中国科学技术大学计算机学院
《数据库系统实验报告》

实验题目：银行管理系统

学生姓名：常文正

学生学号：PB21111706

完成时间：2024年5月25日

1 需求分析

1.1 应用场景

银行管理系统是一个典型的数据库应用，涉及银行支行信息、客户信息、账户信息、贷款信息、银行部门信息、员工信息等实体，提供开户、销户、存款、取款、转账、贷款、查询等功能。

1.2 数据需求：

1. **银行支行信息**：
   - 包括支行名称、地址、联系方式等。
2. **客户信息**：
   - 包括客户姓名、身份证号、联系方式、地址、账户数量。
3. **账户信息**：
   - 包括账户ID，所属支行，所属用户，余额，开户日期。
4. **账单信息**：
   - 包括账单ID，所属账户，交易金额，交易类型，交易时间。
5. **贷款信息**：
   - 包括贷款ID、客户ID、贷款金额、贷款期限、审批状态、还款状态、未还金额。
6. **银行部门信息**：
   - 包括部门ID，部门名称、部门主管、部门联系方式等。
7. **员工信息**：
   - 包括员工ID，所属部门，员工姓名，联系方式、照片，地址。

1.3 功能需求：

1.3.1 客户端功能：

1. **开户**：
   - 客户填写个人信息和开户信息，系统生成账户号码。
2. **销户**：
   - 客户申请销户，系统验证身份并处理相关信息。
3. **存款**：
   - 客户输入存款金额，系统更新账户余额。
4. **取款**：
   - 客户输入取款金额，系统验证余额并更新账户余额。
5. **转账**：

- 客户输入转账金额和目标账户号码，系统验证并更新两个账户余额。

6. **贷款**：
   - 客户填写贷款申请表单，系统进行贷款审核并记录贷款信息。
7. **查账**：
   - 客户查询账户余额、交易记录、贷款信息等。

1.3.2 银行端功能：

1. **客户信息管理**：
   - 员工查看、修改客户信息，包括个人信息和账户信息。
2. **贷款审核**：
   - 员工审核客户的贷款申请，记录审核结果。
3. **部门管理**：
   - 员工查看、管理银行部门信息，包括部门名称、主管、联系方式等。
4. **员工管理**：
   - 员工查看、管理员工信息，包括姓名、部门、联系方式等。

2 总体设计

2.1 系统模块结构

由于本次实验采用Django框架进行开发，因此后端采用MVC模式进行设计，前端采用HTML、CSS、JavaScript等技术进行设计，系统模块结构如下：

2.1.1 后端：

通过建立了多个应用，每个应用负责一个模块的功能，如accounts应用负责账户管理，bills应用负责账单管理，branch应用负责支行管理，department应用负责部门管理，frontend应用负责前端页面管理，loans应用负责贷款管理，staffs应用负责员工管理，users应用负责用户管理。

每个应用中包含了模型、视图、路由、模板等文件，通过Django的ORM框架实现数据库操作，通过Django的template引擎实现前端页面渲染。

其中，模型文件定义了数据库模型，视图文件定义了视图处理函数，路由文件定义了路由映射，模板文件定义了前端页面模板。

2.1.2 前端：

前端采用HTML、CSS、JavaScript等技术，使用Bootstrap框架进行页面布局，实现用户界面。

根据登录用户的不同，显示不同的页面，如客户登录后显示客户页面，银行工作人员登录后显示支行页面。

其中客户页面包括开户、销户、存款、取款、转账、贷款、查账等功能，支行页面包括客户信息管理、贷款审核、部门管理、员工管理等功能。

前端页面通过Django的模板引擎渲染，通过Django的路由映射实现页面跳转，主要部分由frontend应用负责，其中包括base.html、index.html、error.html等页面，分别用于页面基础模板、首页模板、错误页面模板。在其他每个应用中也包含了相应的模板文件，如accounts应用中包含了accounts.html、create_account.html、transfer.html等页面模板，这些页面大多都是通过继承base.html模板实现的，在每个页面中包含了相应的表单、表格、按钮等元素，通过JavaScript实现页面交互功能。

2.2 系统工作流程

用户注册与登录

- 用户通过前端页面进行注册，提交个人信息。
- 后端接收注册请求，验证信息的合法性，创建新用户。
- 用户通过前端页面登录，后端验证用户名和密码，成功后进入系统主页。

账户管理

- 用户登录后，可以在账户管理模块查看和管理个人账户信息。
- 后端从数据库中获取用户账户信息，通过视图传递给前端页面进行展示。
- 用户可以在前端页面进行账户信息的修改，前端将修改请求发送到后端，后端处理后更新数据库。

账单管理

- 用户可以查看账户的历史账单信息。
- 后端从数据库中获取账单信息，通过视图传递给前端页面进行展示。

支行和部门管理

- 管理员可以在支行和部门管理模块查看和管理各个支行和部门的信息。
- 后端从数据库中获取支行和部门信息，通过视图传递给前端页面进行展示。
- 管理员可以在前端页面修改或删除部门信息，前端将请求发送到后端，后端处理后更新数据库。

员工管理

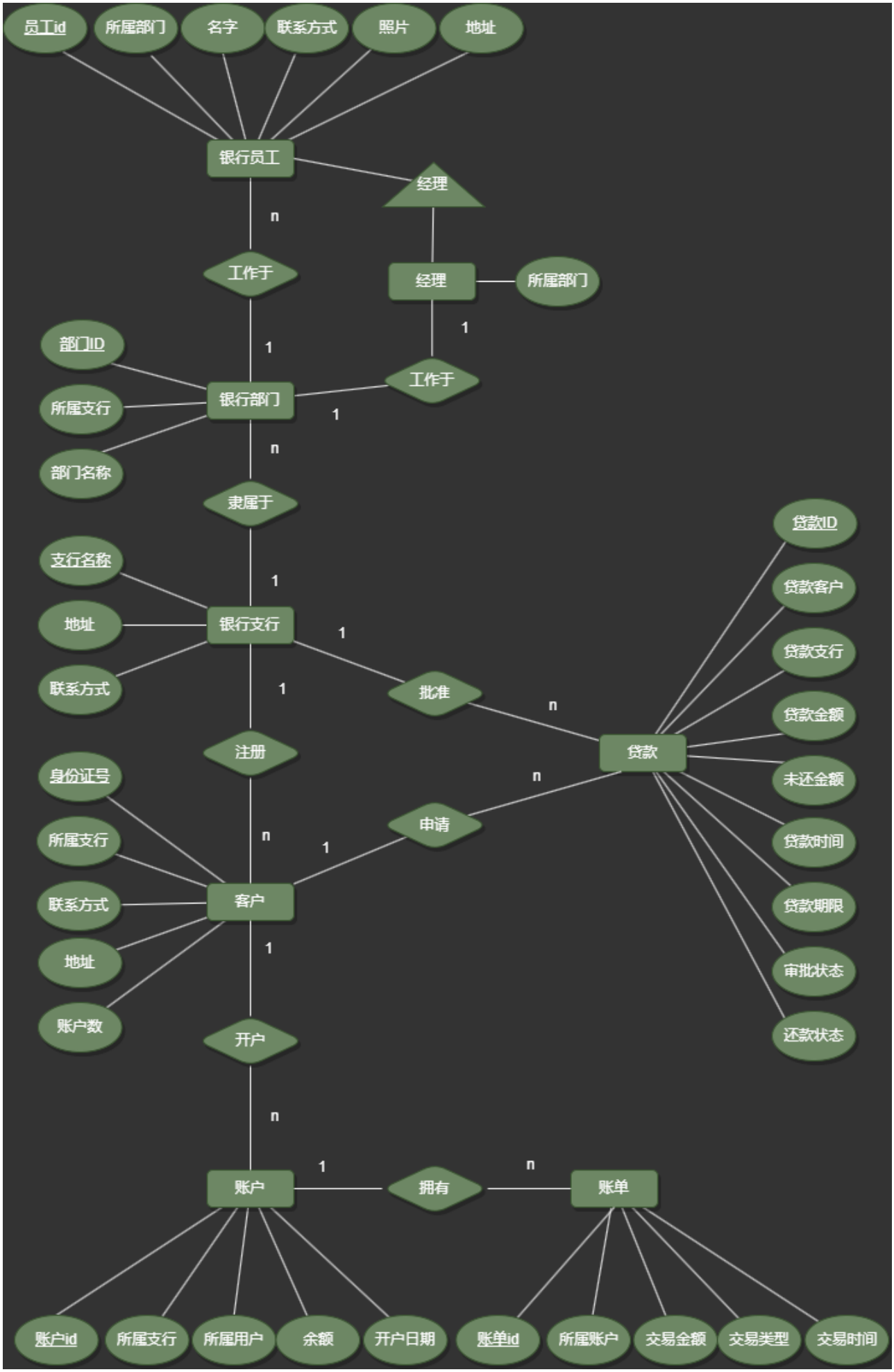
- 管理员可以在员工管理模块查看和管理员工信息。
- 后端从数据库中获取员工信息，通过视图传递给前端页面进行展示。
- 管理员可以在前端页面添加、修改或删除员工信息，前端将请求发送到后端，后端处理后更新数据库。

贷款管理

- 用户可以申请贷款，查看贷款状态。
- 后端处理贷款申请请求，验证信息的合法性后创建贷款记录，更新数据库。
- 支行工作人员可以在贷款管理模块查看贷款申请，审批贷款。
- 用户可以查看贷款的还款记录和未还清余额等信息。

2.3 数据库设计

2.3.1 ER图：


员工id
所属部门
名字
联系方式
照片
地址
银行员工
经理
n
工作于
经理
所属部门
1
1
部门ID
所属支行
部门名称
银行部门
工作于
1
n
隶属于
支行名称
地址
联系方式
银行支行
1
1
批准
n
注册
贷款ID
贷款客户
贷款支行
贷款金额
未还金额
贷款时间
贷款期限
审批状态
还款状态
贷款
身份证号
所属支行
联系方式
地址
账户数
客户
申请
n
1
n
1
开户
n
1
拥有
n
账户
账单
账户id
所属支行
所属用户
余额
开户日期
账单id
所属账户
交易金额
交易类型
交易时间

2.3.2 模式分解：

关系数据库模式如下：（表名：主键名 属性1 属性2 ...）

- 银行支行：银行名称 联系电话 所在城市
- 银行部门：部门号 银行名称 部门名称
- 员工：工号 部门号 姓名 电话，家庭住址 员工照片
- 经理：工号 部门号
- 客户：身份证号码 银行名称 姓名 电话 家庭住址 拥有账户数量
- 账户：账户号 身份证号码 银行名称 余额 开户日期
- 账单：账单号 账户号 发生时间 修改净值 余额 账单类型 交易备注
- 贷款：贷款号 身份证号码 银行名称 贷款总额 未还清余额 贷款日期 还款期限 贷款状态

可以明显地观察到，每个关系模式中，所有属性都是原子的，没有多值属性，也没有派生属性，所有属性都是完全依赖于主键，也没有传递依赖，其所有的函数依赖集如下：

- 函数依赖集 $_{\{银行支行\}}$ = { 银行名称 → 联系电话, 银行名称 → 所在城市 }
- 函数依赖集 $_{\{部门\}}$ = { 部门号 → 银行名称, 部门号 → 部门名称 }
- 函数依赖集 $_{\{员工\}}$ = { 工号 → 部门号, 工号 → 姓名, 工号 → 电话, 工号 → 家庭住址, 工号 → 员工照片 }
- 函数依赖集 $_{\{客户\}}$ = { 身份证号码 → 银行名称, 身份证号码 → 姓名, 身份证号码 → 电话, 身份证号码 → 家庭住址, 身份证号码 → 拥有账户数量 }
- 函数依赖集 $_{\{账户\}}$ = { 账户号 → 身份证号码, 账户号 → 银行名称, 账户号 → 余额, 账户号 → 开户日期 }
- 函数依赖集 $_{\{账单\}}$ = { 账单号 → 账户号, 账单号 → 发生时间, 账单号 → 修改增值, 账单号 → 余额, 账单号 → 账单类型, 账单号 - → 交易金额 }
- 函数依赖集 $_{\{贷款\}}$ = { 贷款号 → 身份证号码, 贷款号 → 银行名称, 贷款号 → 贷款总额, 贷款号 → 未还清余额, 贷款号 → 贷款日期, 贷款号 → 还款期限, 贷款号 → 贷款状态 }

该函数依赖集合中，所有的函数依赖都是完全依赖于主键的，没有部分依赖，也没有传递依赖，因此这个关系模式是符合第三范式的。因此不再需要进行分解。

2.3.3 存储过程、触发器、函数设计思路

由于本次实现采用的是Django框架，因此不需要设计存储过程、触发器、函数等数据库操作，而是通过Django的ORM框架来实现数据库操作，不过可以通过Django的信号机制来实现类似的功能，接下来在每个模块的设计中会详细介绍。

3 核心代码解析

3.1 仓库地址

https://github.com/AmberHeart/Database_Lab

3.2 目录

如下显示的文件结构中已经去除部分不重要的文件夹，比如migrations数据库迁移文件与pycache等缓存文件

```
D:.
|  manage.py ---- 项目管理命令文件
|
├─accounts ---- 应用目录，用于管理账户
|  |  admin.py ---- 管理员设置文件
|  |  apps.py ---- 应用配置文件
|  |  forms.py ---- 表单定义文件
|  |  models.py ---- 数据模型文件
|  |  tests.py ---- 测试文件
|  |  urls.py ---- 路由配置文件
|  |  views.py ---- 视图处理文件
|  |  __init__.py ---- 应用初始化文件
|  |
|  └─templates ---- 模板目录
|      └─accounts
|              accounts.html ---- 账户列表页面模板
|              create_account.html ---- 创建账户页面模板
|              transfer.html ---- 转账页面模板
|
├─bank ---- 项目设置目录
|      asgi.py ---- ASGI 配置文件
|      settings.py ---- 项目设置文件
|      urls.py ---- 项目路由配置文件
|      wsgi.py ---- WSGI 配置文件
|      __init__.py ---- 项目初始化文件
|
├─bills ---- 应用目录，用于管理账单
|  |  admin.py ---- 管理员设置文件
|  |  apps.py ---- 应用配置文件
|  |  forms.py ---- 表单定义文件
|  |  models.py ---- 数据模型文件
|  |  tests.py ---- 测试文件
|  |  urls.py ---- 路由配置文件
|  |  views.py ---- 视图处理文件
|  |  __init__.py ---- 应用初始化文件
|  |
|  └─templates ---- 模板目录
|      └─bills
|              bills.html ---- 账单列表页面模板
|              create_bill.html ---- 创建账单页面模板
|
├─branch ---- 应用目录，用于管理分行
|  |  admin.py ---- 管理员设置文件
```

```
44  │   │  apps.py ---- 应用配置文件
45  │   │  models.py ---- 数据模型文件
46  │   │  tests.py ---- 测试文件
47  │   │  urls.py ---- 路由配置文件
48  │   │  views.py ---- 视图处理文件
49  │   │  __init__.py ---- 应用初始化文件
50  │   │
51  │   └─templates ---- 模板目录
52  │      └─branches
53  │              branches.html ---- 分行列表页面模板
54  │
55  ├─department ---- 应用目录，用于管理部门
56  │   │  admin.py ---- 管理员设置文件
57  │   │  apps.py ---- 应用配置文件
58  │   │  models.py ---- 数据模型文件
59  │   │  tests.py ---- 测试文件
60  │   │  urls.py ---- 路由配置文件
61  │   │  views.py ---- 视图处理文件
62  │   │  __init__.py ---- 应用初始化文件
63  │   │
64  │   └─templates ---- 模板目录
65  │      └─departments
66  │              departments.html ---- 部门列表页面模板
67  │              staffs.html ---- 员工列表页面模板
68  │
69  ├─frontend ---- 应用目录，用于管理前端
70  │   │  admin.py ---- 管理员设置文件
71  │   │  apps.py ---- 应用配置文件
72  │   │  models.py ---- 数据模型文件
73  │   │  tests.py ---- 测试文件
74  │   │  urls.py ---- 路由配置文件
75  │   │  views.py ---- 视图处理文件
76  │   │  __init__.py ---- 应用初始化文件
77  │   │
78  │   └─templates ---- 模板目录
79  │      └─frontend
80  │              base.html ---- 基础页面模板
81  │              error.html ---- 错误页面模板
82  │              index.html ---- 首页模板
83  │
84  ├─loans ---- 应用目录，用于管理贷款
85  │   │  admin.py ---- 管理员设置文件
86  │   │  apps.py ---- 应用配置文件
87  │   │  forms.py ---- 表单定义文件
88  │   │  models.py ---- 数据模型文件
89  │   │  tests.py ---- 测试文件
90  │   │  urls.py ---- 路由配置文件
91  │   │  views.py ---- 视图处理文件
92  │   │  __init__.py ---- 应用初始化文件
93  │   │
94  │   └─templates ---- 模板目录
95  │      └─loans
96  │              apply_loan.html ---- 申请贷款页面模板
97  │              approve_loan.html ---- 批准贷款页面模板
98  │              branch_loans.html ---- 分行贷款页面模板
99  │              edit_loan.html ---- 编辑贷款页面模板
```

```
│            loans.html ---- 贷款列表页面模板
│            pay_loan.html ---- 还款页面模板
│
├─media ---- 媒体文件目录
│  └─photos
│      │  default.jpg ---- 默认图片
│      │
│      ├─20240528
│      │      default.jpg
│      │      default_4GOn7vn.jpg
│      │      default_6U1CuOm.jpg
│      │      default_f6i9WX5.jpg
│      │      default_jnDUwWQ.jpg
│      │      default_PjoXi99.jpg
│      │
│      └─20240529
│             QQicon.jpg
│
├─staffs ---- 应用目录，用于管理员工
│  │  admin.py ---- 管理员设置文件
│  │  apps.py ---- 应用配置文件
│  │  forms.py ---- 表单定义文件
│  │  models.py ---- 数据模型文件
│  │  signals.py ---- 信号处理文件
│  │  tests.py ---- 测试文件
│  │  urls.py ---- 路由配置文件
│  │  views.py ---- 视图处理文件
│  │  __init__.py ---- 应用初始化文件
│  │
│  └─templates ---- 模板目录
│      └─staffs
│             create_staff.html ---- 创建员工页面模板
│             edit_staff.html ---- 编辑员工页面模板
│             staffs.html ---- 员工列表页面模板
│
├─static ---- 静态文件目录
│  └─css
│         style.css ---- 样式文件
│
└─users ---- 应用目录，用于管理用户
    │  admin.py ---- 管理员设置文件
    │  apps.py ---- 应用配置文件
    │  forms.py ---- 表单定义文件
    │  models.py ---- 数据模型文件
    │  tests.py ---- 测试文件
    │  urls.py ---- 路由配置文件
    │  views.py ---- 视图处理文件
    │  __init__.py ---- 应用初始化文件
    │
    └─templates ---- 模板目录
       └─registration
              change_pwd.html ---- 修改密码页面模板
              edit.html ---- 编辑用户页面模板
              get_users.html ---- 用户列表页面模板
              logged_out.html ---- 退出登录页面模板
              login.html ---- 登录页面模板
```

可以看到整个Django项目的文件结构，其中包含了accounts、bills、branch、department、frontend、loans、staffs、users等应用，每个应用中包含了模型、视图、表单、路由、模板等文件，通过Django的ORM框架实现数据库操作，通过Django的template引擎实现前端页面渲染。由于每个app中主要的工作都在`models.py`、`views.py`、`forms.py`、`urls.py`、`templates`中，而urls中的路由设置和templates中的模板文件前端代码并不是我们这门课程的重点，因此我们主要关注`models.py`、`views.py`中的代码，部分应用涉及填写表单，因此也会关注`forms.py`中的代码。

接下来我将按照ER图中的实体关系，从上到下，从左到右，逐一介绍每个实体对应的应用的设计思路和核心代码。

3.3 银行员工（staffs）

作为介绍的第一个实体，我将全面且详细地介绍应用中每个主要工作文件的设计思路和核心代码，包括`models.py`、`views.py`、`forms.py`、`urls.py`、`templates`中的文件，后续其他的应用实现将主要介绍前两个文件。

3.3.1 数据模型

```python
# staffs/models.py
class Staff(models.Model):
    objects = models.Manager()
    staff_id = models.AutoField(primary_key=True)
    department = models.ForeignKey(BranchDepartments, on_delete=models.CASCADE, related_name='DepartmentStaff')
    name = models.CharField(max_length=20)
    tel = models.CharField(max_length=11)
    address = models.CharField(max_length=100)
    photo = models.ImageField(upload_to='photos/%Y%m%d/', default='photos/default.jpg')
    def __str__(self):
        return f"{self.staff_id}-{self.name}"

class Manager(models.Model):
    objects = models.Manager()
    staff = models.OneToOneField(Staff, on_delete=models.CASCADE, related_name='StaffManager', primary_key=True)
    department = models.ForeignKey(BranchDepartments, on_delete=models.CASCADE, related_name='DepartmentManager')

    def __str__(self):
        return f"{self.staff.name}-{self.department.name}"
```

在`models.py`文件中，定义了员工`Staff`和经理`Manager`两个数据模型，其中员工`Staff`包含了员工号、部门、姓名、电话、地址、照片等属性，经理`Manager`包含了员工、部门等属性，通过外键和一对一键实现了员工和部门的关联，员工和经理的关联。

其中有一个特殊的地方是照片属性，这里使用了`ImageField`类型，用于存储员工的照片，通过`upload_to`参数指定了照片的存储路径，通过`default`参数指定了默认照片。（这里也是本项目中唯一涉及到文件管理的地方）

3.3.2 视图处理

`views.py`文件中定义了员工列表、创建员工、编辑员工、删除员工、设置经理、删除经理等视图处理函数，通过Django的视图处理函数实现了员工的增删改查功能。

具体的功能通过`urls.py`中的路由映射实现，如下：

```
# staffs/urls.py
app_name = 'staffs'
urlpatterns = [
    path('staffs/', views.staffs, name='staffs'),
    path('edit/<int:staff_id>/', views.edit_staff, name='edit_staff'),
    path('create/<int:department_id>/', views.create_staff, name='create_staff'),
    path('delete/<int:staff_id>/', views.delete_staff, name='delete_staff'),
    path('set_manager/<int:staff_id>/<int:department_id>/', views.set_manager,
name='set_manager'),
    path('delete_manager/<int:department_id>/', views.delete_manager,
name='delete_manager'),
]
```

`urls.py`指定了应用名称，并通过path函数指定了路径和视图处理函数的映射关系，如代码所示：

`staffs/`路径对应`views.staffs`视图处理函数，

`edit/<int:staff_id>/`路径对应`views.edit_staff`视图处理函数，

`create/<int:department_id>/`路径对应`views.create_staff`视图处理函数，

`delete/<int:staff_id>/`路径对应`views.delete_staff`视图处理函数，

`set_manager/<int:staff_id>/<int:department_id>/`路径对应`views.set_manager`视图处理函数，

`delete_manager/<int:department_id>/`路径对应`views.delete_manager`视图处理函数。

> 后续的应用设计也是类似的，将不再赘述路由映射的设计。

3.3.2.1 获取员工列表

获取员工列表视图处理函数如下：

```
# staffs/views.py
@login_required
def staffs(request):
    user = request.user
    if user.is_superuser:
        if user.username == 'admin':
            staffs_lists = Staff.objects.all()
        else:
            staffs_lists = Staff.objects.filter(department__branch_id=user.username)
    else:
        messages.warning(request, '你没有权限查看员工信息')
        return render(request, 'frontend/error.html')
    paginator = Paginator(staffs_lists, 4)
    page = request.GET.get('page')
    staffs_list = paginator.get_page(page)
    context = {'staffs': staffs_list}
```

```
    return render(request, 'staffs/staffs.html', context)
```

员工列表视图处理函数staffs，首先判断用户是否登录，然后判断用户是否是管理员，如果是总管理则可以查看所有员工信息，否则只能查看管理员所在支行的员工信息。通过`Staff.objects.all()`获取所有员工信息，通过`Staff.objects.filter(department__branch_id=user.username)`获取管理员所在支行的员工信息，通过`Paginator`实现了分页功能，通过`messages.warning`实现了消息提示功能，通过`render`实现了页面渲染功能。

前端页面模板staffs.html如下：

```
<!-- templates/staffs/staffs.html -->
{% extends "frontend/base.html" %}
{% load bootstrap4 %}

{% block page_header %}
<div class="text-center my-4">
    <h1 class="text-white">员工信息</h1>
</div>
{% endblock page_header %}

{% block content %}
<div class="container">
    <div class="row row-cols-1 row-cols-md-2 g-4">
        {% for staff in staffs %}
        <div class="col">
            <div class="card h-100">
                <div class="card-body">
                    <div class="d-flex align-items-center mb-3">
                        {% if staff.photo %}
                        <img src="{{ staff.photo.url }}" class="rounded-circle me-3" style="width: 80px; height: 80px; object-fit: cover;">
                        {% else %}
                        <div class="rounded-circle bg-secondary me-3" style="width: 80px; height: 80px;"></div>
                        {% endif %}
                        <div>
                            <h5 class="card-title mb-0">{{ staff.name }}</h5>
                            <p class="card-text text-muted mb-0">{{ staff.department }}</p>
                        </div>
                    </div>
                    <p class="card-text mb-1"><strong>工号:</strong> {{ staff.staff_id }}</p>
                    <p class="card-text mb-1"><strong>电话:</strong> {{ staff.tel }}</p>
                    <p class="card-text mb-1"><strong>地址:</strong> {{ staff.address }}</p>
                </div>
                {% if user.is_superuser %}
                <div class="card-footer d-flex justify-content-between">
                    <div>
                        <a href="{% url 'staffs:set_manager' staff.staff_id staff.department.department_id %}" class="btn btn-sm btn-outline-primary">设置经理</a>
                        <a href="{% url 'staffs:edit_staff' staff.staff_id %}" class="btn btn-sm btn-outline-primary">修改信息</a>
                    </div>
                    <a href="{% url 'staffs:delete_staff' staff.staff_id %}" class="btn btn-sm btn-outline-danger">删除员工</a>
```

```
                </div>
                {% endif %}
            </div>
        </div>
        {% endfor %}
    </div>
</div>

<div class="d-flex justify-content-center mt-4">
    <nav aria-label="Page navigation">
        <ul class="pagination">
            {% if staffs.has_previous %}
            <li class="page-item">
                <a href="?page=1" class="page-link">&laquo; 1</a>
            </li>
            <li class="page-item">
                <a href="?page={{ staffs.previous_page_number }}" class="page-link">{{ staffs.previous_page_number }}</a>
            </li>
            {% endif %}
            <li class="page-item active">
                <span class="page-link">{{ staffs.number }}</span>
            </li>
            {% if staffs.has_next %}
            <li class="page-item">
                <a href="?page={{ staffs.next_page_number }}" class="page-link">{{ staffs.next_page_number }}</a>
            </li>
            <li class="page-item">
                <a href="?page={{ staffs.paginator.num_pages }}" class="page-link">{{ staffs.paginator.num_pages }} &raquo;</a>
            </li>
            {% endif %}
        </ul>
    </nav>
</div>
{% endblock content %}
```

这段前端代码继承了`frontend/base.html`模板，通过`bootstrap4`模板标签加载了bootstrap4样式，通过`block`标签定义了页面标题和内容，通过`for`循环遍历员工列表，通过`if`判断员工是否有照片，通过`url`模板标签实现了跳转链接，通过`pagination`实现了分页功能。

> 其他视图处理函数和前端页面模板的设计思路和核心代码也是类似的，继承`base.html`模板，再根据具体的功能需求，通过`for`循环、`if`判断、`url`模板标签、`pagination`等实现了具体交互功能，由于前端代码并非本次实验的重点，因此后续将不再赘述。

3.3.3 新增员工

新增员工视图处理函数如下：

```
# staffs/views.py
@login_required
def create_staff(request, department_id):
    name = None
    if request.user.is_superuser:
        name = request.user.username
    branch = BranchDepartments.objects.get(department_id=department_id).branch_id
    if not (name and name == branch or name == 'admin'):
        messages.warning(request, '你没有权限操作该部门')
        return render(request, 'frontend/error.html')
    # judge if user is superuser
    if not request.user.is_superuser:
        messages.warning(request, '你没有权限创建员工')
        return render(request, 'frontend/error.html')
    department = BranchDepartments.objects.get(department_id=department_id)
    if request.method != 'POST':
        form = StaffCreateForm(initial={'department': department})
    else:
        form = StaffCreateForm(initial={'department': department}, data=request.POST,
files=request.FILES)
        if form.is_valid():
            name = form.cleaned_data.get("name")
            tel = form.cleaned_data.get("tel")
```

```
            address = form.cleaned_data.get("address")
            staff = Staff.objects.create(department=department, name=name, tel=tel,
address=address)
            if 'photo' in request.FILES:
                staff.photo = form.cleaned_data.get("photo")
            staff.save()
            return redirect('staffs:staffs')
    context = {'form': form, 'department': department}
    return render(request, 'staffs/create_staff.html', context)
```

新增员工视图处理函数create_staff，首先判断用户是否登录，然后判断用户是否是管理员，如果是总管理则可以创建员工，否则只能创建管理员所在支行的员工。通过`BranchDepartments.objects.get(department_id=department_id).branch_id`获取部门所在支行，通过`StaffCreateForm`创建员工表单，通过`form.is_valid()`判断表单是否合法，通过`Staff.objects.create`创建员工，通过`redirect`实现页面跳转。

其中表单信息是通过`forms.py`中的表单类实现的，如下：

```
# staffs/forms.py
class StaffCreateForm(forms.ModelForm):
    department = forms.ModelChoiceField(label='所属部门',
queryset=BranchDepartments.objects.all(), disabled=True)
    name = forms.CharField(label='姓名', max_length=20)
    tel = forms.CharField(label='电话', max_length=11)
    address = forms.CharField(label='地址', max_length=100)
    photo = forms.ImageField(label='照片', required=False)

    class Meta:
        model = Staff
        fields = ('department', 'name', 'tel', 'address', 'photo')
```

> 只要涉及到用户填写的表单，都需要通过`forms.py`中的表单类实现，通过`ModelForm`类实现表单的定义，通过`ModelChoiceField`、`CharField`、`ImageField`等字段类型实现表单的定义，后续

表单类StaffCreateForm继承了`forms.ModelForm`，定义了员工的所属部门、姓名、电话、地址、照片等字段，通过`ModelChoiceField`、`CharField`、`ImageField`等字段类型实现了表单的定义。

通过将表单类和视图处理函数结合，实现了员工的创建功能。

这个功能前端嵌入在了部门管理页面中，如下所示：

4 实验与测试

4.1 依赖

所需的库、运行环境

4.2 部署

代码运行步骤，建议使用命令行运行代码

4.3 实验结果

如增删改查、验证存储过程、函数、触发器、文件管理

5　参考

如前端使用的模板、引用的图片来源、第三方库的官网等等