# 数据库概论综合实习2

## 罗昊 1700010686 常辰 1500012710 易士程 1500012789

# 任务一 层次数据-姓氏起源

## 数据对象

表*PS*用于存放父姓和子姓的关系，有两列组成，*Parent*和*Son*，类型都是*nvarchar(50)*.

```
create table PS(Parent nvarchar(50),Son nvarchar(50));
```

## 导入数据

使用bulk insert导入CSV中的数据.

```
BULK insert dbo.PS
from 'xingshiqiyuan.txt'
WITH (
FIELDTERMINATOR = ',',
ROWTERMINATOR = '\n'
)
```

## 递归查询所有祖先

传入参数*param1*,即需要查询所有祖先的姓.

表*Components*保存所有后代为*param1*的**祖先-后代**对.

使用*with-union all*进行递归查询,每轮找到*param1*的上一级祖先,合并到*Components*中.

最后select并返回找到的祖先姓氏表.

```
CREATE FUNCTION ancestors
(
        @param1 nvarchar(50)
)
RETURNS @returntable TABLE
(
        xing nvarchar(50)
)
AS
BEGIN
        with Components(Ancestor, Derives) as
                (select Parent, Son
                from    PS
                where Son = @param1
                union all
                select  A.Parent, C.Derives
                from    PS A, Components C
                where   A.Son = C.Ancestor
                )
        INSERT @returntable
        SELECT Ancestor from Components
```

```
        RETURN
END
```

## 递归查询所有后代和层次

与之前类似,传入参数*param1*,即需要查询所有后代的姓.

与之前类似,表*Components*保存所有祖先为*param1*的**祖先-后代**对,以及后代相对于*param1*的层次(*lvl*).

注: 后代相对于*param1*的层次指该子孙在以*param1*为起点,该子孙为终点的有向路的路径长度.

使用*with-union all*进行递归查询,每轮找到*param1*的下一级子孙,合并到*Components*中,并令新加入的元组的*lvl*值为生成他的元组的*lvl*值加1.

最后select并返回找到的后代姓氏及层次表.

```
CREATE FUNCTION derives
(
        @param1 nvarchar(50)
)
RETURNS @returntable TABLE
(
        xing nvarchar(50),
        lvl int
)
AS
BEGIN
        with Components(Ancestor, Derives, lvl) as
                (select Parent, Son, 1
                from    PS
                where Parent = @param1
                union all
                select  C.Ancestor, A.Son, C.lvl + 1
                from    PS A, Components C
                where   A.Parent = C.Derives
                )
        INSERT @returntable
        SELECT Derives, lvl from Components
        RETURN
END
```

## 测试结果

测试查询如下:

### 祖先查询

```
select distinct *
from dbo.ancestors(N'田')
```

查询结果:

```
xing
--------------------------------------------------
风
妫
姬
姚
```

## 后代及层级查询

```
select distinct *
from dbo.derives(N'田')
```

查询结果:

```
xing                                               lvl
-------------------------------------------------- -----------
安                                                  4
白                                                  3
班                                                  3
包                                                  3
曹                                                  2
查                                                  3
车                                                  1
成                                                  3
楚                                                  3
第                                                  2
第八                                                 1
第二                                                 1
第六                                                 1
第七                                                 1
第三                                                 1
第四                                                 1
第五                                                 1
第一                                                 1
东方                                                 4
斗                                                  3
鄂                                                  3
法                                                  1
勾                                                  3
苟                                                  3
顾                                                  3
关                                                  5
己                                                  2
金                                                  3
荆                                                  3
景                                                  3
句                                                  3
李                                                  5
理                                                  4
廖                                                  3
龙                                                  5
陆                                                  1
路                                                  3
潞                                                  3
罗                                                  4
羊                                                  2
苗                                                  4
莫                                                  3
倪                                                  3
潘                                                  3
彭                                                  2
蒲                                                  3
戚                                                  2
戚                                                  4
钱                                                  3
屈                                                  3
权                                                  3
阮                                                  4
上官                                                 3
沈                                                  3
舒                                                  4
孙                                                  1
孙                                                  3
```

| | |
|---|---|
| 谈 | 3 |
| 郯 | 3 |
| 童 | 4 |
| 韦 | 3 |
| 伍 | 3 |
| 熊 | 3 |
| 许 | 1 |
| 严 | 4 |
| 颜 | 3 |
| 偃 | 3 |
| 阳 | 3 |
| 叶 | 4 |
| 尹 | 3 |
| 尤 | 4 |
| 郁 | 3 |
| 云 | 2 |
| 张 | 3 |
| 斟 | 2 |
| 朱 | 3 |
| 邾 | 3 |
| 祝 | 3 |
| 庄 | 3 |
| 邹 | 3 |
| 左 | 3 |

# 任务二 图-冲突/视图可串行化判定

## 数据对象

### 调度-表*Ord*

| 列名 | 类型 | 意义 | 约束 |
|---|---|---|---|
| Order_num | int | 操作序号 | primary key |
| Trans_id | varchar(10) | 事务编号 | |
| Operation_type | varchar(10) | 操作类型 | 读(read) 或 写(write) |
| Data_item | varchar(10) | 操作的数据项 | |

定义代码:

```
create table Ord(
Order_num int primary key,
Trans_id varchar(10),
Operation_type varchar(10) check(Operation_type in ('read','write')),
Data_item varchar(10),
)
```

### 基本图类型*GraphType*

定义了基本图类型*GraphType*,实质是边的集合 | 列名 | 类型 | 意义 | 约束 | | ------ | ------ | ------ | ------ | | Prior_Tran | varchar(10) | 边的起点 | | | Posterior_Tran | varchar(10) | 边的终点 | |

定义代码:

```
create type GraphType as table (Prior_Tran varchar(10), Posterior_Tran varchar(10))
```

## 基本判圈过程*DetactCycle*

*DetactCycle* 是一个表值参数储存过程, 他接受一个*GraphType*类型的表作为参数, 然后通过递归查询查找是否有圈存在.

如果有圈就返回1, 否则返回0

```
create proc DetactCycle(@Graph as GraphType READONLY)
as
begin
    with PriorGraph(Prior_Tran, Posterior_Tran) as
        (select Prior_Tran,Posterior_Tran
        from    @Graph
        union all
        select  A.Prior_Tran, B.Posterior_Tran
        from    @Graph A, PriorGraph B
        where   A.Posterior_Tran = B.Prior_Tran and A.Prior_Tran <> B.Posterior_Tran
         and not exists
         (select *
         from @Graph C
         where C.Prior_Tran = A.Prior_Tran and C.Posterior_Tran = B.Posterior_Tran
         )
        )
        select *
        into #PriorGraph
        from PriorGraph
        if( exists(
                select *
                from #PriorGraph G1, #PriorGraph G2
                where G1.Prior_Tran = G2.Posterior_Tran and G1.Posterior_Tran = G2.Prior_Tran
                )
                )
                return 1
        return 0
end
```

## 冲突可串行化判定*ConflictSer*

*ConflictSer*是一个储存过程, 用于判定*Ord*操作表是否是冲突可串行化的.

他先通过*Ord*中的信息构建冲突图*ConflictGraph*,

然后调用*DetactCycle*过程判圈.

```
create proc ConflictSer
as
begin
        declare @ConflictGraph GraphType

        -- 构建冲突图
        insert into @ConflictGraph
        select distinct O1.Trans_id, O2.Trans_id
        from Ord O1, Ord O2
        where O1.Order_num>O2.Order_num and O1.Trans_id <> O2.Trans_id
                        and O1.Data_item = O2.Data_item
                        and not(O1.Operation_type = 'read' and O2.Operation_type = 'read')

        select N'下一张表:冲突可串行化判定图';
        select * from @ConflictGraph;

        declare @result int

        -- 执行判圈过程
        exec @result = DetactCycle @ConflictGraph
```

```
        return @result
end
```

## 视图可串行化判定*ViewSer*

*ViewSer*是一个储存过程, 用于判定*Ord*操作表是否是视图可串行化的.

他先通过*Ord*中的信息找出所有Tj读取Ti写入的数据项Q, 存放在临时表 `#ViewGraphInit` 中

然后找出所有:对于数据项Q，如果Tj读取Ti写入的Q值，Tk执行write (Q)操作的(Ti,Tj,Tk)对及编号e, 存放在临时表 `#ViewGraphFinalPart3` 中,用于下一步向图中添加边.

然后构建视图可串行化判定图 `#ViewGraph` , 由 `#ViewGraphInit` 和 如下三条规则组成:

1. 如果Ti = Tb且Tj ≠ Tf，则在带标记的优先图中插入边Tj--0-->Tk ;
2. 如果Ti ≠ Tb且Tj = Tf，则在带标记的优先图中插入边Tk--0-->Ti ;
3. 如果Ti ≠ Tb且Tj ≠ Tf，则在带标记的优先图中插入边Tk--e-->Ti与Tj--e-->Tk , 其中e为上一步生成的编号.

然后枚举所有标号>0 的边的选择情况.

设最大标号为 `@total_label` , 则共有 `@total_comb = power(2,@total_label)` 种情况,

通过 `@i` 从0到 `@total_comb` 枚举, 遍历所有可能的情况并为其生成测试图 `@TestGraphDistinct` ,

再调用*DetactCycle*储存过程, 判定测试图中是否存在圈.

只要有一种枚举情形测试图中没有圈, 则该调度就是视图可串行化的.

```
create proc ViewSer
as
begin
        -- 创建初始图， Read_Tran读取From_Tran写入的Data_item
        with ViewGraphInit(From_Tran, Read_Tran, Data_item) as
        (select distinct O1.Trans_id, O2.Trans_id, O1.Data_item
        from Ord O1, Ord O2
        where O1.Operation_type = 'write' and O2.Operation_type = 'read' and O1.Order_num < O2.Order_num
                and O1.Trans_id <> O2.Trans_id and O1.Data_item = O2.Data_item
                and (not exists( select *
                        from Ord O3
                        where O3.Operation_type = 'write' and O3.Order_num > O1.Order_num
                                and O3.Order_num < O2.Order_num and O3.Data_item = O1.Data_item
                                ))
        union
        select 'Tb', O2.Trans_id, O2.Data_item
        from Ord O2
        where O2.Operation_type = 'read'
                and not exists( select *
                        from Ord O3
                        where O3.Operation_type = 'read' and O3.Order_num < O2.Order_num
                                and O3.Data_item = O2.Data_item)
        union
        select O1.Trans_id, 'Te', O1.Data_item
        from Ord O1
        where O1.Operation_type = 'write'
                and not exists( select *
                        from Ord O3
                        where O3.Operation_type = 'write' and O3.Order_num > O1.Order_num
                                and O3.Data_item = O1.Data_item)
        )
        select *
        into #ViewGraphInit
        from ViewGraphInit
```

```
select *
into #ViewGraphSecond
from #ViewGraphInit
;

-- 用与构建有标号的边的中间表
select row_number() over (ORDER BY G1.From_Tran) as Edge_Label,
G1.From_Tran as Prior_Tran,G1.Read_Tran as Posterior_Tran,
O1.Trans_id as Other_Tran
into #ViewGraphFinalPart3
from #ViewGraphInit G1, Ord O1
where G1.From_Tran<>'Tb' and G1.Read_Tran<>'Te'
        and O1.Operation_type = 'write' and O1.Trans_id<>G1.From_Tran
        and O1.Trans_id<>G1.Read_Tran and O1.Data_item = G1.Data_item
;

-- 构建视图可串行化判定图
with ViewGraphFinal(Edge_Label,Prior_Tran,Posterior_Tran) as
(select 0,From_Tran,Read_Tran
from #ViewGraphSecond
union
select 0,G1.Read_Tran,O1.Trans_id
from #ViewGraphSecond G1, Ord O1
where G1.From_Tran = 'Tb' and O1.Operation_type = 'write'
        and O1.Data_item = G1.Data_item and G1.Read_Tran <> O1.Trans_id
union
select 0,O1.Trans_id, G1.From_Tran
from #ViewGraphInit G1, Ord O1
where G1.Read_Tran = 'Te' and O1.Operation_type = 'write'
        and G1.Data_item = O1.Data_item and O1.Trans_id<> G1.From_Tran
union
select Edge_Label,Other_Tran,Prior_Tran
from #ViewGraphFinalPart3
union
select Edge_Label,Posterior_Tran,Other_Tran
from #ViewGraphFinalPart3
)
select *
into #ViewGraph
from ViewGraphFinal
;

-- 标号>0的边加入可选择边表
select Edge_Label,Prior_Tran,Posterior_Tran,
               rank() over (partition by Edge_Label order by Prior_Tran) as tag
into #OptionGraph
from #ViewGraph
where Edge_Label > 0

-- 表示组合的码
declare @total_comb int
declare @total_label int
set @total_label = (select max(Edge_Label) from #OptionGraph)
set @total_comb = power(2,@total_label)

select N'下一张表:视图可串行化判定图';
select * from #ViewGraph;
--select * from #OptionGraph

-- 枚举所有组合
declare @i int
declare @succeed int
set @i = 0
set @succeed = 0
while( @i < @total_comb )
begin
        declare @TestGraph GraphType;
```

```
                -- 将0边插入测试图
                insert into @TestGraph
                select Prior_Tran, Posterior_Tran
                from #ViewGraph
                where Edge_Label = 0

                -- 按照前面选定的组合将有标号的边插入测试图
                declare @j int, @k int
                set @j = 1
                set @k =@i
                while @j <= @total_label
                begin
                        declare @c int
                        set @c = @k % 2
                        set @k = @k / 2
                        insert into @TestGraph
                                select Prior_Tran, Posterior_Tran
                                from #OptionGraph
                                where Edge_Label = @i
                                        and tag = @c + 1
                        set @j += 1
                end
                set @i += 1;

                declare @TestGraphDistinct GraphType
                insert into @TestGraphDistinct
                select distinct *
                from @TestGraph

                -- 调用基本图判圈过程
                declare @tmp int
                exec @tmp = DetactCycle @TestGraphDistinct;

                if( @tmp = 0)
                begin
                        set @succeed = 1
                end

                delete from @TestGraph
                delete from @TestGraphDistinct
        end

        --返回
        if(@succeed = 0)
                return 1

        return 0
end
```

## 测试代码

```
insert into Ord Values
(1,'T1','read','A'),
(2,'T2','write','A'),
(3,'T3','read','A'),
(4,'T1','write','A'),
(5,'T3','write','A');

declare @a int, @b int
exec @a = ConflictSer
if(@a=1)
        select N'当前调度不是冲突可串行化的' as result;
else
        select N'当前调度是冲突可串行化的' as result;
exec @b = ViewSer
```

```
if(@b=1)
        select N'当前调度不是视图可串行化的' as result;
else
        select N'当前调度是视图可串行化的' as result;
go
```

## 运行结果

```
--------------
下一张表:冲突可串行化判定图

Prior_Tran Posterior_Tran
---------- --------------
T1         T2
T1         T3
T2         T1
T3         T1
T3         T2

result
-------------
当前调度不是冲突可串行化的

--------------
下一张表:视图可串行化判定图

Edge_Label           Prior_Tran Posterior_Tran
-------------------- ---------- --------------
0                    T1         T2
0                    T1         T3
0                    T2         T3
0                    T3         Te
0                    Tb         T1
1                    T1         T2
1                    T3         T1

result
------------
当前调度是视图可串行化的
```

# 任务3 序列数据-谈股论金

## 指标计算

我们这里计算的是威廉指标.

$%R=\frac{H_n-C_n}{H_n-L_n}\times-100$

1. n:是交易者设定的交易期间(常用为14天);
2. 第n日的最新收盘价;
3. H_n:是过去n日内的最高价(如14天的最高价);
4. L_n:是过去n日内的最低价(如14天的最低价).

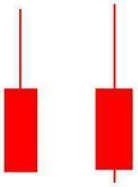
```
create function WR(
        @param1 int
)
returns @returntable table(ID int, "W%R" real)
begin
   insert into @returntable
```

```
    select ID,
    (((max(HighPrice) over (order by IndexDate rows between 13 preceding and 0 following))
     - ClosePrice)
     /((max(HighPrice) over (order by IndexDate rows between 13 preceding and 0 following))
     -(min(LowPrice) over (order by IndexDate rows between 13 preceding and 0 following)))
     * (-100))
     as "W%R"
    from SP
    where ID = @param1
    return
end
```

## K线识别

我们这里识别的是长上影线.

长上影线是指在股票交易过程当中，做空者和做多者之间的一种力量的博弈中出现的一种情况，在开盘的时候，股票或者期指当中低开高走然后又回调到开盘的价格，这是就会出现长长的上影线。当出现这种情况的时候，需要提高自己的投资风险警惕性。



1. 底部长上影:

   个股在底部出现长上影，一般是庄家想拉升个股，不巧因为大盘走坏或是抛盘过多，结果造成个股收成长上影！（建议加入自选股关注，一般收出长上影后，还需要一些时间来洗盘振荡，什么时候放量突破这根上影线，就证明拉升行情的开始）

2. 上涨中期长上影

   个股在上涨中期出现长上影，可让散户误认为是上涨末期长上影，导致技术形散户被庄家洗出，第二日不跌反涨，让自以为技术形逃顶成功的散户大跌眼镜！怎么样才能区分呢？下跌无量、前期总体涨幅并不是很大，是其主要特点。

3. 上涨末期长上影

   个股在经过长期的拉升之后，收一根长上影。下跌放巨量，各个技术指标相继形成死叉，表示庄家已无心恋战，散户最好果断出局，保住利润为宜！如果周线形成长上影线，那就可成形成长期顶部区域，更要警惕其巨大风险！

```
create function LongUpShadowLine(
        @param1 int
)
returns @returntable table(ID int, "W%R" real)
begin
    insert into @returntable
    select *, 'yes'
    from SP
    where ID=@param1 and ((HighPrice - OpenPrice)/OpenPrice > 0.02) and
                ((OpenPrice=ClosePrice) or (OpenPrice<>ClosePrice and
                 ((HighPrice-(case when OpenPrice > ClosePrice then OpenPrice else ClosePrice end))
        /(case when OpenPrice > ClosePrice
                        then OpenPrice - ClosePrice
                        else ClosePrice - OpenPrice end) > 7)))
        return
end
```

# 测试代码

```
-- 创建股票数据表
create table SP(
IndexDate date,
OpenPrice money,
HighPrice money,
LowPrice money,
ClosePrice money,
Volume real,
AdjClose money);
-- 创建纯字符股票数据表
create table SP_O(
t1 varchar(20),
t2 varchar(20),
t3 varchar(20),
t4 varchar(20),
t5 varchar(20),
t6 varchar(20),
t7 varchar(20)
);
go
--导入股票数据
bulk insert dbo.SP_O
from 'C:\\Dropbox\\sql\\shixi2\\sp500.csv'
with
(
fieldterminator = ',',
rowterminator = '\n'
);
go
--转换股票数据为正确类型
insert into SP
select convert(date,t1),convert(money,t2),convert(money,t3),
convert(money,t4),convert(money,t5),convert(real,t6),convert(money,t7)
from SP_O
go
-- 计算威廉指标
select *,
(((max(HighPrice) over (order by IndexDate rows between 13 preceding and 0 following))
 - ClosePrice)
 /((max(HighPrice) over (order by IndexDate rows between 13 preceding and 0 following))
 -(min(LowPrice) over (order by IndexDate rows between 13 preceding and 0 following)))
 * (-100))
 as "W%R"
from SP
go
-- 识别长上影线
select *, 'yes'
from SP
where ((HighPrice - OpenPrice)/OpenPrice > 0.02) and
                ((OpenPrice=ClosePrice) or (OpenPrice<>ClosePrice and
                 ((HighPrice-(case when OpenPrice > ClosePrice then OpenPrice else ClosePrice end))
        /(case when OpenPrice > ClosePrice
        then OpenPrice - ClosePrice
        else ClosePrice - OpenPrice end) > 7)))
```

# 查询结果

```
威廉指标(只显示top 10)
IndexDate  OpenPrice HighPrice LowPrice ClosePrice Volume    AdjClose W%R
---------- --------- --------- -------- ---------- --------- -------- ------
1970-01-02 92.06     93.54     91.79    93.00      8050000   93.00    -30.85
1970-01-05 93.00     94.25     92.53    93.46      1.149E+07 93.46    -32.11
```

```
1970-01-06 93.46     93.81     92.13    92.82      1.146E+07 92.82    -58.13
1970-01-07 92.82     93.38     91.93    92.63      1.001E+07 92.63    -65.85
1970-01-08 92.63     93.47     91.99    92.68      1.067E+07 92.68    -63.82
1970-01-09 92.68     93.25     91.82    92.40      9380000   92.40    -75.20
1970-01-12 92.40     92.67     91.20    91.70      8900000   91.70    -83.60
1970-01-13 91.70     92.61     90.99    91.92      9870000   91.92    -71.47
1970-01-14 91.92     92.40     90.88    91.65      1.038E+07 91.65    -77.15
1970-01-15 91.65     92.35     90.73    91.68      1.112E+07 91.68    -73.01
```

长上影线(所有长上影线)

```
IndexDate  OpenPrice HighPrice LowPrice ClosePrice Volume        AdjClose
---------- --------- --------- -------- ---------- ------------ -------- ----
1970-06-09 76.29     79.96     75.58    76.25      7050000       76.25    yes
1970-06-17 76.15     78.04     75.63    76.00      9870000       76.00    yes
1973-12-19 94.74     96.83     93.81    94.82      2.067E+07     94.82    yes
1974-02-19 92.27     94.44     91.68    92.12      1.594E+07     92.12    yes
1974-07-18 83.70     85.39     83.13    83.78      1.398E+07     83.78    yes
1980-11-05 130.77    135.65    130.77   131.33     8.408E+07     131.33   yes
1980-12-18 132.89    135.90    131.89   133.00     6.957E+07     133.00   yes
1987-10-28 233.19    238.58    226.26   233.28     2.794E+08     233.28   yes
1998-10-20 1062.39   1084.06   1060.61  1063.93    9.582E+08     1063.93  yes
2003-04-07 878.85    904.89    878.85   879.93     1.494E+09     879.93   yes
2008-10-08 988.91    1021.06   970.97   984.94     8.71633E+09   984.94   yes
2008-10-10 902.31    936.36    839.80   899.22     1.145623E+10  899.22   yes
2008-10-17 942.29    984.64    918.74   940.55     6.58178E+09   940.55   yes
2008-12-19 886.96    905.47    883.02   887.88     6.70531E+09   887.88   yes
2009-03-06 684.04    699.09    666.79   683.38     7.33183E+09   683.38   yes
```