

课程名称: 数据库系统

高级SQL

单 位: 重庆大学计算机学院

系统角色的区分

- 在使用学生信息管理系统，学生能做什么操作？教师能做什么操作？有区别吗？
- SQL语言能否提供相应的支持帮助实现以上功能？

主要目标

- 复杂形式的SQL查询
- 视图与事务的定义和使用
- 完整性约束、SQL数据类型和模式
- 授权的定义和使用
- 触发器的使用

思考问题

- 在使用操作系统时，如何进行安全控制？
- 又是如何进行用户管理？不同类型的用户的操作权限是否相同？

一 视图

什么是视图，视图上还可定义新视图吗？

视图有何作用？

视图采用**create view**语句定义，
可以是任何一个**SQL**语句。
无实际数据的'**虚表**'，有利于数据一致性！
视图上可以在定义新的视图！

* (讲解)可以通过视图更新数据吗?

案例1

```
create view faculty as  
select ID, name, dept_name  
from instructor ;      (P.68)
```

create view *physics_fall_2009* as (P.68)
 select *course.course_id, sec_id, building, room_number*
 from *course, sec_id*
 where *course.course_id = sec_id.course_id*
 and *course.course_id = sec_id.course_id*
 and *sec_id.course_id = sec_id.course_id*
 and *sec_id.course_id = sec_id.course_id*

view level 视图

view 1 view 2 ... view n

1级映射

where *building* = 'Watson'; (P.68)

- 1) 可以在任何**SQL**语句中像表一样的被使用!
- 2) 增强查询能力且方便(用户/程序员)使用!
- 3) 还可以提供数据访问安全控制(隐藏数据)!
- 4) 作为外模式(**1级映射**)有利于应用独立性!

仅在少数简单视图(“行列”视图)上可以更新数据! P.70

二 完整性约束

2.1 完整性约束

SQL提供哪些方式的数据完整性约束?

1. 键完整性约束 (主码/主键)

关系(模式)必需有一个主码, 来区分不同元组!
SQL采用 **primary key** ... 来定义!

案例2.a

2. 参照完整性约束 (外码/外键)

用另一关系的主码, 来约束属性取值的有效性!
SQL采用 **foreign key** ... **references** ... 来定义!

案例2.b

3. 其它数据完整性约束:

属性(非空)完整性约束

属性(唯一)完整性约束

属性(范围)完整性约束

键完整性约束

参照完整性约束

create table instructor

(**ID** **varchar** (5),
→ **name** **varchar** (20) **not null** **unique**,
→ **dept_name** **varchar** (20),
→ **salary** **numeric** (8,2) **Check** (salary>10000),
→ **primary key** (ID),
→ **foreign key** (dept_name) references department);

案例2.c

create table section (

course_id **varchar** (8),
sec_id **varchar** (8),
semester **varchar** (6),
year **numeric** (4,0),
building **varchar** (15),
room_number **varchar** (7),
time_slot_id **varchar** (4),
primary key (course_id, sec_id, semester, year),
→ **check** (semester in ('Fall', 'Winter', 'Spring', 'Summer'))
→ **check** (time_slot_id in (select time_slot_id from time_slot));

请解释各标注处数据完整性约束的作用?

(注: 复杂条件甚至可以是表上多个属性间表达式的限制约束)

复杂条件(标量集合限定取值范围)完整性约束

复杂条件(来自他表select结果限定取值范围)完整性约束

2.2 外键约束方式

三种参照约束方式:

案例3

```
create table course (
...
dept_name varchar(20),
foreign key (dept_name) references department
on delete cascade
on update cascade,
...);
```

(P.73-74)

1)不写时拒绝删除
---2)连带删除(修改)
---3)设置为空/默认值

这里的on约束
起到什么作用
?

*(讲解)什么是
断言,有什么
作用?

当数据更新时,
保持谓词为真。
(否则拒绝更新)
但作用有利有弊

2.3 断言

例子: 约束要求: **student**每个元组的**tot_cred**(学生的总学分)取值应等于该生所修完课程的学分总和(关系**takes**∞**course**的**credits**)
create assertion <assertion-name> **check** <predicate>;

```
create assertion credits_earned_constraint check
(not exists (select ID
from student
where tot_cred < > (select sum(credits)
from takes natural join course
where student.ID = takes.ID
and grade is not null and grade < > 'F' ));
```

案例4 一个断言的例子

三 授权

3.1 表(关系)上的授权

SQL如何限制用户对表中数据的合法访问?

通过授权!

只有授权用户才能查看(/插入/修改/删除)相关表中的数据。

注:表的创建者,自然拥有表上的一切权限。

这些语句说明了什么访问权限变化?

grant select on instructor to U_1, U_2, U_3 ;

**insert
update
delete**

all privileges

public ?

所有用户

将instructor表上的查看(插入/...)权授予用户。

revoke select on branch from U_1, U_2, U_3 ;

...

all

用户在branch表上的查看(...)权被收回。

作用及好处? 简化权限管理

create role instructor; 角色名

grant instructor to Amit; 用户名

可以建立角色roll(用户群)。当将某权限授予角色时,该用户群均有该使用权限。

grant select on takes to instructor;

create role teaching_assistant;

grant teaching_assistant to instructor;

还可以创建子角色

● **grant select on department to Amit with grant option;**

● **revoke select on department from Amit, Satoshi cascade;**

● **revoke select on department from Amit, Satoshi restrict;**

允许转授权限

级联回收权限(默认值)

防止级联回收权限

3.2 视图上的授权

三 授权

案例6

视图上也可以授权吗(查看/修改/删除数据)?

```
create view geo_instructor as  
(select *  
from instructor  
where dept_name='Geology');
```

在表*instructor*上创建一个视图*geo_instructor*

```
grant select on geo_instructor to geo_staff  
(roll角色)
```

将视图上的查看权授予一个角色*geo_staff*

*geo_staff*中用户能通过该语句查看到数据吗?

```
select *  
from geo_instructor;
```

如果该用户在*instructor*上没有获得select授权, 则他仍然看不到数据!
注: 视图上的update权限也类似

用户可以定义函数与过程(p.83,&5.2), 并可对其他用户授予**execute**执行权。

注: 1) 视图的创建者, 自然拥有该视图上的所有权限!
2) 函数与过程的创建者, 自然拥有其上所有权限!
3) 从SQL2003标准开始, 允许在定义函数和过程时指定**sql security invoker**, 执行时与调用者相同的权限下运行!

*(讲解) 授权图
如何形成, 有何
作用?

变化示例1:

U_2 执行语句:

grant update on teaches To U_6 ;

变化示例2:

U_1 执行语句:

revoke update on teaches from U_5 ;

DBA 执行语句:

revoke update on teaches from U_2 ;

作用:

- 1) 描述在一张表上某种授权的当前状态, 便于系统动态管理授权;
- 2) 当DBA或具有权限的用户(树上节点)进行授权时, 树扩展(生长);
- 3) 当DBA或具有权限的用户(树上节点)回收权限时, 树收缩(枯萎);

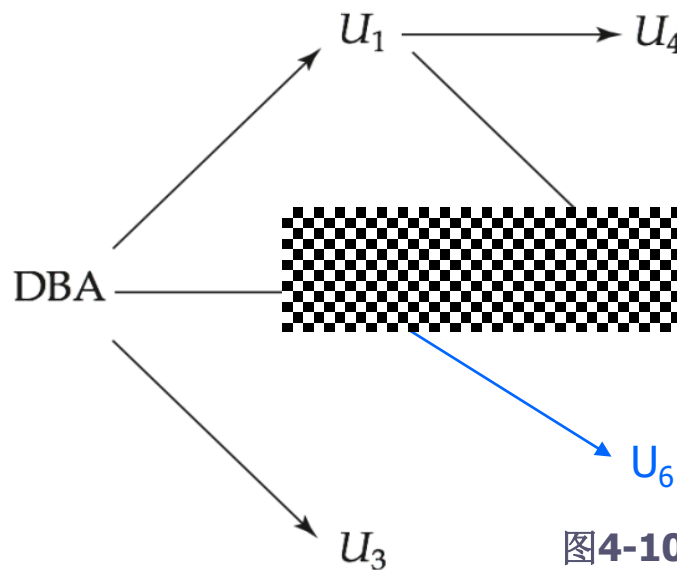


图4-10

案例7: 表teaches上的update更新权

四 SQL存储过程

SQL除了提供一些常用的内建函数(聚集、日期、字符串转换等)外,可编写存储过程(业务逻辑)并存于库中,可在SQL/应用代码中调用!

4.1 SQL函数

SQL如何定义和使用函数与过程?

案例8.b

作用:
给定系
名返回
系人数

案例8.a

定义(声明)一个函数:有一个输入参数,有一个输出参数(返回值),begin-end为函数体。

```
create function dept_count (dept_name varchar(20))
returns integer
begin
    declare d_count integer;
    select count (*) into d_count
    from instructor
    where instructor.dept_name = dept_name
    return d_count;
end
```

```
create function instructors_of (dept_name char(20))
returns table ( ID varchar(5),
               name varchar(20),
               dept_name varchar(20),
               salary numeric(8,2))
```

```
return table
(select ID, name, dept_name, salary
 from instructor
 where instructor.dept_name = instructors_of.dept_name)
```

```
select *
from table (instructors_of ('Music'))
```

使用该函数的参数↓

```
select dept_name, budget
from department
where dept_count (dept_name ) > 1
```

SQL查询语句中,可以使用用户定义的函数,就像使用系统固有函数一样。

定义的函数甚至可返回一个表,需仔细描述返回值(各属性)类型。这里函数体仅一个SQL语句,故函数体不需begin-end界定。

函数返回表可以用在SQL查询中允许表出现的位置!

4.2 SQL过程

四 SQL存储过程

SQL函数与
SQL过程有何
不同?

过程：一段SQL语句程序；
函数：有处理还有返回值

该SQL过程的
作用及使用方
法?

案例9.b

```
declare n integer default 0;
while n < 10 do
    set n = n + 1
end while
```

```
declare n integer default 0;
for r as
    select budget from department
    where dept_name = 'Music'
do
    set n = n - r.budget
end for
```

案例9.a 该过程的作用与前面的dept_count函数类似!

```
create procedure dept_count_proc (
    in dept_name varchar(20), out d_count integer)
begin
    select count(*) into d_count
    from instructor
    where instructor.dept_name =
        dept_count_proc.dept_name
end
```

```
declare d_count integer;
call dept_count_proc( 'Physics', d_count);
```

```
If n < 0
then
    return(-1)
else
    then return(1)
endif
```

```
repeat
    set n = n
    - 1
until n = 0
end repeat
```

begin ... end

存储过程可在SQL过程中或
嵌入式SQL中通过call命令
调用，还可在动态SQL中用

编写存储过程可以
使用这些常规控制
语句吗?

可以使用!

4.3 存储过程示例*

*(讲解)该SQL函数
是如何完成课程注册
业务逻辑?

统计课程人数,暂存于 *currEnrol*

查看课程人数限制,暂存于 *limit*

满足人数限制时,注册一个学生
(且注册成功时,返回0)

否则,输出错误信息到 *errorMsg*
(且注册不成功时,返回-1)

——在确保教室能容纳下的前提下注册一个学生
——如果成功注册,返回0,如果超过教室容量则返回-1

```
create function registerStudent(  
    in s_id varchar(5),  
    in s_courseid varchar(8),  
    in s_secid varchar(8),  
    in s_semester varchar(6),  
    in s_year numeric(4,0),  
    out errorMsg varchar(100)
```

输入课程信息参数

输出错误参数

returns integer

begin

declare currEnrol int;

select count(*) into currEnrol

from takes

where course_id = s_courseid and sec_id = s_secid

and semester = s_semester and year = s_year;

declare limit int;

select capacity into limit

from classroom natural join section

where course_id = s_courseid and sec_id = s_secid

and semester = s_semester and year = s_year;

if (currEnrol < limit)

begin

insert into takes values

(s_id, s_courseid, s_secid, s_semester, s_year, null);

return(0);

end

——否则,已经达到课程容量上限

set errorMsg = 'Enrollment limit reached for course ' || s_courseid
|| ' section ' || s_secid;

return(-1);

end;

图 5-7 学生注册课程的过程

*(讲解) 什么是外部语言过程, 有何利弊?

```
create procedure dept_count_proc(  
    in dept_name varchar(20), out count integer)  
language C  
external name '/usr/avi/bin/dept_count_proc'
```

```
create function dept_count(dept_name varchar(20))  
returns integer  
language C  
external name '/usr/avi/bin/dept_count'
```

外部语言过程: **SQL**允许用程序语言(**Java,C#,C,C++**)来定义函数或过程, 运行效率要高于**SQL**中定义的函数, 用于完成无法在**SQL**中执行的计算。

例如:

Oracle, IBM DB2: 允许Java编写的函数;

MS SQL Server: 允许用C#, VB编写的过程;

PostgreSQL: 允许Python, Perl, Tcl定义的函数。

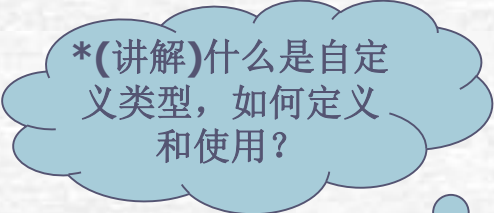
六 *(略讲)用户自定义的类型

UDT (User-Defined Types) & Domain

- 定义一个新类型:

```
create type Dollars as numeric (12,2) final;
```

```
create table department  
(dept_name varchar (20),  
building varchar (15),  
budget Dollars);
```



*(讲解)什么是自定义类型，如何定义和使用？

- 定义一个新域:

```
create domain person_name char(20) not null;
```

```
create domain degree_level varchar(10)
```

```
constraint degree_level_test
```

```
check (value in ( ' Bachelors' , ' Masters' , ' Doctorate' ));
```

- 域与类型的区别:

- 1) 域上允许声明(定义)约束;

- 2) 域不是强制类型，一个域类型可以被赋予另一个域类型，只有它们的基本类型是相容的（可强制转换）；

五 触发器

* 触发器包含哪些部分及其含义？

begin-end可有多个语句
atomic-表示为一个事务

将根据takes的课程成绩记录，
重新计算相应学生的总学分数。

```
create trigger timeslot_check1 after insert on section
referencing new row as nrow
for each row
when ( nrow.time_slot_id not in (
    select time_slot_id
    from time_slot ) ) /* time_slot 中不存在该 time_slot_id */
begin
    rollback
end;
```

撤销新插入的记录！

```
create trigger timeslot_check2 after delete on time_slot
referencing old row as orow
for each row
when ( orow.time_slot_id not in (
    select time_slot_id
    from time_slot ) /* 在 time_slot 中刚刚被删除的 time_slot_id */
and orow.time_slot_id in (
    select time_slot_id
    from section ) ) /* 在 section 中仍含有该 time_slot_id 的引用 */
begin
    rollback
end;
```

撤销删除的记录！

当section中还有参照记录存在时，
拒绝删除time_slot中被参照记录。

```
create trigger credits_earned after update of takes on ( grade )
{
    referencing new row as nrow
    referencing old row as orow
    for each row
    {
        when nrow.grade < > 'F' and nrow.grade is not null
            and ( orow.grade = 'F' or orow.grade is null )
        begin atomic
            update student
            set tot_cred = tot_cred +
                ( select credits
                  from course
                  where course.course_id = nrow.course_id )
            where student.id = nrow.id;
        end;
    }
}
```

当属性值改变时
将对改变前后的记录比较

有新记录成绩且非F
旧记录成绩为F或空

案例10.a 使用触发器来维护 credits_earned 值
(student表的属性)

* 这些触发器
起到什么作用？

注:触发器还可用于
复制或备份数据库!

本节小结

- 视图
- 完整性约束
- 授权
- 函数与存储过程
- 触发器

课后作业安排

- 作业

- 1) 复习：授权图知识(第4章， p. 84， 4. 6. 5节)；

- 2) 第4章(p. 87)： 4. 12, 4. 14, 4, 17;

- 预习

- 第10讲-关系模式设计优化（[课前预习资料](#)）