第7章 数据库的完整性

北京理工大学 计算机学院
张文耀
zhwenyao@bit.edu.cn

主要内容

7.1 数据库的完整性概述

7.2 实体完整性

7.3 参照完整性

7.4 用户定义的完整性

7.5 触发器

7.6 **SQL Server**中数据库完整性的实现

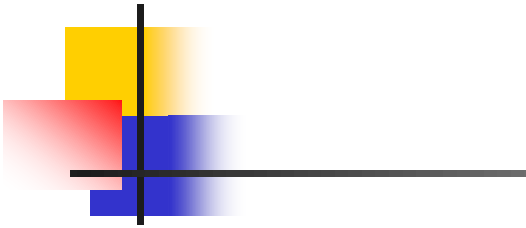
7.7 小结

7.1 数据库的完整性概述

- 数据库完整性的含义
  - 正确性
    指数据的合法性
  - 有效性
    指数据是否属于所定义的有效范围
  - 相容性
    表示同一事实的两个数据应相同
- 保持完整性的目的
  - 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（防止不合语义的、不正确的数据）
  - 保证数据库的数据质量

- 为维护数据的完整性，需要对数据库中的数据施加一些语义约束条件（完整性约束条件或完整性规则），对数据库进行完整性控制。
- 数据库的完整性控制是**DBMS**的基本功能之一，由完整性子系统负责，其功能包括：
  - 提供定义完整性约束条件的机制
  - 提供检查完整性的方法
  - 提供违约处理
    - 拒绝执行
    - 按照完整性控制策略处理

- 完整性约束条件
  - 一种语义概念；
  - 是对数据库中数据本身的某种语义限制、数据间的逻辑约束、数据变化时所遵循的规则等。
  - 关系数据库对数据的各种限制是以完整性约束条件的形式在关系数据库模式中指定的。
  - **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。
- 完整性检查
  - **DBMS**检查数据库中的数据是否满足完整性约束条件



- 关系数据库的完整性约束条件分为：
  - 实体完整性约束
  - 参照完整性约束
  - 其他（用户定义的）完整性约束
- 完整性约束条件的作用对象
  - 关系（关系约束）
  - 元组（元组约束）
  - 属性列（列级约束）

- 列级约束

  主要是对属性的数据类型、数据格式和取值范围、精度等的约束。具体包括：

  - 对数据类型的约束，包括数据类型、长度、精度等的约束。例如学生姓名的数据类型是字符型，长度是**8**。
  - 对数据格式的约束

    例如规定日期的格式为**YYYY/MM/DD**
  - 对取值域的约束

    例如学生成绩的取值范围必须是**0～100**。
  - 对空值的约束

- 元组约束

  元组中各个属性之间的约束关系

  例如：订货关系中发货日期不能小于订货日期，发货量不得超过订货量等。

- 关系约束

  一个关系的各个元组之间、或者多个关系之间存在的各种联系或约束。常见的关系约束有：

  - 实体完整性约束
  - 参照完整性约束
  - 函数依赖约束
  - 统计约束等

- 在关系数据库系统中，完整性控制策略包括默认值、规则、约束、触发器和存储过程等
  - 默认值（**Default**）
    - 如果在插入行中没有指定列的值，那么默认值指定列中所使用的值，例如：自动增长值，内置函数、数学表达式等
  - 约束（**Check**）（检查、校验）
    - 是自动强制数据完整性的方法。
    - 定义关系列中允许值的规则，是通用的强制完整性的标准机制。
    - 使用**Check**优于使用触发器、规则和默认值

- 规则（**Rule**）
  - 规则是大多数数据库系统中一个向后兼容的功能，用于执行一些与**Check**约束相同的功能。
  - 规则以单独的对象创建，然后绑定到列上。
- 触发器（**Trigger**）
  - 触发器是数据库系统中强制业务规则和数据完整性的主要机制
  - 用编程的方法实现复杂的业务规则和约束
- 存储过程（**Procedure**）
  与触发器类似
- 断言（**Assertion**）
  create assertion <name> check(predicate)

- 数据完整性的另一种分类
  - 声明式数据完整性（非编程的方式）
    - 作为对象定义的一部分来定义数据必须达到的标准
    - DBMS自动强制完整性
    - 通过使用约束、默认值和规则来实现
  - 过程式数据完整性（编程方式）
    - 在脚本中定义数据必须达到的标准
    - 在脚本中强制完整性
    - 通过使用触发器和存储过程来实现
    - 可在客户端或服务器用其他编程语言和工具来实现

7.2 实体完整性

- 实体完整性规则规定：
  - 主键的值唯一
  - 主键的值不能取空值
- 实现方法：通过对主键值的约束实现实体完整性
- 关系模型的实体完整性定义
  - 用**PRIMARY KEY**定义
    - 定义为列级约束条件
    - 定义为表级约束条件
  - 单属性构成的主键有两种定义方式
  - 对多个属性构成的主键只能定义为表级约束条件

【例】实体完整性约束定义

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

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

```
CREATE TABLE SC
       (Sno    CHAR(9)  NOT NULL ,
        Cno    CHAR(4)  NOT NULL ,
        Grade  SMALLINT ,
        PRIMARY KEY (Sno, Cno) );
/*只能在表级定义主键*/
```

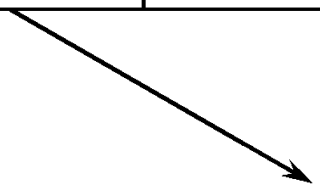
- **PRIMARY KEY** 约束可以作为表定义的一部分在创建表时定义，也可以在表创建之后再添加。
- 为了实施实体完整性，系统一般会在主键属性上自动创建唯一的索引来强制唯一性约束。

- 实体完整性检查和违约处理
  定义表的主键后，每当对该表插入一条记录或者对主键进行更新操作时，**DBMS**自动进行实体完整性的检查
  - 检查主键是否唯一，
    - 如果不唯一则拒绝进行插入或修改；
  - 检查主键的各个属性（字段）值是否为空
    - 如果有空的字段值，则拒绝操作，从而保证实体完整性。
  - 检查方法
    全表扫描法，索引扫描法

- 使用全表扫描法检查记录中主键值是否唯一

待插入记录

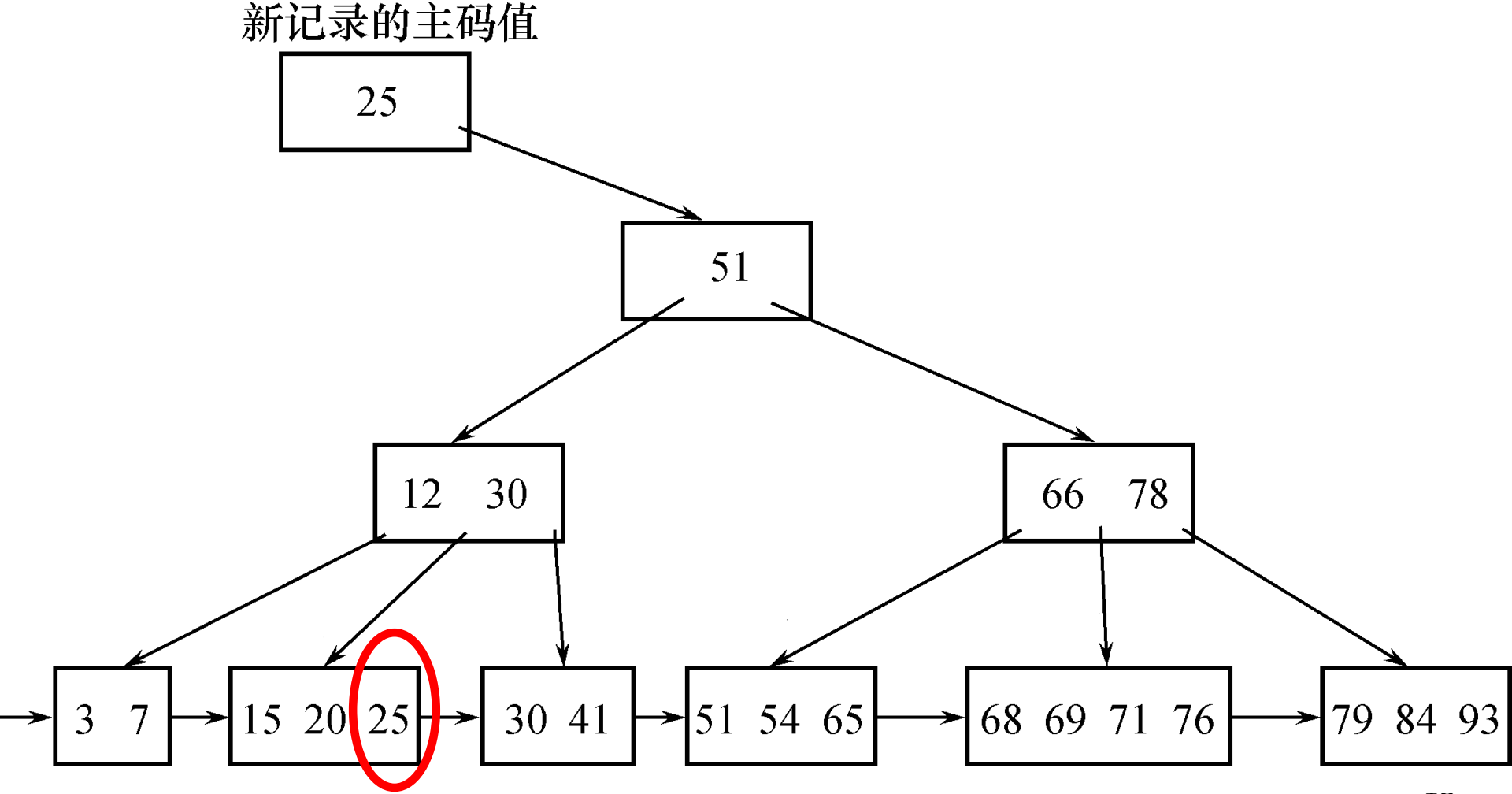
Keyi	F2i	F3i	F4i	F5i



基本表

Key1	F21	F31	F41	F51
Key2	F22	F32	F42	F52
Key3	F23	F33	F43	F53
⋮				

- 使用索引判断主键是否存在



7.3 参照完整性

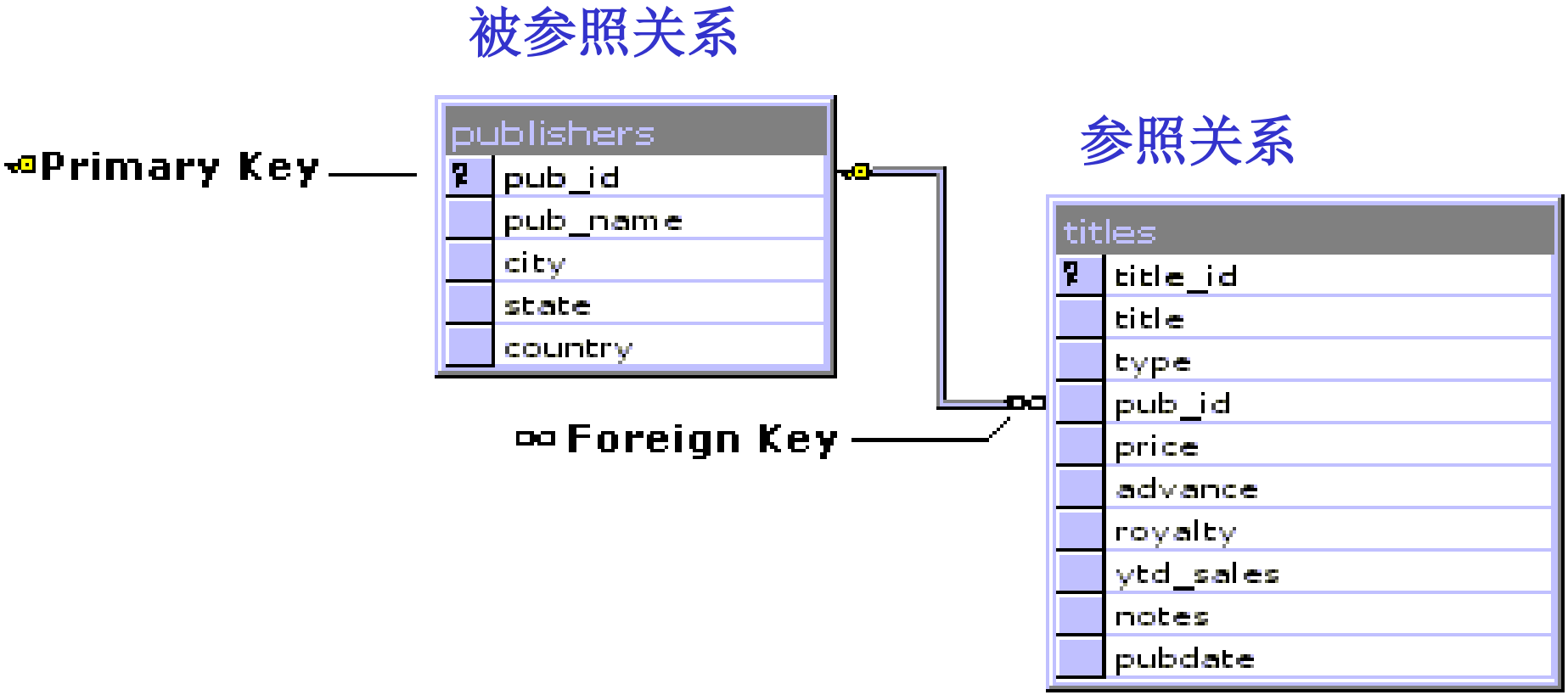
- 参照完整性
  - 刻画不同关系之间的联系
  - 约束同一关系内部不同属性之间的联系
- 参照完整性约束规则
  - 不允许引用不存在的元组
  - 在关系模型中，外键字段的值要么为空值，要么是被引用关系中元组的对应值
- 关系模型的参照完整性定义
  - 用**FOREIGN KEY**短语定义哪些列为外键,
  - 用**REFERENCES**短语指明外键参照哪些表的主键

【例 】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)
   );
```

参照完整性检查和违约处理

- 外键将两个表之间的元组联系起来，建立了参照与被参照的联系，需要维护参照完整性



- 破坏参照完整性的情况及其处理

被参照表publishers	参照表titles	违约处理
可能破坏参照完整性 ←	插入不存在外键值的元组	拒绝
可能破坏参照完整性 ←	修改外键值	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
修改主键值 →	可能破坏参照完整性	拒绝/级连修改/设置为空

- 维护参照完整性的策略

1. 参照关系中外键空值的问题
   - 需要定义外键是否允许为空值
   - 外键是其主键的组成部分，外键值不允许为空
   - 外键不是其主键的组成部分，可以根据具体的语义确定外键值是否允许空值
   - 允许空值的存在使得**SQL**中参照完整性约束的语义变得更加复杂

2. 在参照关系中插入元组的问题

- 受限插入
  - 向**titles**中插入新的元组，但该元组的**pub_id**属性值在表**publishers**中不存在，则系统拒绝
- 递归插入【级联**(CASCADE)**插入】
  - 首先向被参照关系插入相应的元组，其主键值等于参照关系插入元组的外键值，然后再向参照关系插入该元组

3. 在参照关系中修改元组的问题

- 先删除、再插入

4. 在被参照关系中删除元组的问题

- 级联删除（**CASCADES**）
  - 将参照关系中所有外键值与被参照关系中要删除元组之间值相同的元组一起删除。
  - 如果参照关系同时又是另一个关系的被参照关系，这种删除操作会继续级联下去。
- 受限删除（**RESTRICTED**）
  - 仅当参照关系中没有任何元组的外键值与被参照关系中要删除元组的主键值相同时，系统才执行删除操作，否则拒绝删除操作。
- 置空值删除
  - 删除被参照关系的元组，并将参照关系中相应元组的外键值置为空值。

5. 在被参照关系中修改主键值的问题

- 先删除、再插入

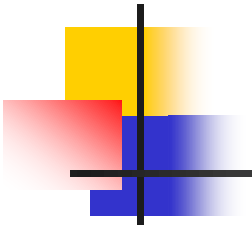
- 违反参照完整性，系统一般选用默认策略，即拒绝执行。如果想让系统采用其他策略则必须在创建表时加以说明。

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*/
      );
```

7.4 用户定义的完整性

- 用户定义的完整性是限定某一具体应用的数据必须满足的语义要求
  - 列值不能为空
  - 列值唯一
  - 一定的数据格式
  - ……
- **RDBMS**提供定义和检验用户定义的完整性的机制；不必由应用程序承担这部分功能。



- 用户定义的完整性的定义
  - 属性上的约束条件的定义
    - 唯一(UNIQUE)、
    - 非空(NOT NULL)、
    - CHECK约束、
    - 默认值DEFAULT等
  - 元组上的约束条件的定义
    - 用CHECK短语定义元组上的约束条件
    - 元组级的限制可以设置不同属性之间的相互约束条件

【例】创建选课表，并在相关属性上加入约束条件。

```
CREATE TABLE SC
   ( Sno    CHAR(9)  NOT NULL,
     Cno    CHAR(4)  NOT NULL ,
     Grade SMALLINT CHECK(Grade>=0 and
                  Grade<=100) ,
     PRIMARY KEY (Sno , Cno) ,
     FOREIGN KEY (Sno) REFERENCES Student(Sno) ,
     FOREIGN KEY (Cno) REFERENCES Course(Cno)
   );
```

【例】创建Students表，其中 age 值必须在 10 到 25 之间，score 值必须在 0 到 100 之间。

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT,
    score DECIMAL(5, 2),
    CHECK (age BETWEEN 10 AND 25
           AND score BETWEEN 0 AND 100) );
```

【例】创建**table1**，指定**c1**字段不能包含重复值，**c2**字段只能取特定值。

```
CREATE TABLE table1
  (   c1  CHAR(2)  UNIQUE ,
      c2  CHAR(4)  CHECK ( c2 IN
                    ('0000', '0001', '0002', '0003')),
      c3  INT  DEFAULT 1
  );
```

- 用户定义的完整性的检查和违约处理
  - 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，如果不满足则拒绝执行
  - **(1) INSERT INTO table1 (c1, c2)**
    VALUES ('10', '0000');
  - **(2) INSERT INTO table1**
    VALUES (‘10’, ‘0001’, 2);

    服务器：消息2627，**级别**14，**状态**2，**行**1
    违反了UNIQUE KEY**约束** ‘UQ_table1_4222D4EF’。**不能在** ‘table1’**中插入重复键。**
    语句已终止。

- 定义**CONSTRAINT** 约束的完整格式
  - **CONSTRAINT <**完整性约束条件名**>**
    [PRIMARY KEY短语 **|**
    FOREIGN KEY短语 **|**
    CHECK短语 **]**

```
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)
  );
```

- 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
  - 先删除原来的约束条件，再增加新的约束条件

【例】修改表**Student**中的约束条件，年龄由小于**30**改为小于**40**

```
ALTER TABLE Student
    DROP CONSTRAINT C3;
ALTER TABLE Student
    ADD CONSTRAINT C3
        CHECK (Sage < 40) ;
```

7.5 触发器

- 数据库系统一般提供两种主要机制来实现业务规则和数据完整性
  - 约束（**constraint**）(声明式、静态的)
    - 完整性约束机制在检测出违反约束条件的操作后，只能作简单的动作，例如：拒绝操作。
  - 触发器（**trigger**）(过程式、动态的)
    - 触发器是用户定义在关系数据表上的一类由事件驱动的特殊过程，用编程的方法实现复杂的业务规则。
    - 触发器比约束更加灵活，可以实现一般的数据完整性约束实现不了的复杂的完整性约束，具有更精细和更强大的数据控制能力。
    - 触发器常常用于强制业务规则和数据完整性。

- 触发器是一种特殊类型的存储过程，在对表或视图发出 **UPDATE**、**INSERT** 或 **DELETE** 语句时自动执行
- 可用触发器完成很多数据库完整性保护的功能
  - 实现复杂的业务规则
  - 实现比声明式约束更复杂的数据完整性
  - 比较数据修改前后的状态
  - 维护非规范化数据

定义触发器

- SQL使用CREATE TRIGGER命令创建触发器，其一般格式为：

```
CREATE TRIGGER <触发器名>
    { BEFORE | AFTER} <触发事件> ON <表名>
      REFERENCING
          NEW | OLD ROW | TABLE AS <变量>
      FOR EACH  {ROW | STATEMENT}
      [WHEN <触发条件>]
       <触发动作体>
```

触发器也叫事件－条件－动作（event-condition-action）规则

【示例】

```
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)
```

- 定义触发器的语法说明:
  1. 触发器的创建者
     - 表拥有者或创建表的用户才可以在表上创建触发器
     - 一个表上可以创建多个触发器
  2. 表名
     - 触发器的目标表
     - 当目标表被更新时，触发器才被激活
  3. 触发事件
     - 定义激活触发器的**SQL**语句的类别
     - **INSERT**、**DELETE**、**UPDATE**，或三者组合

4. 触发时间

指明触发器何时执行

- **BEFORE**
  - 在触发事件执行之前，判断触发条件是否满足。
  - 若满足条件则先执行触发动作部分的操作，然后再执行触发事件的操作。
- **AFTER**
  - 在触发事件完成之后，判断触发条件是否满足。
  - 若满足条件则执行触发动作部分的操作。
  - 如果触发事件因错误(如违反约束或语法错误)而失败，触发器将不会执行

5. 触发器类型

按照触发动作的间隔尺度，触发器可分为：

- 行级触发器(FOR EACH ROW)。
  对每一个修改的元组都会触发触发器的检查和执行
- 语句级触发器(FOR EACH STATEMENT)。
  只在SQL语句执行时候进行触发条件的检查和触发器的执行

【例】 Teacher表上创建一个AFTER UPDATE触发器。如果表Teacher有1000行，执行如下语句：

UPDATE Teacher SET Deptno=5;

语句级触发器，触发动作只发生1次;
行级触发器，触发动作将执行1000次。

6.触发条件

- 由**When**子句指定
- 只有当触发条件为真，触发动作体才执行
- 省略**WHEN**触发条件，触发动作体在触发器激活后立即执行

7.触发动作体

- 触发动作体是满足触发器条件后，执行的一系列数据库操作。
- 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。**【rollback】**

8. 引用

REFERENCING NEW | OLD ROW AS <变量>

- 行级触发器
- 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

REFERENCING NEW | OLD TABLE AS <变量>

- 语句级触发器
- 仅适用于**after**触发器

【例】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
```

【例】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;
```

【例】create trigger *credits_earned* ？？

after update of *takes* on (*grade*)
referencing new row as *nrow*
referencing old row as *orow*
for each row
when *nrow.grade* <> 'F' and *nrow.grade* is not null
and (*orow.grade* = 'F' or *orow.grade* is null)
begin
update *student*
set *tot_cred*= *tot_cred* +
(select *credits* from *course*
where *course.course_id*= *nrow.course_id*)
where *student.id* = *nrow.id*;
end;

使用触发器

- 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

  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（触发事件）；

  （3） 执行该表上的**AFTER**触发器。

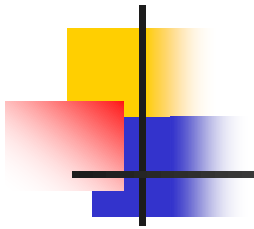
  注意：如果激活触发器的**SQL**语句违反了约束条件，则不会执行**AFTER**触发器。
- 存在级联触发问题

删除触发器

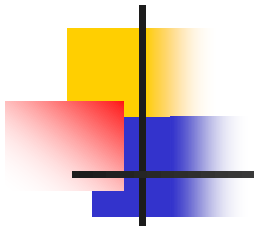
- **DROP TRIGGER <触发器名> ON <表名>**
- 触发器的删除不影响数据表和数据
- 删除表会自动删除其上的所有触发器
- 表的所有者拥有删除权限

7.6 SQL Server完整性的实现

- 实际的数据库产品对完整性的支持不尽相同
- **SQL Server**提供了对实体完整性、参照完整性和用户定义的完整性的支持
- 实体完整性的实现
  - 索引
  - **UNIQUE**约束
  - **PRIMARY KEY**约束
  - **IDENTITY**属性
- 参照完整性的实现
  - **FOREIGN KEY**约束
  - **CHECK**约束
  - 触发器



- 用户定义的完整性的实现
  - **CREATE TABLE**中的列级和表级约束
    - **CHECK**约束
    - **DEFAULT**约束
    - **NOT NULL**约束
    - 规则
  - 存储过程
  - 触发器



- **SQL Server用于维护数据完整性的对象**
  - 约束
  - 规则
  - 默认值
  - 触发器
- **默认值和规则对象**
  - 是可以绑定到一个或多个列或用户定义数据类型的对象
  - 可以一次定义，多次使用
  - 非标准内容
  - 可以用**DEFAULT**约束和**CHECK**约束代替

- 创建默认值和规则的示例
  - 创建一个大于**0**的规则，并绑定到产品表的单价字段上

    CREATE RULE R_jg AS @jg>0;

    sp_bindrule ‘R_jg’, ‘产品.单价’ ;
  - 创建一个默认值为**100081**对象，并将其绑定到**Student**表的**Zipcode**字段上

    CREATE DEFUALT DF_zip_code AS 100081;

    sp_binddefault 'DF_zip_code ',

    'Student.zipcode'

SQL Server的数据完整性对象—约束

- **Constraints**，定义关于列中允许值的规则
- 是强制完整性的标准机制
- 使用约束（**Constraints**）优于使用触发器、规则和默认值
- **SQL Server**提供多种强制列中数据完整性的机制
  - **PRIMARY KEY**约束
  - **FOREIGN KEY**约束
  - **UNIQUE**约束
  - **CHECK**约束
  - **DEFAULT**约束
  - **NOT NULL**约束

完整性类型	约束类型	描述
域	**DEFAULT**	如果在INSERT语句中未显式提供值，则指定为列提供的值
	NOT NULL	列值不允许取空值
	CHECK	指定列中可接受的数据值
实体完整性	**PRIMARY KEY**	惟一标识每一列，确保用户没有输入重复的值。同时创建一个索引以增强性能。不允许空值。
	UNIQUE	确保在非主键列中不输入重复值，并创建一个索引以增强性能。允许空值
参照完整性	**FOREIGN KEY**	定义一列或多列的值与同表或其他表中主键的值匹配
	CHECK	基于同表中其他列的值，指定列中可接受的数据值

CHECK 约束

- 可以将列的取值限定在指定范围内；
- 可以引用同一表中的其他列，约束同一个表中多个列之间的取值关系。
- 当执行**INSERT**语句或者**UPDATE**语句时，**CHECK**约束将验证数据值。
- **CHECK**约束中不能包含子查询。
- 示例

```
ALTER TABLE Employees
   ADD CONSTRAINT CK_birthdate
   CHECK (BirthDate > '01-01-1900' AND
                BirthDate < getdate());
```

PRIMARY KEY约束

- 表示唯一标识表中的行的列或列集
- 可以在创建表时创建，也可以在随后添加、修改或删除
- 必须非空且没有重复值
- **SQL Server自动创建唯一的索引来强制PRIMARY KEY约束所要求的唯一性**
- 示例

```
ALTER TABLE Customers
    ADD CONSTRAINT PK_Customers
    PRIMARY KEY (CustomerID);
```

【例 】使用SQL语句建立主键约束

```
CREATE TABLE publishers
(
  pub_id     char(4) NOT NULL
      PRIMARY KEY
      CHECK (pub_id IN ('1389', '0736', '0877',
                        '1622', '1756')    OR
                  pub_id LIKE '99[0-9][0-9]'),
  pub_name    varchar(40) ,
  city            varchar(20),
  state           char(2) NULL,
  country       varchar(30)
);
```

FOREIGN KEY 约束

- 外键所引用的列必须是有**PRIMARY KEY**约束或**UNIQUE**约束的列
- 列的取值范围只能是被引用的列的取值或空值
- **FOREIGN KEY**约束可以引用其他表
- **FOREIGN KEY**可以是单列键或者是多列键
- 示例

```
ALTER TABLE  Orders
   ADD CONSTRAINT FK_Orders_Customers
   FOREIGN KEY (CustomerID)
   REFERENCES   Customers(CustomerID);
```

- 当使用外键约束时，应该考虑以下几个因素：
  - 主键和外键的数据类型必须严格匹配；
  - 外键约束不能自动创建索引，需要用户手动创建；
  - 用户想要修改外键约束的数据，必须有对外键约束所参考表的**SELECT**权限或者**REFERENCES**权限；
  - 一个表可以有多个 **FOREIGN KEY** 约束，最多可以有**31**个外键约束；
  - 在临时表中，不能使用外键约束。

DEFAULT约束

- 用于提供列的默认值
- 默认值可以是任何取值为常数的表达式，如：常量、NULL、系统函数、数学表达式等
- 默认值必须符合此列上的任何CHECK约束
- 在使用INSERT语句时，如果没有提供值，则DEFAULT约束会指定列中使用的值
- 只有向表中插入数据时系统才检查DEFAULT约束

```
ALTER TABLE Customers
    ADD CONSTRAINT DF_contactname
            DEFAULT  'UNKNOWN'
    FOR ContactName;
```

UNIQUE约束

- 用于限制一个列中不能有非空的重复值
- **SQL Server自动创建UNIQUE索引来强制UNIQUE约束的唯一性要求**
- 不能对现有的包含重复值的列添加**UNIQUE**约束
- 注意**UNIQUE**约束与**PRIMARY KEY**约束的区别

```
ALTER TABLE Suppliers
    ADD CONSTRAINT U_CompanyName
    UNIQUE (CompanyName);
```

- 当使用唯一性约束时，需要考虑以下几个因素：
  - 使用唯一性约束的字段允许为空值。
  - 一个表中可以允许有多个唯一性约束。
  - 可以把唯一性约束定义在多个字段上。
  - 唯一性约束用于强制在指定字段上创建一个唯一性索引。
  - 缺省情况下，创建的索引类型为非聚集索引

NOT NULL约束

- 指定一列不允许空值
- **NULL**的存在表明值未知或未定义

```
CREATE TABLE SC
 ( Sno    CHAR(9)  NOT NULL,
   Cno    CHAR(4)  NOT NULL ,
   Grade SMALLINT CHECK(Grade>=0 and
                Grade<=100) ,
   PRIMARY KEY (Sno , Cno) ,
   FOREIGN KEY (Sno) REFERENCES Student(Sno) ,
   FOREIGN KEY (Cno) REFERENCES Course(Cno)
 );
```

使用约束的注意事项

- **CREATE TABLE**：在创建表时创建约束
- **ALTER TABLE**：在一个已有的表上创建约束
- 可添加单列或多列约束
  - 若约束应用于单列，称为列级约束;
  - 若约束引用了多列，称为表级约束。
- 可直接在表上创建、更改和删除约束，而不必删除并重建表
- 当给一个表添加约束的时候，**SQL Server** 将检查现有数据是否违反约束
- 建议创建约束的时候指定名称，否则系统将为约束自动产生一个复杂的名称

SQL Server的触发器

- 用途
  - 定义用户定制的错误信息
    - 通过使用触发器，可以在特定条件出现时调用预定义或动态定义的定制错误信息
    - 约束、规则和默认只能通过标准系统错误信息来表达错误。若需要定制信息或更复杂的错误处理，需要使用触发器
  - 实现用户定义的完整性
  - 在数据库中的相关表上实现级联更改
  - 实现比声明式约束更复杂的业务规则
  - 维护非标准数据，特别是处理较为复杂的逻辑

- **SQL Server的触发器定义**

```
CREATE TRIGGER <触发器名>
        ON {<表名>|<视图名>}
        {FOR|AFTER|INSTEAD OF} <触发事件>
    AS
        <触发动作体>
```

【例1】创建限制更新数据的触发器，限制将SC表中不及格学生的成绩改为及格。

【例2】创建删除触发器，当删除一本书时，先检查这本书是否已经被卖过了，即是否和订单关联，如果已经有关联则该书的信息不能被删除，删除动作需要回滚。

【例1】

```
CREATE TRIGGER tri_grade
    ON SC FOR UPDATE
AS
   IF UPDATE (Grade)
      IF EXISTS (SELECT * FROM INSERTED JOIN
          DELETED ON
                  INSERTED.Sno = DELETED.Sno
          WHERE INSERTED.Grade >= 60 AND
                  DELETED.Grade < 60)
      BEGIN
         RAISERROR ('不许将不及格的成绩改为及格！')
         ROLLBACK
      END
```

- **INSERTED表和DELETED表**

  SQL Server触发器专用的临时虚拟表，由SQL Server自动管理的
  - **INSERTED表**
    - 保存**insert**操作中新插入的数据或**update**操作中更新后的数据，即：插入到触发表的新数据行的副本
  - **DELETED表**
    - 存放**delete** 操作删除的数据或**update**操作更新前的数据，即：从触发表中删除的旧数据行的副本
  - 触发操作完成后，与触发器相关的临时表自动删除

- 对带触发器的表的操作
  - **INSERT**操作时，新插入的数据被记录在**INSERTED**表中
  - **DELETE**操作时，删除的数据被记录在**DELETED**表中。
  - **UPDATE**操作时，相当于先执行删除操作，再执行插入操作，即：先在触发器表中删除更新前的行，并将这些行复制倒**DELETED**表中，然后将更新后的新行复制到触发器表和**INSERTED**表中。
  - 触发器中对**INSERTED**表和**DELETED**表的使用同普通表一样
  - 可以通过**INSERTED**表和**DELETED**表所记录的数据，判断对数据的修改是否正确。

【例2】

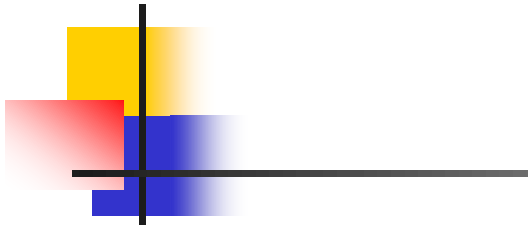
```
CREATE TRIGGER Products_Delete
        ON Products FOR DELETE
AS
    IF (SELECT COUNT(*)
         FROM [Order Details] INNER JOIN deleted
                ON [Order Details].ProductID
                    =deleted.ProductID)>0
    BEGIN
        RAISERROR ('该产品有定购历史,
                            事务无法进行！', 10, 1)
        ROLLBACK TRANSACTION
    END
```

Oracle触发器示例

```
create trigger teacher_sal -- 触发器名字
before update of salary on teacher -- 作用表
referencing new x, old y -- 定义更新前后的值
for each row
when(x.salary<y.salary) -- 对每一条记录都要检查,
 begin
  --如果违反则执行，提示无效更新
  raise_application_error(-20003,
    'invalid salary on update');
 end;
```

PostgreSQL触发器示例

```
-- 创建一个触发器函数
CREATE OR REPLACE FUNCTION check_salary()
RETURNS TRIGGER AS $$
BEGIN
    -- 检查工资是否为正数
    IF NEW.salary <= 0 THEN
        RAISE EXCEPTION 'Salary must be a positive number';
    END IF;
    -- 返回 NEW 表示继续执行插入操作
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```



```
-- 创建 BEFORE INSERT 触发器
CREATE TRIGGER before_insert_employees
BEFORE INSERT ON employees
FOR EACH ROW
EXECUTE FUNCTION check_salary();
```

触发器与CHECK 约束

- **CHECK** 约束只能根据逻辑表达式或同一表中的另一列来验证列值。
  - 如果应用程序要求根据另一个表中的列验证列值，则必须使用触发器（看实际系统的情况）。
- 约束只能通过标准的系统错误提示传递错误信息。
  - 如果应用程序要求使用（或能从中获益）自定义信息和较为复杂的错误处理，则必须使用触发器。
- 触发器可以引用其它表中的列。
- 触发器可以支持约束的所有功能；但并不总是最好的方法，因为触发器的系统开销更大。

使用何种完整性强制方法？

- 应综合考虑功能和性能开销，来决定使用何种强制数据完整性的方法
  - 对于基本的完整性逻辑，例如有效值和维护表间的联系，最好使用声明式完整性约束；
  - 如果要维护复杂的、大量的、非主键或外键关系的部分数据，使用触发器或存储过程；
  - 约束比较简单、开销低，适合于完整性逻辑比较简单的场合；
  - 触发器比较复杂、开销大，适合于完整性逻辑比较复杂的场合。

本章小结

- 数据库的完整性是为了保证数据库中存储的数据的正确性、有效性和相容性。
- **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
- 实体完整性
- 参照完整性
- 用户定义的完整性
- 实现数据库完整性的一个重要方法——触发器

作业和思考题

作业：

P158(e164)习题10, 12, 13

[参照SQL-Server示例写]

思考题：

P158 (e163)习题

1，2，3，4，5，6，7，8，9，11