBMS必须：

- **1.**提供定义完整性约束条件的机制
- **2.**提供完整性检查的方法
- **3.**违约处理

第五章　课程内容矩阵

知识单元	内容与要求		掌握程度
	一级知识点	二级知识点	
数据库完整性	实体完整性	定义、主键（主码）、检查、违约处理	3
	参照完整性	定义、外键（外码）、完整性检查、违约处理、Cascade操作、No Action操作	3
	用户定义完整性	属性约束、属性约束定义、属性约束检查、元组约束、元组约束定义、元组约束检查、表中完整性约束修改、违约处理	3
	域完整性**	属性取值范围	2
	触发器	定义、类型、激活、触发事件、触发条件、触发动作体	3

第五章 数据库完整性

5.1 实体完整性

5.2 参照完整性

5.3 用户定义的完整性

5.4 完整性约束命名字句

*5.5 域中的完整性限制

5.6 触发器

5.7 小结

5.1 实体完整性

- 5.1.1 实体完整性定义
- 5.1.2 实体完整性检查和违约处理

5.1.1 实体完整性定义

- 关系模型的实体完整性
  - **CREATE TABLE中用PRIMARY KEY定义**
- 单属性构成的码有两种说明方法
  - 定义为列级约束条件
  - 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
  - 定义为表级约束条件

实体完整性定义(续)

［例1］ 将Student表中的Sno属性定义为码

(1)在列级定义主码

```
CREATE TABLE Student
 (Sno  CHAR(9)  PRIMARY KEY，
  Sname  CHAR(20) NOT NULL，
  Ssex  CHAR(2) ，
  Sage  SMALLINT，
  Sdept  CHAR(20));
```

实体完整性定义(续)

(2)在表级定义主码

```
CREATE TABLE Student
 (Sno  CHAR(9),
  Sname  CHAR(20) NOT NULL,
  Ssex  CHAR(2) ,
  Sage  SMALLINT,
  Sdept  CHAR(20),
  PRIMARY KEY (Sno)
);
```

实体完整性定义(续)

［例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 实体完整性

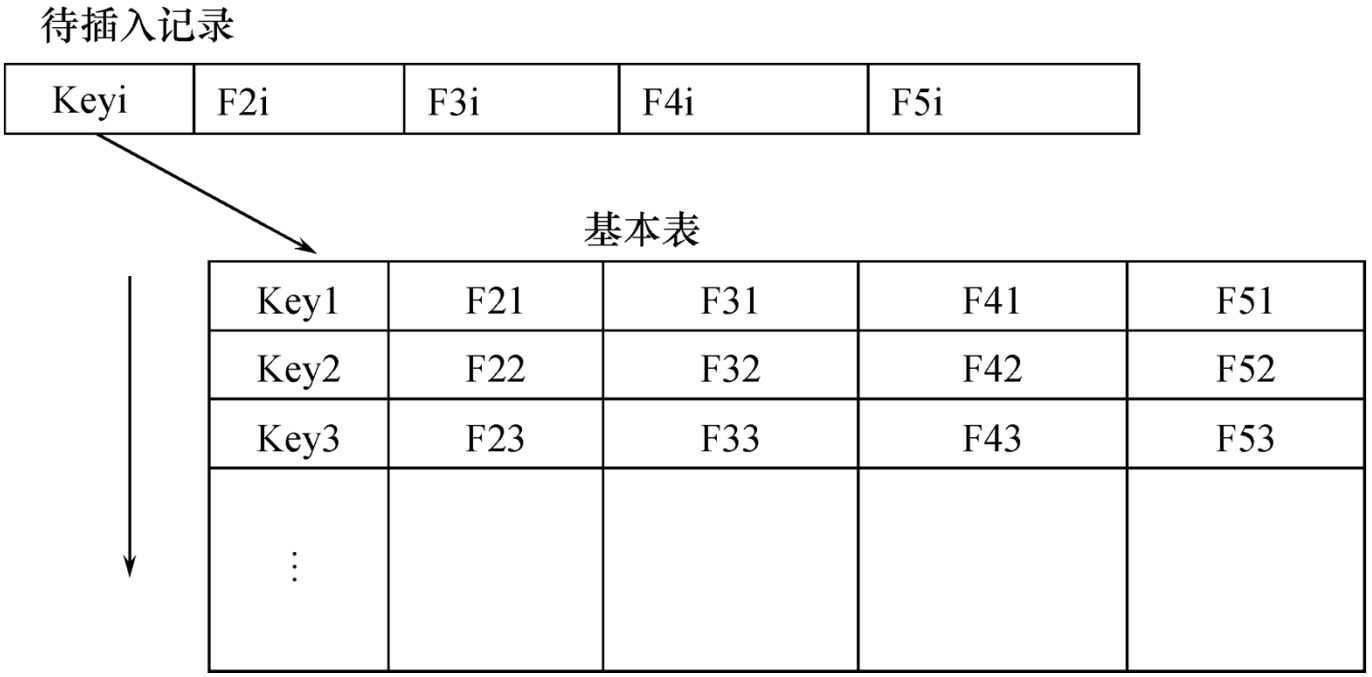
- 5.1.1 实体完整性定义
- 5.1.2 实体完整性检查和违约处理

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，RDBMS按照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

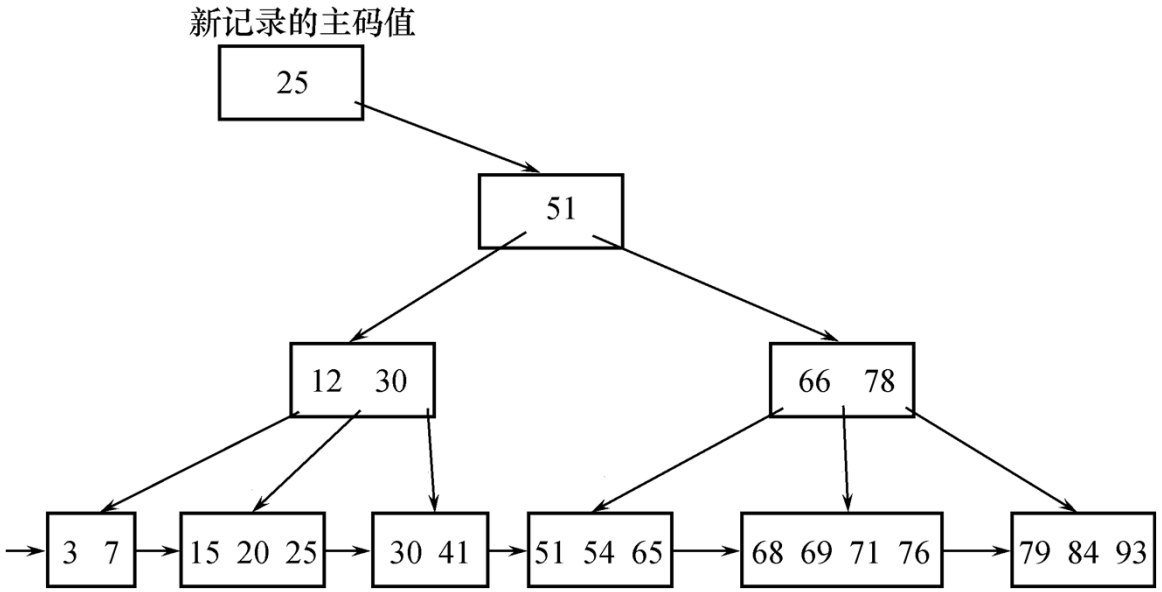
实体完整性检查和违约处理(续)

- 检查记录中主码值是否唯一的一种方法是进行全表扫描



实体完整性检查和违约处理(续)

- 索引



第五章 数据库完整性

5.1 实体完整性

5.2 参照完整性

5.3 用户定义的完整性

5.4 完整性约束命名字句

*5.5 域中的完整性限制

5.6 触发器

5.7 小结

5.2 参照完整性

- 5.2.1 参照完整性定义
- 5.2.2 参照完整性检查和违约处理

5.2.1 参照完整性定义

- 关系模型的参照完整性定义
  - 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

参照完整性定义(续)

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 参照完整性

- **5.2.1** 参照完整性定义
- **5.2.2** 参照完整性检查和违约处理

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

违约处理

- 参照完整性违约处理
  - **1. 拒绝(NO ACTION)执行**
    - 默认策略
  - **2. 级联(CASCADE)操作**
  - **3. 设置为空值（SET-NULL）**
    - 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

违约处理(续)

［例4］ 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)
        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*/
   );
```

第五章 数据库完整性

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
- RDBMS提供，而不必由应用程序承担

5.3 用户定义的完整性

- 5.3.1 属性上的约束条件的定义
- 5.3.2 属性上的约束条件检查和违约处理
- 5.3.3 元组上的约束条件的定义
- 5.3.4元组上的约束条件检查和违约处理

5.3.1 属性上的约束条件的定义

- CREATE TABLE时定义
  - 列值非空（**NOT NULL**）
  - 列值唯一（**UNIQUE**）
  - 检查列值是否满足一个布尔表达式（**CHECK**）

属性上的约束条件的定义(续)

1.不允许取空值

［例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，
  Cno  CHAR(4)  NOT NULL，
  Grade  SMALLINT NOT NULL，
  PRIMARY KEY (Sno， Cno)，
  /* 如果在表级定义实体完整性，隐含了Sno，Cno不允许取空值
，则在列级不允许取空值的定义就不必写了 * /
）；
```

属性上的约束条件的定义(续)

2.列值唯一

［例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，/*要求Dname列值唯一*/
     Location  CHAR(10),
     PRIMARY KEY (Deptno)
    );
```

属性上的约束条件的定义(续)

- **3. 用CHECK短语指定列值应该满足的条件**

［例7］ 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’) )，
      /*性别属性Ssex只允许取'男'或'女' */
    Sage  SMALLINT，
    Sdept  CHAR(20)
   );
```

5.3 用户定义的完整性

- 5.3.1 属性上的约束条件的定义
- 5.3.2 属性上的约束条件检查和违约处理
- 5.3.3 元组上的约束条件的定义
- 5.3.4元组上的约束条件检查和违约处理

5.3.2 属性上的约束条件检查和违约处理

- 插入元组或修改属性的值时，**RDBMS**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 用户定义的完整性

- 5.3.1 属性上的约束条件的定义
- 5.3.2 属性上的约束条件检查和违约处理
- 5.3.3 元组上的约束条件的定义
- 5.3.4元组上的约束条件检查和违约处理

5.3.3 元组上的约束条件的定义

- 在CREATE TABLE时可以用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

元组上的约束条件的定义(续)

［例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

5.3 用户定义的完整性

- 5.3.1 属性上的约束条件的定义
- 5.3.2 属性上的约束条件检查和违约处理
- 5.3.3 元组上的约束条件的定义
- 5.3.4 元组上的约束条件检查和违约处理

5.3.4 元组上的约束条件检查和违约处理

- 插入元组或修改属性的值时，**RDBMS**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

第五章 数据库完整性

5.1 实体完整性

5.2 参照完整性

5.3 用户定义的完整性

5.4 完整性约束命名子句

*5.5 域中的完整性限制

5.6 触发器

5.7 小结

5.4 完整性约束命名子句

- CONSTRAINT 约束

CONSTRAINT <完整性约束条件名>

［**PRIMARY KEY**短语

|FOREIGN KEY短语

|CHECK短语］

完整性约束命名子句(续)

［例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、C4四个列级约束。

完整性约束命名子句(续)

- **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

完整性约束命名子句(续)

［例13］ 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

- 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);
```

第五章 数据库完整性

5.1 实体完整性

5.2 参照完整性

5.3 用户定义的完整性

5.4 完整性约束命名字句

***5.5** 域中的完整性限制

5.6 触发器

5.7 小结

5.5 域中的完整性限制

- 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。

［例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10］中对Ssex的说明可以改写为

```
Ssex GenderDomain
```

［例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') );
```

域中的完整性限制(续)

［例16］删除域GenderDomain的限制条件GD。

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

［例17］在域GenderDomain上增加限制条件GDD。

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

- ✓ 通过［例16］和［例17］，就把性别的取值范围由('男'，'女')改为 ('1'，'0')

补充：断言

- 定义

  CREATE ASSERTION <断言名> CHECK <条件>

  断言是谓词，表达数据库总应该满足的条件

- 一旦定义了断言，系统验证其有效性，并且对每个可能违反该断言的更新操作都进行检查
- 这种检查会带来巨大的系统负载，因此应该谨慎使用断言
- 对断言“所有X, P(X)”，是通过检查“not exists X, ¬P(X)”来实现的

断言

- 示例：不允许男同学选修张老师课程

```
create assertion  ASSE1  check
  (not exists
          (select *
           from SC
           where C# in
                   (select C#
                    from C
                    where TEACHER = ‘张’ )
           and S# in
                   (select S#
                    from S
                    where SEX = ‘M’)))
```

断言

- 示例：每门课最多50名男同学选修

```
create assertion ASSE2 check
  (50 >= all  (select count(SC.S#)
                from S, SC
                where S.S# = SC.S#
                      and  SEX = 'M'
                group by C#)))
```

- 撤消：drop assertion 断言名

```
drop assertion ASSE1
```

综合练习

考虑下面的关系模式：

- 研究人员（人员编号，姓名，年龄，职称）
- 项目（项目编号，名称，负责人编号，类别）
- 参与（项目编号，人员编号，工作时间）/*一个研究人员可以参加多个项目，一个项目有多个研究人员参加，工作时间给出某研究人员参加某项目的月数*/

一、写出下面的完整性约束：

（1）定义三个关系中的主码、外码、参照完整性；

（2）每个研究人员的年龄不超过35岁；

（3）每个研究人员 的职称只能是“讲师”、“副教授”或“教授”；

- （4）一个研究人员参加各种项目的总工作时间不能超过12个月；
- （5）每个项目至少有5位研究人员；
- （6）每个研究人员参加的项目数不能超过3个。（4）-（6）用断言实现

二、考虑上述关系模式，使用ALTER TABLE ADD CONSTRAINT声明如下完整性约束：

（1）负责人编号参照研究人员的“人员编号属性”，当对“研究人员”更新时，若违反约束则拒绝操作；

（2）同（1），但当违反约束时将负责人编号置为NULL；

（3）同（1），但当违反约束时将项目中的相应元组删除或修改；

（4）工作时间在1到12之间；

（5）项目名称不能为空。

三、使用CHECK短语写出“项目”关系的参照完整性约束。

一、（1）Create table 研究人员

(人员编号 int primary key,

姓名 char(8),

年龄 smallint check(年龄<=35),

职称 char(8) check(职称 in('讲师','副教授','教授'))

);

（2）Create table 项目

(项目编号 int primary key,

名称 char(20),

负责人编号 int,

类别 char(8) ,

Foreign key(负责人编号) references 研究人员(人员编号)

);

（3）Create table 参与

(项目编号 int,

人员编号 int,

工作时间 smallint,

Primary key(项目编号 ,人员编号),

Foreign key(项目编号) references 项目(项目编号),

Foreign key(人员编号) references 项目(人员编号)

);

（4）Create assertion 工作时间限制

Check(12>=all(select sum(工作时间) from 参与 group by 人员编号));

（5）Create assertion 项目参加人数

Check (5<=all(select count(人员编号)from 参与 group by 项目编号));

（6）Create assertion 研究员参加项目

Check (3>all(select count(项目编号) from 参与 group by 人员编号));

二、（1）Alter table 项目add constraint c1 foreign key(负责人编号) references 研究人员(人员编号) on update no action;

（2）Alter table 项目add constraint c2 foreign key(负责人编号) references 研究人员(人员编号) on delete set null on update set null;

（3）Alter table 项目add constraint c3 foreign key(负责人编号) references 研究人员(人员编号) on delete cascade on update cascade;

（4）Alter table 项目参与 add constraint c4 check(工作时间>=1 and 工作时间<=12) ;

（5）Alter table 项目 add constraint c5 check(名称 is not null) ;

三、Check (负责人编号 in(select 人员编号 from 研究人员));

第五章 数据库完整性

5.6 触发器

- 触发器（Trigger）是一种特殊类型的存储过程，类似于其他编程语言中的事件函数，当有操作影响到触发器保护的数据时，触发器就会自动发生。
- 触发器也是保护完整性的一种重要方法，可以进行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。与存储过程不同的是，触发器主要是通过事件进行触发而被执行，而存储过程通过存储过程名字被直接调用。

5.6 触发器（续）

触发器的优点

1）触发器可以实现比CHECK约束更为复杂的数据完整性约束。在数据库中为了实现数据完整性约束，可以使用CHECK约束或触发器。CHECK约束不允许引用其它表中的列来完成检查工作，而触发器可以引用其它表中的列。例如，在STUDENT数据库中，向学生表中插入记录时，当输入所在院系时，必须先检查院系表中是否存在该系。这只能通过触发器实现，而不能通过CHECK约束完成。

2）触发器可以评估数据修改前后的表状态，并根据其差异采取对策。

3）一个表中可以存在多个同类触发器（INSERT、UPDATE或DELETE），对于同一个修改语句可以有多个不同的对策以响应。

5.6 触发器（续）

在进行触发器的基本操作之前，介绍两张特殊的临时表，分别是**inserted表和deleted表**。这两张表都存在于高速缓存中。用户可以使用这两张临时表来检测某些修改操作所产生的效果。例如，可以使用**SELECT** 语句来检查**INSERT**、**UPDATE**、**DELETE**语句执行的操作是否成功，触发器是否被这些语句触发等。但是**不允许用户直接修改inserted表和deleted表中数据**。

5.6 触发器（续）

deleted表中存储着被**DELETE**和**UPDATE**语句影响的旧数据行。在执行**DELETE**和**UPDATE**语句过程中，指定的数据行被用户从基本表中删除，然后转移到了**deleted**表中。一般来说，在基本表中**deleted**表中不会存在有相同的数据行。

inserted表中存储着被**INSERT** 和**UPDATE**语句影响的新的数据行。当用户执行**INSERT** 和**UPDATE**语句时，新的数据行被添加到基本表中，同时这些数据行的备份被复制到**inserted**临时表中。

一个典型的**UPDATE事务**实际上是由两个操作组成。首先，旧的数据行从基本表中转移到**deleted**表中，前提是这个过程没有出错；紧接着将新的数据行同时插入基本表和**inserted**表。

Oracle提供了5种类型的触发器：

- 语句触发器
- 行触发器
- INSTEAD OF 触发器
- 系统条件触发器
- 用户事件触发器

（1）创建语句触发器

- 语句触发器是在表上或某些情况下的视图上执行的特定语句或者语句组上的触发器，其能够与INSERT、UPDATE、DELETE或者组合上进行关联。
- **语句触发器只会针对指定语句激活一次。**
- **无论UPDATE有多少行记录，也只会调用一次UPDATE语句触发器。**

- **CREATE OR REPLACE TRIGGER 触发器名称**
- **BEFORE | AFTER | INSTEADOF** INSERT | UPDATE | DELETE
- **ON 数据表名**
- **BEGIN**
- **　触发执行语句体**
- **END;**

例：数据库ORCL中有学生表student，选修表SC。创建一个语句触发器TRI_YJ。

- CREATE OR REPLACE TRIGGER TRI_YJ
- BEFORE UPDATE ON STUDENT
- BEGIN
- IF UPDATING THEN
- DBMS_OUTPUT.PUTLINE('触发器语句触发器');
- END IF;
- END;
- /

- Oracle支持对INSERT、UPDATE、DELETE3种操作触发器，因此上例中可以将UPDATE替换为INSERT或DELETE或者加上这两种操作，以OR连接，即表示插入、修改、删除数据都会触发该触发器。
- CREATE OR REPLACE TRIGGER TRI_YJ
- BEFORE INSERT OR UPDATE OR DELETE ON STUDENT
- BEGIN
- IF UPDATING THEN
- DBMS_OUTPUT.PUT_LINE(‘触发器语句触发器’);
- END IF;
- END;

（2）语句触发器被触发

- 与存储过程不同的是，触发器不需要用户手动去调用，当用户执行了某些操作时该触发器自动运行，称为被触发。前例只是创建了一个语句级触发器，下面对表Student进行Update操作，使触发器TRI_YJ被触发。
- 当用户修改Student表年龄为26的学生数据时，触发器会被触发。
- SET SERVEROUTPUT ON
- UPDATE STUDENT
- SET SAGE=SAGE+1 WHERE SAGE=26
- /
- 在SQL*Plus环境中会显示一次“触发语句触发器”字样。

（3）查看触发器

- 创建了一个触发器后，如果用户需要查看触发器的内容、状态等信息，Oracle提供了User_source视图和User_Triggers视图存储有关触发器的信息，用户可以对这两个视图进行查询操作。
- 例：查看前面创建的语句触发器TRI_YJ的名称、状态和内容信息。
- SELECT TRIGGER_NAME,TRIGGER_TYPE
- FROM USER_TRIGGERS
- WHERE TABLE_NAME='STUDENT'
- /
- SELECT TEXT FROM USER_SOURCE
- WHERE TYPE='TRIGGER' AND NAME='TRI_YJ'
- /

- 事实上，也可以在GUI界面的第三方工具PL SQL Developer中找到触发器，点击右键查看命令打开该触发器的具体内容。

（4）创建并触发行触发器

- 前面实例创建了语句触发器，其在一个数据表中只触发一次。而行触发器是指为受到影响的各个行激活的触发器，即每影响到一行记录就触发一次。行触发器的定义与语句触发器类似，只是需要加上For each row参数。
- 例：创建一个行触发器TRI_HJ，该触发器的功能是当用户对数据表Student进行Update操作时，每更新一行记录，显示一次“该行记录已更新”字符串。

- CREATE OR REPLACE TRIGGER TRI_HJ
- AFTER UPDATE ON STUDENT
- FOR EACH ROW
- BEGIN
- IF UPDATING THEN
- DBMS_OUTPUT.PUT_LINE('该行记录已更新');
- END IF;
- END;
- /

- 对student表进行update操作，触发触发器TRI_HJ。
- UPDATE STUDENT
- SET SAGE=SAGE+1
- WHERE SDEPT='IS'
- /

- 如果IS院系有3行记录，则会显示3行“该行记录已更新”。除此外，还在之前显示“触发语句触发器”，因为语句触发器TRI_YJ对STUDENT表的UPDATE操作也会被触发

（5）INSERT触发器

- 在许多应用程序中，当用户插入一行数据，程序要能自动执行一系列命令，为实现该功能创建的触发器称为INSERT触发器。
- 例：创建一个INSERT触发器，当用户往 STUDENT 表中插入一行数据后，显示“已插入一行数据至数据表STUDENT”。

- CREATE OR REPLACE TRIGGER TRI_IN
- AFTER INSERT ON STUDENT
- BEGIN
- DBMS_OUTPUT.PUT_LINE(‘已插入一行数据至数据表STUDENT’);
- END;
- /
- SET SERVEROUTPUT ON
- INSERT INTO STUDENT VALUES(‘950010’,’李红’,’女’,20,’CS’);
- /

（6）UPDATE指定列触发器

- 前面实例UPDATE触发器实现时使用的都是“BEFORE/AFTER UPDATE ON 表名”的结构，其功能是当用户对表中任意列有UPDATE操作时都会触发该触发器，然而在具体程序中，有些应用要求触发器只有在修改表指定的一列或几列时才会被触发，此时就要求对表的指定列创建UPDATE触发器。
- 例：对表STUDENT 的SNO列创建指定列触发器，当用户试图修改SNO 列的数据时，显示“不能修改SNO列的数据”。

表示用户修改表的哪一列时触发

自定义错误

输出错误信息

- CREATE OR REPL… …R_UP
- BEFORE UPDATE OF SNO… …E
- FOR EACH ROW
- BEGIN
- RAISE_APPLICATION_ERROR(-20001,’不能修改SNO列的数据’);
- END;
- /
- SET SERVEROUTPUT ON
- UPDATE STUDENT SET SNO=‘95001’ WHERE SNAME=‘李勇’
- /

（7）DELETE触发器

- 当用户准备删除表中的某些数据时，如果待删除的数据是受保护的数据，用户不能删除，此时就应该创建一个DELETE触发器，在用户试图执行删除操作的时候阻止并显示提示信息。
- 例：创建一个DELETE触发器TRI_DE，功能是当用户试图删除STUDENT表中的任意一行数据时阻止并显示“不能删除STUDENT表中的数据”。

- CREATE OR REPLACE TRIGGER TRI_DE
- BEFORE DELETE ON STUDENT
- FOR EACH ROW
- BEGIN
- RAISE_APPLICATION_ERROR(-20
  除STUDENT表中的数据');
- END;
- /
- SET SERVEROUTPUT ON
- DELETE FROM STUDENT WHERE
  SNO='95001'

与INSERT触发器的实现类似，区别在于此处使用了行触发器，且要求在用户执行DELETE操作之前触发，因此使用BEFORE子句；而INSERT触发器要在用户执行INSERT操作后触发，因此使用AFTER子句实现。

如果用户要求不能删除某些指定的行，而非STUDENT表的任意行，可以在BEGIN…..END语句块中加入IF判断。

（8）创建INSTEAD OF 触发器

- INSTEAD OF 触发器是在视图上而不是在数据基本表上定义的触发器，其是用来替换所使用实际语句的触发器。
- 例：事先建立了一个视图V_GRADE，该视图为信息系的所有学生的基本信息及所修课程和成绩。该视图涉及基本表STUDENT和SC表。向视图V_GRADE中插入一行数据（’950011’,’张明’,25,’男’,’2’,90）。
- 执行：INSERT INTO V_GRADE
- VALUES （’950011’,’张明’,25,’男’,’2’,90）
- /
- 出现错误提示：无法通过连接视图修改多个基表

- 此时，可以为视图V_GRADE创建一个进行INSERT操作的INSTEAD OF 触发器，该触发器的功能是将INSERT语句中的值分别插入到基本表STUDENT和SC中。
- CREATE OF REPLACE TRIGGER tr_v_grade
- INSTEAD OF INSERT ON V_GRADE
- FOR EACH ROW
- BEGIN
- INSERT INTO STUDENT(SNO,SNAME,SAGE,SSEX,SDEPT)
- VALUES (:NEW.SNO,:NEW.SNAME,:NEW.SAGE,:NEW.SSEX,:NEW.SDEPT);
- INSERT INTO SC(SNO,CNO,GRADE)
- VALUES(:NEW.SNO,:NEW.CNO,:NEW.GRADE);
- END;

INSERT INTO V_GRADE VALUES （’950011’,’张明’,25,’男’,’2’,90）
/
可执行完成。

- 本实例使用INSTEAD OF 触发器实现了对涉及两个基本表连接的视图进行插入操作，可以看出，通过INSTEAD OF 触发器可以将所有的视图变成可更新的，实现所有数据来源于多个表的视图可更新。
- 基于多个表的视图必须使用INSTEAD OF 触发器来支持引用多个表中数据的插入、更新和删除操作。
- 在视图上，每个INSERT 、UPDATE 或DELETE 语句最多可以定义一个INSTEAD OF 触发器。

- 使用INSTEAD OF 触发器需要注意4个事项：
- （1）只能被创建在视图上，并且该视图没有指定WITH CHECK OPTION 选项
- （2）不能指定BEFORE或AFTER选项
- （3）FOR EACH ROW 子句是可选的，即INSTEAD OF 触发器只能在行级上触发，或只能是行级触发器，没有必要指定
- （4）没有必要在针对一个表的视图上创建INSTEAD OF 触发器

（9）创建用户事件触发器

- 用户事件触发器是在用户事件上触发的触发器，一般包括：用户登录、注销、修改结构等，可以在CREATE、ALTER、DROP等DDL 操作或数据库系统上被触发。
- 用户事件以DDL关键字表示，数据对象以SCHEMA表示。其中DDL代表对数据库对象进行操作的各种SQL 语句，SCHEMA代表各种数据库对象，如基本表、视图、索引等。

- 例：创建一个用户事件触发器TRI_YH，功能是当用户执行了诸如创建表、删除表之类的数据定义语句则输出“执行了DDL语句”。
- CREATE OR REPLACE TRIGGER TRI_YH
- AFTER DDL ON SCHEMA
- BEGIN
- DBMS_OUTPUT.PUT_LINE(‘执行了DDL语句’);
- END；
- /
- 执行SET SERVEROUTPUT ON
- DROP VIEW TEST
- /
- 显示“执行了DDL语句”。

（10）创建系统事件触发器

- 系统事件触发器是在系统事件上触发的触发器，系统事件一般包括数据库的启动、关闭，用户的登录与退出，服务器错误等事件。Oracle提供的这5种事件用于触发系统事件触发器，且其触发时机也已限制。

事件	允许的时机	说明
STARTUP	AFTER	启动数据库实例之后触发
SHUTDOWN	BEFORE	关闭数据库实例之前触发（非正常关闭不触发）
SERVERERROR	AFTER	数据库服务器发生错误之后触发
LOGON	AFTER	成功登录连接到数据库后触发
LOGOFF	BEFORE	开始断开数据库连接之前触发

- 注意：要在数据库之上建立触发器时，要求用户具有ADMINISTER DATABASE TRIGGER权限。
- 例：创建一个系统事件触发器TRI_XT，功能是当用户登录数据库时 将登录名、登录地址、登录时间等信息写入到数据表LOGON _EVENT中。其中表LOGON _EVENT是在创建触发器之前创建的基本表，用于保存登录数据。

- CREATE TABLE LOGON_EVENT
- (user_name VARCHAR2(10),
- adress VARCHAR(20),
- Logon_date timestamp,
- Logoff_date timestamp);
- CREATE OR REPLACE TRIGG
- AFTER LOGON ON DATABASE
- BEGIN
- INSERT INTO LOGON_EVENT(user_name,address,logon_date)
- VALUES(ora_login_user,ora_client_ip_address,systimestamp);
- END;
- /

当用户登录SQL*Plus后，触发器TRI_XT会自动被触发，将登录信息作为一行记录插入到表LOGON _EVENT表中。ora_login_user,ora_client_ip_address等都是常用的事件函数。

触发器禁止和启动

针对某个表创建的触发器，可以根据需要，禁止或启用其执行。其语法格式为：

ALTER TRIGGER 触发器名称 **ENABLE |DISABLE**

ENABLE：该选项为启用触发器

DISABLE：该选项为禁用触发器

以表为单位禁用/启动触发器

ALTER TABLE 表名称 **ENABLE |DISABLE ALL TRIGGERS**

菜单方式：第三方工具**PL/SQL Developer**中在对应触发器上点右键“禁止/允许”命令。

触发器删除

当不再需要某个触发器时，可以将其删除。可以使用下面的方法将触发器删除：

1、菜单方式：第三方工具PL/SQL Developer中在对应触发器上点右键“删除”命令。

2、使用SQL语句删除触发器

可以使用DROP TRIGGER语句，语法如下：

DROP TRIGGER 触发器名称

3、删除表同时删除触发器

当某个表被删除后，该表上的所有触发器将同时被删除，但是删除触发器不会对表中数据有影响。

第五章 数据库完整性

5.7 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

- **RDBMS完整性实现的机制**
  - 完整性约束定义机制
  - 完整性检查机制
  - 违背完整性约束条件时**RDBMS**应采取的动作