银行业务管理系统

系统设计与实现报告

姓名：钟溯飏；学号：PB18111764

精神收获
主要教训

1.概述

系统目标

开发一个C/S架构的银行业务管理系统。数据处理由Python完成，数据库交互使用sqlalchemy，页面交互由Pyqt实现，UI设计使用Qt creator。

需求说明

数据需求

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。

功能需求

- 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，统计的结果以表格形式展示。

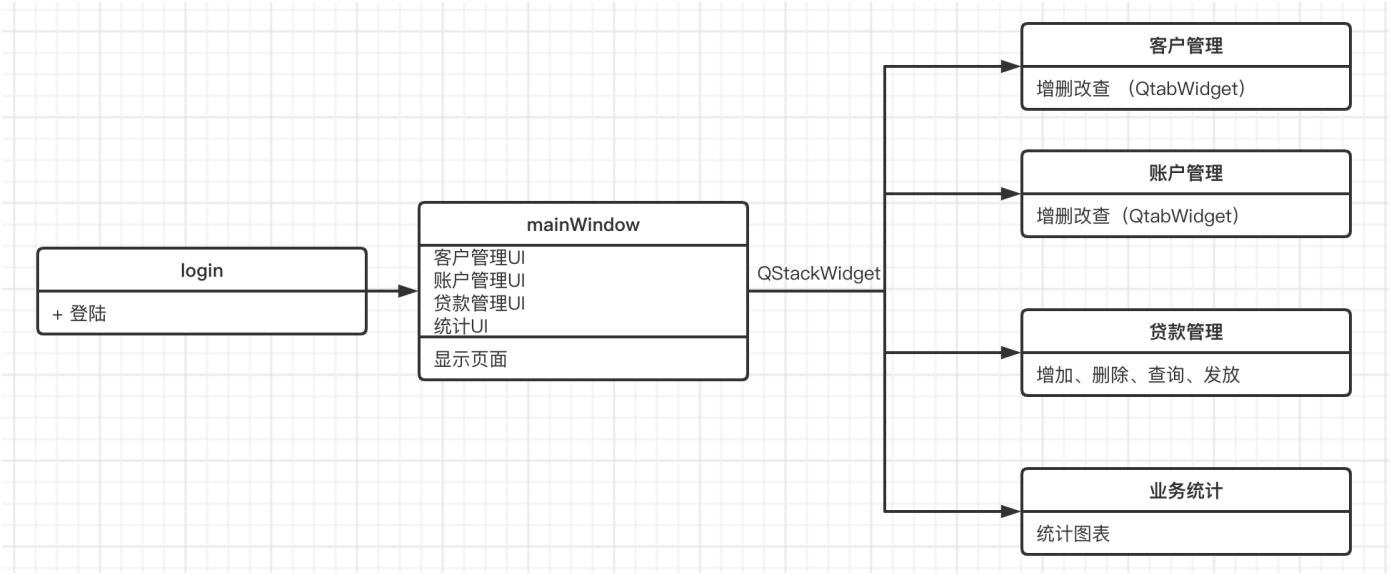
主要贡献

- 总体概括系统的设计结构
- 详细介绍系统的设计方法和程序流程
- 对系统设计进行正确性测试和分析

2.总体设计

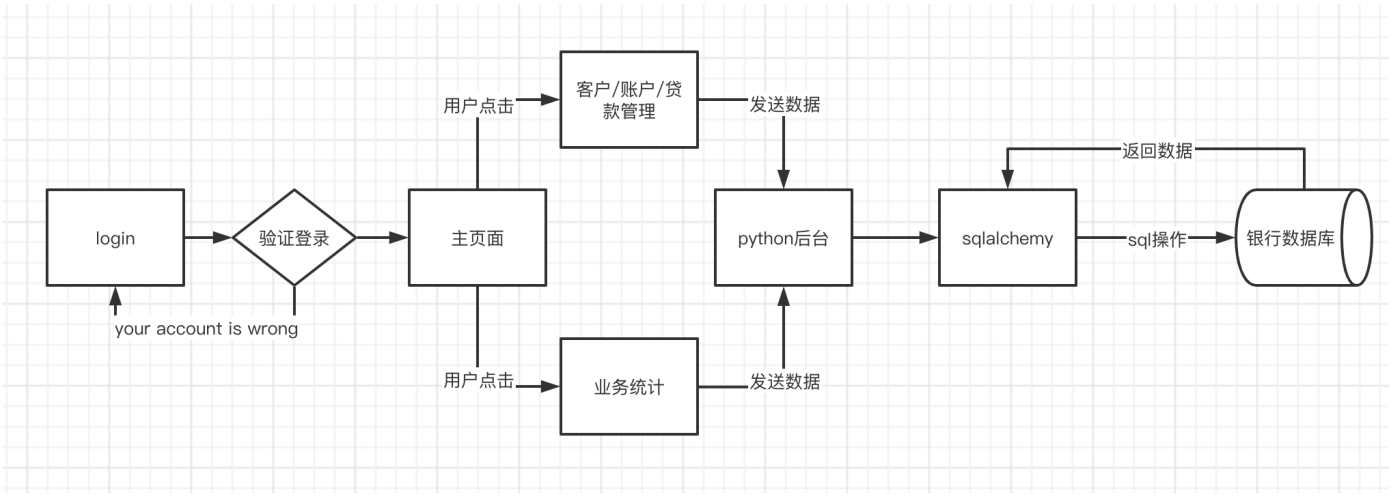
系统模块结构

使用Pyqt实现如下功能



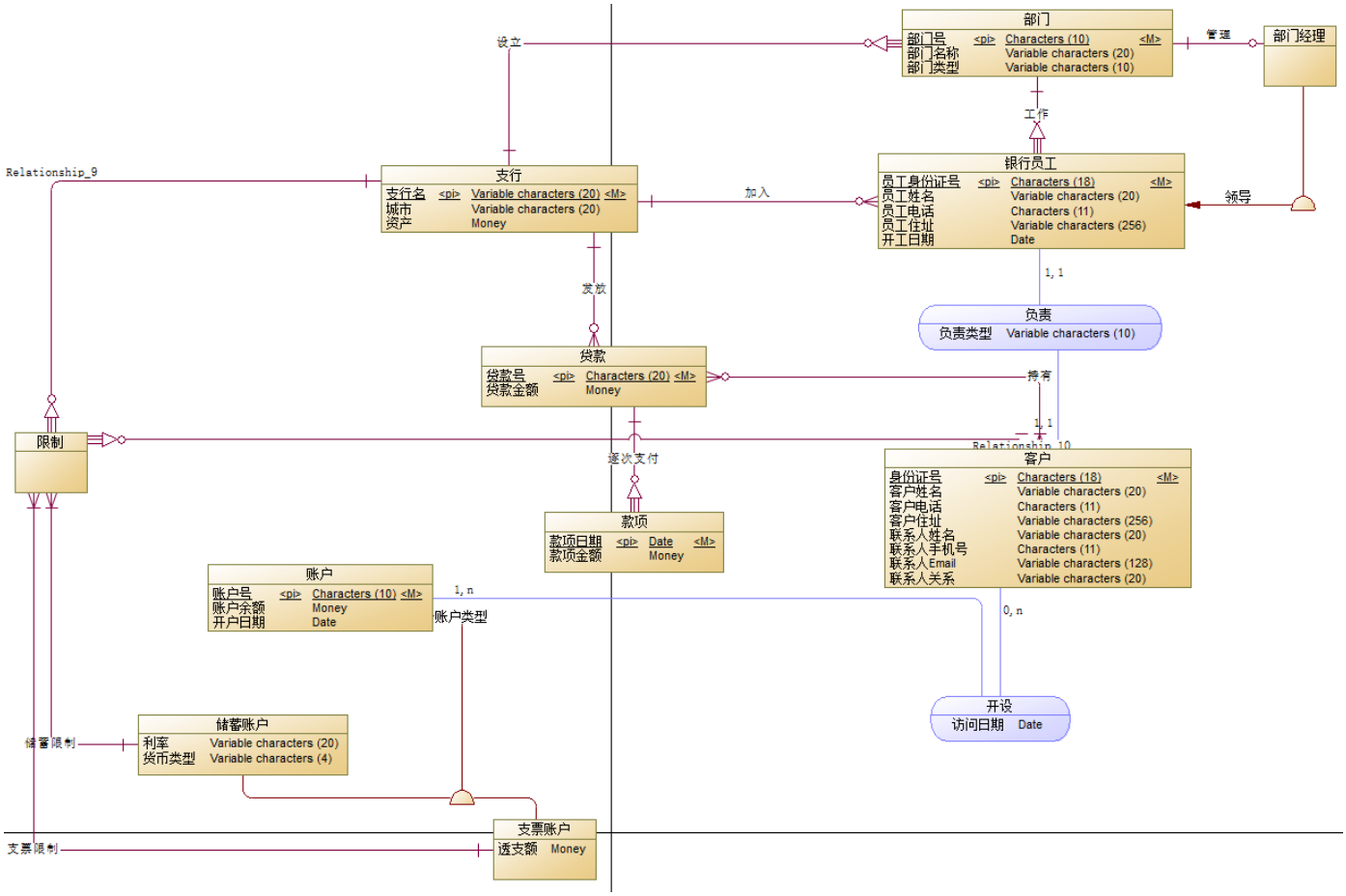
- 登陆界面实现登陆，输入账户密码后跳转到mainWindow
- mainwindow界面由stackedWidget组成，使用menu菜单栏进行跳转
    - 客户管理由四个不同功能，用tabWidget隔开
    - 账户管理同上
    - 贷款管理实现增删查和发放
    - 业务统计实现时间上对银行，用户数和金额的统计
- 数据库使用Mysql和sqlalchemy的ORM实现

系统工作流程

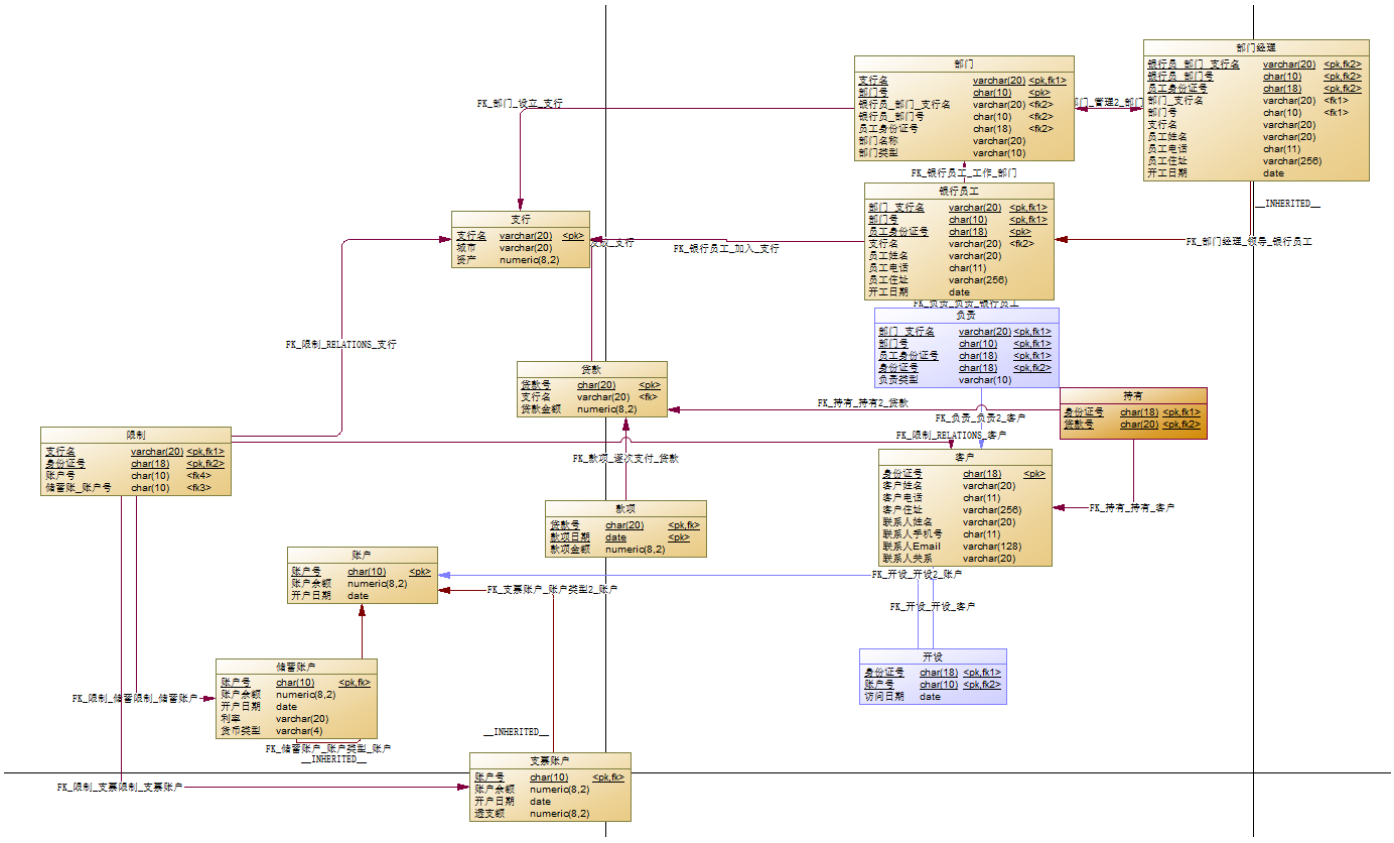


数据库设计

E-R模型图


部门
部门号 <pi> Characters (10) <M>
部门名称 Variable characters (20)
部门类型 Variable characters (10)
设立
管理
部门经理
工作
银行员工
员工身份证号 <pi> Characters (18) <M>
员工姓名 Variable characters (20)
员工电话 Characters (11)
员工住址 Variable characters (256)
开工日期 Date
领导
Relationship_9
支行
支行名 <pi> Variable characters (20) <M>
城市 Variable characters (20)
资产 Money
加入
发放
1, 1
负责
负责类型 Variable characters (10)
贷款
贷款号 <pi> Characters (20) <M>
贷款金额 Money
持有
限制
逐次支付
Relationship_10
客户
身份证号 <pi> Characters (18) <M>
客户姓名 Variable characters (20)
客户电话 Characters (11)
客户住址 Variable characters (256)
联系人姓名 Variable characters (20)
联系人手机号 Characters (11)
联系人Email Variable characters (128)
联系人关系 Variable characters (20)
款项
款项日期 <pi> Date <M>
款项金额 Money
账户
账户号 <pi> Characters (10) <M>
账户余额 Money
开户日期 Date
1, n
账户类型
0, n
开设
访问日期 Date
储蓄账户
利率 Variable characters (20)
货币类型 Variable characters (4)
储蓄限制
支票账户
透支额 Money
支票限制

物理数据库


部门
支行名 varchar(20) <pk,fk1>
部门号 char(10) <pk>
银行员_部门_支行名 varchar(20) <fk2>
银行员_部门号 char(10) <fk2>
员工身份证号 char(18) <fk2>
部门名称 varchar(20)
部门类型 varchar(10)
部门经理
银行员_部门_支行名 varchar(20) <pk,fk2>
银行员_部门号 char(10) <pk,fk2>
员工身份证号 char(18) <pk,fk2>
部门_支行名 varchar(20) <fk1>
部门号 char(10) <fk1>
支行名 varchar(20)
员工姓名 varchar(20)
员工电话 char(11)
员工住址 varchar(256)
开工日期 date
FK_部门_设立_支行
FK_银行员工_工作_部门
__INHERITED__
FK_部门经理_领导_银行员工
支行
支行名 varchar(20) <pk>
城市 varchar(20)
资产 numeric(8,2)
FK_限制_RELATIONS_支行
FK_银行员工_加入_支行
银行员工
部门_支行名 varchar(20) <pk,fk1>
部门号 char(10) <pk,fk1>
员工身份证号 char(18) <pk>
支行名 varchar(20) <fk2>
员工姓名 varchar(20)
员工电话 char(11)
员工住址 varchar(256)
开工日期 date
FK_负责_负责2_银行员工
负责
部门_支行名 varchar(20) <pk,fk1>
部门号 char(10) <pk,fk1>
员工身份证号 char(18) <pk,fk1>
身份证号 char(18) <pk,fk2>
负责类型 varchar(10)
贷款
贷款号 char(20) <pk>
支行名 varchar(20) <fk>
贷款金额 numeric(8,2)
FK_持有_持有2_贷款
FK_负责_负责2_客户
持有
身份证号 char(18) <pk,fk1>
贷款号 char(20) <pk,fk2>
限制
支行名 varchar(20) <pk,fk1>
身份证号 char(18) <pk,fk2>
账户号 char(10) <fk4>
储蓄账_账户号 char(10) <fk3>
FK_限制_RELATIONS_客户
FK_款项_逐次支付_贷款
客户
身份证号 char(18) <pk>
客户姓名 varchar(20)
客户电话 char(11)
客户住址 varchar(256)
联系人姓名 varchar(20)
联系人手机号 char(11)
联系人Email varchar(128)
联系人关系 varchar(20)
FK_持有_持有_客户
款项
贷款号 char(20) <pk,fk>
款项日期 date <pk>
款项金额 numeric(8,2)
账户
账户号 char(10) <pk>
账户余额 numeric(8,2)
开户日期 date
FK_开设_开设2_账户
FK_支票账户_账户类型2_账户
FK_开设_开设_客户
开设
身份证号 char(18) <pk,fk1>
账户号 char(10) <pk,fk2>
访问日期 date
储蓄账户
账户号 char(10) <pk,fk>
账户余额 numeric(8,2)
开户日期 date
利率 varchar(20)
货币类型 varchar(4)
FK_限制_储蓄限制_储蓄账户
FK_储蓄账户_账户类型_账户
__INHERITED__
__INHERITED__
支票账户
账户号 char(10) <pk,fk>
账户余额 numeric(8,2)
开户日期 date
透支额 numeric(8,2)
FK_限制_支票限制_支票账户

sqlalchemy形式

```
class Department(baseClass):
    __tablename__ = 'department'
    id = sql.Column(sql.String(25), primary_key=True)
    name = sql.Column(sql.String(50))
    manager_id = sql.Column(sql.String(25))
    type = sql.Column(sql.String(20))

    bank_name = sql.Column(sql.String(50), sql.ForeignKey("bank.name"))


class Employee(baseClass):
    __tablename__ = 'employee'
    id = sql.Column(sql.String(25), primary_key=True)
    name = sql.Column(sql.String(50))
    phone = sql.Column(sql.String(12))
    address = sql.Column(sql.String(50))
    begin_date = sql.Column(sql.Date)
    type = sql.Column(sql.String(20))

    bank_name = sql.Column(sql.String(50), sql.ForeignKey('bank.name'))
    depart_id = sql.Column(sql.String(25), sql.ForeignKey('department.id'))
    customers = relationship('Customer')


class SaveAccount(baseClass):
    __tablename__ = 'save_account'
    id = sql.Column(sql.String(25), primary_key=True)
    rest = sql.Column(sql.Float)
    date = sql.Column(sql.Date)
    rate = sql.Column(sql.String(10))
    money_type = sql.Column(sql.String(10))
    bank_name = sql.Column(sql.String(50), sql.ForeignKey("bank.name"),
onupdate="CASCADE")
    payRel = relationship('CustomerToSA', backref='SaveAccount', passive_deletes=True)


class CustomerToCA(baseClass):
    __tablename__ = 'customer_to_ca'
    last_access_date = sql.Column(sql.Date)
    ca_id = sql.Column(sql.String(25), sql.ForeignKey('check_account.id',
ondelete="CASCADE", onupdate="CASCADE"), primary_key=True)
    cus_id = sql.Column(sql.String(25), sql.ForeignKey("customer.id"),
primary_key=True)
    bank_name = sql.Column(sql.String(50), sql.ForeignKey('bank.name'))
    __table_args__ = (UniqueConstraint('cus_id', 'bank_name', name='one_cus_to_ca'),)
```

```
class CustomerToSA(baseClass):
    __tablename__ = 'customer_to_sa'
    last_access_date = sql.Column(sql.Date)
    sa_id = sql.Column(sql.String(25), sql.ForeignKey('save_account.id',
ondelete="CASCADE",onupdate="CASCADE"), primary_key=True)
    cus_id = sql.Column(sql.String(25), sql.ForeignKey("customer.id"),
primary_key=True)
    bank_name = sql.Column(sql.String(50), sql.ForeignKey('bank.name'))
    __table_args__ = (UniqueConstraint('cus_id', 'bank_name', name='one_cus_to_sa'),)


class CheckAccount(baseClass):
    __tablename__ = 'check_account'
    id = sql.Column(sql.String(25), primary_key=True)
    rest = sql.Column(sql.Float)
    date = sql.Column(sql.Date)
    extra = sql.Column(sql.Float)
    bank_name = sql.Column(sql.String(50), sql.ForeignKey("bank.name"))

    payRel = relationship('CustomerToCA', backref='CheckAccount', passive_deletes=True)


class CustomerToLoan(baseClass):
    __tablename__ = 'customer_to_loan'
    cus_id = sql.Column(sql.String(25), sql.ForeignKey("customer.id"),
primary_key=True, onupdate="CASCADE")
    loan_id = sql.Column(sql.String(25), sql.ForeignKey('loan.id', ondelete="CASCADE"),
primary_key=True)


class Loan(baseClass):
    __tablename__ = 'loan'
    id = sql.Column(sql.String(25), primary_key=True)
    total_money = sql.Column(sql.Float)
    status = sql.Column(sql.Integer)

    bank_name = sql.Column(sql.String(50), sql.ForeignKey('bank.name'))
    c2l = relationship('CustomerToLoan', backref='loan', passive_deletes=True)
    payRel = relationship('Pay', backref='pay',passive_deletes=True)


class Pay(baseClass):
    __tablename__ = 'pay'
    pid = sql.Column(sql.Integer, primary_key=True, autoincrement=True)
    date = sql.Column(sql.Date)
    money = sql.Column(sql.Float)

    cus_id = sql.Column(sql.String(50), sql.ForeignKey('customer.id'))
    loan_id = sql.Column(sql.String(25), sql.ForeignKey('loan.id', ondelete="CASCADE")
```

3.详细设计

Feature

实时查询

为了方便用户进行查询、以及删改操作，人性化的加入了实时查询功能：在用户输入主键（客户ID，账户ID等）的时候，会实时进行关键词搜索，并实时在表格中显示。

具体做法是为Widget中的`lineEdit`的`textEdited`信号绑定一个query函数，即

```
self.lineEdit.textEdited.connect(self.queryLoan)
```

函数中使用`tableWidget`的`setItem`功能，进行实时的显示。具体演示可见后

自动填充

同时，为了提升用户体验，在删改的时候，都可以点击右侧表格，程序将会把所在行内容填充到左侧文本框内，用户只需略微修改（或者不修改）即可直接进行更改（删除）操作。

如下为贷款中的操作，将指针所在行内容进行拷贝。

```
def setTextFromTable(self):
    row = self.tableWidget.currentRow()
    id = self.getTableText(row, 0)
    total_money = self.getTableText(row, 1)
    bank_name = self.getTableText(row, 2)
    self.lineEdit.setText(id)
    self.lineEdit_2.setText(total_money)
    self.comboBox.setCurrentText(bank_name)
```

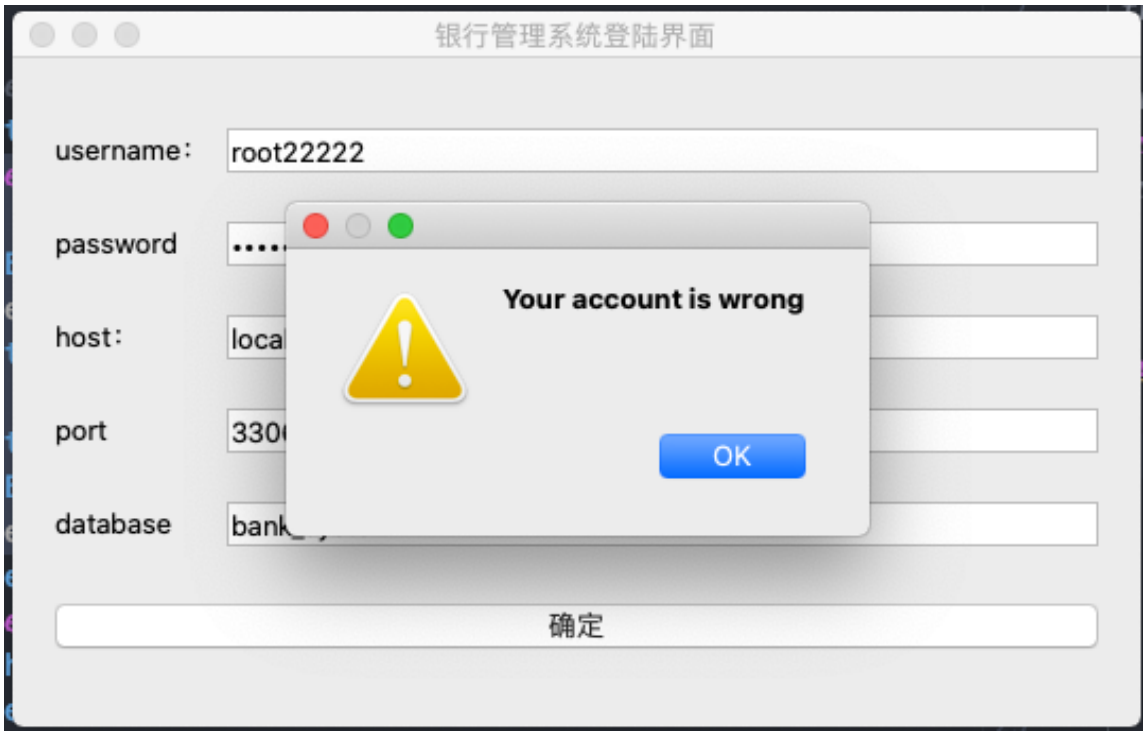
智能屏蔽

考虑到账户具有两种类型，具有不同的属性，因此使用了`setEnable`方法使得在选取储蓄账户时，支票账户的属性不可编辑，这样减少了出错的可能性。

```
def setBoxReadOnly(self):
    self.savingState = self.isSaving.checkState()
    if self.savingState:
        self.AccRate.setEnabled(True)
        self.currencyType.setEnabled(True)
        self.AccExtra.setEnabled(False)
    else:
        self.AccExtra.setEnabled(True)
        self.AccRate.setEnabled(False)
        self.currencyType.setEnabled(False)
```

4.实现与测试

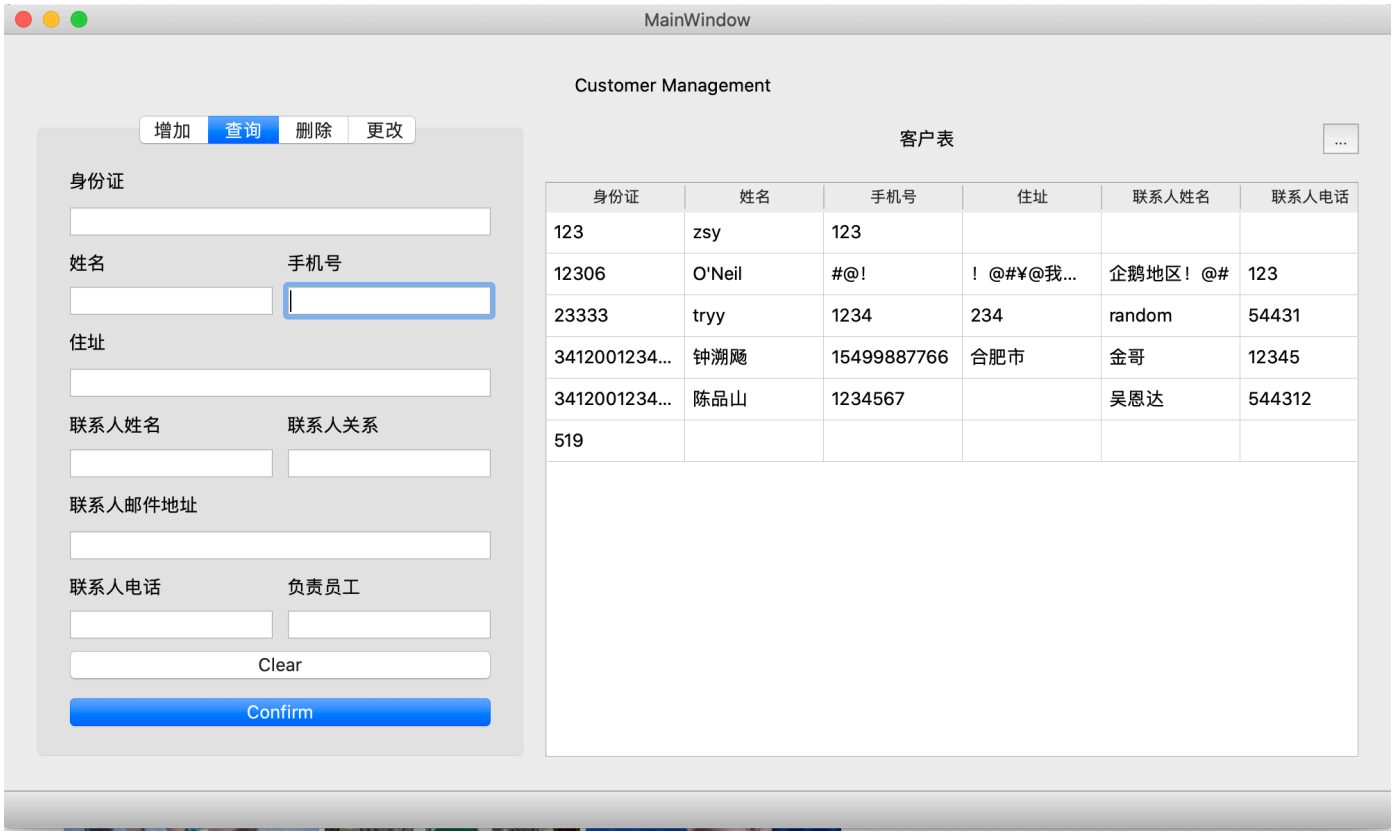
登陆



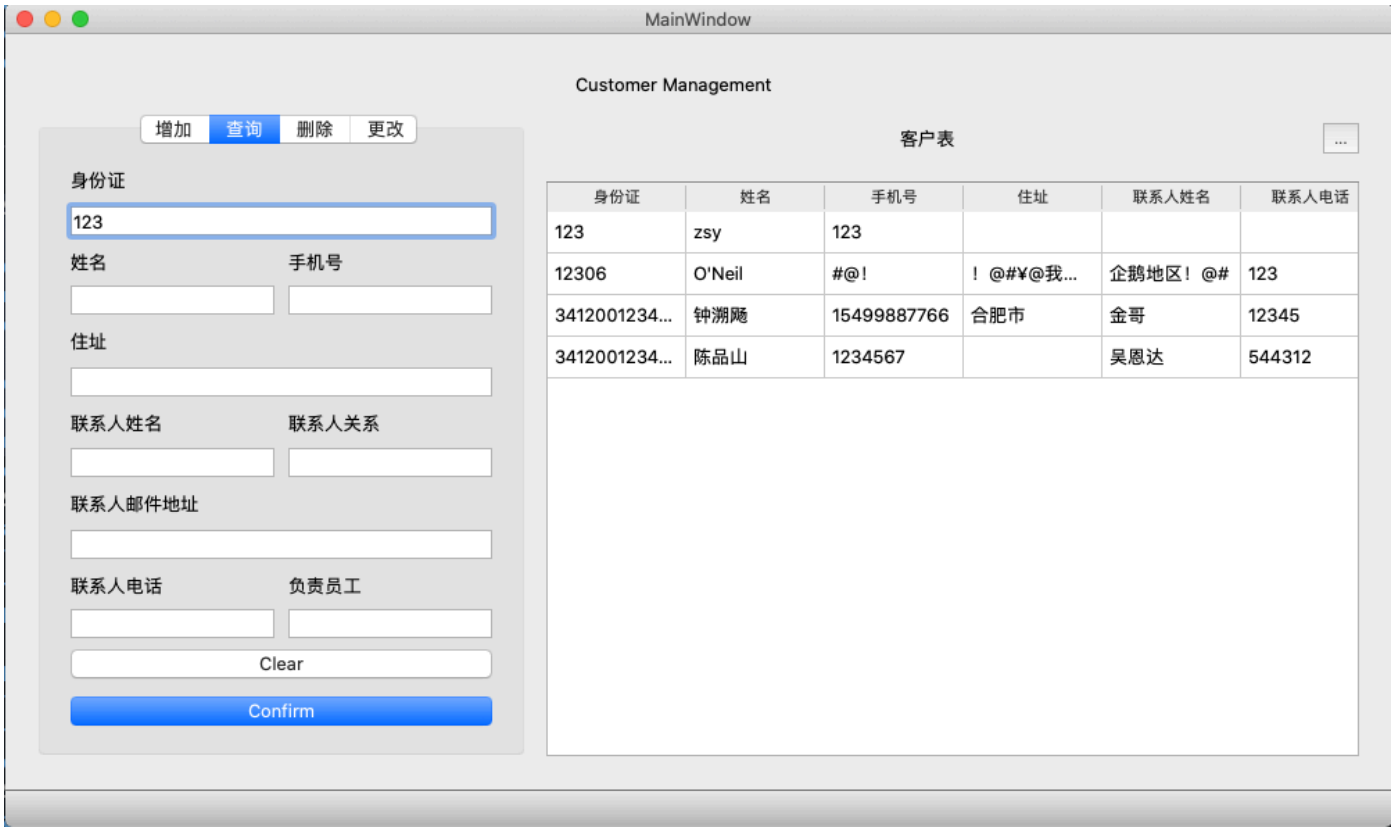
支持错误纠正功能

客户管理页面

查询

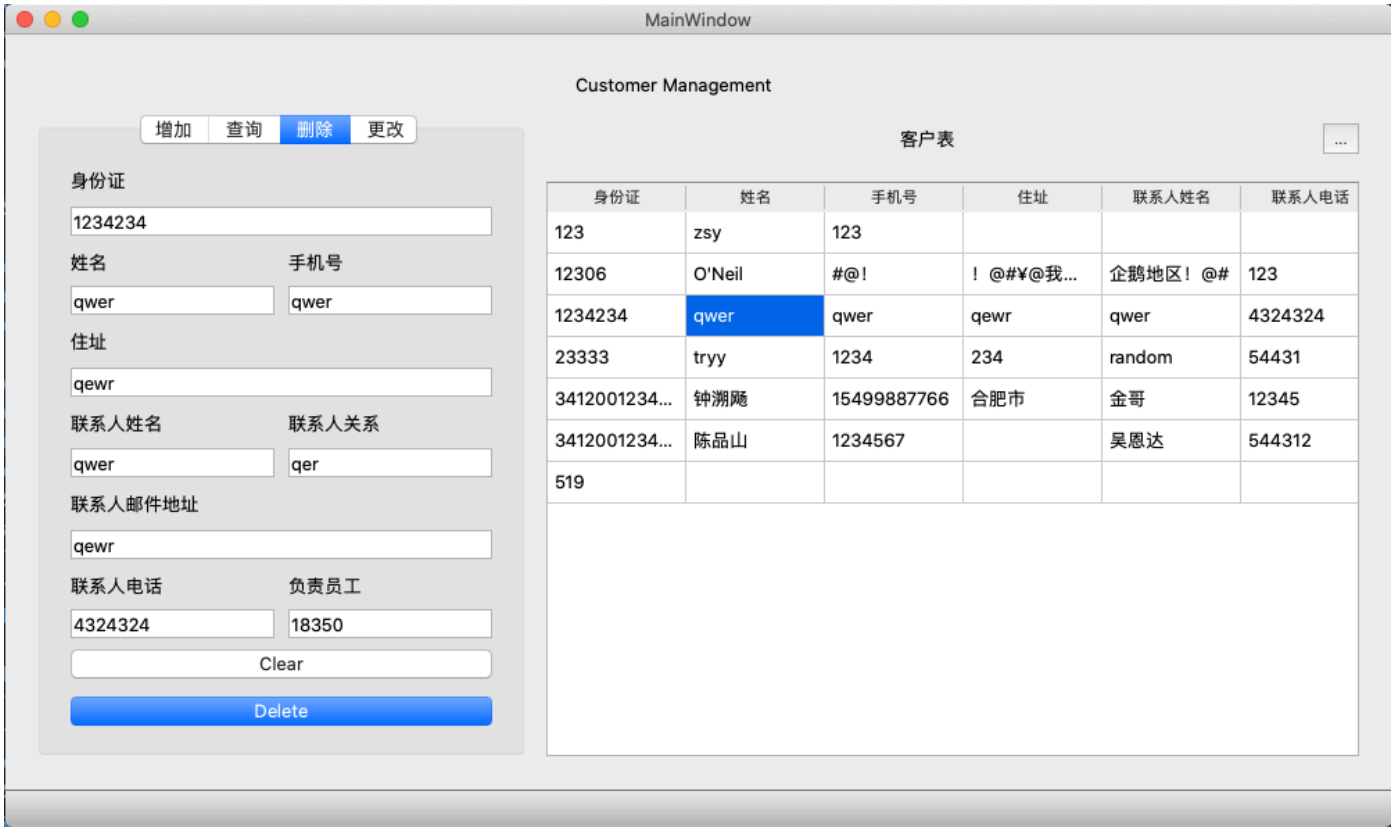


使用空查询或者右上角`toolButton`都可显示全部内容，也支持如下实时关键词搜索



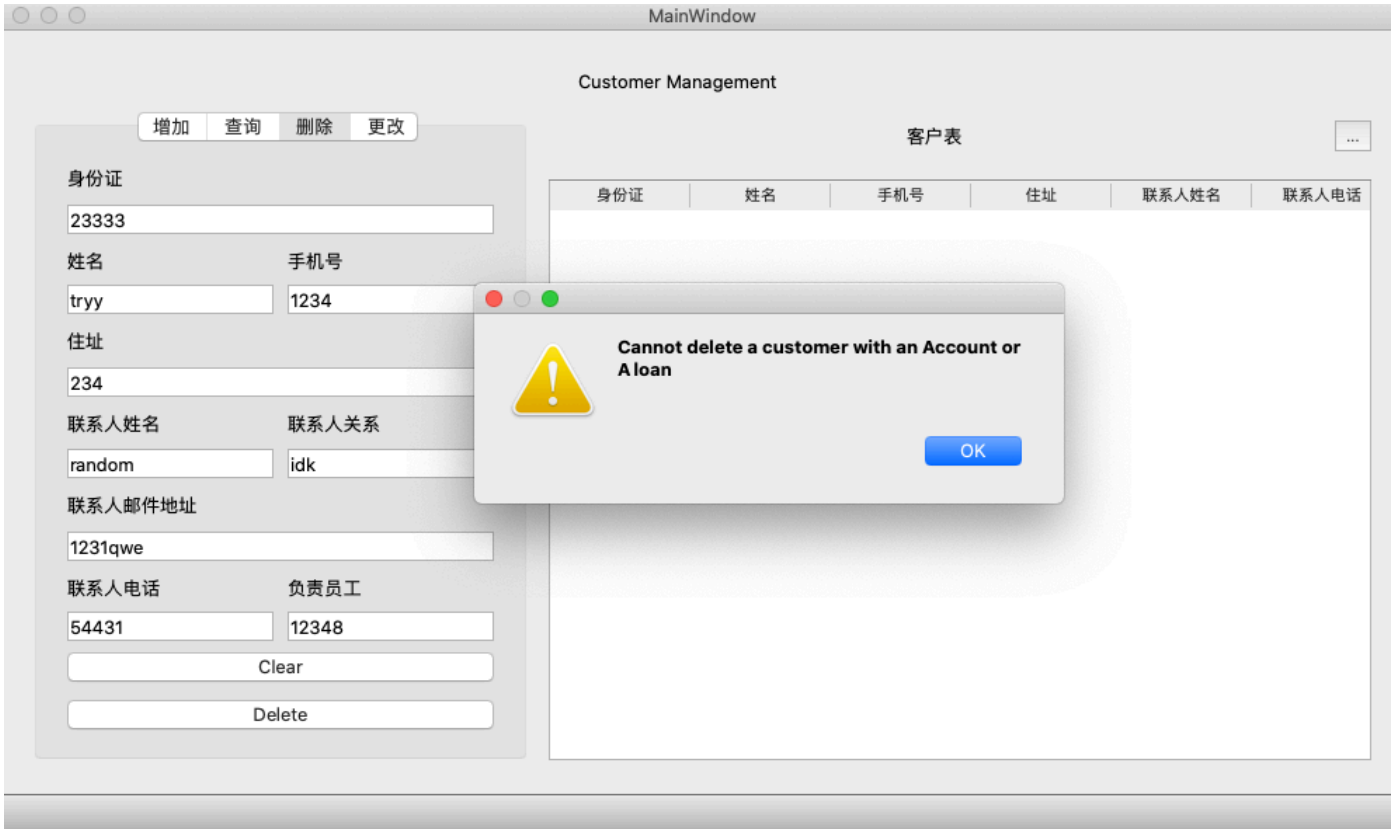
增添

删除/更新



可点击右侧表格内容，将一行内容全部复制到编辑栏内

进行更改后如图所示

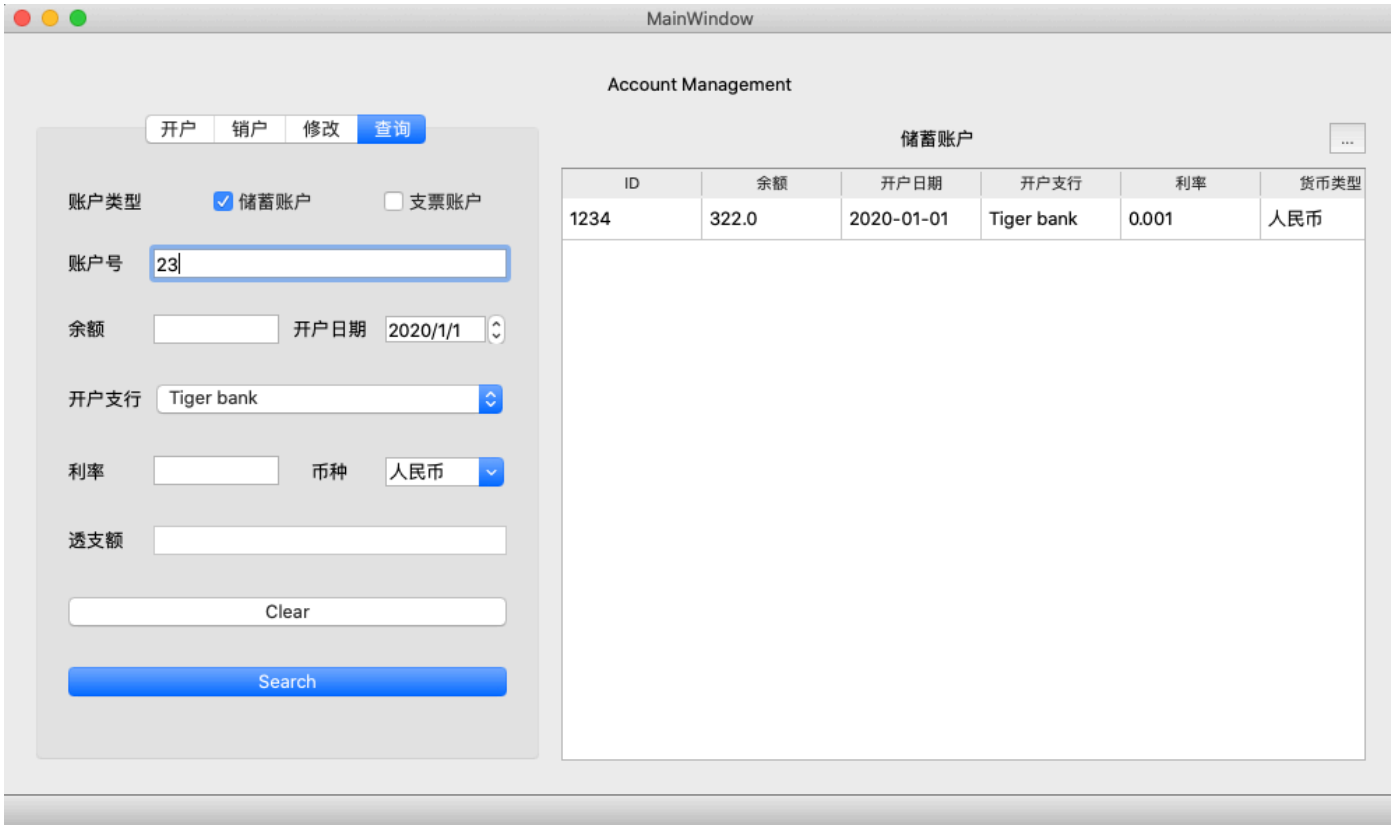


删除一个拥有账户的客户会进行错误提示

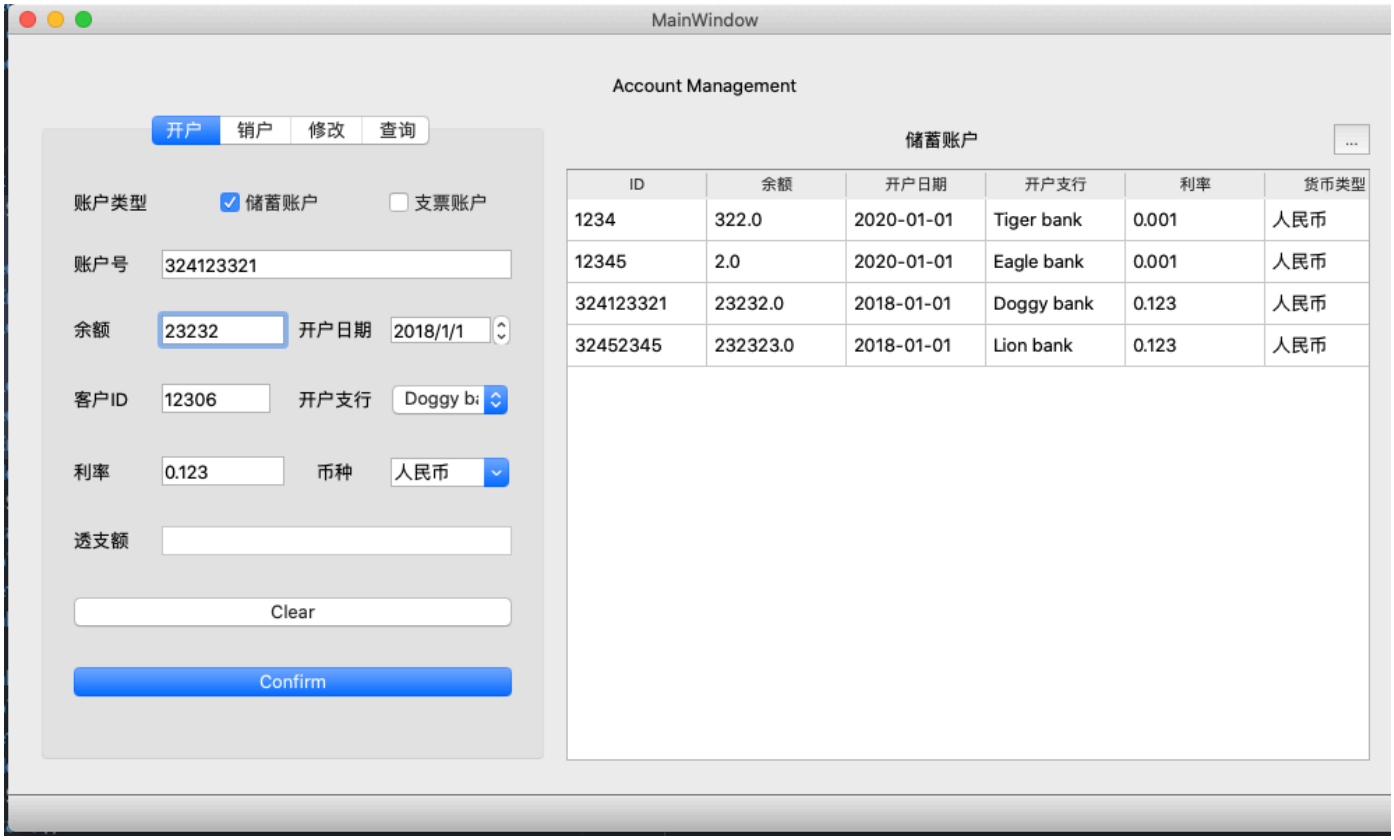
账户管理页面

查询

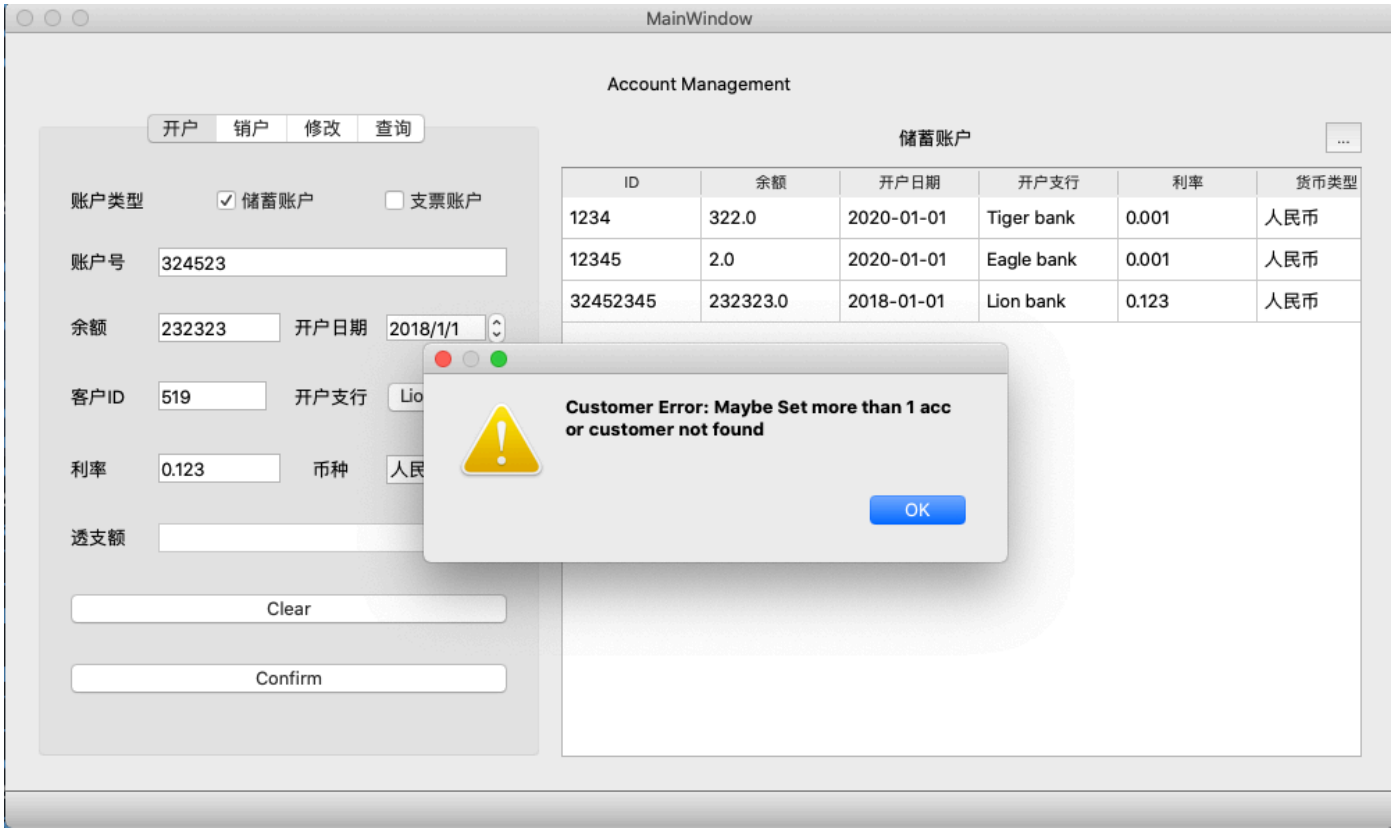
与之前客户管理一样，都支持关键词实时搜索。



开户

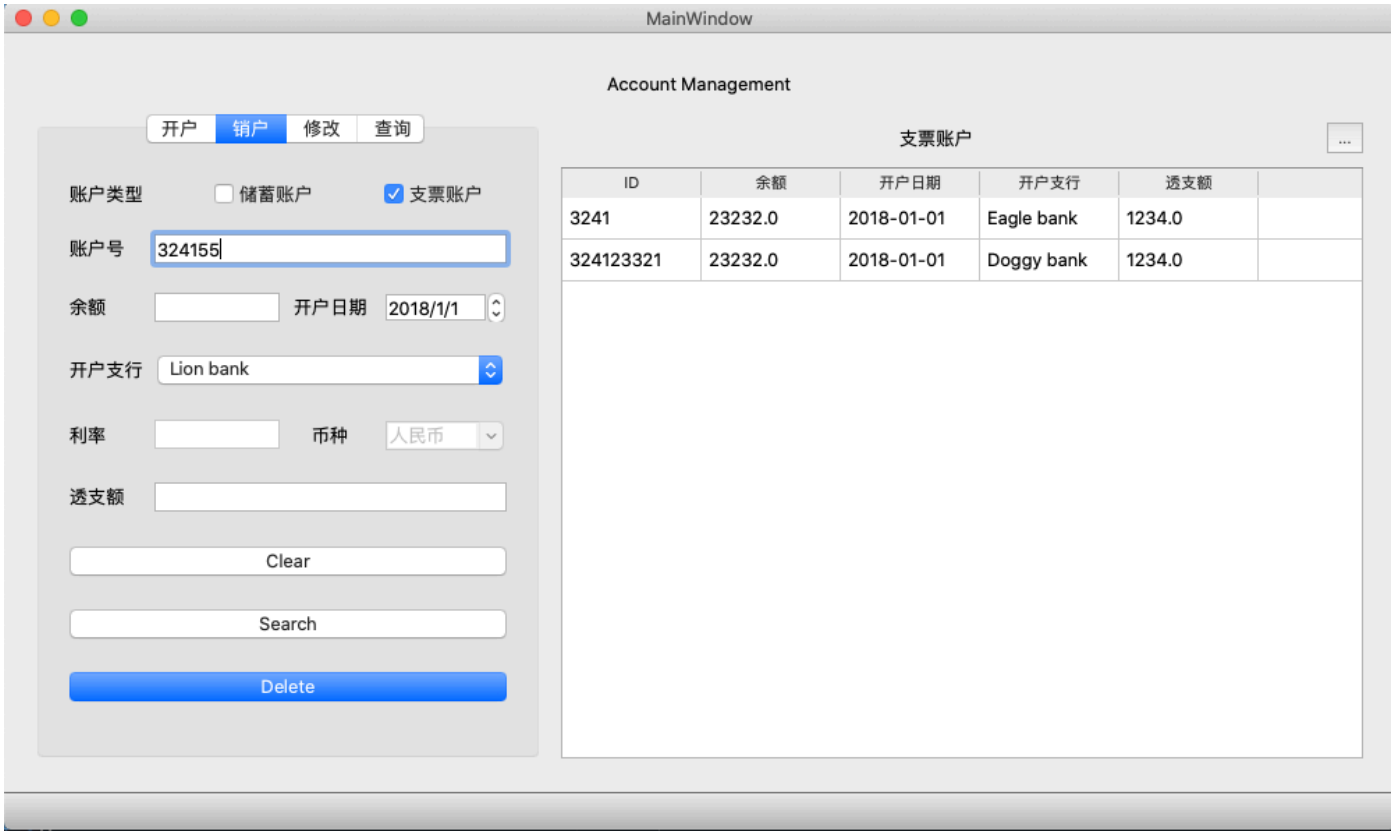


如图所示，为12306客户创建一个324123321账户



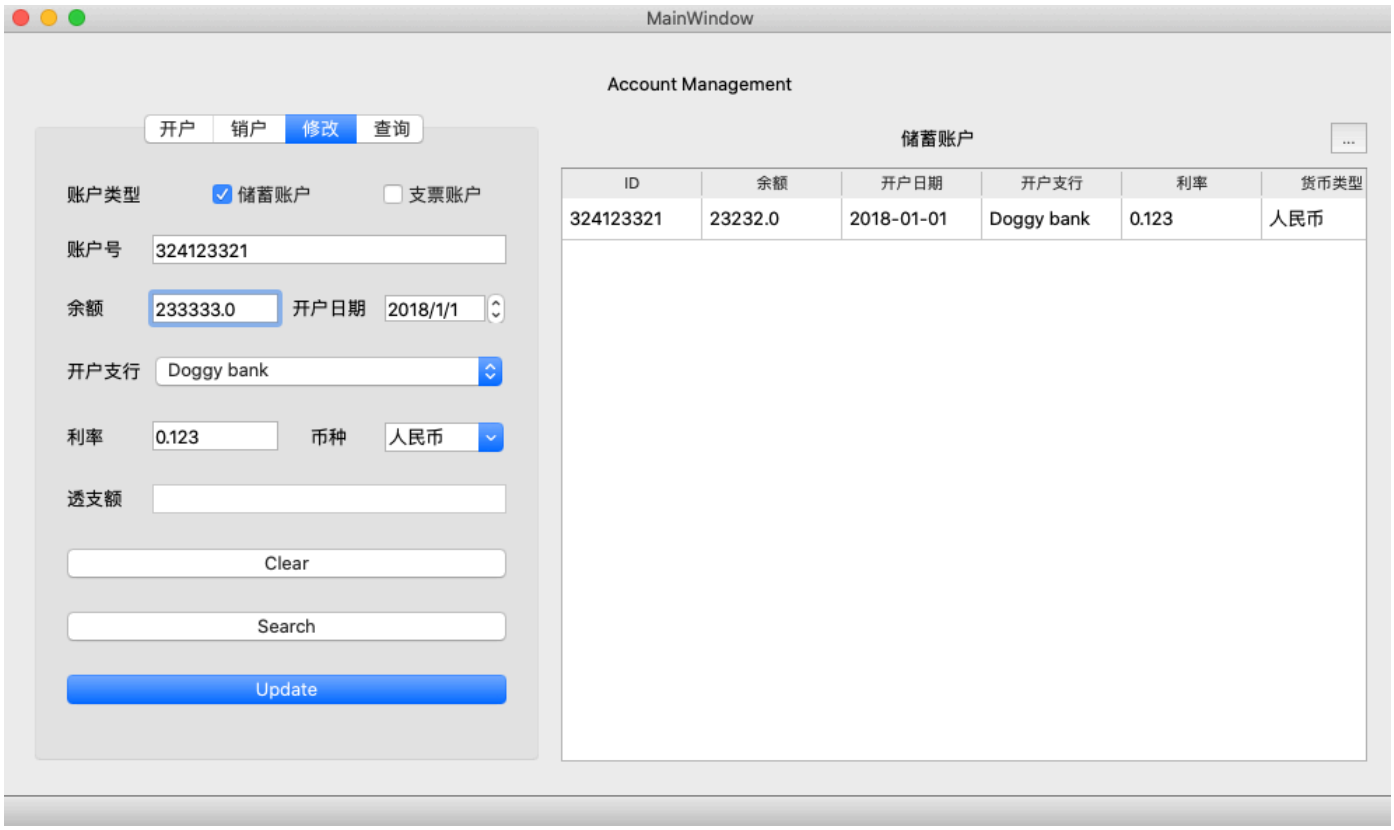
如图错误所示：不能在一家银行为一个人开两个储蓄账户

销户

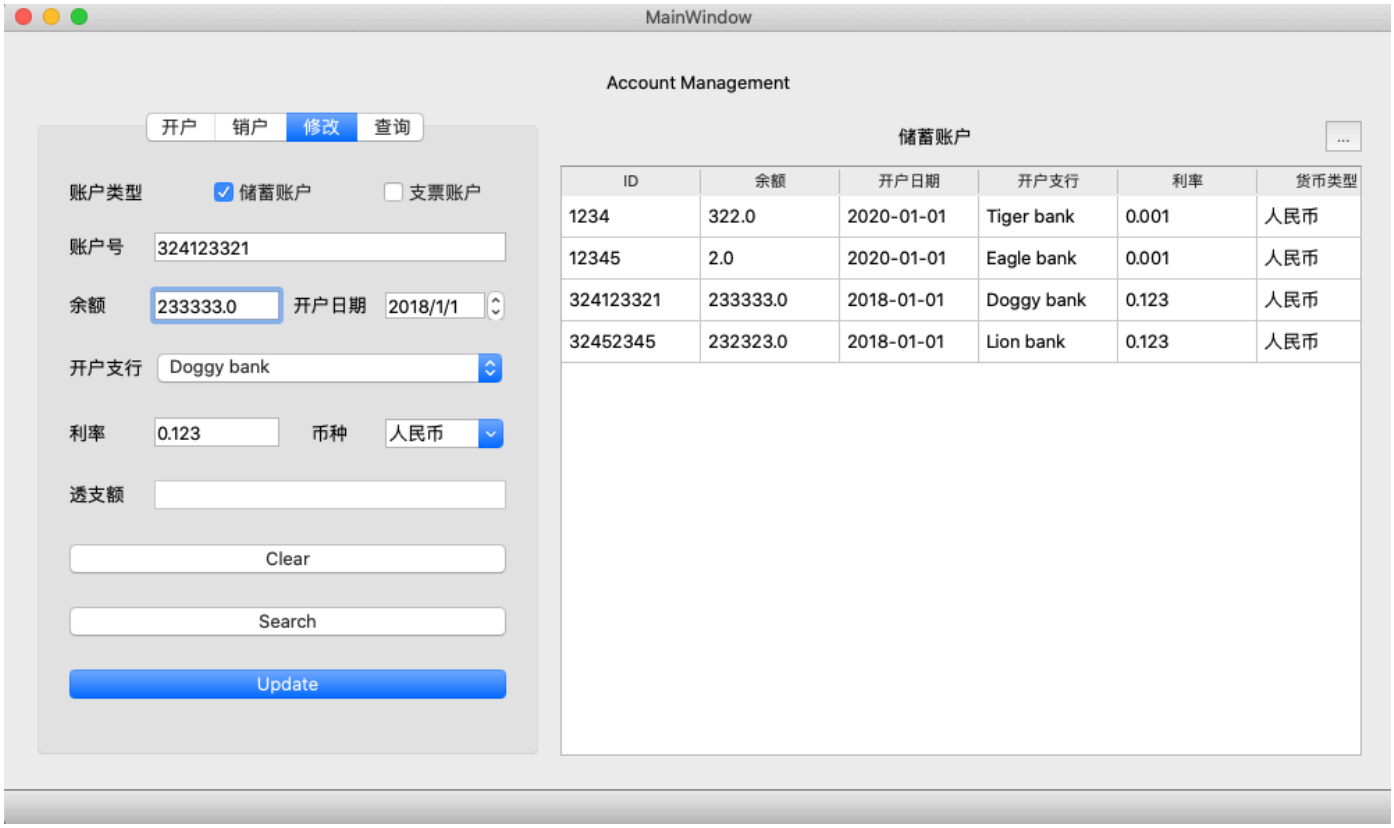


对支票账户进行销户（可见右表中已经无324155这一项）

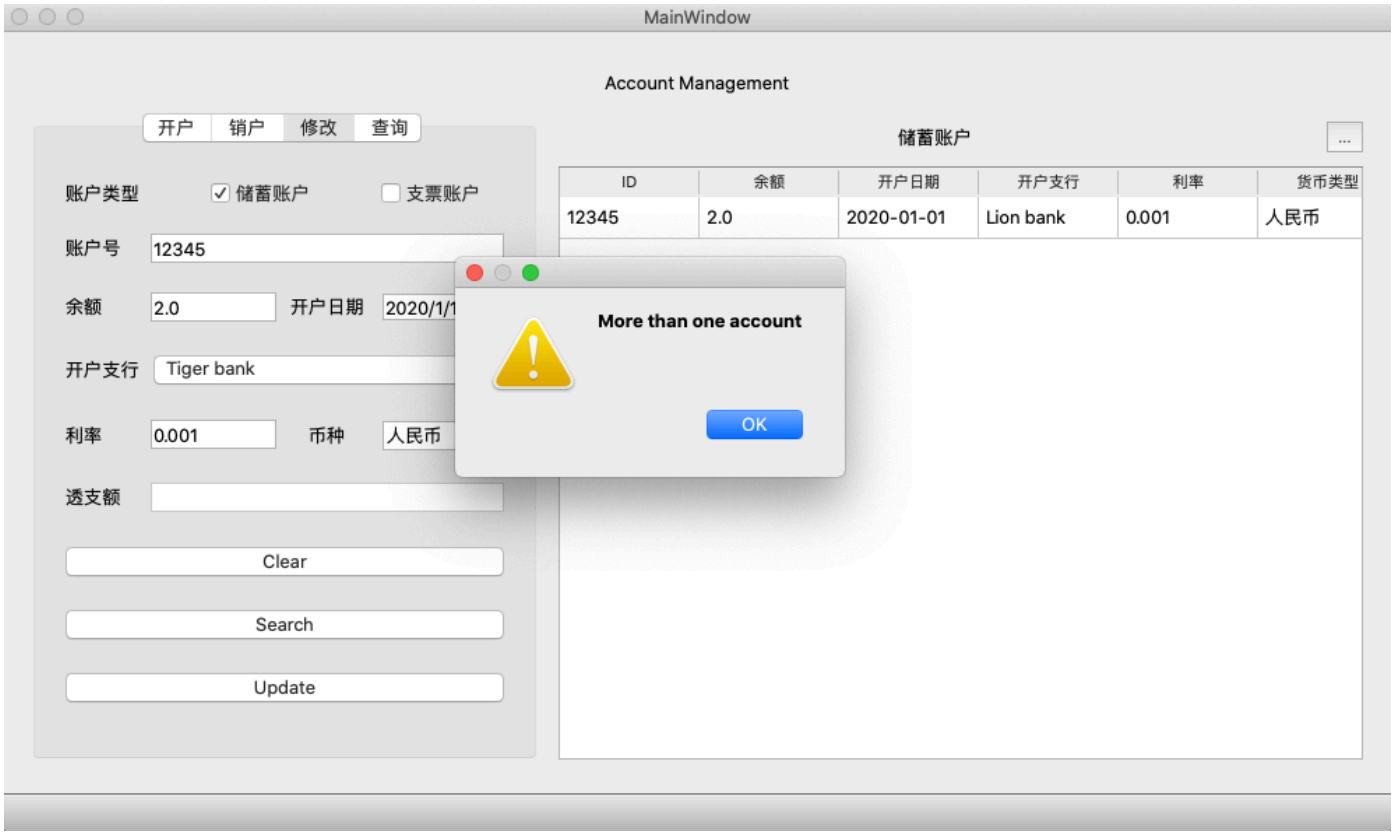
修改



修改前后

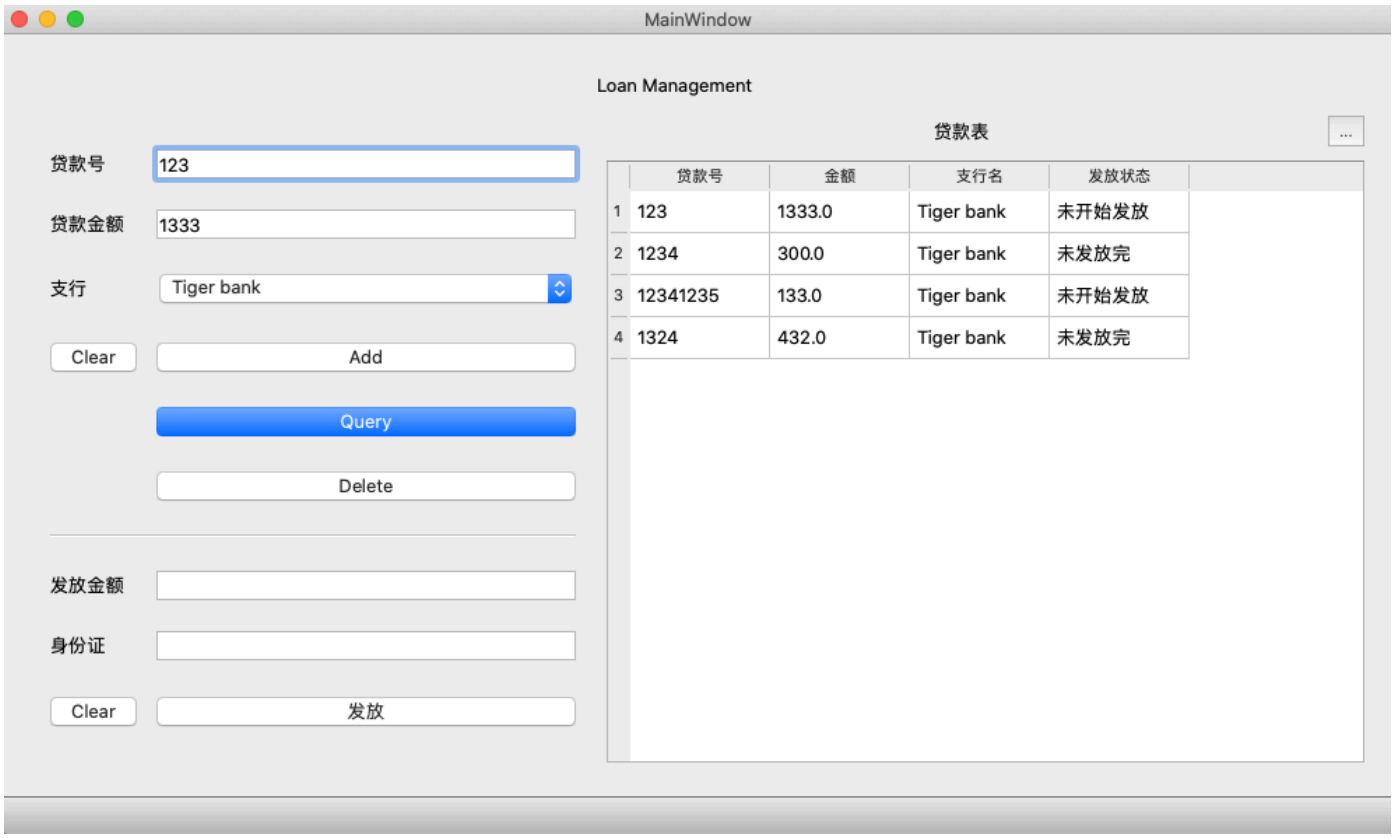


注意，即便是更新，也不能将开户支行更改使得一个人在一个支行下有两个同类账户。



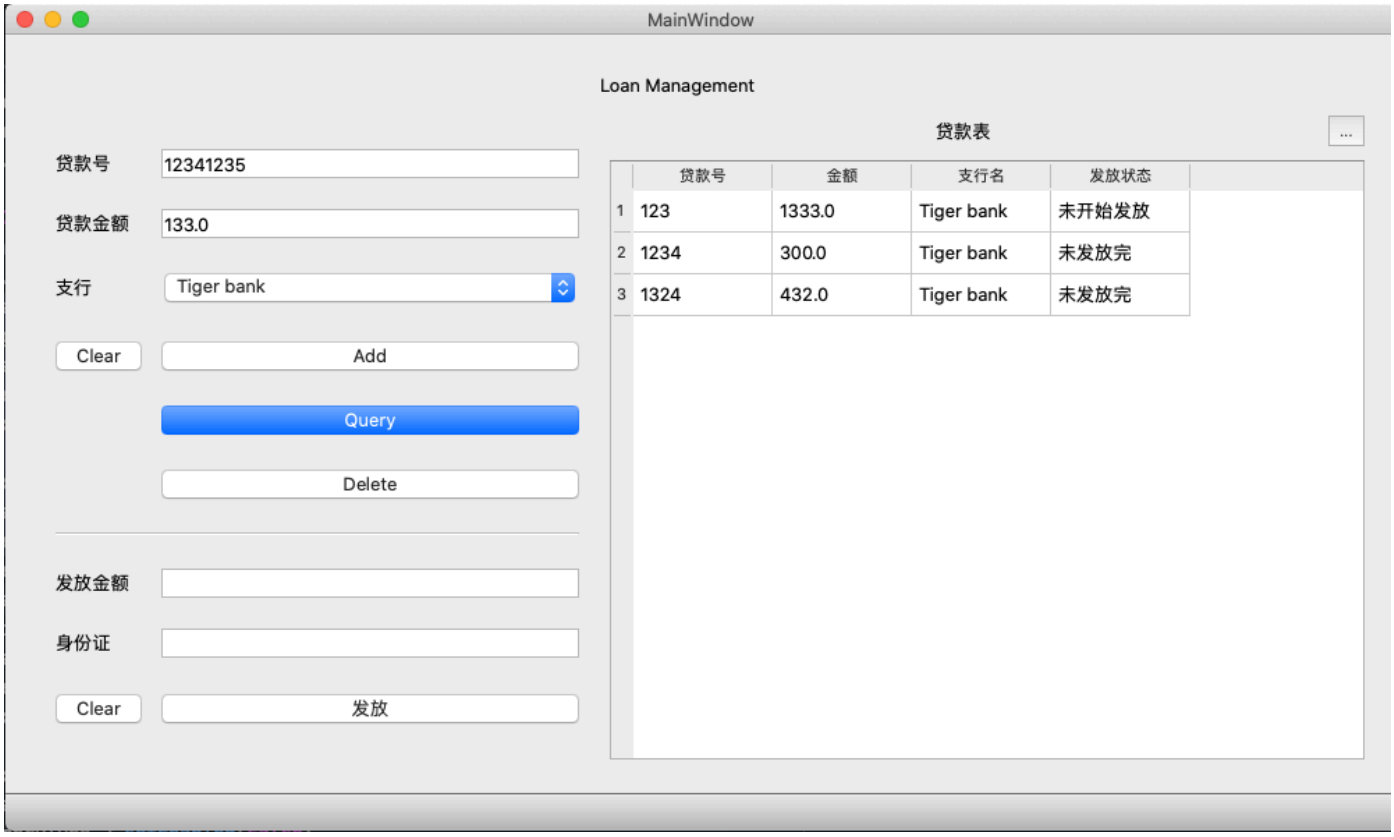
贷款管理

增添

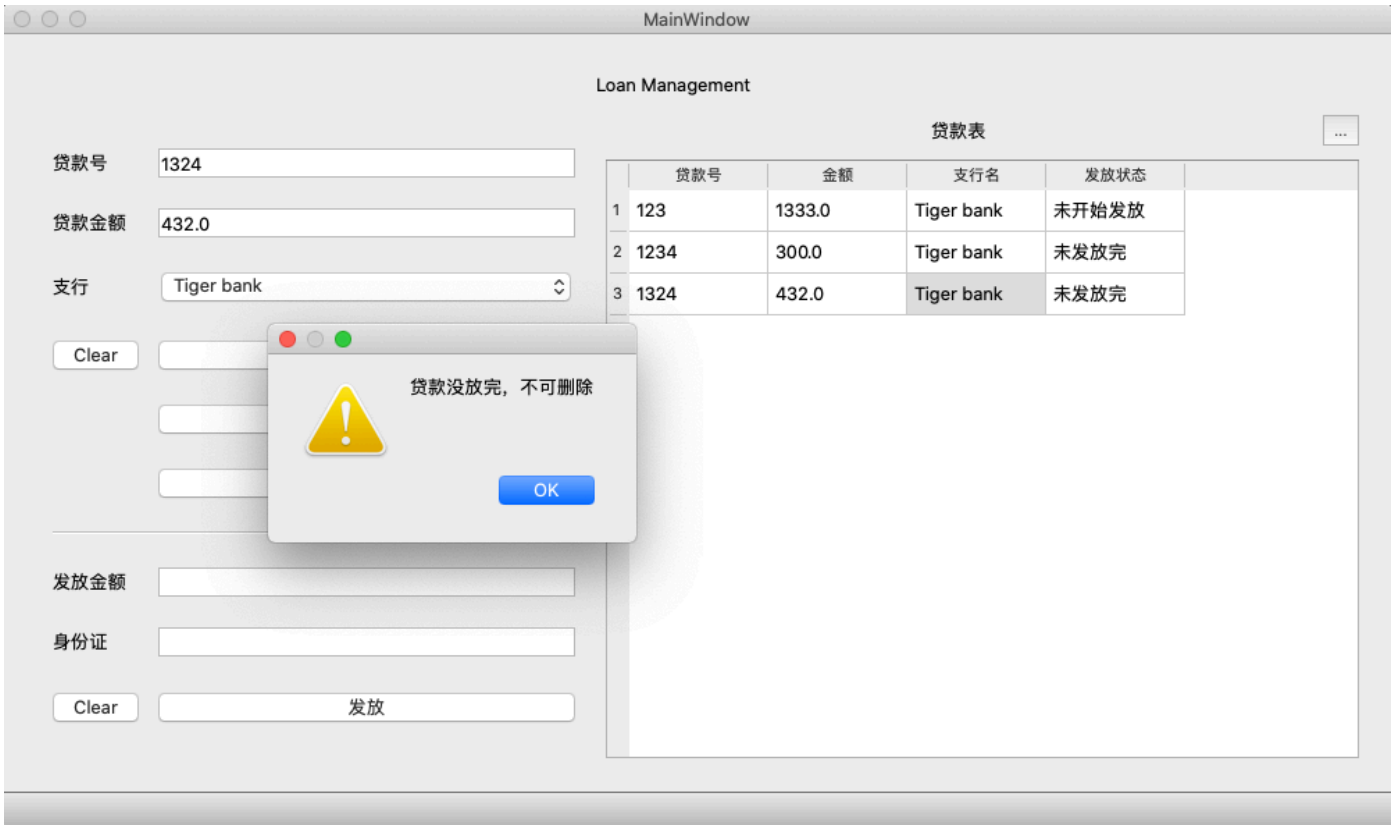


注意默认情况下是未开始发放

删除



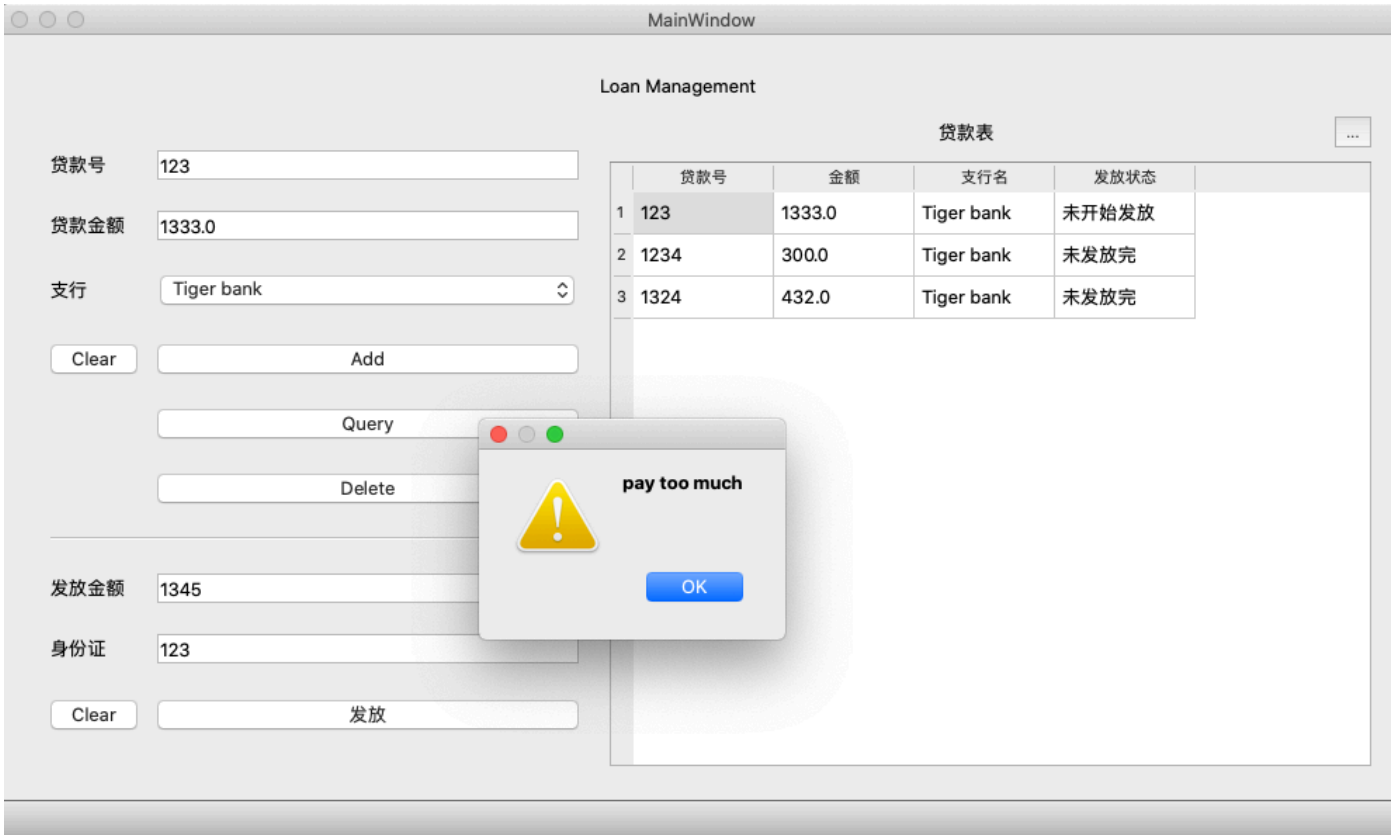
可以删除未发放或者已发放完的贷款，不可以删除未发放完的贷款



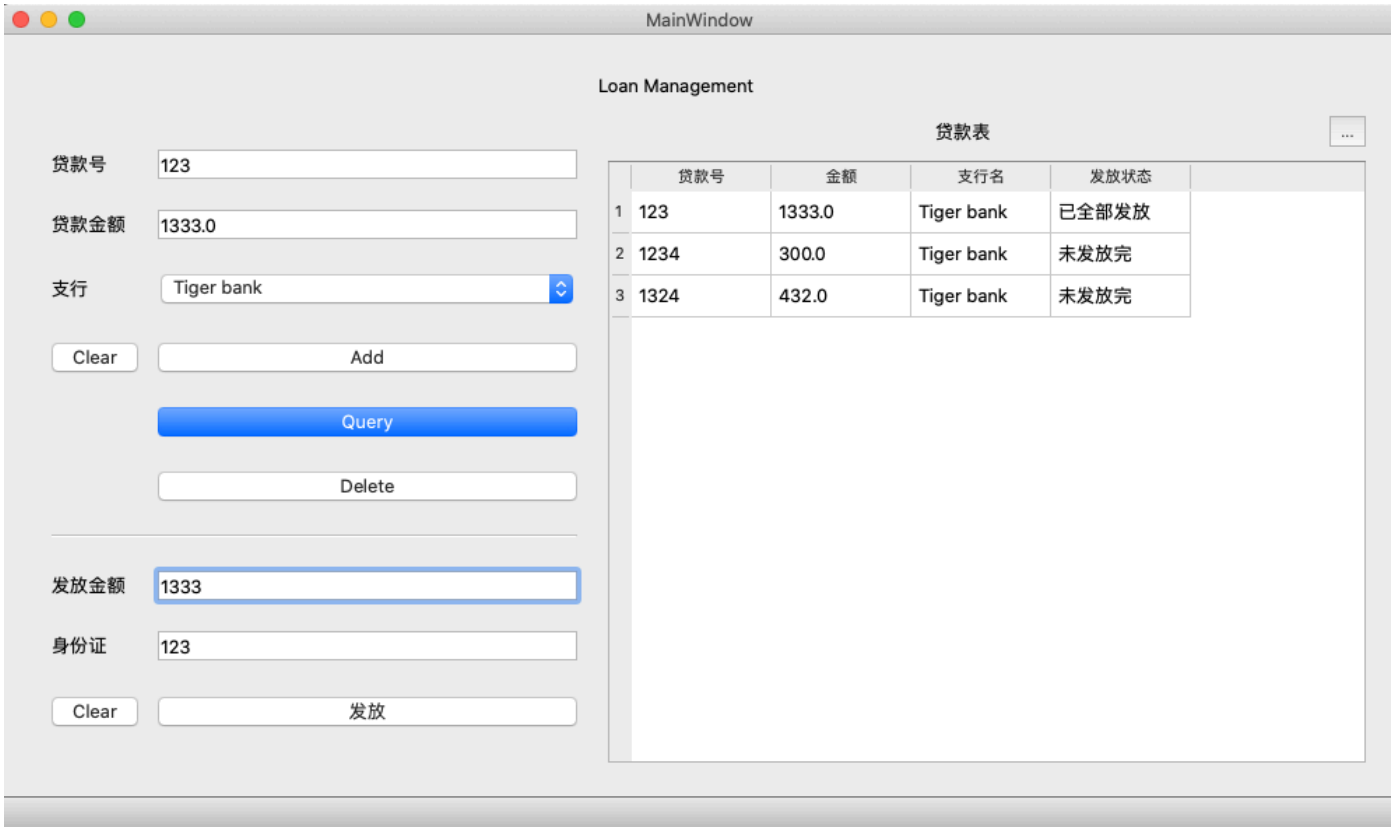
查询

同前述一样，支持关键词实时搜索。以及点击表格自动填充到编辑栏

发放贷款



如果超过贷款金额，将拒绝接收数据，即不发放。

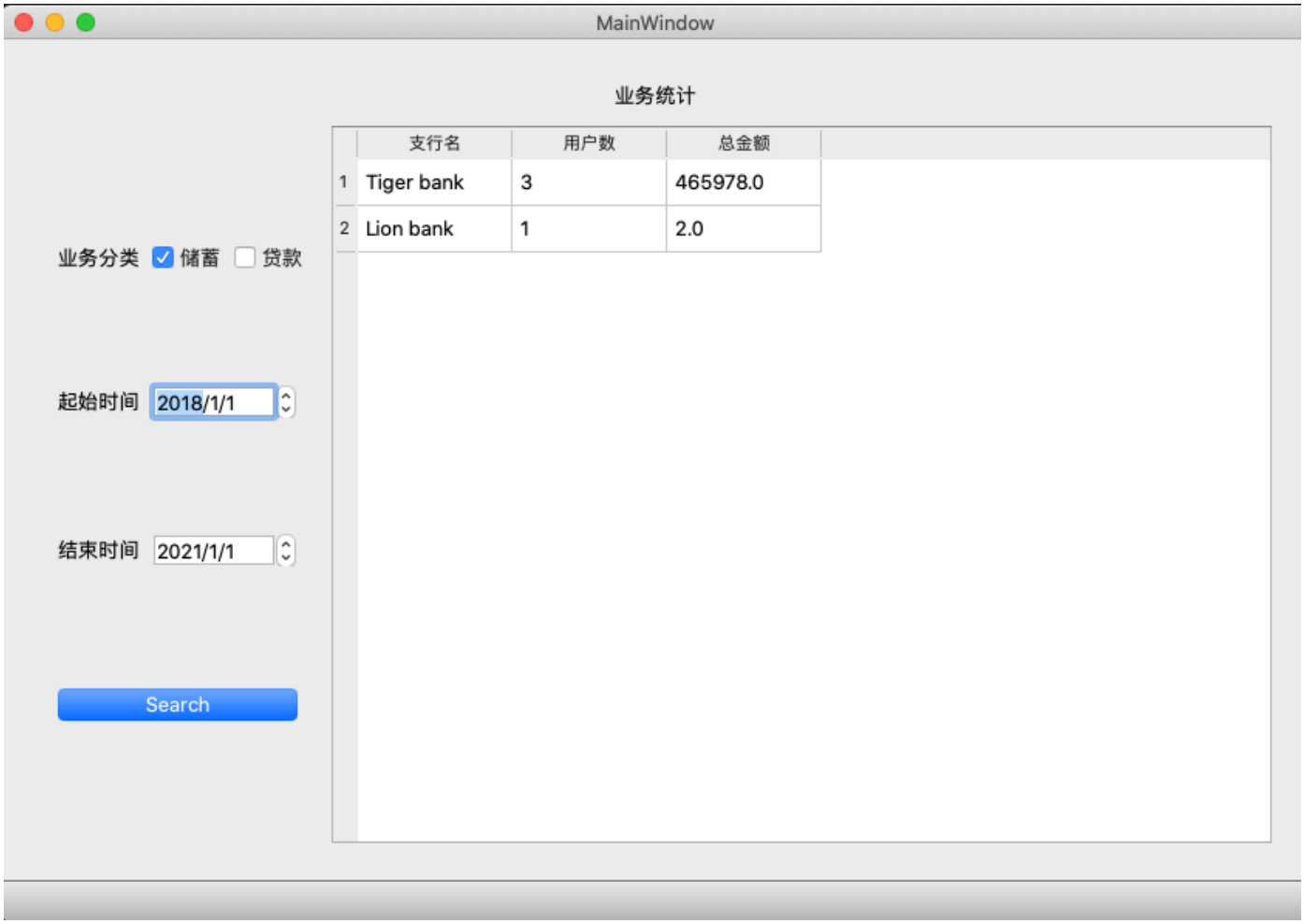


发放刚好则显示已全部发放。否则将为某一客户进行部分的发放（同一账户可有多个贷款）

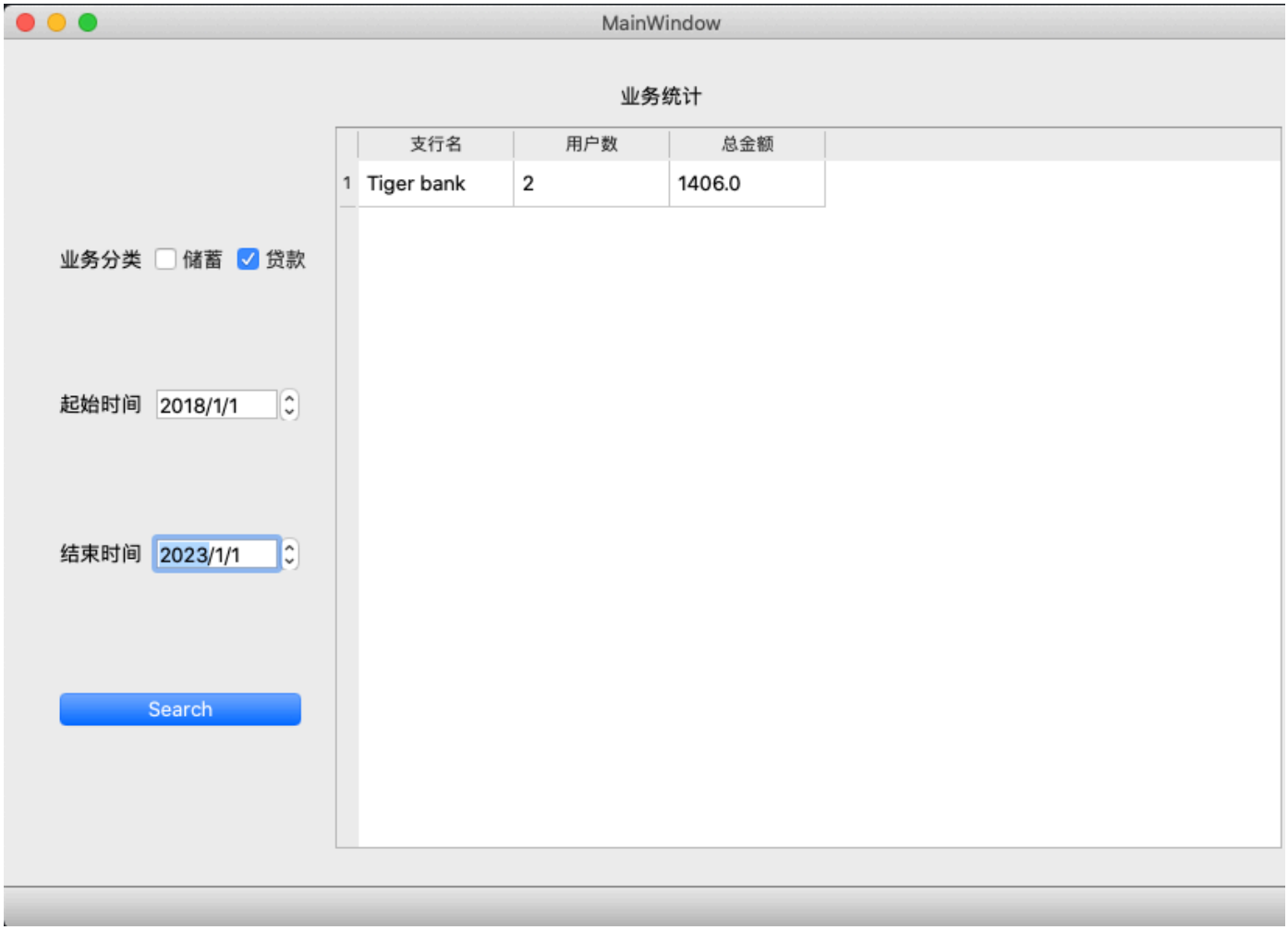
业务统计

储蓄统计

使用时间栏进行图形化选取，并聚合操作和显示


MainWindow
业务统计
支行名
用户数
总金额
1
Tiger bank
3
465978.0
2
Lion bank
1
2.0
业务分类
储蓄
贷款
起始时间
2018/1/1
结束时间
2021/1/1
Search

贷款统计



统计支行发放的贷款，如图

5.总结与讨论

工具收获

学会使用了PyQT，QT Creator，sqlalchemy进行GUI图形化开发，了解到了all in python 的方便之处。同时使用Pycharm IDE进行代码编写，发现其功能比VS Code对于Python的适配还是好上很多。

学习收获

- 对Python类的设计有了更深一步的认识，以及整个软件开发的流程也有很大程度的掌握
- 高内聚，低耦合对于程序的重要性

精神收获

- 独立写好一个大项目真的是太有意思辣

主要教训

- GUI编程是，要看清某个构件是否是我想要的构件，我曾不止一次的出错：把某构件当成当前页面上的构件，但其实他在别的页面，结果对其的操作（获取状态值和赋值）在当前页面都没反应「废话，在别的页面」。后来debug很久才发现是变量名不一致，其实弄到另一个构件了。
- 对于数据库设计要很早就敲定，并且保证其能有一致性，良好的满足实验要求：我曾不止一次为了修改一些外键约束，而删掉整个库重新来过。（不太清楚sqlalchemy的逻辑，导致不能一键重来）

- 先把软件的框架大体做好再进行开发，否则就会出现很多冗余设计（比如增删改查其实只需四个按钮，但用了四个tab页面去实现）；同时先做好框架有助于选择合适的变量名