第一篇 基础篇

第五章 数据库完整性

福州大学数计学院

程 烨

chengye@fzu.edu.cn

2013年4月15日

数据库完整性

数据库的完整性是指数据库中的数据必须始终满足数据库语义约束。保证数据库中数据的正确性、有效性和相容性，防止错误的数据进入数据库。

通过对数据库中的数据进行语义定义来实现。

数据库安全性与完整性

数据库安全性与完整性是两个不同的概念：

数据库安全性的防范对象是非法用户与非法操作；

数据库完整性的防范对象是不合语义的数据。

数据库安全性与完整性

两者相似之处：

DBMS提供定义用户对数据库的操作权限的机制与定义完整性约束的机制；

将定义存储在**DBMS**的数据字典中；

DBMS完成对数据库安全性，完整性的验证功能。

数据库完整性(续)

数据库完整性机制的任务是确保数据库中存储的数据符合现实世界语义。

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 数据库完整性

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

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

第五章 数据库完整性

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*/
);
```

5.2.2参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

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
    - 需要进行一系列相关操作,保证数据的一致性

  - **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)

- 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)
    );
```

5.3.2 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

5.3.3 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

5.3.4 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

5.4 完整性约束命名子句

- CONSTRAINT—对完整性 约束条件命名

  CONSTRAINT <完整性约束条件名>

  [PRIMARY KEY短语

  |FOREIGN KEY短语

  |CHECK短语]

- 作用—可灵活地增加、删除完整性 约束条件

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)

- 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

ALTER TABLE 表名
  DROP CONSTRAINT 已定义的约束;

ALTER TABLE 表名
  ADD CONSTRAINT定义一个新的约束条件;

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')

域中的完整性限制(续)

［例］建立一个颜色域**COLOR**，并对其中的限制命名

```
CREATE DOMAIN COLOR CHAR(6) DEFAULT  '???'
CONSTRAINT VALID_COLORS  CHECK ( VALUE IN
('Red'，  'Yellow'，  'Blue'，  'Green'，  '???') );
```

断言

- 如果完整性约束牵涉面较广，与多个关系有关，或者与聚合操作有关，可以使用“断言”（**Assertions**）机制。
- 句法：
  - **CREATE ASSERTION <断言名> CHECK (条件);**
  - **DROP ASSERTION <断言名>**

［例］每位教师开设的课程不能超过**10**门

```
CREATE ASSERTION ASSE1 CHECK
        ( 10 >= ALL(   SELECT COUNT(C#)
                       FROM C
                       GROUP BY TEACHER) );
```

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

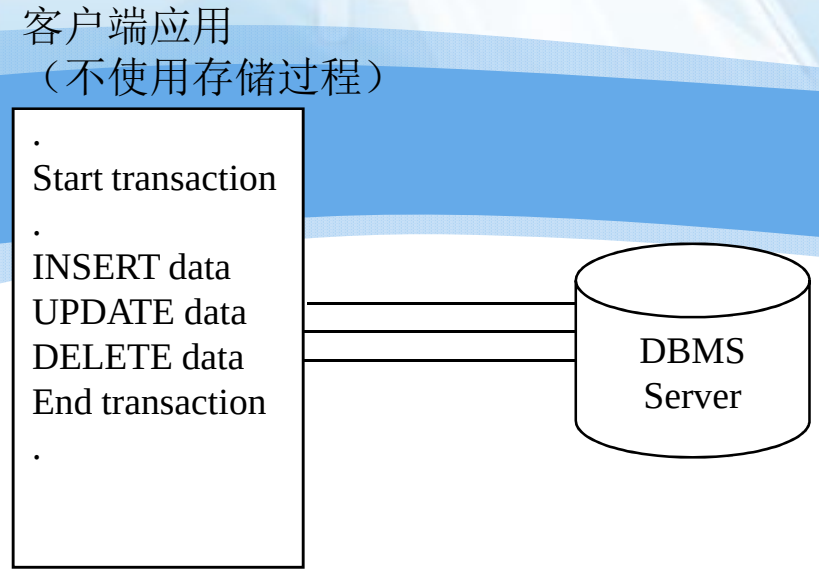
存储过程的概念

客户/服务器数据库与传统的数据库结构的一个很重要的区别是，在传统的数据库中只存放数据，所有的应用程序都在用户端，都与用户实际运行的应用程序捆绑在一起；而在客户/服务器结构的数据库中，在数据库中还可以存放程序，即存储过程。

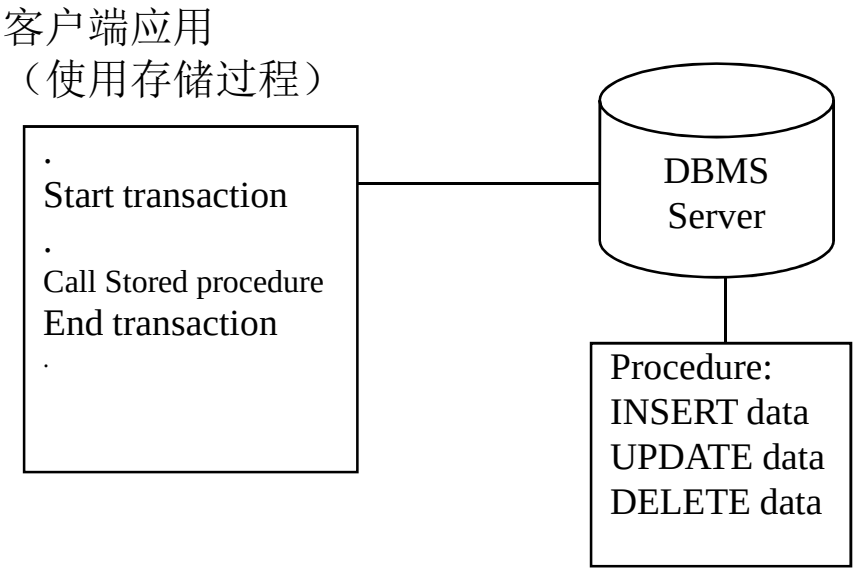
存储过程的概念(续)

存储过程是事先编好的、存储在数据库中的程序，这些程序用来完成对数据库的指定操作。

利用存储过程，可以避免在网络上传输大量无用的信息或原始数据，只需要传输调用存储过程的指令和数据库服务器返回的处理结果。且可以反复调用，从而减轻程序编写的工作量。



(a)



(b)

不使用存储过程时，所有的数据处理都在客户端完成；而使用存储过程时，可以使数据处理在服务器端完成。

存储过程的概念(续)

系统存储过程

系统本身提供了一些存储过程，用于管理数据库服务器和显示有关数据库和用户的信息，我们称之为系统存储过程

用户存储过程

用户也可以编写自己的存储过程，并把它存放在数据库中。这样安排的主要目的就是要充分发挥数据库服务器的功能，尽量减少网络上的堵塞。

存储过程的概念(续)

- 此外，还可以利用存储过程间接的实现一些安全控制功能。

- 例如：不允许直接接触某些数据对象，可以授权用户执行某个查询功能的存储过程来完成信息查询，达到隔离用户的目的。

存储过程的概念(续)

存储过程是客户/服务器机制的一个重要组成部分，如果使用客户/服务器机制的数据库管理系统，但是不理解存储过程或没有充分利用存储过程，那将使客户/服务器机制的功能大打折扣，使系统的整体性能可能降低很多。

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，它在满足某个特定条件时自动触发执行。
  - 由服务器自动激活
  - 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

触发器的用途

1. 触发器可以通过级联的方式对相关的表进行修改。比如，对父表的修改，可以引起对子孙表的一系列修改，从而保证数据的一致性和完整性。

2. 触发器可以禁止或撤消违反参照完整性的修改。

3. 触发器可以强制比用CHECK约束定义更加复杂的限制。

触发器是依附于表的数据库对象

- 一个触发器和三部分内容有关：
  - 定义触发器的表
  - 激活触发器的数据操作语句
  - 触发器要采取的动作

5.6 触发器

- 5.6.1 定义触发器
- 5.6.2 激活触发器
- 5.6.3 删除触发器

5.6.1 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>

FOR EACH {ROW | STATEMENT}

［WHEN <触发条件>］

<触发动作体>

定义触发器(续)

- 定义触发器的语法说明:
  - 1. 创建者：表的拥有者
  - 2. 触发器名
  - 3. 表名：触发器的目标表
  - 4. 触发事件：INSERT、DELETE、UPDATE
  - 5. 触发器类型
    - 行级触发器（FOR EACH ROW）
    - 语句级触发器（FOR EACH STATEMENT）

定义触发器(续)

- 例如,假设在［例11］的TEACHER表上创建了一个AFTER UPDATE触发器。如果表TEACHER有1000行，执行如下语句：

  UPDATE TEACHER SET Deptno=5;

  - 如果该触发器为语句级触发器，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

定义触发器(续)

- **6.** 触发条件
  - 触发条件为真
  - 省略**WHEN**触发条件

- **7.** 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

定义触发器(续)

［例18］ 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    /*行级触发器*/
   AS BEGIN          /*定义触发动作体，是PL/SQL过程块*/
       IF (new.Job='教授') AND (new.Sal < 4000) THEN
       new.Sal :=4000;
       END IF;
   END;
```

定义触发器(续)

［例19］定义AFTER行级触发器，当教师表Teacher的工资发生变化后就自动在工资变化表Sal_log中增加一条相应记录

首先建立工资变化表Sal_log

```
CREATE TABLE Sal_log
   (Eno    NUMERIC(4)  references teacher(eno)，
    Sal     NUMERIC(7，2)，
    Username  char(10)，
    Date   TIMESTAMP
   )；
```

定义触发器(续)

[例19]（续）

```
CREATE TRIGGER Insert_Sal
    AFTER INSERT ON Teacher      /*触发事件是INSERT*/
    FOR EACH ROW
    AS BEGIN
        INSERT INTO Sal_log VALUES(
            new.Eno，new.Sal，CURRENT_USER，
    CURRENT_TIMESTAMP);
    END;
```

定义触发器(续)

[例19]（续）

```
CREATE TRIGGER Update_Sal
  AFTER UPDATE ON Teacher          /*触发事件是UPDATE */
  FOR EACH ROW
  AS BEGIN
   IF (new.Sal <> old.Sal) THEN INSERT INTO Sal_log VALUES(
       new.Eno，new.Sal，CURRENT_USER，
   CURRENT_TIMESTAMP);
   END IF;
END;
```

5.6 触发器

- 5.6.1 定义触发器
- 5.6.2 激活触发器
- 5.6.3 删除触发器

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
  - 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；
    - （3） 执行该表上的**AFTER**触发器。

激活触发器(续)

［例20］执行修改某个教师工资的SQL语句，激活上述定义的触发器。

UPDATE Teacher SET Sal=800 WHERE Ename='陈平';

执行顺序是：

- 执行触发器**Insert_Or_Update_Sal**
- 执行**SQL**语句“**UPDATE Teacher SET Sal=800 WHERE Ename='陈平';**”
- 执行触发器**Insert_Sal**；
- 执行触发器**Update_Sal**

5.6 触发器

- 5.6.1 定义触发器
- 5.6.2 激活触发器
- 5.6.3 删除触发器

5.6.3 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。

［例21］ 删除教师表Teacher上的触发器Insert_Sal

DROP TRIGGER Insert_Sal ON Teacher;

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