
# 第五章 数据库完整性

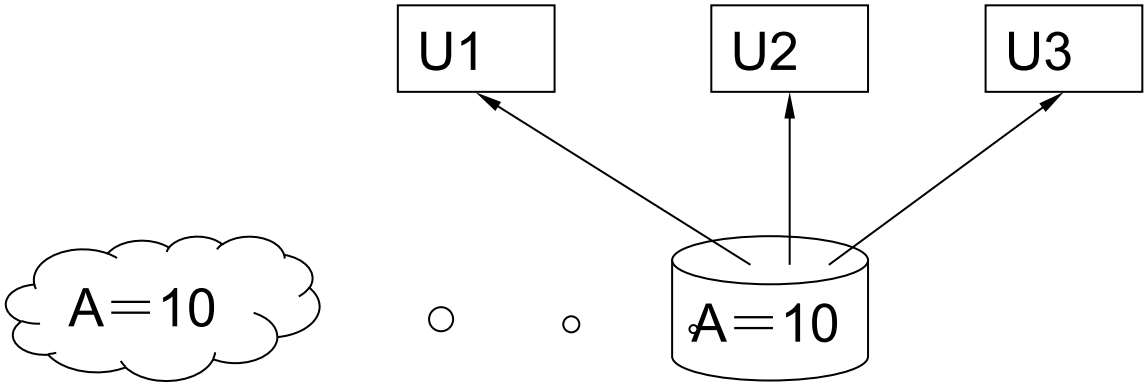
*Principles of Database Systems*

# 数据库完整性

- 数据库的完整性：

  数据的正确性和相容性。

  - **正确性**指数据库中数据与现实世界的实际情况是相符的。
  - **相容性**指数据库中数据自身不存在自相矛盾的现象。




若现实世界的A与DB中的A保持一致，且U1、U2、U3等所有用户从DB中查询的结果均为10，则称A具有完整性，否则就是不完整的。

# 数据库完整性



- 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



## DBMS的完整性控制功能:

1. 为用户提供定义完整性约束条件的机制

   DBMS将这些约束条件作为模式的一部分存入数据库中。

2. 监督系统中执行的操作是否违反完整性限制条件

   应进行完整性检查的操作有INSERT/DELETE/UPDATE。

3. 对违反完整性约束的情况进行相应处理

   拒绝执行 / 级联操作

**思考：完整性控制由DBMS实现和由应用程序实现有什么区别？哪种方式更好？**

# 第五章　数据库完整性

# 5.1 实体完整性



实体是客观存在且**相互区别**的事物。

1. 如何定义实体完整性？

实体完整性在关系模型中通过关系的主码（PRIMARY KEY）来体现。

PRIMARY KEY在CREATE TABLE语句中定义。

- 方法一：在列定义子句中说明（列级约束） 。

```
CREATE TABLE COURSE (
        cno CHAR(8) PRIMARY KEY,
        cname CHAR(30),
        credit SMALLINT
);
```

# 5.1 实体完整性



- 方法二：用单独的表约束子句说明。（表级约束）

```
CREATE  TABLE   COURSE (
        cno  CHAR(8),
        cname  CHAR(30),
        credit   SMALLINT,
        PRIMARY KEY (cno)
);
```

如果主码由多个列组成，则只能用第二种方法定义主码。

```
CREATE  TABLE   SC (
        sno  CHAR(8),
        cno  CHAR(8),
        grade  SMALLINT,
        PRIMARY KEY(sno, cno)
);
```

# 5.1 实体完整性



2. 如何检查实体完整性

对基表插入数据或对主码列进行更新时，DBMS将自动对该操作进行实体完整性检查。包括：

- 主码值是否唯一，否则拒绝执行该操作；
- 各主码列的值是否为空，是则拒绝执行该操作。

检查记录中主码值是否唯一的一种方法是进行全表扫描。

索引是整表的“目录”，有提高数据访问速度的作用：

- 索引比整表小得多，所以查索引比在整表中查询要快；
- 随机查询时利用索引可以快速定位整表中的元组；
- 有些查询如果只涉及索引项，则可避免读整表。

# 5.2 参照完整性



引用数据与被引用数据应该是一致的。

- 如何定义参照完整性

  参照完整性在关系模型中通过关系的外码（FOREIGN KEY）来体现。

  FOREIGN KEY同样在CREATE TABLE语句中定义。

- 方法一：在列定义子句中说明（列级约束） 。

```
CREATE TABLE SC (
        sno CHAR(8) REFERENCES STUDENT(sno),
        cno CHAR(8) REFERENCES COURSE(cno),
        grade SMALLINT,
        PRIMARY KEY(sno, cno)
);
```

# 5.2 参照完整性



- 方法二：用单独的表约束子句说明。（表级约束）

```
CREATE TABLE SC (
    sno CHAR(8),
    cno CHAR(8),
    grade SMALLINT,
    PRIMARY KEY(sno, cno),
    FOREIGN KEY(sno) REFERENCES STUDENT(sno),
    FOREIGN KEY(cno) REFERENCES COURSE(cno)
);
```

在参照完整性约束中，参照关系（即外码所在的关系，如SC），称为**子表**；被参照关系（如STUDENT和COURSE）称为**父表**。

# 5.2 参照完整性



2. 如何检查参照完整性

**外码的取值限制：**

- NULL（思考：如果该外码同时又是子表的主码呢？）
- 引用父表中的某值

- 可能破坏参照完整性的情况及违约处理：

| 被参照表（如Stu） | 参照表（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 |

# 5.2 参照完整性



## 1）在子表中插入元组

仅当父表中存在与插入元组相应的元组时，系统执行在子表中插入元组的操作，否则拒绝此插入操作。

例：假设STUDENTS表的clno引用了CLASS表的clno。

在STUDENTS表中插入一条clno＝'01'的学生记录，如果CLASS表中存在clno＝'01'的记录，则执行；如果不存在，则拒绝执行。

## 2）修改子表外码

如果父表中存在待修改值，则执行；否则不允许执行此类修改操作。

例：Sno＝'991001'的学生的clno＝'05'，将其clno改为'01'，已知CLASS表中存在clno＝'01'但不存在clno＝'AA'的记录

则执行；如果改为 'AA'，则拒绝执行。

# 5.2 参照完整性



3）修改父表主码（若子表中有相关参照记录）

- **拒绝修改：**拒绝执行此类操作。
- **级联修改：**将子表中相关记录在外码上的值一起自动修改。

例：要将CLASS表中的Clno＝'01'改为'101'，
则由DBMS自动将STUDENTS表中的所有Clno＝'01'的记录的Clno都修改为'101'。

- **置空修改：**将子表中相关记录在外码上的值全部置为NULL。

例：要将CLASS表中的Clno＝'01'改为'101'，
则由DBMS自动将STUDENTS表中的所有Clno＝'01'的记录的Clno属性置NULL。

# 5.2 参照完整性



4) 删除父表元组（若子表中有相关参照记录）

- **拒绝删除**: 发出警告，拒绝执行此类操作。
- **级联删除**: 删除父表中元组的同时，自动删除子表中的相关元组。

例：删除CLASS表clno＝'01'的元组时，
DBMS自动将STUDENTS表中所有clno＝'01'的元组一起删除。

- **置空删除**: 删除父表中元组的同时，自动将子表中的相关元组的外码置NULL值。

例：删除CLASS表clno＝'01'的元组时，
DBMS自动将STUDENTS表中所有clno＝'01'的元组的clno改为NULL。

# 5.2 参照完整性



如何指定修改或删除父表时的违约处理动作？

在定义参照完整性约束时，可同时指定删除或修改时的违约处理动作，如NOT ACTION、SET NULL或CASCADE。缺省动作为NOT ACTION。

形如：

```
CREATE  TABLE   SC (
        ……
        FOREIGN  KEY(sno) REFERENCES STUDENT(sno) ON
        DELETE SET NULL
        ON UPDATE CASCADE,
        ……
);
```

# 违约处理



［例］ 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*/
);
```

# 5.3 用户定义的完整性约束



在一个具体应用中，数据需满足一些特定的语义要求。

1）空值约束（列级）

指定属性的值不允许为空值（NOT NULL）

```
CREATE  TABLE   COURSE (
        cno  CHAR(8),
        cname  CHAR(30) NOT NULL,
        credit   SMALLINT,
        PRIMARY KEY (cno)
 ) ;
```

# 5.3 用户定义的完整性约束



## 2）唯一性约束（列级/表级）

指定属性或属性组的值必须唯一（UNIQUE）

方法一：
```
CREATE  TABLE   COURSE (
        cno  CHAR(8),
        cname  CHAR(30) NOT NULL UNIQUE ,
        credit   SMALLINT,
        PRIMARY KEY (cno)
  );   /* 列级约束 */
```

方法二：
```
CREATE  TABLE   COURSE (
        cno  CHAR(8),
        cname  CHAR(30) NOT NULL,
        credit   SMALLINT,
        PRIMARY KEY (cno),
        UNIQUE(cname)
  );   /* 表级约束 */
```

# 5.3 用户定义的完整性约束



## 3）CHECK约束（列级/表级）

检查一个属性或多个属性的取值是否满足一个布尔表达式。

方法一：
```
CREATE TABLE SC (
    sno CHAR(8),
    cno CHAR(8),
    grade SMALLINT CHECK(grade <=100),
    ……
);      /* 列级约束 */
```

方法二：
```
CREATE TABLE EMPLOYEE(
    ……
    age SMALLINT,
    work_year SMALLINT,
    CHECK(age>work_year),
    ……
); /* 表级约束 */
```

# 5.4 完整性约束命名子句



- 完整性约束与列一样是关系模式的构成元素。
- 关系中的列是必须命名的，但是约束却不一定要命名。

思考：为什么列必须命名而约束可以不命名？约束不命名可能带来什么问题？

- 如何给完整性约束命名

约束命名子句

CONSTRAINT <约束名> <约束说明>

其中，<约束说明> 可以是PRIMARY KEY子句，FOREING KEY 子句，NOT NULL子句，UNIQUE子句和CHECK子句。

# 5.4 完整性约束命名子句



在CREATE TABLE语句中使用约束命名子句为约束命名：

```
CREATE  TABLE   SC (
  sno  CHAR(8),
  cno  CHAR(8),
  grade  SMALLINT
  CONSTRAINT C1 CHECK(grade<=100),
  CONSTRANT PK PRIMARY KEY(sno, cno),
  CONSTRANT FK1 FOREIGN KEY(sno)
          REFERENCES STUDENT(sno),
  CONSTRANT FK2 FOREIGN KEY(cno)
          REFERENCES COURSE(cno)
 );
```

# 5.4 完整性约束命名子句



- 在ALTER TABLE语句中通过约束名删除约束：

```
ALTER TABLE SC DROP CONSTRANT C1;
```

- 在ALTER TABLE语句中使用约束命名子句增加新的约束：

```
ALTER  TABLE   S
ADD CONSTRAINT C2
        CHECK(age>=15 AND age<=30) ;
```

# 例：SQLServer命名约束检测



```
CREATE TABLE Parent
    (pkey1 INT NOT NULL
    CONSTRAINT pk_Parent PRIMARY KEY (pkey1))
    GO
CREATE TABLE ConstraintName
    (Pkey INT NOT NULL
    CONSTRAINT pk_CnstNm primary key,
    Parent_pkey1 INT NOT NULL,
    col1 INT NULL
    CONSTRAINT ck_CnstNm_col1 CHECK (col1 IN ( 'a','b' ) )
    CONSTRAINT df_CnstNm_col1 DEFAULT 1,
    CONSTRAINT fk_Parent_CnstNm FOREIGN KEY (Parent_pkey1)
    REFERENCES Parent (pkey1)
    )
    GO
exec sp_helpconstraint ConstraintName   //存储过程查看表ConstraintName上的约束
    GO
DROP TABLE ConstraintName
    GO
```

请上机时检查，带命名约束 和非命名约束的差异

# SQLServer命名约束检测（续）



```
CREATE TABLE ConstraintName
     (Pkey INT NOT NULL primary key,
     Parent_pkey1 INT NOT NULL
     FOREIGN KEY (Parent_pkey1) REFERENCES PARENT(pkey1),
     col1 INT NULL
     CHECK (col1 IN ( 'a','b' ) )
     DEFAULT 1
     )
     GO
exec sp_helpconstraint ConstraintName
     GO
DROP TABLE ConstraintName
     GO
CREATE TABLE ConstraintName
     (Pkey INT NOT NULL  primary key,
    Parent_pkey1 INT NOT NULL
     FOREIGN KEY (Parent_pkey1) REFERENCES PARENT(pkey1),
     col1 INT NULL
     CHECK (col1 IN ( 'a','b' ) )
     DEFAULT 1
     )
     GO
```

```
exec sp_helpconstraint ConstraintName
     GO
DROP TABLE ConstraintName
     GO
DROP TABLE Parent
     GO
```

# 5.7 触发器



## 什么是触发器？

触发器是定义在基表上的一种数据库对象，它指定：在执行对表的某些操作的时候，另一些操作也同时被执行。

## 定义一个触发器应包含哪些要素？

- 触发器的名字
- 触发器关联的表名
- 哪些操作执行时将引发其它操作被执行（触发事件）
- 被触发后执行的操作（触发动作）
- 执行触发动作的时机

# 5.7.1 定义触发器



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>;

说明：

- <触发事件>可以是INSERT、DELETE或UPDATE，或这些事件的组合，UPDATE后面可以带OF <触发列, ...>，指明只当哪些列被修改时才激活触发器。BEFORE或AFTER表示触发器是在触发事件之前还是之后被激活。
- FOR EACH ROW表示该触发器为行级触发器，触发事件每影响一条元组都将激活一次触发器。
- FOR EACH STATEMENT表示该触发器为语句级触发器，触发事件只激活一次触发器。

# 5.7.1 定义触发器



DBMS如何执行触发器：

- 触发器被激活后，先检查<触发条件>，当其为真时，<触发动作体>才执行；如果没有指定<触发条件>，则<触发动作体>在触发器被激活后立即执行。
- <触发动作体>是一段程序，如果触发器是行级触发器，则这段程序中还可以使用NEW和OLD分别引用触发事件发生前后的元组值。

例：保留表BOOKS中被删除的书的记录。

```
CREATE TRIGGER BOOKS_DELETE
AFTER DELETE ON BOOKS
FOR EACH ROW
INSERT INTO BOOKS_DLETED_LOG
        VALUES(OLD.TITLE);
```

# 5.7.1 定义触发器



例：教授的工资不得低于3000元，否则自动改为3000元。

```
CREATE  TRIGGER  Update_Sal
    BEFORE INSERT OR UPDATE OF Sal, Pos
    ON Teacher
    FOR EACH ROW
    WHEN ( NEW.Pos = '教授'    AND    NEW.Sal<3000 )
        NEW.Sal := 3000;
```

下列语句执行时将产生什么结果？

A. INSERT INTO Teachar(Sal, Pos) VALUES(2000, '讲师')

B. INSERT INTO Teachar(Sal, Pos) VALUES(2500, '教授')

C. UPDATE教师A的职称为教授，且教师A的工资为2000

D. UPDATE教师A的工资为3500，且该教师的职称为教授

# 5.7.2 激活触发器



- 用户只能定义**不能执行**触发器。触发器由DBMS自动执行。
- 触发事件发生时触发器被激活。DBMS按顺序执行如下操作：
  - BEFORE触发器；
  - 触发事件；
  - AFTER触发器。
- 有多个触发器时，这些触发器按创建时间顺序依次执行。
- 任一触发器的执行失败都将中止整个操作。

例：部门预算在下午5时后不能改变。

```
CREATE TRIGGER Department_Update
BEFORE UPDATE OF budget ON Departments
FOR EACH STATEMENT
WHEN (CURRENT_TIME > TIME '17:00:00')
  SELECT 1/0 FROM Departments;
```

# 3. 触发器的撤销



DROP TRIGGER <触发器名>

例：DROP TRIGGER Department_Update

注：触发器是与表相关联的，因此，表的撤销将引起该表上的所有触发器同时被撤销。

不同的DBMS对于触发器的支持方式和语法是有所区别的。这是因为SQL标准直到SQL99才将触发器纳入标准，而在此之前几乎所有主流的商用DBMS都已经实现了对触发器的支持。

# 触发器的作用



- 实现由主键和外键所不能保证的参照完整性和数据的一致性；
- 实现比CHECK语句更为复杂的约束；
- 找到数据修改前后表状态的差异，并基于此差异采取行动；
- 跟踪变化，禁止不合规则的操作；
- 级联运行；
- 自动调用存储过程。

# 小结

本章作业：P173　6，7
下周五第3-5章作业一起交



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。为此，RDBMS提供了一套完整性机制，即：

- 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。

- 关系数据库中的三类完整性约束为：
  - 实体完整性约束（由主键约束来实现）
  - 参照完整性约束（由外键约束来实现）
  - 用户定义的完整性约束（NOT　NULL，UNIQUE，CHECK，触发器）
- 触发器是由事件驱动的特殊过程，它的功能非常强大，并不仅限于完整性控制，目前被广泛应用于数据库应用中。