3 SQL

Author： 中山大学 17数据科学与计算机学院 YSY

https://github.com/ysyisyourbrother

数据类型

SQL标准支持多种类型，包括：

- char(n) 固定长度字符串，如果保存的长度没到n，会追加空格。
- varchar(n)可变长度字符串，用户指定最大长度n，不追加空格。
- int 整数类型
- smallint 小整数类型
- numeric(p,d)定点数，精度由用户指定。这个数有p位（二进制中包含符号位），其中d位数字在小数点右边，numeric(3,1)可以精确储存44.5，但不能保存444.5
- real，double precision 浮点数和双精度浮点数，精度与机器相关
- float(n)精度至少为n位的浮点数

基本模式定义

创建表

用create table命令定义SQL关系：

```
create table department
    (dept_name varchar (20),
    building varchar (15),
    budget numeric (12, 2),
    primary key (dept_name));
```

完整性约束：

1. primary key**实体完整性**： 非空且唯一
2. forrign key**参照完整性**：foreign key(dept_name) reference department 表示属性dept_name必然存在于department关系的主键里。 外键可以为空值。
3. not null **用户自定义完整性**：在该属性不允许空值。
4. check **用户自定义完整性**： 数据要满足check条件，后面加的check不会检查原来表的字段
5. 触发器 **用户自定义完整性**：

```
create table department
    (dept_name        varchar(20),
    building        varchar (15),
    budget        numeric (12,2),
    primary key (dept_name));
create table course
    (course_id        varchar (7),
    title        varchar (50),
    dept_name        varchar (20),
    credits        numeric (2,0),
    primary key (course_id),
    foreign key (dept_name) references department);
create table instructor
    (ID        varchar (5),
    name        varchar (20) not null,
    dept_name        varchar (20),
    salary        numeric (8,2),
    primary key (ID),
    foreign key (dept_name) references department);
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),
    foreign key (course_id) references course);
create table teaches
    (ID        varchar (5),
    course_id        varchar (8),
    sec_id        varchar (8),
    semester        varchar (6),
    year        numeric (4,0),
    primary key (ID, course_id, sec_id, semester, year),
    foreign key (course_id, sec_id, semester, year)
references section,
    foreign key (ID) references instructor);
```

创建外键也可以在属性后面加：

`sno char(5) REFERENCES Stu_Union(sno) on delete cascade` 设置了级联删除

`on delete set null` 在删除参照表中被引用值后，引用处变成null

`on delete no action` 不允许被引用表的主键update或delete操作

删除表

`delete from student` 将关系表中所有元组删除

`drop table student` 将整个关系删除

添加属性

`alter table r add A D` 选择关系r，添加属性A，D是属性的域

`alter table drop A` 从关系中去掉属性A

SQL基本查询结构

单表查询：

`select * from instructor` 选取所有instructor名字

`select distinct name from instructor` 去除重复导师名字

`select all name from instructor` 显式指明不去处重复

`select ID,name,salary*1.1 from instructor` select子句还可以带有运算符的算数表达式

多表查询：

可以将多个关系放在from后面，用where指定匹配条件：

```
select name, instructor.dept_name, building
from instructor, department
where instructor.dept_name = department.dept_name;
```

相当于对instructor 和department表进行了**笛卡尔积**

例如，关系 *instructor* 和 *teaches* 的笛卡儿积的关系模式为：

(*instructor.ID*, *instructor.name*, *instructor.dept_name*, *instructor.salary*
teaches.ID, *teaches.course_id*, *teaches.sec_id*, *teaches.semester*, *teaches.year*)

然后用where中的谓词筛选出满足条件元组

最后用select子句指定要输出的属性。

where子句谓词

`select name from Instructor where salary<>7000`在where子句中使用逻辑连词and or not 和比较运算符号。

可以用**(not) between and**来说明一个值在两个去值之间：

```
select name
from instructor
where salary between 90000 and 100000;
```

可以用(v1,v2,v3...)表示一个元组，使用比较运算符。例如(a1,a2)<=(b1,b2) 在a1<=b1 且a2<=b2的时候为真。

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

排序：

用order by 子句将结果排序 **desc表示降序 asc表示升序。**如果同时对多个排序，就先排前面一样再排后面：

```
select *
from instructor
order by salary desc, name asc;
```

自然连接：

笛卡尔积：将第一个关系的每个元组和第二个关系的所有元组连接

自然连接：只结合两个关系上中都出现的**所有属性上**取值相同的元组对

比如查询：

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID;
```

可以简洁的写成：

```
select name, course_id
from instructor natural join teaches;
```

在相同的属性ID上，只有值相同才会连接。

在一个SQL的from中，可以用自然连接将多个关系结合在一起：

```
select A1, A2, ···, An
from r1 natural join r2 natural join ···natural join rm
where P;
```

更一般的，from子句可以为如下形式：

```
from E1, E2, ···, En
```

其中每个E都是一个关系，或者包含自然连接的表达式。加入要列出教师名字和他们教授的课程：

```
select name, title
from instructor natural join teaches, course
where teaches.course_id = course.course_id;
```

先计算instructor和teaches在teacher_id上的自然连接，然后和course做笛卡尔积，再用where谓词选取。

如果用下面的式子就无法计算出相同结果：

```
select name, title
from instructor natural join teaches natural join course;
```

因为instructor和teachers的连接结果中，和course关系有dept_name和course_id两个共同属性，需要都相同才会连接。因此结果不同。

可以用join … using 运算来指定需要连接的属性名列表，允许用户来指定需要那些列相等：

```
select name, title
from (instructor natural join teaches) join course using (course_id);
```

连接条件

on

on条件允许在参与连接的关系上设置通用谓词。

```
select *
from student join takes on student. ID = takes. ID;
```

用on条件丽娜姐可以比一般的自然连接更丰富。不过使用on条件的连接表达式都可以用不带on条件的等价表达式替换，**只要把on中的谓词移到where中即可**

外连接

要查看所有学生的选课情况，用下面的方法：

```
select *
from student natural join takes;
```

但上述查询和想要的结果不一样，**如果有学生没有选任何课程**，就不会在结果中展示。这种连接就是

外连接会在结果中创建包含空值的元组，保留在连接丢失的元组。比如上面如果有左外连接，原本没有的学生元组也会被加入，来自takes中的属性被设为null

左外连接left outer join：保留运算左边的关系中的元组

右外连接right outer join：保留运算右边的关系中的元组

全外连接full outer join：两边都保存

内连接inner join：自然连接这种不保留任何不匹配元组的（一般不显式声明就默认）

`natural left outer join`

`left outer join on ...`

我们可以用他们来找到一门课都没上的学生：

```
select ID
from student natural left outer join takes
where course_id is null;
```

基本运算

更名运算

用as来选去的属性重命名，既可以用在select也可以用在from

```
select name as instructor_name, course_id
from instructor, teaches
where instructor. ID = teaches. ID;
```

用在from子句中，可以将别名用在别的地方：

```
select T. name, S. course_id
from instructor as T, teaches as S
where T. ID = S. ID;
```

字符串运算

upper(s): 字符串转大写

lower(s): 字符串转小写

trim(s): 去掉字符串后面的空格

like运算符:

- 百分号% : 匹配任意字串 比如: `%Comp%` 匹配包含Comp的。要表示%用转译字符 `\%`
- 下划线 _ : 匹配一个字符比如: `___` 匹配三个字符的 `___%` 匹配至少三个字符的。

  sql中用比较运算符like来匹配字符串:

```
select dept_name
from department
where building like '% Watson% ';
```

集合运算

并运算（union）

```
(select course_id
 from section
 where semester = 'Fall' and year = 2009)
 union
(select course_id
 from section
 where semester = 'Spring' and year = 2010);
```

union自动去除重复，要保留重复，用`union all`

交运算（intersect）

```
(select course_id
 from section
 where semester = 'Fall' and year = 2009)
intersect
(select course_id
 from section
 where semester = 'Spring' and year = 2010);
```

intersect自动去除重复，要保留重复，用`intersect all`

差运算（except）

```
(select course_id
 from section
 where semester = 'Fall' and year = 2009)
except
(select course_id
 from section
 where semester = 'Spring' and year = 2010);
```

except运算从第一个输入中输出所有不在第二个输入中的元组。自动去除重复，要保留用`except all`

集合比较：

some：至少比一个

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = 'Biology');
```

=some 等价于in <>some不等价于not in

all：比所有的都

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```

<>all 等价于not in 但 =all不等于in

当all后面的子查询查询结果为空集时，一定返回为true

空值

NULL用法：

- **算数表达式任一输入为空**，结果为空。比如r.A+5，若A是空，结果为空
- **和空值的比较运算**，因为说真或者假都无意义，因此结果为**unknown**，这是除了True False之外的第三个逻辑值
- 布尔运算被拓展到了unknown上：
  - **and**：*true* **and** *unknown* 的结果是 *unknown*，*false* **and** *unknown* 结果是 *false*，*unknown* **and** *unknown* 的结果是 *unknown*。
  - **or**：*true* **or** *unknown* 的结果是 *true*，*false* **or** *unknown* 结果是 *unknown*，*unknown* **or** *unknown* 结果是 *unknown*。
  - **not**：**not** *unknown* 的结果是 *unknown*。

  只有**true or unknown**才有可能正确，其他都是错误的

  如果where子句对一个元组计算出false和unknown，都不能加入到结果。
- 用`is null`和`is not null`来判断是否为空值
- 用`is unknown`和`is not unknown`判断结果是否为unknown
- 使用distinct去除重复时，对应列同为null的也会被认为是重复被去除。但比较两个null是否相等的结果是unkonwn

聚集

基本聚集函数

avg

找出CS教师的平均工资，用as可以给聚集后的属性一个有意义的名字。

默认是保留重复，如果要去除重复可以加distinct

```
select avg (salary) as avg_salary
from instructor
where dept_name = 'Comp. Sci.';
```

count

使用聚集函数count计算一个关系中元组的个数。可以用count(*)

```
select count (distinct ID)
from teaches
where semester = 'Spring' and year = 2010;
```

SQL不允许在count(*)时使用distinct

max和min

选出指定属性值最大或最小的元组

分组聚集

group by

使用group by可以把属性上取值相同的元组作为一个分组，然后再用基本聚集函数对每个分组内的元组处理。如果不加group by相当于整个关系当作一个分组。

比如我们要找出每个系的平均工资：

```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name;
```

1. 分组时计算查询的第一步，将dept_name相同的元组分组聚集。
2. 接着在每个分组上进行指定的聚集计算。

dept_name	avg_salary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

没有被聚集的属性，必须出现在group by子句中，否则查询是错误的。

having子句

我们可以对分组限定条件，比如我们要选出平均工资超过42000的系：

```
select dept_name, avg (salary) as avg_salary
from instructor
group by dept_name
having avg (salary) > 42000;
```

dept_name	avg_salary
Physics	91000
Elec. Eng.	80000
Finance	85000
Comp. Sci.	77333
Biology	72000
History	61000

出现在having中的属性，但没出现在聚集函数里，就要出现在group by中

1. 首先根据from子句计算出一个关系
2. 如果有where，就把where子句的谓词作用于from的结果
3. 如果有group by ，形成分组，否则整个当作一个分组
4. 如果出现having，应用在每个分组上，筛选出满足having的分组
5. select选取查询结果，每个分组用聚集函数的展示单个元组

对空值和布尔值聚集

对salary可能为null的关系，使用下面查询：

```
select sum(salary)
from instructor;
```

输出的总和不是null，而是**在计算sum的时候忽略null值**

除了count(*)外所有聚集函数都忽略null。

如果因为输入都是null导致输入为空集，count会输出0，其他输出null

嵌套查询

相关子查询：

外层查询的名称可以用在where子句的子查询中。

空关系测试：

exists：在子查询非空的时候返回True

```
select course_id
from section as S
where semester = 'Fall' and year = 2009 and
    exists (select *

from section as T
where semester = 'Spring' and year = 2010 and
    S. course_id = T. course_id);
```

重复元组存在性测试

unique：当子查询没有**重复**元组的时候，返回true

比如要查询"找出所有在2009年最多开设一次的课程"：

```
select T. course_id
from course as T
where unique ( select R. course_id
               from section as R
               where T. course_id = R. course_id and
                     R. year = 2009 ) ;
```

一种不使用unique的等价方法是：

```
select T. course_id
from course as T
where 1 >= ( select count( R. course_id )
             from section as R
             where T. course_id = R. course_id and
                   R. year = 2009 ) ;
```

from子句中的嵌套查询

from中是对关系操作的，可以用子句来代替被操作的关系

我们可以给**子查询的结果关系命名，并对属性重命名**

```
select dept_name, avg_salary
from ( select dept_name, avg ( salary )
       from instructor
       group by dept_name )
       as dept_avg ( dept_name, avg_salary )
where avg_salary > 42000 ;
```

with子句

with子句定义临时的方法，只对包含with子句的查询有效：

```
with max_budget ( value ) as
     ( select max( budget )
       from department )
select budget
from department, max_budget
where department. budget = max_budget. value ;
```

这里定义了一个临时关系max_budget。

数据库的修改

删除

delete

`delete from r where p` 从关系r中删除满足p的元组

`delete from r` 将关系r的元组全部删除

插入

insert

`insert into r values(...)` 向关系r中插入记录，值为...

这种方法要求values中的属性排序和 r 中相同，另一种在insert语句中指定属性的方式是：

```
insert into course (course_id, title, dept_name, credits)
    values ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
insert into course (title, course_id, credits, dept_name)
    values ('Database Systems', 'CS-437', 4, 'Comp. Sci.');
```

这两种插入方法是一样的。如果只插入了部分属性，其他属性用null表示。

select into

`select * into result from students` select into 语句可以实现同时创建表和插入，从students中选出所有记录和属性，插入到result表中

更新

update

`update r set a=... where p` 将关系r中满足p的记录的a属性更新

case

SQL提供了case结构，可以在一条update中执行**多种不同条件的更新**，比如：

```
update instructor
set salary = case
        when salary <= 100000 then salary * 1.05
        else salary * 1.03
    end
```

case语句格式一般如下：

```
case
    when pred1 then result1
    when pred2 then result2
    ...
    when predn then resultn
    else result0
end
```

视图

附录

实验课学习内容：

```
--- 第二讲
create table CUSTOMERS(
    CID char(10) not null Unique,
    ...,
    primary key(CID),
    check(page>18)
)

create table orders(
    ...,
    primary key(ORDNA),
    foreign key(CID) references CUSTOMERS on delete cascade,
    foreign key(AID) references AGENTS on delete cascade,
    foreign key...
)

alter table PRODUCTS add CITY char(10)

create index XCNO on orders(ordna ASC)

drop index orders.XCNO

--- 第三讲
--- 查询所有选择了database课程的学生编号
select choices.sid
from choices join courses on choices.cid=courses.cid
where courses.cname='database'


--- 第四讲
--- 复杂嵌套查询
select distinct s.sid,s.sname
from STUDENTS s,CHOICES c
where s.sid=c.sid and c.cid=(select cid from COURSES where cname='c++') and
c.score>(select c.score

from STUDENTS s,CHOICES c

where s.sname='ZNKOO' and s.sid=c.sid and c.cid=(select cid from COURSES where
cname='c++'))

--- 第五讲
--- select into
select CID
into ChoicesResult
```

```
from choices
group by cid


--- 第六讲 视图
create view cs(a,b,c,d)--- 重命名
as
select NO,SID,CID,SCORE-- score也可以直接带运算比如score+5 这样得到的是一个常量属性，修改视图就报错，因为无法直接修改表
from CHOICES
where SCORE>=60
with check option--- 视图限制 with check option 要求视图之后也要满足创建时的where谓词

drop view cs --- 删除视图

--- 使用聚集函数
create view S_G(SID,SAVG)
as
select sid,AVG(Score)
from choices
group by sid

--- 第七讲 用户权限
GRANT select
ON COURSES
TO PUBLIC

GRANT insert,update(cid) --- 可以直接赋予更新哪个字段的权利
ON COURSES
TO USER1
with grant option --- 授予传播这两个权限的权力

REVOKE select
ON COURSES
FROM USER1
CASCADE --- 同时将user1授予别人的权限一并取消

REVOKE select --
ON COURSES
FROM PUBLIC


--- 第六讲
--- 空值的用法
select distinct grade
from STUDENTS as s1
where s1.grade>=all(
    select s2.grade
    from STUDENTS as s2
    where s2.grade is not NULL)
---如果嵌套的子查询中存在NULL，那么s1.grade>=all的结果就是假的，因此查询到空的,所以需要提前把NULL的值去掉才能查询到结果


--- 第九讲
--- 创建实体完整性约束
create table class(class_id varchar(4),
```

```
                    CONSTRAINT PK_class_id PRIMARY KEY(class_id)); --- 相当于给主键
取了名字
alter table class add CONSTRAINT PK_class_id PRIMARY KEY(class_id) --- 建表后再加
约束

--- 事务
SET XACT_ABORT ON --- 出错时整个事务回退并回滚
---如果为OFF 只回滚错误的SQL语句 出错后面的语句会继续执行、
BEGIN TRANSACTION T1
COMMIT TRANSACTION T1

--- 第十讲
--- 参照完整性约束
create table t1(
sno char(5) REFERENCES Stu_Union(sno) on delete cascade
);
--- on delete set null
--- on delete no action

alter table choices add CONSTRAINT [FK_CHOICES] foreign KEY([sid]) references
[dbo].[STUDENTS]([sid])
--- dbo代表这个数据表是databaseOwner这个账户拥有，用User2登录要引用USER1创建的表，要加
User1.前缀，但如果不知道就很麻烦，统一给dbo就很方便。
--- 中括号是用来指明是一个属性

--- 一个老师可以授多门课，每个课程只能指定一个老师去听??:
create table listen_course(
    teacher_id char(6),
    course_id char(4)
    CONSTRAINT PK_LISTEN_COURSE primary KEY(teacher_id)
    --- CONSTRAINT PK_LISTEN_COURSE foreign KEY(course_id) REFERENCES
teach_course(course_id) 会出错
)
create table teach_course(
    course_id char(6),
    teacher_id char(4)
    CONSTRAINT PK_TEACH_COURSE primary KEY(course_id)
    CONSTRAINT PK_TEACH_COURSE foreign KEY(teacher_id) REFERENCES
listen_course(teacher_id)
)
alter table listen_course add CONSTRAINT PK_LISTEN_COURSE foreign KEY(course_id)
REFERENCES teach_course(course_id)

--- 第十一讲 用户自定义约束
--- 自定义U1 U2约束
create table Worker(
    Number char(5),
    Name char(8) CONSTRAINT U1 Unique,
    Sage int CONSTRAINT U2 check (Sage<=28)
    primary key (Number)
)
alter table Worker drop U1 --- 删除约束U1 主键也是这样删

--- 创建规则
--- 规定字段值只能是M或F
go --- 一定要有go
create rule rule_sex as @value in ('F','M')
go
```

```
exec sp_bindrule rule_sex ,'worker.[sex]';

exec sp_unbindrule 'worker.[sex]';
drop rule rule_sex

--- 第十二讲
--- 触发器的使用
--- inserted表是要插入的数据不管是插入还是更新
go
create trigger T4 on worker
for insert , update
as
if (select sage from inserted)<(select max(sage)from worker)
begin
print'it dissatisfy the rule'
Rollback transaction
End
---deleted表是要删除的数据
go
create trigger T6 on worker
for delete,update
as
if (select number from deleted)='00001'
Begin
    print'不可以被更改'
    Rollback transaction
End
--- 不允许update 某一字段
go
create trigger T6 on worker
for update
as
if update(number)
Begin
    print'不可以更改编号'
    Rollback transaction
End
--- update时，旧的数据会被放进deleted而新的在inserted 下面是新的年龄要比旧的年龄大
go
create trigger T5 on worker
for update
as
if update(sage)
begin
    if (select sage from deleted) > (select sage from inserted)
    begin
        print'it dissatisfy the rule'
        Rollback transaction
    End
End

--- instead of 结局多表连接视图更新
create view ss as
select st.sid,st.sname,sc.card_id,sc.remained_money
from STUDENTS st,stu_card sc
where st.sid=sc.stu_id

create trigger tri_ins_ss on ss
```

```
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON --- 将不向客户端发送存储过程每个语句的DONE_IN_proc消息
    IF (NOT EXISTS
            (SELECT s.sid from STUDENTS s,inserted i
                where s.sid=i.sid
                )
            )
    BEGIN
        INSERT INTO STUDENTS
            select sid,sname,null,null from inserted
        INSERT INTO Stu_Card
            select card_id,sid,remained_money from inserted
    END
    ELSE PRINT'the data existed'
END

--- 第十三讲
--- 事务
--- 事务和批处理的区别：批处理是由一条或多条SQL语句组成，用go终止语句组。一个批处理中每条语句单独完成或失败，不会影响其他语句的执行
---第一部分
--- 第一题
SET   XACT_ABORT   ON--如果产生错误自动回滚
go -- 用来分割不同的事务
begin tran outside
    update STUDENTS set sname='ysy' where sid='800001216'
    begin tran inside
        insert TEACHERS values('200003125','','','')
    commit tran inside
commit tran outside
--- 内层插入失败后，外层没有执行全部回滚，去students表中验证
select * from STUDENTS where sid='800001216'
--- 名字并不叫ysy

--- 第二题
go -- 用来分割不同的事务
begin tran
    update TEACHERS set tname='ysy2' where tid='200003125'
    SAVE TRAN tran_upd_teachers_done
    insert COURSES values('10001','','')

    if @@ERROR!=0 OR @@ROWCOUNT>1
        BEGIN
            ------ ROLLBACK
            ROLLBACK TRAN tran_upd_teachers_done
            PRINT 'erroR happen when insert courses'
            return
        END
commit tran
--- 查看teachers表可以看到名字被改成了ysy

---第三题
go
create procedure updatecourseINFO
    @courseid char(10),
    @hour int,
```

```
    @returnString varchar(100) out
AS
BEGIN TRAN
    IF not EXISTS(SELECT CID FROM COURSES WHERE cid=@courseid)
        BEGIN
            SELECT @returnString='课程信息不存在'
            GOTO ONERROE
        END
-- NO ERROR INSERT NEW COURSE INFO
    update COURSES set hour=@hour where cid=@courseid
    IF @@ERROR<>0
        BEGIN
            SELECT @returnString='修改课时失败'
            GOTO ONERROE
        END
    SELECT @returnString='课程修改成功'
    PRINT @returnString
commit TRAN
ONERROE:
    PRINT @returnString
    ROLLBACK TRAN
Go

---失败的案例：
declare @courseid char(10)
declare @hour int
declare @returnString varchar(100)
exec insertcourseINFO2 '100010',90,@returnString out
--- 输出： 课程信息不存在

---成功的案例
exec insertcourseINFO2 '10001',90,@returnString out
--- (1 行受影响) 课程修改成功

---第二部分
--- 提交读不允许还没提交就被读取，但允许事务中被写
--- 可重复读 在事务执行中不允许被写，但允许事务中被删除 出现幻象读问题
--- 可串行化 事务隔离最高级别，防止用户在事务完成前更新插入删除数据

---第一题
---代码1
go
BEGIN TRAN
    UPDATE STUDENTS SET sname='ysysb' where sid='800001216'
    WAITFOR DELAY '00:00:05'
    SELECT * FROM STUDENTS WHERE sid='800001216'
    ROLLBACK TRAN
    SELECT * FROM STUDENTS WHERE sid='800001216'
---代码2
go
set transaction isolation level read uncommitted
select * from STUDENTS where sid='800001216'
if @@ROWCOUNT<>0
    BEGIN
        WAITFOR DELAY '00:00:05'
        SELECT * FROM STUDENTS WHERE sid='800001216'
    END
---在执行代码1的时候执行代码2 得到两个sname一个是ysysb一个是ysy，读了脏数据。
```

```
---第二题
---修改后的代码2
go
set transaction isolation level read committed
select * from STUDENTS where sid='800001216'
if @@ROWCOUNT<>0
    BEGIN
        WAITFOR DELAY '00:00:05'
        SELECT * FROM STUDENTS WHERE sid='800001216'
    END
---可以看到输出结果都是ysy

---第三题：
---代码1
set transaction isolation level repeatable read
begin tran
select * from students where sid='800001216'
if @@ROWCOUNT<>0
    BEGIN
        WAITFOR DELAY '00:00:05'
        SELECT * FROM STUDENTS WHERE sid='800001216'
    END
ROLLBACK TRAN

--- 代码2
set transaction isolation level repeatable read
UPDATE STUDENTS SET sname='ysyhello' where sid='800001216'
---可以看到代码1输出的结果名字都是ysy，
--- 而查看数据库发现名字已经被修改成了ysyhello
select * from STUDENTS where sid='800001216'

--- 重新编写代码2，让代码2删除对应的数据项： 因为存在外键的约束，所以先重新插入一个学生：
--- 重写代码1
insert STUDENTS values('123','test','','')
set transaction isolation level repeatable read
begin tran
select * from students where sid='123'
if @@ROWCOUNT<>0
    BEGIN
        WAITFOR DELAY '00:00:05'
        SELECT * FROM STUDENTS WHERE sid='123'
    END
ROLLBACK TRAN

---重写代码2
set transaction isolation level repeatable read
delete STUDENTS where sid='123'
---看到代码1读出了两个相同的数据，但其实在第二次读取的时候，sid为123的学生已经被删除了，但还是读取了，此时就是幻象读。


--- 第四题：
--- 代码一：设置可串行化连续读两次存在的id 123
set transaction isolation level serializable
begin tran
select * from students where sid='123'
WAITFOR DELAY '00:00:05'
```

```
SELECT * FROM STUDENTS WHERE sid='123'
ROLLBACK TRAN

--- 代码二 插入id为123的记录
set transaction isolation level serializable
update STUDENTS set sname='hello' where sid='123'
--- 代码1查询结果显示都一样名字都是ysy，但在数据表中查询，名字发生了改变变成了hello，因此可串行化放置了其他用户更新数据


--- 第十四讲
--- 锁冲突
--- 首先执行第一个
set transaction isolation level repeatable read

begin tran
update courses set hour=80 where cid='10001'

--- 再执行第二个
set transaction isolation level repeatable read
begin tran
    select * from COURSES where cid='10001'
commit tran
--- 第二个一直显示正在执行查询被阻塞
--- 加上了超时后的第二个
set transaction isolation level repeatable read
set lock_timeout 2000 -- 2秒
begin tran
    select * from COURSES where cid='10001'
commit tran
--- 消息1222，级别16，状态51，第4 行 已超过了锁请求超时时段。

--- 同时执行两个下面代码
set transaction isolation level repeatable read
begin tran
    select * from TEACHERS where tid='200003125'
    waitfor delay '00:00:05'
    update TEACHERS set salary=4000 where tid='200003125'
commit tran
    select * from TEACHERS where tid='200003125'
--- 第一个执行成功，第二个出错：
--- 消息1205，级别13，状态51，第6 行 事务(进程ID 53)与另一个进程被死锁在锁资源上，并且已被选作死锁牺牲品。请重新运行该事务。


--- 创建用户
exec sp_addlogin '王二','123456','school','English'
go
use School
go
exec sp_grantdbaccess '王二'

go
create view grade2000 as
    select * from STUDENTS where grade='2000'

go
grant select on grade2000
```

```
    to 王二

go
grant update on dbo.[grade2000]([sname])
    to 王二
```