实习二 数据库约束设计

成员：吴悦欣1900012946 赵衔麟1900013063 张介宾1900013359

```
%load_ext sql
```

```
import pymysql
pymysql.install_as_MySQLdb()
%sql mysql://stu1900013063:stu1900013063@162.105.146.37:43306
```

```
%sql use stu1900013063;
```

本次实习的目标是利用函数、存储过程、触发器实现一些较为复杂的业务约束，分为两个练习。练习一需要实现产生身份证校验码的函数、构造身份证号的存储过程，练习二需要实现触发器，完成对股票表中股票信息的更新。

练习一 生成身份证号

1. 生成身份证校验码

身份证号共有18位，最后一位是校验码，它需要根据前17位计算得出。计算的方法是，将前17位分别乘不同的系数（7、9、10、5、8、4、2、1、6、3、7、9、10、5、8、4、2），再把17个相乘的结果相加并除以11，余数0-10分别对应校验码1、0、X、9、8、7、6、5、4、3、2。

根据上述计算过程，我们可以写出getCheckDigit函数，它的输入参数是17位的字符串bef，返回值是1个字符，可将其视作长度为1的字符串。

在这个函数中，用substr(bef, k, 1)可以获得第k位号码，与系数相乘后，它会被强制转换为整数类型。mod函数则用于取模运算，我们能够得到总和模11的结果。最后，使用"case... when... then..."语句来判断每种情况，并返回对应的校验码。

```
%%sql
drop function if exists getCheckDigit;

create function getCheckDigit(bef varchar(17))
returns varchar(1)
return (case
mod(substr(bef,1,1)*7+substr(bef,2,1)*9+substr(bef,3,1)*10+substr(bef,4,1)*5

+substr(bef,5,1)*8+substr(bef,6,1)*4+substr(bef,7,1)*2+substr(bef,8,1)*1

+substr(bef,9,1)*6+substr(bef,10,1)*3+substr(bef,11,1)*7+substr(bef,12,1)*9

+substr(bef,13,1)*10+substr(bef,14,1)*5+substr(bef,15,1)*8+substr(bef,16,1)*4+su
bstr(bef,17,1)*2,
          11)
       when 0 then '1'
       when 1 then '0'
       when 2 then 'x'
       when 3 then '9'
       when 4 then '8'
```

```
        when 5 then '7'
        when 6 then '6'
        when 7 then '5'
        when 8 then '4'
        when 9 then '3'
        when 10 then '2'
        end
)
```

对getCheckDigit函数进行测试，下面给出两个例子：

```
# 例1
%sql select getCheckDigit('610302200010022O4');
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

getCheckDigit('610302200010022O4')
3

```
# 例2
%sql select getCheckDigit('610103197310022O5');
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

getCheckDigit('610103197310022O5')
X

2. 构造身份证号

我们已经可以根据前17位数字来计算最后一位校验码，接下来则要根据个人信息来产生前17位号码。每一位的含义是这样的：1、2位，3、4位，5、6位分别代表所在省、市、区县；7-14位代表出生年月日；15、16位是所在地派出所的代码，17位的奇偶性代表性别。

构造身份证号的函数名为generate_ID，输入参数有5个：省份province，城市city，区县district，出生年月日birth，性别gender，都是字符串类型。输出为ans，是18位字符串。

平台上dataset数据集中的xzqh是行政区划表（有两列，分别是name和code），省市区都在同一张表里。省对应的code的3-6位为0，市对应的code的5-6位为0，区县的1-2位与所在省份code的前2位相同，1-4位与所在城市code的前4位相同。当我们查询的时候，为防止区县的名字重复，需要对前4位进行检查。在函数中，首先选择出name与参数province、city、district相同的code，分别放入tmp_1, tmp_2, tmp_3表中，属性分别命名为c_p,c_c,c_d。然后，为了确保唯一性，我们从c_d中选出与c_p和c_c前两位相同、且与c_c的3-4位相同的码，这样，身份证号的前6位就确定了。

7-14位的出生年月日直接由birth参数给出，用concat函数拼接即可。15-16位的派出所代码是随机生成的，利用rand函数实现。

对于第17位，偶数代表女性，奇数代表男性。我们先判断gender参数，然后利用rand函数生成随机的偶数或奇数。最终，调用上面实现的getCheckDigit函数计算出校验码，拼接得到完整的身份证号。

```
%%sql
drop procedure if exists generate_ID;

CREATE PROCEDURE generate_ID
(IN province varchar(10), IN city varchar(10), IN district varchar(10),
 IN birth varchar(8), IN gender varchar(10), OUT ans varchar(18))
BEGIN
    declare district_code varchar(6);
    declare r_c1 varchar(1);
    declare r_c2 varchar(1);
    declare gender_num varchar(1);

    /* 选出name与province, city, district参数相同的code */
    with tmp1 as (select code c_p from dataset.xzqh where name = province)
        ,tmp2 as (select code c_c from dataset.xzqh where name = city)
        ,tmp3 as (select code c_d from dataset.xzqh where name = district)

    /* 验证1、2位和3、4位，确保c_d的唯一性。最终的前6位存入district_code变量 */
    select c_d from tmp1, tmp2, tmp3 where
    (
        substr(c_p,0,2) = substr(c_c,0,2)
        and substr(c_c,0,2) = substr(c_d,0,2)
        and substr(c_c, 2, 2) = substr(c_d, 2, 2)
    ) into district_code;
    if district_code is not null then
        /* 随机生成15、16位的派出所代码 */
        select cast(floor(rand() * 10) as char(1)) into r_c1;
        select cast(floor(rand() * 10) as char(1)) into r_c2;
        set ans = concat(district_code, birth, r_c1, r_c2);

        /* 女性为0、2、4、6、8，男性为1、3、5、7、9 */
        if gender = '女' then
            begin
            select cast(floor(rand() * 5) * 2 as char(1)) into gender_num;
            set ans = concat(ans, gender_num);
            end;
        else
            begin
            select cast(floor(rand() * 5) * 2 + 1 as char(1)) into gender_num;
            set ans = concat(ans, gender_num);
            end;
        end if;

        /* 调用getCheckDigit产生第18位，拼接 */
```

```
        set ans = concat(ans, getCheckDigit(ans));
    end if;
END;
```

对generate_ID函数进行测试，下面给出两个例子：

```
%sql set @ans = '';
```

```
# 例1
%sql call generate_ID('浙江省', '台州市', '椒江区', '20010725', '女', @ans);
%sql select @ans;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

@ans
331002200107257081

```
# 例2
%sql call generate_ID('陕西省', '宝鸡市', '渭滨区', '20001002', '女', @ans);
%sql select @ans;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

@ans
610302200010029106

练习二 触发器

题目的场景是股票交易，有两张表：股票表和交易表。股票表名为my_stock，具有的属性是：股票编号stock_id，数量volume，持仓平均价格avg_price，利润profit。交易表名为trans，具有的属性是：交易编号trans_id，股票编号stock_id，交易日期trans_date，成交价格price，成交数量amount，买入还是卖出sell_or_buy。

我们首先创建这两张表：

```
%%sql
/* 创建股票表 */
set @@foreign_key_checks=0;
drop table if exists my_stock;

CREATE TABLE my_stock
(
    stock_id int PRIMARY KEY,
    volume int NOT NULL,
    avg_price float,
    profit int NOT NULL
);
set @@foreign_key_checks=1;
```

```
%%sql
/* 创建交易表 */
set @@foreign_key_checks=0;
drop table if exists trans;

CREATE TABLE trans
(
    trans_id int auto_increment,
    stock_id int NOT NULL,
    trans_date int UNIQUE NOT NULL,
    price int NOT NULL,
    amount int NOT NULL,
    sell_or_buy enum('B','S'),
    primary key(trans_id,stock_id)
);
set @@foreign_key_checks=1;
```

分析

每次往trans表中插入一条交易记录时，我们都要更新my_stock表。

如果本次交易涉及的股票不在my_stock表中，则在trans表中插入一条新记录，profit置为0。如果股票本身就在my_stock表中，则要更新该股票的volume、avg_price、profit:

- 数量volume的计算很容易，根据买入或卖出进行加减即可。唯一要特别注意的是，如果卖出的数量大于当前该股票的volume，应该禁止这一交易。
- 持仓平均价格avg_price也有专门的计算公式：

$$avg\_price = \frac{volume*avg\_price+price*amount}{volume+amount}$$

- 利润profit的计算稍显复杂，当交易的类型是“卖出”时，需要将其与最远的买入交易进行匹配，记作交易A。如果A中买入的股票数量大于等于本次交易的amount，则根据两次交易的差价和本次交易的数量来更新profit。否则，将交易A的股票数量卖空，计算profit，对于余下还没卖出的数量，继续找下一次买入交易，例如B，对B进行同样的检查和计算。重复这个过程，直到本次交易的数量全部卖出。

我们已经分析了与更新my_stock表相关的操作，可以通过触发器来实现。但在此之前，还面临一个问题，trans表要记录所有的交易，不能随意改变之前的记录，但计算profit时却需要更改之前的记录。为了解决这一矛盾，我们另外建了一张表trans_buy，记录所有的买入交易，每次往trans表中插入记录时，也同时往trans_buy表中插入同样的记录，并为此设立一个触发器：trans_buy_copy。

```
%%sql
/* 创建买入交易表，相比trans表，无需记录trans_date和sell_or_buy */
set @@foreign_key_checks=0;
drop table if exists trans_buy;

CREATE TABLE trans_buy
(
    trans_id int,
    stock_id int NOT NULL,
    price int NOT NULL,
    amount int NOT NULL,
    primary key(trans_id,stock_id)
);
set @@foreign_key_checks=1;
```

```
%%sql
/* 触发器，每次往trans中插入一条买入记录时，也往trans_buy_copy中插入同样的记录 */
drop trigger if exists trans_buy_copy;

create trigger trans_buy_copy
after insert on trans
for each row
begin
    if new.sell_or_buy = 'B' then
        INSERT INTO trans_buy(trans_id, stock_id, price, amount)
VALUES(new.trans_id, new.stock_id, new.price, new.amount);
    end if;
end;
```

最后，我们构造用于更新my_stock表的触发器：trans_first。上文已经包括了具体的思路，给出了volume、avg_price、profit的计算方法。由于profit的计算比较复杂，而且要更新trans_buy表，所以把它的计算封装到一个存储过程change_profit中。

```
%%sql

drop procedure if exists change_profit;
/* 三个参数分别是交易涉及的股票编号、卖出的量、卖出价格 */
CREATE PROCEDURE change_profit(sell_stock_id INT, vol INT, sell_p INT)
BEGIN
    declare sell_vol INT;  /* 还需要卖出的量，在循环中不断减少，初值为vol */
    declare sell_trans_id INT;  /* 买入交易的编号 */
    declare sell_vol_single INT;  /* 买入交易目前还剩下的量 */
    declare sell_p_single INT;  /* 买入交易的价格 */
    declare buy_costs INT;  /* 当初买入时的花销，在循环中不断累加，初值为0 */
    set sell_vol = vol;
    set buy_costs = 0;

    while sell_vol > 0 do
        /* 从trans_buy中找出最久远的买入交易，获得余量和买入价格 */
```

```
        select min(trans_id) from trans_buy where stock_id = sell_stock_id into
sell_trans_id;
        select amount from trans_buy where stock_id = sell_stock_id and trans_id
= sell_trans_id into sell_vol_single;
        select price from trans_buy where stock_id = sell_stock_id and trans_id
= sell_trans_id into sell_p_single;

        /* 如果这个买入交易的余量不够卖，就更新还需要卖出的量和买入时的花销，并把trans_buy中
的这条记录删除 */
        if sell_vol_single <= sell_vol then begin
            set sell_vol = sell_vol - sell_vol_single;
            set buy_costs = buy_costs + sell_vol_single * sell_p_single;
            delete from trans_buy where stock_id = sell_stock_id and trans_id =
sell_trans_id;
        end;
        /* 足够卖，就可以结束了。更新这个买入交易的余量，并累加买入时的花销buy_costs */
        else begin
            update trans_buy set amount = amount - sell_vol where stock_id =
sell_stock_id and trans_id = sell_trans_id;
            set buy_costs = buy_costs + sell_vol * sell_p_single;
            set sell_vol = 0;
        end;
        end if;
    end while;

    /* 更新卖出后的股票利润 */
    update my_stock set profit = profit - buy_costs + vol * sell_p where
stock_id = sell_stock_id;
END;
```

```
%%sql

drop trigger if exists trans_first;

create trigger trans_first
after insert on trans
for each row
begin
    declare old_vol INT;

    /* 第一次买入某种股票，往my_stock表插入一条新记录 */
    if (new.stock_id not in (select stock_id from my_stock) and new.sell_or_buy
= 'B') then
        INSERT INTO my_stock(stock_id, volume, avg_price, profit) VALUES
(new.stock_id, new.amount, new.price, 0);

    /* 本来没有某种股票，却要卖。这种操作应予以报错，delete会报错 */
    elseif (new.stock_id not in (select stock_id from my_stock) and
new.sell_or_buy = 'S') then
        DELETE from trans where new.stock_id = stock_id and new.trans_id =
trans_id;
    else
    begin
        /* 买入，只更新avg_price和volume */
        if new.sell_or_buy = 'B' then begin
            UPDATE my_stock set avg_price = (volume * avg_price + new.price *
new.amount)/(volume + new.amount) where stock_id = new.stock_id;
```

```
            UPDATE my_stock set volume = volume + new.amount where stock_id =
new.stock_id;
        end;
        /* 卖出，要先判断现有的股票数量够不够 */
        else begin
            select volume from my_stock where stock_id = new.stock_id into
old_vol;
            /* 不够，就拒绝交易（delete会报错） */
            if old_vol < new.amount then
                DELETE from trans where new.stock_id = stock_id and new.trans_id
= trans_id;
            else
            /* 够，除了更新avg_price和volume，还要调用change_profit存储过程来更新profit
*/
            begin
                call change_profit(new.stock_id, new.amount, new.price);
                UPDATE my_stock set avg_price = (volume*avg_price -
new.price*new.amount)/(volume - new.amount) where stock_id = new.stock_id;
                UPDATE my_stock set volume = volume - new.amount where stock_id
= new.stock_id;
            end;
            end if;
        end;
        end if;
    end;
    end if;
end;
```

测试

下面我们使用老师提供的7条数据进行测试，并给出必要的运行结果：

```
# 数据1和数据2，都是买入操作
%sql insert into trans(stock_id, trans_date, price, amount, sell_or_buy)
values(1, 1, 10, 1000, 'B');
%sql insert into trans(stock_id, trans_date, price, amount, sell_or_buy)
values(1, 2, 11, 500, 'B');
```

```
# 检查my_stock，可以看到共买入1500股，持仓均价是(1000*10+500*11)/(1000+500) = 10.3333，
因为还没卖出，所以利润为0
%sql select * from my_stock;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

stock_id	volume	avg_price	profit
1	1500	10.3333	0

```
# 数据3，卖出800股
%sql insert into trans(stock_id, trans_date, price, amount, sell_or_buy)
values(1, 3, 12, 800, 'S');
```

```
# 检查my_stock，发现volume减少了800，利润为(12-10)*800 = 1600
%sql select * from my_stock;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

stock_id	volume	avg_price	profit
1	700	8.42857	1600

```
# 检查自定义的表trans_buy，trans_id为1的那笔交易原先的amount是1000，现在减少了800
%sql select * from trans_buy;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
2 rows affected.
```

trans_id	stock_id	price	amount
1	1	10	200
2	1	11	500

```
# 数据4，卖出1000股，但实际上只持有700股，应拒绝此操作，报错
%sql insert into trans(stock_id, trans_date, price, amount, sell_or_buy)
values(1, 4, 12, 1000, 'S');
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
(pymysql.err.OperationalError) (1442, "Can't update table 'trans' in stored
function/trigger because it is already used by statement which invoked this
stored function/trigger.")
[SQL: insert into trans(stock_id, trans_date, price, amount, sell_or_buy)
values(1, 4, 12, 1000, 'S' );]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

```
# 检查trans表，发现这笔非法交易确实没有插入表
%sql select * from trans;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
3 rows affected.
```

trans_id	stock_id	trans_date	price	amount	sell_or_buy
1	1	1	10	1000	B
2	1	2	11	500	B
3	1	3	12	800	S

```
# 数据5，买入1000股
%sql insert into trans(stock_id, trans_date, price, amount, sell_or_buy)
values(1, 5, 9, 1000, 'B');
```

```
# 检查trans_buy表，发现插入了新的买入交易。当然，这条交易也插入了trans表
%sql select * from trans_buy;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
3 rows affected.
```

trans_id	stock_id	price	amount
1	1	10	200
2	1	11	500
5	1	9	1000

```
# 数据6，卖出800股
%sql insert into trans(stock_id, trans_date, price, amount, sell_or_buy)
values(1, 6, 12, 800, 'S');
```

```
# 一共800股，trans_id为1的交易提供200股，trans_id为2的交易提供500股，trans_id为5的交易提供100股。前两笔交易的amount变为0了，从trans_buy表删除
%sql select * from trans_buy;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

trans_id	stock_id	price	amount
5	1	9	900

```
# 利润增加了2800-1600 = 1200。计算方法是：200*(12-10)+500*(12-11)+100*(12-9) = 400+500+300 = 1200
%sql select * from my_stock;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

stock_id	volume	avg_price	profit
1	900	5.88889	2800

```
# 数据7，卖出800股
%sql insert into trans(stock_id, trans_date, price, amount, sell_or_buy)
values(1, 7, 7, 800, 'S');
```

```
# 最后检查trans表，可以回顾所有的合法交易
%sql select * from trans;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
6 rows affected.
```

trans_id	stock_id	trans_date	price	amount	sell_or_buy
1	1	1	10	1000	B
2	1	2	11	500	B
3	1	3	12	800	S
5	1	5	9	1000	B
6	1	6	12	800	S
7	1	7	7	800	S

```
# 最后检查trans_buy表，trans_id为5的交易的amount只剩100，之前是900
%sql select * from trans_buy;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

trans_id	stock_id	price	amount
5	1	9	100

```
# 最后检查my_stock表，发现profit减少了2800-1200 = 1600，这是因为trans_id为7的交易亏了800*(9-7) = 1600
%sql select * from my_stock;
```

```
 * mysql://stu1900013063:***@162.105.146.37:43306
1 rows affected.
```

stock_id	volume	avg_price	profit
1	100	-3.0	1200