数据库系统基础

哈尔滨工业大学

数据库管理与维护

数据库管理与维护

4.1 数据库管理员的基本职责

4.2 数据库存储与性能管理

4.3 数据库完整性与安全性控制

---- 数据库完整性的概念及分类

---- SQL-DDL中关于完整性的命令

---- 数据库安全性的概念

---- SQL-DCL中关于安全性的命令

---- 数据字典/系统目录和模式

4.4 数据库故障恢复

数据库的完整性与安全性控制

---- 数据库完整性的概念

❑数据库完整性(DB Integrity)是指DBMS应保证DB在任何情况下的

正确性、有效性和一致性

✓广义完整性：语义完整性、并发控制、安全控制、DB故障恢复等

✓狭义完整性：专指语义完整性，DBMS通常有专门的完整性管理机制与程序来处理语义完整性问题。(本讲义指语义完整性)

➢关系模型中有完整性要求

实体完整性

参照完整性

用户自定义完整性

➢数据库设计中，在E-R图中有很多的完整性约束条件，如何施加到数据库的定义中，如何起作用呢？

数据库的完整性与安全性控制

---- 数据库完整性的概念

- 为什么会引发数据库完整性的问题呢？
  - 不正当的数据库操作，如输入错误、操作失误、程序处理失误等
- 数据库完整性管理的作用
  - 防止和避免数据库中不合理数据的出现（salary<0）
  - DBMS应尽可能地自动防止DB中语义不合理现象
  - 如DBMS不能自动防止，则需要应用程序员和用户在进行数据库操作时处处加以小心，每写一条SQL语句都要考虑是否符合语义完整性，这种工作负担是非常沉重的，因此应尽可能多地让DBMS来承担。

数据库的完整性与安全性控制

- DBMS怎样自动保证完整性呢？
  - DBMS允许用户定义一些完整性约束规则(用SQL-DDL来定义)
  - 当有DB更新操作时，DBMS自动按照完整性约束条件进行检查，以确保更新操作符合语义完整性

数据库的完整性与安全性控制

- 完整性约束条件(或称完整性约束规则)的一般形式 （Quad四元组）
  - Integrity Constraint ::= (O，P，A，R)
    - ✓ O—数据集合：约束的对象？
      - ❖ 列、多列(元组)、元组集合
    - ✓ P—谓词条件：什么样的约束？
    - ✓ A—触发条件：什么时候检查？
    - ✓ R—响应动作：不满足时怎么办？

数据库的完整性与安全性控制

按约束对象分类

- 完整性约束条件的类别
  - ❑ 域完整性约束条件
    - ✓ 施加于某一列上，对给定列上所要更新的某一候选值是否可以接受进行约束条件判断，这是孤立进行的。
  - ❑ 关系完整性约束条件
    - ✓ 施加于关系/table上，对给定table上所要更新的某一候选元组是否可以接受进行约束条件判断，或是对一个关系中的若干元组和另一个关系中的若干元组间的联系是否可以接受进行约束条件判断。

数据库的完整性与安全性控制

➢ 按约束来源分类

- ❑ **结构约束**：来自于模型的约束，例如函数依赖约束、主键约束(实体完整性)、外键约束(参照完整性)，只关心数值相等与否、是否允许空值等；
- ❑ **内容约束**：来自于用户的约束，如用户自定义完整性，关心元组或属性的取值范围。例如Student表的Sage属性值在15岁至40岁之间等。

Student

S#	Sname	Ssex	Sage	D#	Sclass
98030101	张三	男	20	03	980301
98030102	张四	女	20	03	980301
98030103	张五	男	19	03	980301
98040201	王三	男	20	04	980402
98040202	王四	男	21	04	980402
98040203	王五	女	19	04	980402

数据库的完整性与安全性控制

➢ 按约束状态分类

- **静态约束：**

  要求DB在任一时候均应满足的约束；例如Sage在任何时候都应满足大于0而小于150(假定人活最大年龄是150)。

- **动态约束：**

  要求DB从一状态变为另一状态时应满足的约束；

  例如工资只能升，不能降：工资可以是800元，也可以是1000元；

  可以从800元更改为1000元，但不能从1000元更改为800元。

Teacher

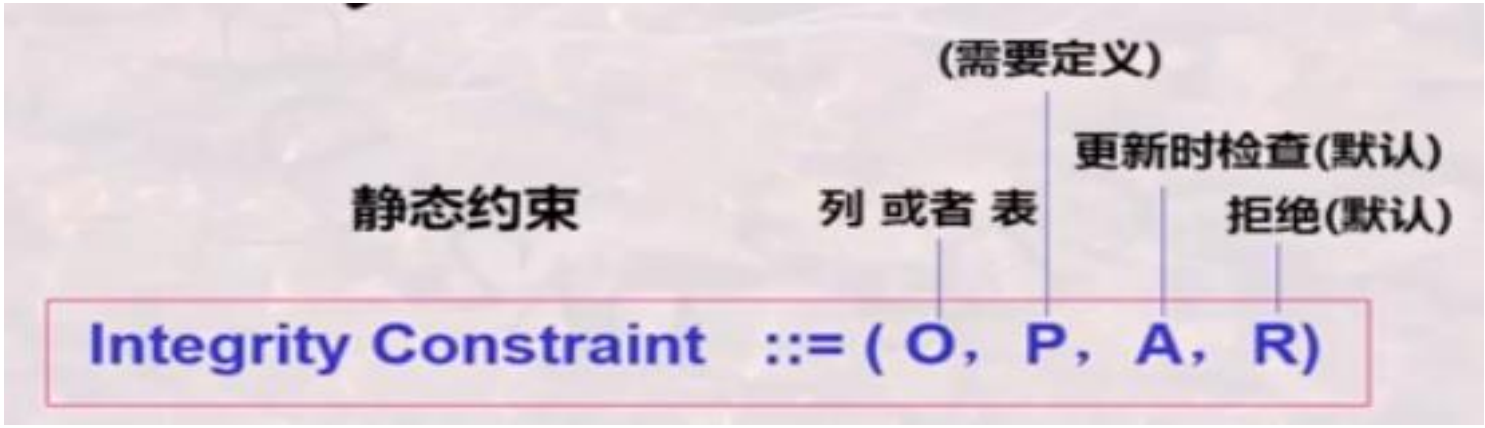
T#	Tname	D#	Salary
001	赵三	01	1200.00
002	赵四	03	1400.00
003	赵五	03	1000.00
004	赵六	04	1100.00

1000变为1100可以，但1100变为1000不可以，只升不降

数据库的完整性与安全性控制

SQL语言支持如下几种约束

- 静态约束
  - 列完整性---域完整性约束
  - 表完整性---关系完整性约束
- 动态约束
  - 触发器



数据库的完整性与安全性控制

Create Table

- Create Table有三种功能：定义关系模式、定义完整性约束、定义物理存储特性
- 定义完整性约束条件：列完整性、表完整性

```
CREATE TABLE   tablename
    ( ( colname   datatype  [ DEFAULT { default_constant | NULL} ]
                    [ col_constr  {col_constr. . .} ]
              | , table_constr
    {,  { colname   datatype   [DEFAULT { default_constant | NULL} ]
                    [col_constr   {col_constr. . .} ]
              | , table_constr }
    . . . } );
```

数据库的完整性与安全性控制

- **Col_constr列约束**：

  一种域约束类型，对单一列的值进行约束

```
{ NOT NULL |                                    //列值非空
    [ CONSTRAINT constraintname ]               //为约束命名，便于以后撤消
        { UNIQUE                                //列值是唯一
        | PRIMARY KEY                           //列为主键
        | CHECK (search_cond)                   //列值满足条件,条件只能使用列当前值
        | REFERENCES tablename [(colname) ]
            [ON DELETE CASCADE] } }
```

//引用另一表tablename的列colname的值，如有ON DELETE CASCADE 或 ON DELETE SET NULL，则删除被引用表的某列值v 时，要将本表的该列值为v 的列值更新为null; 缺省为无操作 。

- **Col_constr列约束：只能应用在单一列上，其后面的约束如UNIQUE, PRIMARY及search_cond只能是单一列唯一、单一列为主键、和单一列相关**

数据库的完整性与安全性控制

- **Col_constr列约束示例**

```
Create Table Student ( S# char(8) not null unique, Sname char(10),
        Ssex char(2) constraint ctssex check (Ssex='男' or Ssex='
        女'), Sage integer check (Sage>=1 and Sage<150),
        D# char(2) references Dept(D#) on delete cascade,
        Sclass char(6) );
```

//假定Ssex只能取{男，女}, 1=<Sage<=150, D# 是外键

Student

S#	Sname	Ssex	Sage	D#	Sclass
98030101	张三	男	20	03	980301
98030102	张玲	女	20	03	980301
98030103	张五	男	19	03	980301
98040201	王二	男	20	04	980402
98040202	王玲	男	21	04	980402
98040203	王五	女	19	04	980402

Dept

D#	Dname	Dean
01	机电	李三
02	经济	李玲
03	计算机	李五
04	自动控制	李六

数据库的完整性与安全性控制

- **Col_constr列约束示例**

```
Create Table Course ( C# char(3) , Cname char(12), Chours integer,
        Credit float(1) constraint ctcredit check (Credit >=0.0 and
        Credit<=5.0 ), T# char(3) references Teacher(T#) on delete
        cascade );
```

//假定每门课学分最多5分，最少0分

数据库的完整性与安全性控制

- **table_constr表约束：**

  一种关系约束类型，对多列或元组的值进行约束

```
[ CONSTRAINT constraintname ]                    //为约束命名，便于以后撤消
    { UNIQUE   (colname {, colname. . .})        //几列值组合在一起是唯一
    | PRIMARY KEY   (colname {, colname. . .})   //几列联合为主键
    | CHECK (search_condition)                   //元组多列值共同满足条件
                                                 //条件中只能使用同一元组的不同列当前值
    | FOREIGN KEY (colname {, colname. . .})
            REFERENCES   tablename [(colname {, colname. . .})]
            [ON DELETE CASCADE] }
                                                 //引用另一表tablename的若干列的值作为外键
```

- **table_constr表约束：是应用在关系上，即对关系的多列或元组进行约束，列约束是其特例**

数据库的完整性与安全性控制

- **table_constr表约束示例**

```
Create Table Student ( S# char(8) not null unique, Sname char(10),
        Ssex char(2) constraint ctssex check (Ssex='男' or Ssex='
        女'), Sage integer check (Sage>1 and Sage<150),
        D# char(2) references Dept(D#) on delete cascade,
        Sclass char(6),primary key(S#) );

Create Table Course ( C# char(3) , Cname char(12), Chours integer,
        Credit float(1) constraint ctcredit check (Credit >=0.0 and
        Credit<=5.0 ), T# char(3) references Teacher(T#) on delete cascade,
        primary(C#), constraint ctcc check(Chours/Credit = 20) );
```

//假定严格约束20学时一个学分

数据库的完整性与安全性控制

➢示例

```
Create  Table  SC ( S#  char(8), C#  char(3),
        Score float(1) constraint ctscore check (Score>=0.0 and
        Score<=100.0),
        forergn  key (S#)  references  student(S#) on delete  cascade,
        forergn  key (C#)  references  course(C#) on delete  cascade );
```

数据库的完整性与安全性控制

➢**check中的条件可以是Select-From-Where内任何Where后的语句，包含子查询。**

➢示例

```
Create  Table  SC ( S#  char(8)  check( S#  in  (select  S#  from student)) ,
                C#  char(3)  check( C#  in (select  C#  from course)) ,
                Score float(1) constraint ctscore check (Score>=0.0 and
                Score<=100.0);
```

数据库的完整性与安全性控制

➢ **Create Table中定义的表约束或列约束可以在以后根据需要进行撤消或追加。撤消或追加约束的语句是 Alter Table(不同系统可能有差异)**

```
ALTER   TABLE   tblname
    [ADD   ( { colname datatype   [DEFAULT {default_const|NULL} ]
              [col_constr {col_constr...} ]  | , table_constr }
              {, colname ...}) ]
   [DROP  { COLUMN  columnname | (columnname {, columnname…})}]
   [MODIFY ( columnname  data-type
              [DEFAULT  {default_const | NULL } ] [   [ NOT ] NULL ]
              {, columnname . . .})]

   [ADD   CONSTRAINT  constr_name]
   [DROP   CONSTRAINT  constr_name]
   [DROP   PRIMARY KEY ] ;
```

数据库的完整性与安全性控制

- 例如，撤消SC表的ctscore约束(由此可见，未命名的约束是不能撤消)

```
Alter  Table  SC
    DROP  CONSTRAINT  ctscore;
```

- 例如，若要再对SC表的score进行约束，比如分数在0~150之间，则可新增加一个约束。在Oracle中增加新约束，需要通过修改列的定义来完成

```
Alter  Table  SC
    Modify ( Score float(1) constraint nctscore check (Score>=0.0 and
        Score<=150.0) );
```

- 有些DBMS支持独立的追加约束, 注意书写格式可能有些差异

```
Alter  Table  SC
    Add  Constraint  nctscore  check  (Score>=0.0 and Score<=150.0) );
```

数据库的完整性与安全性控制

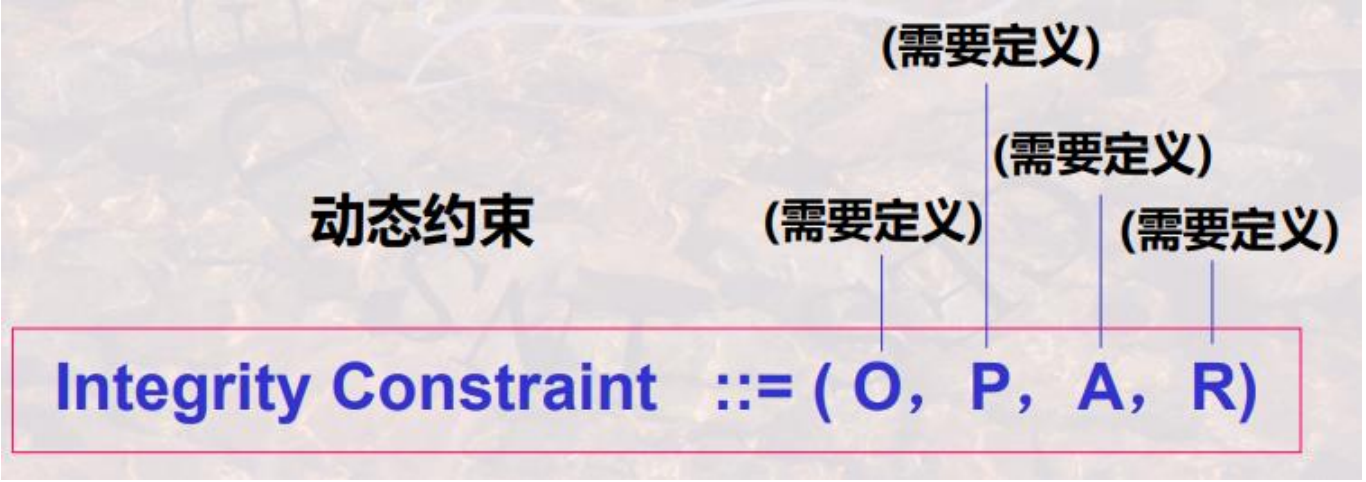
断言ASSERTION

- 语法：CREATE ASSERTION <NAME> CHECK <predicate>
- 表约束，列约束是特殊的断言
- Predicate写法如where子句
- DB每次更新都要检查断言，增加负担

数据库的完整性与安全性控制

SQL语言支持如下几种约束

- 静态约束
  - 列完整性---域完整性约束
  - 表完整性---关系完整性约束
- 动态约束
  - 触发器



数据库的完整性与安全性控制

触发器Trigger

- 为实现动态约束以及多个元组之间的完整性约束，就需要触发器技术**Trigger**
- 而**Create Table**中的表约束和列约束基本上都是静态的约束，也基本上都是对单一列或单一元组的约束(尽管有参照完整性)，
- **Trigger**是一种过程性完整性约束(相比之下，**Create Table**中定义的都是非过程性约束), 是一段程序，该程序可以在特定的时刻被自动触发执行，比如在一次更新操作之前执行，或在更新操作之后执行。

数据库的完整性与安全性控制

- 触发器Trigger

```
CREATE  TRIGGER  trigger_name  BEFORE | AFTER
    { INSERT | DELETE | UPDATE  [OF colname {, colname...}] }
    ON  tablename  [REFERENCING corr_name_def {, corr_name_def...} ]
    [FOR EACH ROW | FOR EACH STATEMENT]
                //对更新操作的每一条结果(前者)，或整个更新操作完成(后者)
    [WHEN  (search_condition)]          //检查条件，如满足执行下述程序
            {  statement                //单行程序直接书写，多行程序要用下行方式
            | BEGIN ATOMIC statement; { statement;...} END }
```

- 触发器Trigger意义：当某一事件发生时(Before|After),对该事件产生的结果(或是每一元组，或是整个操作的所有元组), 检查条件search_condition, 如果满足条件，则执行后面的程序段。条件或程序段中引用的变量可用corr_name_def来限定。

数据库的完整性与安全性控制

- 事件：**BEFORE | AFTER { INSERT | DELETE | UPDATE …}**
  - 当一个事件(Insert, Delete, 或Update)发生之前Before或发生之后After触发
  - 操作发生，执行触发器操作需处理两组值：更新前的值和更新后的值，这两个值由corr_name_def的使用来区分

数据库的完整性与安全性控制

- corr_name_def的定义

```
{ OLD [ROW] [AS]    old_row_corr_name    //更新前的旧元组命别名为
| NEW [ROW] [AS] new_row_corr_name    //更新后的新元组命别名为
| OLD TABLE [AS] old_table_corr_name  //更新前的旧Table命别名为
| NEW TABLE [AS] new_table_corr_name //更新后的新Table命别名为
}
```

- corr_name_def将在检测条件或后面的动作程序段中被引用处理

数据库的完整性与安全性控制

- 触发器Trigger示例一
- 设计一个触发器当进行Teacher表更新元组时，使其工资只能升不能降

```
create trigger  teacher_chgsal  before  update of  salary
    on  teacher
    referencing  new  x, old  y
    for  each  row  when (x.salary < y.salary)
        begin
    raise_application_error(-20003, 'invalid salary on update');
            //Oracle的错误处理函数
        end;
```

数据库的完整性与安全性控制

➢ 触发器Trigger示例二

➢假设student(S#, Sname, SumCourse), SumCourse为该同学已学习课程的门数，初始值为0，以后每选修一门都要对其增1 。设计一个触发器完成上述功能

```
create  trigger  sumc  after  insert  on  sc
referencing new row newi
 for each row
 begin
        update    student
        set SumCourse = SumCourse + 1
        where      S# = :newi.S# ;
end;
```

数据库的完整性与安全性控制

- 触发器Trigger示例三
- 假设student(S#, Sname, Sage, Ssex, Sclass)中某一学生要变更其主码S#的值，如使其原来的98030101变更为99030131, 此时sc表中该同学已选课记录的S#也需自动随其改变。设计一个触发器完成上述功能

```
create trigger updS# after update of S# on student
 referencing old oldi, new newi
 for each row
    begin
        update sc set S# = newi.S#
        where S#= :oldi.S#;
     end;
```

数据库的完整性与安全性控制

- 触发器Trigger示例四
- 假设student(S#, Sname, SumCourse), 当删除某一同学S#时，该同学的所有选课也都要删除。设计一个触发器完成上述功能

```
create trigger delS# after delete on Student
referencing old oldi
for each row
    begin
        delete from sc where S#= :oldi.S#;
    end;
```

数据库的完整性与安全性控制

- 触发器Trigger示例五
- 假设student(S#, Sname, SumCourse), 当删除某一同学S#时，该同学的所有选课中的S#都要置为空值。设计一个触发器完成上述功能

```
create trigger delS# after delete on Student
referencing old oldi
for each row
    begin
        update sc set S#= Null where S#= :oldi.S#;
    end;
```

数据库的完整性与安全性控制

- 触发器Trigger示例六
- 假设Dept(D#, Dname, Dean), 而Dean一定是该系教师Teacher(T#, Tname, D#, Salary)中工资最高的教师。设计一个触发器完成上述功能

```
create  trigger  upddean  before update of Dean on Dept
        referencing new  newi
 for  each row when ( dean not in
       (select  Tname  from  Teacher  where  D# = :newi.D#
and  salary >=
              all(select salary from Teacher where D# = :newi.D#))
      begin
raise_application_error(-20003, 'invalid Dean on update');
      end;
```

数据库的完整性与安全性控制

- 数据库安全性是指DBMS能够保证使DB免受非法、非授权用户的使用、泄漏、更改或破坏的机制和手段
- 数据库安全性管理涉及许多方面
  - 社会法律及伦理方面：私人信息受到保护，未授权人员访问私人信息会违法
  - 公共政策/制度方面：例如，政府或组织的信息公开或非公开制度
  - 安全策略：政府、企业或组织所实施的安全性策略，如集中管理和分散管理，需者方知策略(也称最少特权策略)
  - 数据库系统DBS的安全级别：物理控制、网络控制、操作系统控制、DBMS控制
  - 数据的安全级别: 绝密(Top Secret), 机密(Secret),可信(Confidential)和无分类(Unclassified)

数据库的完整性与安全性控制

- **DBMS的安全机制：**
  - **自主安全性机制：存取控制(Access Control)**
    - ✓ 通过权限在用户之间的传递，使用户自主管理数据库安全性
  - **强制安全性机制：**
    - ✓ 通过对数据和用户强制分类，使得不同类别用户能够访问不同类别的数据
  - **推断控制机制：(可参阅相关文献)**
    - ✓ 防止通过历史信息，推断出不该被其知道的信息；
    - ✓ 防止通过公开信息(通常是一些聚集信息)推断出私密信息(个体信息)，通常在一些由个体数据构成的公共数据库中此问题尤为重要
  - **数据加密存储机制：(可参阅相关文献)**
    - ✓通过加密、解密保护数据，密钥、加密/解密方法与传输

数据库的完整性与安全性控制

- DBA在安全性方面的责任和义务
  - 熟悉相关的法规、政策，协助组织的决策者制定好相关的安全策略
  - 规划好安全控制保障措施，例如，系统安全级别、不同级别上的安全控制措施，对安全遭破坏的响应，
  - 划分好数据的安全级别以及用户的安全级别
  - 实施安全性控制：DBMS专门提供一个DBA帐户，该帐户是一个超级用户或称系统用户。DBA利用该帐户的特权可以进行用户帐户的创建以及权限授予和撤消、安全级别控制调整等。

数据库的完整性与安全性控制

- **自主安全性机制**
- 通常情况下，自主安全性是通过授权机制来实现的。用户在使用数据库前必须由DBA处获得一个帐户，并由DBA授予该帐户一定的权限，该帐户的用户依据其所拥有的权限对数据库进行操作; 同时，该帐户用户也可将其所拥有的权利转授给其他的用户(帐户)，由此实现权限在用户之间的传播和控制。
  - 授权者：决定用户权利的人
  - 授权：授予用户访问的权利

数据库的完整性与安全性控制

DBMS如何实现自主安全性

- **DBMS允许用户定义安全规则（DCL）**
- 发生DB访问操作时，DBMS自动按照安全性控制规则进行检查，通过则可访问，否则拒绝访问

数据库的完整性与安全性控制

- DBMS将权利和用户(帐户)结合在一起，形成一个访问规则表，依据该规则表可以实现对数据库的安全性控制

  AccessRule ::= (S, O, t, P)

  - ✓ S: 请求主体(用户)
  - ✓ O: 访问对象
  - ✓ t: 访问权利
  - ✓ P: 谓词

  ❑ { AccessRule } 通常存放在数据字典或称系统目录中，构成了所有用户对DB的访问权利;

  ❑ 用户多时，可以按用户组建立访问规则

  ❑ 访问对象可大可小(目标粒度Object granularity):属性/字段、记录/元组、关系、数据库

  ❑ 权利：create，增删改查等

  ❑ P：拥有权利需满足条件

数据库的完整性与安全性控制

- 访问权利被分成以下几种
  - (级别1)Select : 读(读DB, Table, Record, Attribute, …)
  - (级别2)Modify : 更新
    - ✓ Insert : 插入(插入新元组, …)
    - ✓ Update : 更新(更新元组中的某些值, …)
    - ✓ Delete : 删除(删除元组, …)
  - (级别3)Create : 创建(创建表空间、模式、表、索引、视图等)
    - ✓ Create : 创建
    - ✓ Alter : 更新
    - ✓ Drop : 删除
- 级别高的权利自动包含级别低的权利。如某人拥有更新的权利，它也自动拥有读的权利。在有些DBMS中，将级别3的权利称为帐户级别的权利，而将级别1和2称为关系级别的权利。
- 超级用户(DBA) ➔ 帐户级别(程序员用户) ➔关系级别(普通用户)

数据库的完整性与安全性控制

- 数据库安全性访问规则示例
- 一个员工管理数据库

  Employee(P#, Pname, Page, Psex, Psalary, D#, HEAD)

  有如下的安全性访问要求：

  - 员工管理人员能访问该数据库的所有内容，具有所有权利
  - 收发（前台）人员也需访问该数据库以确认某员工是哪一个部门的，便于工作，只能访问基本信息，其他信息不允许其访问
  - 每个员工允许其访问自己的记录，以便查询自己的工资情况，但不能修改
  - 部门领导，能够查询其所领导部门人员的所有情况
  - 高层领导能访问该数据库的所有内容，但只能读

数据库的完整性与安全性控制

- 安全性访问规则示例
  - 按名控制安全性
  - 按内容控制安全性

S	O	t	P
员工管理人员	Employee	读、删、插、改	
收发员	Employee(Pname, D#)	读	
高级领导	Employee	读	
员工	Employee	读	P# = UserId
部门领导	Employee	读	Head = UserId

数据库的完整性与安全性控制

➢ **第1种：存储矩阵**

主体 \ 数据对象	O_1	O_2	O_3	…	O_n
S_1	All	All	All	….	All
S_2	Read	–	Read	….	–
…	…	…	…	…	….
S_n	–	Read	–	…	–

数据库的完整性与安全性控制

- 第2种：视图
  - 视图是安全性控制的重要手段
  - 通过视图可以限制用户对关系中某些数据项的存取, 例如：
    - ✓视图1：Create EmpV1 as select * from Employee
    - ✓ 视图2：Create EmpV2 as select Pname, D# from Employee
  - 通过视图可将数据访问对象与谓词结合起来，限制用户对关系中某些元组的存取，例如：
    - ✓ 视图1： Create EmpV3 as select * from Employee where P# = :UserId
    - ✓视图2： Create EmpV4 as select * from Employee where Head = :UserId
  - 用户定义视图后，视图便成为一新的数据对象，参与到存储矩阵与能力表中进行描述

数据库的完整性与安全性控制

SQL-DCL中关于安全性的命令

- 授权命令

GRANT {all PRIVILEGES | privilege {,privilege…}}

ON [TABLE] tablename | viewname

TO {public | user-id {, user-id…}}

[WITH GRANT OPTION];

- user-id ，某一个用户帐户，由DBA创建的合法帐户
- *public*，允许所有有效用户使用授予的权利
- privilege是下面的权利
  - ✓ SELECT | INSERT | UPDATE | DELETE | ALL PRIVILEDGES
- WITH GRANT OPTION选项是允许被授权者传播这些权利

数据库的完整性与安全性控制

示例

- 假定高级领导为Emp0001, 部门领导为Emp0021,员工管理员为Emp2001, 收发员为Emp5001(均为Userld, 也即员工的P#)

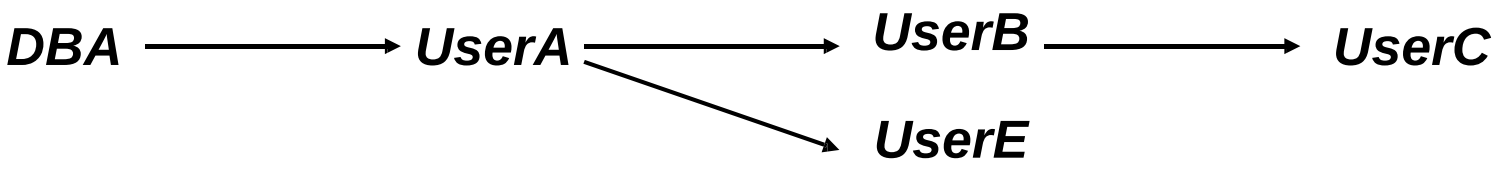
```
Grant  All Priviledges ON  Employee  TO  Emp2001;

Grant  SELECT  ON  EmpV2  TO  Emp5001 ;
Grant  SELECT  ON  EmpV3  TO  public;
Grant  SELECT  ON  EmpV4  TO  Emp0021;
```

- 授予视图访问的权利，并不意味着授予基本表访问的权利(两个级别：基本关系级别和视图级别)
- 授权者授予的权利必须是授权者已经拥有的权利

数据库的完整性与安全性控制

- 授权过程示例(续)

```
Grant Select ON Employee TO UserB WITH GRANT OPTION;
Grant Select ON Employee TO UserC WITH GRANT OPTION;
```



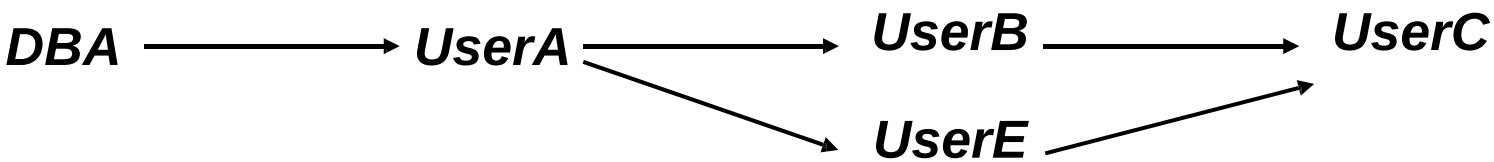
- 注意授权的传播范围问题：
- 传播范围包括两个方面：水平传播数量和垂直传播数量
  - 水平传播数量是授权者的再授权用户数目(树的广度)
  - 垂直传播数量是授权者传播给被授权者，再被传播给另一个被授权者，...传播的深度(树的深度)
- 有些系统提供了传播范围控制，有些系统并没有提供，SQL标准中也并没有限制。

数据库的完整性与安全性控制

- 收回授权命令

  REVOKE {all privilEges | priv {, priv…} } ON tablename | viewname

  FROM {public | user {, user…} };

- 

- 例如

  revoke select on *employee* from *UserB;*

- 当一个用户的权利被收回时，通过其传播给其他用户的权利也将被收回

- 如果一个用户从多个用户处获得了授权，则当其中某一个用户收回授权时，该用户可能仍保有权利。例如UserC从UserB和UserE处获得了授权，当UserB收回时，其还将保持UserE赋予其的权利。

数据库的完整性与安全性控制

- 强制安全性机制
- 强制安全性通过对数据对象进行安全性分级

  绝密(Top Secret), 机密(Secret),可信(Confidential)和无分类(Unclassified)
- 以及对用户也进行上述的安全性分级，从而强制实现不同级别用户访问不同级别数据的一种机制
- 访问规则如下：
  - 用户S, 不能读取数据对象O, 除非Level(S)>=Level(O)
  - 用户S, 不能写数据对象, 除非Level(S)<=Level(O)。

数据库的完整性与安全性控制

- 强制安全性机制的实现
- DBMS引入强制安全性机制, 可以通过扩展关系模式来实现
  - 关系模式:　R(A1: D1, A2: D2, …, An:Dn)
  - 对属性和元组引入安全性分级特性或称分类特性

    R(A1: D1, C1, A2: D2, C2…, An:Dn, Cn, TC)

    其中 C1,C2,…,Cn分别为属性D1,D2,…,Dn的安全分类特性; TC为元组的分类特性

数据库的完整性与安全性控制

- 这样关系中的每个元组, 都将扩展为带有安全分级的元组,例如

P#	C	Pname	C	Psalary	C	TC
Emp001	S	张三	U	10,000	S	S
Emp002	S	李四	U	8,000	S	C
Emp003	S	王五	U	4,000	C	C
Emp004	S	李六	U	2,000	C	C
Emp005	U	张四	U	1,000	S	U

- 强制安全性机制使得关系形成为多级关系(不同级别用户所能看到的关系的子集)，也出现多重实例、多级关系完整性等许多新的问题或新的处理技巧，在使用中需注意仔细研究。

一个例子

```
import pymysql

#建立数据库的连接
db = pymysql.connect(host="127.0.0.1",user="root",password="wu",charset="utf8",database="test_db")
#获取连接下的游标
cursor = db.cursor()
def func(sql):
    try:
        # 执行sql语句
        cursor.execute(sql)
        # 提交到数据库执行
        db.commit()
        #返回结果
        result = cursor.fetchall()
        print(result)
    except:
        # 如果发生错误则回滚
        db.rollback()
sql_create = """CREATE TABLE EMPLOYEE (
         FIRST_NAME  CHAR(20) NOT NULL,
         LAST_NAME  CHAR(20),
         AGE INT,
         SEX CHAR(1),
         INCOME FLOAT )"""
sql_insert = """INSERT INTO EMPLOYEE(FIRST_NAME,
          LAST_NAME, AGE, SEX, INCOME)
          VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
sql_select = """ SELECT * FROM EMPLOYEE WHERE INCOME > 500 """
sql_update =  """ UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = 'M' """
sql_delete = """DELETE FROM EMPLOYEE WHERE AGE > 20 """
func(sql_create)
func(sql_insert)
func(sql_select)
func(sql_update)
func(sql_delete)
db.close()
```

数据库的完整性与安全性控制

关于数据字典/系统目录和模式

- 系统目录
- 系统目录(System Catalogs)是系统维护的，包含数据库中定义的各类对象信息的表或视图，这些对象包括用Create语句定义的表、列、索引、视图、权限、约束等, 这些信息又称数据库的元数据----关于数据的数据。在不同DBMS中，又称数据字典(Data Dictionary(Oracle))、目录表(DB2 UDB)、系统目录(INFORMIX)、系统视图(X/Open)
- 不同DBMS中系统目录存储方式可能是不同的, 但会有一些信息对DBA公开。这些公开的信息, DBA可以使用一些特殊的SQL命令来检索。

数据库的完整性与安全性控制

关于数据字典/系统目录和模式(续)

- **DBA**
- **DBA需要清楚地知道系统目录的内容构成，并知道这些信息的含义和作用，以便能更有效地维护DB以及DBS系统的效率。**
- **DBA需要熟悉DBMS提供的各种检索系统目录的命令，以便能更好地操作系统目录**

数据库的完整性与安全性控制

- 典型的系统目录
- X/Open标准中有一个目录表Info_Schem.Tables, 该表中的一行是一个已经定义的表的有关信息

列名	描述
Table_Schem	表的模式名(通常是表所有者的用户名)
Table_Name	表名
Table_Type	'Base_Table'或 'View'

- 可以使用SQL语句来访问这个表中的信息，比如了解已经定义了哪些表，可如下进行：

```
Select  Table_Name  From  Tables;
```

数据库的完整性与安全性控制

- 模式
- 系统目录的Tables中有一列是模式Schema，模式的含义是指某一用户所设计和使用的表、索引及其他与数据库有关的对象的集合，因此表的完整名应是：模式名.表名。这样做可允许不同用户使用相同的表名，而不混淆。
- 一般而言，一个用户有一个模式。可以使用Create Schema语句来创建模式(用法略，参见相关文献)，在Create Table等语句可以使用所定义的模式名称。

数据库的完整性与安全性控制

- Oracle的数据字典
- Oracle数据字典由视图组成，分为三种不同形式，由不同的前缀标识
  - USER_ :用户视图，用户所拥有的对象，在用户模式中
  - ALL_ :扩展的用户视图，用户可访问的对象
  - DBA_ :DBA视图(所有用户都可访问的DBA对象的子集)
- Oracle数据字典中定义了三个视图USER_Tables, ALL_Tables, 和DBA_Tables供DBA和用户使用数据字典中关于表的信息

ALL_Tables

DBA_Tables

User_Tables:没有 Owner 列

列名	描述
Owner	表的所有者
Table_Name	表名
(其他列...)	磁盘存取和更新事务信息

数据库的完整性与安全性控制

➢同样, Oracle数据字典中也定义了三个视图USER_TAB_Columns, ALL_TAB_Columns(Accessible_Columns), 和DBA_TAB_Columns供DBA和用户使用数据字典中关于表的列的信息

ALL_TAB_Columns
DBA_TAB_Columns
User_TAB_Columns:没有 Owner 列

列名	描述
Owner	表的所有者
Table_Name	表名
Column_Name	列名
Data_Type	列的数据类型
Data_Length	列的数据长度，以字节为单位
(其他列...)	其他属性，空值、缺省值等

➢ 可以使用SQL语句来访问这些表中的信息：

```
Select  Column_Name  From  ALL_TAB_Columns
Where  Table_Name  = 'STUDENT' ;
```

数据库的完整性与安全性控制

- Oracle数据字典中还定义了其他视图
  - TABLE_PRIVILEDGE(或ALL_TAB_GRANTS)
  - COLUMN_PRIVILEDGE(或ALL_COL_GRANTS)

    可访问表的权限，列的权限
  - CONSTRAINT_DEFS(或ALL_CONSTRAINTS)

    可访问表的各种约束
  - 还有其他视图… …
- 可以使用下述命令获取Oracle定义的所有视图信息

```
Select view_name from all_views where owner = 'SYS' and
view_name like 'ALL_%' or view_name like 'USER_%';
```

- 如果用户使用Oracle,可使用其提供的SQL*PLUS进行交互式访问

数据库的完整性与安全性控制

---- 本章小结……

- 充分了解了DB完整性的有关概念, 充分了解了SQL-DDL关于完整性约束的定义方法
  - 包括列约束和表约束; 结构约束和内容约束; 静态约束和动态约束; 触发器等
  - 要进一步理解SQL-DDL的使用方法, 以便能更有效地维护数据库
  - 进一步理解Triggers, 有优点，也有不足
- 充分了解了DB安全性的有关概念, 充分了解了SQL-DCL中关于授权与收回授权的使用方法, 进一步理解了视图作为安全性控制的重要意义
  - 包括安全性管理的责任素质要求, 自主安全性机制(访问权利、存取矩阵、授权), 强制安全性机制
  - 进一步理解安全性，掌握DBA应掌握的安全性常识与能力
- 基本了解了数据字典，初步了解DBA通过SQL命令访问数据字典的方法