银行业务管理系统 系统设计与实验报告

- 姓名：德斯别尔
- 学号：PB19111675

银行业务管理系统 系统设计与实验报告

1 概述
- 1.1系统目标
- 1.2需求说明

2 总体设计
- 2.1系统模块结构
- 2.2系统工作流程
- 2.3数据库设计

3 详细设计
- 3.0 前言
- 3.1后端
  - 工具链
  - app/database.py
  - app/functions.py
- 3.2 前端
  - 工具链及文件组织
  - 网页

4 实现与测试
- 4.1实现结果
- 4.2测试结果
- 4.3 实验中的难点问题及解决

5 总结与讨论

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

- **数据库**:

  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。

- **功能**：
    - 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，要求对统计结果提供表格形式展示，这里为了增加可视化程度，我自己又添加了图形化展示的功能。

2 总体设计

2.1系统模块结构

本系统是一个前后端分离的网页APP，采用BS架构，主要的逻辑模块分为前端部分和后端部分。下面分别进行阐述：

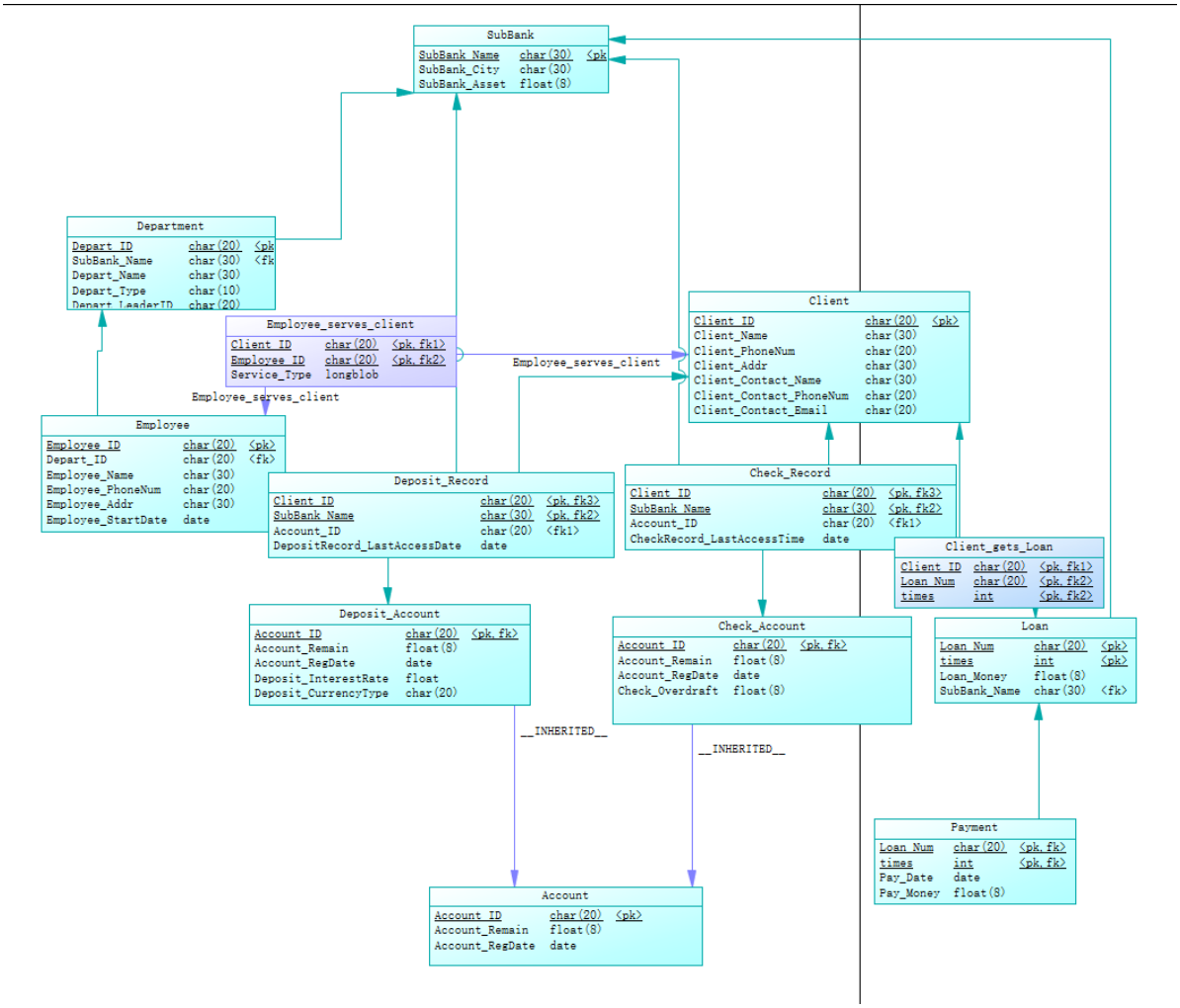
- **前端**：主要分为5个模块，对应网页应用的5个界面，分别是银行主页、客户管理、账户管理、贷款管理和业务统计，每个界面分别对应需求中一大类功能，并提供了一些按钮和输入框来与用户交互。
- **后端**：主要分为两个模块，数据库模块和接口（API）模块，数据库模块中定义了Mysql数据库中各个table中的字段的类型以及table之间的外键关系；接口模块负责接收、处理并回复来自前端的信息。

2.2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。当输入了满足要求的数据并点击`COMMIT`按钮后，前端向后端服务器`post`用户填入的所有数据，服务器在解析用户的信息后，按照对应的操作检查参数是否合法，若合法则执行对应的操作并返回执行后的状态信息到前端服务器，前端服务器将接收到的消息渲染在页面上，从而完成一次与服务器的交互。

2.3数据库设计

数据需求与 Lab 02 相同，可参考1.2需求说明中关于数据库的部分，这里展示一下当时生成的物理模型图：



需要额外说明的是，在具体进行lab3时，为了避免循环依赖，我把所有的外键属性均设置为可以为空，通过限定操作数据库的代码保证部分不能为空的外键列满足需求。

3 详细设计

3.0 前言

- 不区分`开户`概念和`拥有账户所有权`概念
- `账户`实体的`开户银行`、`开户日期`、`账户类型`不能直接修改，仅可以通过删除再新建的形式修改
- 为了使UI逻辑清晰，将`开户`和`创建贷款`这两类的行为分解为创建`账户`或者`贷款`，再添加`账户所有人`或着`贷款所有人`这两个行为序列，这可能会导致存在无人所有的账户
- 删除`账户`或者`贷款`的时候，会自动将`账户`或者`贷款`关联的`顾客`一并删除
- 支持批量操作，例如：把所有`联系人电话`为`123`的`顾客`的`贷款负责人`改为`staff_2`
- 更改`顾客`的`身份证号`的同时会更改相应的`借贷信息`和`账户所有`信息
- 统计界面，储蓄业务的统计为在规定时间段内所开户的所有储蓄账户的余额总数，贷款业务的统计为在规定时间段内所有贷款发放的金额总和
- 当输入的贷款支付的金额大于贷款剩余未支付的额度时，会反馈为超额，提醒用户重新设置支付金额
- 本实验所需环境可参考 back\requirements.txt 文件

本实验工程量较大，故结合部分伪代码讲解既简洁又便于理解。

以下先介绍模块使用的工具链：

3.1后端

工具链

用`Flask`作为后端框架，基于`flask_restful`提供`RESTful`的API，基于`flask_sqlalchemy`这个orm框架对数据库操作，对于一个轻量级Web App来说，使用`Flask`框架有助于加快开发的速度。

app/database.py

在这个文件中我实现了对所有数据库table的定义，以这段代码为例：

```
class Customer(db.Model):
    # start attributes
    c_identity_code = db.Column(db.String(IDENTITY_CODE_LEN), primary_key=True)
    c_name = db.Column(db.String(NAME_LEN), nullable=False)
    c_phone = db.Column(db.String(PHONE_LEN), nullable=False)
    c_address = db.Column(db.String(ADDRESS_LEN), nullable=False)
    c_contact_name = db.Column(db.String(IDENTITY_CODE_LEN), nullable=False)
    c_contact_phone = db.Column(db.String(NAME_LEN), nullable=False)
    c_contact_email = db.Column(db.String(EMAIL_LEN), nullable=False)
    c_contact_relationship = db.Column(db.String(RELATIONSHIP_LEN),
nullable=False)
    # end attributes

    # start foreign keys
    c_loan_staff_identity_code = db.Column(
        db.String(IDENTITY_CODE_LEN),
        db.ForeignKey('staff.s_identity_code'),
        nullable=True
    )
    c_account_staff_identity_code = db.Column(
        db.String(IDENTITY_CODE_LEN),
        db.ForeignKey('staff.s_identity_code'),
        nullable=True
    )
    # end foreign keys

    def __init__(
            self, c_identity_code, c_name, c_phone, c_address, c_contact_name,
            c_contact_email, c_contact_phone, c_contact_relationship,
            c_loan_staff_identity_code=None,
c_account_staff_identity_code=None):
        self.c_identity_code = c_identity_code
        self.c_name = c_name
        self.c_phone = c_phone
        self.c_address = c_address
        self.c_contact_name = c_contact_name
        self.c_contact_email = c_contact_email
        self.c_contact_phone = c_contact_phone
        self.c_contact_relationship = c_contact_relationship
        if c_loan_staff_identity_code is not None:
            self.c_loan_staff_identity_code = c_loan_staff_identity_code
        if c_account_staff_identity_code is not None:
            self.c_account_staff_identity_code = c_account_staff_identity_code
```

```
    def to_json(self):
        return {
            'c_identity_code': self.c_identity_code,
            'c_name': self.c_name,
            'c_phone': self.c_phone,
            'c_address': self.c_address,
            'c_contact_name': self.c_contact_name,
            'c_contact_email': self.c_contact_email,
            'c_contact_phone': self.c_contact_phone,
            'c_contact_relationship': self.c_contact_relationship,
            'c_loan_staff_identity_code': self.c_loan_staff_identity_code,
            'c_account_staff_identity_code': self.c_account_staff_identity_code
        }

    pass
```

定义了名为`Customer`的tabel和若干对应的信息，其余的数据库定义不再赘述，可以参照database.py文件。

app/functions.py

在这个文件中我实现了所有的接口定义，本实验中共有4个后端接口，分别为`/api/business-statistic`、`/api/customer-management`、`/api/account-management`、`/api/loan-management`。不妨使用伪代码的方式以`/api/customer-management`为例子进行讲解：

```
class CustomerManagement(Resource):
    # 使用flask内置的parser定义post的参数
    parser = reqparse.RequestParser()
    parser.add_argument(
            'xxx', ...
    )
    # 定义post时的行为
    def post(self):
        # 解析参数
        args = self.parser.parse_args()

        # 若是增加记录
            # 检查贷款负责人是否存在，不存在则返回对应错误信息

            # 检查账户负责人是否存在，不存在则返回对应错误信息

            # 检查用户id是否存在，若存在则返回对应错误信息

            # 插入记录，返回插入成功的信

        # 若是删除记录：
            # 查找所有满足要求的客户id

            # 若没找到符合要求的客户，返回对应错误信息

            # 若要删除的客户关联了贷款或账户，不允许删除，返回对应错误信息

            # 删除账户，返回删除了几个账户
```

```
        # 若是更改记录：
            # 查找所有满足要求的客户id

            # 若没找到符合要求的客户，返回对应错误信息

            # 若要更改账户负责人或者贷款负责人，检查是否存在，若不存在则返回错误信息

            # 若不修改顾客的id
                    # 直接修改，并返回修改成功的消息

            # 否则
                # 若满足修改条件的顾客有多个，不允许修改，返回错误信息
                # 若要修改的顾客id已经存在，不允许修改，返回错误信息
                # 修改顾客id和所有相关的账户信息和贷款信息，返回成功消息

                # 若是查找记录

        else:
            # 查找所有满足要求的客户id

            # 若没找到符合要求的客户，返回对应错误信息

            # 返回查找到的结果

            # 返回查找到的结果
        return result
```

至于真实的实现，具体可以查看functions.py

3.2 前端

工具链及文件组织

前端使用`Vue.js`开发，为了保证用户的交互体验和内容呈现的美观性，使用了UI框架`Vuetify`，文件的组织如下：

```
front/src
├── App.vue                                 # 应用主组件
├── components                              # 一些子组件
|   ├── AccountManage.vue                       # 账户管理界面
|   ├── BusinessStatistic.vue                   # 数据统计界面
|   ├── CustomerManage.vue                      # 客户界面
|   ├── Homepage.vue                            # 主页界面
|   ├── LoanManage.vue                          # 贷款管理界面
|   ├── PageNotFound.vue                        # 404界面
├── main.js                                 # 主js文件，注册一些用到的模块
└── router.js                               # 定义路由规则
```

完成开发后，可以使用`npm run build`生成js和html文件

网页

在此仅选择介绍一个组件：PageNotFound.vue

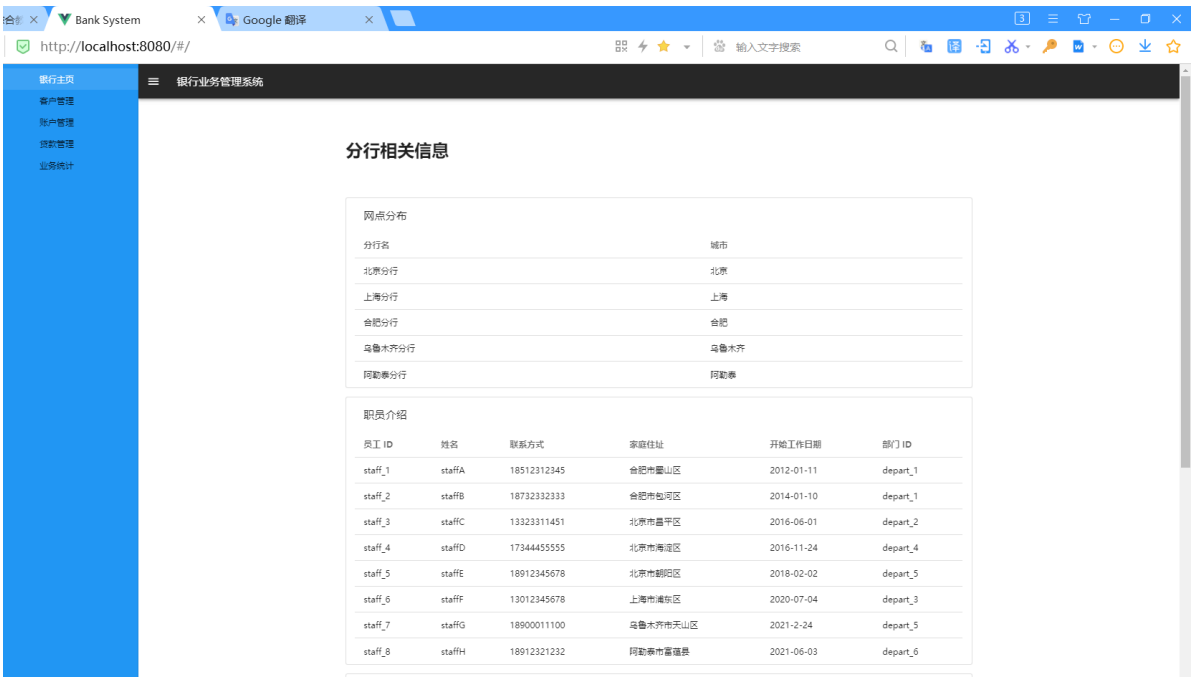
```
<template>
  <v-container class="my-12 py-12">
    <v-row>
      <v-col cols="12" align="center" justify="center">
        <span class="display-4">404</span>
      </v-col>
    </v-row>
    <v-row>
      <v-col align="center" justify="center">
        <span class="display-4">Page Not Found!</span>
      </v-col>
    </v-row>

  </v-container>
</template>

<script>
  export default {
    name: 'PageNotFound',

    data: () => ({

    }),
  }
</script>
```

如果想要了解更多的前端代码，可以查看src文件夹。

4 实现与测试

4.1实现结果

银行主页，显示所有的内置数据：



部门介绍

部门 ID	部门名	类型	部门经理	所属分行
depart_1	市场部	普通	staff_1	Hefei Bank
depart_2	市场部	普通	staff_3	Beijing Bank
depart_3	市场部	特殊	staff_6	Shanghai Bank
depart_4	人事部	普通	staff_4	Beijing Bank
depart_5	人事部	普通	staff_7	Urumqi Bank
depart_6	人事部	特殊	staff_8	Altay Bank

部门经理介绍

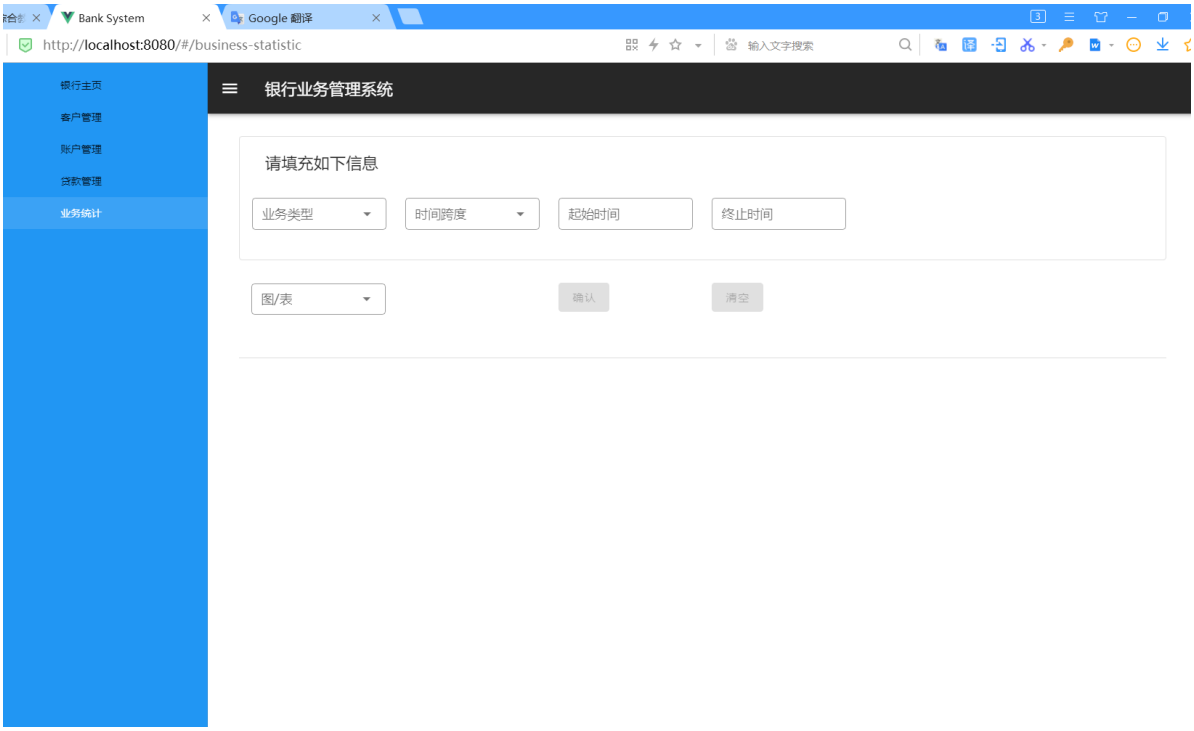
员工 ID	部门 ID
staff_1	depart_1
staff_3	depart_2
staff_6	depart_3
staff_4	depart_4
staff_7	depart_5
staff_8	depart_6

客户管理：

账户管理：

贷款管理：

业务统计：



4.2测试结果

由于实验检查时已经现场进行了多次测试与验证，在这里仅展示部分关键点的测试：

1.尝试输入不合法的客户ID/电话号码：对话框变红提示不合法

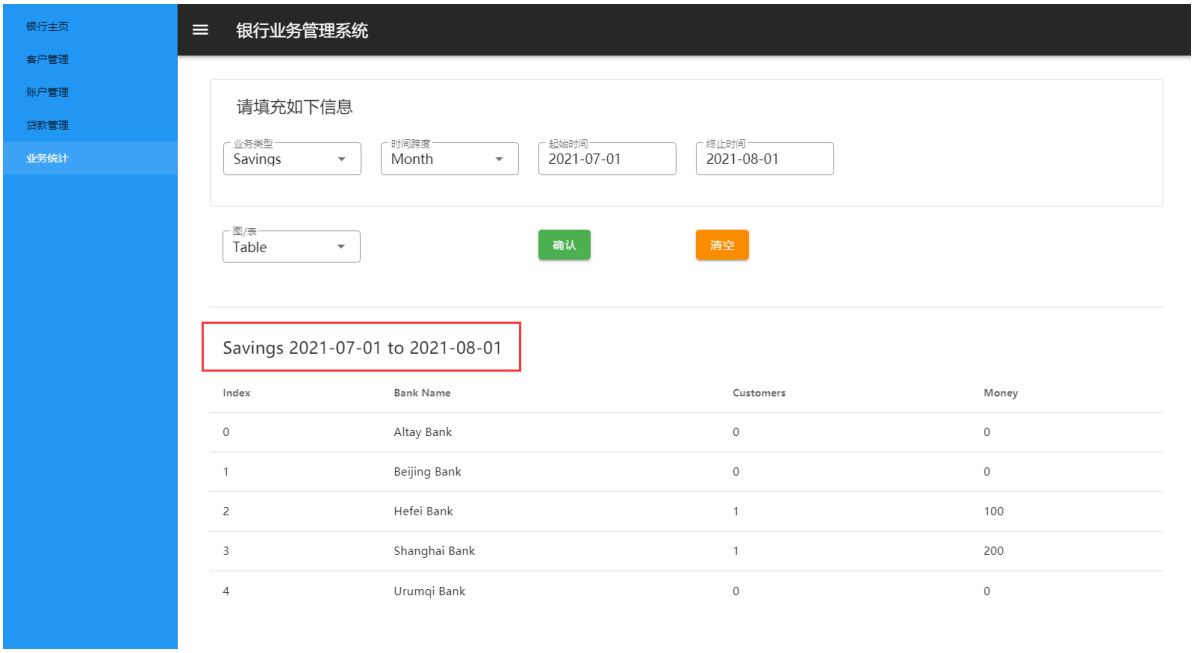
2.尝试删除关联账户或者贷款的客户：被禁止

3.企图修改账户ID，被禁止。

4.企图删除未完成支付的贷款信息，被禁止。

5.妥善处理贷款发放超过剩余所需发放金额的情况，提示不合法信息：

6.业务统计展示：



其他常规测试结果也均正确。

更多的测试可配置好实验环境后拿本人的源码进行，配置参考要求在上文前言部分已叙述。

4.3 实验中的难点问题及解决

1.级联删除

关于不同表之间的联系，会有外键关联的问题，如果需求中存在级联删除请求这时在相应表的外键设置时要注意添加`On Delete Cascade`，有时会存在一个父表关联多个子表的情况，要注意分析，将相应位置都加上`On Delete Cascade`

举例

```
create table loan_payment (
    lp_Serialnumber INT ,
    lp_l_code       CHAR(20) ,
    lp_date         DATE    not null,
    lp_money        FLOAT   not null,
    constraint PK_PAYMENT primary key (lp_Serialnumber, lp_l_code),
    constraint FK_LPLCODE Foreign Key(lp_l_code) References loan(l_code) On
Delete Cascade    /*loan_payment表中lp_l_code作为外键引用loan表中的l_code*/
);

create table loan_customer(
    lc_l_code  CHAR(20),
    lc_c_identity_code    CHAR(20),
    constraint PK_LCL primary key (lc_l_code, lc_c_identity_code),
    constraint FK_LCLLC foreign key (lc_l_code) references loan(l_code) On
Delete Cascade, /*loan_customer表中lc_l_code作为外键引用loan表中的l_code*/
    constraint FK_LCLCLIC foreign key (lc_c_identity_code) references
customer(c_identity_code)
);
```

2.前后端不一致，前端给用户显示400/500错误。

在实际写实验的过程处理了若干个bugs都和这两种错误有关，其大体上都是由于后端和前端在接口交时互不相认造成的。

个人的经验表明：

应在具体构建编码前，将命名统一化，标准化；关注点尽量分离，以“高内聚，低耦合”的原则进行。尤其注意前后端交互部分的变量声明以及一些定义，一定要保证其一致性。

另外前端显示的报错信息过于抽象，可以参考vscode的终端（其他终端应该也行）返回的信息，会更加具体，便于理解问题所在，有利于进一步解决。

5 总结与讨论

本实验实现了一个前后端分离的BS架构的银行管理系统。

通过本次实验熟练掌握了`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。

本实验工程量很大且为独立实验，建议尽早进行。