

# 数据库系统之二

## --数据库语言





# 第8讲 SQL语言与数据库完整性和安全性

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



# 本讲学习什么？



## 基本内容

1. 数据库完整性的概念及分类
2. SQL语言之列约束与表约束—静态约束
3. SQL语言之触发器—动态约束
4. 数据库安全性的概念及分类
5. SQL语言之安全性实现

## 重点与难点

- 数据库完整性的概念，完整性规则，静态约束，动态约束(触发器)
- 数据库安全性的概念，安全性访问规则，权利与授权



# 数据库完整性的概念及分类

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



# 数据库完整性的概念及分类

## (1)什么是数据库完整性?



### 数据库完整性

□数据库完整性(DB Integrity)是指DBMS应保证的DB的一种特性--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。(本讲专指语义完整性)

### 完整性涉及到的...

➤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/IDEF1X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？



# 数据库完整性的概念及分类

## (2)为什么会产生完整性问题？

### 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

❑ 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。

DBMS应该怎样  
保证完整性呢？

是让每个人时刻都  
担心火警，还是把  
火警消除呢？



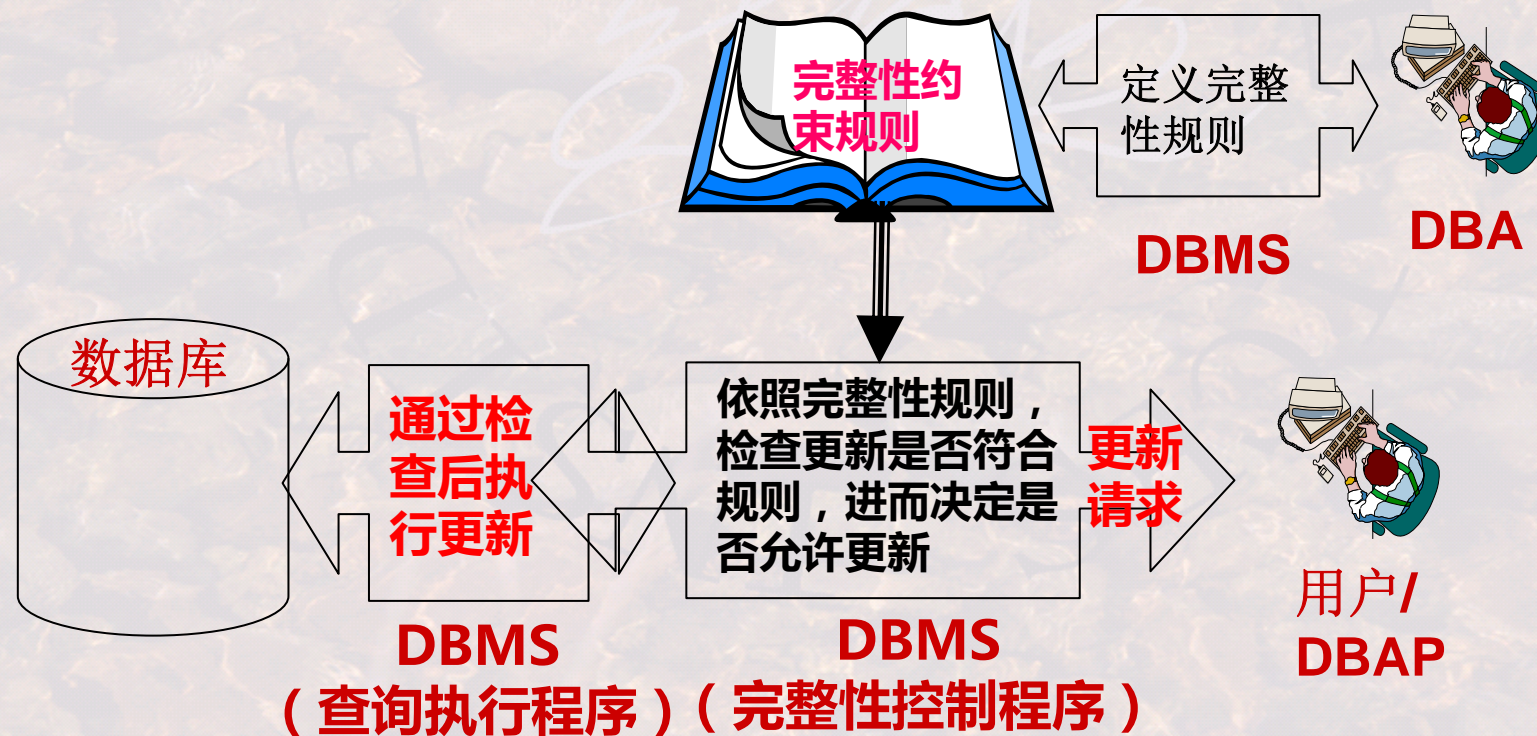


# 数据库完整性的概念及分类

## (3)怎样保证数据库完整性

### 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





# 数据库完整性的概念及分类

## (3)怎样保证数据库完整性

### 完整性约束条件(或称完整性约束规则)的一般形式

□ 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?





# 数据库完整性的概念及分类

## (4)数据库完整性的分类



### 按约束对象分类

#### □ 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

**Sage>15 and Sage<=40**

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

**6<=Chours/Credit<=7**

Course				
C#	Cname	Chours	Credit	T#
001	数据库	40	6	001
003	数据结构	40	6	003
004	编译原理	40	6	001
005	C 语言	30	4.5	003
002	高等数学	80	12	004



# 数据库完整性的概念及分类

## (4)数据库完整性的分类



### 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	

Sage>15 and Sage<=40

S#不允许有重复

S#不允许有空值

D#必须在Dept表中存在



# 数据库完整性的概念及分类

## (4)数据库完整性的分类



### 按约束状态分类

#### □静态约束

✓要求DB在任一时间均应满足的约束；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

SQL语言怎  
样实现这些  
完整性呢？



# 利用SQL语言实现数据库的静态完整性

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



# 利用SQL语言实现数据库的静态完整性

## (1)SQL语言支持的约束类别



### 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





# 利用SQL语言实现数据库的静态完整性

## (2)SQL语言实现约束的方法-Create Table



### Create Table

□ Create Table有三种功能：定义关系模式、定义完整性约束和定义物理存储特性

#### □ 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 }
  ... } );
```



## 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 | SET NULL } ] }

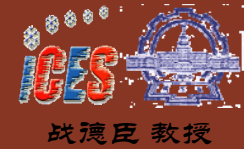
//引用另一表tablename的列colname的值，如有ON DELETE CASCADE 或ON DELETE SET NULL语句，则删除被引用表的某列值v 时，要将本表该列值为v 的记录删除或列值更新为null；缺省为无操作。

➤ Col\_constr列约束：只能应用在单一列上，其后面的约束如UNIQUE, PRIMARY KEY及search\_cond只能是单一列唯一、单一列为主键、和单一列相关



# 利用SQL语言实现数据库的静态完整性

## (2)SQL语言实现约束的方法-Create Table



### 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

Dept		
D#	Dname	Dean
01	机电	李三
02	能源	李四
03	计算机	李五
04	自动控制	李六

```
{ NOT NULL |                                     //列值非空  
[ CONSTRAINT constraintname ]                   //为约束命名, 便于以后撤销  
{ 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 | SET NULL } ] }
```

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



# 利用SQL语言实现数据库的静态完整性

## (2)SQL语言实现约束的方法-Create Table



### 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

Course				
C#	Cname	Chours	Credit	T#
001	数据库	40	6	001
003	数据结构	40	6	003
004	编译原理	40	6	001
005	C 语言	30	4.5	003
002	高等数学	80	12	004

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

```
{ NOT NULL |                                     //列值非空
[ CONSTRAINT constraintname ]                     //为约束命名，便于以后撤销
{ 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 | SET NULL } ] } }
```

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



## 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



# 利用SQL语言实现数据库的静态完整性

## (2)SQL语言实现约束的方法-Create Table



### 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 key(C#),  
constraint ctcc check(Chours/Credit = 20) );
```

//假定严格约束20学时一个学分



# 利用SQL语言实现数据库的静态完整性

## (2)SQL语言实现约束的方法-Create Table



### 示例

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



# 利用SQL语言实现数据库的静态完整性

## (2)SQL语言实现约束的方法-Create Table



➤**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),
```



# 利用SQL语言实现数据库的静态完整性

## (2)SQL语言实现约束的方法-Create Table



➤ **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 ] ;



## 利用SQL语言实现数据库的静态完整性

### (2)SQL语言实现约束的方法-Create Table



示例：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;
```

示例：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

示例：

```
Alter Table SC
```

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



## 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
- 表约束和列约束就是一些特殊的断言
- **SQL**还提供了复杂条件表达的断言。其语法形式为：

**CREATE ASSERTION** <assertion-name> **CHECK** <predicate>

- 当一个断言创建后，系统将检测其有效性，并在每一次更新中测试更新是否违反该断言。

- 断言测试增加了数据库维护的负担，要小心使用复杂的断言。

□ “每位教师不能在同一个学期的同一时间段在两个不同的教室上课”



# 利用SQL语言实现数据库的静态完整性

## (3)SQL语言实现约束的方法-断言



### 示例

“每笔贷款,要求至少一位借款者账户中存有最低数目的余额,例如**1000元**”

```
create assertion balance_constraint check
(not exists (
  select * from loan
  where not exists (
    select * from borrower, depositor, account
    where loan.loan_number = borrower.loan_number
      and borrower.customer_name = depositor.customer_name
      and depositor.account_number = account.account_number
      and account.balance >= 1000)))
```

**borrower(customer\_name, loan\_number,...)** //客户及其贷款(一笔贷款的借款者)  
**account(account\_number,..., balance)** //账户及其余额  
**depositor(account\_number, customer\_name)** //客户及其账户(一个借款者的账户)  
**loan(loan\_number, amount)** //每一笔贷款



# 利用SQL语言实现数据库的静态完整性

## (3)SQL语言实现约束的方法-断言



### 示例

“每个分行的贷款总量必须小于该分行所有账户的余额总和”

```
create assertion sum_constraint check
(not exists (select * from branch
              where (select sum(amount) from loan
                      where loan.branch_name =
                          branch.branch_name )
              >= (select sum (balance) from account
                  where account.branch_name =
                      branch.branch_name )))
```

```
account(branch_name, account_number,..., balance) //分行, 账户及其余额
loan(branch_name , loan_number, amount,) //分行的每一笔贷款
branch(branch_name, ... ) //分行
```



# 利用SQL语言实现数据库的动态完整性

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



# 利用SQL语言实现数据库的动态完整性

## (1)SQL语言支持的约束类别-回顾



### 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

动态约束

(需要定义)

(需要定义)

(需要定义)

(需要定义)

**Integrity Constraint ::= ( O, P, A, R )**



### 触发器Trigger

➤ **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)，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**

➤ **Trigger**是一种过程完整性约束(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。



# 利用SQL语言实现数据库的动态完整性

## (2)实现数据库动态完整的方法—触发器Trigger



### ➤基本语法

```
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。



# 利用SQL语言实现数据库的动态完整性

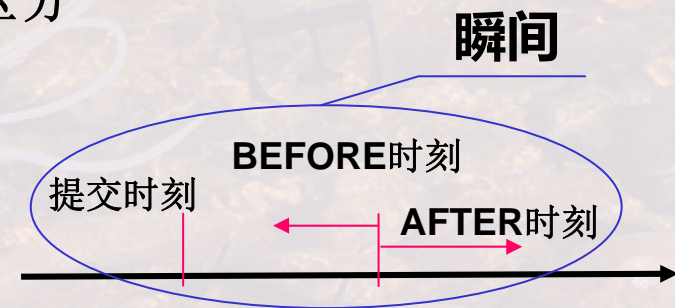
## (2)实现数据库动态完整的方法—触发器Trigger



➤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



```
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 }
```



# 利用SQL语言实现数据库的动态完整性

## (2)实现数据库动态完整的方法—触发器Trigger



### ➤ 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

```
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 }
```



# 利用SQL语言实现数据库的动态完整性

## (3)触发器Trigger示例



战德臣 教授

### 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');
    //此条语句为Oracle的错误处理函数
end;
```

**Integrity Constraint ::= ( O, P, A, R)**



# 利用SQL语言实现数据库的动态完整性

## (3)触发器Trigger示例



### 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。设计一个触发器自动完成上述功能。

```
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;
```

**Integrity Constraint ::= ( O, P, A, R)**



# 利用SQL语言实现数据库的动态完整性

## (3)触发器Trigger示例



### 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;
```

**Integrity Constraint ::= ( O, P, A, R)**



# 利用SQL语言实现数据库的动态完整性

## (3)触发器Trigger示例



战德臣 教授

### 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 sc where S# = :oldi.S# ;
end;
```

Integrity Constraint ::= ( O, P, A, R)



# 利用SQL语言实现数据库的动态完整性

## (3)触发器Trigger示例



### 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;
```

Integrity Constraint ::= ( O, P, A, R)



# 利用SQL语言实现数据库的动态完整性

## (3)触发器Trigger示例



### 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 old oldi, 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;
```

**Integrity Constraint ::= ( O, P, A, R)**



# 回顾本讲学了什么？

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



# 本讲之数据库完整性讲了什么





# 数据库安全性的概念及分类

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



## 数据库安全性

数据库安全性是指**DBMS**应该保证的数据库的一种特性(机制或手段): 免受非法、非授权用户的使用、泄漏、更改或破坏

➤数据库安全管理涉及许多方面

- ❑ 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)
- ❑ 数据的安全级别: 绝密(**Top Secret**), 机密(**Secret**), 可信(**Confidential**)和无分类(**Unclassified**)
- ❑ 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



### 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



### DBA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**账户，该账户是一个超级用户或称系统用户。**DBA**利用该账户的特权可以进行用户账户的创建以及权限授予和撤消、安全级别控制调整等

自主安全性  
如何实现  
呢...?



# 数据库自主安全性机制

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



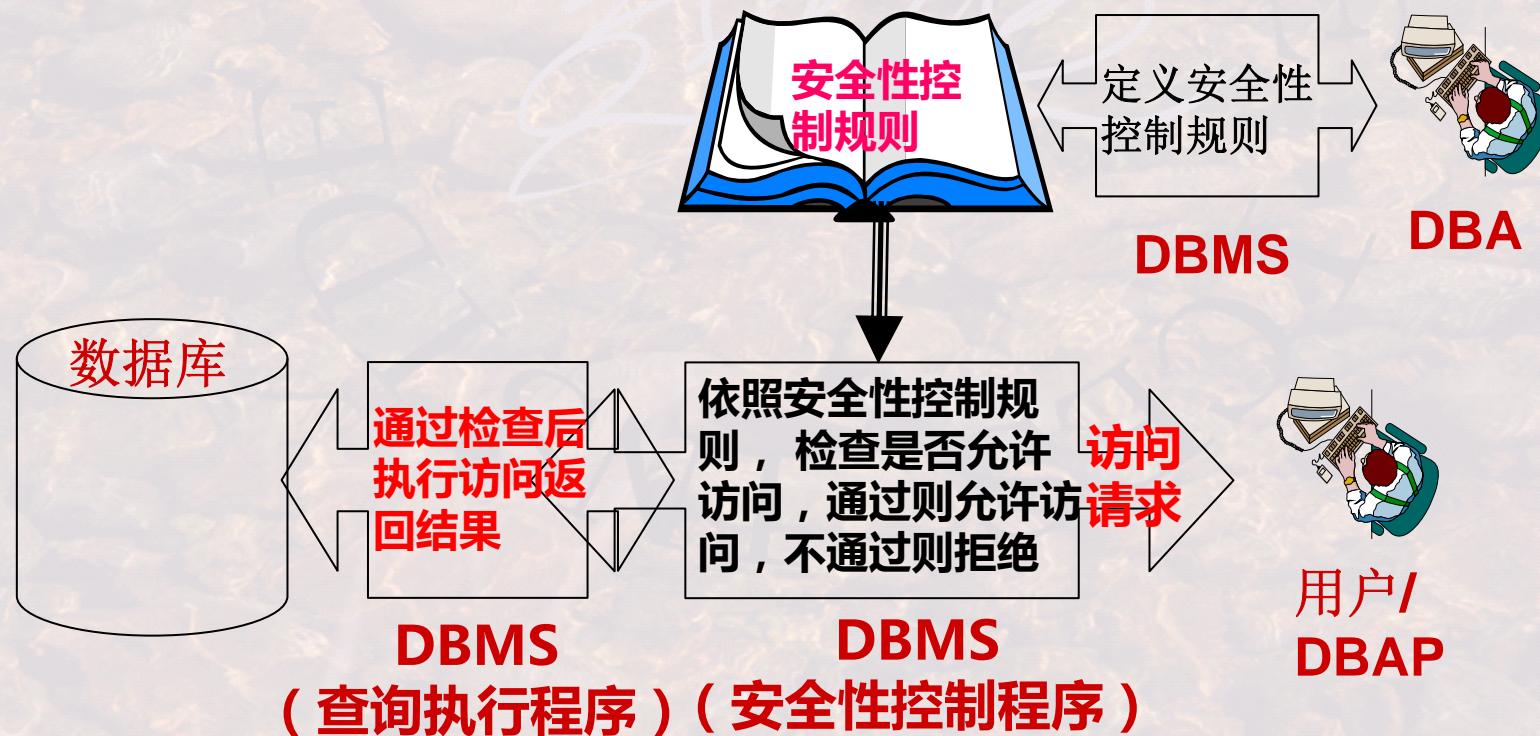
### 自主安全性机制

- 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**处获得一个账户，并由**DBA**授予该账户一定的权限，该账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(账户)，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**



### DBMS怎样自动实现自主安全性呢？

- DBMS允许用户定义一些安全性控制规则(用SQL-DCL来定义)
- 当有DB访问操作时，DBMS自动按照安全性控制规则进行检查，检查通过则允许访问，不通过则不允许访问





➤DBMS将权利和用户(账户)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：包括创建、增、删、改、查等
- 谓词：拥有权利需满足的条件



### 员工管理数据库的安全性控制示例

**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，便于维护员工信息
- **收发人员**：访问该数据库以确认某员工是哪一个部门的，便于收发工作，只能访问基本信息，其他信息不允许其访问
- **每个员工**：允许其访问关于自己的记录，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

这些不同需求如何实现呢...?



### 两种控制示例

- 按名控制安全性：存储矩阵
- 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

这个谓词如何实现呢...?



# 数据库自主安全性机制

## (4)自主安全性的实现方式

### 第1种：存储矩阵

主体 \ 数据对象	O <sub>1</sub>	O <sub>2</sub>	O <sub>3</sub>	...	O <sub>n</sub>
S <sub>1</sub>	All	All	All	....	All
S <sub>2</sub>	Read	—	Read	....	—
...	...	...	...	...	....
S <sub>n</sub>	—	Read	—	...	—

➤ 请同学们给出一个算法：已知如上形式的一个存储矩阵，某用户提交了一条SQL语句(Select-From-Where), 设计一个算法，判断是否允许该用户检索到相应的结果？





## 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

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



# 利用SQL语言实现数据库自主安全性

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



# 利用SQL语言实现数据库自主安全性

## (1)SQL语言的用户与权利



➤SQL语言包含了DDL, DML和DCL。数据库安全性控制是属于DCL范畴

➤授权机制---自主安全性；视图的运用

➤关系级别(普通用户) ← 账户级别(程序员用户) ← 超级用户(DBA)

□ (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**账户级别**的权利，而将级别1和2称为**关系级别**的权利。



### 授权命令

```
GRANT {all PRIVILEGES | privilege {,privilege...}}  
ON [TABLE] tablename | viewname  
TO {public | user-id {, user-id...}}  
[WITH GRANT OPTION];
```

- ❑ **user-id** , 某一个用户账户, 由**DBA**创建的合法账户
- ❑ **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 PRIVILEGES**
- ❑ **WITH GRANT OPTION**选项是允许被授权者传播这些权利



# 利用SQL语言实现数据库自主安全性

## (3)SQL-DCL的控制安全性示例



### 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**UserId**, 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



# 利用SQL语言实现数据库自主安全性

## (3)SQL-DCL的控制安全性示例



### 收回授权命令

```
REVOKE {all priviEges | priv {, priv...} } ON tablename | viewname  
FROM {public | user {, user...} };
```

### 示例

```
revoke select on employee from UserB;
```



# 安全性控制的其他简介

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



### 授权过程

- 第一步：**DBA**创建**DB**, 并为每一个用户创建一个账户

假定建立了五个用户：**UserA, UserB, UserC, UserD, UserE**

- 第二步：**DBA**授予某用户账户级别的权利

假定授予**UserA**

- 第三步：具有账户级别的用户可以创建基本表或视图, 他也自动成为该表或该视图的属主账户, 拥有该表或该视图的所有访问 权利

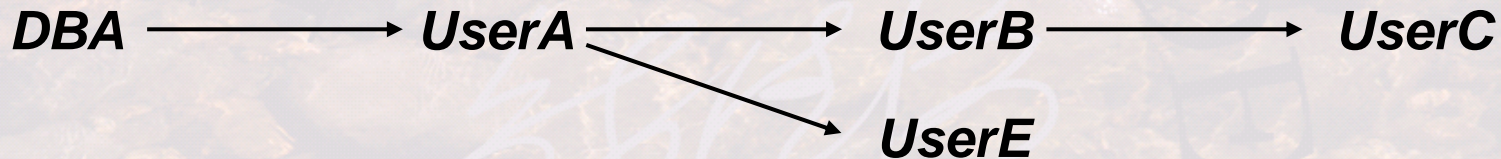
假定**UserA**创建了**Employee**, 则**UserA**就是**Employee**表的属主账户

- 第四步：拥有属主账户的用户可以将其中的一部分权利授予另外的用户, 该用户也可将权利进一步授权给其他的用户...

假定**UserA**将读权限授予**UserB**, 而**UserB**又将其拥有的权限授予**UserC**, 如此将权利不断传递下去。



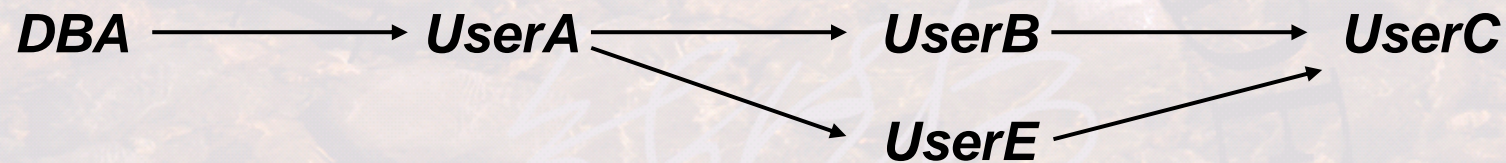
```
Grant Select ON Employee TO UserB WITH GRANT OPTION;  
Grant Select ON Employee TO UserC WITH GRANT OPTION;
```



### 注意授权的传播范围

- 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。





- 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。



## 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**)
- 同时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

- ❑ 用户**S**, 不能读取数据对象**O**, 除非 $\text{Level}(\text{S}) \geq \text{Level}(\text{O})$
- ❑ 用户**S**, 不能写数据对象, 除非 $\text{Level}(\text{S}) \leq \text{Level}(\text{O})$ 。

为什么要这样呢?



## 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:  $R(A_1: D_1, A_2: D_2, \dots, A_n: D_n)$

□ 对属性和元组引入安全性分级特性或称分类特性

$R(A_1: D_1, C_1, A_2: D_2, C_2, \dots, A_n: D_n, C_n, TC)$

其中  $C_1, C_2, \dots, C_n$  分别为属性  $D_1, D_2, \dots, D_n$  的安全分类特性;  $TC$  为元组的分类特性



## 安全型控制的其他简介

### (2)强制安全性

➤ 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。
- 关于强制安全性机制的内容, 可参看其他有关的文献。



# 回顾本讲学了什么？

战德臣

哈尔滨工业大学 教授·博士生导师

黑龙江省教学名师

教育部大学计算机课程教学指导委员会委员

Research Center on **I**ntelligent  
**C**omputing for **E**nterprises & **S**ervices,  
**H**arbin **I**nstitute of **T**echnology



# 本讲之数据库安全性讲了什么

