2026年春季学期

数据库系统概论

An Introduction to Database Systems

第五章 数据库完整性

中国科学技术大学
人工智能与数据科学学院

黄振亚，huangzhy@ustc.edu.cn

复习：数据由DBMS统一管理和控制

- **DBMS提供的数据控制功能**
  - **1. 数据的安全性（Security）保护 （第4章）**

    保护数据，以防止不合法的使用造成的数据的泄密和破坏。
  - **2. 数据的完整性（Integrity）检查 （第5章）**

    将数据控制在有效的范围内，或保证数据之间满足一定的关系。
  - **3. 数据库恢复（Recovery） （第10章）**

    将数据库从错误状态恢复到某一已知的正确状态。
  - **4. 并发（Concurrency）控制 （第11章）**

    对多用户的并发操作加以控制和协调，防止相互干扰而得到错误的结果

第五章 数据库完整性

5.1 数据库完整性概述

- 数据库的完整性
  - 数据的正确性和相容性
  - 正确性：数据是符合现实世界语义，反映了当前实际状况
  - 相容性：数据库同一对象在不同关系表中的数据符合逻辑

例如，

- 学生的学号必须唯一
- 性别只能是男或女
- 本科学生年龄的取值范围为14~50的整数
- 学生所选的课程必须是学校开设的课程
- 学生所在的院系必须是学校已成立的院系
- 。。。

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(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
  - 完整性约束条件也称为完整性规则，是数据库中的数据必须满足的语义约束条件
  - 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
  - 这些完整性一般由SQL的数据定义语言语句DDL来实现

数据库完整性(续)

为维护数据库的完整性，DBMS必须：

- 2.提供完整性检查的方法
  - 数据库管理系统中检查数据是否满足完整性约束条件的机制称为完整性检查。
  - 什么时候检查，怎么查
  - 一般在INSERT、UPDATE、DELETE语句执行后开始检查，也可以在事务提交时检查
- 3.违约处理
  - 数据库管理系统若发现用户的操作违背了完整性约束条件，就采取一定的动作
  - 拒绝（NO ACTION）执行，级联（CASCADE）执行

第五章 数据库完整性

复习： 实体完整性

规则： 实体完整性规则（Entity Integrity）

若属性*A*是基本关系*R*的主属性，则属性*A*不能取空值

例，导师指导研究生：

SAP(SUPERVISOR，SPECIALITY，POSTGRADUATE)

POSTGRADUATE：主码（假设研究生不会重名）

不能取空值

复习：参照完整性规则

外码（Foreign Key）

- 设*F*是基本关系*R*的一个或一组属性，但不是关系*R*的码。如果F与基本关系*S*的主码K_s相对应，则称F是基本关系*R*的外码
- 基本关系*R*称为参照关系（Referencing Relation）
- 基本关系*S*称为被参照关系（Referenced Relation）或目标关系（Target Relation）

复习：参照完整性规则

规则2.2 参照完整性规则

若属性（或属性组）F是基本关系R的外码，它与基本关系S的主码K_s相对应（基本关系R和S不一定是不同的关系），则对于R中每个元组在F上的值必须为：

- 或者取空值（F的每个属性值均为空值）
- 或者等于S中某个元组的主码值

复习：参照完整性规则

〔例〕：

选修（学号，课程号，成绩）

“学号”和“课程号”可能的取值：

（1）选修关系中的主属性，不能取空值

（2）只能取相应被参照关系中已经存在的主码值

复习：用户定义的完整性

- 针对某一具体关系数据库的约束条件，反映某一具体应用所涉及的数据必须满足的语义要求
- 关系模型应提供定义和检验这类完整性的机制，以便用统一的系统的方法处理它们，而不要由应用程序承担这一功能

复习：用户定义的完整性(续)

例:

课程(课程号，课程名，学分)

- “课程号”属性必须取唯一值
- 非主属性“课程名”也不能取空值
- “学分”属性只能取值{1，2，3，4}

5.2 实体完整性

- 5.2.1 实体完整性定义
- 5.2.2 实体完整性检查和违约处理

5.2.1 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*/
);
```

5.2 实体完整性

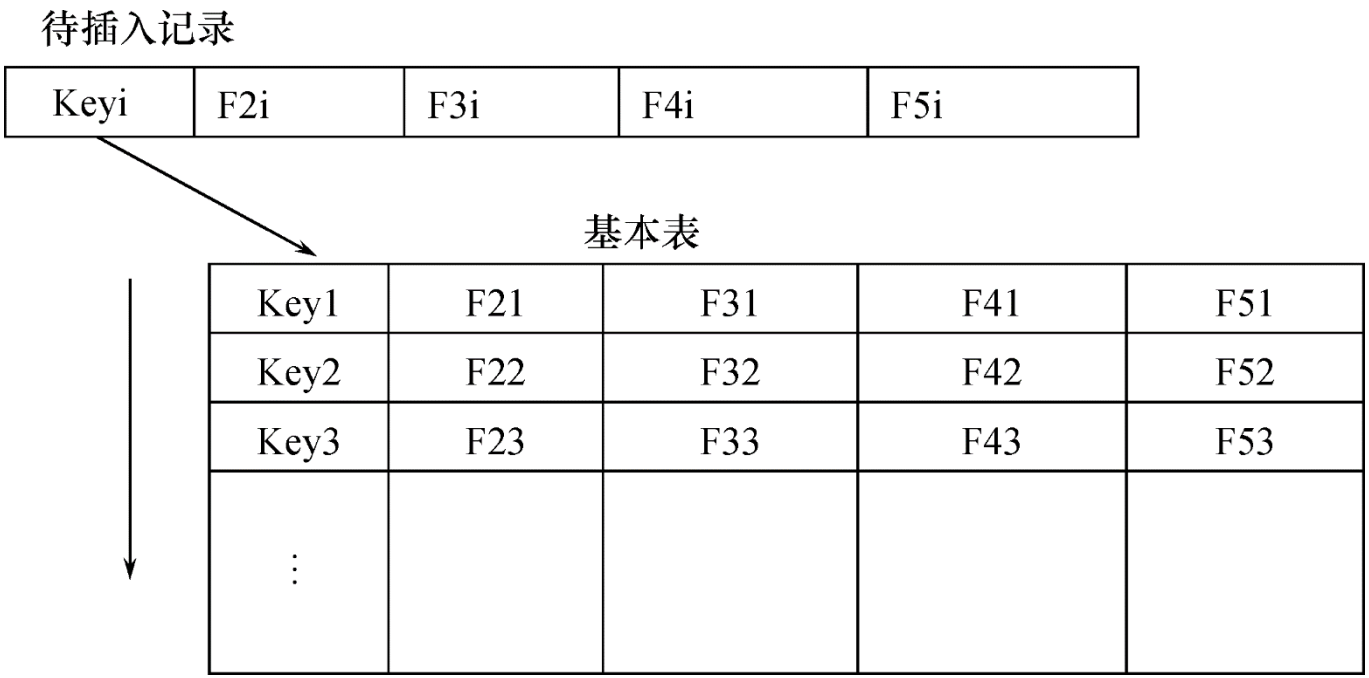
- 5.2.1 实体完整性定义
- 5.2.2 实体完整性检查和违约处理

5.2.2 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)

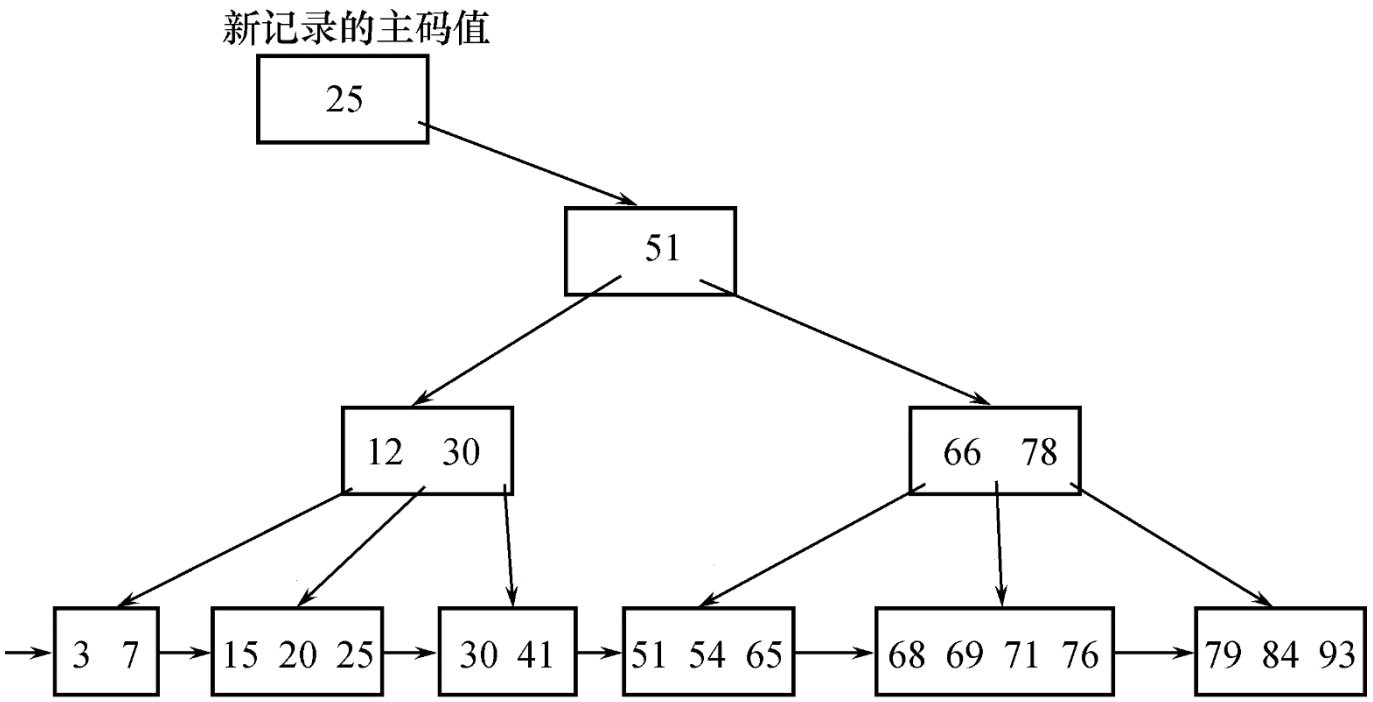
- 检查记录中主码值是否唯一的一种方法是进行全表扫描



为避免对基本表进行全表扫描，RDBMS核心一般都在主码上自动建立一个索引

实体完整性检查和违约处理(续)

- 索引（B+树）
  - 插入25
  - 插入86



第五章 数据库完整性

5.3 参照完整性

- 5.3.1 参照完整性定义
- 5.3.2 参照完整性检查和违约处理

5.3.1 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5.3］ 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)，
 FOREIGN KEY (Cno) REFERENCES Course(Cno)
 );
```

在表级定义参照完整性

5.3 参照完整性

- 5.3.1 参照完整性定义
- 5.3.2 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/级联删除/设置为空值
修改主码值 →	可能破坏参照完整性	拒绝/级联修改/设置为空值

参照完整性违约处理

- 学生表：Student（Sno, Sname, Dno）
- 课程表：Course（Cno，Cname）
- 院系表：Dept（Dno，Dname）
- 学生选课表：SC（Sno，Cno，Grade）
  - 1. SC插入（003,1,90）？（插入参照表数据）
  - 2. Student的002元组Dno的2改为3？（修改参照表的外码）
  - 3. Dept删除2？（删除被参照表元组）
  - 4. Dept的2改为3？（修改被参照表主码）

Sno	Sname	Dno
001	李勇	1
002	刘晨	2

Dno	Dname
1	CS
2	DS

Cno	Cname
1	数据库
2	数学
3	数据结构

Sno	Cno	Grade
001	1	92
001	2	85
001	3	88

参照完整性检查和违约处理

- 例如，对表SC和Student有四种可能破坏参照完整性的情况：
  - SC表中增加一个元组，该元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。
  - 修改SC表中的一个元组，修改后该元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。
  - 从Student表中删除一个元组，造成SC表中某些元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等。
  - 修改Student表中一个元组的Sno属性，造成SC表中某些元组的Sno属性的值在表Student中找不到一个元组，其Sno属性的值与之相等 。

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
    - 例如，学生退学，则删除SC中记录
  - 3. 设置为空值（SET-NULL）
    - 当删除或修改被参照表的一个元组时造成了不一致，则将参照表中的所有造成不一致的元组的对应属性设置为空值

违约处理

例如，有下面2个关系

学生（学号，姓名，性别，专业号，年龄）

专业（专业号，专业名）

- 假设专业表中某个元组被删除，专业号为12
- 按照设置为空值的策略，就要把学生表中专业号=12的所有元组的专业号设置为空值
- 对应语义：某个专业删除了，该专业的所有学生专业未定，等待重新分配专业

✓ 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

根据具体应用而定

参照完整性违约处理

- 学生表：Student（Sno, Sname, Dno）
- 课程表：Course（Cno，Cname）
- 院系表：Dept（Dno，Dname）
- 学生选课表：SC（Sno，Cno，Grade）
  - 1. SC插入（003,1,90）？（插入参照表数据）
  - 2. Student的002元组Dno的2改为3？（修改参照表的外码）
  - 3. Dept删除2？（删除被参照表元组）
  - 4. Dept的2改为3？（修改被参照表主码）

Sno	Sname	Dno
001	李勇	1
002	刘晨	2

Dno	Dname
1	CS
2	DS

Cno	Cname
1	数据库
2	数学
3	数据结构

Sno	Cno	Grade
001	1	92
001	2	85
001	3	88

违约处理(续)

［例5.4］ 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

5.4 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

DBMS VS. 应用程序？

5.4 用户定义的完整性

- 5.4.1 属性上的约束条件的定义
- 5.4.2 属性上的约束条件检查和违约处理
- 5.4.3 元组上的约束条件的定义
- 5.4.4元组上的约束条件检查和违约处理

5.4.1 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5.5］ 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5.6］ 建立学院表School，要求学院名称SHname列取值唯一，学院编号SHno列为主码

```
CREATE TABLE School
        (SHno  CHAR(8) PRIMARY KEY，/*Shno列为主码*/
        SHname  CHAR(9)  UNIQUE，/*要求SHname列值唯一*/
        Shfounddate Date，                /*学院创建日期*/
        );
```

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5.7］ 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)
   );
```

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5.8] SC表的Grade的值应该在0和100之间。

```
CREATE TABLE SC
    ( Sno    CHAR(9) ,
      Cno    CHAR(4),
      Grade  SMALLINT CHECK (Grade>=0 AND Grade <=100),
                                        /*Grade取值范围是0到100*/
      PRIMARY KEY (Sno,Cno),
      FOREIGN KEY (Sno) REFERENCES Student(Sno),
      FOREIGN KEY (Cno) REFERENCES Course(Cno)
    );
```

5.4 用户定义的完整性

- 5.4.1 属性上的约束条件的定义
- 5.4.2 属性上的约束条件检查和违约处理
- 5.4.3 元组上的约束条件的定义
- 5.4.4元组上的约束条件检查和违约处理

5.4.2 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

5.4 用户定义的完整性

- 5.4.1 属性上的约束条件的定义
- 5.4.2 属性上的约束条件检查和违约处理
- 5.4.3 元组上的约束条件的定义
- 5.4.4 元组上的约束条件检查和违约处理

5.4.3 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5.9］ 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

5.4 用户定义的完整性

- 5.4.1 属性上的约束条件的定义
- 5.4.2 属性上的约束条件检查和违约处理
- 5.4.3 元组上的约束条件的定义
- 5.4.4 元组上的约束条件检查和违约处理

5.4.4 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

5.5 完整性约束命名子句

- 完整性约束命名子句
  - CONSTRAINT 约束
  - 语法结构

CONSTRAINT <完整性约束条件名><完整性约束条件>

- <完整性约束条件>包括NOT NULL、UNIQUE、PRIMARY KEY短语、FOREIGN KEY短语、CHECK短语等

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)

- **[例11]建立老师表TEACHER，要求每个教师的应发工资不低于3000元，应发工资是工资列Sal与扣除项Deduct之和。**

```
Create table TEACHER
        (Eno NUMERIC(4) PRIMARY KEY,
        Ename CHAR(10),
        Job CHAR(8),
        Sal NUMERIC(7,2),
        Deduct NUMERIC(7,2),
        Deptno MUMERIC(2),
        CONSTRAINT TEACHERFKEY FOREIGN KEY(Deptno),
        REFERENCES DEPT(Deptno),
        CONSTRAINT C1 CHECK(Sal+Deduct>=3000)
);
```

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**

Alter Table <表名>

[ADD]<新列名><数据类型>[完整性约束]]

[DROP][完整性约束名]]

[Alter] COLUMN<列名><数据类型>]]

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

[例5.12]删除例5.10 Student表中对出生日期的限制。

```
ALTER TABLE Student
    DROP CONSTRAINT C3;
```

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);
```

完整性约束命名子句(续)

(Mysql 例子) 在表Student增加一个约束条件，要求年龄大于10岁

```
Alter Table Student ADD CONSTRAINT C5 CHECK (Sage > 10);
SELECT * FROM information_schema.`TABLE_CONSTRAINTS`
Where table_name='Student' and table_schema='trigger';
```

每次添加约束都是将添加的约束的信息存储到了information_schema这个schema的table_constraints表里

	CONSTRAINT_CATALOG	CONSTRAINT_SCHEMA	CONSTRAINT_NAME	TABLE_SCHEMA	TABLE_NAME	CONSTRAINT_TYPE	ENFORCED
▶	def	trigger	PRIMARY	trigger	Student	PRIMARY KEY	YES
	def	trigger	C1	trigger	Student	CHECK	YES
	def	trigger	C3	trigger	Student	CHECK	YES
	def	trigger	C4	trigger	Student	CHECK	YES
	def	trigger	C5	trigger	Student	CHECK	YES

第五章 数据库完整性

5.6 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（使用完整性约束命名子句）

```
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')

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
  - 定义后，触发器保存在数据库服务器中
  - 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
  - SQL99后写入SQL标准（1999年）

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

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>

- 触发器又叫做事件-条件-动作（Event-Condition-Action）规则。
- 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。

定义触发器(续)

```
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>
```


- 定义触发器的语法说明:
  - **1. 创建者：**表的**拥有者**
    - **一个表上定义一定数量的触发器。具体限制不定**
  - **2. 触发器名**
    - 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，并且触发器名和表名必须在同一模式下
  - **3. 表名：触发器的目标表**
    - 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

定义触发器(续)

```
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>
```


- 定义触发器的语法说明:
  - **4. 触发事件(Event):**
    - 触发事件可以是INSERT、DELETE或UPDATE，也可以是这几个事件的组合
    - 还可以UPDATE OF <触发列，...>，即指明修改哪些列时激活触发器
    - 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

定义触发器(续)


```
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>
```

- 定义触发器的语法说明:
  - **5. 触发器类型**
    - 行级触发器（FOR EACH ROW）
    - 语句级触发器（FOR EACH STATEMENT）

例如, 假设在［例5.11］的TEACHER表上创建了一个AFTER UPDATE触发器触发事件是:

UPDATE TEACHER SET Deptno=5;

- 如果表TEACHER有1000行：
  - 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

行级触发器对DML语句影响的每个行执行一次，
语句级触发器（默认）对每个DML语句执行一次

定义触发器(续)

- **[例5.11]建立老师表TEACHER，要求每个教师的应发工资不低于3000元，应发工资是工资列Sal与扣除项Deduct之和。**

```
CREATE Table TEACHER
 ( Eno NUMERIC(4) PRIMARY KEY,
   Ename CHAR(10),
   Job CHAR(8),
   Sal NUMERIC(7,2),
   Deduct NUMERIC(7,2),
   Deptno MUMERIC(2),
   CONSTRAINT TEACHERFKEY FOREIGN KEY(Deptno),
   REFERENCES DEPT(Deptno),
);
```

创建after update触发器：UPDATE TEACHER set Deptno = 5;

定义触发器(续)

```
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>
```

- **6. 触发条件(Condition)**
  - 触发条件为真， action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
- **7. 触发动作体(Action)**
  - 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
  - 行级触发器：可用New和Old引用事件之后的新值，事件之前的旧值
  - 语句级触发器：不能在触发动作中使用New或Old引用
  - 触发动作体执行失败，激活触发器的事件(增删改)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会发生任何变化（ROLL BACK）

注意：不同的RDBMS产品触发器语法各部相同

定义触发器(续)

[例5.18]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）

其中, Oldgrade是修改前的分数，Newgrade是修改后的分数。

```
CREATE TRIGGER  SC_T
AFTER UPDATE OF Grade ON SC
REFERENCING
     OLD row  AS  OldTuple,          /*SQL标准，定义引用变量*/
     NEW row AS  NewTuple
FOR EACH ROW                                   /行级触发器/
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5.19] 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
                                    /*指明触发器激活的时间是在执行INSERT后*/
REFERENCING   NEW TABLE AS DELTA
FOR EACH STATEMENT
        /*语句级触发器, 即执行完INSERT语句后下面的触发动作体才执行一次*/
    INSERT INTO StudentInsertLog (Numbers)
      SELECT COUNT(*)
      FROM DELTA
```

定义触发器(续)

TEACHER (Eno, Ename, Job, Sal, Deduct, Deptno)

［例5.20］ 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
                                   /*触发事件是插入或更新操作*/
   REFERENCING  NEW row AS  NewTuple
   FOR EACH ROW                    /*行级触发器*/
   BEGIN                           /*定义触发动作体，是PL/SQL过程块*/
      IF (newtuple.Job='教授') AND (newtuple.Sal < 4000)
      THEN   newtuple.Sal :=4000;
      END IF;
  END;
```

定义触发器(续)

TEACHER (Eno, Ename, Job, Sal, Deduct, Deptno)

［例5.20续］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

1. 建立工资变化表Sal_log

```
CREATE TABLE Sal_log
   (Eno    NUMERIC(4)
    Sal    NUMERIC(7，2)，
    Username  char(10)，                    /修改人/
    Date   TIMESTAMP                        /修改日期/
    foreign key (Eno) references Teacher(Eno)，
   );
```

定义触发器(续)

TEACHER (Eno, Ename, Job, Sal, Deduct, Deptno)

［例5.20续］（续）

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

定义触发器(续)

TEACHER (Eno, Ename, Job, Sal, Deduct, Deptno)

［例5.20续］（续）

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

定义触发器(续)

触发器与完整性约束

- 完整性约束是被动的约束机制
  - 当被限制的数据对象发生变化时，系统检查对象变化后是否满足限制，如果不满足，则拒绝变化操作
- 触发器是主动的规则检查(特定事件发生)：
  - 当特定系统事件（对表的增删改）发生时，对规则的条件进行检查，如果条件成立，执行规则动作。否则不执行
  - 触发事件的操作可能执行，也可能不执行

定义触发器(续)

(Mysql例子) 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表SC_U中： SC_U（Sno, Cno, Oldgrade, Newgrade）

原SC表

Sno	Cno	Grade	Semester
20180001	1	92	20192
20180001	2	85	20201
20180001	3	88	20202
20180002	2	90	20192
20180002	3	80	20201

触发器

```
delimiter //
CREATE TRIGGER SC_T
AFTER UPDATE ON SC
FOR EACH ROW
begin
IF(New. Grade >= 1.1*Old.Grade) THEN
INSERT INTO SC_U
VALUES (Old.Sno, Old.Cno, Old.Grade, New.Grade);
END IF;
end //
delimiter ;
```

更新

```
UPDATE SC SET Grade=100 where Sno='20180002' and Cno='3';
```

SC_U的结果:

Sno	Cno	Oldgrade	Newgrade
20180002	3	80	100

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

- 触发器执行，由触发事件激活，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
  - 对于同一个表上的多个Before(after)触发器，顺序
    - “谁先创建谁先执行”
    - 按触发器名称的字母排序顺序执行
    - 不同RDBMS实现方式不同

激活触发器(续)

［例］执行修改某个教师工资的SQL语句。

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

激活上述定义的触发器执行顺序是：

- 执行触发器 Insert_Or_Update_Sal　　【例5.20】 before触发器
- 执行SQL语句“UPDATE Teacher SET Sal=800 WHERE Ename='陈平';”
- 执行触发器Insert_Sal　　【例5.20续】after触发器
- 执行触发器Update_Sal　　【例5.20续】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>;

- 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］ 删除教师表Teacher上的触发器Insert_Sal

DROP TRIGGER Insert_Sal ON Teacher;

不同的DBMS有不同的语法定义

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

- 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