

“银行业务管理系统”

系统设计与实验报告

王嵘晟 PB17111614

2020 年 6 月 30 日

目录

1	概述	2
1.1	系统目标	2
1.2	需求说明	2
1.3	本报告的主要贡献	3
2	总体设计	3
2.1	系统模块结构	3
2.2	系统工作流程	3
2.3	数据库设计	4
3	详细设计	4
3.1	后端 mysql 建立数据表并插入初始数据	4
3.2	后端 python+flask 架构	5
3.2.1	建立数据库连接	5
3.2.2	定义数据库表的类	5
3.2.3	填写函数完成对数据库的增删改查操作	6
3.2.4	后端接口的创建	8
3.3	前端	8
3.3.1	主框架 APP.vue	9
3.3.2	功能网页	10

4 实现与测试	11
4.1 实现结果	11
4.2 测试结果	13

1 概述

1.1 系统目标

本实验要求搭建一个银行管理系统，完成对客户信息、账户信息和贷款信息的增删改查功能，同时后台数据库还需维护和保持这三类数据之间的一些约束，进而完成业务查询绘制表格

1.2 需求说明

根据以下描述完成银行业务管理系统： 银行有多个支行。各个支行位于某个城市，每个支行有唯一的名字。银行要监控每个支行的资产。银行的客户通过其身份证号来标识。银行存储每个客户的姓名、联系电话以及家庭住址。为了安全起见，银行还要求客户提供一位联系人的信息，包括联系人姓名、手机号、**Email** 以及与客户的关系。客户可以有帐户，并且可以贷款。客户可能和某个银行员工发生联系，该员工是此客户的贷款负责人或银行帐户负责人。银行员工也通过身份证号来标识。员工分为部门经理和普通员工，每个部门经理都负责领导其所在部门的员工，并且每个员工只允许在一个部门内工作。每个支行的管理机构存储每个员工的姓名、电话号码、家庭地址、所在的部门号、部门名称、部门类型及部门经理的身份证号。银行还需知道每个员工开始工作的日期，由此日期可以推知员工的雇佣期。银行提供两类帐户——储蓄帐户和支票帐户。帐户可以由多个客户所共有，一个客户也可开设多个账户，但在一个支行内最多只能开设一个储蓄账户和一个支票账户。每个帐户被赋以唯一的帐户号。银行记录每个帐户的余额、开户日期、开户的支行名以及每个帐户所有者访问该帐户的最近日期。另外，每个储蓄帐户有利率和货币类型，且每个支票帐户有透支额。每笔贷款由某个分支机构发放，能被一个或多个客户所共有。每笔贷款用唯一的贷款号标识。银行需要知道每笔贷款所贷金额以及逐次支付的情况（银行将贷款分几次付给客户）。虽然贷款号不能唯一标识银行所有为贷款所付的款项，但可以唯一标识为某贷款所付的款项。对每次的付款需要记录日期和金额。

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

针对以上要求加入了自己的设计与实现
描述了使用的技术栈以及实验编写环境
对于实验成品做了相应的测试
有助于今后反思与总结，以实现更加复杂的作品

2 总体设计

2.1 系统模块结构

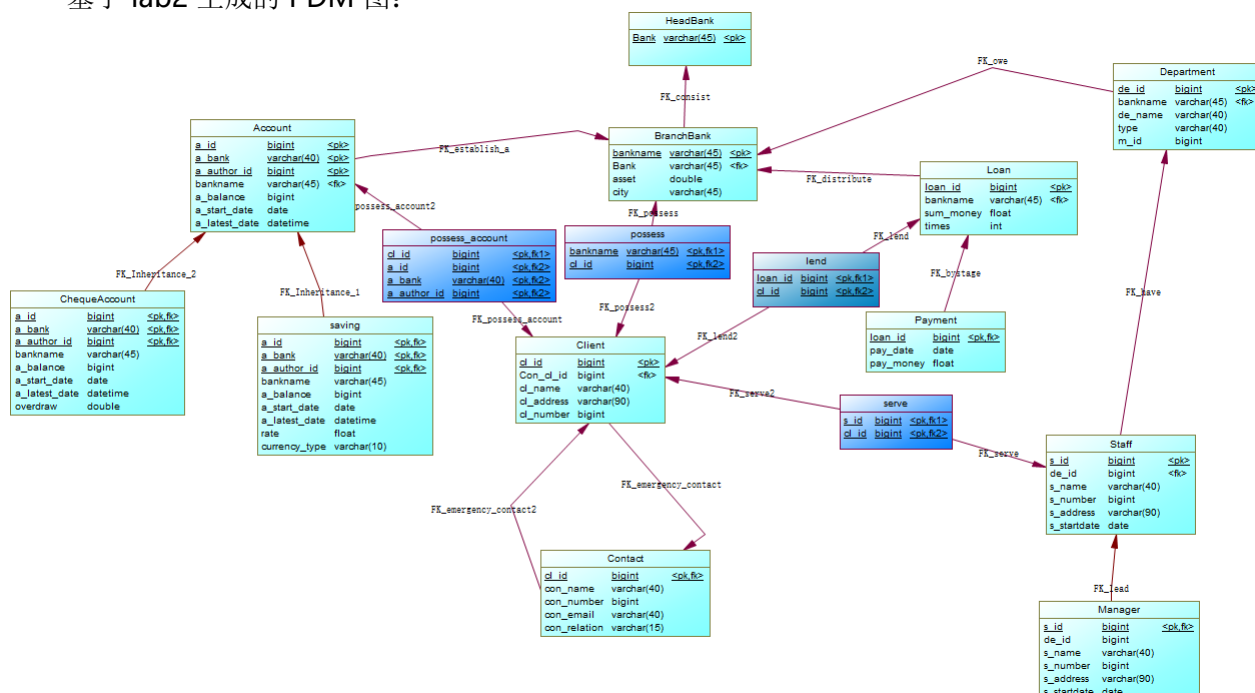
本实验使用了 `python+flask+mysql` 作为后端，`vue.js` 作为前端，实现了一个前后端分离的网页端 APP

2.2 系统工作流程

网页端有输入框以及确认和清空按钮，只有当输入框内填入足够的内容才可以点击确认按钮，然后前端程序从输入框中提取所需要的数据，通过路由传递给后端 `python` 程序，进而在 `python` 程序的函数中将数据写入数据库或者对于数据库中的内容做出查找、修改、删除

2.3 数据库设计

基于 lab2 生成的 PDM 图:



同时结合助教给出的 **demo**，最终设计出来了 13 个表，同时为了避免循环依赖带来的死锁，将部分外键设置为可以为空

3 详细设计

综述： 在设计数据库时，认为所有的 ID 内容都是完全由数字构成的，输入非数字内容会被认为不合法。客户姓名允许使用标点，因为某些拉丁语系名字中会有’ 等符号。

在创建账户时，创建账户和与客户建立联系是分开的，因此默许存在无主账户。（此设计类似先做好借记卡等待客户办理激活）

3.1 后端 mysql 建立数据表并插入初始数据

以创建客户表为例:

```
create table customer (
```

```

        c_identity_code          CHAR(18)          not null,
        c_name                   VARCHAR(10)        not null,
        c_phone                  CHAR(11)           not null,
        c_address                VARCHAR(50),
        c_contact_phone          CHAR(11)           not null,
        c_contact_name           VARCHAR(10)        not null,
        c_contact_email          VARCHAR(20),
        c_contact_relationship    VARCHAR(10)       not null,
        c_loan_staff_identity_code CHAR(18),
        c_account_staff_identity_code CHAR(18),
        constraint PK_CUSTOMER primary key (c_identity_code),
        Constraint FK_CL_LOAN Foreign Key(c_loan_staff_identity_code) References st
        Constraint FK_CL_ACCOUNT Foreign Key(c_account_staff_identity_code) Referen
    );

```

插入初始数据包括分行情况、部门情况、员工情况、部门经理

3.2 后端 python+flask 架构

3.2.1 建立数据库连接

相关 Python 代码如下:

```

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = "mysql://david:1234@localhost/bank"
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

```

3.2.2 定义数据库表的类

此处以创建账户 `account` 表为例:

```

class Account(db.Model):
    a_code = db.Column(db.String(CODE_LEN), primary_key=True)
    a_balance = db.Column(db.Float, nullable=False)

```

```

a_open_date = db.Column(db.Date, nullable=False)
# 外键
a_open_bank = db.Column(
    db.String(NAME_LEN),
    db.ForeignKey('sub_bank.sb_name'),
    nullable=False
)
# 赋值语句
def __init__(self, a_code, a_balance, a_open_date, a_open_bank):
    self.a_code = a_code
    self.a_balance = float(a_balance)
    self.a_open_date = a_open_date
    self.a_open_bank = a_open_bank

```

定义了 account 表以及 a_code, a_balance, a_open_date, a_open_bank 这四个属性同时将 a_open_bank 作为外键

3.2.3 填写函数完成对数据库的增删改查操作

在此以对于客户的管理为例，创建：

```

if args['tab'] == '0':
if args['loan_staff_id'] != '' and result['status']:
    staff = Staff.query.filter(
        Staff.s_identity_code == args['loan_staff_id']).first()
    if staff is None:
        result['status'] = False
        result['message'] = '员工不存在'

if args['account_staff_id'] != '' and result['status']:
    staff = Staff.query.filter(
        Staff.s_identity_code == args['account_staff_id']).first()
    if staff is None:
        result['status'] = False

```

```

result['message'] = '员工不存在'

if result['status']:
    customer = Customer.query.filter(
        Customer.c_identity_code == args['customer_id']).first()
    if customer is not None:
        result['status'] = False
        result['message'] = '客户 ID 已经存在'

    if result['status']:
        init_data = {
            convert_dict[k]: args[k]
            for k in args if args[k] != '' and not k.startswith('m_')
            and not k.startswith('tab')
        }
        db.session.add(Customer(**init_data))
        db.session.commit()
        result['message'] = '插入成功'

```

对于前端读入的数据，首先判断贷款负责人和账户负责人是否存在，不存在则直接报错，否则向下判断客户 ID 是否存在，不存在则继续向下，可以创建该客户的信息，完成创建。
删除：

```

elif args['tab'] == '1':
    queries = {
        convert_dict[k]: args[k]
        for k in convert_dict if args[k] != ''
        and not k.startswith('m_') and not k.startswith('tab')
    }
    customers = Customer.query.filter_by(**queries).all()
    if len(customers) == 0:
        result['status'] = False
        result['message'] = '没有查找到待删除内容'
    if result['status']:

```

```

for c in customers:
    car_account = CheckingAccountRecord.query.filter_by(
        car_c_identity_code=c.c_identity_code).first()
    sar_account = SavingsAccountRecord.query.filter_by(
        sar_c_identity_code=c.c_identity_code).first()
    customer_loan = LoanCustomer.query.filter_by(
        lc_c_identity_code=c.c_identity_code).first()
    if car_account is not None or sar_account is not None or customer_loan is not None:
        result['status'] = False
    result['message'] = '客户不可删除'
    if result['status']:
        customers_num = len(customers)
    for c in customers:
        db.session.delete(c)
    db.session.commit()
    result['message'] = '删除成功'

```

删除数据时，直接根据主键查找数据表，找到对应的要删除的元素，如果客户已经绑定了账户，则不可删除。当查找数据不为空时，满足条件可以删除。对于数据表的更改与查找操作类似，不在此赘述。

3.2.4 后端接口的创建

对于要实现的 4 个功能，创建了相应的 4 个 post 用来做前后端的接口

```

api.add_resource(BusinessStatistic, '/api/business-statistic')
api.add_resource(CustomerManagement, '/api/customer-management')
api.add_resource(AccountManagement, '/api/account-management')
api.add_resource(LoanManagement, '/api/loan-management')

```

3.3 前端

前端分为银行主页、客户管理页、账户管理页、贷款管理页、业务统计页这五个网页，使用 vue.js 架构中的 drawer 来呈现。

前端主模块 APP.vue 定义了网页层次等内容，router.vue 是路由协议，完成了前后端接口的连接，components 文件夹下的 vue 文件则是对于具体功能网页的实现。

3.3.1 主框架 APP.vue

```
<template>
  <v-app>
    <v-navigation-drawer v-model=" drawer" app color=" blue" >
      <v-list dense>
        <v-list-item v-for=" item in drawerData" :key=" item.
          <v-list-item-icon>
            <v-icon>{{ item.icon }}</v-icon>
          </v-list-item-icon>
          <v-list-item-content>
            <v-list-item-title>{{ item.title }}</v-list-item-content>
          </v-list-item>
        </v-list>
      </v-navigation-drawer>
    <v-app-bar app color=" darkgrey" dark>
      <v-app-bar-nav-icon @click.stop=" drawer=!drawer" ></v-app-
      <v-toolbar-title>银行业务管理系统</v-toolbar-title>
    </v-app-bar>
    <v-content>
      <router-view></router-view>
    </v-content>
  </v-app>
</template>

<script>
  export default {
    name: " App" ,

    data: () => ({
      drawer: null ,
      drawerData: [
        {

```

```

        title: " 银行主页" ,
        targetPath: "/"
    },
    {
        title: " 客户管理页" ,
        targetPath: "/customer-manage"
    },
    {
        title: " 账户管理页" ,
        targetPath: "/account-manage"
    },
    {
        title: " 贷款管理页" ,
        targetPath: "/loan-manage"
    },
    {
        title: " 业务统计页" ,
        targetPath: "/business-statistic"
    }
]
    })
};
</script>

```

使用 **vue.js** 架构大多都是重复工作，所以在此只详细列举最顶层的 **APP.vue** 模块的代码，其余代码不做说明。

3.3.2 功能网页

完成输入框、按钮的模式设计，同时将读入的 **model data** 通过路由传递给后端，此处不展开代码。

4 实现与测试

4.1 实现结果

银行主页: 显示了一些提前插入的数据

≡ 银行业务管理系统

银行分行与职员介绍

网点分布

分行名	城市
北京分行	北京
上海分行	上海
合肥分行	合肥
广州分行	广州
深圳分行	深圳

客户管理界面:

≡ 银行业务管理系统

需要创建的用户的信息

ID

0 / 18

姓名

0 / 10

电话号码

0 / 11

家庭住址

0 / 50

联系人姓名

0 / 10

联系人电话

0 / 11

联系人邮箱

0 / 20

与客户关系

0 / 10

贷款负责人

0 / 10

账户负责人

0 / 10

账户管理界面:

≡ 银行业务管理系统

账户管理

账户所有人管理

创建

删除

更改

查找

需要创建的账户的信息

ID

0 / 18

余额

0 / 10

开户银行

0 / 20

开户日期

账户类型

▼

确认

清空

贷款管理界面：

≡ 银行业务管理系统

贷款信息

贷款人信息

支付情况

创建

删除

查找

需要创建的贷款的信息

贷款 ID

0 / 18

贷款银行

0 / 20

Loan Money

0 / 10

确认

清空

业务统计界面：

Please Input Information

业务类型

时间跨度

起始时间

终止时间

图/表

COMMIT

RESET

4.2 测试结果

插入数据后的数据库截图:

客户表:

```
mysql> select * from customer;
```

c_identity_code	a_name	c_phone	c_address	c_contact_phone	c_contact_name	c_contact_email	c_contact_relationship	c_loan_staff_identity_code
0001	ca	13313562656	Becker S. T. 221B	13535617461	S. H.	1546@mail	friends	staff_1
0002	cb	13313562656	Becker S. T. 221B	13535617461	S. H.	1546@mail	friends	staff_1
0003	cc	13313562656	Becker S. T. 221B	13535617461	S. H.	1546@mail	friends	staff_1
0004	cd	13313562656	Becker S. T. 221B	13535617461	S. H.	1546@mail	friends	staff_1
100	1	1	2	1	a	a	1	NULL
2	2	2	2	2	2	2	2	NULL
200	11	11	11	11	11	11	11	NULL

账户表:

```
mysql> select * from account;
```

a_code	a_balance	a_open_date	a_open_bank
001	500	2020-06-02	beijing bank
002	500	2020-06-02	guangzhou bank
003	500	2020-06-02	guangzhou bank
004	500	2020-06-02	guangzhou bank
100	100	2020-06-01	beijing bank
2	2	2020-06-01	beijing bank
400	500	2020-06-06	shanghai bank

```
7 rows in set (0.00 sec)
```

给客户连接账户:

创建:

[创建](#)
[删除](#)
[更改](#)
[查找](#)

需要创建的所有者的信息

账户 ID

001

3 / 18

客户 ID

0001

4 / 18

开户银行

beijing bank

12 / 20

最近访问日期

2020-06-24

账户类型

Checking

▼

确认

清空

创建所有者结果

插入成功

删除: (选择了未创建的数据删除失败):

需要删除的所有者的信息

账户 ID0013 / 18

客户 ID00024 / 18

开户银行beijing bank12 / 20

最近访问日期2020-06-24

账户类型Checking

确认

清空

删除所有者结果

删除失败，无待删除内容

更改 (贷款账户只能更改访问日期):

≡ 银行业务管理系统

beijing bank12 / 20

2020-06-24

Checking

更新前的所有者信息

账户 ID0 / 18

客户 ID0 / 18

开户银行0 / 20

最近访问日期2020-06-26

账户类型

确认

清空

更新所有者结果

更改成功

查找:

三 银行业务管理系统

需要搜索的所有者信息

账户 ID
001
3 / 18

客户 ID
0001
4 / 18

开户银行
beijing bank
12 / 20

最近访问日期
2020-06-26

账户类型
Checking

确认

清空

搜索所有者结果

Index	Account ID	Account Type	Customer ID	Account Open Bank	Account Last Visit Date
0	001	Checking	0001	beijing bank	2020-06-26

输入不合法数据:

账户管理

账户所有人管理

创建

删除

更改

查找

需要创建的账户的信息

ID
ojok
Invalid input!
4 / 18

余额
0 / 10

开户银行
0 / 20

开户日期

账户类型

新建贷款:

创建

删除

查找

需要创建的贷款的信息

贷款 ID
01
2 / 18

贷款银行
shanghai bank
13 / 20

Loan Money
600
3 / 10

确认

清空

创建贷款状态

新建贷款成功

贷款与不同客户连接:

贷款信息

贷款人信息

支付情况

创建

删除

查找

需要创建的贷款所有人的信息

贷款 ID
01
2 / 18

客户 ID
0001
4 / 18

确认

清空

创建贷款所有人状态

插入成功

创建

删除

查找

需要搜索的贷款所有人信息

贷款 ID
01
2 / 18

客户 ID
0001
4 / 18

确认

清空

搜索贷款所有人状态

Index	Loan ID	Customer ID
0	01	0001

贷款信息

贷款人信息

支付情况

创建

删除

查找

需要创建的贷款所有人的信息

贷款 ID
01
2 / 18

客户 ID
0002
4 / 18

确认

清空

创建贷款所有人状态

插入成功

需要搜索的贷款所有人信息

贷款 ID
01

2 / 18

客户 ID

0 / 18

确认

清空

搜索贷款所有人状态

Index	Loan ID	Customer ID
0	01	0001
1	01	0002

贷款分两次发放：

≡ 银行业务管理系统

需要创建的贷款支付的信息

贷款 ID
01

2 / 18

贷款号
01.1

4 / 18

贷款支付金额
60

2 / 10

贷款支付日期
2020-06-01

确认

清空

创建贷款支付状态

插入成功

需要创建的贷款支付的信息

贷款 ID 01	贷款号 01.2
2 / 18	4 / 18
贷款支付金额 60	
2 / 10	
贷款支付日期 2020-06-11	

确认

清空

创建贷款支付状态

插入成功

贷款支付日期

确认

清空

搜索贷款支付状态

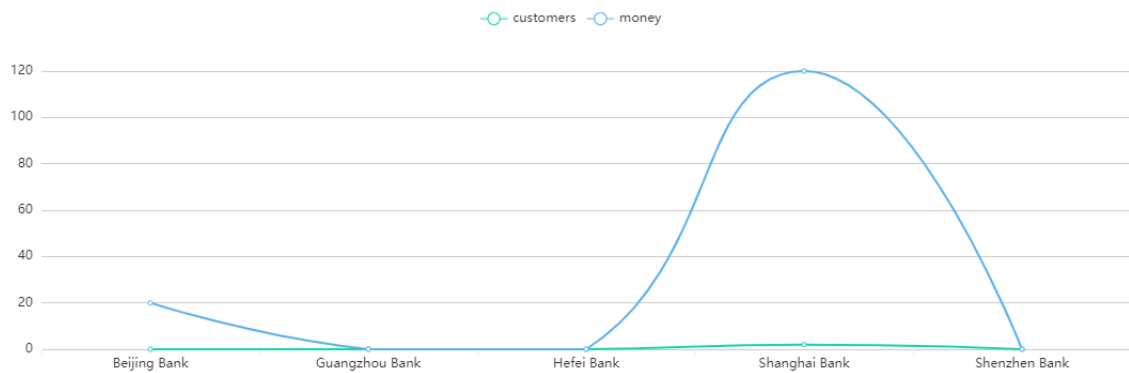
Index	Loan ID	Loan Payment ID	Loan Payment Date	Loan Payment Money
0	01	01.1	2020-06-01	60
1	01	01.2	2020-06-11	60

业务统计（这里北京支行的客户被我不小心删除了…）：

Loan 2020-06-01 to 2020-06-30

Index	Bank Name	Customers	Money
0	Beijing Bank	0	20
1	Guangzhou Bank	0	0
2	Hefei Bank	0	0
3	Shanghai Bank	2	120
4	Shenzhen Bank	0	0

Loan 2020-06-01 to 2020-06-30



5 总结与讨论

本实验实现了一个前后端分离的银行管理系统，并完成了所有需求。我在进行实验的过程中熟练掌握了 ‘SQLAlchemy’ 的用法，掌握了主流前端框架 ‘Vue.js’ 和后端框架 ‘Flask’ 的使用，对跨域请求和反向代理等知识也有了深入的了解，对开发前后端分离应用也积累了宝贵的经验，收获非常大。