银行业务管理系统

系统设计与实现报告

目录

1 概述

1.1 系统目标

构建一个银行业务管理系统，实现系统的前端页面、后台服务器和数据库的设计构建，最终系统运行实现功能完整流畅，数据库操作安全规范，界面美观友好，满足银行业务管理的实际需求，保证数据的一致性，具有一定的鲁棒性和可用性。

1.2 需求说明

- 客户管理：提供客户所有信息的增、删、改、查功能；如果客户存在着关联账户或者贷款记录，则不允许删除；
- 账户管理：提供账户开户、销户、修改、查询功能，包括储蓄账户和支票账户；账户号不允许修改；
- 贷款管理：提供贷款信息的增、删、查功能，提供贷款发放功能；贷款信息一旦添加成功后不允许修改；要求能查询每笔贷款的当前状态（未开始发放、发放中、已全部发放）；处于发放中状态的贷款记录不允许删除；
- 业务统计：按业务分类（储蓄、贷款）和时间（月、季、年）统计各个支行的业务总金额和用户数，要求对统计结果同时提供表格和曲线图两种可视化展示方式。

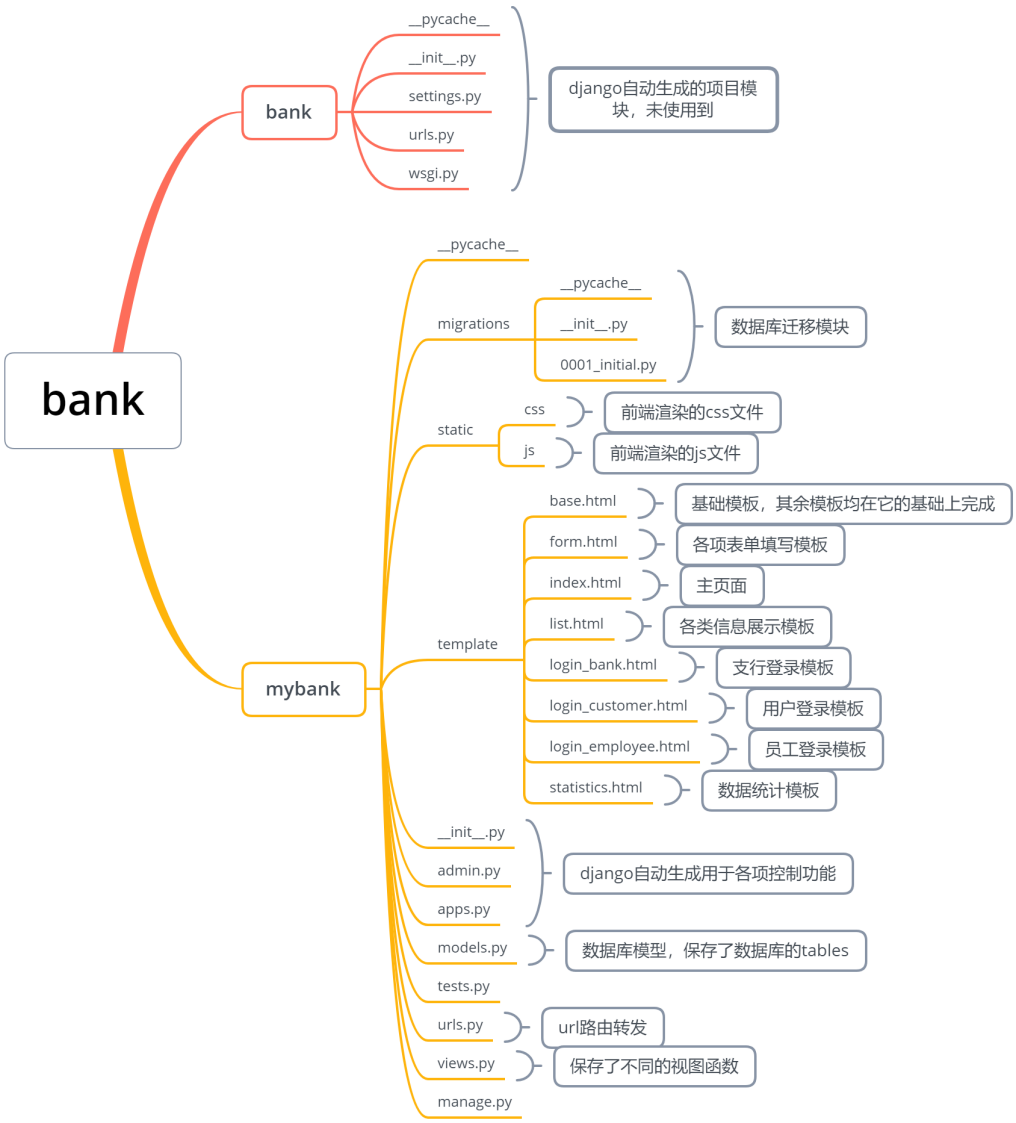
1.3 本报告的主要贡献

- 介绍了该系统的主要模块和功能
- 介绍了该系统的具体使用流程和使用中的注意事项
- 对使用到的数据库的逻辑结构、物理结构进行了说明，并对相应属性进行了解释
- 介绍了运行环境的部署方案
- 对各个模块的具体实现方法进行了简单介绍，并说明了程序执行的具体流程

2 总体设计

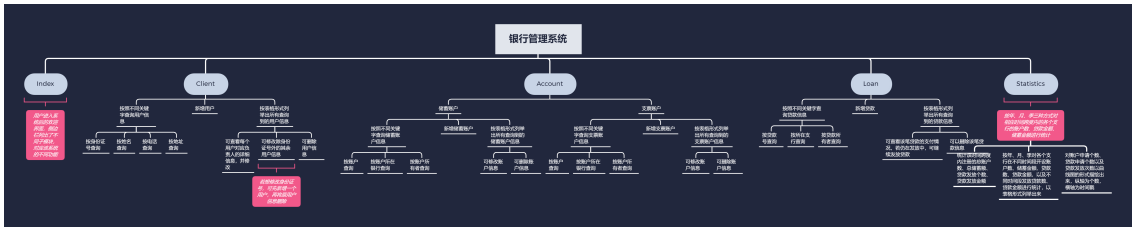
2.1 系统模块结构

- 目录树结构：



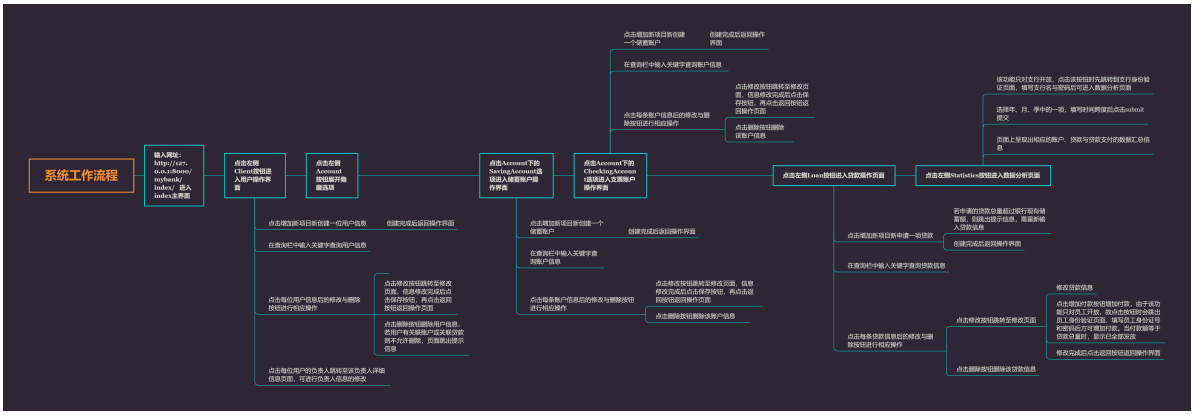
- 功能模块结构：

  由于图像太大，故将图像作为附件添加在文件夹中。



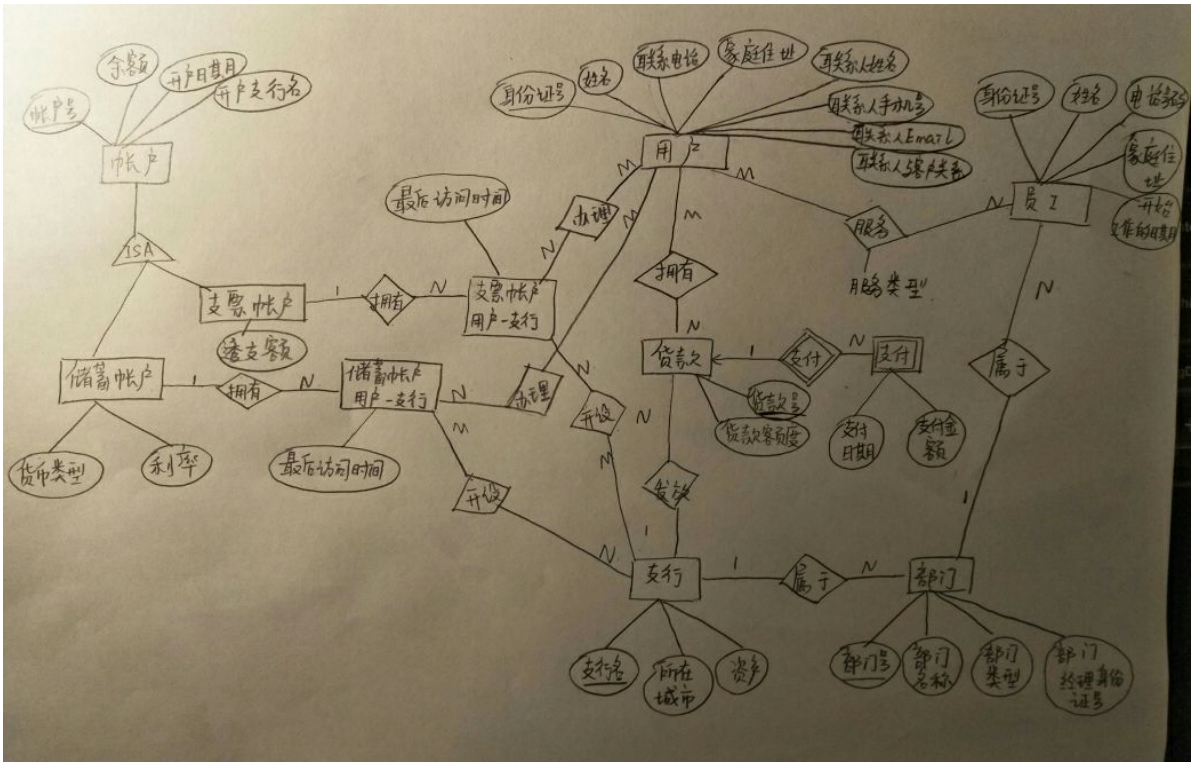
2.2 系统工作流程

由于图像太大，故将图像作为附件添加在文件夹中。



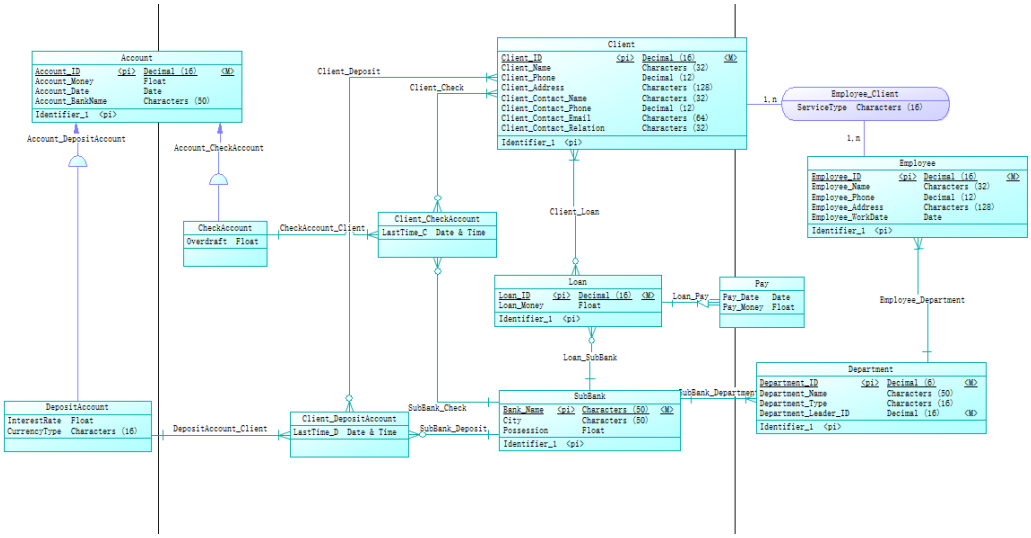
2.3 数据库设计

2.3.1 E-R图

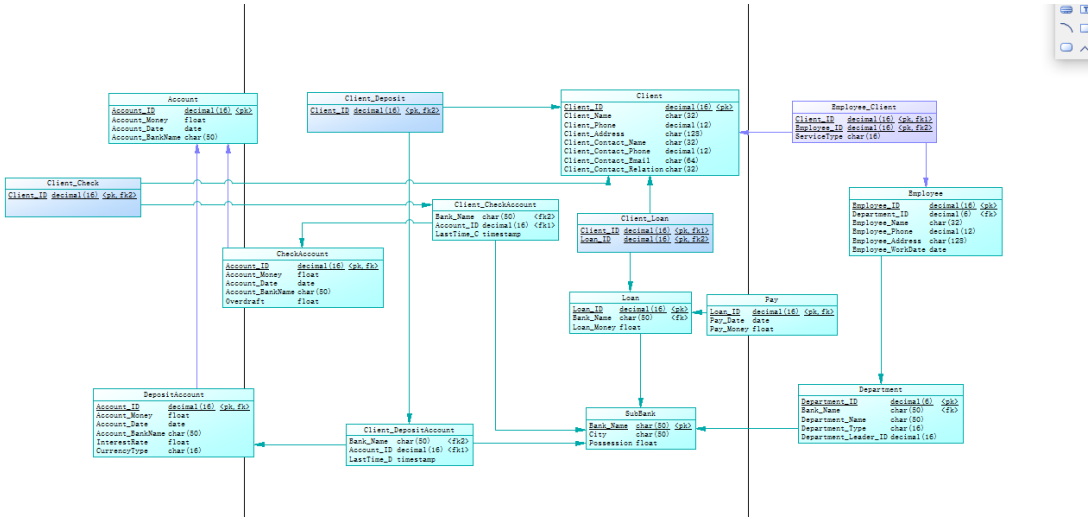


2.3.2 数据库逻辑、物理结构

- cdm结构图：



- pdm结构图：



为在页面上进行表格信息的完整展示，所有的属性都设置成not null，避免出现表头与内容错位的情况。

- 支行表：
  - 关系模式：

    支行(支行名，所在城市，资产)
  - mysql表：

```
create table branch (
    bankname     VARCHAR(50)      not null,
    city         VARCHAR(20)      not null,
    money        DOUBLE           not null,
    constraint PK_BANK primary key (bankname)
);
```

  - django中的存储结构：

    为实现支行的登录验证，加一个密码属性

```
class Branch(models.Model):
    city = models.CharField(max_length=20)
    bankname = models.CharField(max_length=50, unique=True)
    money = models.DecimalField(default=0.0, decimal_places=2,
max_digits=15)
    password = models.CharField(default='123456',max_length=6)
    def __str__(self):
        return f'{self.city} {self.bankname}'
```

- 部门表：
  - 关系模式：

    部门(部门号，部门名称，部门类型，部门领导身份证号，支行名)
  - mysql表：

```
create table department (
    ID              CHAR(4)         not null,
    departname      VARCHAR(20)     not null,
    departtype      VARCHAR(15)     not null,
    manager         CHAR(18)        not null,
    branch          VARCHAR(20)     not null,
    constraint PK_DEPARTMENT primary key (departID),
    Constraint FK_BANK_DEPART Foreign Key(branch) References
branch(bankname)
);
```

- django中的存储结构：

```
class Department(models.Model):
    id = models.CharField(max_length=4,primary_key=True)
    departname = models.CharField(max_length=20)
    departtype = models.CharField(max_length=15)
    manager = models.CharField(max_length=18)
    branch = models.ForeignKey(Branch, on_delete=models.CASCADE,
related_name='departments')
    def __str__(self):
        return f'{self.branch} {self.departname}'
```

- 员工表：
  - 关系模式：

    为区分普通员工和经理，加一个员工类型属性，0表示普通员工，1表示经理

    员工(身份证号，姓名，电话号码，家庭住址，开始工作的日期，员工类型，部门号)
  - mysql表：

```
create table employee (
    ID              CHAR(18)        not null,
    empname         VARCHAR(20)     not null,
    empphone        CHAR(11)        not null,
    empaddr         VARCHAR(50)     not null,
    emptype         CHAR(1)         not null,
    empstart        DATE            not null,
    department      CHAR(4)         not null,
    constraint PK_EMPLOYEE primary key (empID),
    Constraint FK_DEPART_EMP Foreign Key(department) References
department(departID),
    Constraint CK_EMPTYPE Check(emptype IN ('0','1'))
);
```

  - django中的存储结构：

    为实现员工的登录验证，加一个密码属性

```
class Employee(models.Model):
    id = models.CharField(max_length=18, primary_key=True)
    empname = models.CharField(max_length=20)
    empphone = models.CharField(max_length=11)
    empaddr = models.CharField(max_length=50)
    emptype = models.CharField(max_length=1,default=0)
    empstart = models.DateField()
    department = models.ForeignKey(Department, on_delete=models.PROTECT,
related_name='employees')
    password = models.CharField(default='123456',max_length=6)

    def __str__(self):
        return f'{self.empname} ({self.id})'
```

- 用户表：
  - 关系模式：

    用户(身份证号，姓名，联系电话，家庭住址，联系人姓名，联系人手机号，联系人Email，联系人与客户关系)
  - mysql表：

```
create table customer (
    ID              CHAR(18)        not null,
    name            VARCHAR(10)     not null,
    phone           CHAR(11)        not null,
    address         VARCHAR(50)     not null,
    contact_phone   CHAR(11)        not null,
    contact_name    VARCHAR(10)     not null,
    contact_Email   VARCHAR(20)     not null,
    relation        VARCHAR(10)     not null,
    constraint PK_CUSTOMER primary key (ID),
);
```

  - django中的存储结构：

    为实现用户的登录验证，加一个密码属性

```
class Customer(models.Model):
    id = models.CharField(max_length=18, primary_key=True)
    name = models.CharField(max_length=10)
    phone = models.CharField(max_length=11)
    address = models.CharField(max_length=50)
    contact_name = models.CharField(max_length=10)
    contact_phone = models.CharField(max_length=11)
    contact_email = models.EmailField(null=True)
    relation = models.CharField(max_length=10)
    password = models.CharField(default='123456',max_length=6)

    def __str__(self):
        return f'{self.name} ({self.id})'
```

- 账户表：
  - 关系模式：

    储蓄账户(账户号，余额，开户日期，货币类型，利率)

支票账户(账户号，余额，开户日期，透支额)

用户-账户(账户号，用户身份证号，支行名，最后访问时间，账户类型)

- mysql表：

  将用户-账户表中的Unique设为(支行，用户身份证号，账户类型)，可实现每个用户在一家支行最多只能开设一个储蓄账户和一个支票账户的要求。

```
create table saveacc (
    accountID       CHAR(6)         not null,
    left_money      DOUBLE          not null,
    creation_time   DATETIME        not null,
    accounttype     VARCHAR(10)     not null,
    interestrate    float           not null,
    currency_type   CHAR(1)         not null,
    constraint PK_SAVEACC primary key (accountID)
);
```

```
create table checkacc (
    accountID       CHAR(6)         not null,
    left_money      DOUBLE          not null,
    creation_time   DATETIME        not null,
    accounttype     VARCHAR(10)     not null,
    overdraw        DOUBLE          not null,
    constraint PK_CHECKACC primary key (accountID)
);
```

```
create table cusforacc (
    accountID       CHAR(6)         not null,
    branch          VARCHAR(20)     not null,
    cusID           CHAR(18)        not null,
    visit_time      DATETIME        not null,
    accounttype     VARCHAR(10)     not null,
    constraint PK_CUSACC primary key (accountID, cusID),
    Constraint FK_BANK_ACCOUT Foreign Key(branch) References
bank(bankname),
    Constraint FK_CUS Foreign Key(cusID) References customer(cusID),
    Constraint FK_CUS_ACC Foreign Key(accountID) References
accounts(accountID) On Delete Cascade,
    Constraint UK Unique Key(branch, cusID, accounttype)
);
```

- django中的存储结构：

  在django中采用模型继承的方法实现储蓄账户和支票账户的建表，将branch和owners属性移到Account中，在代码中通过循环遍历判断同一用户是否在同一支行建立了两个储蓄账户或两个支票账户。

```
class Account(models.Model):
    branch = models.ForeignKey(Branch, on_delete=models.PROTECT,
related_name='accounts')
    owners = models.ManyToManyField(Customer, related_name='accounts')
    left_money = models.DecimalField(default=0.0, decimal_places=2,
max_digits=15)
    creation_time = models.DateTimeField()
    visit_time = models.DateTimeField()

    def __str__(self):
        return f'{self.pk}'
```

```
class SavingAccount(Account):
    currency_type = models.IntegerField()
    interest_rate = models.DecimalField(decimal_places=2, max_digits=15)

    def __str__(self):
        return f'S{self.pk}'
```

```
class CheckingAccount(Account):
    overdraw = models.DecimalField(decimal_places=2, max_digits=15)

    def __str__(self):
        return f'C{self.pk}'
```

- 贷款表：

  为方便比较当前贷款的状态，加一个count属性，表示已发放的贷款金额

  - 关系模式：

    贷款(贷款号，贷款额度，支行名，已发放贷款量)

    用户-贷款(贷款号，用户身份证号)

  - mysql表：

```
create table loan (
    loanID          CHAR(4)         not null,
    amount          DOUBLE          not null,
    bank            VARCHAR(20)     not null,
    count           DOUBLE          not null,
    constraint PK_LOAN primary key (loanID),
    Constraint FK_BANK_LOAN Foreign Key(bank) References bank(bankname)
);
```

```
create table cusforloan (
    loanID          CHAR(4),
    cusID           CHAR(18),
    constraint PK_CUSLOAN primary key (loanID, cusID),
    Constraint FK_LOAN Foreign Key(loanID) References loan(loanID) On
Delete Cascade,
    Constraint FK_CUSL Foreign Key(cusID) References customer(cusID)
);
```

  - django中的存储结构：

```
class Loan(models.Model):
    branch = models.ForeignKey(Branch, on_delete=models.PROTECT,
related_name='loans')
    amount = models.DecimalField(decimal_places=2, max_digits=15)
    customers = models.ManyToManyField(Customer)
    date = models.DateTimeField()
    count = models.DecimalField(decimal_places=2, max_digits=15)

    def __str__(self):
        return f'{self.pk}'
```

- 贷款支付表：
  - 关系模式：

    贷款支付(贷款号，支付日期，支付金额)

    主键(贷款号，支付日期，支付金额)
  - mysql表：

```
create table loanpay (
    loanID          CHAR(4),
    money           DOUBLE,
    paytime         DATETIME,
    constraint PK_LOANPAY primary key (loanID, cusID, money, paytime),
    Constraint FK_LOANPAY_LOAN Foreign Key(loanID) References
loan(loanID) On Delete Cascade
);
```

  - django中的存储结构：

```
class LoanPay(models.Model):
    loan = models.ForeignKey(Loan, on_delete=models.CASCADE,
related_name='payments')
    amount = models.DecimalField(decimal_places=6, max_digits=19)
    date = models.DateTimeField()
    class Meta:
        unique_together=("loan","amount","date")
    def __str__(self):
        return f'{self.loan} {self.amount} {self.date}'
```

- 用户-员工表：

  为方便负责人的查找，设置成一个用户只能由一个员工服务，服务类型为贷款或账户，而每个员工可以服务多个用户。
  - 关系模式：

    用户-员工(用户身份证号，员工身份证号，服务类型)
  - mysql表：

```
create table cusemp (
    empID          CHAR(18)    not null,
    cusID          CHAR(18)    not null,
    servicetype    CHAR(1)     not null,
    constraint PK_CUSEMP primary key (empID),
    Constraint FK_CUSEMP_EMP Foreign Key(empID) References Employee(ID)
On Delete Cascade,
    Constraint FK_CUSEMP_CUS Foreign Key(cusID) References Customer(ID)
On Delete Cascade,
)
```

- django中的存储结构：

```
class CusEmp(models.Model):
    empID = models.ForeignKey(Employee, on_delete=models.CASCADE,
related_name='served', primary_key=True)
    cusID = models.ForeignKey(Customer, on_delete=models.CASCADE,
related_name='serves')
    servicetype = models.CharField(max_length=10)
```

3 详细设计

3.1 运行环境配置

- 建立虚拟环境

  采用系统为ubuntu18.04或ubuntu16.04

注：要在系统安装python3的前提下进行

- 安装pip3

```
sudo apt install python3-pip
```

- 安装virtualenv

```
sudo pip3 install virtualenv
```

- 创建一个独立的python运行环境，命名为env

```
virtualenv -p python3 env
```

- 激活虚拟环境

```
. env/bin/activate
```

- 安装Django

```
cd env/bin
pip install Django==2.1.2
```

- 将文件banksystem解压到本地，启动

```
cd banksystem/bank
python3 manage.py runserver 127.0.0.1:8000
```

- 进入系统操作界面

```
在浏览器中输入网址：http://127.0.0.1:8000/mybank/index/    进入index主界面
```

3.2 模块概述

3.2.1 前后端交互

django采用的是MTV模型，具体指：

- Model(模型)：数据库相关的操作(ORM)
- Template(模版)：模板语法--->将变量（数据库数据）如何巧妙嵌入html页面中
- View(视图)：逻辑处理
- 此外，Django还有一个urls分发器：路径与视图函数的映射关系

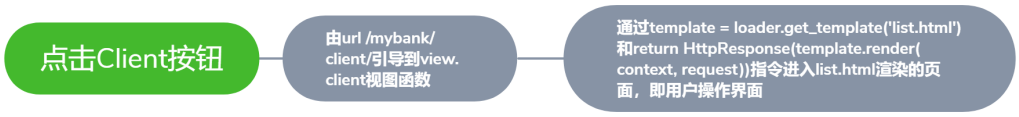
具体操作方式为：

- 在bank/settings.py中将后端数据库更改成为mysql
- 在bank/mybank/models.py中安装django的格式创建模型，相当于mysql的建表
- 在bank/mybank/template中创建前端html文件，即要显示的模板
- 在bank/mybank/urls.py中创建url路由和视图函数的对应关系
- 在bank/mybank/views.py中创建视图函数

3.2.2 增删改查功能的实现

以用户Client模块为例：

- 点击侧边栏查看操作界面：



- 增加新项目：

- 删除项目：

- 改动原项目信息：

- 查询项目：

3.3 用户模块

3.3.1 向后端发送的数据

- 查询：

  下述查询均为模糊查询

  id(身份证号)

  name(姓名)

  phone(联系电话)

  address(地址)

- 修改：

  id(身份证号)

  name(姓名)

  phone(电话号码)

  address(地址)

  empID(负责人身份证号)

  servicetype(服务类型)

  contact_name(联系人姓名)

  contact_phone(联系人电话号码)

contact_email(联系人邮箱)

relation(联系人关系)

- 删除：

  id(身份证号)

- 新增：

  id(身份证号)

  name(姓名)

  phone(电话号码)

  address(地址)

  empID(负责人身份证号)

  servicetype(服务类型)

  contact_name(联系人姓名)

  contact_phone(联系人电话号码)

  contact_email(联系人邮箱)

  relation(联系人关系)

3.3.2 后端的反馈

- 查询：查询得到的数据
- 修改：若无错误提示信息则证明修改成功
- 删除：删除成功/错误提示信息
- 新增：若无错误提示信息则证明新增项目成功

3.4 账户模块

3.4.1 向后端发送的数据

- 查询：

  下述查询均为模糊查询：

  id(账户)

  branch__id(账户所在银行)

  owners__name(账户所有者)

- 修改

  branch_id(开户支行)

  owners(账户所有者)

  creation_time(创建时间)

  visit_time(最后访问日期)

  储蓄账户：

  interest_rate(利率)

  currency_type(货币类型)

  支票账户：

  overdras(透支额)

- 删除

id(账户号)

- 新增

  branch_id(开户支行)

  owners(账户所有者)

  creation_time(创建时间)

  visit_time(最后访问日期)

  储蓄账户：

  interest_rate(利率)

  currency_type(货币类型)

  支票账户：

  overdras(透支额)

3.4.2 后端的反馈

- 查询：查询得到的数据
- 修改：若无错误提示信息则证明修改成功
- 删除：删除成功/错误提示信息
- 新增：若无错误提示信息则证明新增项目成功

3.5 贷款模块

3.5.1 向后端发送的数据

- 查询

  下述查询均为模糊查询：

  id(贷款号)

  branch__bankname(所在支行)

  customers__name(贷款所有者)

- 修改

  branch_id(开户支行)

  customer_id(贷款所有者)

  amount(贷款总量)

- 删除

  id(贷款号)

- 新增

  branch_id(开户支行)

  customer_id(贷款所有者)

  amount(贷款总量)

3.5.2 后端的反馈

- 查询：查询得到的数据
- 修改：若无错误提示信息则证明修改成功
- 删除：删除成功/错误提示信息
- 新增：若无错误提示信息则证明新增项目成功

3.6 贷款支付模块

3.6.1 向后端发送的数据

- 新增：

  loan_id(贷款号)

  amount(金额)

  date(日期)

3.6.2 后端的反馈

- 新增：若无错误提示信息则证明新增贷款支付成功

3.7 业务统计模块

3.7.1 向后端发送的数据

- 查询

  freq(查询时间频率：年、月、季)

  start_date(查询开始时间)

  end_date(查询结束时间)

3.7.2 后端的反馈

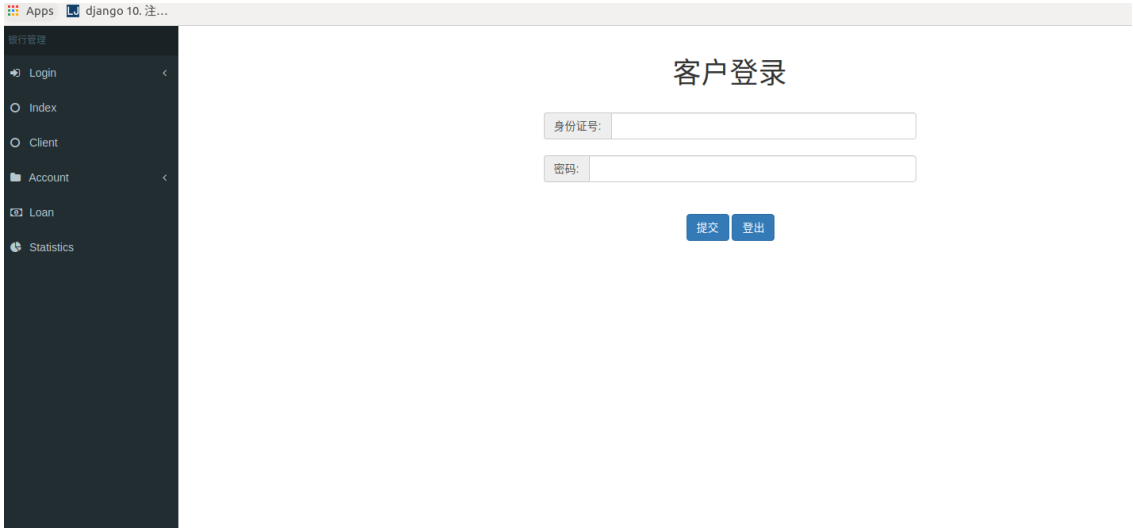
- 统计该时间跨度内注册的总账户数、总储蓄额、贷款发放个数、贷款发放金额
- 按年、月、季对各个支行在不同时间段开设账户数、储蓄金额、贷款数、贷款金额，以及不同时间段发放贷款数、贷款金额进行统计，以表格形式列举出来
- 对账户申请个数、贷款申请个数以及贷款发放次数以曲线图的形式描绘出来，纵轴为个数，横轴为时间戳

4 实现与测试

4.1 实现结果

- 银行管理系统主页

- 客户登录界面



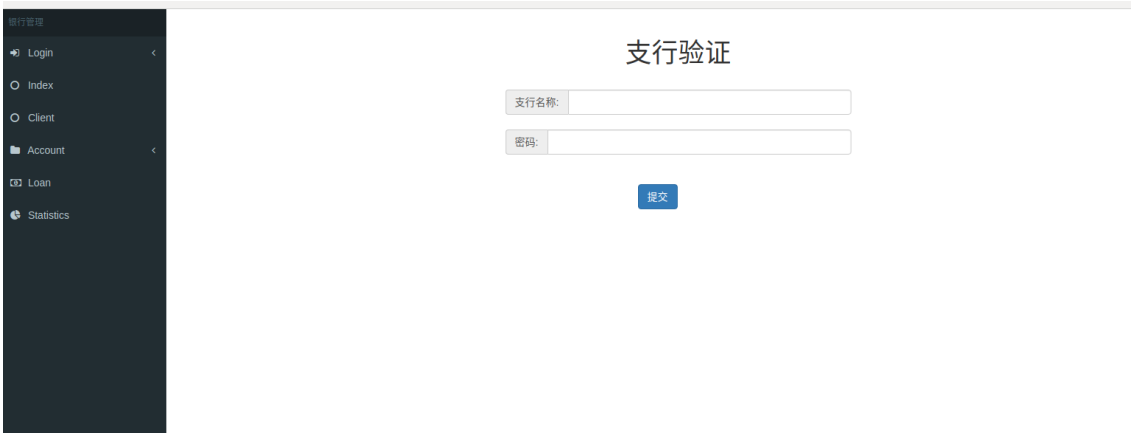
- 用户操作界面

- 储蓄账户操作界面

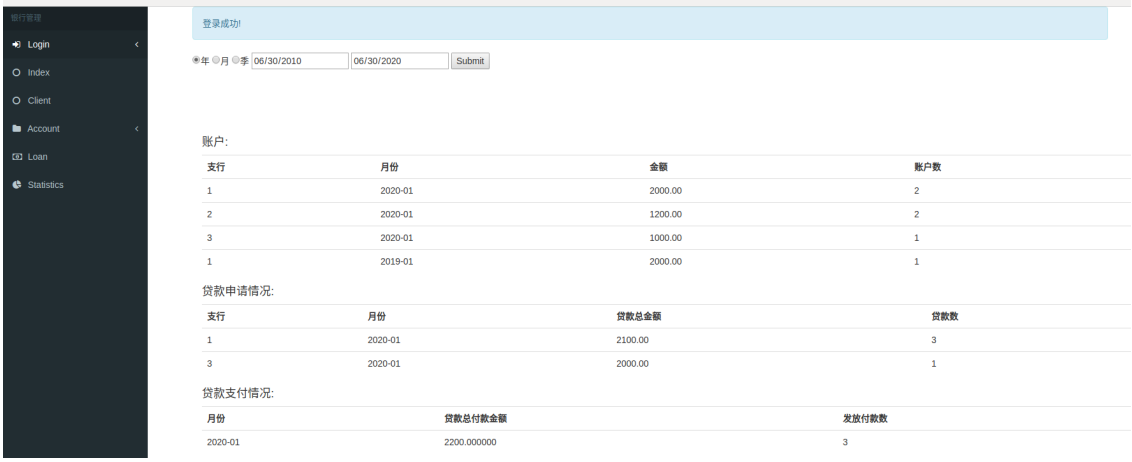
- 支票账户操作界面

- 贷款操作界面

- 支行验证界面

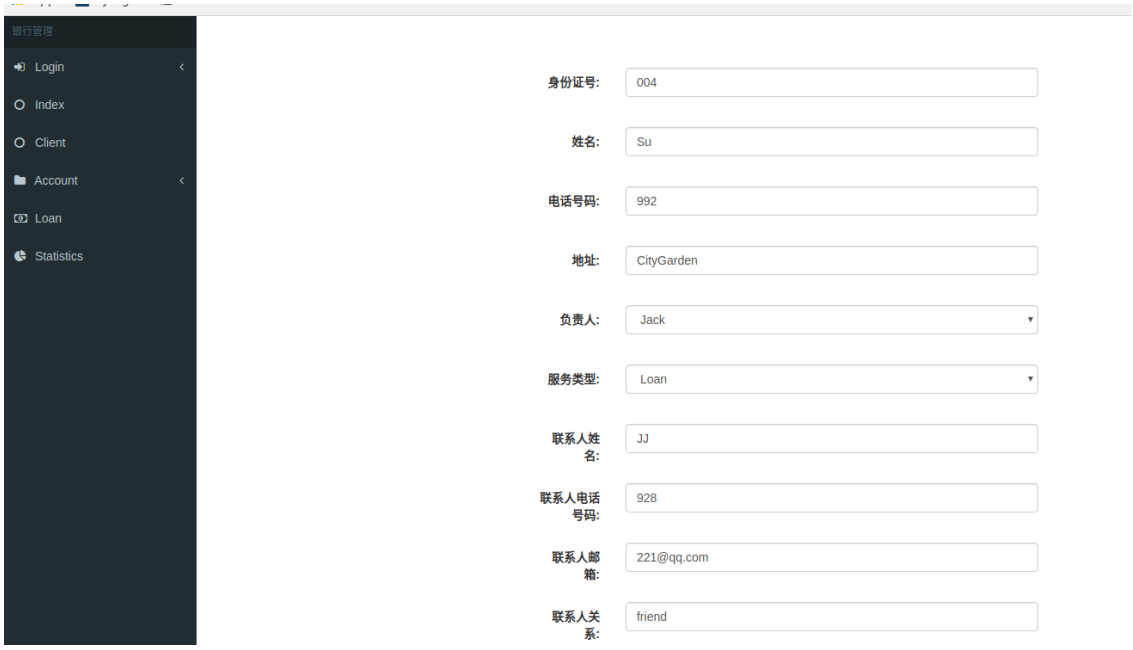


- 业务统计界面



4.2 测试结果

- 用户项目修改及新增页面：



- 用户项目修改及新增页面错误测试：

- 用户项目查询界面：

身份证号	姓名	电话号码	地址	负责人	服务类型	联系人姓名	联系人电话号码	联系人邮箱	联系人关系		
002	May	689	CentrePark	Jane	Account	Fei	908	190@qq.com	friend	修改	删除
003	Kelly	229	CityGarden	Kit	Loan	mm	339	228@qq.com	friend	修改	删除

- 储蓄账户新增及修改页面：

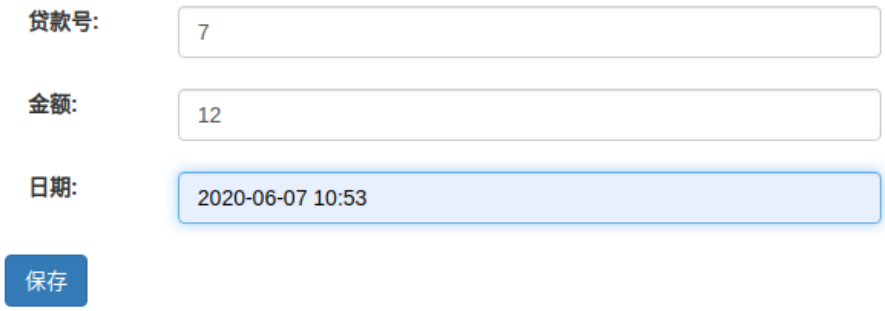
- 储蓄账户新增及修改页面错误测试：

- 储蓄账户查询界面

账户号	开户支行	账户所有者	余额	创建时间	最后访问日期	利率	货币类型		
5	Agricuture Bank of China	May Kelly	1000.00	2020年6月3日 09:09	2020年6月30日 08:08	0.30	2	修改	删除
6	Bank of China	May Kelly	1000.00	2020年6月30日 11:11	2020年6月30日 14:22	1.00	1	修改	删除

- 贷款申请界面：

- 贷款增加界面：



- 贷款状态查看界面：

- 按月业务统计界面:

账户:

支行	月份	金额	账户数
1	2020-06	2000.00	2
2	2020-06	1000.00	1
1	2019-09	2000.00	1
2	2020-02	200.00	1
2	2019-12	1000.00	1

贷款申请情况:

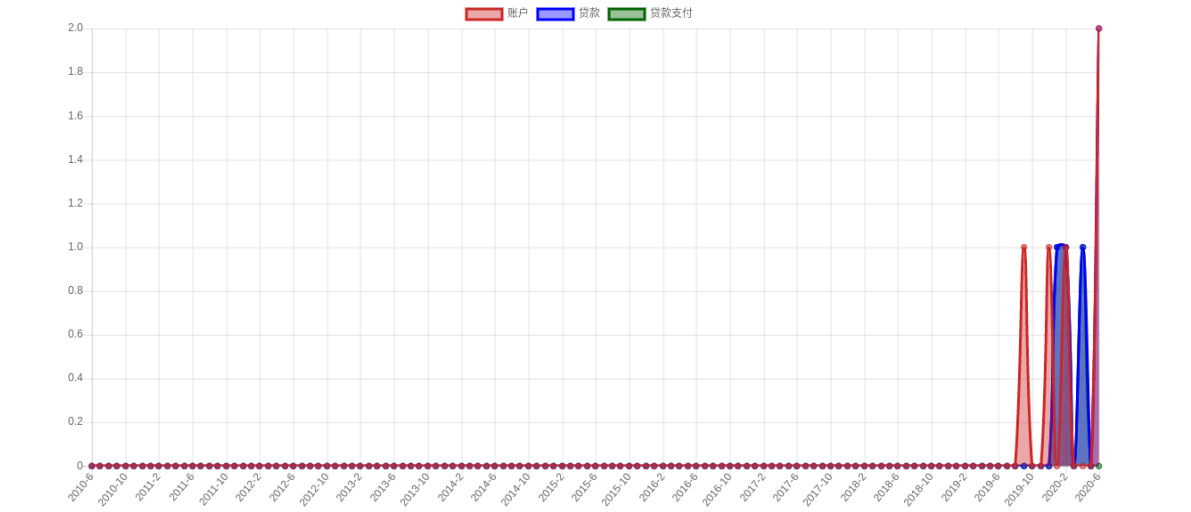
支行	月份	贷款总金额	贷款数
1	2020-04	1000.00	1
1	2020-02	1000.00	1
1	2020-06	100.00	1
3	2020-01	2000.00	1
3	2020-06	12.00	1

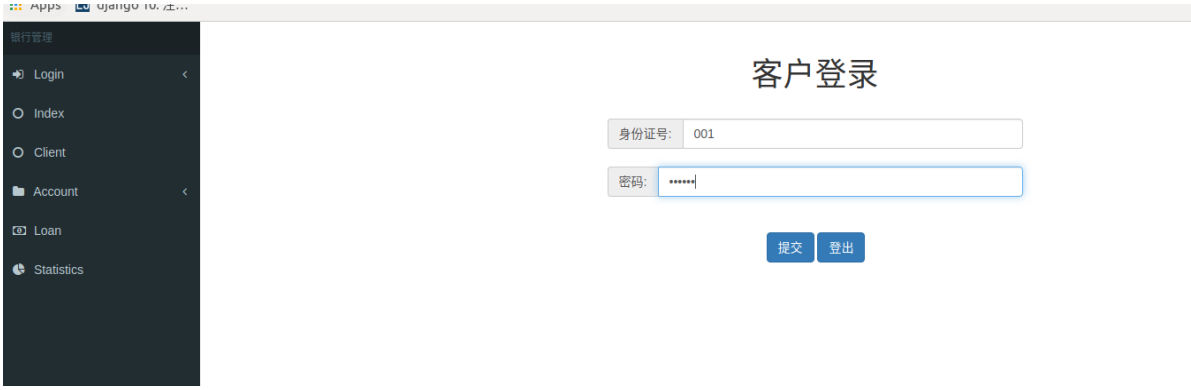
贷款支付情况:

月份	贷款总付款金额	发放付款数
2020-04	100.000000	1
2020-02	100.000000	1
2020-01	2000.000000	1

- 业务统计曲线图:

5 权限与登录管理





如图，以001的身份登录后无法删除与其无关的用户、贷款、账户信息

6 总结与讨论

收获：学习了django及前端开发技术，深入理解了数据库原理，对数据库应用到具体开发中有了深刻的体会和认识。同时锻炼了代码能力和数据库的设计能力！