南京大学 NANJING UNIVERSITY

奋进行动 团结奋斗 争先进位 坐言起行 应势而动

数 据 管 理 基 础

第5章 数据库完整性

智能软件与工程学院

数据库完整性

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

完整性 vs. 安全性

- 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

完整性机制 1

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

完整性机制 2

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
- 完整性约束条件的定义，一般由SQL的数据定义语句来实现

完整性机制 3

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。
- 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

完整性机制 4

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）执行该操作
  - 级连（CASCADE）执行其他操作

完整性约束定义的分类 1

- 按照定义对象可分为三种类型：与表有关的约束、域（Domain）约束、断言（Assertion）

1. 与表有关的约束：
   - 在CREATE TABLE命令中定义的完整性约束，是单张表中的数据取值约束，包括PRIMARY KEY、FOREIGN KEY、UNIQUE、NOT NULL、CHECK约束、DEFAULT(缺省值)约束。
   - 定义方式可分为：列约束（作为列的定义成分）和表约束（作为表定义的组成成分单独定义）
2. 域（Domain）约束：在域定义中被定义的一种约束。
3. 断言（Assertion）：
   - 一个断言就是一个谓词表达式，它表达了希望数据库总能满足的条件
   - 断言可以与一个或多个表关联，可以理解为能应用于多个表的check约束
   - 因此，必须在表定义之外独立创建断言

完整性约束定义的分类 2

- 按照约束状态分类：
  1. 静态约束：要求数据库在任何时候都要满足的约束
  2. 动态动态：在数据库改变状态时要满足的约束

- SQL支持如下几种约束：
  - 静态约束
    - 与表有关的约束，域约束
    - 断言（请谨慎使用）
  - 动态约束：触发器
    - 触发器可以细分引起完整性约束检查的条件
    - 触发器不仅可以检查数据完整性，还可以进行数据处理、实现某些业务处理逻辑。

Basic SQL Syntax for Create Table

```
CREATE TABLE [schema_name.]table_name ( relational_properties );
```

relational_properties:

```
column_name datatype [ DEFAULT expr ] { col_constraint }
{ , column_name datatype [ DEFAULT expr ] { col_constraint } }
|   [ , table_constraint { , table_constraint } ]
```

- schema name & table name
- column defintion
  - column name & data type
  - an optional DEFAULT clause
- table constraints
  - integrity constraint in a single column
  - integrity constraints in multiple columns

SQL语言对完整性约束规则的支持

- 创建基表命令 （CREATE TABLE）
  - 需要定义的内容
    - 模式名 & 表名
    - 列的定义
      - 列名 & 数据类型
      - 列的缺省值定义：
        DEFAULT { default_constant | NULL }
      - 列级约束的定义：单个列上的数据完整性约束定义
    - 表级约束的定义：多个列之间的数据完整性约束定义

SQL语言对完整性约束规则的支持

```
{   NOT NULL   |
   [ CONSTRAINT constraint_name ]
        UNIQUE
      | PRIMARY KEY
      | CHECK ( search_condition )
      | REFERENCES table_name [ ( column_name ) ]
          [ ON DELETE CASCADE | RESTRICT | SET NULL ]
          [ ON UPDATE CASCADE | RESTRICT | SET NULL ] }
```

图(a) 基于‘单个列’的取值约束
(可作为列定义的组成成分)

SQL语言对完整性约束规则的支持

```
[ CONSTRAINT constraint_name ]
  {   UNIQUE ( colname { , colname ... } )
    | PRIMARY KEY ( colname { , colname ... } )
    | CHECK ( search_condition )
    | FOREIGN KEY ( colname { , colname ... } )
        REFERENCES table_name [ ( colname { ,colname... } ) ]
          [ ON DELETE CASCADE | RESTRICT | SET NULL ]
          [ ON UPDATE CASCADE|RESTRICT|SET NULL ]   }
```

图(b) 基于‘多个列’的取值约束
（作为表级约束，需另起一行单独定义）

第5章 数据库完整性

5.1.1 实体完整性定义

- 在 CREATE TABLE 命令中
  - 用 PRIMARY KEY 定义主码
  - 或者，用 UNIQUE + NOT NULL 定义唯一码

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，关系数据库管理系统按照实体完整性规则自动进行检查。包括：
  - 检查主码值是否唯一，如果不唯一则拒绝插入或修改
  - 检查主码的各个列是否为空，只要有一个为空就拒绝插入或修改

实体完整性定义 1

- 关系模型的实体完整性定义
  - 在 CREATE TABLE 命令中用 PRIMARY KEY 定义主码，或者用 UNIQUE + NOT NULL 定义唯一码
- 单列构成的码有两种说明方法
  - 定义为列级约束条件（作为某一列的定义成分）
  - 定义为表级约束条件（作为表的定义成分）
- 对多个列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 2

- [例5.1] 将Student表中的Sno列定义为码

- 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)
);
```

- 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 3

- [例5.2] 将SC表中的Sno，Cno列组定义为码

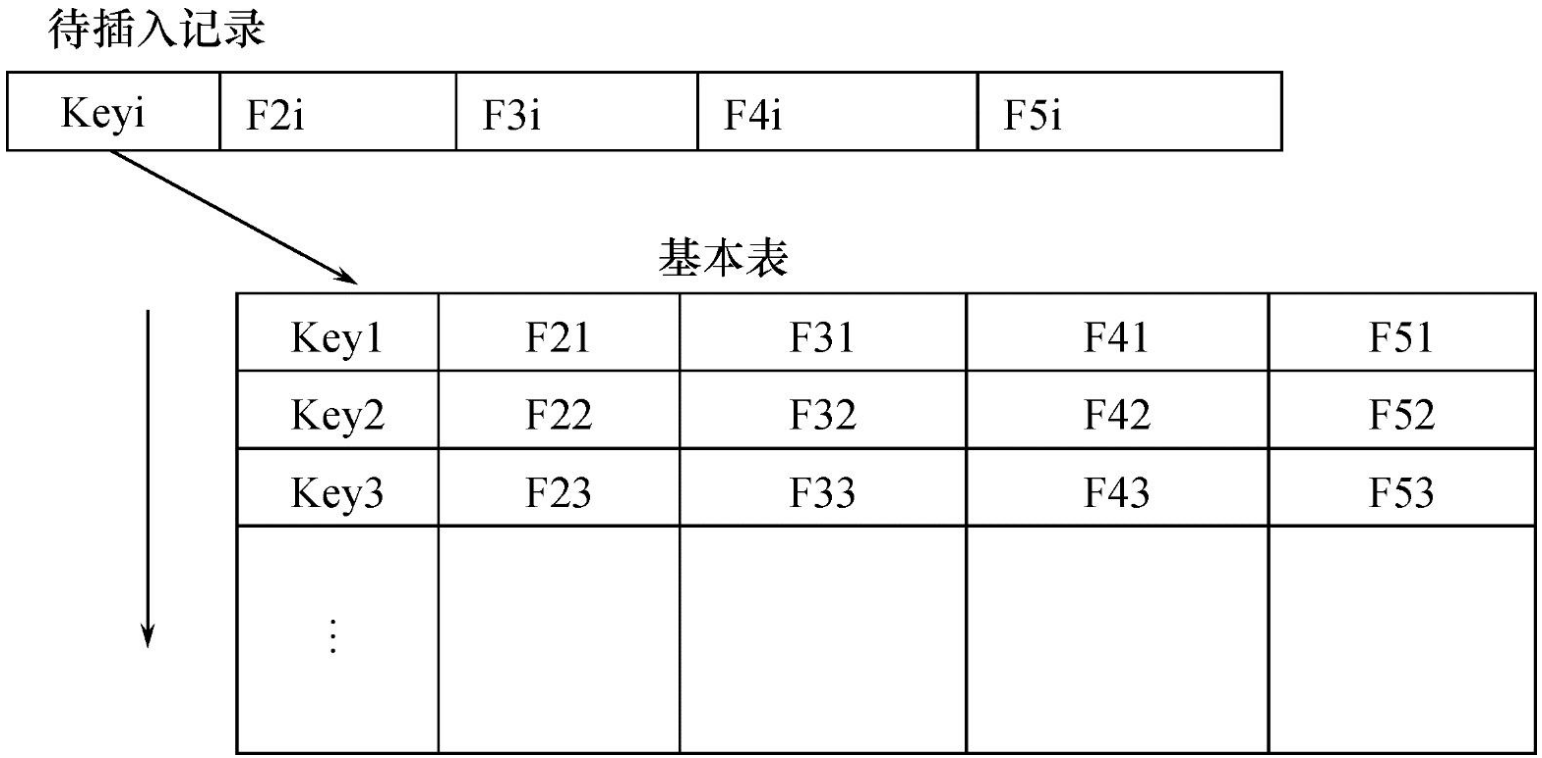
```
CREATE TABLE SC (
    Sno   CHAR(9)  NOT NULL,
    Cno  CHAR(4)  NOT NULL,
    Grade    SMALLINT,
    PRIMARY KEY (Sno,Cno)        /*只能在表级定义主码*/
);
```

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，关系数据库管理系统按照实体完整性规则自动进行检查。包括：
  - 检查主码的各个列是否为空，只要有一个为空就拒绝插入或修改
  - 检查主码值是否唯一，如果不唯一则拒绝插入或修改

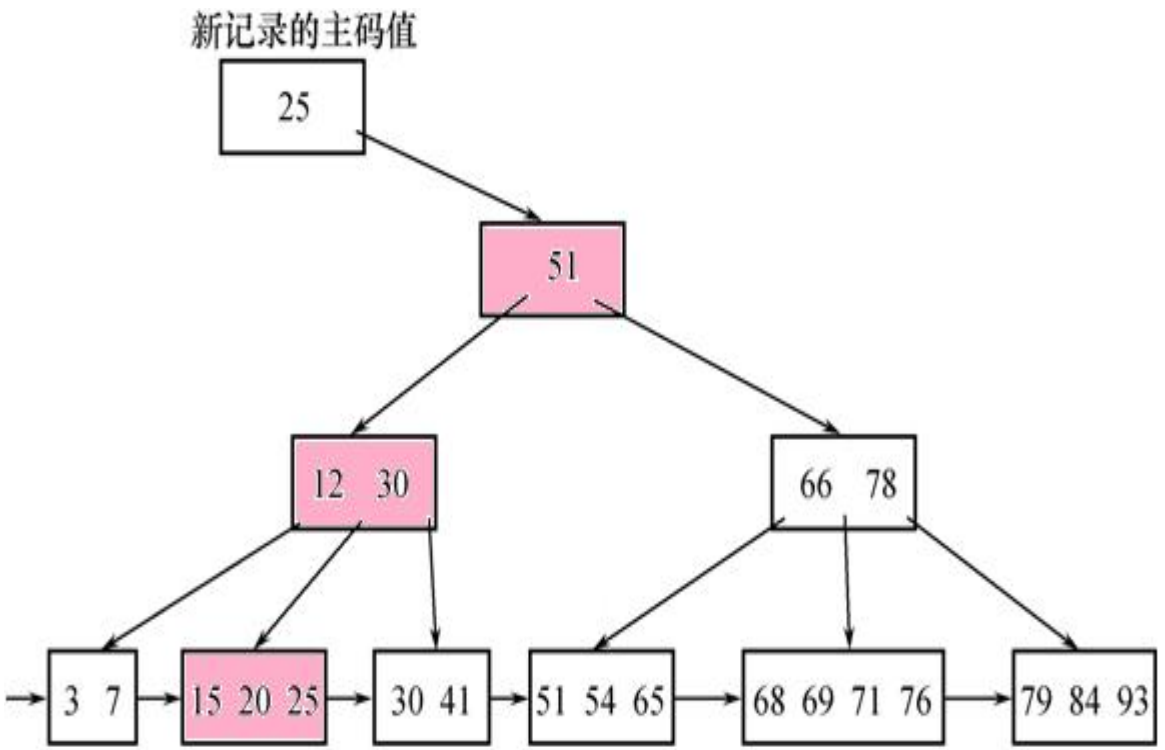
实体完整性检查 1

- 检查记录中主码值是否唯一的一种方法是进行全表扫描
  - 依次判断表中每一条记录的主码值与将插入记录上的主码值（或者修改的新主码值）是否相同
  - 缺点：十分耗时



实体完整性检查 2

- 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，如B+树索引



- 例如：
  - 新插入记录的主码值是25
  - 通过主码索引，从B+树的根结点开始查找
  - 读取3个结点：
    - 根结点（51）
    - 中间结点（12 30）
    - 叶结点（15 20 25）
  - 该主码值已经存在，不能插入这条主码值为25的新记录

第5章 数据库完整性

5.2.1 参照完整性定义

- 在 CREATE TABLE 命令中用 FOREIGN KEY 短语定义外码，用 REFERENCES短语指明这些外码参照哪些表的主码
- 单列构成的外码，既可以作为列级约束、也可以作为表级约束来说明
- 多列构成的外码，只能作为表级约束来说明

5.2.2 参照完整性检查和违约处理

- 一个参照完整性将两个表中的相应元组联系起来
- 对被参照表和参照表进行增删改操作时有可能破坏参照完整性，必须进行参照完整性检查

参照完整性定义

- [例5.3] 定义SC中的参照完整性

```
CREATE TABLE SC (
    Sno    CHAR(9)  NOT NULL  REFERENCES Student(Sno),
                /* 在列级定义参照完整性 - 外码 Sno */
    Cno    CHAR(4)  NOT NULL,
    Grade    SMALLINT,
    PRIMARY KEY (Sno, Cno),
                /* 在表级定义实体完整性 - 主码 (Sno,Cno) */
    FOREIGN KEY (Cno) REFERENCES Course(Cno)
                /* 在表级定义参照完整性 - 外码 Cno */
);
/* 单列构成的主码和外码，可以作为列级约束、也可以作为表级约束进行说明 */
/* 多列构成的主码和外码，只能作为表级约束进行说明 */
```

参照完整性检查 1

- 例如，对参照表SC和被参照表Student，有四种可能破坏参照完整性的情况：
  1. 在参照表SC中增加一个元组，该元组的Sno列的值v在被参照表Student中不存在（找不到一个Student元组，其Sno列的值与v相等）
  2. 修改参照表SC中的一个元组，修改后该元组的Sno列的值new_v在被参照表Student中不存在（找不到一个Student元组，其Sno列的值与new_v相等)
  3. 从被参照表Student中删除一个元组，造成参照表SC中某些元组的Sno列的值在被参照表Student中不存在
  4. 修改被参照表Student中一个元组的Sno列，造成参照表SC中某些元组的Sno列的值在被参照表Student中不存在

参照完整性检查 2

表5.1 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

参照完整性违约处理 1

- 参照完整性违约处理
  - 拒绝（NO ACTION）执行
    - 不允许该操作执行。该策略一般设置为默认策略
  - 级联（CASCADE）操作
    - 当删除或修改被参照表（Student）的一个元组造成了与参照表（SC）的不一致，则删除或修改参照表中的所有造成不一致的元组
  - 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列设置为空值。
    - 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

不同DBMS对外码约束的违约处理规则（ON DELETE / ON UPDATE）

X/Open Full SQL-99	MySQL	ORACLE INFORMIX	DB2 UDB SQLserver
NO ACTION CASCADE SET NULL SET DEFAULT RESTRICT	RESTRICT CASCADE SET NULL NO ACTION SET DEFAULT	RESTRICT CASCADE SET NULL SET DEFAULT	NO ACTION CASCADE SET NULL RESTRICT

- 说明：
  - 带下划线的是缺省处理规则（没有定义ON DELETE/UPDATE时采用的处理规则）
  - NO ACTION 等同于 RESTRICT

参照完整性违约处理 2

- 例如，有下面2个关系

外码

  - 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

参照完整性违约处理 3

- [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*/
);
```

总结：在定义列级约束时需要考虑的内容1

```
{   NOT NULL   |
   [ CONSTRAINT constraint_name ]
         UNIQUE
       | PRIMARY KEY
       | CHECK ( search_condition )
       | REFERENCES table_name [ ( column_name ) ]
           [ ON DELETE CASCADE | RESTRICT | SET NULL ]
           [ ON UPDATE CASCADE | RESTRICT | SET NULL ] }
```

- **NOT NULL vs. DEFAULT NULL**
- **Constraint name**
  - 对某个数据约束条件进行命名（可选项）
  - 以利于以后使用 **ALTER TABLE**命令来修改表中的数据约束定义

总结：在定义列级约束时需要考虑的内容 2

```
{   NOT NULL   |
   [ CONSTRAINT constraint_name ]
        UNIQUE
      | PRIMARY KEY
      | CHECK ( search_condition )
      | REFERENCES table_name [ ( column_name ) ]
          [ ON DELETE CASCADE | RESTRICT | SET NULL ]
          [ ON UPDATE CASCADE | RESTRICT | SET NULL ] }
```

- UNIQUE vs. NOT NULL
  - UNIQUE 列可以取空值
  - 候选码(candidate key)：UNIQUE+NOT NULL
- PRIMARY KEY vs. NOT NULL
  - 主码定义自动隐含着‘非空’约束要求

总结：在定义列级约束时需要考虑的内容 3

```
{   NOT NULL   |
    [ CONSTRAINT constraint_name ]
          UNIQUE
        | PRIMARY KEY
        | CHECK ( search_condition )
        | REFERENCES table_name [ ( column_name ) ]
            [ ON DELETE CASCADE | RESTRICT | SET NULL ]
            [ ON UPDATE CASCADE | RESTRICT | SET NULL ] }
```

- **REFERENCES**
  - **FOREIGN KEY（外码） vs. PRIMARY KEY（主码）**
  - **外码上的取值约束及其一致性的保证措施：**

    CASCADE | RESTRICT | SET NULL

- **CHECK**
  - **其他任意的单个列上的取值约束**

总结：Forengn Key 约束

```
FOREIGN KEY ( colname { , colname ... } )
  REFERENCES table_name [ ( colname { ,colname... } ) ]
    [ ON DELETE CASCADE | RESTRICT | SET NULL ]
    [ ON UPDATE CASCADE | RESTRICT | SET NULL ]  }
```

- **foreign key ..... references .....**

  定义主码、外码之间的引用关系。当对参照表中的外码进行赋值时，需要检查外码值的正确性。

- **on delete ..... / on update .....**

  当在被参照表中删除元组或修改主码值时，需要维护参照表中外码值的正确性，具体方式如下：

  ① cascade：同步做连带删除/更新

  ② restrict：如果在参照表中存在与被删除或修改的主码相关的元组，则拒绝本次对被参照表的delete/update操作 (缺省的维护模式)

  ③ set null：如果在参照表中存在与被删除或修改的主码相关的元组，则自动地将相关元组上的外码值置为空(NULL)

第5章 数据库完整性

5.3 用户定义的完整性

- 用户定义的完整性是：针对某一具体应用的数据必须满足的语义要求
  - 列级约束：单个列上的约束条件
  - 表级约束：元组上的约束条件（涉及单个或多个列）
    - 同单个列上的列级约束相比，表级约束可以设置不同列之间的取值的相互约束条件

- 关系数据库管理系统提供了定义和检验用户定义完整性的机制，不必由应用程序承担
  - 插入元组或修改列的值时，关系数据库管理系统检查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

- 在执行 CREATE TABLE 命令时，给出用户定义的完整性的定义

- 列上的约束条件
  - 列值非空：NOT NULL
  - 列值唯一：UNIQUE
  - 定义列值应该满足的条件：CHECK <条件表达式>
    - <条件表达式>仅涉及到当前的单个列

- 元组上的约束条件
  - 列值唯一：UNIQUE(<列名> [, <列名>]...)
  - 定义列值应该满足的条件：CHECK <条件表达式>
    - <条件表达式>可能涉及单个或多个列

列上约束条件的定义 - 列值非空

❑[例5.5] 在定义SC表时，说明Sno、Cno、Grade列不允许取空值。

```
CREATE TABLE SC (
    Sno CHAR(9)  NOT NULL,
    Cno CHAR(4)  NOT NULL,
    Grade  SMALLINT NOT NULL,
    PRIMARY KEY (Sno, Cno),
    …
    /* 在表级定义了实体完整性，Sno和Cno作为主码的组成成分，
       自动隐含着不允许取空值的约束，因此，这两个列上的列级
      NOT NULL约束可以不写 * /
);
```

列上约束条件的定义 - 列值唯一

- [例5.6] 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  NOT NULL,
    /*要求Dname列值唯一, 并且不能取空值*/
    Location  CHAR(10),
    PRIMARY KEY (Deptno)
);
```

列上约束条件的定义 - 列值是否满足条件表达式 1

- 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’)),
        /*性别列Ssex只允许取'男'或'女' */
    Sage  SMALLINT,
    Sdept  CHAR(20)
);
```

列上约束条件的定义 - 列值是否满足条件表达式 2

- [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)
);
```

元组上约束条件的定义

- [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两个列值之间的约束条件*/
);
```

  - 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

第5章 数据库完整性

完整性约束命名子句

- 完整性约束的定义与修改
  - 在CREATE TABLE语句中定义完整性约束
  - 在定义完整性约束时，可以对完整性约束进行命名
  - 使用ALTER TABLE语句来修改表中的完整性约束的定义，包括：
    - 定义新的完整性约束
    - 删除现有的(命名)完整性约束

- 完整性约束命名子句

  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

创建完整性约束 1

- [例5.10] 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。

创建完整性约束 2

- [例5.11] 建立教师表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,    /*在列级定义主码*/
    Ename  CHAR(10),
    Job    CHAR(8),
    Sal    NUMERIC(7, 2),
    Deduct  NUMERIC(7, 2),
    Deptno  NUMERIC(2),
    CONSTRAINT TEACHERFKey
        FOREIGN KEY (Deptno)  REFERENCES  DEPT(Deptno),
    CONSTRAINT  C1  CHECK (Sal + Deduct >= 3000)
);
```

修改完整性约束

- [例5.12] 去掉例5.10 Student表中对性别的限制。

```
ALTER TABLE Student DROP CONSTRAINT C4;
```

- [例5.13] 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5章 数据库完整性

5.5 域中的完整性限制

- 在用CREATE DOMAIN语句定义域时，可以定义域上的完整性约束，即该域中的值应该满足的约束条件。
- 可以方便对所有使用相同‘域’的列的完整性约束的统一定义
- 例5.15 至 例5.17

```
CREATE DOMAIN GenderDomain CHAR(2)
        CONSTRAINT  GD  CHECK(VALUE IN (‘男’, ‘女’)) ;

ALTER DOMAIN GenderDomain DROP CONSTRAINT  GD ;

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

第5章 数据库完整性

5.6 断言

- SQL中，可以使用 CREATE ASSERTION 语句，通过声明性断言来指定更具一般性的约束。
- 可以定义涉及多个表的或聚集操作的比较复杂的完整性约束。
- 断言创建以后，任何对断言中所涉及的关系的操作都会触发关系数据库管理系统对断言的检查，任何使断言不为真值的操作都会被拒绝执行。
- 如果断言很复杂，则系统在检测和维护断言的开销较高，这是在使用断言时应该注意的。

断言相关语句

- 创建断言的语句格式

  CREATE ASSERTION <断言名> <CHECK 子句>;

  - 每个断言都被赋予一个名字，<CHECK 子句>中的约束条件与WHERE子句的条件表达式类似。

- 删除断言的语句格式为

  DROP ASSERTION <断言名>;

创建断言 1

- [例5.18] 限制数据库课程最多60名学生选修

```
CREATE  ASSERTION  ASSE_SC_DB_NUM
CHECK ( 60 >= ( Select  count(*)  /*此断言的谓词涉及聚集操作count*/
                    From  Course, SC
                    Where  SC.Cno = Course.Cno  and
                           Course.Cname ='数据库')
);
```

创建断言 2

- [例5.19] 限制每一门课程最多60名学生选修

```
CREATE  ASSERTION  ASSE_SC_CNUM1
CHECK ( 60 >= ALL ( SELECT  count(*)
                    FROM    SC
                    GROUP by  Cno )
);
 /* 此断言的谓词，涉及使用到聚集操作count 和分组函数group by
    的SQL语句 */
```

创建断言 3

- [例5.20] 限制每个学期每一门课程最多60名学生选修。
  - 首先需要修改SC表的模式，增加一个“学期(TERM)”列

```
ALTER TABLE SC ADD TERM DATE;
```

  - 然后，定义断言

```
CREATE ASSERTION ASSE_SC_CNUM2
CHECK ( 60 >= ALL ( SELECT count(*)
                    FROM SC
                    GROUP by cno, TERM )
);
```

例5.19 和 例5.20 存在语法错误！

❑[例5.18] 限制数据库课程最多60名学生选修

```
CREATE  ASSERTION  ASSE_SC_DB_NUM
CHECK (
    60 >= ( Select  count(*)
            From  Course, SC
            Where  SC.Cno = Course.Cno
                       and
                   Course.Cname ='数据库')
);

/*此断言的谓词涉及聚集操作count*/
```

❑[例5.19] 限制每一门课程最多60名学生选修

```
CREATE  ASSERTION  ASSE_SC_CNUM1
CHECK (
     60 >= ALL ( SELECT  count(*)
                 FROM  SC
                 GROUP by  Cno )
);

 /* 此断言的谓词，涉及使用到聚集操作
    count 和分组函数group by的SQL语句*/
```

第5章 数据库完整性

5.7 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
  - 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

- 不同的RDBMS产品实现的触发器语法各不相同、互不兼容。

定义触发器 1

- **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‘<触发事件> ON <表名>’发生时，对规则的<触发条件>进行检查，如果<触发条件>成立则执行规则中的动作‘<触发动作体>’，否则不执行该动作。
  - 规则中的<触发动作体>可以很复杂，通常是一段SQL存储过程。

详细的触发器定义命令 1

```
CREATE TRIGGER trigger_name { BEFORE | AFTER }
    { INSERT | DELETE
     | UPDATE [ OF colname { , colname ... } ] }
       ON table_name
    [ REFERENCING corr_name_def { , ..... } ]
    [ FOR EACH ROW | FOR EACH STATEMENT ]
    [ WHEN ( search_condition ) ]
        { statement
        | BEGIN ATOMIC statement; { statement; ... } END
```

详细的触发器定义命令 2

```
CREATE TRIGGER trigger_name { BEFORE | AFTER }
    { INSERT | DELETE
      | UPDATE [ OF colname { , colname ... } ] }
        ON table_name
    [ REFERENCING corr_name_def { , ..... } ]
    [ FOR EACH ROW | FOR EACH STATEMENT ]
    [ WHEN ( search_condition ) ]
        { statement
        | BEGIN ATOMIC statement; { statement; ... } END
```

触发事件

触发条件 / 执行方式

结果事件 / 触发动作体

定义触发器 2

- 表的拥有者才可以在表上创建触发器
- <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
  - 触发器名和表名必须在同一模式下
- { BEFORE | AFTER } <触发事件> ON <表名>
  - 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
  - 触发事件可以是INSERT、DELETE或UPDATE，也可以是这几个事件的组合
  - 触发事件也可以是UPDATE OF <列名>[, <列名>]...，即进一步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 4

❑触发器的定义命令（续）

The corr_name_def that defines a correlation name follows:

```
{     OLD [ ROW ] [ AS ] old_row_corr_name
   | NEW [ ROW ] [ AS ] new_row_corr_name

   | OLD TABLE [ AS ] old_table_corr_name
   | NEW TABLE [ AS ] new_table_corr_name   }
```

定义触发器 3

- 触发器类型
  - 行级触发器（FOR EACH ROW）
  - 语句级触发器（FOR EACH STATEMENT）

- 触发条件
  - 触发器被激活时，只有当<触发条件>为真时才执行<触发动作体>；否则<触发动作体>不执行。
  - 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。

- 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
  - 如果是行级触发器，用户可以在过程体中使用NEW和OLD引用事件之后的新值和事件之前的旧值
  - 如果是语句级触发器，则不能在触发动作体中使用NEW或OLD进行引用
  - 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

触发器类型

- 例如，先在例5.11的TEACHER表上创建一个AFTER UPDATE触发器，触发事件是UPDATE语句。然后执行下述UPDATE操作：

```
UPDATE TEACHER SET Deptno=5;
```

- 假设在表TEACHER有1000行，在执行完上述的UPDATE语句后
  - 如果是语句级触发器，则触发动作只被执行一次
  - 如果是行级触发器，则触发动作将被执行1000次

触发器示例 1

- [例5.21]当对表SC的Grade列进行修改时，若分数增加了10%则将此次操作记录到表SC_U(Sno,Cno,Oldgrade,Newgrade)中去。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,
                    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);
```

触发器示例 2

- [例5.22] 将每次对表Student的插入操作所增加的学生人数记录到表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，对应的触发动作体只
       执行一次 */
INSERT  INTO  StudentInsertLog (Numbers)
SELECT COUNT(*) FROM DELTA ;
```

触发器示例 3

- [例5.23] 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，是一个PL/SQL过程块*/
    IF (newTuple.Job = '教授') AND (newTuple.Sal < 4000)
    THEN newTuple.Sal := 4000;
    END IF;
END;
```

触发器示例 4

[例] 在MyCAP数据库中，当删除一个客户元组时，需要将该客户所有订单上的cid置为空值(set null)

```
create trigger foreign_cid
    after delete on customers
    referencing old as old_custom
    for each row
    begin
        update orders
        set cid = null
        where cid = :old_custom.cid ;
    end;
```

- 思考：
  - ① 在这里为什么使用after触发器？
  - ② 触发器的执行会不会出现违反数据完整性约束的现象？

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，遵循如下的执行顺序：
  - 执行该表上的BEFORE触发器；
  - 激活触发器的SQL语句；
  - 执行该表上的AFTER触发器。

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。

总结：断言 与 触发器

	断言	触发器
1	要求完整性约束条件永远为真。	触发条件：可能有，也可能没有；可能成立，也可能不成立。
2	当数据库中的数据发生变化时，DBMS将在全数据库范围内启动断言检查。	仅当触发事件发生且触发条件成立时，DBMS才启动与之对应的触发器的执行
3	断言仅仅用于完整性约束检查，并在违反约束的情况下拒绝执行用户的数据更新操作。	触发器有助于维护数据库表中的完整性约束，尤其是在未定义主码和外码约束时。
4	断言无法对表中所做更改进行任何跟踪。	触发器可以通过执行SQL程序块或调用存储过程来跟踪表中发生的所有更改，包括用户业务逻辑的实现。
5	断言执行检查的代价较高，现代数据库系统一般不使用断言。（慎用）	触发器不仅可以用来实现复杂的完整性约束检查，还可以用于数据库安全性检查和用户业务逻辑的实现，在现代数据库系统中得到了很好的应用。

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

- 关系数据库管理系统完整性实现的机制
  - 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关系数据库管理系统应采取的动作

课后作业 -- 实验一

- 要求：实验环境的准备、安装与使用
  - 数据库管理系统的安装，初始数据的准备
  - 基本表的创建
  - 初始数据的插入
  - 简单的数据查询与数据操纵