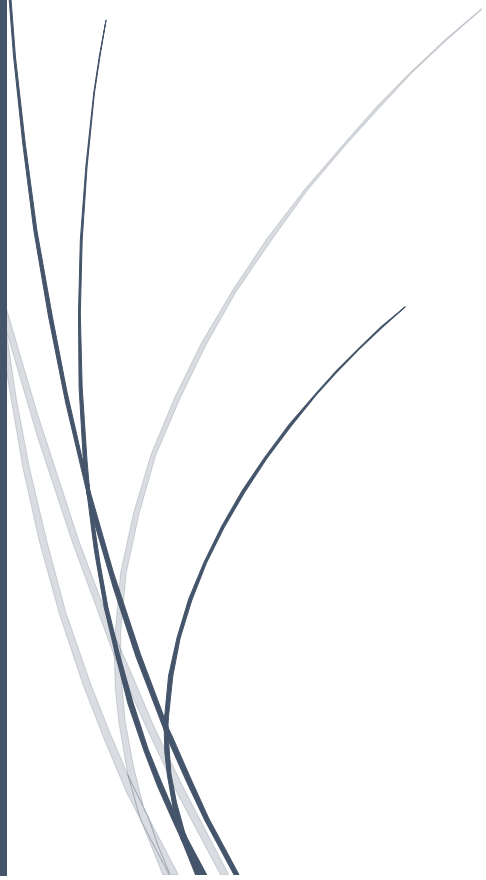
2016-5-26

数据库实习二

汪思学 1300012784
孙志玉 1300012840
万炽洋 1300012882



Cecil Wang
PEKING UNIVERSITY

目录

约束设计

一、 函数约束练习

1. 题目

定义一个函数，它接收一个身份证号，并计算校验码，正确返回 1，错误返回 0。然后在一个表中定义身份证列，并为该列添加一个 check 约束，约束条件就是该校验函数。身份证第十八位数字的计算方法如下：

1. 将前面的身份证号码 17 位数分别乘以不同的系数。从第一位到第十七位的系数分别为：7 9 10 5 8 4 2 1 6 3 7 9 10 5 8 4 2
2. 将这 17 位数字和系数相乘的结果相加用加出来和除以 11，看余数是多少
3. 余数 0 1 2 3 4 5 6 7 8 9 10 分别对应的最后一位身份证的号码为 1 0 X 9 8 7 6 5 4 3 2

2. 代码与分析

由于身份证验证的算法已经给出，所以整个算法分为三个部分：用户自定义函数、建立数据库表、测试算法。以下给出代码

首先我们要清空就旧数据和残留数据

```
1 --清理数据库
 2 if OBJECT_ID('ID') is not null
 3     drop table ID
 4 if object_id('CheckIDNumber') is not null
 5     drop function CheckIDNumber
 6 go
 7
```

然后给出用户自定义的函数约束。具体算法是：

1. 接收一个 varchar(18)的身份证序号
2. 判断其长度是否为 18
3. 建立余数与最后一位对应关系的映射表
4. 计算前 17 位与对应系数乘积的和，在这一步中可以利用 convert 函数将字符转换成数字，并且可以使用 substring 来获取每一位的数字
5. 计算余数并取得映射后的字符
6. 对比结果是否和第十八位一致

```
 8 --创建函数约束
 9 create function CheckIDNumber (
10     @IDNumber varchar(18)
11 )
12 returns bit
```

```
as
begin
    --判断身份证号码长度是否为 18
    if(len(@IDNumber)<>18)
        return 0

    --定义最后一位映射表 7
    declare @MapTable table(value char, idn int)
    insert into @MapTable (value, idn) values ('1', 0)
    insert into @MapTable (value, idn) values ('0', 1)
    insert into @MapTable (value, idn) values ('X', 2)
    insert into @MapTable (value, idn) values ('9', 3)
    insert into @MapTable (value, idn) values ('8', 4)
    insert into @MapTable (value, idn) values ('7', 5)
    insert into @MapTable (value, idn) values ('6', 6)
    insert into @MapTable (value, idn) values ('4', 7)
    insert into @MapTable (value, idn) values ('3', 8)
    insert into @MapTable (value, idn) values ('2', 9)

    --定义计算表，result 列对应每一位身份证号与对应的系数相乘的结果
    declare @CalcTable table(result int)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 1, 1)) * 7)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 2, 1)) * 9)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 3, 1)) * 10)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 4, 1)) * 5)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 5, 1)) * 8)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 6, 1)) * 4)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 7, 1)) * 2)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 8, 1)) * 1)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 9, 1)) * 6)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 10, 1)) * 3)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 11, 1)) * 7)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 12, 1)) * 9)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 13, 1)) * 10)
```

```
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 14, 1)) * 5)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 15, 1)) * 8)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 16, 1)) * 4)
    insert into @CalcTable  values (CONVERT(int,
SUBSTRING(@IDNumber, 17, 1)) * 2)

    --计算余数
    declare @checksum int
    select @checksum = sum(result) from @CalcTable
    set @checksum = @checksum  % 11

    --取出映射
    declare @mapvalue char
    select @mapvalue=value from @MapTable where idn = @checksum

    --判断是否符合映射
    if(SUBSTRING(@IDNumber, 18, 1) <> @mapvalue )
        return 0

    return 1
end;
go
```

最后将函数约束添加到数据库表

```
--建立数据库表并添加约束
create table ID (IDNumber char(18)
check(dbo.CheckIDNumber(IDNumber)=1));
go

```

3. 测试与结果

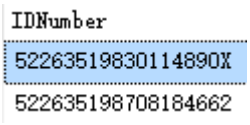
```
--测试样例
insert into ID values('52263519830114890X');
insert into ID values('130231252361241241');
insert into ID values('522635198708184662');
insert into ID values('522635198708184612');
go

--输出结果
select * from ID
go
```

在测试中，我们从百度搜索出正确的身份证号作为数据，其中正确的有

52263519830114890X 和 522635198708184662

以下是运行结果的截图



在观察运行结果输出的信息

```
(1 row(s) affected)
Msg 547, Level 16, State 0, Line 75
The INSERT statement conflicted with the CHECK constraint "CK__ID__IDNumber__17AD7836". The conflict occurred in database "master", table "dbo.ID", column 'IDNumber'.
The statement has been terminated.

(1 row(s) affected)
Msg 547, Level 16, State 0, Line 77
The INSERT statement conflicted with the CHECK constraint "CK__ID__IDNumber__17AD7836". The conflict occurred in database "master", table "dbo.ID", column 'IDNumber'.
The statement has been terminated.

(2 row(s) affected)
```

可以看到第二条和第四条 insert 语句触发了 check 约束

一、触发器练习

1. 题目

如下三个表，员工可以在多个部门工作，pct_time 表示员工在某个部门工作的百分比

Emp(eid, ename, age, salary)
Works(eid, did, pct_time)
Dept(did, budget, managerid)

1、使用函数约束保证管理者的工资必须高于他所管理的任何一个员工

2、使用触发器保证任何一个员工工资的增加，都必须按照他在部门工作的百分比加到相应部门的预算上

2. 代码与分析

首先我们在建立表的时候使用了外部约束具体的讲:Dept 的 managerid 是外码应用了 Emp 的 eid，而 Works 的 eid 和 did 都是外码分别引用了 Emp 的 eid 和 Dept 的 did。于是有如下创建数据库代码

```
 2 --建立数据库表
 3 create table Emp
 4     (eid int,
 5      ename varchar(20),
 6      age int,
 7      salary int,
 8      primary key(eid))
 9
10 create table Dept
```

```
11      (did int,
12       budget int,
13       managerid int,
14       primary key (did),
15       foreign key (managerid) references Emp(eid))
17
18 create table Works
19      (eid int,
20       did int,
21       pct_time real,
22       primary key (eid, did),
23       foreign key (eid) references Emp(eid),
24       foreign key (did) references Dept(did))
25
26 go
```

函数约束的设计很简单，最外层使用 exists 判断是否存在员工，里层使用嵌套子查询，将条件限定在 manager 所在部门的员工和工资比 manager 高的员工

```
 2 --创建函数约束
 3 create function CheckSalary (
 4      @did int,
 5      @managerid int
 6 )
 7 returns bit
 8 as
 9 begin
10      --判断是否存在比管理员薪水高的员工
11      if exists (
12      select eid from Emp
13      where (eid) in (select eid from Works where did=@did and eid <>
@managerid) --判断员工是否在对应的部门工作
14      and salary >= (select salary from Emp where eid = @managerid) )
--获得管理员的薪水与其比较
15          return 0
16
17      return 1
18 end
19 go
```

然后将函数约束添加到 Dept 表中

```
58 alter table Dept add check(dbo.CheckSalary(did, managerid)=1)
```

最后是触发器的设计，我们选择使用游标来满足当有多行被更新时也可以正确运行。

触发器首先会判断影响的行数是否大于 0，如果否则会直接退出。

紧接着会利用 update 函数来判断是否对 salary 属性进行了 update，如果否则不会做任何事情。

之后进入触发器的核心部分，我们选择先计算更新时 salary 属性的变化值，方法是通过 inserted 和游标 fetch 结合，两者做差取得。之后是更新部门预算，我们使用 update 操作并在 from 中使用 join…on…操作将 Dept 和一个 select 子查询的结果连接，而 select 子查询的结果是此名员工工作所在的部门的子表，和 Dept 连接后实际上是得到了此名员工所在的部门并且表中有部门的预算属性，由此可以使用 set 更新即可。Update 之后 fetch 下一个更新的元组。

```
 99 --薪水更新触发器
100 create trigger ChangeSalary on Emp after update as
101     if ( @@ROWCOUNT = 0 ) return
102     --判断是否是更新了 salary 属性
103     if update (salary)
104     begin
105         declare @eidd int, @salaryd int, @salaryi int, @increment
int
106         --通过游标的方式取出修改前的元组
107         declare eDeleted cursor for select eid, salary from deleted
108         open eDeleted
109         fetch next from eDeleted into @eidd, @salaryd
110         while @@FETCH_STATUS = 0
111         begin
112             --取出修改后的值
113             select @salaryi=salary from inserted where eid=@eidd
114             --计算差值
115             set @increment = @salaryi - @salaryd
116
117             --更新部门预算
118             update Dept
119             set budget = budget + @increment * t2.pct_time
120             from Dept join (select * from Works where eid=@eidd) t2
on Dept.did = t2.did
121             where Dept.did = t2.did
122
123             --取出下一个元组
124             fetch next from eDeleted into @eidd, @salaryd
125         end
126         close eDeleted
127         deallocate eDeleted
128     end
129 go
```

3. 测试与结果

测试内容分两部分做：第一部分是函数约束的测试，第二部分是触发器的测试。

函数约束测试中，由此之前建立数据库表是设置了外码给接下来的数据插入造成了一定的影响，意味着我们首先要插入 Emp 数据之后插入 Dept 数据最后插入 Works 数据。但由于 check 约束仅仅在对添加了 check 约束的表做操作时才会触发，所以我们添加 Dept 数据时并不添加 managerid，否则之后添加 Works 时并不会触发 Dept 的 check，而之前添加在 Dept 表中的 manager 是违规的。事实上这种操作顺序更符合实际，首先有员工，之后有部门，然后分配工作，最后设置主管。

以下是我们测试使用的代码

在第一组测试样例中，我们按照错误的方式（即创建部门的同时安排主管）插入数据

```
61 --样例一
62 --添加员工
63 insert into Emp values(1,'1',1,2)
64 insert into Emp values(2,'1',1,1)
65 insert into Emp values(3,'1',1,1)
66 insert into Emp values(4,'1',1,1)
67 select * from Emp
```

	eid	ename	age	salary
1	1	1	1	2
2	2	1	1	1
3	3	1	1	1
4	4	1	1	1

```
69 --添加部门
70 insert into Dept(did, budget, managerid) values (1,1,1)
71 insert into Dept(did, budget, managerid) values (2,1,4)
72 select * from Dept
```

	did	budget	managerid
1	1	1	1
2	2	1	4

```
74 --分配工作
75 insert into Works values(1,1, 0.5)
76 insert into Works values(2,1, 0.5)
77 insert into Works values(3,2, 0.2)
78 insert into Works values(4,2, 0.5)
79 insert into Works values(3,1, 0.8)
80 select * from Works
```

	eid	did	pct_time
1	1	1	0.5
2	2	1	0.5
3	3	1	0.8
4	3	2	0.2
5	4	2	0.5

此时我们可以看到部门 2 的主管是 4 号员工，但是 4 号员工的工资和他管理的员工工资一样，这是违规的，但是当我们分配工作的时候，只对 Works 表做了操作，并不会触发 Dept 的 check 约束，所以 Dept 此仍不知道内部数据发生了违规。

```
82 --重新分配主管
83 update Dept set managerid = 3 where did = 2
84 select * from Dept
```

	did	budget	managerid
1	1	1	1
2	2	1	4

如果此时对 Dept 表的部门 2 做操作，会触发 check 约束，

```
(5 row(s) affected)
Msg 547, Level 16, State 0, Line 83
The UPDATE statement conflicted with the CHECK constraint "CK__Dept__2D9CB955". The conflict occurred in database "master", table "dbo.Dept".
The statement has been terminated.
```

但我们看到虽然 managerid 没有改变为 3，但是它仍保持为 4，这意味着 check 约束只会对操作检查，并不会对已经在表中数据做检查

在第二组测试样例是在第一组测试样例的基础上的，我们按照正确的方式（即创建部门的时候不会安排主管）插入数据

```
86 --样例二
87 insert into Dept(did, budget) values(3,1)
88 select * from Dept
```

	did	budget	managerid
1	1	1	1
2	2	1	4
3	3	1	NULL

```
90 insert into Works values (2,3,0.1)
91 insert into Works values (3,3,0.1)
92 insert into Works values (4,3,0.1)
93 select * from Works
```

	eid	did	pct_time
1	1	1	0.5
2	2	1	0.5
3	2	3	0.1
4	3	1	0.8
5	3	2	0.2
6	3	3	0.1
7	4	2	0.5
8	4	3	0.1

```
95 update Dept set managerid = 3 where did = 3
96 select * from Dept
```

	did	budget	managerid
1	1	1	1
2	2	1	4
3	3	1	NULL

可以看到此时触发了 check 约束，同时数据库拒绝了操作

```
(8 row(s) affected)
Msg 547, Level 16, State 0, Line 95
The UPDATE statement conflicted with the CHECK constraint "CK__Dept__39OE6CO1". The conflict occurred in database "master", table "dbo.Dept".
The statement has been terminated.
```

触发器的测试是基于函数约束测试的

```
2 --测试样例
3 update Emp set salary = 21 where eid = 3
4 select * from Emp
5 select * from Dept
```

	eid	ename	age	salary
1	1	1	1	2
2	2	1	1	1
3	3	1	1	21
4	4	1	1	1

	did	budget	managerid
1	1	17	1
2	2	5	4
3	3	3	NULL

可以看到当对 3 号员工工资提高 20 的时候，会对其相应的部门 1、2 按照百分比增加。

同样的由于薪水更新操作只影响了 Emp 表，此时 Dept 表中的 check 约束不会触发，而其内部已经有违规数据存在了。

二、 函数依赖练习

1. 题目

表 STC(sno, tno, cno)上成立函数依赖 tno-> cno 和(sno,cno)-> tno

- 分别使用函数约束和触发器来维护这两个函数依赖，拒绝违反函数依赖的数据插入
- 往表中插入数据，限定 sno, tno, cno 的范围分别为 1~10000，1~1000，1~100。随机产生行，进行插入，尝试 10 万次插入
- 比较两种约束维护方法的性能
- 插入完成后，统计(tno, cno)的冗余，这就是 3NF 为维护函数依赖所付出的代价

2. 分析与代码

满足函数依赖 tno->cno 意味着 tno 属性相同的元组在 cno 属性上取值一样，由于在实现时是从一个空表开始，意味着可以运用归纳的方法，空表是满足函数依赖的，那么当插入一个新的元组时，只要保证新的元组不破坏函数依赖即可以保证表满足函数依赖，那么我们只需要验证新加入元组的 cno 属性是否和表中与其 tno 属性一样元组中的任意一个的 cno 属性等值。

于是有如下代码

```
 2 create function Check3NF(
 3     @sno int,
 4     @tno int,
 5     @cno int
 6 ) returns bit as
 7 begin
 8     --判断是否满足 tno->cno
 9     if exists(
10     select *
11     from STC
12     where tno = @tno and cno <> @cno)
13         return 0
14
15     --判断是否满足(sno, cno)->tno
16     if exists(
17     select *
18     from STC
19     where sno = @sno and cno = @cno and tno <> @tno)
20         return 0
21
22     return 1
23 end
24 go
```

判断是否满足函数依赖等价于判断是否存在这样的数据，即插入元组的 tno 与其 tno 属性相同，但是插入元组的 cno 与其 cno 属性不同，类似的函数依赖(sno,cno)->tno 也是通过 exists 判断，对于触发器通过游标 fetch 后可使用类似的代码尽心查询，在此不再赘述。

3. 结果

首先为了测试的准确性，在使用两种方法之前我们都会清空缓存。即

```
--清除缓存
dbcc dropcleanbuffers
dbcc freeproccache
go
```

随机行的生成我们使用 RAND 函数，为了保证触发器不会遇到 error 而终止，我们每次插入都会 try…catch…并在 catch 中记录失败的次数，最后会输出结果并做统计。

```
--随机插入
declare @totaltime int
set @totaltime = 100000
declare @starttime datetime
set @starttime=GETDATE()
declare @failtime int
set @failtime = 0
declare @i int
set @i = 0
while @i < @totaltime
begin
    begin try
        insert into STC values(
            FLOOR(RAND()*10000),
            FLOOR(RAND()*1000),
            FLOOR(RAND()*100))
    end try
    begin catch
        set @failtime = @failtime + 1
    end catch
    set @i = @i + 1
end
--select * from STC
select [函数约束时间（ms）]=datediff(ms, @starttime, getdate())
select [成功次数]=@totaltime - @failtime
select [失败次数]=@failtime
select [tno,cno 不同的对数]=count(*) from (select count(*)as B from STC group by tno, cno)A
```

函数约束的结果

	函数约束时间（ms）
1	38310

	成功次数
1	1980

	失败次数
1	98020

	tno, cno不同的对数
1	1000

	触发器时间（ms）
1	40780

触发器的结果

	触发器时间（ms）
1	40780

	成功次数
1	2018

	失败次数
1	97982

	tno, cno不同的对数
1	1000

4. 改进

为了让函数约束和触发器执行效率的比较更为准确，我们重新写了触发器的查询语句。这一次我们选择不适用游标的方式，直接从 inserted 表中获取数据，然后进行范式约束代码如下。

```
 2 create trigger Check3NF_trigger on STC after insert as
 3     if ( @@ROWCOUNT = 0 ) return
 4     if ( @@ROWCOUNT > 1 )
 5     begin
 6         return
 7     end
 8
 9     declare @sno int, @tno int, @cno int
10     select @sno=sno from inserted
11     select @tno=tno from inserted
12     select @cno=cno from inserted
13
14     if exists(
15     select *
```

```
    from STC
    where tno = @tno and cno <> @cno)
    begin
        rollback tran
        return
    end

    if exists(
    select *
    from STC
    where sno = @sno and cno = @cno and tno <> @tno)
    begin
        rollback tran
        return
    end
```

而在这种情况下，我们得到了如下的运行结果

	函数约束时间（ms）
1	33103

	成功次数
1	2014

	失败次数
1	97986

	tno，cno不同的对数
1	1000

	触发器时间（ms）
1	31250

	成功次数
1	1972

	失败次数
1	98028

	tno，cno不同的对数
1	1000

可见函数约束和触发器执行内容几乎一致的时候，触发器的执行效率更高。