中国科学技术大学计算机学院

《数据库系统实验报告》

实验题目：	课程设计-银行管理系统
学生姓名：	刘牧辛
学　　号：	PB21111729
完成时间：	2024. 6. 16

一、需求分析

1. 实现用户的增、删、改、查操作，如果用户存在着关联账户或者贷款记录，则不允许删除；
2. 实现账户（储蓄账户和支票账户）的增、删、改、查操作，账户号不允许更改；
3. 实现一个储蓄账户向另一个储蓄账户转账的功能，需要判断两个账户是否存在、转出账户的金额是否够用、货币类型是否相同等；
4. 实现贷款信息的增、删、查功能，提供贷款支付功能，不处于全部支付状态的贷款记录不允许删除；
5. 实现业务统计，统计储蓄和贷款信息。

二、总体设计

系统模块结构：

本系统是一个前后端分离的 B/S 架构，主要的逻辑模块分为前端部分和后端部分。

1. 前端部分：主要有以下模块界面：初始界面、客户管理界面、账户管理界面、贷款管理和贷款支付界面和业务统计界面，界面搭建使用了 BootStrap 模板。

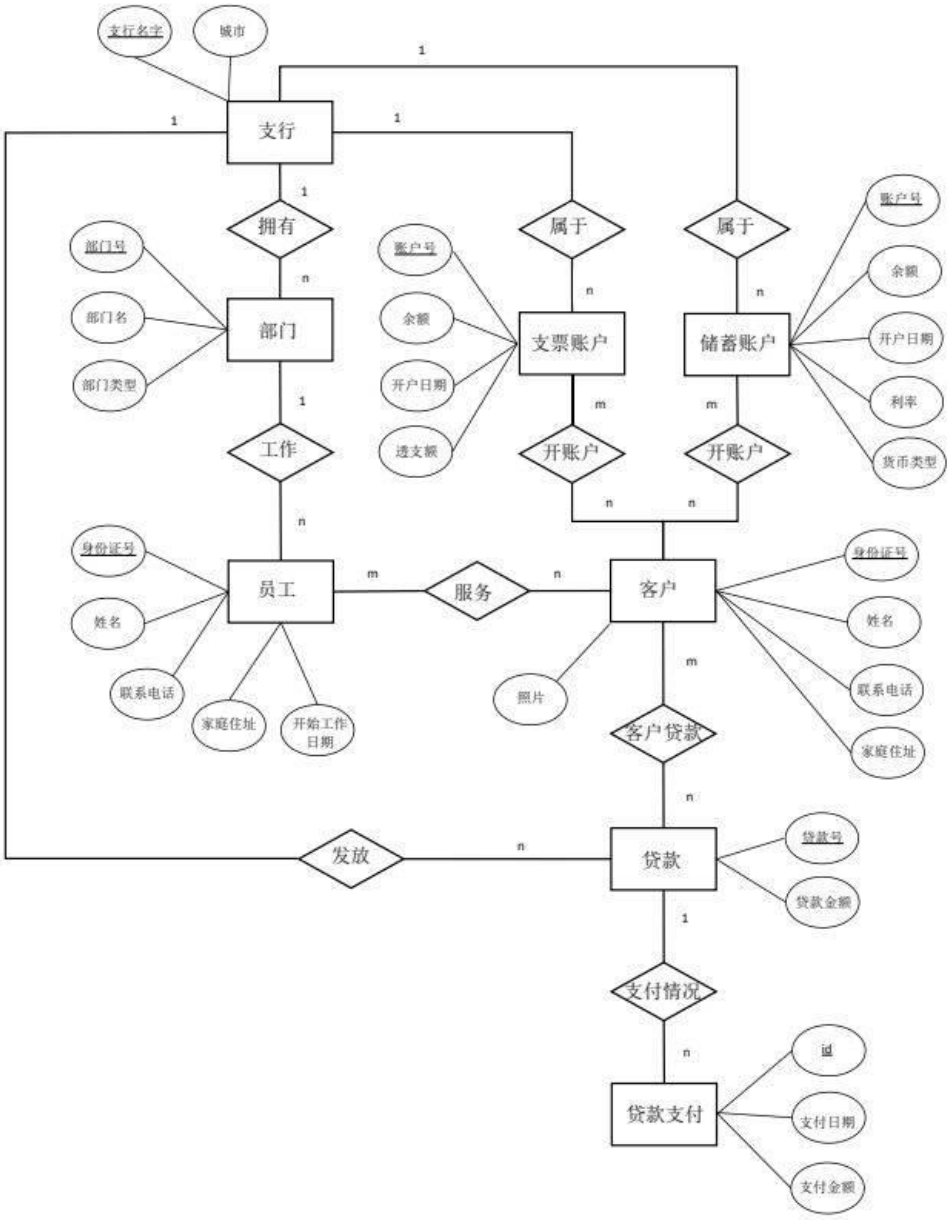
2. 后端部分：使用了 Django 模板，核心部分为 BankSys/BankManage/models.py（创建基本表）、BankSys/BankManage/views.py（实现增删改查、转账、业务统计等操作）以及 sql 文件（向数据库中事先插入一些数据，以及存储过程、函数、触发器等），数据库平台使用 MySQL。

系统工作流程：

当用户进入到银行系统的页面后，可以在四种主要的功能中进行选择和切换。当用户输入了满足要求的数据并点击提交按钮后，前端向后端服务器 post 用户填入的数据，服务器在解析用户的信息后，先检查参数是否合法，若合法则执行对应的操作并返回执行后的状态信息（例如查询结果等）到前端，前端将接收到的信息渲染在页面上，从而完成一次与服务器的交互。

三、数据库设计

ER 图：



模式分解：

本数据库的模式设计如下：

Bank(title, city, assset) #银行

Department(Depart_ID, fk_bank, title, dtype) #部门

Employee(ID_card, fk_bank, fk_department, name, phone, address, entry_date) #员工

Client(ID_card, name, phone, address, img) #客户

Service(id, client, employee, stype) #客户、员工间的服务

StoreAccount(Account_ID, fk_bank, balance, open_date, rate, currency_type) #储蓄账户

CheckAccount(Account_ID, fk_bank, balance, open_date, overdraft) #支票账户

OpenStore(id, fk_client, fk_account, last_access) #（客户）开储蓄账户

OpenCheck(id, fk_client, fk_account, last_access) #（客户）开支票账户

Loan(Loan_ID, fk_bank, money, status) #贷款

Payment(Payment_ID, fk_loan, pay_date, pay_money) #贷款支付

ClientLoan(id, fk_loan, fk_client) #客户发起贷款

该模式分解满足 3NF。

存储过程、触发器、函数等在核心代码解析部分进行详细说明。

四、核心代码解析

仓库地址：

https://github.com/liumuxin1/lab2.git

目录：

.

├── account.sql

├── bank.sql

├── client.sql

├── loan.sql --前 4 个为向数据库事先添加的数据

├── procedure.sql --实现的存储过程、事务、函数、触发器等

├── BankSys --创建的 project

│ ├── BankManage --Django 创建的应用

│ │ ├── admin.py

│ │ ├── apps.py

│ │ ├── __init__.py

```
│   │   ├── migrations --迁移文件
│   │   ├── models.py --使用 model 来创建数据库的基本表
│   │   ├── static
│   │   │   └── assets --Bootstrap 模板，用来对前端进行渲染
│   │   │       ├── images
│   │   │       ├── libs
│   │   │       └── vendor
│   │   ├── templates
│   │   │   └── BankManage --前端界面
│   │   │       ├── account.html --账户管理
│   │   │       ├── client.html --用户管理
│   │   │       ├── index.html --初始界面
│   │   │       ├── loan_issue.html --贷款管理
│   │   │       ├── loan_manage.html --贷款支付
│   │   │       └── statistics.html --数据统计
│   │   ├── tests.py
│   │   ├── urls.py --将用户请求的 URL 映射到相应的视图函数
│   │   └── views.py --创建视图，处理用户请求（增删改查、转账、贷
                       款还款等）和返回响应（查询结果等）的函数
│   ├── BankSys
│   │   ├── asgi.py
│   │   ├── __init__.py
│   │   ├── __pycache__
│   │   ├── settings.py --项目的所有配置，例如项目应用配置、数据
                          库配置、路由和 URL 配置等
│   │   ├── urls.py
│   │   └── wsgi.py
│   ├── manage.py
│   └── media
│       └── img --数据库对图片的处理
```

客户管理：

在 models.py 定义了 client 表，存储客户的信息

```
# BankSys/BankManage/models.py
class Client(models.Model):
    ID_card = models.CharField(max_length=18,primary_key=True)
    #身份证号
    name = models.CharField(max_length=100) #姓名
    phone = models.CharField(max_length=11) #电话号码
    address = models.CharField(max_length=200) #地址
    img = models.ImageField(upload_to='img', null=True, blank=True)
    #客户照片
```

这里实现了数据库对图片的管理，使用 Django 的 ImageField 进行图片存储。在 views.py 中实现客户的增删改查，并输出相应结果。

```
# BankSys/BankManage/views.py 以客户的增加为例
if func == 'add':
    request.session['client_view_change'] = 'add'
    #对应添加客户的功能
    attrs = request.POST.dict()
    if len(Client.objects.filter(ID_card=attrs['ID_card'])) == 0:
        #如果不存在该客户，可以添加
        obj = Client(ID_card=attrs['ID_card'],
                     name=attrs['name'],
                     phone=attrs['phone'],
                     address=attrs['address'],
                     img=request.FILES.get('img')
                     )
        try:
            obj.save()
        except djangoDBError as e: #如果出错，输出报错信息
            messages.error(request, 'ERROR: '+str(e))
        except:
            messages.error(request, 'ERROR: Failed to save client!')
```

```
            raise
        else: #添加成功
            messages.info(request, 'SUCCESS: New client saved!')
    else:
        messages.error(request, 'ERROR: Client already exists!')
```

注意对于普通的文本，使用 request.POST.dict()获取用户输入的信息，对于图片的获取要使用 request.FILES.get()。用 obj.save()存储新客户信息。

客户查询的实现是根据用户在界面输入的信息，使用 objects.filter()对 client 表进行筛选，返回所有满足条件的客户。对于客户修改/删除部分，根据用户勾选的一个或多个客户，进行相应的修改/删除，但是有关联账户或贷款信息的用户不可删除。由于原本实现的修改操作只能修改文本，不支持修改图片，所以增加了修改客户图片的模块。

账户管理：

定义的相关的表

```
# BankSys/BankManage/models.py 以储蓄账户为例，支票账户类似
class StoreAccount(models.Model): #储蓄账户
    Account_ID = models.CharField(max_length=50,primary_key=True) #ID
    fk_bank = models.ForeignKey(Bank,on_delete=models.SET_NULL,null=True)
    #银行
    balance = models.FloatField(default=0) #余额
    open_date = models.DateField() #开户日期
    rate = models.FloatField() #利率
    currency_type = models.CharField(max_length=20) #货币类型（CNY、USD 等）
    class Meta:
        unique_together=("Account_ID","fk_bank")
class OpenStore(models.Model): #客户与账户的联系
    id = models.AutoField(primary_key=True) #id
    fk_client = models.ForeignKey(Client,on_delete=models.PROTECT,null=True)
    #客户
    fk_account = models.ForeignKey(StoreAccount,on_delete=models.CASCADE,
null=True) #账户
    last_access = models.DateField(auto_now=True) #上次访问时间
```

储蓄/支票账户的增删改查与客户的增删改查类似，不再重复讲解。这里重点讲

解储蓄账户的转账操作。

在 procedure.sql 中定义了一个存储过程 bank_transfer，实现转账操作，注意转账是需要使用事务的，因为转出账户减少金额、转入账户增加金额这两个操作是不可分的。输入转出、转入账户和金额，如果账户存在、转出账户余额足够、两账户货币类型一致，则执行转账、commit 事务并返回状态 state=0，否则 rollback 事务，返回相应的 state，对应不同的错误类型。

```
--procedure.sql

delimiter //

create procedure bank_transfer (in ac_from varchar(50), in ac_to varchar(50), in money double, out state int) --实现转账操作的存储过程

begin

    declare s int default 0;

    declare countf int;

    declare countt int;

    declare bal double;

    declare type1 varchar(20);

    declare type2 varchar(20);

    declare continue handler for sqlexception set s = 1;

    start transaction;

    select count(*) from BankManage_storeaccount where Account_ID = LPAD(ac_from, 10, '0') into countf;

    select count(*) from BankManage_storeaccount where Account_ID = LPAD(ac_to, 10, '0') into countt;

    if countf = 0 or countt = 0 then --不存在 ID 为 ac_from 或 ac_to 的账户

        set s = 2;

    end if;

    select balance from BankManage_storeaccount where Account_ID = LPAD(ac_from, 10, '0') into bal;

    if bal < money then --余额不足

        set s = 3;

    end if;

    select currency_type from BankManage_storeaccount where Account_ID = LPAD(ac_from, 10, '0') into type1;

    select currency_type from BankManage_storeaccount where Account_ID = LPAD(ac_to, 10, '0') into type2;

    if type1 != type2 then --货币类型不同

        set s = 4;
```

```
    end if;
    update BankManage_storeaccount set balance = balance - money where Account_ID = LPAD(ac_from, 10, '0');
    update BankManage_storeaccount set balance = balance + money where Account_ID = LPAD(ac_to, 10, '0'); --转账操作
    if s = 0 then
        commit; --状态为 0，提交事务
    else
        rollback; --否则回滚事务
    end if;
    set state = s;
end //
delimiter ;
```

在 views.py 中，根据用户在浏览器输入的账户号和金额，调用上述存储过程，根据 state 的值来给出响应（成功/失败）。

```
# BankSys/BankManage/views.py
elif func == 'transfer':
    attrs = request.POST.dict()
    with connection.cursor() as cursor:
        sql = "call bank_transfer ({0}, {1}, {2}, @state)".format(str(attrs['Account_ID_from']), str(attrs['Account_ID_to']), attrs['money'])
        cursor.execute(sql)
        sql = 'select @state'
        cursor.execute(sql)
        row = cursor.fetchone()[0]
        if (row == 0):
            messages.info(request, 'SUCCESS!')
            request.session['account_view'] = 'query'
        elif (row == 1):
            messages.error(request, 'ERROR: sqlexception!')
        elif (row == 2):
            messages.error(request, 'ERROR: no storeaccount!')
        elif (row == 3):
            messages.error(request, 'ERROR: balance is not enough!')
        elif (row == 4):
            messages.error(request, 'ERROR: currency type not match!')
```

```
        else:
            messages.error(request, 'ERROR')
```

贷款管理：

定义的相关的表

```
# BankSys/BankManage/models.py
class Loan(models.Model): #贷款
    Loan_ID = models.CharField(max_length=50,primary_key=True) #贷款号
    fk_bank = models.ForeignKey(Bank,on_delete=models.SET_NULL,null=True)
    #银行
    money = models.FloatField(default=0) #金额
    status = models.CharField(max_length=20, default='未支付')
    #状态（未支付/支付中/已全部支付）
class Payment(models.Model): #贷款支付记录
    Payment_ID = models.AutoField(primary_key=True) #支付号
    fk_loan = models.ForeignKey(Loan,on_delete=models.SET_NULL,null=True)
    #贷款号
    pay_date = models.DateField() #支付日期
    pay_money = models.FloatField() #支付金额
class ClientLoan(models.Model): #用户贷款
    id = models.AutoField(primary_key=True)
    fk_loan = models.ForeignKey(Loan,on_delete=models.SET_NULL,null=True)
    fk_client = models.ForeignKey(Client,on_delete=models.PROTECT,null=True)
    class Meta:
        unique_together=("fk_loan","fk_client")
```

注意，这里实现的是贷款的增删查，不能更改，删除贷款记录只能删除已全部支付的贷款记录。实现方法与客户、账户类似。在贷款查询结果中的已支付金额一栏使用 pay 函数进行计算。输入贷款号，在 payment 表中计算已支付金额的和。

```
--procedure.sql
delimiter //
create function pay (loan_id varchar(50))
returns double
reads sql data
begin
    declare flag int;
```

```
    declare money double;
    select count(*) from BankManage_payment where fk_loan_id = LPAD(loan_id, 10, '0') into flag;
    select sum(pay_money) from BankManage_payment where fk_loan_id = LPAD(loan_id, 10, '0') into money;
    if flag = 0 then --没有还款记录
        return 0;
    else
        return money;
    end if;
end //
delimiter ;
```

下一功能是贷款支付。根据用户在前端输入的贷款号、支付时间、支付金额，在 payment 表中插入贷款记录。这里定义了一个触发器，每当 payment 表中插入数据，则计算当前贷款的支付金额，再与贷款表中的贷款金额比较，来更改贷款表中相应的 status。

```
--procedure.sql
delimiter //
create trigger loan_status after insert on BankManage_payment for each row
begin
    declare m double;
    declare p double;
    select money from BankManage_loan where Loan_ID = new.fk_loan_id into m;
    select sum(pay_money) from BankManage_payment where fk_loan_id = new.fk_loan_id into p;
    if p > 0 and p < m then
        update BankManage_loan set status = '支付中' where Loan_ID = new.fk_loan_id;
    elseif p >= m then
        update BankManage_loan set status = '已全部支付' where Loan_ID = new.fk_loan_id;
    end if;
end //
delimiter ;
```

业务统计：

根据当前表的内容统计各个银行的贷款业务总金额、用户数和储蓄业务总额。贷款业务可以直接查 loan 表，统计各个银行的贷款总金额。计算用户数比较复杂，

需要查找储蓄账户、支票账户和贷款三个表，统计每个银行这三部分的客户的身份证号，最后取并集。储蓄业务要统计储蓄账户和支票账户，储蓄账户有两种货币类型（CNY/USD），支票账户默认货币类型为 CNY。分别统计 CNY 的储蓄总额和 USD 的每一年的储蓄总额，最后输出到前端显示。

五、实验与测试

依赖

需要使用 Django 模板（4.2.13）、pymysql、BootStrap 模板等。

运行环境:

操作系统 linux ubuntu 20.04.6 LTS

编程语言版本 python 3.8.10; mysql 8.0.36

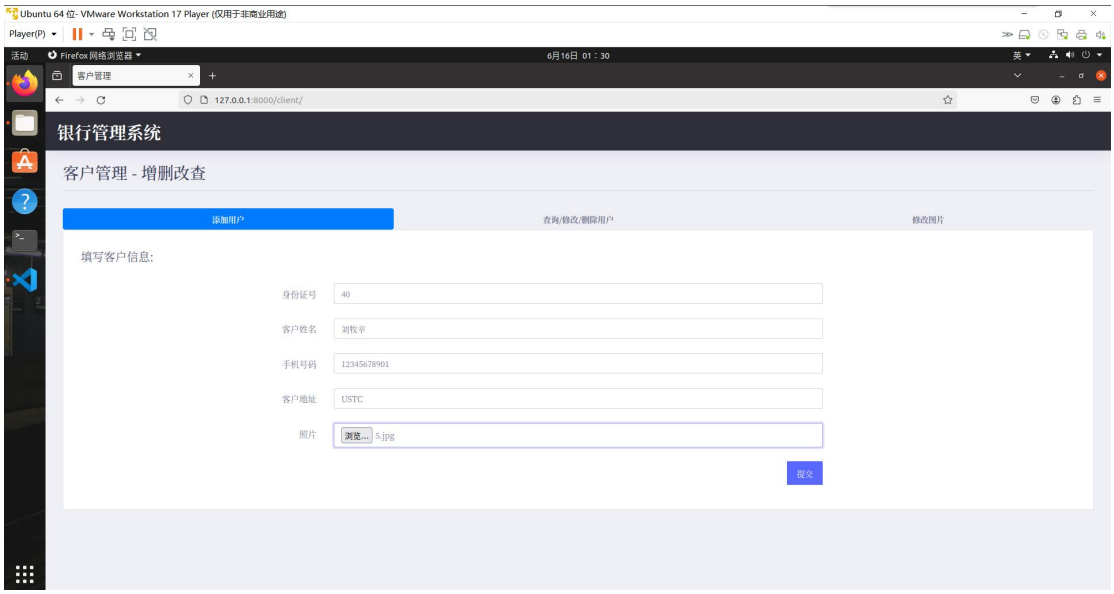
部署

在./BankSys 目录下输入命令 python3 manage.py runserver 0.0.0.0:8000，然后在浏览器中打开 http://127.0.0.1:8000 即可进入银行管理系统的初始界面。

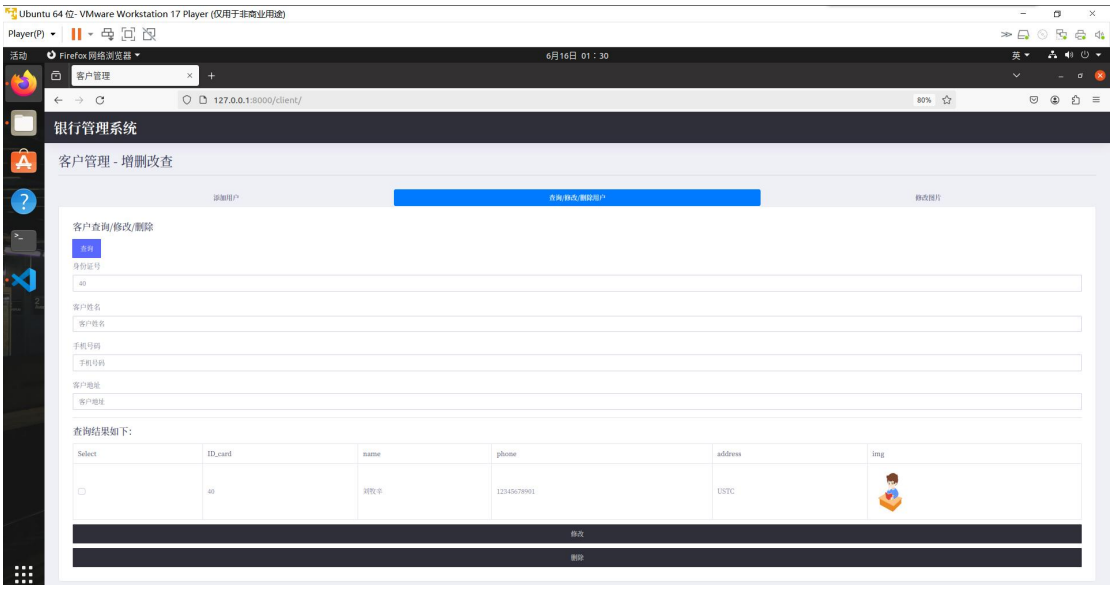
实验结果

1. 客户管理——增删改查

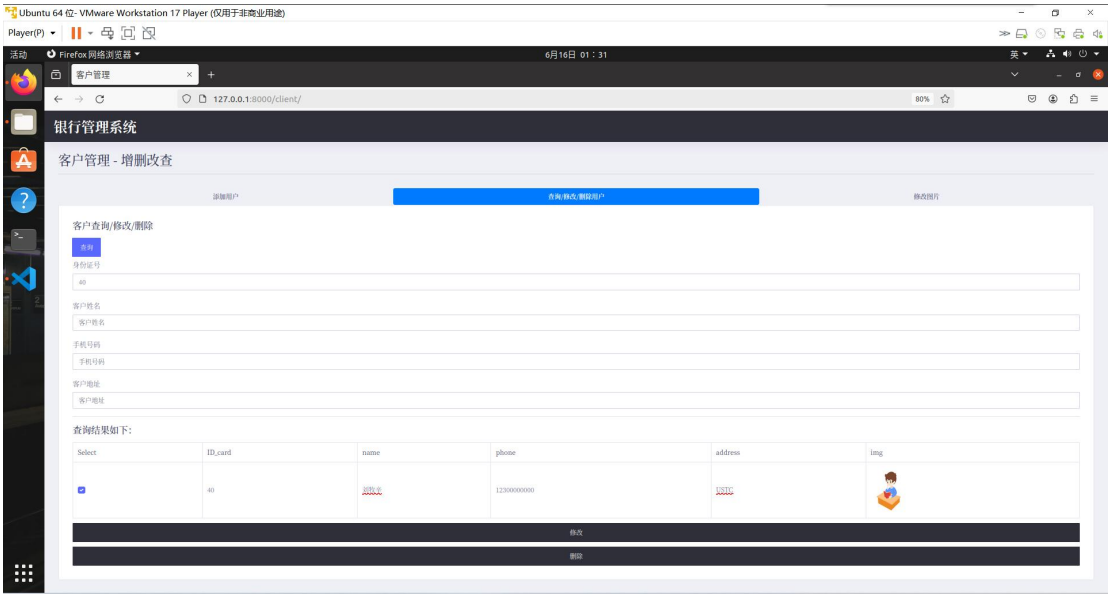
添加用户，点击提交

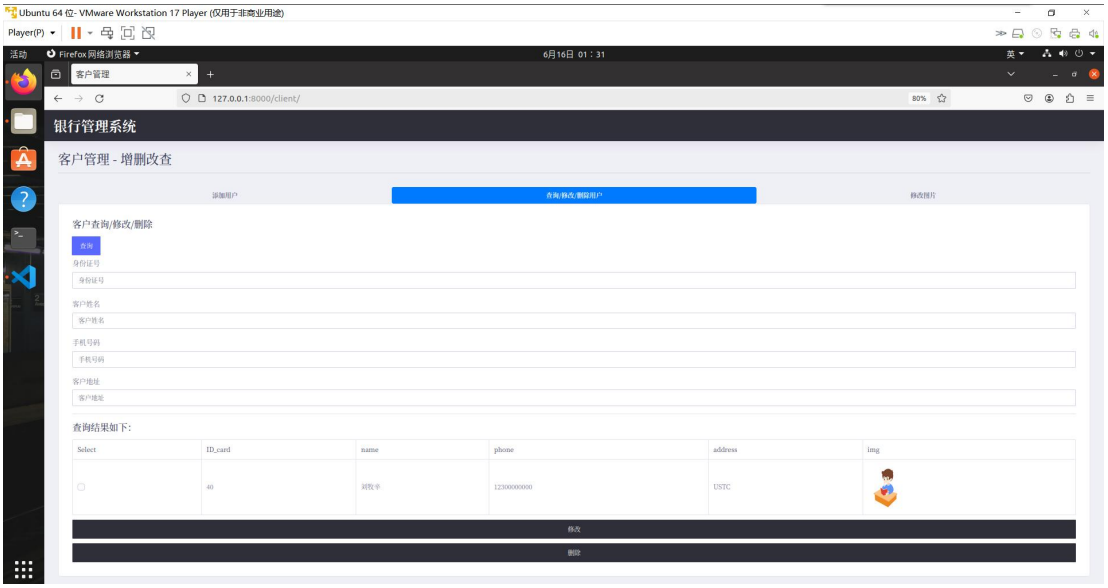


查询身份证号为 40 的客户

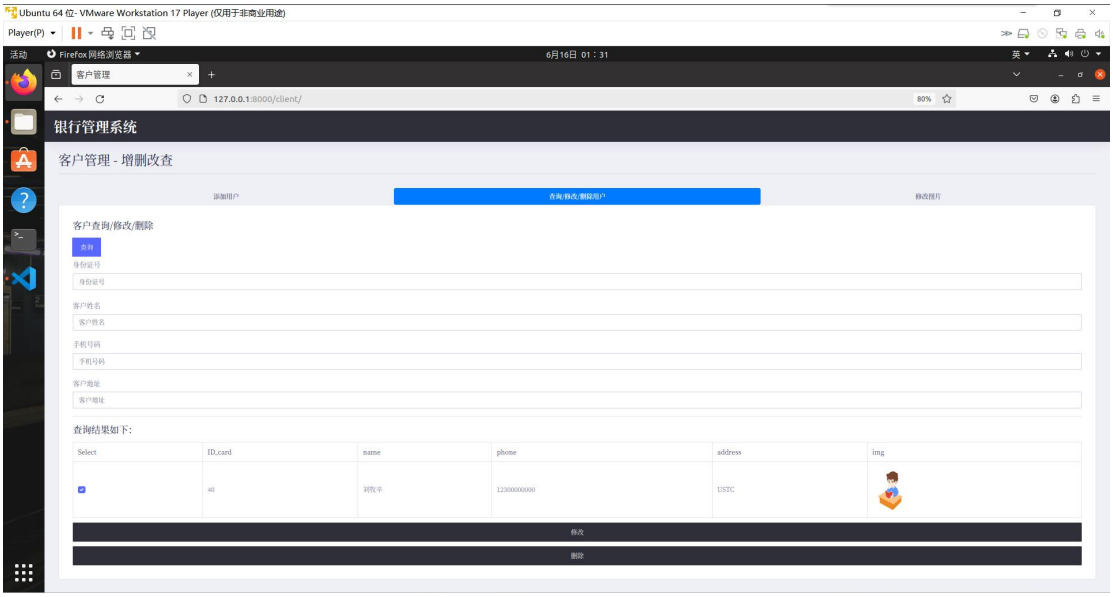


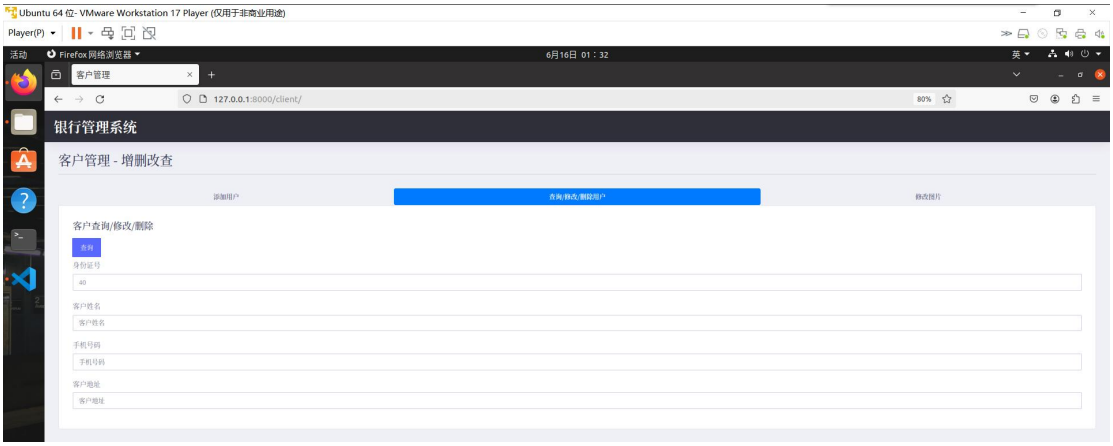
在表中将客户的手机号修改为 12300000000，勾选，点击修改按钮，再查询





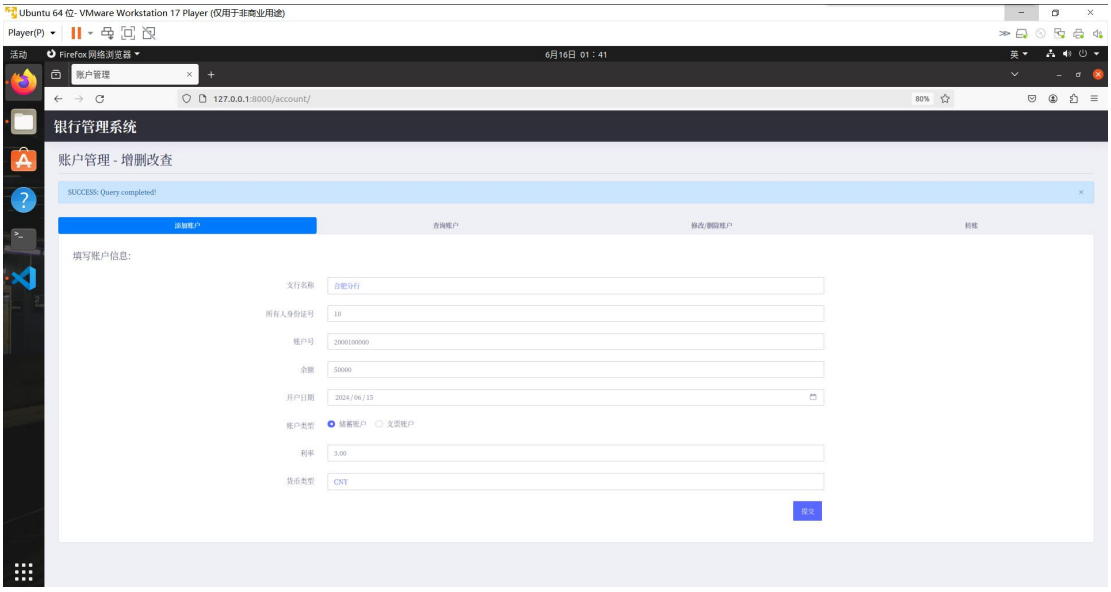
删除该用户，勾选点击删除。再次查询身份证号 40 的客户，返回空的表。



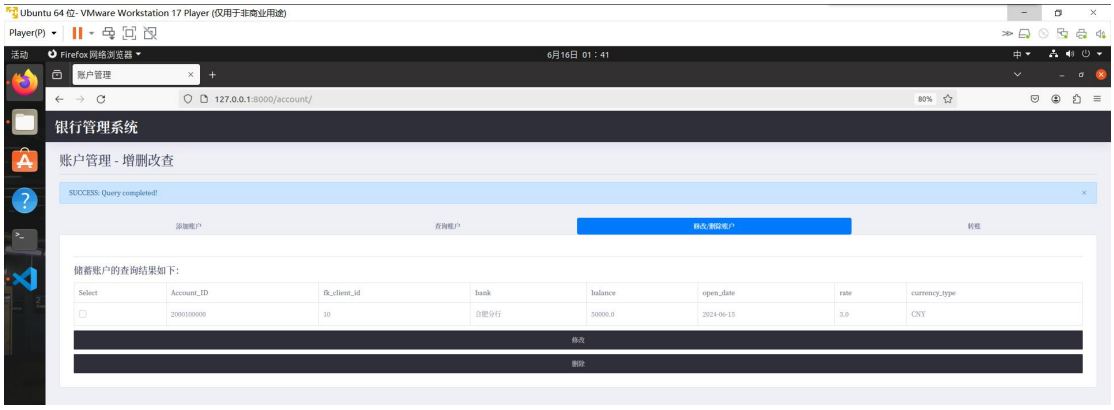


2. 账户管理——增删改查+转账

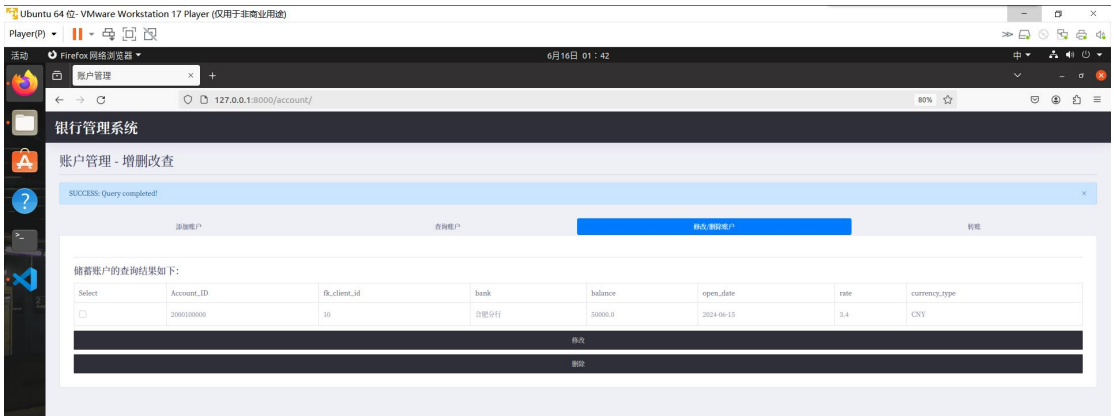
添加储蓄账户



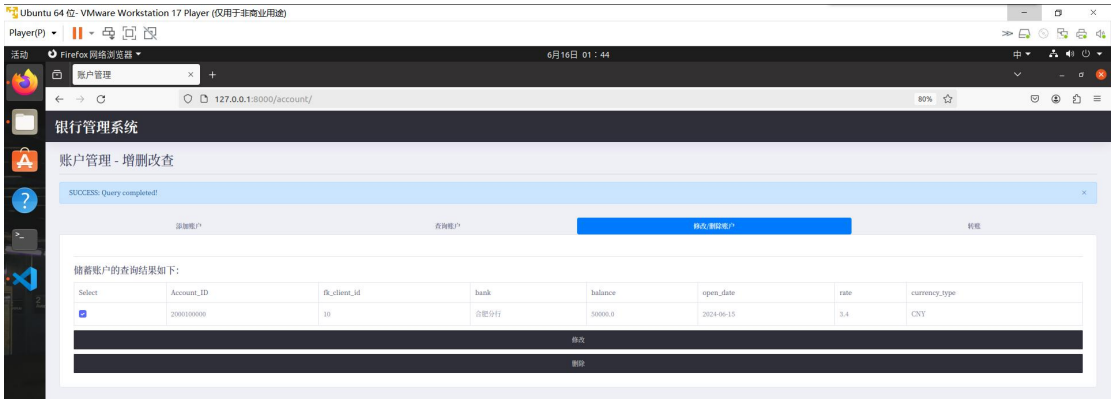
查询账户号为 2000100000 的储蓄账户

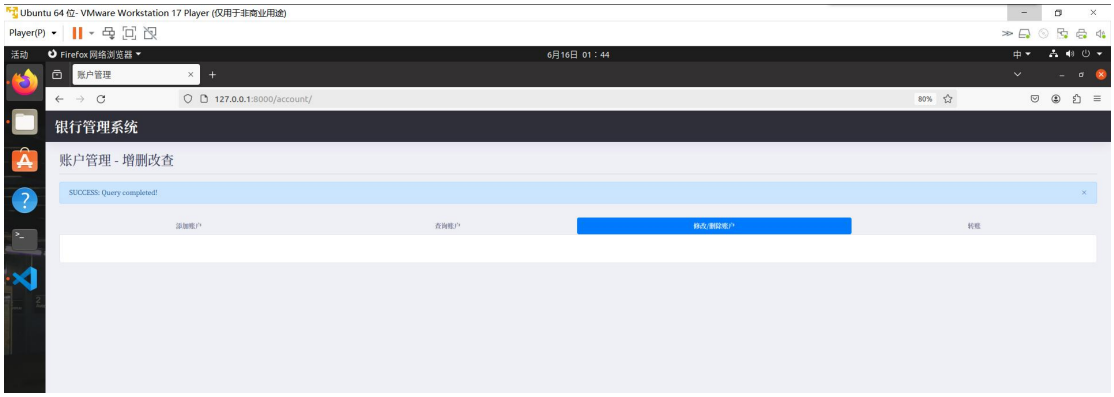


更改利率为 3.4

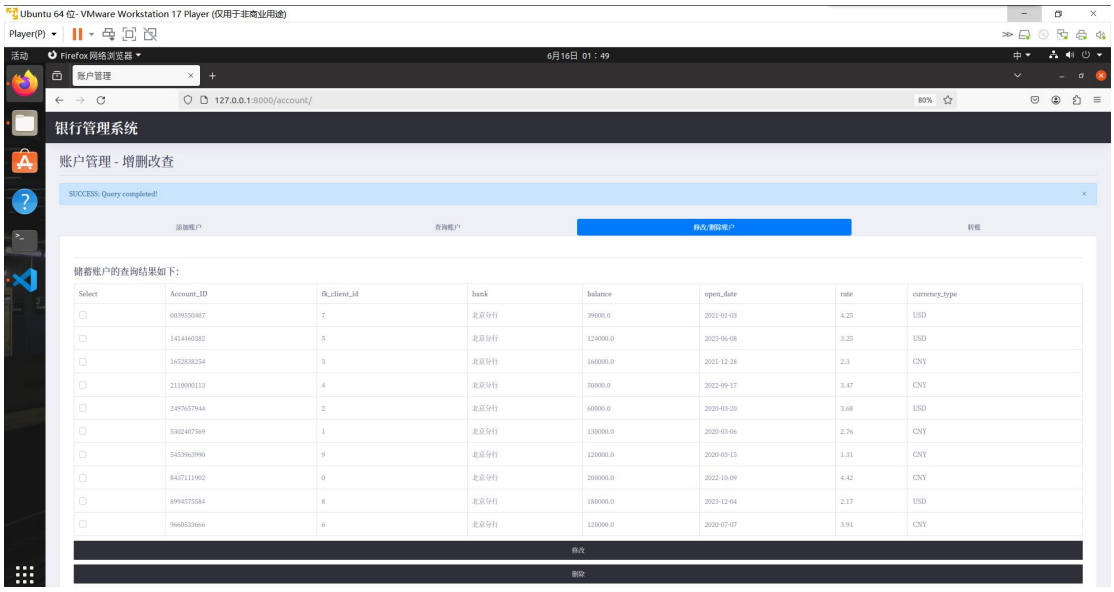


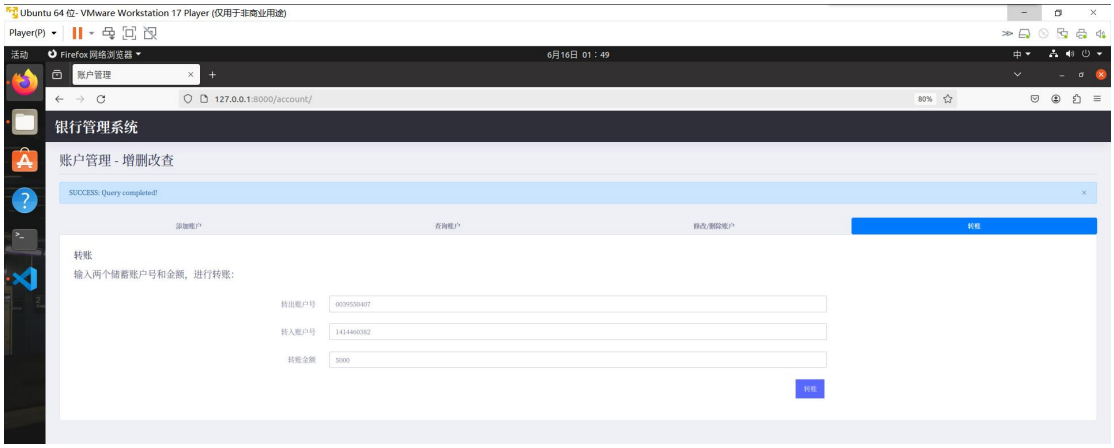
删除该账户

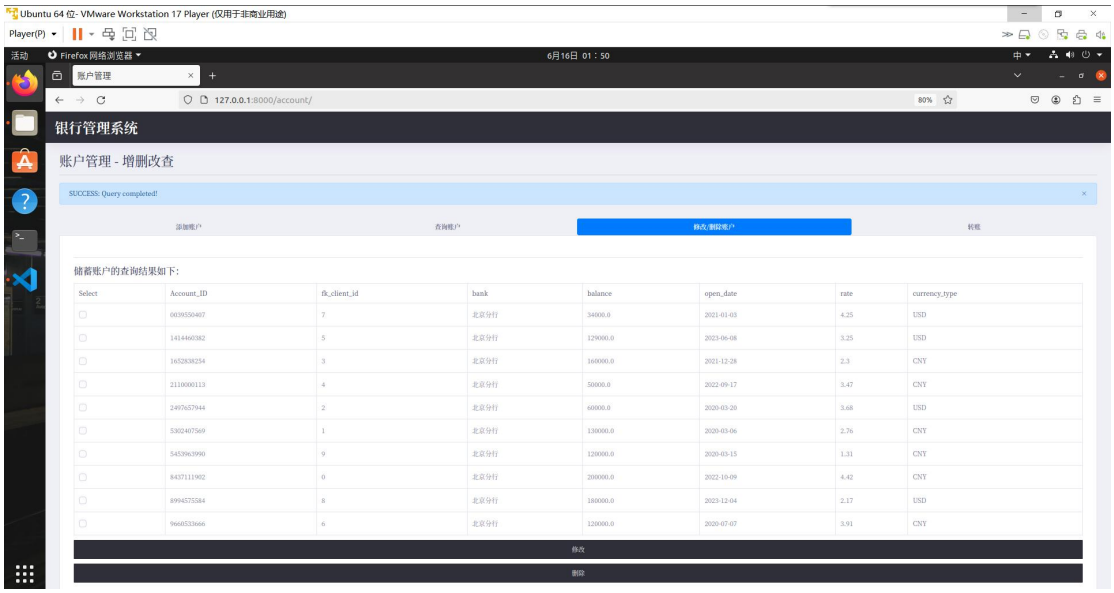




转账

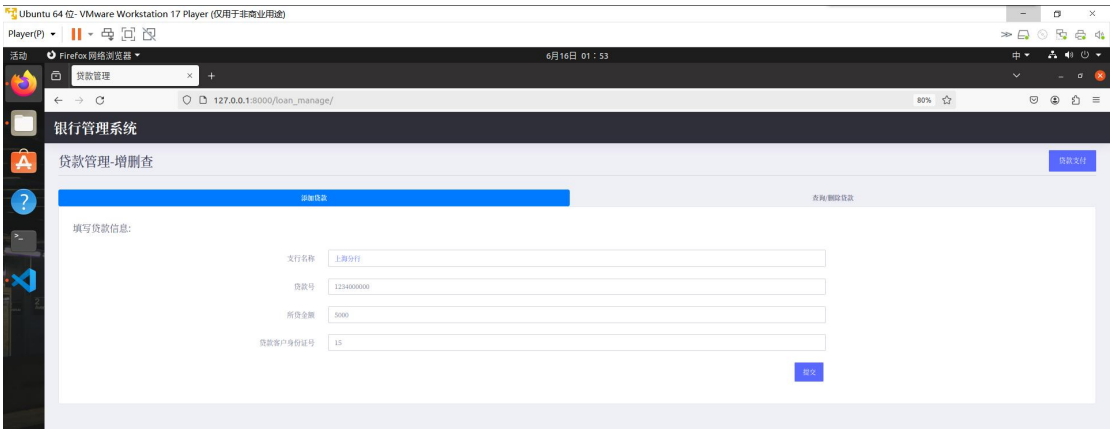




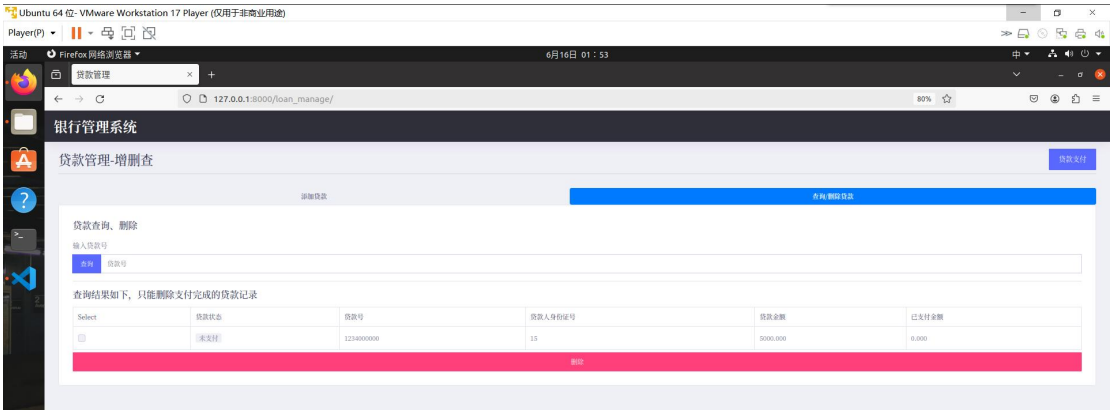


3. 贷款管理——增删查+贷款支付

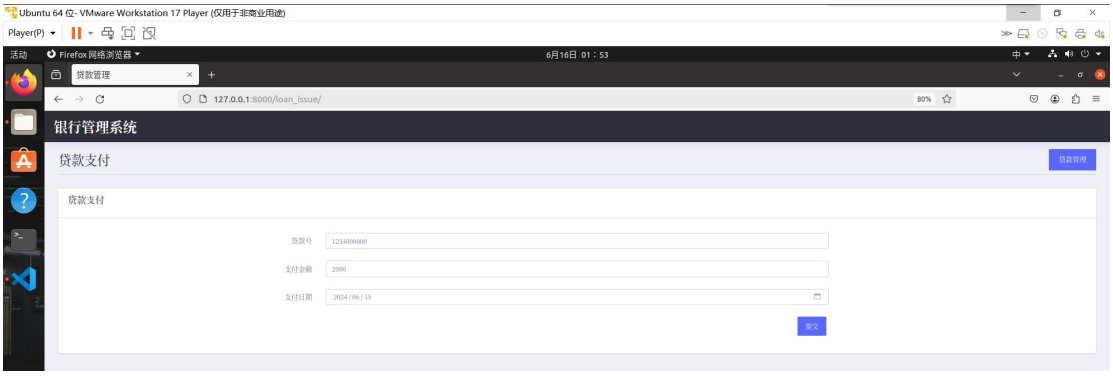
添加贷款

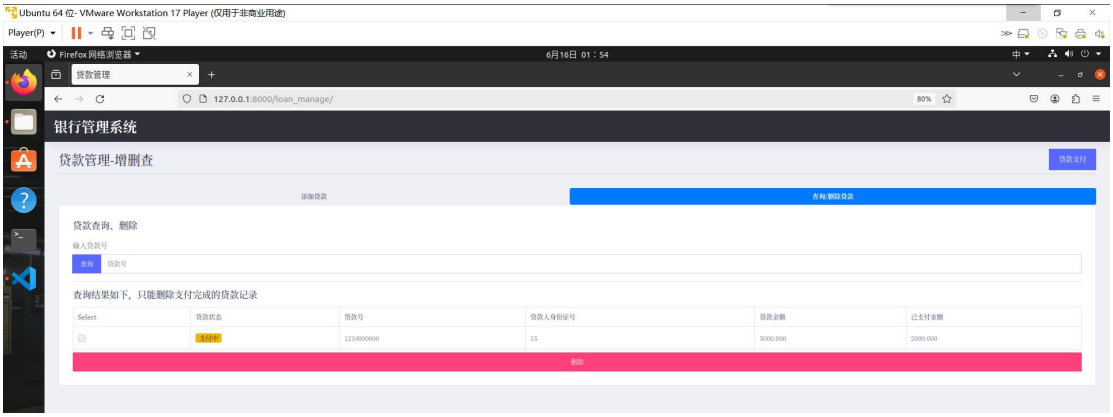


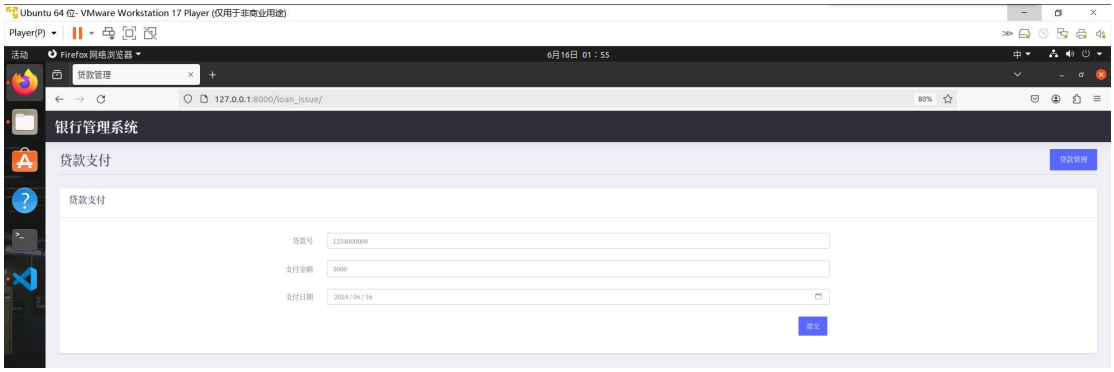
查询贷款

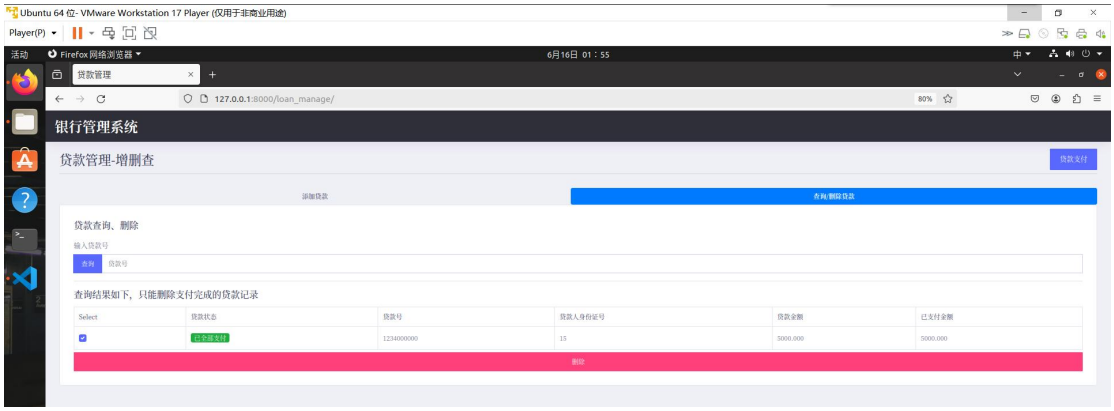


添加贷款支付记录

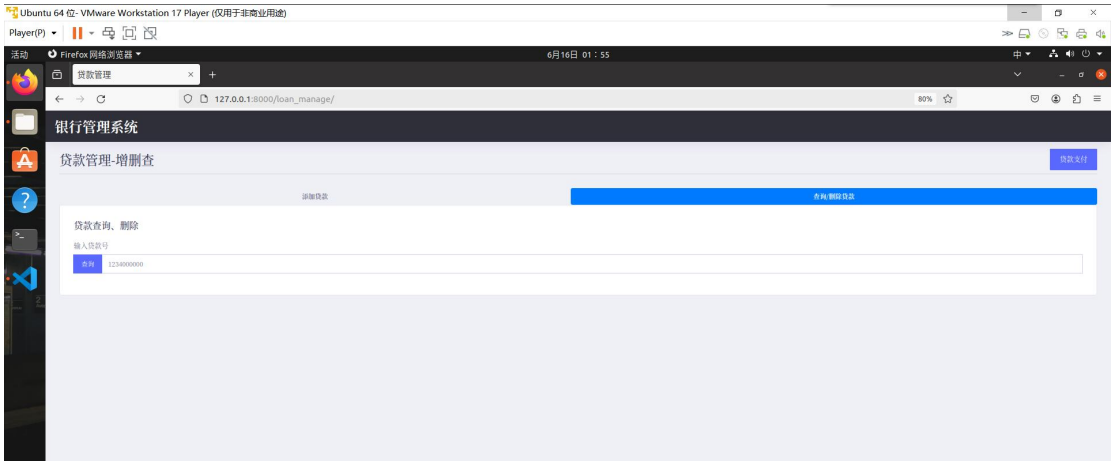




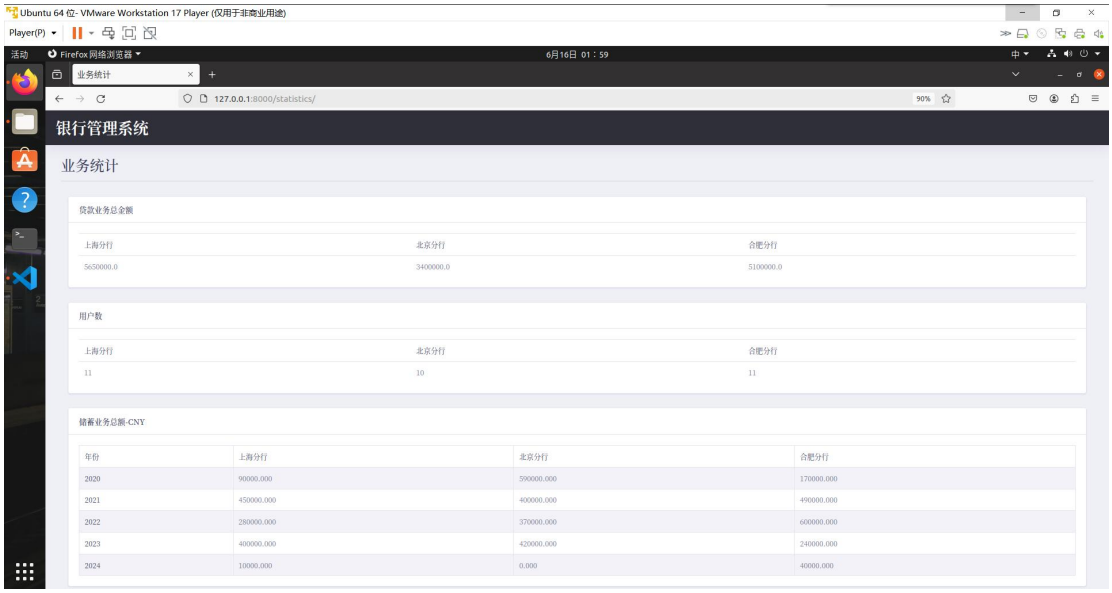


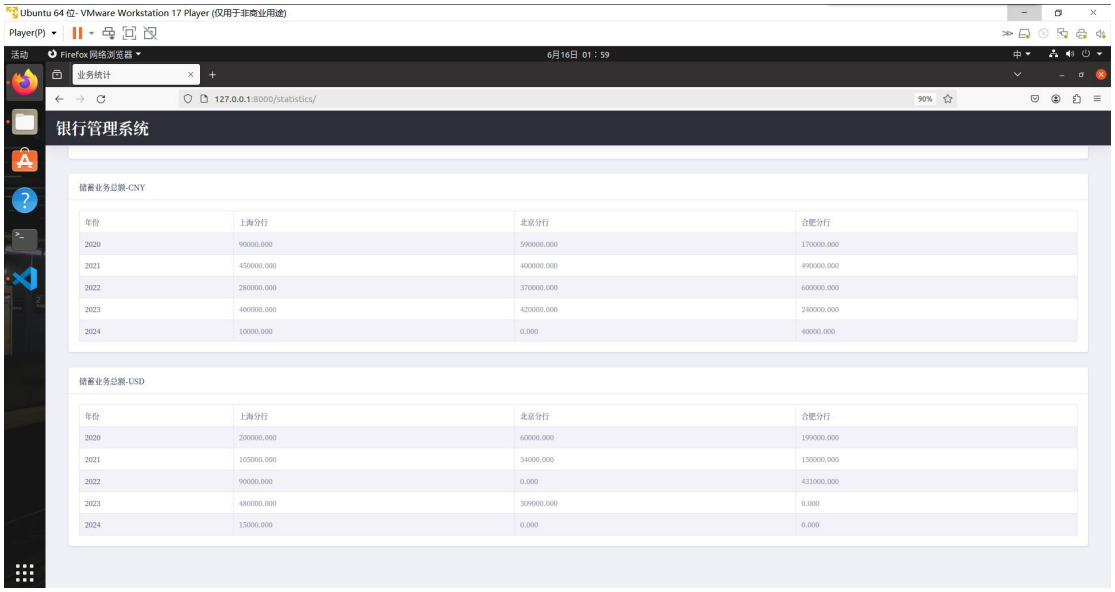


删除贷款（必须是已全部支付的贷款）



4. 业务统计


Ubuntu 64 位- VMware Workstation 17 Player (仅用于非商业用途)
Player(P)
活动
Firefox 网络浏览器
6月16日 01：59
业务统计
127.0.0.1:8000/statistics/
90%
银行管理系统
业务统计
贷款业务总金额
上海分行
北京分行
合肥分行
5650000.0
3400000.0
5100000.0
用户数
上海分行
北京分行
合肥分行
11
10
11
储蓄业务总额-CNY
年份
上海分行
北京分行
合肥分行
2020
90000.000
590000.000
170000.000
2021
450000.000
400000.000
490000.000
2022
280000.000
370000.000
600000.000
2023
400000.000
420000.000
240000.000
2024
10000.000
0.000
40000.000


Ubuntu 64 位- VMware Workstation 17 Player (仅用于非商业用途)
Player(P)
活动
Firefox 网络浏览器
6月16日 01：59
业务统计
127.0.0.1:8000/statistics/
90%
银行管理系统
储蓄业务总额-CNY
年份
上海分行
北京分行
合肥分行
2020
90000.000
590000.000
170000.000
2021
450000.000
400000.000
490000.000
2022
280000.000
370000.000
600000.000
2023
400000.000
420000.000
240000.000
2024
10000.000
0.000
40000.000
储蓄业务总额-USD
年份
上海分行
北京分行
合肥分行
2020
200000.000
60000.000
199000.000
2021
105000.000
34000.000
150000.000
2022
90000.000
0.000
431000.000
2023
480000.000
309000.000
0.000
2024
15000.000
0.000
0.000