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

[一、 函数约束练习 2](#_Toc452110526)

[1. 题目 2](#_Toc452110527)

[2. 代码与分析 2](#_Toc452110528)

[3. 测试与结果 4](#_Toc452110529)

[一、 触发器练习 5](#_Toc452110530)

[1. 题目 5](#_Toc452110531)

[2. 代码与分析 5](#_Toc452110532)

[3. 测试与结果 8](#_Toc452110533)

[二、 函数依赖练习 11](#_Toc452110534)

[1. 题目 11](#_Toc452110535)

[2. 分析与代码 11](#_Toc452110536)

[3. 结果 12](#_Toc452110537)

[4. 改进 13](#_Toc452110538)

约束设计

# 函数约束练习

## 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

## 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**

13 **as**

14 **begin**

15 --判断身份证号码长度是否为18

16 **if**(len(@IDNumber)<>18)

17 **return** 0

18

19 --定义最后一位映射表7

20 **declare** @MapTable **table**(value **char**, idn **int**)

21 **insert into** @MapTable (value, idn) **values** ('1', 0)

22 **insert into** @MapTable (value, idn) **values** ('0', 1)

23 **insert into** @MapTable (value, idn) **values** ('X', 2)

24 **insert into** @MapTable (value, idn) **values** ('9', 3)

25 **insert into** @MapTable (value, idn) **values** ('8', 4)

26 **insert into** @MapTable (value, idn) **values** ('7', 5)

27 **insert into** @MapTable (value, idn) **values** ('6', 6)

28 **insert into** @MapTable (value, idn) **values** ('4', 7)

29 **insert into** @MapTable (value, idn) **values** ('3', 8)

30 **insert into** @MapTable (value, idn) **values** ('2', 9)

31

32 --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

33 **declare** @CalcTable **table**(result **int**)

34 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 1, 1)) \* 7)

35 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 2, 1)) \* 9)

36 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 3, 1)) \* 10)

37 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 4, 1)) \* 5)

38 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 5, 1)) \* 8)

39 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 6, 1)) \* 4)

40 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 7, 1)) \* 2)

41 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 8, 1)) \* 1)

42 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 9, 1)) \* 6)

43 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 10, 1)) \* 3)

44 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 11, 1)) \* 7)

45 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 12, 1)) \* 9)

46 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 13, 1)) \* 10)

47 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 14, 1)) \* 5)

48 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 15, 1)) \* 8)

49 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 16, 1)) \* 4)

50 **insert into** @CalcTable **values** (CONVERT(**int**, SUBSTRING(@IDNumber, 17, 1)) \* 2)

51

52 --计算余数

53 **declare** @checksum **int**

54 **select** @checksum = sum(result) **from** @CalcTable

55 **set** @checksum = @checksum % 11

56

57 --取出映射

58 **declare** @mapvalue **char**

59 **select** @mapvalue=value **from** @MapTable **where** idn = @checksum

60

61 --判断是否符合映射

62 **if**(SUBSTRING(@IDNumber, 18, 1) <> @mapvalue )

63 **return** 0

64

65 **return** 1

66 **end**;

67 **go**

最后将函数约束添加到数据库表

80 --建立数据库表并添加约束

81 **create table** ID (IDNumber **char**(18) **check**(dbo.CheckIDNumber(IDNumber)=1));

82 **go**

83

## 测试与结果

84 --测试样例

85 **insert into** ID **values**('52263519830114890X');

86 **insert into** ID **values**('130231252361241241');

87 **insert into** ID **values**('522635198708184662');

88 **insert into** ID **values**('522635198708184612');

89 **go**

90

91 --输出结果

92 **select** \* **from** ID

93 **go**

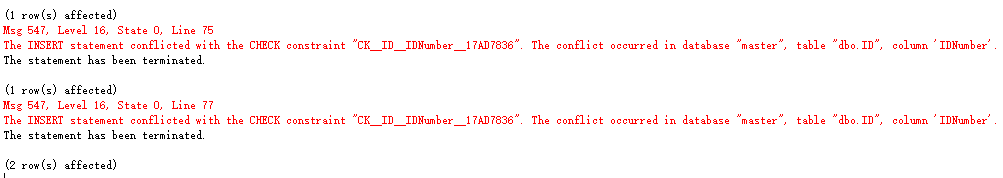
在测试中，我们从百度搜索出正确的身份证号作为数据，其中正确的有

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



可以看到第二条和第四条insert语句触发了check约束

# 触发器练习

## 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

## 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**

## 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

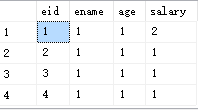
63 **insert into** Emp **values**(1,'1',1,2)

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

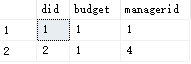


69 --添加部门

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



74 --分配工作

75 **insert into** Works **values**(1,1, 0.5)

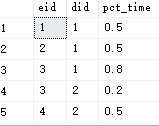
76 **insert into** Works **values**(2,1, 0.5)

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

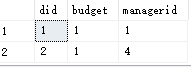


此时我们可以看到部门2的主管是4号员工，但是4号员工的工资和他管理的员工工资一样，这是违规的，但是当我们分配工作的时候，只对Works表做了操作，并不会触发Dept的check约束，所以Dept此仍不知道内部数据发生了违规。

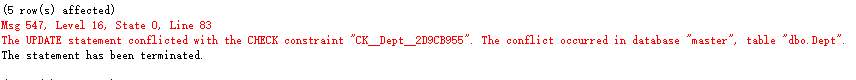
82 --重新分配主管

83 **update** Dept **set** managerid = 3 **where** did = 2

84 **select** \* **from** Dept



如果此时对Dept表的部门2做操作，会触发check约束，



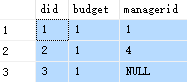
但我们看到虽然managerid没有改变为3，但是它仍保持为4，这意味着check约束只会对操作检查，并不会对已经在表中数据做检查

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

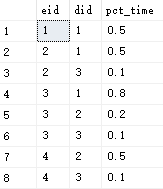


90 **insert into** Works **values** (2,3,0.1)

91 **insert into** Works **values** (3,3,0.1)

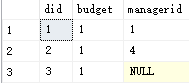
92 **insert into** Works **values** (4,3,0.1)

93 **select** \* **from** Works

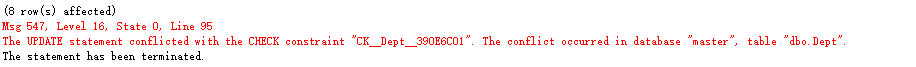


95 **update** Dept **set** managerid = 3 **where** did = 3

96 **select** \* **from** Dept



可以看到此时触发了check约束，同时数据库拒绝了操作



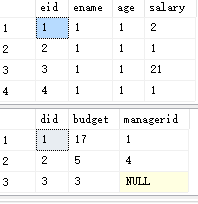
触发器的测试是基于函数约束测试的

2 --测试样例

3 **update** Emp **set** salary = 21 **where** eid = 3

4 **select** \* **from** Emp

5 **select** \* **from** Dept



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。

# 函数依赖练习

## 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

## 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。

## 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

1 --清除缓存

2 **dbcc dropcleanbuffers**

3 **dbcc freeproccache**

4 **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。

55 --随机插入

56 **declare** @totaltime **int**

57 **set** @totaltime = 100000

58 **declare** @starttime **datetime**

59 **set** @starttime=GETDATE()

60 **declare** @failtime **int**

61 **set** @failtime = 0

62 **declare** @i **int**

63 **set** @i = 0

64 **while** @i < @totaltime

65 **begin**

66 **begin** try

67 **insert into** STC **values**(

68 FLOOR(RAND()\*10000),

69 FLOOR(RAND()\*1000),

70 FLOOR(RAND()\*100))

71 **end** try

72 **begin** catch

73 **set** @failtime = @failtime + 1

74 **end** catch

75 **set** @i = @i + 1

76 **end**

77 --select \* from STC

78 **select** [函数约束时间（ms）]=datediff(ms, @starttime, getdate())

79 **select** [成功次数]=@totaltime - @failtime

80 **select** [失败次数]=@failtime

81 **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



触发器的结果



## 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** \*

16 **from** STC

17 **where** tno = @tno **and** cno <> @cno)

18 **begin**

19 **rollback tran**

20 **return**

21 **end**

22

23 **if exists**(

24 **select** \*

25 **from** STC

26 **where** sno = @sno **and** cno = @cno **and** tno <> @tno)

27 **begin**

28 **rollback tran**

29 **return**

30 **end**

而在这种情况下，我们得到了如下的运行结果



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。