“银行业务管理系统”

系统设计与实现报告

姓名：和泳毅

学号：PB19010450

大数据学院

中国科学技术大学

2021 年 6 月

目 录

1 概述

1.1 系统目标

本系统主要目标为开发一个银行管理系统。采用 B/S 架构，Python 语言，前端开发工具为 DW，后台 DBMS 使用 MySQL。

1.2 需求说明

1.2.1 数据需求

- **支行**：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。
- **客户**：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。
- **员工**：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。
- **账户**：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。
- **贷款**：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。

1.2.2 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金额和用户数，统计的结果以表格形式展示。

1.2.3 附加实现

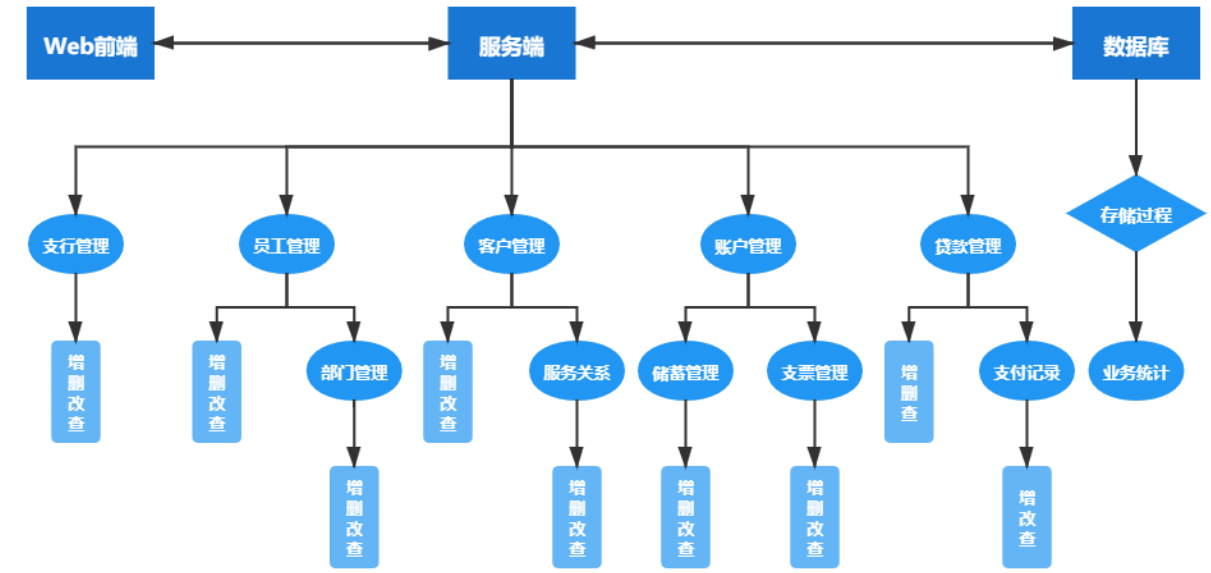
- **用户管理**：提供系统使用者的注册、登录功能；
- **支行管理**：提供支行所有信息的增、删、改、查功能；如果支行存在着关联信息，如员工、账户等，则不允许删除；
- **部门管理**：提供部门所有信息的增、删、改、查功能；如果部门存在着关联员工，则不允许删除；
- **员工管理**：提供支行员工所有信息的增、删、改、查功能；如果员工存在着关联服务关系， 则不允许删除；
- **客户管理**：提供客户员工关系的增、删、改、查功能。
- **业务统计**：统计的结果以饼图和表格形式展示。

1.3 本报告的主要贡献

本报告主要根据需求提供相应的体系架构设计，概述数据库设计、数据库实现、具体编程细节以及网页设计与实现。并根据具体实现细节提供相关功能的说明，展示最终效果。

2 总体设计

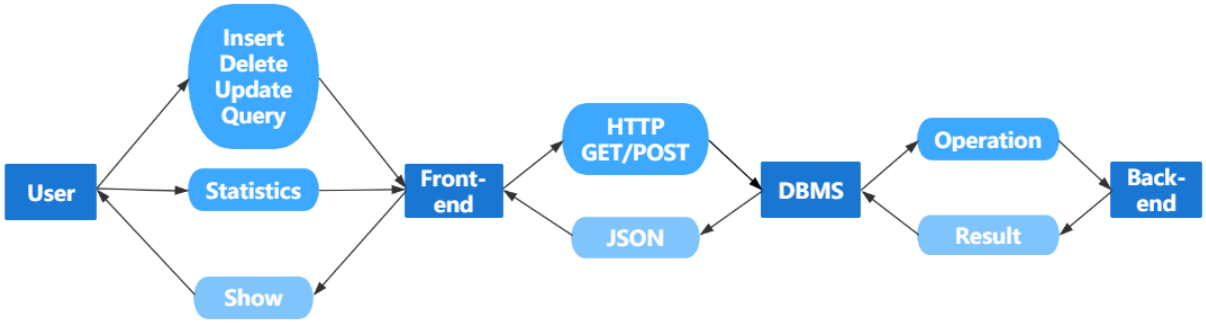
2.1 系统模块结构



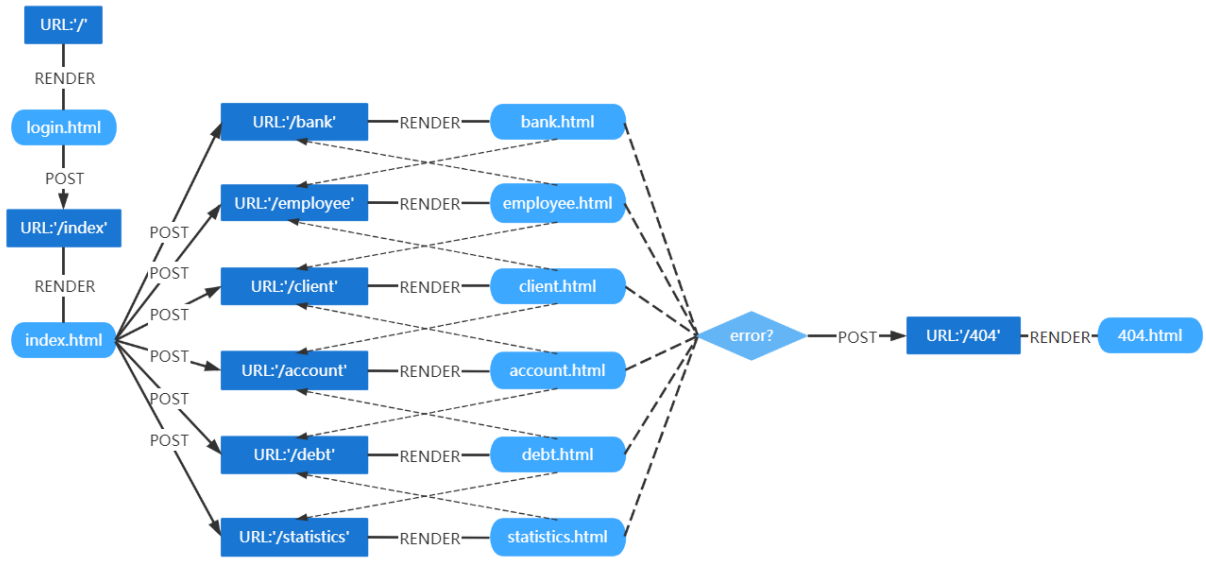
- Web 前端部分实现用户交互界面，提供相关操作接口；
- 服务端处理前端的操作请求，与数据库进行交互，实现下述子模块的功能：
  支行管理模块：实现对支行信息的增删改查；
  员工管理模块：实现对员工信息的增删改查，和对部门的增删改查；
  客户管理模块：实现对客户信息的增删改查，和对员工客户关系的增删改查；
  账户管理模块：实现对账户信息的增删改查，包括贷款账户和储蓄账户；
  贷款管理模块：实现对贷款信息的增删查，和对逐次支付记录的增改查。
- 数据库模块提供了上述信息的存储结构，并且通过存储过程，实现在数据库部分的业务统计功能。

2.2 系统工作流程

总体流程：



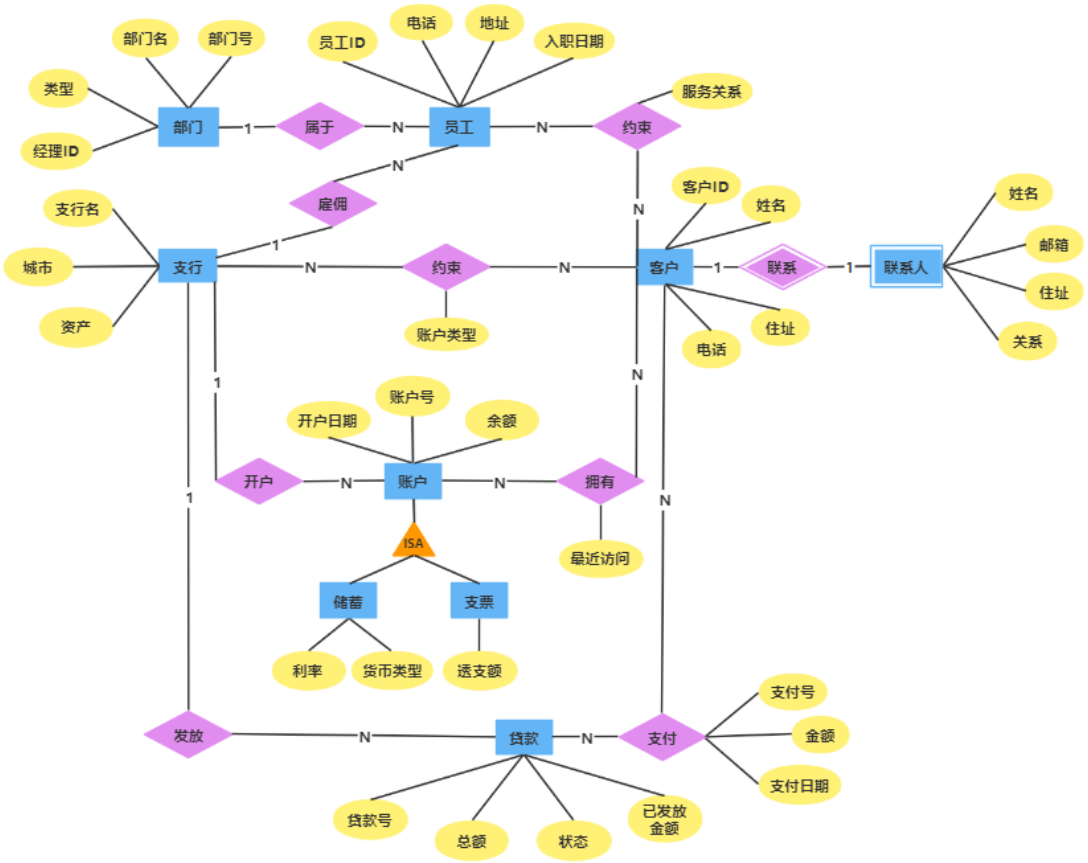
各模块通信关系：



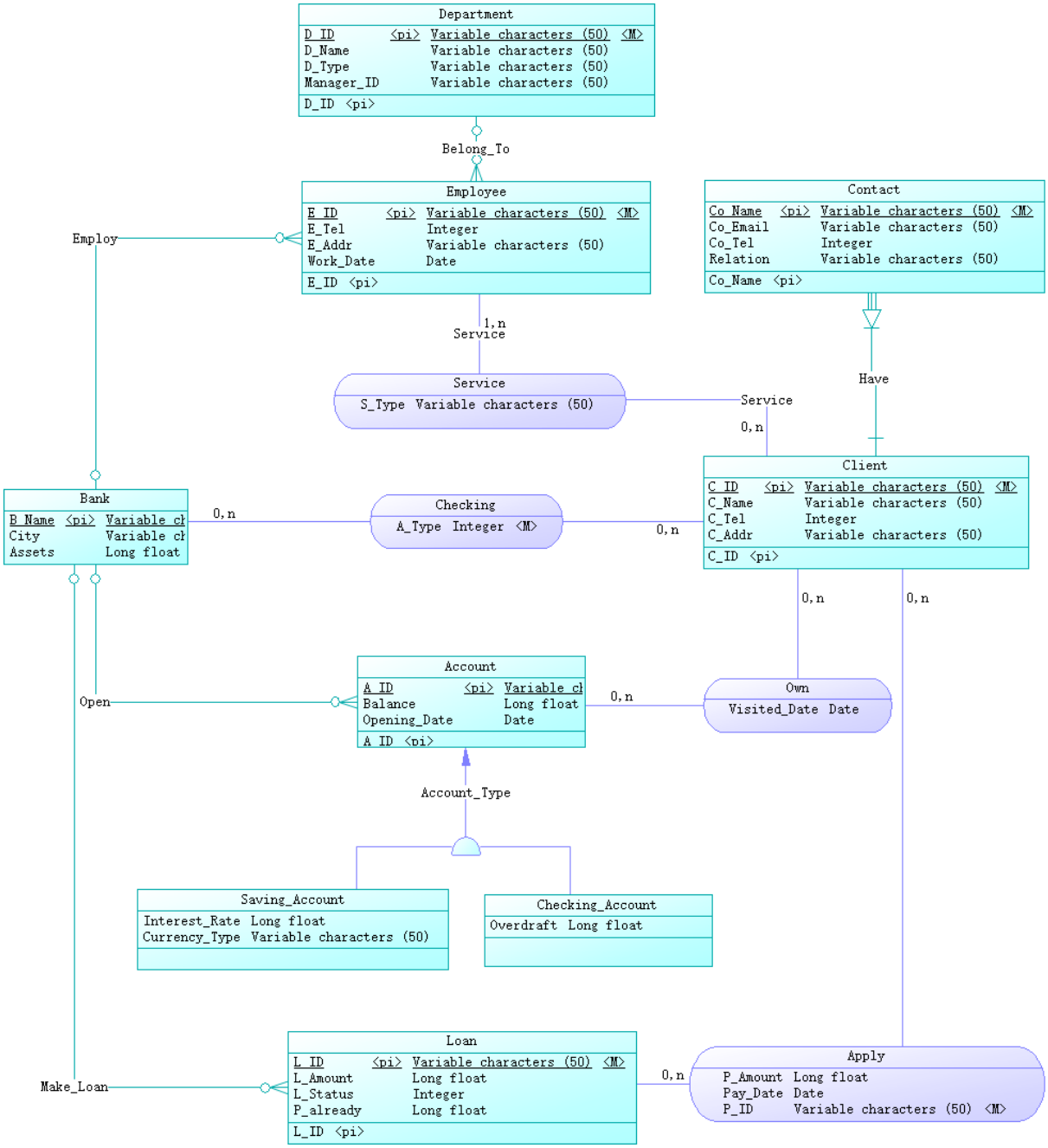
除了图示部分，各模块之间也可以互相通信，各模块与各接口间还可以通过给定的交互按钮进行 GET 请求的通信。

2.3 数据库设计

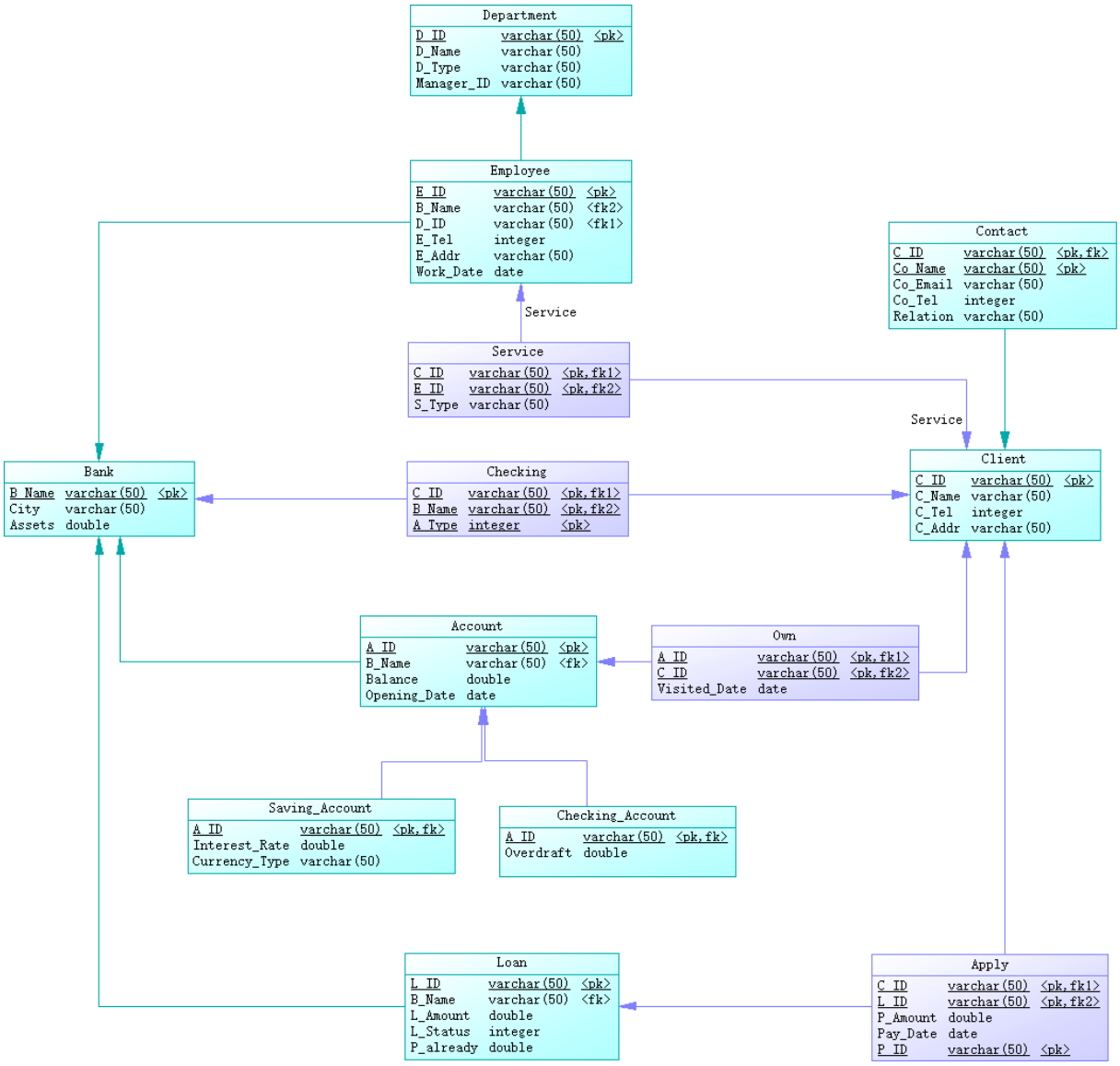
ER 图：



CDM：


Department
D_ID <pi> Variable characters (50) <M>
D_Name Variable characters (50)
D_Type Variable characters (50)
Manager_ID Variable characters (50)
D_ID <pi>
Belong_To
Employee
E_ID <pi> Variable characters (50) <M>
E_Tel Integer
E_Addr Variable characters (50)
Work_Date Date
E_ID <pi>
Contact
Co_Name <pi> Variable characters (50) <M>
Co_Email Variable characters (50)
Co_Tel Integer
Relation Variable characters (50)
Co_Name <pi>
Employ
1,n
Service
Have
Service
S_Type Variable characters (50)
Service
0,n
Client
C_ID <pi> Variable characters (50) <M>
C_Name Variable characters (50)
C_Tel Integer
C_Addr Variable characters (50)
C_ID <pi>
Bank
B_Name <pi> Variable ch
City Variable ch
Assets Long float
0,n
Checking
A_Type Integer <M>
0,n
0,n
0,n
Account
A_ID <pi> Variable ch
Balance Long float
Opening_Date Date
A_ID <pi>
Open
0,n
Own
Visited_Date Date
Account_Type
Saving_Account
Interest_Rate Long float
Currency_Type Variable characters (50)
Checking_Account
Overdraft Long float
Loan
L_ID <pi> Variable characters (50) <M>
L_Amount Long float
L_Status Integer
P_already Long float
L_ID <pi>
Make_Loan
0,n
Apply
P_Amount Long float
Pay_Date Date
P_ID Variable characters (50) <M>

PDM：


Department
D_ID varchar(50) <pk>
D_Name varchar(50)
D_Type varchar(50)
Manager_ID varchar(50)
Employee
E_ID varchar(50) <pk>
B_Name varchar(50) <fk2>
D_ID varchar(50) <fk1>
E_Tel integer
E_Addr varchar(50)
Work_Date date
Service
C_ID varchar(50) <pk,fk1>
E_ID varchar(50) <pk,fk2>
S_Type varchar(50)
Contact
C_ID varchar(50) <pk,fk>
Co_Name varchar(50) <pk>
Co_Email varchar(50)
Co_Tel integer
Relation varchar(50)
Service
Bank
B_Name varchar(50) <pk>
City varchar(50)
Assets double
Checking
C_ID varchar(50) <pk,fk1>
B_Name varchar(50) <pk,fk2>
A_Type integer <pk>
Client
C_ID varchar(50) <pk>
C_Name varchar(50)
C_Tel integer
C_Addr varchar(50)
Account
A_ID varchar(50) <pk>
B_Name varchar(50) <fk>
Balance double
Opening_Date date
Own
A_ID varchar(50) <pk,fk1>
C_ID varchar(50) <pk,fk2>
Visited_Date date
Saving_Account
A_ID varchar(50) <pk,fk>
Interest_Rate double
Currency_Type varchar(50)
Checking_Account
A_ID varchar(50) <pk,fk>
Overdraft double
Loan
L_ID varchar(50) <pk>
B_Name varchar(50) <fk>
L_Amount double
L_Status integer
P_already double
Apply
C_ID varchar(50) <pk,fk1>
L_ID varchar(50) <pk,fk2>
P_Amount double
Pay_Date date
P_ID varchar(50) <pk>

MYSQL 建表实现：

用户表：

```
create table User(
    username         varchar(50) not null,
    userkey          varchar(50) not null,
    primary key (username));
```

支行表：

```
create table Bank(
    B_ID         int not null,
    B_Name       varchar(50) not null,
    City         varchar(50) not null,
    Assets       float(15) not null,
    primary key (B_Name),
    unique key AK_B_ID (B_ID));
```

部门表：

```
create table Department(
    D_ID         varchar(50) not null,
    D_Name       varchar(50) not null,
    D_Type       varchar(50),
    Manager_ID   varchar(50),
    primary key (D_ID));
```

员工表：

```
create table Employee(
    E_ID         varchar(50) not null,
    E_Name       varchar(50) not null,
    B_Name       varchar(50) not null,
    D_ID         varchar(50),
    E_Tel        int,
    E_Addr       varchar(50),
    Work_Date    date,
    primary key (E_ID));

alter table Employee add constraint FK_Employ1 foreign key (D_ID)
      references Department (D_ID) on delete restrict on update restrict;
```

```
13.
14. alter table Employee add constraint FK_Employ2 foreign key (B_Name)
15.       references Bank (B_Name) on delete restrict on update restrict;
```

客户表:

```
1.  create table Client(
2.     C_ID          varchar(50) not null,
3.     C_Name        varchar(50) not null,
4.     C_Tel         int,
5.     C_Addr        varchar(50),
6.     primary key (C_ID));
```

联系人表:

```
1.  create table Contact(
2.     C_ID          varchar(50) not null,
3.     Co_Name       varchar(50) not null,
4.     Co_Email      varchar(50),
5.     Co_Tel        int,
6.     Relation      varchar(50),
7.     primary key (C_ID, Co_Name));
8.
9.  alter table Contact add constraint FK_Have foreign key (C_ID)
10.       references Client (C_ID) on delete restrict on update restrict;
```

账户表:

```
1.  create table Account(
2.     A_ID           varchar(50) not null,
3.     B_Name         varchar(50) not null,
4.     Balance        float(15),
5.     Opening_Date   date,
6.     primary key (A_ID));
7.
8.  alter table Account add constraint FK_Open foreign key (B_Name)
9.       references Bank (B_Name) on delete restrict on update restrict;
```

储蓄账户表：

```
create table Saving_Account(
    A_ID           varchar(50) not null,
    Interest_Rate  float(15),
    Currency_Type  varchar(50),
    primary key (A_ID));

alter table Saving_Account add constraint FK_Account_Type2 foreign key (A_ID)
       references Account (A_ID) on delete restrict on update restrict;
```

支票账户表：

```
create table Checking_Account(
    A_ID         varchar(50) not null,
    Overdraft    float(15),
    primary key (A_ID));

alter table Checking_Account add constraint FK_Account_Type foreign key (A_ID)
       references Account (A_ID) on delete restrict on update restrict;
```

贷款表：

```
create table Loan(
    L_ID         varchar(50) not null,
    B_Name       varchar(50) not null,
    L_Amount     float(15) not null,
    L_Status     int default 0 not null,
    P_already    float(15) not null,
    primary key (L_ID));

alter table Loan add constraint FK_Make_Loan foreign key (B_Name)
       references Bank (B_Name) on delete restrict on update restrict;
```

服务关系表：

```
create table Service(
    C_ID         varchar(50) not null,
    E_ID         varchar(50) not null,
    S_Type       varchar(50),
    primary key (C_ID, E_ID));
```

```
6.
7.  alter table Service add constraint FK_Service foreign key (C_ID)
8.        references Client (C_ID) on delete restrict on update restrict;
9.
10. alter table Service add constraint FK_Service2 foreign key (E_ID)
11.       references Employee (E_ID) on delete restrict on update restrict;
```

开户约束表:

```
1.  create table Checking(
2.     C_ID          varchar(50) not null,
3.     B_Name        varchar(50) not null,
4.     A_Type        int not null,
5.     primary key (C_ID, B_Name, A_Type));
6.
7.  alter table Checking add constraint FK_Checking1 foreign key (C_ID)
8.        references Client (C_ID) on delete restrict on update restrict;
9.
10. alter table Checking add constraint FK_Checking2 foreign key (B_Name)
11.       references Bank (B_Name) on delete restrict on update restrict;
```

账户持有表:

```
1.  create table Own(
2.     C_ID          varchar(50) not null,
3.     A_ID          varchar(50),
4.     Visited_Date date,
5.     primary key (C_ID, A_ID));
6.
7.  alter table Own add constraint FK_Own1 foreign key (C_ID)
8.        references Client (C_ID) on delete restrict on update restrict;
9.
10. alter table Own add constraint FK_Own2 foreign key (A_ID)
11.       references Account (A_ID) on delete restrict on update restrict;
```

贷款发放表:

```
1.  create table Apply(
2.     C_ID          varchar(50) not null,
3.     L_ID          varchar(50) not null,
4.     P_ID          varchar(50) not null,
5.     P_Amount      float(15),
```

```
    Pay_Date     date,
    primary key (C_ID, L_ID, P_ID));

alter table Apply add constraint FK_Apply foreign key (C_ID)
      references Client (C_ID) on delete restrict on update restrict;
alter table Apply add constraint FK_Apply2 foreign key (L_ID)
      references Loan (L_ID) on delete restrict on update restrict;
```

3 详细设计

3.1 用户模块

目标：提供系统使用者的注册、登录功能。

URL：http://127.0.0.1:8000/login

model：

```
class User(db.Model):
    __tablename__ = 'user'

    username = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'), primary_key=True)
    userkey = db.Column(db.String(50), nullable=False)
```

app：

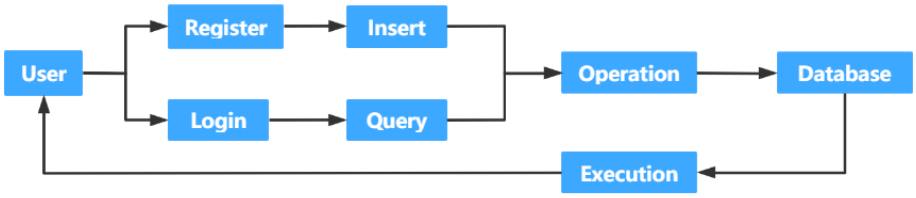
```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'GET':
        return render_template('login.html')
    else:
        if request.form.get('type') == 'signup':
            name = request.form.get('name')
            key = request.form.get('password')

            newUser = User(
                username=name,
                userkey=key,
            )

            db.session.add(newUser)
```

```
            db.session.commit()
            return render_template('login.html')
        elif request.form.get('type') == 'login':
            name = request.form.get('name')
            key = request.form.get('password')
            UserNotExist = db.session.query(User).filter_by(username=name).scalar()
is None

            if UserNotExist == 1:
                error_title = '登录错误'
                error_message = '用户名不存在'
                return render_template('404.html', error_title=error_title, error_me
ssage=error_message)

            user_result = db.session.query(User).filter_by(username=name).first()
            if user_result.userkey == key:
                return render_template('index.html')
            else:
                error_title = '登录错误'
                error_message = '密码错误'
                return render_template('404.html', error_title=error_title, error_me
ssage=error_message)
    return render_template('login.html')
```

输入：用户名、密码

输出：注册（向用户表插入新用户内容）、登录（在用户表中查询比对密码，转向 404.html 或 index.html）

流程图：



页面设计：

3.2 支行模块

目标：提供支行所有信息的增、删、改、查功能，如果支行存在着关联信息，如员工、账户等，则不允许删除。

URL：http://127.0.0.1:8000/bank

model：

```
class Bank(db.Model):
    __tablename__ = 'bank'

    B_ID = db.Column(db.Integer, nullable=False, unique=True)
    B_Name = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'), primary_key=True)
    City = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'), nullable=False)
    Assets = db.Column(db.Float, nullable=False)
```

app：

```
@app.route('/bank', methods=['GET', 'POST'])
def bank():
    labels = ['支行号', '支行名', '支行资产', '所在城市']
    result_query = db.session.query(Bank)
    result = result_query.all()
    if request.method == 'GET':
        return render_template('bank.html', labels=labels, content=result)
    else:
        if request.form.get('type') == 'query':
            bank_id = request.form.get('id')
```

```
            bank_name = request.form.get('name')
            bank_city = request.form.get('city')
            bank_asset = request.form.get('assets')

            if bank_id != "":
                result_query = result_query.filter(Bank.B_ID == bank_id)
            if bank_name != "":
                result_query = result_query.filter(Bank.B_Name == bank_name)
            if bank_asset != "":
                result_query = result_query.filter(Bank.Assets == bank_asset)
            if bank_city != "":
                result_query = result_query.filter(Bank.City == bank_city)

            result = result_query.all()

            return render_template('bank.html', labels=labels, content=result)

        elif request.form.get('type') == 'update':
            old_num = request.form.get('key')
            bank_name = request.form.get('bank_name')
            bank_asset = request.form.get('bank_asset')
            bank_city = request.form.get('bank_city')
            bank_result = db.session.query(Bank).filter_by(B_ID=old_num).first()
            bank_result.B_Name = bank_name
            bank_result.Assets = bank_asset
            bank_result.City = bank_city
            db.session.commit()

        elif request.form.get('type') == 'delete':
            old_num = request.form.get('key')

            BankNotExist = db.session.query(Employee).filter_by(B_Name=old_num).scal
ar() is None

            if BankNotExist != 1:
                error_title = '删除错误'
                error_message = '支行在存在关联员工'
                return render_template('404.html', error_title=error_title, error_me
ssage=error_message)

            BankNotExist = db.session.query(Account).filter_by(B_Name=old_num).scala
r() is None

            if BankNotExist != 1:
```

```
52.                 error_title = '删除错误'
53.                 error_message = '支行在存在关联账户'
54.                 return render_template('404.html', error_title=error_title, error_me
    ssage=error_message)
55.
56.             BankNotExist = db.session.query(Loan).filter_by(B_Name=old_num).scalar()
     is None
57.
58.             if BankNotExist != 1:
59.                 error_title = '删除错误'
60.                 error_message = '支行在存在关联贷款'
61.                 return render_template('404.html', error_title=error_title, error_me
    ssage=error_message)
62.
63.             BankNotExist = db.session.query(Checking).filter_by(B_Name=old_num).scal
    ar() is None
64.
65.             if BankNotExist != 1:
66.                 error_title = '删除错误'
67.                 error_message = '支行在存在关联信息'
68.                 return render_template('404.html', error_title=error_title, error_me
    ssage=error_message)
69.
70.             bank_result = db.session.query(Bank).filter_by(B_ID=old_num).first()
71.             db.session.delete(bank_result)
72.             db.session.commit()
73.
74.         elif request.form.get('type') == 'insert':
75.             bank_id = request.form.get('id')
76.             bank_name = request.form.get('name')
77.             bank_asset = request.form.get('estate')
78.             bank_city = request.form.get('city')
79.
80.             newBank = Bank(
81.                 B_ID=bank_id,
82.                 B_Name=bank_name,
83.                 Assets=bank_asset,
84.                 City=bank_city
85.             )
86.
87.             db.session.add(newBank)
88.             db.session.commit()
89.
90.     result = db.session.query(Bank).all()
```

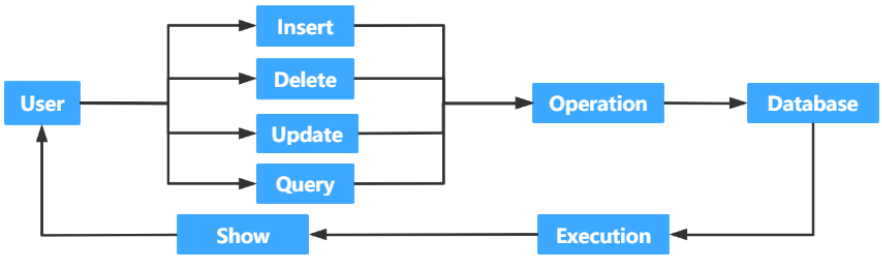
```
91.     return render_template('bank.html', labels=labels, content=result)
```

各模块对增删改查部分的实现方法与步骤都一致，之后的模块将不再赘述。

输入：支行号、支行名、支行资产、所在城市

输出：添加（向支行表插入新支行内容）、删除（在支行表中删除符合要求的内容）、修改（在支行表中修改符合要求的内容）、查询（在支行表中查询指定的内容）

流程图：



页面设计：

3.3 员工模块

目标：提供支行员工所有信息的增、删、改、查功能，如果员工存在着关联服务关系，则不允许删除。提供部门所有信息的增、删、改、查功能；如果部门存在着关联员工，则不允许删除。

URL：http://127.0.0.1:8000/employee

model:

```
class Employee(db.Model):
    __tablename__ = 'employee'

    E_ID = db.Column(db.String(50), primary_key=True)
    E_Name = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'), nullable=False)
    B_Name = db.Column(db.ForeignKey('bank.B_Name', ondelete='RESTRICT', onupdate='RESTRICT'), nullable=False)
    D_ID = db.Column(db.ForeignKey('department.D_ID', ondelete='RESTRICT', onupdate='RESTRICT'))
    E_Tel = db.Column(db.Integer)
    E_Addr = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'))
    Work_Date = db.Column(db.Date)

    bank = db.relationship('Bank', primaryjoin='Employee.B_Name == Bank.B_Name', backref='employees')
    department = db.relationship('Department', primaryjoin='Employee.D_ID == Department.D_ID', backref='employees')

class Department(db.Model):
    __tablename__ = 'department'

    D_ID = db.Column(db.String(50), primary_key=True)
    D_Name = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'), nullable=False)
    D_Type = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'))
    Manager_ID = db.Column(db.String(50))
```

app:

```
@app.route('/employee', methods=['GET', 'POST'])
def employee():
    labels1 = ['员工 ID', '员工姓名', '员工电话', '员工住址', '雇佣日期', '所在支行', '部门号', '部门名称', '部门类型', '部门经理 ID']
    labels2 = ['部门号', '部门名称', '部门类型', '部门经理 ID']
    result_query = db.session.query(Employee, Department).filter(Employee.D_ID == Department.D_ID)
    result = result_query.all()
    result_query2 = db.session.query(Department)
    result2 = result_query2.all()

    if request.method == 'GET':
        return render_template('employee.html', labels1=labels1, labels2=labels2, content=result, content2=result2)
```

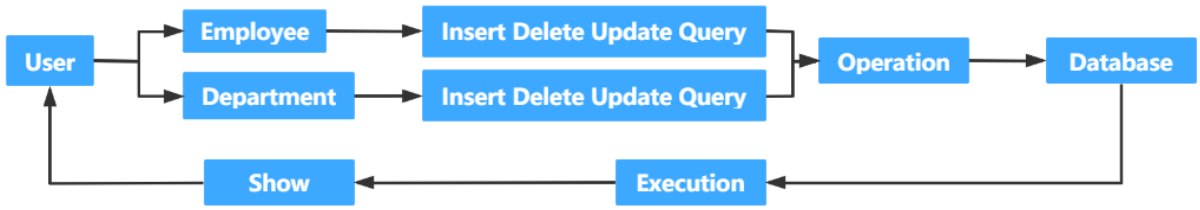
```
    else:
        if request.form.get('type') == 'query1':
            '''...'''
            result = result_query.all()
            return render_template('employee.html', labels1=labels1, labels2=labels2, content=result, content2=result2)

        elif request.form.get('type') == 'query2':
            '''...'''
            result = result_query.all()
            result2 = result_query2.all()
            return render_template('employee.html', labels1=labels1, labels2=labels2, content=result, content2=result2)

        elif request.form.get('type') == 'update1':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'update2':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'delete1':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'delete2':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'insert1':
            '''...'''
            db.session.commit()
            result = db.session.query(Employee, Department).filter(Employee.D_ID == Department.D_ID).all()
            return render_template('employee.html', labels1=labels1, labels2=labels2, content=result, content2=result2)

        elif request.form.get('type') == 'insert2':
            '''...'''
            db.session.commit()

    result = db.session.query(Employee, Department).filter(Employee.D_ID == Department.D_ID).all()
```

```
51.     result2 = db.session.query(Department).all()
52.     return render_template('employee.html', labels1=labels1, labels2=labels2, conten
    t=result, content2=result2)
```

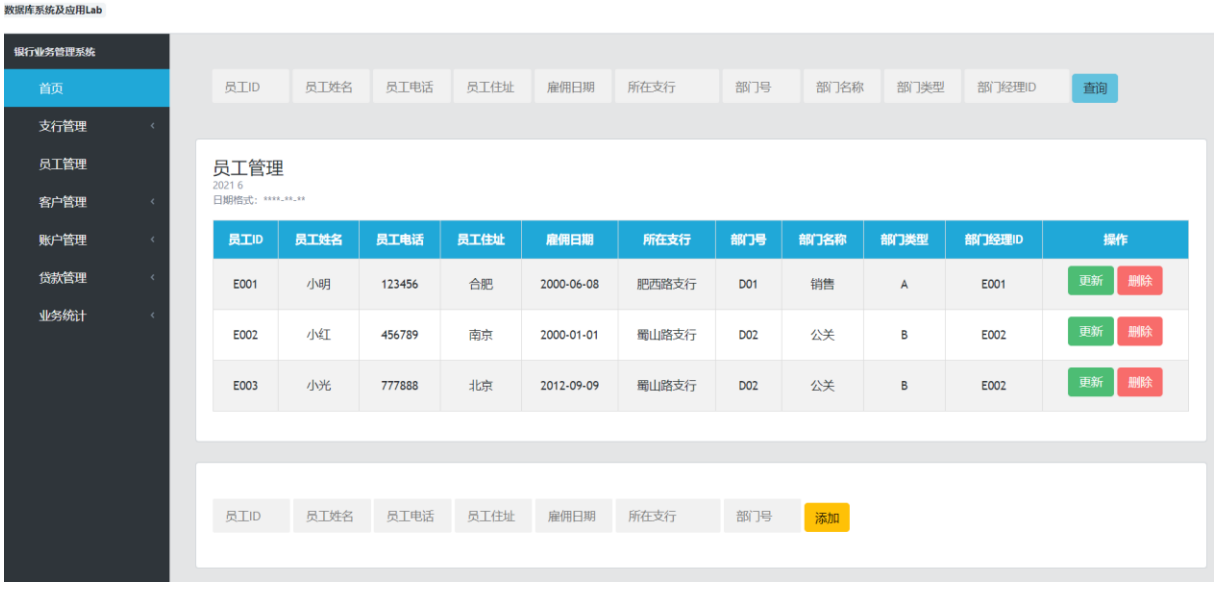
输入： 员工 ID、员工姓名、员工电话、员工住址、雇佣日期、所在支行、部门号；
部门号、部门名称、部门类型、部门经理 ID

输出：添加（向员工表、部门表插入新内容）、删除（在员工表、部门表中删除符合要求的内容）、修改（在员工表、部门表中修改符合要求的内容）、查询（在员工表、部门表中查询指定的内容）

流程图：



页面设计：



3.4 客户模块

目标：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。同时记录客户与员工的服务关系。

URL：http://127.0.0.1:8000/client

model：

```
class Client(db.Model):
    __tablename__ = 'client'

    C_ID = db.Column(db.String(50), primary_key=True)
    C_Name = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'), nullable=False)
    C_Tel = db.Column(db.Integer)
    C_Addr = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'))

class Contact(db.Model):
    __tablename__ = 'contact'

    C_ID = db.Column(db.ForeignKey('client.C_ID', ondelete='RESTRICT', onupdate='RESTRICT'), primary_key=True, nullable=False)
    Co_Name = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'), primary_key=True, nullable=False)
    Co_Email = db.Column(db.String(50))
    Co_Tel = db.Column(db.Integer)
    Relation = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'))

```

```
    client = db.relationship('Client', primaryjoin='Contact.C_ID == Client.C_ID', backref='contacts')

class Service(db.Model):
    __tablename__ = 'service'

    C_ID = db.Column(db.ForeignKey('client.C_ID', ondelete='RESTRICT', onupdate='RESTRICT'), primary_key=True, nullable=False)
    E_ID = db.Column(db.ForeignKey('employee.E_ID', ondelete='RESTRICT', onupdate='RESTRICT'), primary_key=True, nullable=False, index=True)
    S_Type = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'))

    client = db.relationship('Client', primaryjoin='Service.C_ID == Client.C_ID', backref='services')
    employee = db.relationship('Employee', primaryjoin='Service.E_ID == Employee.E_ID', backref='services')
```

app：

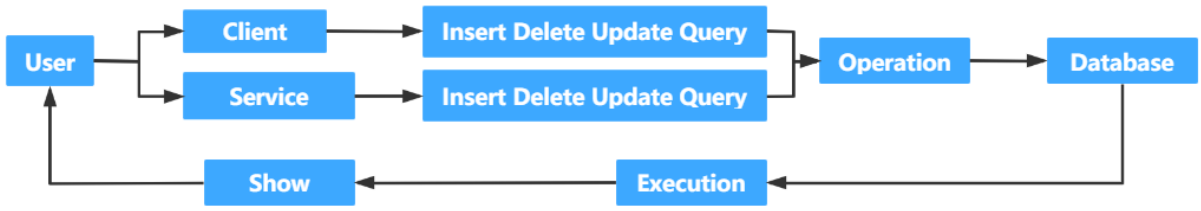
```
@app.route('/client', methods=['GET', 'POST'])
def client():
    labels1 = ['客户 ID', '客户姓名', '客户电话', '客户住址', '联系人姓名', '联系人电话', '联系人邮箱', '关系']
    labels2 = ['客户 ID', '员工 ID', '服务类型']
    result_query1 = db.session.query(Client, Contact).filter(Client.C_ID == Contact.C_ID)
    result_query2 = db.session.query(Client, Service).filter(Client.C_ID == Service.C_ID)
    result1 = result_query1.all()
    result2 = result_query2.all()

    if request.method == 'GET':
        return render_template('client.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)
    else:
        if request.form.get('type') == 'query1':
            '''...'''
            result1 = result_query1.all()
            return render_template('client.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)

        elif request.form.get('type') == 'query2':
            '''...'''
            result2 = result_query2.all()
```

```
            return render_template('client.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)

        elif request.form.get('type') == 'update1':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'update2':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'delete1':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'delete2':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'insert2':
            '''...'''
            db.session.commit()
            result2 = db.session.query(Client, Service).filter(Client.C_ID == Service.C_ID).all()
            return render_template('client.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)

        elif request.form.get('type') == 'insert1':
            '''...'''
            db.session.commit()
    result1 = db.session.query(Client, Contact).filter(Client.C_ID == Contact.C_ID).all()

    return render_template('client.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)
```

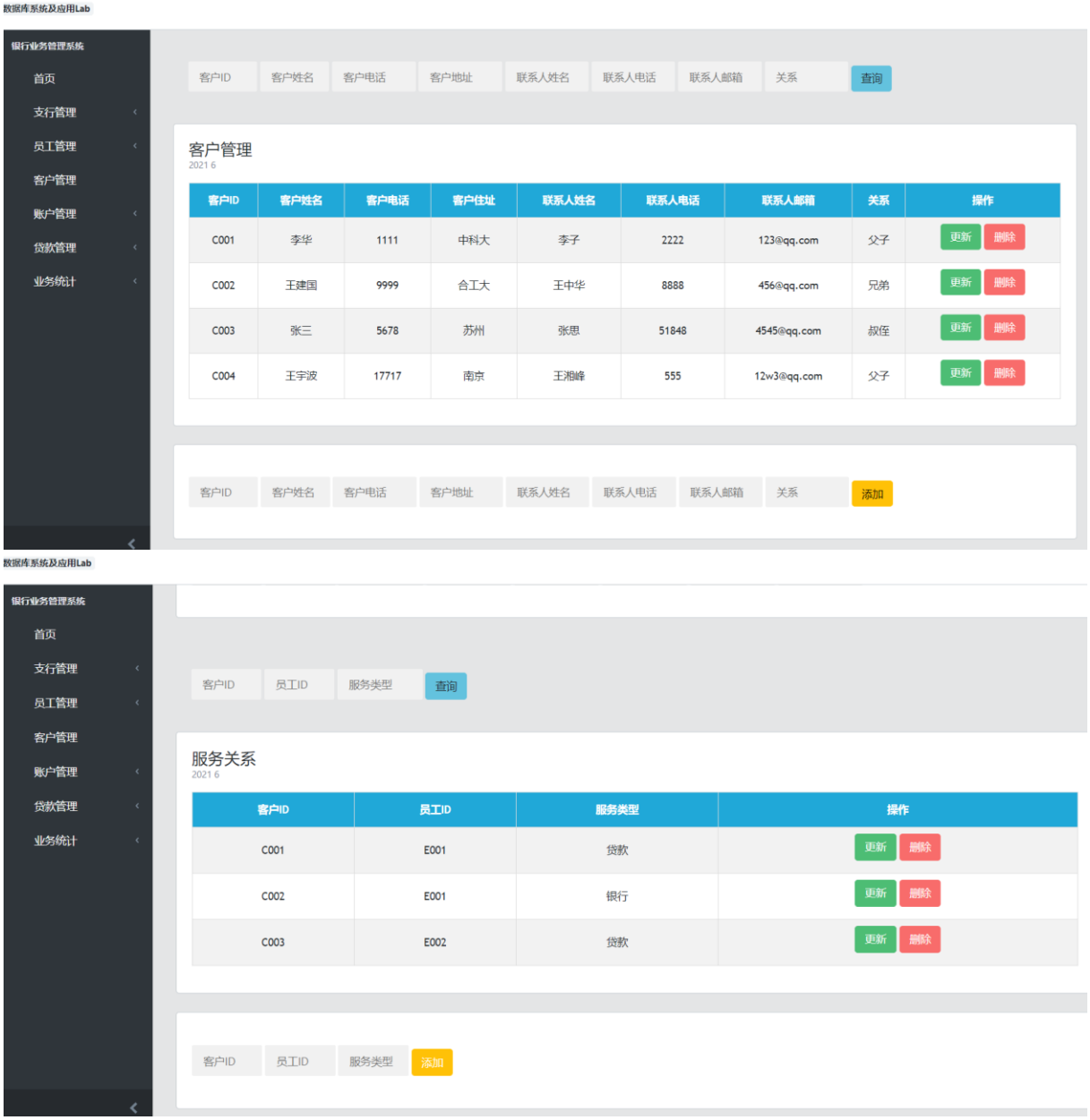
输入：客户 ID、客户姓名、客户电话、客户住址、联系人姓名、联系人邮箱、关系；
客户 ID、员工 ID、服务关系。

输出：添加（向客户表、服务表插入新内容）、删除（在客户表、服务表中删除符合要求的内容）、修改（在客户表、服务表中修改符合要求的内容）、查询（在客户表、服务表中查询指定的内容）

流程图：



页面设计：



3.5 账户模块

目标：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。

URL：http://127.0.0.1:8000/account

model：

```
class Account(db.Model):
    __tablename__ = 'account'

    A_ID = db.Column(db.String(50), primary_key=True)
    B_Name = db.Column(db.ForeignKey('bank.B_Name', ondelete='RESTRICT', onupdate='RESTRICT'), nullable=False, index=True)
    Balance = db.Column(db.Float)
    Opening_Date = db.Column(db.Date)

    bank = db.relationship('Bank', primaryjoin='Account.B_Name == Bank.B_Name', backref='accounts')

class CheckingAccount(db.Model):
    __tablename__ = 'checking_account'

    A_ID = db.Column(db.ForeignKey('account.A_ID', ondelete='RESTRICT', onupdate='RESTRICT'), primary_key=True)
    Overdraft = db.Column(db.Float)

class SavingAccount(db.Model):
    __tablename__ = 'saving_account'

    A_ID = db.Column(db.ForeignKey('account.A_ID', ondelete='RESTRICT', onupdate='RESTRICT'), primary_key=True)
    Interest_Rate = db.Column(db.Float)
    Currency_Type = db.Column(db.String(15, 'utf8mb4_0900_ai_ci'))
```

app：

```
@app.route('/client', methods=['GET', 'POST'])
def client():
    labels1 = ['客户 ID', '客户姓名', '客户电话', '客户住址', '联系人姓名', '联系人电话', '联系人邮箱', '关系']
    labels2 = ['客户 ID', '员工 ID', '服务类型']
    result_query1 = db.session.query(Client, Contact).filter(Client.C_ID == Contact.C_ID)
```

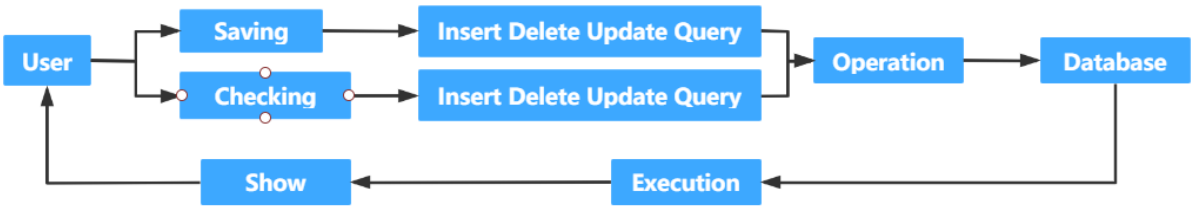
```
    result_query2 = db.session.query(Client, Service).filter(Client.C_ID == Service.C_ID)
    result1 = result_query1.all()
    result2 = result_query2.all()

    if request.method == 'GET':
        return render_template('client.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)
    else:
        if request.form.get('type') == 'query1':
            '''...'''
            result1 = result_query1.all()
            return render_template('client.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)

        elif request.form.get('type') == 'query2':
           '''...'''
            result2 = result_query2.all()
            return render_template('client.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)

        elif request.form.get('type') == 'update1':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'update2':
           '''...'''
            db.session.commit()

        elif request.form.get('type') == 'delete1':
           '''...'''
            db.session.commit()

        elif request.form.get('type') == 'delete2':
           '''...'''
            db.session.commit()

        elif request.form.get('type') == 'insert2':
            '''...'''
            db.session.commit()
            result2 = db.session.query(Client, Service).filter(Client.C_ID == Service.C_ID).all()
            return render_template('client.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)
```

```
44.
45.         elif request.form.get('type') == 'insert1':
46.             '''...'''
47.             db.session.commit()
48.     result1 = db.session.query(Client, Contact).filter(Client.C_ID == Contact.C_ID).
    all()
49.
50.     return render_template('client.html', labels1=labels1, labels2=labels2, content1
    =result1, content2=result2)
```

输入：账户号、客户 ID、客户姓名、开户支行、开户时间、账户余额、最近访问、（利率、货币类型）/（透支额）

输出：添加（向账户表插入新内容）、删除（在账户表中删除符合要求的内容）、修改（在账户表中修改符合要求的内容）、查询（在账户表中查询指定的内容）

流程图：



页面设计：

3.6 贷款模块

目标：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、已全发放），处于发放中状态的贷款记录不允许删除。

URL：http://127.0.0.1:8000/debt

model：

```
1.  class Loan(db.Model):
2.      __tablename__ = 'loan'
3.
4.      L_ID = db.Column(db.String(50), primary_key=True)
5.      B_Name = db.Column(db.ForeignKey('bank.B_Name', ondelete='RESTRICT', onupdate='R
    ESTRICT'), nullable=False, index=True)
6.      L_Amount = db.Column(db.Float, nullable=False)
7.      L_Status = db.Column(db.Integer, nullable=False, server_default=db.FetchedValue(
    ))
8.      P_already = db.Column(db.Float)
9.
10.     bank = db.relationship('Bank', primaryjoin='Loan.B_Name == Bank.B_Name', backref
    ='loans')
11.
12. class Apply(db.Model):
13.     __tablename__ = 'apply'
14.
```

```
    C_ID = db.Column(db.ForeignKey('client.C_ID', ondelete='RESTRICT', onupdate='RESTRICT'), primary_key=True, nullable=False)
    L_ID = db.Column(db.ForeignKey('loan.L_ID', ondelete='RESTRICT', onupdate='RESTRICT'), primary_key=True, nullable=False, index=True)
    P_ID = db.Column(db.String(50), primary_key=True)
    P_Amount = db.Column(db.Float)
    Pay_Date = db.Column(db.Date)

    client = db.relationship('Client', primaryjoin='Apply.C_ID == Client.C_ID', backref='applies')
    loan = db.relationship('Loan', primaryjoin='Apply.L_ID == Loan.L_ID', backref='applies')
```

app：

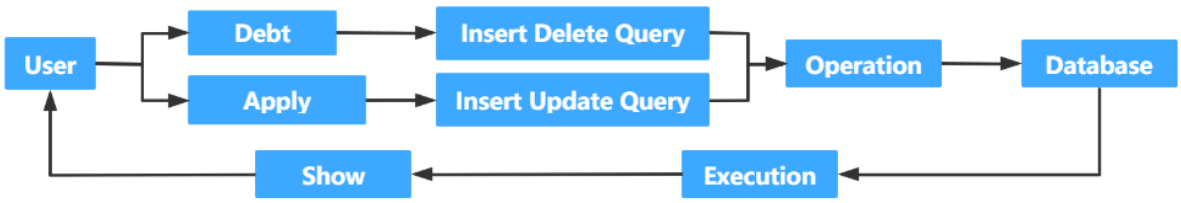
```
@app.route('/debt', methods=['GET', 'POST'])
def debt():
    labels1 = ['贷款号', '发放支行', '贷款金额', '贷款状态']
    labels2 = ['支付号', '贷款号', '客户 ID', '支付金额', '支付日期']

    content_query1 = db.session.query(Loan)
    content_query2 = db.session.query(Apply)
    result1 = content_query1.all()
    result2 = content_query2.all()

    if request.method == 'GET':
        return render_template('debt.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)
    else:
        if request.form.get('type') == 'main_query':
            '''...'''
            result1 = content_query1.all()
            return render_template('debt.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)

        elif request.form.get('type') == 'update':
            '''...'''

        elif request.form.get('type') == 'delete':
            '''...'''
            db.session.commit()

        elif request.form.get('type') == 'query':
            '''...'''
```

```
            result2 = content_query2.all()
            return render_template('debt.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)

        elif request.form.get('type') == 'give':
            '''...'''

    content_query1 = db.session.query(Loan)
    content_query2 = db.session.query(Apply)
    result1 = content_query1.all()
    result2 = content_query2.all()

    return render_template('debt.html', labels1=labels1, labels2=labels2, content1=result1, content2=result2)
```

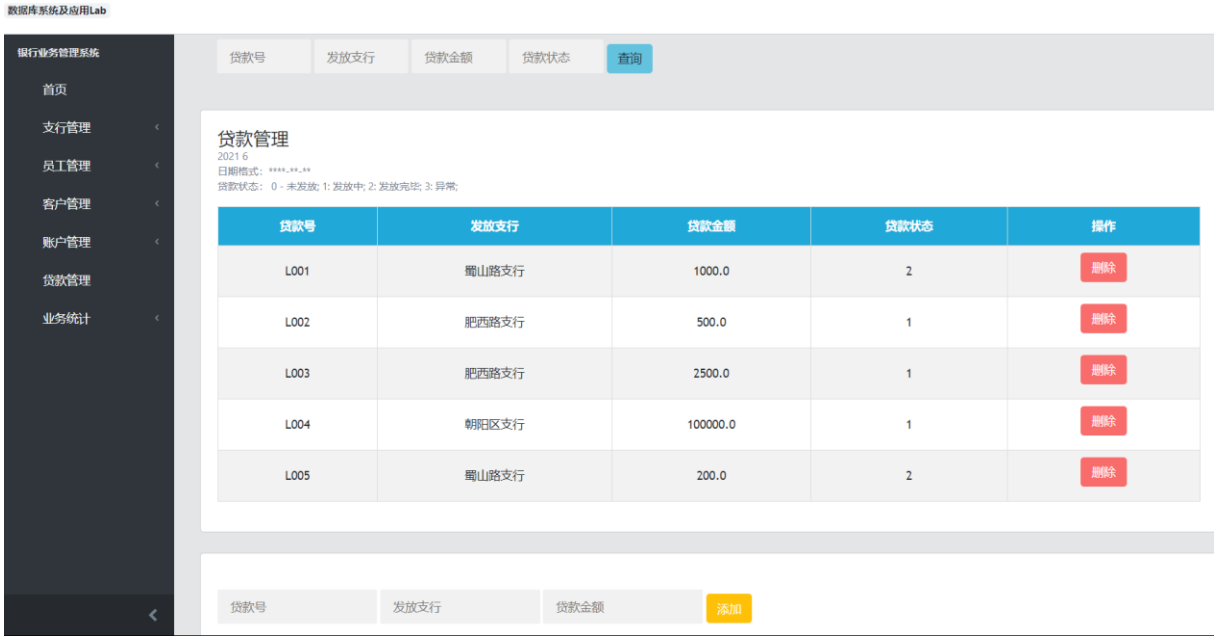
输入： 贷款号、发放支行、贷款余额、贷款状态；
待发放贷款号、发放客户 ID、支付号、发放日期、发放金额。

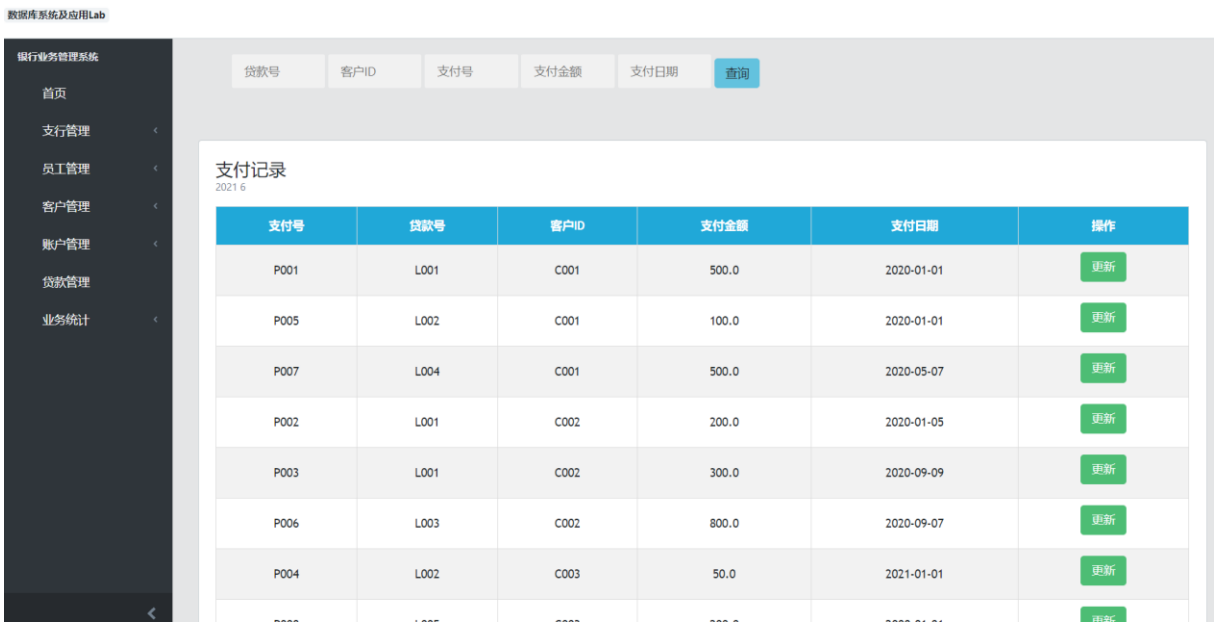
输出：添加（向贷款表、发放表插入新内容）、删除（在贷款表、发放表中删除符合要求的内容）、修改（在贷款表、发放表中修改符合要求的内容）、查询（在贷款表、发放表中查询指定的内容）

流程图：



页面设计：





3.7 统计模块

目标：按业务分类（储蓄、贷款）和时间（月、季、年）统计各个支行的业务金额和用户数，统计的结果以饼图、表格形式展示。

URL：http://127.0.0.1:8000/statistics

PROCEDURE：

```
Delimiter //
DROP PROCEDURE IF EXISTS cxyear;
create
    definer = root@localhost procedure cxyear(
    IN ye int, in bank_name varchar(50), OUT money float, OUT num int)
BEGIN

    select sum(Balance)
    from account,saving_account
    where year(Opening_Date) = ye
      and account.A_ID = saving_account.A_ID
      and B_Name = bank_name
    into money;

    select count(C_ID)
    from checking
    where A_Type = 1
    and B_Name = bank_name
    and A_ID in (   select A_ID
                    from account
                    where year(Opening_Date) = ye)
    into num;

    if money is null then
        set money = 0;
    end if;

    if num is null then
        set num = 0;
    end if;
END //
Delimiter ;
```

统计的实现大体一致，这里只展示统计年份储蓄总金额和总用户。

app：

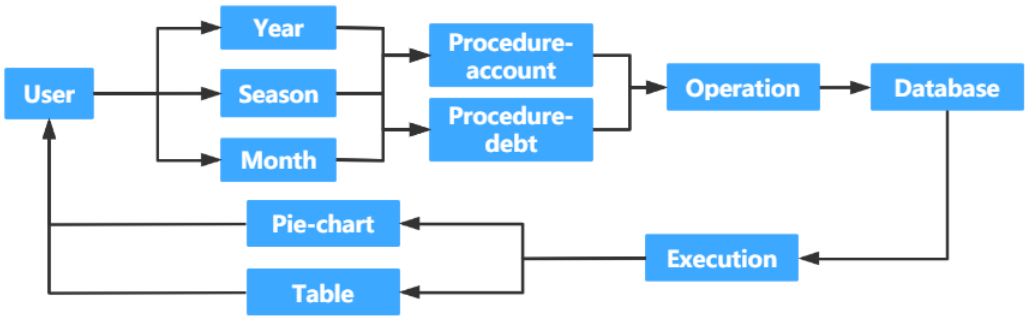
```
@app.route('/statistics', methods=['GET', 'POST'])
def statistics():
    bank_list = [['蜀山路支行', 100, 100, 100, 100], ['肥西路支行
', 100, 100, 100, 100],
```

```
                ['朝阳区支行', 100, 100, 100, 100], ['天府路支行
', 100, 100, 100, 100]]
    bank_all = db.session.query(Bank).all()
    new_bank_list = []
    for i in bank_all:
        new_bank_list.append([i.B_Name])
    if request.method == 'GET':
        return render_template('statistics.html', bank_list=bank_list)
    else:
        # i[0]: 支行名
        # i[1]: 储蓄总额
        # i[2]: 贷款总额
        # i[3]: 储蓄总人
        # i[4]: 贷款总人
        if request.form.get('type') == 'year':
            year = request.form.get('year1')
            if int(year) > 2021 or int(year) < 1990:
                error_title = '输入错误'
                error_message = '年份输入错误'
                return render_template('404.html', error_title=error_title, error_me
ssage=error_message)

            for i in new_bank_list:
                cx = cursor.callproc('cxyear', (int(year), i[0], None, None))
                dk = cursor.callproc('dkyear', (int(year), i[0], None, None))
                i.append(cx[2])  # i[1]
                i.append(dk[2])  # i[2]
                i.append(cx[3])  # i[3]
                i.append(dk[3])  # i[4]

        elif request.form.get('type') == 'season':
            '''''...'''

        elif request.form.get('type') == 'month':
            '''''...'''

    bank_list = new_bank_list
    return render_template('statistics.html', bank_list=bank_list)
```

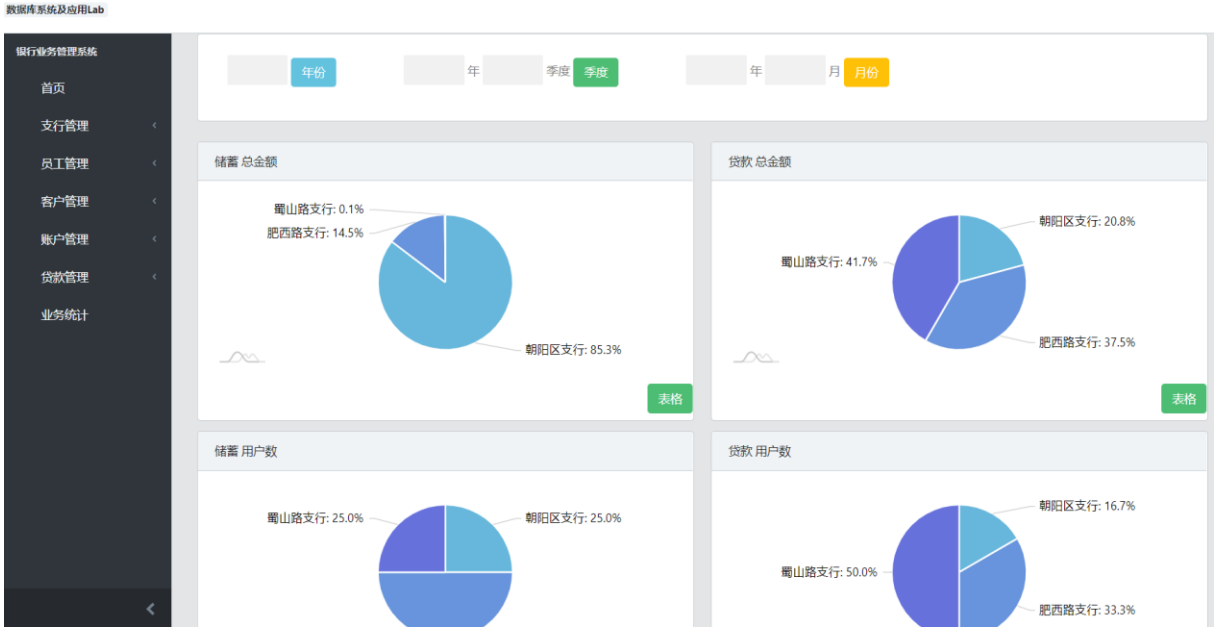
输入：年份、季度、月份。

输出：饼图、表格展示。

流程图：


User
Year
Season
Month
Procedure-account
Procedure-debt
Operation
Database
Pie-chart
Table
Execution

页面设计：

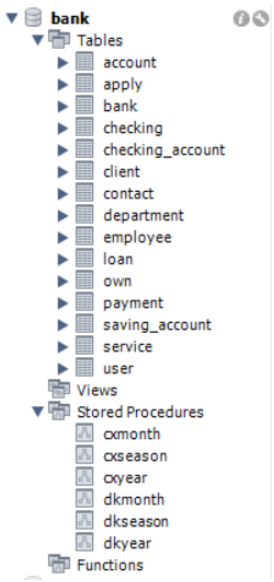

数据库系统及应用Lab
银行业务管理系统
首页
支行管理
员工管理
客户管理
账户管理
贷款管理
业务统计
年份
季度
月份
储蓄 总金额
蜀山路支行: 0.1%
肥西路支行: 14.5%
朝阳区支行: 85.3%
表格
贷款 总金额
朝阳区支行: 20.8%
蜀山路支行: 41.7%
肥西路支行: 37.5%
储蓄 用户数
蜀山路支行: 25.0%
朝阳区支行: 25.0%
贷款 用户数
朝阳区支行: 16.7%
蜀山路支行: 50.0%
肥西路支行: 33.3%


数据库系统及应用Lab
银行业务管理系统
首页
支行管理
员工管理
客户管理
账户管理
贷款管理
业务统计
年份
季度
月份
储蓄 总金额
蜀山路支行: 0.1%
肥西路支行: 14.5%
朝阳区支行: 85.3% (58,800)
朝阳区支行: 85.3%
表格
贷款 总金额
朝阳区支行: 20.8%
蜀山路支行: 41.7%
肥西路支行: 37.5%
储蓄 用户数
蜀山路支行: 25.0%
朝阳区支行: 25.0%
贷款 用户数
蜀山路支行: 50.0%
贷款总金额表格
支行名
总贷款
朝阳区支行
500.0
肥西路支行
900.0
蜀山路支行
1000.0
关闭

4 实现与测试

4.1 实现结果

系统：



支行：

支行信息

2021 6

支行号	支行名	支行资产	所在城市	操作
3	朝阳区支行	999999.0	北京	更新 删除
1	肥西路支行	100000.0	合肥	更新 删除
2	蜀山路支行	123456.0	合肥	更新 删除

B_ID	B_Name	City	Assets
3	朝阳区支行	北京	999999
1	肥西路支行	合肥	100000
2	蜀山路支行	合肥	123456
NULL	NULL	NULL	NULL

员工：

员工管理

2021 6

日期格式：****-**-**

员工ID	员工姓名	员工电话	员工住址	雇佣日期	所在支行	部门号	部门名称	部门类型	部门经理ID	操作
E001	小明	123456	合肥	2000-06-08	肥西路支行	D01	销售	A	E001	更新 删除
E002	小红	456789	南京	2000-01-01	蜀山路支行	D02	公关	B	E002	更新 删除
E003	小光	777888	北京	2012-09-09	蜀山路支行	D02	公关	B	E002	更新 删除

部门管理

2021 6

部门号	部门名称	部门类型	部门经理ID	操作
D01	销售	A	E001	更新 删除
D02	公关	B	E002	更新 删除

E_ID	E_Name	B_Name	D_ID	E_Tel	E_Addr	Work_Date
E001	小明	肥西路支行	D01	123456	合肥	2000-06-08
E002	小红	蜀山路支行	D02	456789	南京	2000-01-01
E003	小光	蜀山路支行	D02	777888	北京	2012-09-09
NULL	NULL	NULL	NULL	NULL	NULL	NULL

D_ID	D_Name	D_Type	Manager_ID
D01	销售	A	E001
D02	公关	B	E002
NULL	NULL	NULL	NULL

客户：

客户管理

2021 6

客户ID	客户姓名	客户电话	客户住址	联系人姓名	联系人电话	联系人邮箱	关系	操作
C001	李华	1111	中科大	李子	2222	123@qq.com	父子	更新 删除
C002	王建国	9999	合工大	王中华	8888	456@qq.com	兄弟	更新 删除
C003	张三	5678	苏州	张思	51848	4545@qq.com	叔侄	更新 删除
C004	王宇波	17717	南京	王湘峰	555	12w3@qq.com	父子	更新 删除

服务关系

2021 6

客户ID	员工ID	服务类型	操作
C001	E001	贷款	更新 删除
C002	E001	银行	更新 删除
C003	E002	贷款	更新 删除

C_ID	C_Name	C_Tel	C_Addr
C001	李华	1111	中科大
C002	王建国	9999	合工大
C003	张三	5678	苏州
C004	王宇波	17717	南京
NULL	NULL	NULL	NULL

C_ID	E_ID	S_Type
C001	E001	贷款
C002	E001	银行
C003	E002	贷款
NULL	NULL	NULL

账户：

储蓄账户管理

2021 6

日期格式：****-**-**

账户号	客户ID	客户姓名	开户支行	开户时间	账户余额	最近访问时间	利率	货币类型	操作
A001	C001	李华	蜀山路支行	2000-09-01	588.0	2021-06-19	0.21	人民币	更新 删除
A003	C003	张三	朝阳区支行	2020-09-07	58800.0	2021-06-20	0.21	美元	更新 删除
A004	C002	王建国	蜀山路支行	2020-07-01	100.0	2021-06-20	3.5	人民币	更新 删除
A005	C001	李华	肥西路支行	2020-10-01	5000.0	2021-06-20	2.14	人民币	更新 删除
A007	C004	王宇波	肥西路支行	2020-09-07	5000.0	2021-06-21	3.5	人民币	更新 删除

支票账户管理

2021 6

日期格式：****-**-**

账户号	客户ID	客户姓名	开户支行	开户时间	账户余额	最近访问时间	透支额度	操作
A002	C001	李华	蜀山路支行	2021-05-07	1001.0	2021-07-05	500.0	更新 删除
A002	C002	王建国	蜀山路支行	2021-05-07	1001.0	2021-06-19	500.0	更新 删除
A006	C003	张三	肥西路支行	2020-10-01	58800.0	2021-06-21	500.0	更新 删除

A_ID	B_Name	Balance	Opening_Date
A001	蜀山路支行	588	2000-09-01
A002	蜀山路支行	1001	2021-05-07
A003	朝阳区支行	58800	2020-09-07
A004	蜀山路支行	100	2020-07-01
A005	肥西路支行	5000	2020-10-01
A006	肥西路支行	58800	2020-10-01
A007	肥西路支行	5000	2020-09-07
NULL	NULL	NULL	NULL

A_ID	Interest_Rate	Currency_Type
A001	0.21	人民币
A003	0.21	美元
A004	3.5	人民币
A005	2.14	人民币
A007	3.5	人民币
NULL	NULL	NULL

A_ID	Overdraft
A002	500
A006	500
NULL	NULL

贷款：

贷款管理

2021 6

日期格式：****-**-**

贷款状态： 0 - 未发放; 1: 发放中; 2: 发放完毕; 3: 异常;

贷款号	发放支行	贷款金额	贷款状态	操作
L001	蜀山路支行	1000.0	2	删除
L002	肥西路支行	500.0	1	删除
L003	肥西路支行	2500.0	1	删除
L004	朝阳区支行	100000.0	1	删除
L005	蜀山路支行	200.0	2	删除

支付记录

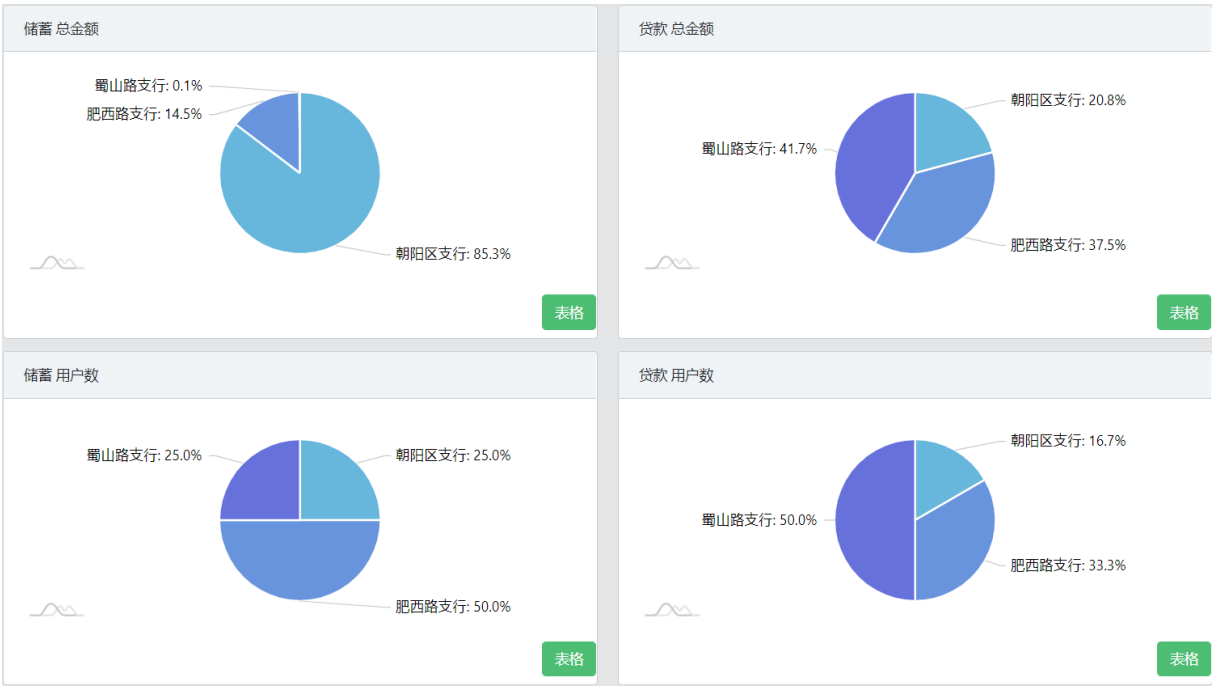
2021 6

支付号	贷款号	客户ID	支付金额	支付日期	操作
P001	L001	C001	500.0	2020-01-01	更新
P005	L002	C001	100.0	2020-01-01	更新
P007	L004	C001	500.0	2020-05-07	更新
P002	L001	C002	200.0	2020-01-05	更新
P003	L001	C002	300.0	2020-09-09	更新
P006	L003	C002	800.0	2020-09-07	更新
P004	L002	C003	50.0	2021-01-01	更新
P008	L005	C003	200.0	2000-01-01	更新

L_ID	B_Name	L_Amount	L_Status	P_already
L001	蜀山路支行	1000	2	1000
L002	肥西路支行	500	1	150
L003	肥西路支行	2500	1	800
L004	朝阳区支行	100000	1	500
L005	蜀山路支行	200	2	200
NULL	NULL	NULL	NULL	NULL

C_ID	L_ID	P_ID	P_Amount	Pay_Date
C001	L001	P001	500	2020-01-01
C001	L002	P005	100	2020-01-01
C001	L004	P007	500	2020-05-07
C002	L001	P002	200	2020-01-05
C002	L001	P003	300	2020-09-09
C002	L003	P006	800	2020-09-07
C003	L002	P004	50	2021-01-01
C003	L005	P008	200	2000-01-01
NULL	NULL	NULL	NULL	NULL

统计：



4.2 测试结果

4.2.1 用户模块

输入未注册的用户名，登录失败：

注册后，登录成功：

4.2.2 支行模块

添加：

删除：

修改：

查询：

4.2.3 员工模块

只展示员工部分，部门部分不赘述。

添加：

员工管理

2021 6

日期格式：****-**-**

员工ID	员工姓名	员工电话	员工住址	雇佣日期	所在支行	部门号	部门名称	部门类型	部门经理ID	操作
E001	小明	123456	合肥	2000-06-08	肥西路支行	D01	销售	A	E001	更新 删除
E002	小红	456789	南京	2000-01-01	蜀山路支行	D02	公关	B	E002	更新 删除
E003	小光	777888	北京	2012-09-09	蜀山路支行	D02	公关	B	E002	更新 删除
E004	小王	12121	中科大	2000-01-09	肥西路支行	D02	公关	B	E002	更新 删除

删除：

员工管理

2021 6

日期格式：****-**-**

员工ID	员工姓名	员工电话	员工住址	雇佣日期	所在支行	部门号	部门名称	部门类型	部门经理ID	操作
E001	小明	123456	合肥	2000-06-08	肥西路支行	D01	销售	A	E001	更新 删除
E002	小红	456789	南京	2000-01-01	蜀山路支行	D02	公关	B	E002	更新 删除
E003	小光	777888	北京	2012-09-09	蜀山路支行	D02	公关	B	E002	更新 删除

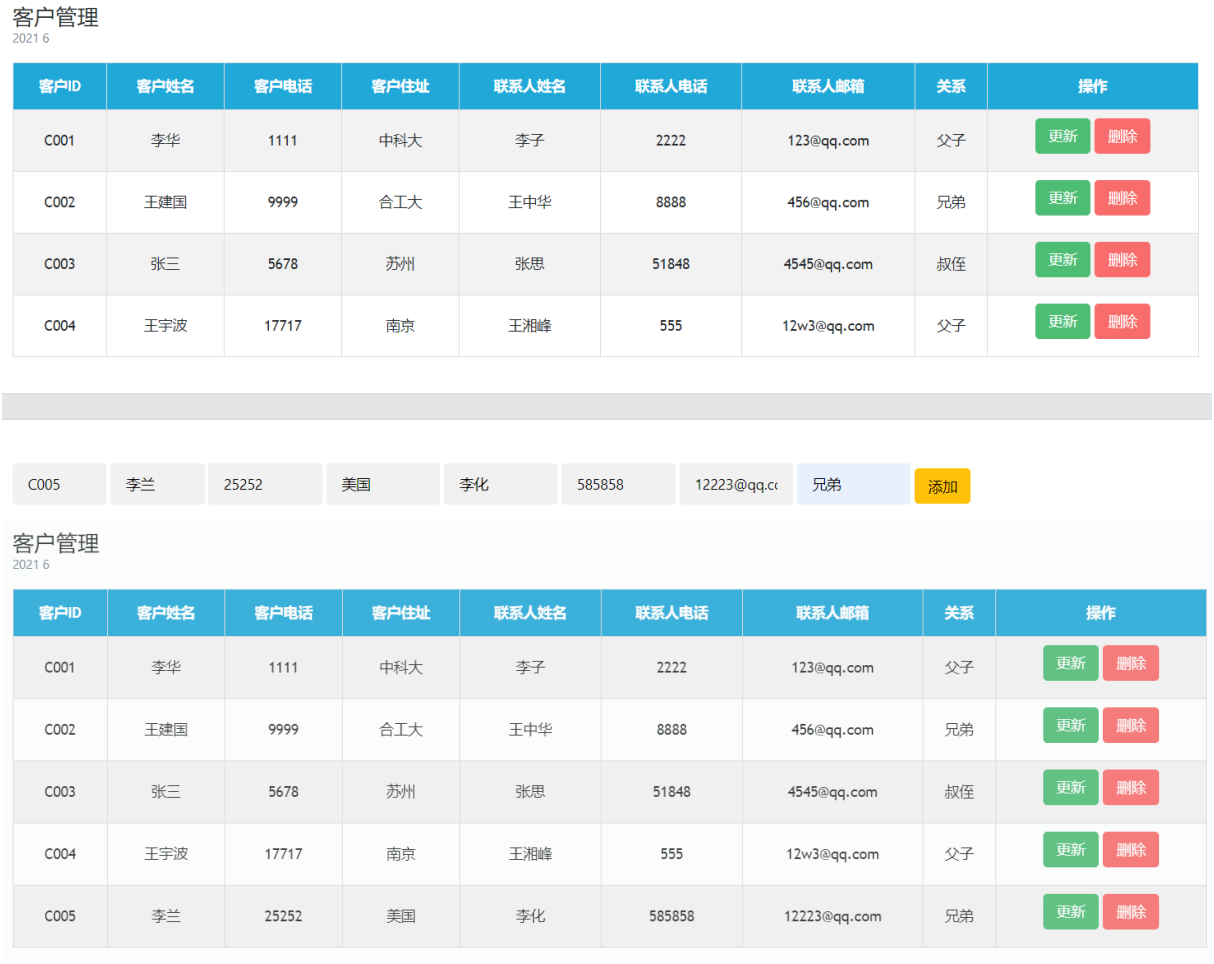
修改：

查询：

4.2.4 客户模块

只展示客户部分，服务部分不赘述。

添加：



删除：

修改：

查询：

4.2.5 账户模块

只展示支票部分，储蓄部分不赘述。

添加：

支票账户管理

2021 6

日期格式：****-**-**

账户号	客户ID	客户姓名	开户支行	开户时间	账户余额	最近访问时间	透支额度	操作
A002	C001	李华	蜀山路支行	2021-05-07	1001.0	2021-07-05	500.0	更新 删除
A002	C002	王建国	蜀山路支行	2021-05-07	1001.0	2021-06-19	500.0	更新 删除
A006	C003	张三	肥西路支行	2020-10-01	58800.0	2021-06-21	500.0	更新 删除

支票账户

A008	C004	蜀山路支行	2020-9-7	58800	500	添加

支票账户管理

2021 6

日期格式：****-**-**

账户号	客户ID	客户姓名	开户支行	开户时间	账户余额	最近访问时间	透支额度	操作
A002	C001	李华	蜀山路支行	2021-05-07	1001.0	2021-07-05	500.0	更新 删除
A002	C002	王建国	蜀山路支行	2021-05-07	1001.0	2021-06-19	500.0	更新 删除
A006	C003	张三	肥西路支行	2020-10-01	58800.0	2021-06-21	500.0	更新 删除
A008	C004	王宇波	蜀山路支行	2020-09-07	58800.0	2021-06-29	500.0	更新 删除

删除：

A008	C004	王宇波	蜀山路支行	2020-09-07	58800.0	2021-06-29	500.0	更新 删除

账户

户号	客户ID	开户支行	开户时间	账户余额	利率	货币类型	添加

确认删除？

确认

取消

支票账户管理

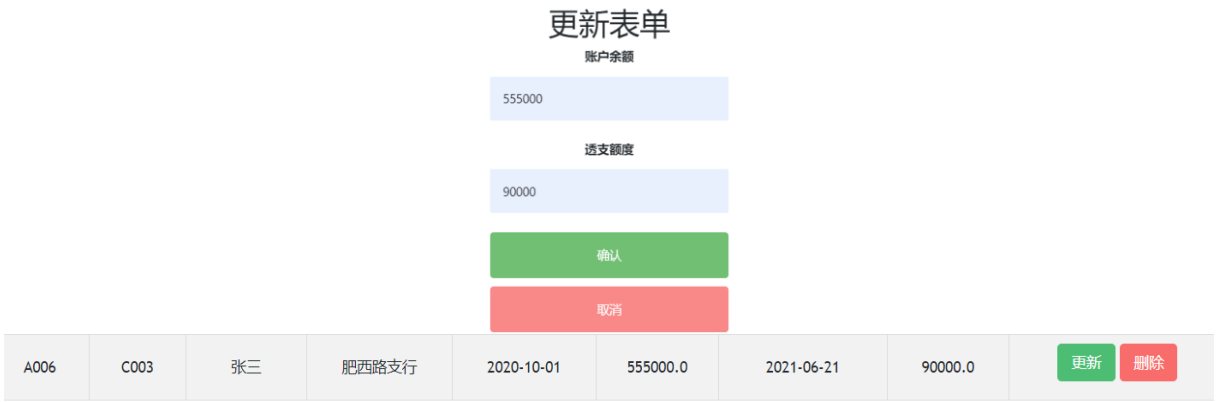
2021 6

日期格式：****-**-**

账户号	客户ID	客户姓名	开户支行	开户时间	账户余额	最近访问时间	透支额度	操作
A002	C001	李华	蜀山路支行	2021-05-07	1001.0	2021-07-05	500.0	更新 删除
A002	C002	王建国	蜀山路支行	2021-05-07	1001.0	2021-06-19	500.0	更新 删除
A006	C003	张三	肥西路支行	2020-10-01	58800.0	2021-06-21	500.0	更新 删除

修改：

A006	C003	张三	肥西路支行	2020-10-01	58800.0	2021-06-21	500.0	更新 删除



查询：

4.2.6 贷款模块

只展示贷款部分，发放部分不赘述。

添加：

贷款管理

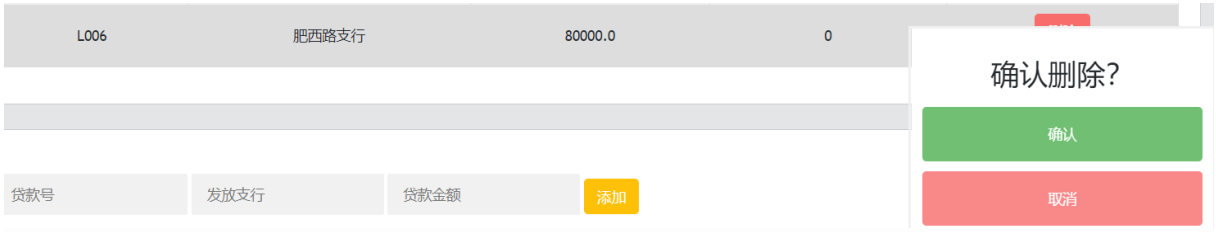
2021 6

日期格式：****-**-**

贷款状态： 0 - 未发放; 1: 发放中; 2: 发放完毕; 3: 异常;

贷款号	发放支行	贷款金额	贷款状态	操作
L001	蜀山路支行	1000.0	2	删除
L002	肥西路支行	500.0	1	删除
L003	肥西路支行	2500.0	1	删除
L004	朝阳区支行	100000.0	1	删除
L005	蜀山路支行	200.0	2	删除
L006	肥西路支行	80000.0	0	删除

删除：



贷款管理

2021 6

日期格式：****-**-**

贷款状态： 0 - 未发放; 1: 发放中; 2: 发放完毕; 3: 异常;

贷款号	发放支行	贷款金额	贷款状态	操作
L001	蜀山路支行	1000.0	2	删除
L002	肥西路支行	500.0	1	删除
L003	肥西路支行	2500.0	1	删除
L004	朝阳区支行	100000.0	1	删除
L005	蜀山路支行	200.0	2	删除

查询：

贷款管理

2021 6

日期格式：****-**-**

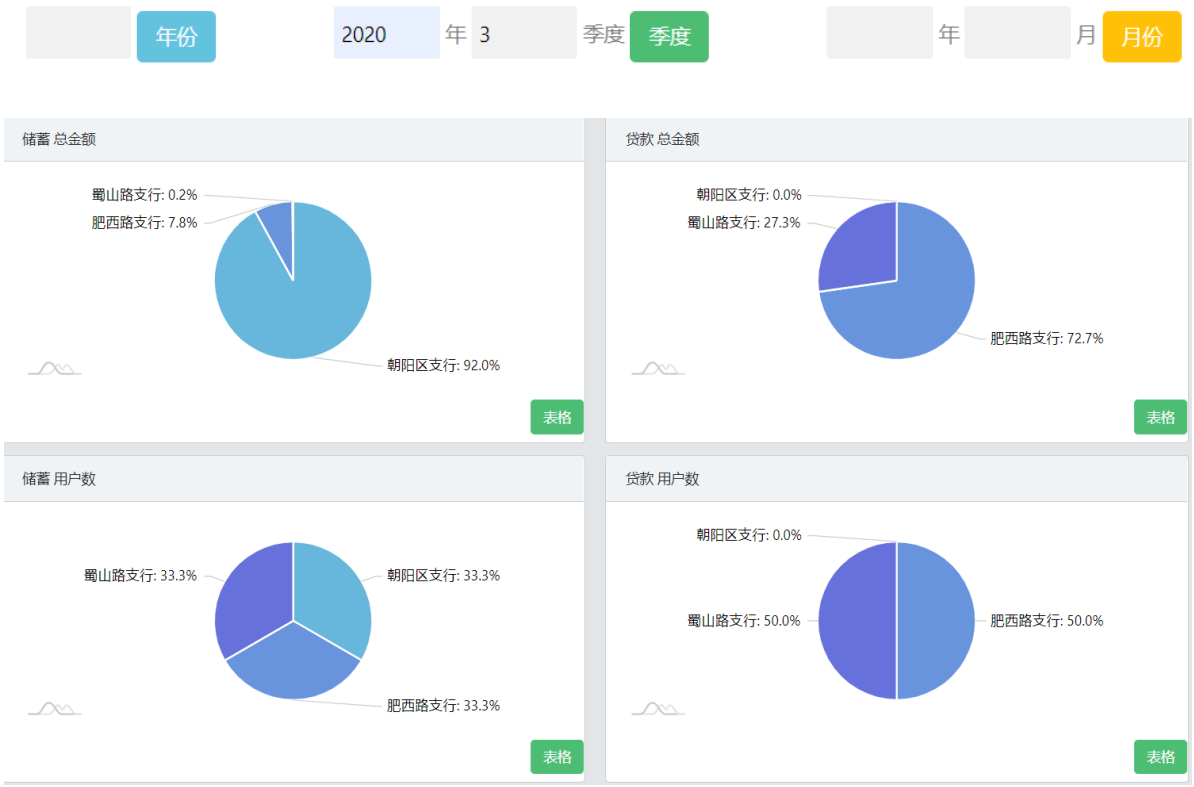
贷款状态： 0 - 未发放; 1: 发放中; 2: 发放完毕; 3: 异常;

贷款号	发放支行	贷款金额	贷款状态	操作
L002	肥西路支行	500.0	1	删除

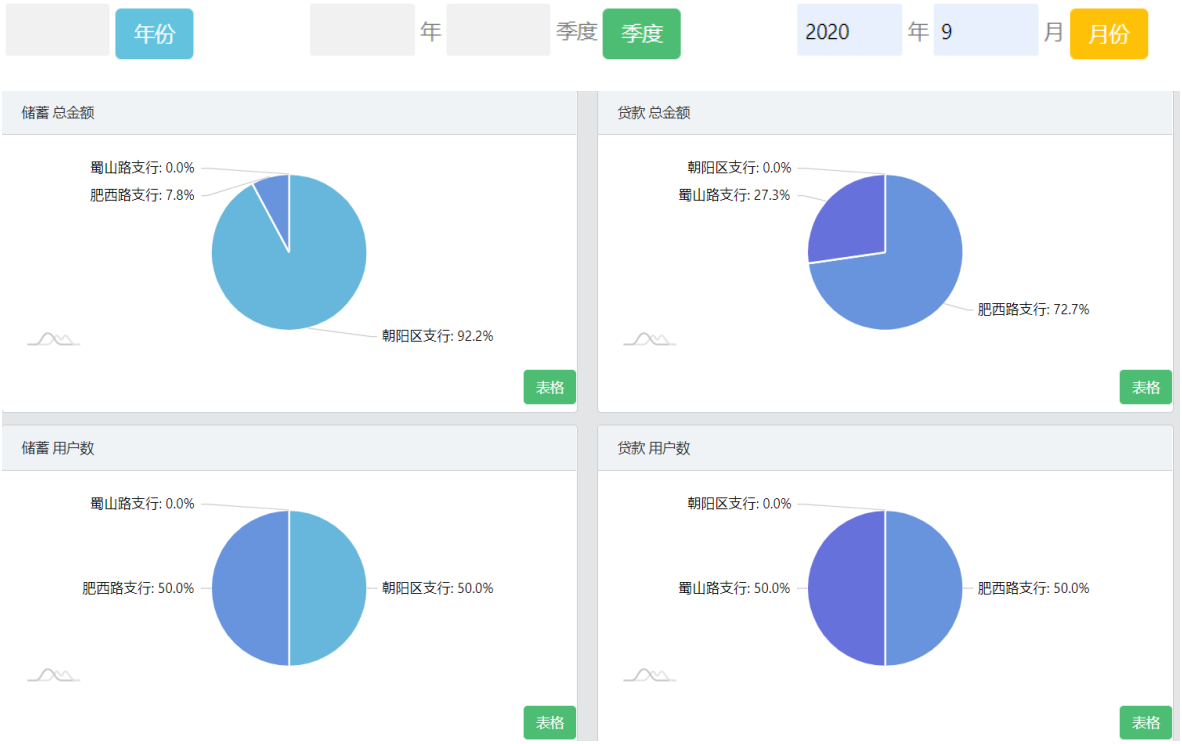
4.2.7 统计模块

2020 年业务统计：

2020 年第 3 季度业务统计：



2020 年 9 月业务统计：



4.3 实现中的难点问题及解决

- **没有学过 web 设计开发，对网页设计无从下手。**
  解决：浏览大量简单的样例与视频学习，掌握基本的 html 语言，学会基本的 web 设计编写，并通过学习 DW，最终呈现了如上效果的网页界面。
- **界面设计需要花大量的时间。**
  解决：使用了 Bootstrap 模板 CoreUI。
- **没有接触过软件工程。**
  解决：浏览 CSDN 了解相关知识。
- **Python 与网页的连接产生大量 bug。**
  解决：耐心比对 type 的 name 与读取元素的 name 是否可以在 python 中对应，或者善用 print 找到 bug 所在。
- **对于表的属性名，在 html、python 的 model、MYSQL 中的对应关系容易出错。**
  解决：按着一定的规律为变量命名。
- **在 web 端使用系统，出错时无法得知错误原因。**
  解决：做错误处理，在错误页面(404.html)输出常规错误原因，或者将后端报错信息显示在网页上。
- **关系表冗余。**
  解决：适当改进实体，增加一些属性。
- **统计功能实现困难。**

解决：在 MYSQL 写存储过程，python 读取输出结果。

- **统计结果以表格实现不够美观。**
  解决：使用 Amcharts 下的饼图模板，并将表格设为弹窗。

5 总结与讨论

通过开发一个简易但完整的银行管理系统，我收获了许多。无论是软件工程基础、Web 开发设计、前端后端的连接还是资料的查找、工具的选取对我均是巨大的挑战。但是不断地学习和尝试，最终完成了这个实验，使自己的学习能力、思考能力、逻辑能力、编程能力获得了较大的提升。总结收获有:

- 开发需要遵循一定的自顶向下或自底向上的流程;
- 学会实行环境控制;
- 从已有样例学习可以较快掌握编程知识;
- 每实现一个小功能可以立即 debug，防止代码过多难以定位;
- 变量名要符合一定规律;
- 适当使用已有模板减少没有必要的工作;
- 实战使用 B/S 框架，有了一定的经验;
- 学会了 python+Flask 的开发思路和开发框架;
- 学会了使用 mysql-connector、pymysql 等便捷的数据库相关库;
- 对于在某一端难以实现的功能，可以考虑在其他端实现;
- 一定要注重错误处理;
- 写实验文档或者适当的注释记录自己的思路。