实习二——数据库约束设计

小组成员：龚治溧 琚敬成 袁昊 周哲恺

实验一：身份证生成与校验

实验内容

本次实习的任务是利用身份证号的前17位生成一个校验码，并根据实际输出的省份、城市、性别等信息输出一个身份证号。

实习涉及到的课程内容有数据库的查询、用`sql`编写函数、用`sql`实现存储过程。

实习内容一：编写一个函数，能够根据输入的前17位，生成一个校验码

在构造`funtion`时主要运用了`substring`和`cast`两个内置函数，将字符串类型的身份证转化为数字并按照要求进行计算，最终得出结果。

```
%%sql
#构造生成校验码的函数
drop function if exists check_generalize;#删除已有的防止冲突
CREATE FUNCTION check_generalize(number CHAR(17))
RETURNS CHAR
BEGIN
    DECLARE a INT;
    #提取身份证号前十七位的各个数字，并进行相应运算
    SET a=(CAST((SUBSTRING(number,1,1)) AS UNSIGNED)*7+CAST((SUBSTRING(number,2,1)) AS
UNSIGNED)*9+
    CAST((SUBSTRING(number,3,1)) AS UNSIGNED)*10+CAST((SUBSTRING(number,4,1)) AS
UNSIGNED)*5+
    CAST((SUBSTRING(number,5,1)) AS UNSIGNED)*8+CAST((SUBSTRING(number,6,1)) AS
UNSIGNED)*4+
    CAST((SUBSTRING(number,7,1)) AS UNSIGNED)*2+CAST((SUBSTRING(number,8,1)) AS
UNSIGNED)*1+
    CAST((SUBSTRING(number,9,1)) AS UNSIGNED)*6+CAST((SUBSTRING(number,10,1)) AS
UNSIGNED)*3+
    CAST((SUBSTRING(number,11,1)) AS UNSIGNED)*7+CAST((SUBSTRING(number,12,1)) AS
UNSIGNED)*9+
    CAST((SUBSTRING(number,13,1)) AS UNSIGNED)*10+CAST((SUBSTRING(number,14,1)) AS
UNSIGNED)*5+
    CAST((SUBSTRING(number,15,1)) AS UNSIGNED)*8+CAST((SUBSTRING(number,16,1)) AS
UNSIGNED)*4+
    CAST((SUBSTRING(number,17,1)) AS UNSIGNED)*2)%11;
    #根据算出来的结果产生最后的校验码
    IF a=2
        THEN RETURN 'X';
    ELSEIF a=0
        THEN RETURN '1';
    ELSEIF a=1
```

```
        THEN RETURN '0';
    ELSE
        RETURN CAST(12-a AS CHAR);
    END IF;
END;
```

实习内容二：构造一个存储过程，它接受的参数包括(省、市、区、出生日期、性别)，返回一个身份证号

主要是考察储存过程的构建，内在逻辑并不复杂，主要是利用xzqh数据库来进行行政代码的查询。因为在dataset下只有select权限，需要在自己的数据库下复制一个同样的表。

```
%%sql
use stu1700010762;
create table xzqh like dataset.xzqh;
insert into xzqh select* from dataset.xzqh;
```

构造的储存过程如下：

```
drop procedure if exists generalize;
CREATE PROCEDURE generalize(In province varchar(8),In city varchar(8),In county
varchar(8),In birthday char(8),In sex char(1),OUT id char(18))#generalize(省、市、区、出生
日期、性别、(输出)身份证号)
BEGIN
    DECLARE province_id CHAR(2);
    DECLARE city_id CHAR(2);
    DECLARE county_id CHAR(2);
    DECLARE police_id CHAR(2) default "00";
    DECLARE sex_id CHAR(1);
    DECLARE seventeen CHAR(17);
    set police_id=ceiling(rand()*100);#派出所代码随机生成
    SET province_id=SUBSTRING(
       (select code
        from xzqh
        where name=province),1,2);#查询省份对应的前两位数字
    if province_id='11'or'12'or'31'or'50'#查询直辖市对应的第3、4位数字
    THEN SET city_id=SUBSTRING(
        (select code
        from xzqh
        where name=county and SUBSTRING(code,1,2)=province_id),3,2);
    else     SET city_id=SUBSTRING(#查询非直辖市城市对应的第3、4位数字
       (select code
        from xzqh
        where name=city and SUBSTRING(code,1,2)=province_id),3,2);
    end if;
    SET county_id=SUBSTRING(#查询所在区、县一级行政区对应的5、6位数字
        (select code
        from xzqh
        where name=county and SUBSTRING(code,1,2)=province_id and
SUBSTRING(code,3,2)=city_id),5,2);
    if sex='M'#用数字3代表男性、用数字2代表女性
    THEN SET sex_id='3';
    else SET sex_id='2';
    end if;
    SET seventeen=CONCAT(province_id,city_id,county_id,birthday,police_id,sex_id);#把前
十七位合在一起
    SET id=CONCAT(seventeen,check_generalize3(seventeen));#生成最后的身份证号码
END;
```

实验结果

- 校验码函数

  ```
  %%sql
  #测试一下生成校验码函数
  select check_generalize('62012119990610002')
  ```

check_generalize4('62012119990610002')
9

- 储存过程

  ```
  %%sql
  call generalize('甘肃省','兰州市','永登县','19990610','M',@id);
  select @id;
  ```

@id
620121199906108338

  ```
  %%sql
  call generalize('重庆市','重庆市','垫江县','20010726','M',@id);
  select @id;
  ```

@id
500231200107264533

  ```
  %%sql
  call generalize('重庆市','重庆市','江北区','19690706','F',@id);
  select @id;
  ```

@id
500105196907061120

实验二：触发器设计

实验内容

为股票相关数据库设计一个触发器，使得增添交易记录时可以同时更新用户持有的股票，并计算出用户在整个交易过程当中的持仓均价、获得利润等信息。

主要考察——触发器的设计、游标的使用、复杂功能的实现。

实验准备

首先连接服务器。在本次试验中，我们选择同时在学校服务器端和本地端进行测试。

```
mysql://stu1900013002:stu1900013002@162.105.146.37:43306
use stu1900013002;
```

根据题目要求，我们将创建如下两张table。

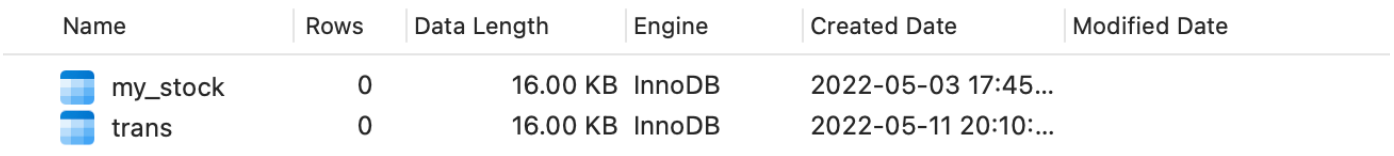
```
create table my_stock
(
    stock_id int not null,
    volume int,
    avg_price float,
    profit float,
    primary key (stock_id)
);
```

其一是`my_stock`表，以`stock_id`为主码，每一条记录有规模、持仓均价、利润三条属性，代表着用户所持有的所有股票信息。

```
create table trans
(
    trans_id int not null,
    stock_id int not null,
    tdate int,
    price float,
    amount int,
    sale_or_by char(1),
    primary key (trans_id),
    check (sale_or_by = 'S' or sale_or_by='B')
);
```

其一是`trans`表，以`trans_id`为主码，每一条记录代表着一次交易信息。其中`tdate`属性可以取`datetime`类型，但是为了构造方便在实验中我们暂时将其认作`int`类型。同时我们需要保证`sale_or_buy`是一个双指属性，避免出错。

```
show tables;
```



Name	Rows	Data Length	Engine	Created Date	Modified Date
my_stock	0	16.00 KB	InnoDB	2022-05-03 17:45...	
trans	0	16.00 KB	InnoDB	2022-05-11 20:10:...	

上图是简单的使用Navicat软件展示的table列表。

实验开始：触发器的设计

整体代码如下所示，我们将逐步分解析设计思路

```
drop trigger if exists test;
delimiter $
create trigger test before insert on trans for each row
begin
  declare remain int;
  declare amount_ int;
  declare price_ float;
  declare sold int;
  declare profit_ float;
  declare cursor1 cursor for (select amount,price from trans where
stock_id=new.stock_id and sale_or_by="B");
  set profit_=0;
  if not exists(select * from my_stock where my_stock.stock_id=new.stock_id) then
    if new.sale_or_by="B" then
      insert into my_stock values (new.stock_id,new.amount,new.price,0);
    ELSE
      signal sqlstate 'HY000' set message_text="not enough for buy";
    END IF;
  ELSE
    IF new.sale_or_by="B" THEN
      UPDATE my_stock SET volume=volume+new.amount,
                          avg_price=((volume-
new.amount)*avg_price+new.amount*new.price)/(volume)
                          WHERE stock_id=new.stock_id;
    ELSE
      IF EXISTS(SELECT * from my_stock where stock_id=new.stock_id and
volume>new.amount) THEN
        set remain=new.amount;
        set sold=(select sum(amount) from trans where stock_id=new.stock_id and
sale_or_by="S");
        if sold is null then set sold=0; end if;
        open cursor1;
        while(remain>0)do
          fetch next from cursor1 into amount_, price_;
          if(amount_<sold) then
            set sold=sold-amount_;
          else
            if(remain<amount_-sold) then
              set profit_=profit_+remain*price_;
              set remain=0;
              set sold=0;
            else
              set profit_=profit_+(amount_-sold)*price_;
              set remain=remain-amount_+sold;
              set sold=0;
            end if;
          end if;
        end while;
        close cursor1;
        UPDATE my_stock SET volume=volume-new.amount,
                            avg_price=((volume+new.amount)*avg_price-profit_)/(volume),
                            profit=profit+(new.amount*new.price-profit_)
                            WHERE stock_id=new.stock_id;
      ELSE
        signal sqlstate 'HY000' set message_text="not enough for buy";
      END IF;
    END IF;
  END IF;
end $
delimiter ;
show triggers;
```

首先是整体框架——最基本的触发器创建语句比较简单，具体情况详见注释，在此不加赘述。

```
drop trigger if exists test; #删除已有的触发器便于修改更新
delimiter $ #切换终止符号
create trigger test #命名为test
before insert on trans #在插入语句之前
for each row #对每一行
begin
#TODO: 函数主体语句
end $
delimiter ; #改回终止符号
show triggers; #检查触发器创建
```

触发器的主体思路如下:

- 如果`my_stock`中有新加的`stock_id`
  - 如果是买入则新建词条

    ```
    insert into my_stock values (new.stock_id,new.amount,new.price,0);
    ```

  - 如果是卖出，则引出一条中断返回，抛弃本次插入。（在Mysql中不支持rollback等操作，利用中断能从更底层进行抛弃。）

    ```
    signal sqlstate 'HY000' set message_text="not enough for buy";
    ```

- 否则，意味着用户已经持仓
    - 如果是买入，对持仓信息按照公式进行更新。

      ```
      UPDATE my_stock SET volume=volume+new.amount,
                                  avg_price=((volume-
      new.amount)*avg_price+new.amount*new.price)/(volume)
                                  WHERE stock_id=new.stock_id;
      ```

        - 这里均价变化公式使用的是ppt上给出的加仓公式$avg_price = \frac{volume \times avg_price + price \times amount}{volume + amount}$
        - 利润保持不变，不用更新
        - `volume`增加
    - 如果是卖出，则要计算利润，并对均价、规模做出相应调整
        - 利润的计算比较复杂，因为要优先卖出最早买入的股票，所以我们考虑利用cursor进行查询。步骤如下
            - 定义一些中间变量：

              ```
              declare remain int; #目前尚未找到买入价格的股票数目
              declare amount_ int; #用于储存cursor的数据
              declare price_ float; #用于储存cursor的数据
              declare sold int; #已经卖出的总量
              declare profit_ float; #当前卖出股票买入的总成本
              ```

            - 定义游标由于逐条访问，这里我们简单的认为越早加入`trans`的数据真实交易时间越早：

              ```
              declare cursor1 cursor for (select amount,price from trans where
              stock_id=new.stock_id and sale_or_by="B"); #用于获取购买记录
              ```

            - 主体实现部分：

              ```
              set remain=new.amount;
              set sold=(select sum(amount) from trans
                        where stock_id=new.stock_id and sale_or_by="S"); #根据定
              义获取初始值
              if sold is null then set sold=0;
              end if; #制定初始0，防止为空
              open cursor1; #打开cursor
              while(remain>0)do
                fetch next from cursor1 into amount_, price_; #读取接下来的数据
                if(amount_<sold) then
                  set sold=sold-amount_; #如果之前已经被买走就更新sold
                else #否则从此条开始属于当前卖出对应的买入记录
                  if(remain<amount_-sold) then
                    set profit_=profit_+remain*price_; #更新原价成本
                    set remain=0;
                    set sold=0;
                  else
              ```

```
      set profit_=profit_+(amount_-sold)*price_; #更新原价成本
      set remain=remain-amount_+sold;
      set sold=0;
    end if; #循环知道remain被全部找到对应的价格
  end if;
end while;
close cursor1; #关闭cursor
UPDATE my_stock SET volume=volume-new.amount,
          avg_price=((volume+new.amount)*avg_price-
profit_)/(volume),
          profit=profit+(new.amount*new.price-profit_)
          WHERE stock_id=new.stock_id; #更新数据
```

- 持仓均价的修改公式与原ppt记录有所不同。这里有两种理解
  - 持仓的净成本——群中讨论的结果。$avg\_price = \frac{volume \times avg\_price - spent}{volume - amount}$

    ```
    avg_price=((volume+new.amount)*avg_price-profit_)/(volume)
    ```

  - 持仓均价的标准定义$avg_price = \frac{volume \times avg_price - amount \times price}{volume - amount}$

    ```
    avg_price=((volume+new.amount)*avg_price-
    new.amount*new.price)/(volume)
    ```

- 最后如果卖出的量过大，应该舍弃本条记录

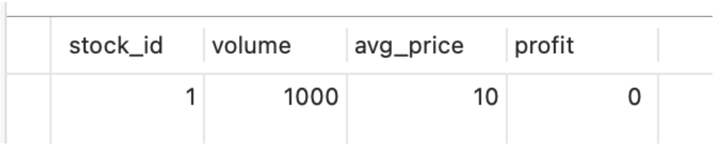
  ```
  signal sqlstate 'HY000' set message_text="not enough for buy";
  ```

- 如此我们完成了整个触发器的设计

实验结果

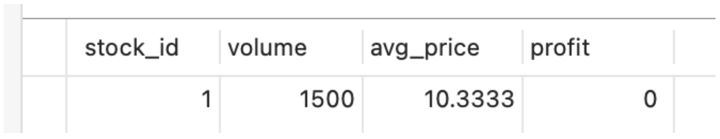
我们按照实验要求构建了模拟数据，其逐步运行结果如下：

```
delete from my_stock;
delete from trans; #清空数据
```



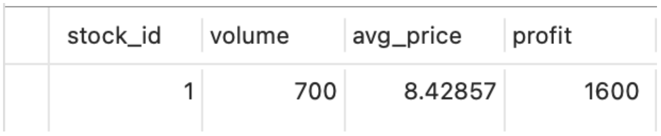
stock_id	volume	avg_price	profit
1	1000	10	0

```
insert ignore into trans values (1,1,1,10,1000,"B");
select * from my_stock;
```



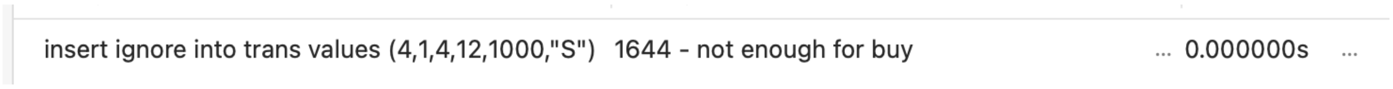
stock_id	volume	avg_price	profit
1	1500	10.3333	0

```
insert ignore into trans values (2,1,2,11,500,"B");
select * from my_stock;
```



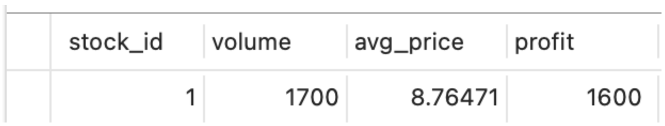
	stock_id	volume	avg_price	profit
	1	700	8.42857	1600

```
insert ignore into trans values (3,1,3,12,800,"S");
select * from my_stock;
```



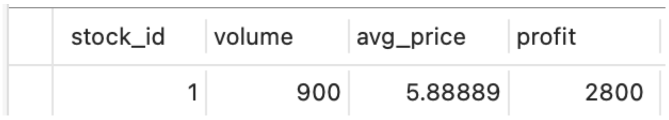
insert ignore into trans values (4,1,4,12,1000,"S") 1644 - not enough for buy ... 0.000000s ...

```
insert ignore into trans values (4,1,4,12,1000,"S");mysql
select * from my_stock;
```



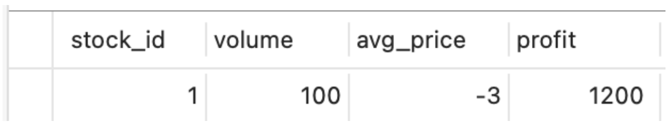
	stock_id	volume	avg_price	profit
	1	1700	8.76471	1600

```
insert ignore into trans values (5,1,5,9,1000,"B");
select * from my_stock;
```



	stock_id	volume	avg_price	profit
	1	900	5.88889	2800

```
insert ignore into trans values (6,1,6,12,800,"S");
select * from my_stock;
```



	stock_id	volume	avg_price	profit
	1	100	-3	1200

```
insert ignore into trans values (7,1,7,7,800,"S");
select * from my_stock;
```

(*注：上述结果均是按照触发器设计中均价按照售卖价格进行计算，否则更改公式即可。如果是用另一种算法，最终的`avg_price`应该为-3)