# 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：熊习乔

学生学号：PB21152828

完成时间：2024年6月4日

# 1 需求分析

本系统的应用场景假定为银行系统的管理员，拥有对所有信息的全部权限（增删改查）。

## 1.1 数据需求

- 支行：每个支行有银行名、地址和总资产三个属性。其中银行名作为支行的主键，总资产定义为该支行所有账户的余额之和。
- 客户：每个客户有身份证号和姓名两个属性。其中身份证号作为主键。
- 账户：每个账户有账户号和余额两个属性。其中账户号作为主键。
- 贷款：每笔贷款有贷款号、未还金额和总额度三个属性。其中贷款号作为主键。
- 部门：每个部门有部门号、部门名、领导身份证三个属性。
- 员工：每个员工有身份证号、姓名和照片三个属性。其中身份证号是主键。每个员工可以属于多个部门，每个部门也可以有多个员工。

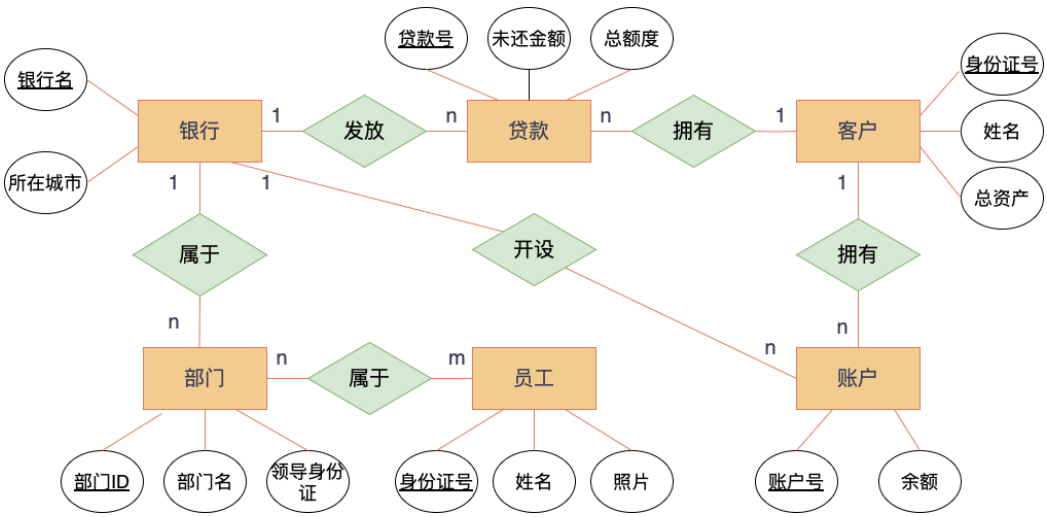
## 1.2 功能需求

每项数据都支持基本的增删改查。除此之外，还支持对员工照片的图片管理，支持新增/修改/删除部门时自动对员工的归属关系进行修改，银行资产的自动计算。同时，保证账户余额不能小于0，贷款金额必须大于0，未还金额不能小于0，未还金额不能大于贷款金额。

# 2 数据库设计

## 2.1 ER图

以下是该系统的ER图。




## 2.2 模式分解

每个实体一个表。银行、客户和员工的属性就是ER图中的三个属性。贷款的属性由于两个1:n的关系，新增了银行名和客户身份证号。账户的属性由于两个1:n的关系，新增了银行名和客户身份证号。部门的属性由于一个1:n的关系，新增了银行名。由于员工和部门的多对多关系，所以新增一个部门-员工表，属性为部门号和员工身份证号，两个属性共同构成主键。

由于每个任一实例的元组的每个属性都只含有一个值，显然该关系模式是1NF的。同时可以看出该关系模式的每一个非主属性都完全依赖于主码，故该关系模式是2NF的。检查该关系属性可以得到并不存在某个非主属性传递依赖于主码，所以该关系模式是3NF的。

创建表的MySQL代码如下：

```
-- 一个银行管理系统，涉及：银行信息、客户信息、账户信息、贷款信息、银行部门信息、员工信息相关实体。
-- 本文件用于创建数据库表
use db_lab2;

SET foreign_key_checks=0;  # 关闭外键检查
DROP TABLE if EXISTS bank, customer, account, loan, department, employee,
employee_department;
SET foreign_key_checks=1;  # 开启外键检查

CREATE TABLE bank (
    bank_name VARCHAR(50) PRIMARY KEY,
    bank_addr VARCHAR(100) NOT NULL
);

CREATE TABLE customer (
    id VARCHAR(50) PRIMARY KEY,
    customer_name VARCHAR(50) NOT NULL
);

CREATE TABLE account (
    account_id VARCHAR(50) PRIMARY KEY,
    balance DECIMAL(20, 2) DEFAULT 0.00,
    customer_id VARCHAR(50) NOT NULL,
    bank_name VARCHAR(50) NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES customer(id) ON DELETE CASCADE,
    FOREIGN KEY (bank_name) REFERENCES bank(bank_name) ON DELETE CASCADE,
    CONSTRAINT CHK_balance CHECK (balance >= 0)
);

CREATE TABLE loan (
    loan_id VARCHAR(50) PRIMARY KEY,
    loan_amount DECIMAL(20, 2) NOT NULL,
    unrepayed_amount DECIMAL(20, 2) NOT NULL,
    customer_id VARCHAR(50) NOT NULL,
    bank_name VARCHAR(50) NOT NULL,
    FOREIGN KEY (customer_id) REFERENCES customer(id) ON DELETE CASCADE,
    FOREIGN KEY (bank_name) REFERENCES bank(bank_name) ON DELETE CASCADE,
    CONSTRAINT CHK_loan_amount CHECK (loan_amount >= 0),
    CONSTRAINT CHK_unrepayed_amount CHECK (unrepayed_amount >= 0),
    CONSTRAINT CHK_unrepayed_loan CHECK (unrepayed_amount <= loan_amount)
);
```

```
CREATE TABLE employee (
    id VARCHAR(50) PRIMARY KEY,
    employee_name VARCHAR(50) NOT NULL,
    path_to_photo VARCHAR(100)
);

CREATE TABLE department (
    department_id VARCHAR(50) PRIMARY KEY,
    bank_name VARCHAR(50) NOT NULL,
    department_name VARCHAR(50) NOT NULL,
    leader_id VARCHAR(50) NOT NULL,
    FOREIGN KEY (bank_name) REFERENCES bank(bank_name) ON DELETE CASCADE,
    FOREIGN KEY (leader_id) REFERENCES employee(id) ON DELETE CASCADE
);

CREATE TABLE employee_department (
    employee_id VARCHAR(50) NOT NULL,
    department_id VARCHAR(50) NOT NULL,
    PRIMARY KEY (employee_id, department_id),
    FOREIGN KEY (employee_id) REFERENCES employee(id) ON DELETE CASCADE,
    FOREIGN KEY (department_id) REFERENCES department(department_id) ON DELETE
CASCADE
);
```

### 2.3 存储过程、触发器、函数等设计思路

针对各个实体的操作，除了查询都使用存储过程实现。这样设计的目的是可以在存储过程中检测操作的合法性并返回相关的错误代码。如果操作失败，返回相关的错误代码，否则返回1作为操作成功的标志。

由于需要在部门`department`和员工-部门`employee_department`两个表中维持一致性，所以我设计在对`department`进行增、删、改操作的时候`employee_department`也会同步更新。由于`employee_department`中的`department_id`属性定义了ON DELETE CASCADE，所以在删除部门的时候会自动级联删除相关的员工-部门关系。剩下的增和改我设计为用触发器实现，当用户对`department`做出修改时，会通过触发器自动修改`employee_department`中的相关记录，并且新的`department`记录创建时，也会通过触发器自动在`employee_department`增加“领导ID—部门ID”的一条记录。触发器的具体设计见“核心代码解析——部门管理”

我设计通过函数计算用户的总资产并返回，而不是通过一条属性存储在`customer`表中。这样可以简化数据库的设计，同时维护数据一致性。

## 3 核心代码解析

### 3.1 仓库地址

https://github.com/psycho-xiong/ustc-database-lab2-2024

### 3.2 目录

```
.
├── 2_db-lab02.pptx -------实验文档
├── __pycache__
```

```
│   └── db.cpython-311.pyc
├── db.py -------与数据库交互
├── main.py -------与网页交互，调用db.py中的函数
├── mysql
│   ├── procedures.sql -------存储过程、函数、触发器
│   └── table_init.sql -------初始化表
├── report
│   ├── ER.drawio -------ER图的drawio文件
│   ├── ER.drawio.png -------ER图
│   ├── src
│   │   └── logo.png
│   ├── 银行管理系统_需求分析.md -------需求分析
│   ├── 银行管理系统_需求分析.pdf -------需求分析
│   ├── 数据库实验报告.md -------实验报告(.md)
│   └── 银行管理系统报告_熊习乔_PB21151828.pdf -------实验报告(.pdf)
├── static
│   └── photos -------员工照片
│       ├── 111.bmp
│       ├── 112.bmp
│       └── 113.bmp
├── templates -------存放html文件
│   ├── account.html
│   ├── back.html
│   ├── bank.html
│   ├── customer.html
│   ├── department.html
│   ├── employee.html
│   ├── employee_department.html
│   ├── homepage.html
│   ├── loan.html
│   └── login.html
└── tree.txt -------目录树

8 directories, 29 files
```

## 3.3 登录

提供登录界面，用户输入用户名和密码，正确则进入主页，否则会弹窗提示检查用户名和密码，并停留在登录界面。

来源：`main.py`

```python
# 主页跳转到登录页面
@app.route('/', methods=['GET', 'POST'])
def login():
    """Log in a registered user by adding the user id to the session."""
    if request.method == 'GET':
        return render_template('login.html')
    else:
        # 连接数据库。如果连接成功，跳转到主页。否则，显示错误信息，停留在登录页面。
        user = request.form['username']
        password = request.form['password']
        conn = db_login(user=user, passward=password)
```

```
        if (conn == None) or (user != 'root'):
            return render_template("login.html",status=-1)
        else:
            session['username'] = user
            session['password'] = password
            return redirect(url_for('homepage'))
```

## 3.4 主页

如果用户输入用户名和密码正确则进入主页，主页可以选择具体功能，包括：支行、用户、账户、贷款、部门、员工、部门-员工。

来源：`main.py`

```
# 主页
@app.route('/homepage', methods=['GET', 'POST'])
def homepage():
    """Display the homepage."""
    if request.method == 'GET':
        return render_template('homepage.html')
    else:
        if 'Customer' in request.form:
            return redirect(url_for('customer'))
        if 'Bank' in request.form:
            return redirect(url_for('bank'))
        if 'Account' in request.form:
            return redirect(url_for('account'))
        if 'Loan' in request.form:
            return redirect(url_for('loan'))
        if 'Employee' in request.form:
            return redirect(url_for('employee'))
        if 'Department' in request.form:
            return redirect(url_for('department'))
        if 'Employee_Department' in request.form:
            return redirect(url_for('employee_department'))
```

## 3.5 支行管理

在主页进入支行部分后，可以对支行进行增删改查。“查”支持对主键外的任意属性或者属性的组合进行查找（主键单独查找）。“改”要求待改的数据存在且修改后的数据不能和原有数据重复。“增”要求新增的数据不能和原有数据重复。“删”要求待删的数据存在。这些功能和要求在之后的各个部分都一样，之后不再赘述。

`main.py`从网页表单获取服务种类，并根据服务种类获取需要的数据。在获取需要的数据之后，调用`db.py`中的函数来执行对应的操作。

需要指出的是我仅在“支行管理”中列出这一部分的完整代码，由于后面的各个管理功能与该部分的代码相似，如无必要在之后我将只给出框架而非完整代码。

来源：`main.py`

```
@app.route("/homepage/bank", methods = (["GET", "POST"]))
def bank():
    if request.method == 'GET':
```

```
        return render_template('bank.html')
    else:
        # 如果 session 中没有连接，返回登录页面
        if 'username' not in session:
            return redirect(url_for('login'))
        # 如果 session 中有连接，从 session 中获取连接，然后执行后续操作
        username = session['username']
        password = session['password']
        conn = db_login(user=username, password=password)
        if 'SEARCH' in request.form:
            search_text = request.form['search_text']
            search_type = request.form['search_type']
            res = bank_search(conn, search_text, search_type)
            return render_template('bank.html', search_res=res)
        elif 'ADD' in request.form:
            new_name = request.form['new_name']
            new_addr = request.form['new_addr']
            res = bank_add(conn, new_name, new_addr)
            return render_template('bank.html', add_res=res)
        elif 'DELETE' in request.form:
            delete_name = request.form['delete_name']
            res = bank_delete(conn, delete_name)
            return render_template('bank.html', delete_res=res)
        elif 'UPDATE' in request.form:
            old_name = request.form['old_name']
            new_name = request.form['new_name']
            new_addr = request.form['new_addr']
            res = bank_update(conn, old_name, new_name, new_addr)
            return render_template('bank.html', update_res=res)
```

其中调用的与数据库交互的函数包含增、删、改、查四个功能。同样地我只在这一节列出完整代码，之后如无必要将只给出对应部分的代码框架。customer使用到的增删改查函数定义如下：

来源：`db.py`

```
# ************** Customer **************
def customer_search(conn: MySQLConnection, search_text: str, search_type: str) ->
list:
    cursor = conn.cursor()
    if search_text == '':
        cursor.execute("SELECT * FROM customer")
    else:
        if search_type == 'ID':
            cursor.execute(f"SELECT * FROM customer WHERE id = '{search_text}'")
        elif search_type == 'Name':
            cursor.execute(f"SELECT * FROM customer WHERE customer_name =
'{search_text}'")
    res = cursor.fetchall()
    for i in range(len(res)):
        cursor.execute(f"SELECT get_total_balance('{res[i][0]}')")
        res[i] = res[i] + cursor.fetchall()[0]
    cursor.close()
```

```
    return res

def customer_add(conn: MySQLConnection, add_id: str, add_name: str) -> int:
    cursor = conn.cursor()
    sta = -1
    try:
        sta = cursor.callproc('create_customer', (add_id, add_name, sta))[-1]
        conn.commit()
        cursor.close()
        return sta
    except:
        conn.rollback()
        cursor.close()
        return -1

def customer_delete(conn: MySQLConnection, delete_id: str) -> int:
    cursor = conn.cursor()
    sta = -1
    try:
        sta = cursor.callproc('delete_customer', (delete_id, sta))[-1]
        conn.commit()
        cursor.close()
        return sta
    except:
        conn.rollback()
        cursor.close()
        return -1

def customer_update(conn: MySQLConnection, old_id: str, new_id: str, new_name: str)
-> int:
    cursor = conn.cursor()
    sta = -1
    try:
        sta = cursor.callproc('change_customer', (old_id, new_id, new_name, sta))
[-1]
        conn.commit()
        cursor.close()
        return sta
    except:
        conn.rollback()
        cursor.close()
        return -1
```

其中只有查询是通过 mysql-connector-python-rf 直接执行查询操作，针对不同的查询信息，直接执行对应的查询语句。但是增、删和改为了检查是否满足各种约束条件并返回相关错误信息，都需要调用写好的存储过程。同样地我只在这一节列出完整代码，之后如无必要将只给出对应部分的代码框架。customer使用到的存储过程定义如下：

来源：`procedures.sql`

```
-- A procedure that changes the name of a bank.
DROP PROCEDURE if exists change_bank;
delimiter //
```

```
CREATE PROCEDURE change_bank(IN old_bank_name VARCHAR(50), IN new_bank_name
VARCHAR(50), In new_bank_addr VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the bank exsit
    SELECT count(*) FROM bank WHERE bank_name = old_bank_name INTO a;
    IF a = 1 THEN
        IF new_bank_name = old_bank_name THEN
            UPDATE bank SET bank_addr = new_bank_addr WHERE bank_name =
old_bank_name;
            SET sta = 1;
        ELSE
            SELECT count(*) FROM bank WHERE bank_name = new_bank_name INTO a;
            IF a = 1 THEN
                SET sta = -4; -- Error code -4: New bank already exists
            ELSE
                INSERT INTO bank (bank_name, bank_addr) VALUES (new_bank_name,
new_bank_addr);
                -- Update the bank_name of all accounts
                UPDATE account SET bank_name = new_bank_name WHERE bank_name =
old_bank_name;
                -- Update the bank_name of all loans
                UPDATE loan SET bank_name = new_bank_name WHERE bank_name =
old_bank_name;
                -- Update the bank_name of all departments
                UPDATE department SET bank_name = new_bank_name WHERE bank_name =
old_bank_name;
                -- Delete the old bank
                DELETE FROM bank WHERE bank_name = old_bank_name;
                SET sta = 1;
            END IF;
        END IF;
    ELSE
        SET sta = -2; -- Error code -2: bank does not exist
    END IF;
END //
delimiter ;


-- A procedure that creates a new bank.
DROP PROCEDURE if exists create_bank;
delimiter //
CREATE PROCEDURE create_bank(IN add_bank_name VARCHAR(50), IN bank_addr VARCHAR(50),
OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the bank exsit
    SELECT count(*) FROM bank WHERE bank_name = add_bank_name INTO a;
    IF a = 0 THEN
        INSERT INTO bank (bank_name, bank_addr) VALUES (add_bank_name, bank_addr);
```

```
        SET sta = 1;
    ELSE
        SET sta = -3; -- Error code -3: bank already exists
    END IF;
END //
delimiter ;


-- A procedure that deletes a bank.
DROP PROCEDURE if exists delete_bank;
delimiter //
CREATE PROCEDURE delete_bank(IN delete_bank_name VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the bank exsit
    SELECT count(*) FROM bank WHERE bank_name = delete_bank_name INTO a;
    IF a = 1 THEN
        DELETE FROM bank WHERE bank_name = delete_bank_name;
        SET sta = 1;
    ELSE
        SET sta = -2; -- Error code -2: bank does not exist
    END IF;
END //
delimiter ;
```

修改记录先要检查被修改的主键是否存在，如过不存在设置错误码。如果存在且主键没有改变直接修改。如果存在但是主键别修改，需要检查新的主键是否已经存在，如果已经存在设置错误码。如果不存在则增加新纪录，修改相关联的表中的相关记录，最后删除旧的记录。

创建记录只需要检查新增的主键是否已经存在。如果已经存在设置错误码，如果不存在就插入一条新纪录。

删除记录需要检查待删除的记录的主键是否存在，如果不存在设置错误码，如果存在则直接删去该条记录。注意由于这里其他表中需要连带删除的地方都设置了ON DELETE CASCADE，所以不再需要显示用语句删除其他表中的相关记录。

通过存储过程进行各种检查，包括数据一致性、操作合法性等，如果执行失败返回错误码，以便在网页显示。

## 3.6 客户管理

从主页进入客户管理的界面后，可以进行客户的增、删、改、查。

来源：`main.py`

```
@app.route("/homepage/costumer", methods = (["GET", "POST"]))
def customer():
    if request.method == 'GET':
        return render_template('customer.html')
    else:
        # 如果 session 中没有连接，返回登录页面
        if 'username' not in session:
            return redirect(url_for('login'))
        # 如果 session 中有连接，从 session 中获取连接，然后执行后续操作
```

```
        username = session['username']
        password = session['password']
        conn = db_login(user=username, password=password)
        if 'SEARCH' in request.form:
            ...
        elif 'ADD' in request.form:
            ...
        elif 'DELETE' in request.form:
            ...
        elif 'UPDATE' in request.form:
           ...
```

其中用到的的与数据库交互的函数定义如下：

来源：`db.py`

```
# ************** Customer **************
def customer_search(conn: MySQLConnection, search_text: str, search_type: str) ->
list:
    cursor = conn.cursor()
    if search_text == '':
        cursor.execute("SELECT * FROM customer")
    else:
        if search_type == 'ID':
            ...
        elif search_type == 'Name':
            ...
    res = cursor.fetchall()
    for i in range(len(res)):
        cursor.execute(f"SELECT get_total_balance('{res[i][0]}')")
        res[i] = res[i] + cursor.fetchall()[0]
    cursor.close()
    return res

def customer_add(conn: MySQLConnection, add_id: str, add_name: str) -> int:
    cursor = conn.cursor()
    sta = -1
    try:
        sta = cursor.callproc('create_customer', (add_id, add_name, sta))[-1]
        ...
    except:
       ...

def customer_delete(conn: MySQLConnection, delete_id: str) -> int:
    cursor = conn.cursor()
    sta = -1
    try:
        sta = cursor.callproc('delete_customer', (delete_id, sta))[-1]
       ...
    except:
        ...
```

```
def customer_update(conn: MySQLConnection, old_id: str, new_id: str, new_name: str)
-> int:
    cursor = conn.cursor()
    sta = -1
    try:
        sta = cursor.callproc('change_customer', (old_id, new_id, new_name, sta))
[-1]
       ...
    except:
        ...
```

其中用户总资产的部分并不是直接作为一个属性存储的，而是通过调用函数`get_total_balance`计算得到。函数`get_total_balance`的定义如下：

```
-- A function to calculate the total balance of a customer
DROP FUNCTION if exists get_total_balance;
delimiter //
CREATE FUNCTION get_total_balance(search_customer_id VARCHAR(50))
RETURNS DECIMAL(20, 2)
READS SQL DATA
BEGIN
    DECLARE total_balance DECIMAL(20, 2);
    SELECT SUM(balance) INTO total_balance
    FROM account
    WHERE customer_id = search_customer_id;
    RETURN total_balance;
END//
delimiter ;
```

增、删、改使用的的MySQL存储过程如下，同样只展示框架：

来源：`procedures.sql`

```
-- A procedure that changes the id of a customer.
DROP PROCEDURE if exists change_customer;
delimiter //
CREATE PROCEDURE change_customer(IN old_customer_id VARCHAR(50), IN new_customer_id
VARCHAR(50), IN new_name VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the customer exsit
    SELECT count(*) FROM customer WHERE id = old_customer_id INTO a;
    IF a = 1 THEN
        IF new_customer_id = old_customer_id THEN
            ...
        ELSE
            SELECT count(*) FROM customer WHERE id = new_customer_id INTO a;
            IF a = 1 THEN
                SET sta = -4; -- Error code -4: New customer already exists
            ELSE
                ...
            END IF;
```

```
        END IF;
    ELSE
        SET sta = -2; -- Error code -2: customer does not exist
    END IF;
END //
delimiter ;

-- A procedure that creates a new customer.
DROP PROCEDURE if exists create_customer;
delimiter //
CREATE PROCEDURE create_customer(IN customer_id VARCHAR(50), IN customer_name
VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the customer exsit
    SELECT count(*) FROM customer WHERE id = customer_id INTO a;
    IF a = 0 THEN
        ...
    ELSE
        SET sta = -3; -- Error code -3: customer already exists
    END IF;
END //
delimiter ;

-- A procedure that deletes a customer.
DROP PROCEDURE if exists delete_customer;
delimiter //
CREATE PROCEDURE delete_customer(IN customer_id VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the customer exsit
    SELECT count(*) FROM customer WHERE id = customer_id INTO a;
    IF a = 1 THEN
        ...
    ELSE
        SET sta = -2; -- Error code -2: customer does not exist
    END IF;
END //
delimiter ;
```

## 3.7 账户管理

从主页进入账号管理后，可以对账号进行增删改查，并且在这里实现了转账功能。

来源：`main.py`

```
@app.route('/homepage/account', methods=['GET', 'POST'])
def account():
    if request.method == 'GET':
        return render_template('account.html')
```

```
    else:
        # 如果 session 中没有连接，返回登录页面
        if 'username' not in session:
            return redirect(url_for('login'))
        # 如果 session 中有连接，从 session 中获取连接，然后执行后续操作
        username = session['username']
        password = session['password']
        conn = db_login(user=username, password=password)
        if 'SEARCH' in request.form:
            ...
        elif 'ADD' in request.form:
            ...
        elif 'DELETE' in request.form:
            ...
        elif 'UPDATE' in request.form:
            ...
        elif 'TRANSFER' in request.form:
            from_id = request.form['from_id']
            to_id = request.form['to_id']
            amount = request.form['amount']
            res = account_transfer(conn, from_id, to_id, amount)
            return render_template('account.html', transfer_res=res)
```

转账需要获取转入账户、转出账户与转账金额。然后调用处理转账的函数`account_transfer，定义如下：

来源：`db.py`

```
def account_transfer(conn, from_id, to_id, amount):
    cursor = conn.cursor()
    sta = -1
    if from_id == to_id:
        return -12
    try:
        (amount, flag) = num_check_trans(amount)
        if flag != 1:
            return flag
        sta = cursor.callproc('transfer_money', (from_id, to_id, amount, sta))[-1]
        conn.commit()
        cursor.close()
        return sta
    except:
        conn.rollback()
        cursor.close()
        return -1
```

同时还调用了检查是否输入的是合法数字的函数`num_check_trans`，该函数输入从网页获取的金额，返回转换为数字的amount和指示是否为合法数字的flag。该函数的定义如下：

来源：`db.py`

```
def num_check_trans(num: str) -> tuple:
    flag = -1
```

```
    # 检查是否为空
    if num == '':
        flag = -10 # Error code -10: not a number
        return (None,flag)
    # 移除所有的空格
    num = num.replace(' ','')
    # 检查是否为负数
    if num[0] == '-':
        flag = -11 # Error code -11: negative number
        return (None, flag)
    for i in num:
        if not i.isdigit() and i != '.':
            flag = -10  # Error code -10: not a number
            return (None,flag)
    # 检查是否为小数
    if '.' in num:
        num = num.split('.')
        if len(num[1]) > 2:
            flag = -7 # Error code -7: decimal part too long
            return (None,flag)
        if len(num[0] + num[1]) > 20:
            flag = -8 # Error code -8: total length too long
            return (None,flag)
        # 遍历检查是否全为数字
        for i in num[0] + num[1]:
            if not i.isdigit() and i != '.':
                flag = -10 # Error code -10: not a number
                return (None,flag)
    else:
        if len(num) > 20:
            flag = -8 # Error code -8: total length too long
            return (None,flag)
        for i in num:
            if not i.isdigit() and i != '.':
                flag = -10 # Error code -10: not a number
                return (None,flag)
    # 转换为数字
    num = float(num)
    flag = 1
    return (num,flag)
```

下面详细介绍转账过程用到的存储过程：

来源：`procedures.sql`

```
-- A procedure that transfer money from one account to another. Transaction is used
to ensure the atomicity of the operation.
DROP PROCEDURE if exists transfer_money;
delimiter //
CREATE PROCEDURE transfer_money(IN from_account_id VARCHAR(50), IN to_account_id
VARCHAR(50), IN amount DECIMAL(20, 2), OUT sta INT)
BEGIN
```

```
    DECLARE s INT DEFAULT 0;
    DECLARE a INT;
    DECLARE continue HANDLER FOR SQLEXCEPTION SET s = 1;
    SET sta = 0;

    START TRANSACTION;
    -- Check whether both accounts exsit
    SELECT count(*) FROM account WHERE account_id = from_account_id or account_id =
to_account_id INTO a;
    IF a != 2 THEN
        SET sta = -2;  -- Error code -2: account does not exist
    END IF;

    IF sta = 0 THEN
        -- Check if the balance is enough
        SELECT balance FROM account WHERE account_id = from_account_id INTO a;
        IF a < amount THEN
            SET sta = -3; -- Error code -3: balance is not enough
        ELSE
            -- Update the balaces of two accounts
            UPDATE account SET balance = balance - amount WHERE account_id =
from_account_id;
            UPDATE account SET balance = balance + amount WHERE account_id =
to_account_id;
        END IF;
    END IF;

    -- Process errors
    IF s = 0 AND sta = 0 THEN
        SET sta = 1;
        COMMIT;
    ELSE
        IF sta = 0 THEN
            SET sta = -9; -- Error code -9: unknown error within MySQL
        END IF;
        ROLLBACK;
    END IF;
END //
delimiter ;
```

首先检查了转入和转出账户是否都存在，然后检查转出账户的余额收否能够满足转账金额。如果两个条件都满足，则修改两个账户的余额，否则设置对应的错误码。这个过程都使用事务来保证转账操作的可靠性和原子操作性。

此外，账户管理的增删改查需要调用以下函数：

来源：`db.py`

```
# ************** Account **************
def account_search(conn, search_text, search_type):
    ...

def account_add(conn, add_account_id, add_customer_id, add_bank_name):
```

```
    cursor = conn.cursor()
    sta = -1
    try:
        ...
    except:
        ...

def account_delete(conn, delete_account_id):
    cursor = conn.cursor()
    sta = -1
    try:
        ...
    except:
        ...

def account_update(conn, old_account_id, new_account_id, new_balance,
new_customer_id, new_bank_name):
    cursor = conn.cursor()
    sta = -1
    try:
        ...
    except:
        ...
```

转账的使用到的MySQL存储过程如下：

来源：`procedures.sql`

```
-- A procedure that changes the id of an account.
DROP PROCEDURE if exists change_account;
delimiter //
CREATE PROCEDURE change_account(IN old_account_id VARCHAR(50), IN new_account_id
VARCHAR(50), IN new_balance DECIMAL(20, 2), IN new_customer_id VARCHAR(50), IN
new_bank_name VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE s INT DEFAULT 0;
    DECLARE a INT;
    DECLARE continue HANDLER FOR SQLEXCEPTION SET s = 1;
    SET sta = 0;

    -- Check whether the new bank exsits
    ...
    -- Check whether the new customer exsits
    ...
    -- Check whether the old account exsits
    ...

    IF sta = 0 THEN
        IF new_account_id = old_account_id THEN
            ...
        ELSE
            -- Check whether the new account exsits. If not update the record.
```

```
            ...
        END IF;
    END IF;

    -- Process errors
    ...

END //
delimiter ;

-- A procedure that deletes an account.
DROP PROCEDURE if exists delete_account;
delimiter //
CREATE PROCEDURE delete_account(IN delete_account_id VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the account exsit
    SELECT count(*) FROM account WHERE account_id = delete_account_id INTO a;
    IF a = 1 THEN
        -- 删除该账户
        DELETE FROM account WHERE account_id = delete_account_id;
        SET sta = 1;
    ELSE
        SET sta = -2; -- Error code -2: account does not exist
    END IF;
END //
delimiter ;

-- A procedure that creates a new account for a customer. Transaction is used to
ensure the atomicity of the operation.
DROP PROCEDURE if exists create_account;
delimiter //
CREATE PROCEDURE create_account(IN add_account_id VARCHAR(50), IN add_customer_id
VARCHAR(50), IN add_bank_name VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE s INT DEFAULT 0;
    DECLARE a INT;
    DECLARE continue HANDLER FOR SQLEXCEPTION SET s = 1;
    SET sta = 0;

    START TRANSACTION;
    -- Check whether the bank exsit
    ...
    -- Check whether the customer exsit
    ...
    -- Check whether the account already exsits
    ...

    -- No problem, insert the new account
    IF sta = 0 THEN
```

```
        INSERT INTO account (account_id, customer_id, bank_name) VALUES
(add_account_id, add_customer_id, add_bank_name);
    END IF;

    -- Process errors
    ...

END //
delimiter ;
```

## 3.8 贷款管理

从主页进入贷款管理，可以对贷款进行增、删、改、查，并且在这里实现了还款的功能。其中还款的实现与转账相似。

来源：`main.py`

```
@app.route('/homepage/loan', methods=['GET', 'POST'])
def loan():
    if request.method == 'GET':
        return render_template('loan.html')
    else:
        # 如果 session 中没有连接，返回登录页面
        if 'username' not in session:
            return redirect(url_for('login'))
        # 如果 session 中有连接，从 session 中获取连接，然后执行后续操作
        username = session['username']
        password = session['password']
        conn = db_login(user=username, password=password)
        if 'SEARCH' in request.form:
            ...
        elif 'ADD' in request.form:
            ...
        elif 'DELETE' in request.form:
            ...
        elif 'UPDATE' in request.form:
            ...
        elif 'REPAY' in request.form:
            repay_loan_id = request.form['repay_loan_id']
            repay_account_id = request.form['repay_account_id']
            repay_amount = request.form['repay_amount']
            res = loan_repay(conn, repay_loan_id, repay_account_id, repay_amount)
            return render_template('loan.html', repay_res=res)
```

转账需要获取转出账户、还款金额与偿还的贷款ID。然后调用处理转账的函数`loan_repay，定义如下：

来源：`db.py`

```
def loan_repay(conn, repay_loan_id, repay_account_id, repay_amount):
    cursor = conn.cursor()
    sta = -1
    try:
        (repay_amount, flag) = num_check_trans(repay_amount)
```

```
        if flag != 1:
            return flag
        if repay_amount <= 0:
            return -13 # Error code -13: repay amount less than or equal to 0
        sta = cursor.callproc('repay_loan', (repay_loan_id, repay_account_id,
repay_amount, sta))[-1]
        conn.commit()
        cursor.close()
        return sta
    except:
        conn.rollback()
        cursor.close()
        return -1
```

下面详细介绍还贷款用到的存储过程`repay_loan`:

```
-- A procedure that repays a loan. Transaction is used to ensure the atomicity of
the operation.
DROP PROCEDURE if exists repay_loan;
delimiter //
CREATE PROCEDURE repay_loan(IN repay_loan_id VARCHAR(50), IN repay_account_id
VARCHAR(50), IN repay_amount DECIMAL(20, 2), OUT sta INT)
BEGIN
    DECLARE s INT DEFAULT 0;
    DECLARE a INT;
    DECLARE continue HANDLER FOR SQLEXCEPTION SET s = 1;
    SET sta = 0;

    START TRANSACTION;
    -- Check whether the loan and account exsit
    SELECT count(*) FROM loan WHERE loan_id = repay_loan_id INTO a;
    IF a != 1 THEN
        SET sta = -2;  -- Error code -2: loan does not exist
    END IF;
    SELECT count(*) FROM account WHERE account_id = repay_account_id INTO a;
    IF a != 1 THEN
        SET sta = -3;  -- Error code -3: account does not exist
    END IF;

    IF sta = 0 THEN
        -- Check if the balance is enough
        SELECT balance FROM account WHERE account_id = repay_account_id INTO a;
        IF a < repay_amount THEN
            SET sta = -4; -- Error code -4: balance is not enough
        ELSE
            -- Update the balaces of the account and the loan
            UPDATE account SET balance = balance - repay_amount WHERE account_id =
repay_account_id;
            UPDATE loan SET unrepayed_amount = unrepayed_amount - repay_amount WHERE
loan_id = repay_loan_id;
        END IF;
    END IF;
```

```
    -- Process errors
    IF s = 0 AND sta = 0 THEN
        SET sta = 1;
        COMMIT;
    ELSE
        IF sta = 0 THEN
            SET sta = -9; -- Error code -9: unknown error within MySQL
        END IF;
        ROLLBACK;
    END IF;
END //
delimiter ;
```

偿还贷款的过程设计为从给定账户中转出给定金额来偿还给定的贷款。与转账的过程相似，存储过程先检查转出账户和贷款ID是否存在。如果存在在检查账户余额是否能满足偿还金额，如果能满足则减少账户金额偿还贷款，同时修改贷款的未偿还金额。如果不能满足要求，返回对应的错误码，并在网页显示对应的错误信息。

下面是贷款的增、删、改、查用到的与数据库交互的函数：

来源：`db.py`

```
# ************** Loan **************
def loan_search(conn, search_text, search_type):
    ...

def loan_add(conn, add_loan_id, add_loan_amount, add_customer_id, add_bank_name):
    ...

def loan_delete(conn, delete_loan_id):
    ...

def loan_update(conn, old_loan_id, new_loan_id, new_loan_amount,
new_unrepayed_amount, new_customer_id, new_bank_name):
    ...
```

增、删、改使用的的MySQL存储过程如下：

来源：`procedures.sql`

```
-- A procedure that changes a loan.
DROP PROCEDURE if exists change_loan;
delimiter //
CREATE PROCEDURE change_loan(IN old_loan_id VARCHAR(50), IN new_loan_id VARCHAR(50),
IN new_loan_amount DECIMAL(20, 2), IN new_unrepayed_amount DECIMAL(20, 2), In
new_customer_id VARCHAR(50), IN new_bank_name VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE s INT DEFAULT 0;
    DECLARE a INT;
    DECLARE continue HANDLER FOR SQLEXCEPTION SET s = 1;
    SET sta = 0;

    -- Check whether the new bank exsits
```

```
    ...
    -- Check whether the new customer exsits
    ...
    -- Check whether the old loan exsits
    ...
    -- Note that the check of nre_loan_amount and new_unrepayed_amount is
complemented in db.py
    IF sta = 0 THEN
        IF new_loan_id = old_loan_id THEN
            UPDATE ...
        ELSE
            -- Check whether the new loan exsits
            SELECT count(*) FROM loan WHERE loan_id = new_loan_id INTO a;
            IF a = 1 THEN
                SET sta = -15; -- Error code -15: New loan already exists
            ELSE
                -- Update
                UPDATE ...
            END IF;
        END IF;
    END IF;

    -- Process errors
    ...

END //
delimiter ;


-- A procedure that creates a new loan for a customer. Transaction is used to ensure
the atomicity of the operation.
DROP PROCEDURE if exists create_loan;
delimiter //
CREATE PROCEDURE create_loan(IN add_loan_id VARCHAR(50), IN add_loan_amount
DECIMAL(20, 2), IN add_customer_id VARCHAR(50), IN add_bank_name VARCHAR(50), OUT
sta INT)
BEGIN
    DECLARE ...

    START TRANSACTION;
    -- Check whether the bank exsit
    ...
    -- Check whether the customer exsits
    ...
    -- Check whether the loan already exsits
    ...

    -- No problem, insert the new loan
    IF sta = 0 THEN
        INSERT INTO loan ...
    END IF;
```

```
    -- Process errors
    ...
END //
delimiter ;


-- A procedure that deletes a loan.
DROP PROCEDURE if exists delete_loan;
delimiter //
CREATE PROCEDURE delete_loan(IN delete_loan_id VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the loan exsit
    SELECT count(*) FROM loan WHERE loan_id = delete_loan_id INTO a;
    IF a = 1 THEN
        -- 删除该贷款
        ...
    ELSE
        SET sta = -2; -- Error code -2: loan does not exist
    END IF;
END //
delimiter ;
```

## 3.9 员工管理

从主页进入员工管理，可以对员工信息进行增删改查。员工的照片是通过存储文件路径实现。一旦导入一个新的图片，就会在指定路径中创建一个同样的副本，并以员工的身份证ID命名。修改图片会在删除旧的图片并存储新的图片。删除员工记录会连带删除文件系统中该员工的照片。

来源：`main.py`

```
@app.route('/homepage/employee', methods=['GET', 'POST'])
def employee():
    if request.method == 'GET':
        return render_template('employee.html')
    else:
        # 如果 session 中没有连接，返回登录页面
        if 'username' not in session:
            return redirect(url_for('login'))
        # 如果 session 中有连接，从 session 中获取连接，然后执行后续操作
        username = session['username']
        password = session['password']
        conn = db_login(user=username, password=password)
        if 'SEARCH' in request.form:
            ...
        elif 'ADD' in request.form:
            save_flag = True
            add_id = request.form['add_id']
            add_name = request.form['add_name']
            # 检查是否有文件在POST请求中
            if 'add_photo' not in request.files:
```

```
            save_flag = False
        file = request.files['add_photo']
        # 如果用户没有选择文件，浏览器也会提交一个没有文件名的空部分
        if file.filename == '':
            save_flag = False
        suffix = file.filename.split('.')[-1]
        add_photo_name = add_id + '.' + suffix
        res = employee_add(conn, add_id, add_name, add_photo_name)
        # 保存文件到指定位置
        if res == 1 and save_flag:
            filename = secure_filename(add_photo_name)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
        return render_template('employee.html', add_res=res)

    elif 'DELETE' in request.form:
        delete_id = request.form['delete_id']
        res = employee_delete(conn, delete_id)
        # 删除文件
        if res == 1:
            # 删除.前是delete_id的文件
            for file in os.listdir(UPLOAD_FOLDER):
                if file.split('.')[0] == delete_id:
                    os.remove(os.path.join(UPLOAD_FOLDER, file))

        return render_template('employee.html', delete_res=res)

    elif 'UPDATE' in request.form:
        ...
        # 检查是否有文件在POST请求中
        if 'new_photo' not in request.files:
            save_flag = False
            new_photo_name = 'special_token_original'
        file = request.files['new_photo']
        # 如果用户没有选择文件，浏览器也会提交一个没有文件名的空部分
        if file.filename == '':
            save_flag = False
            new_photo_name = 'special_token_original'
        else:
            suffix = file.filename.split('.')[-1]
            new_photo_name = new_id + '.' + suffix
        res = employee_update(conn, old_id, new_id, new_name, new_photo_name)
        # 如果成功，保存文件到指定位置，删除旧的
        if res == 1 and save_flag:
            # 删除旧的文件
            for tmp_file in os.listdir(UPLOAD_FOLDER):
                if tmp_file.split('.')[0] == old_id:
                    os.remove(os.path.join(UPLOAD_FOLDER, tmp_file))
            # 保存新的文件
            filename = secure_filename(new_photo_name)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))

        return render_template('employee.html', update_res=res)
```

上面的代码详细展示了和文件系统交互的过程。增添一个新的员工，在数据库插入成功后，将从网页端获取的他的照片存储到指定的地址。照片以员工身份证ID为文件名，这样可以避免文件命名冲突。删除员工的时候同步在文件系统中通过身份证ID索引到对应的照片然后删除。更新照片需要先删除旧的，在保存新的，这也是通过身份证ID索引实现的。这里也支持不上传员工照片，即员工记录的照片属性可以为空，这是通过在提交的照片为空的时候设置照片路径为`special_token_original`，然后在存储过程中特殊处理`special_token_original`实现的。

其中调用的与数据库交互的函数定义如下：

来源：`db.py`

```
# ************** Employee **************
def employee_search(conn, search_text, search_type):
    ...

def employee_add(conn, add_id, add_name, add_path):
    ...

def employee_delete(conn, delete_id):
    ...

def employee_update(conn, old_id, new_id, new_name, new_path):
    ...
```

增、删、改用到的MySQL存储过程如下：

来源：`procedures.sql`

```
-- A procedure that changes the id of a employee.
DROP PROCEDURE if exists change_employee;
delimiter //
CREATE PROCEDURE change_employee(IN old_id VARCHAR(50), IN new_id VARCHAR(50), IN
new_name VARCHAR(50), IN new_path VARCHAR(100), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the employee exsit
    SELECT count(*) FROM employee WHERE id = old_id INTO a;
    IF a = 1 THEN
        IF new_id = old_id THEN
            IF new_path = 'special_token_original' THEN
                SELECT path_to_photo FROM employee WHERE id = old_id INTO new_path;
            END IF;
            UPDATE ...
            SET sta = 1;
        ELSE
            SELECT count(*) FROM employee WHERE id = new_id INTO a;
            IF a = 1 THEN
                SET sta = -4; -- Error code -4: New employee already exists
            ELSE
                -- If new_path is 'special_token_original', keep the old path
                IF new_path = 'special_token_original' THEN
```

```
                    SELECT path_to_photo FROM employee WHERE id = old_id INTO
new_path;
                END IF;
                -- Insert the new employee
                ...
                -- Update the id of all employee_department relationships
                ...
                -- Delete the old employee
                ...
                SET sta = 1;
            END IF;
        END IF;
    ELSE
        SET sta = -2; -- Error code -2: employee does not exist
    END IF;
END //
delimiter ;

-- A procedure that creates a new employee.
DROP PROCEDURE if exists create_employee;
delimiter //
CREATE PROCEDURE create_employee(IN add_id VARCHAR(50), IN add_employee_name
VARCHAR(50), IN add_path_to_photo VARCHAR(100), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the employee exsit
    SELECT count(*) FROM employee WHERE id = add_id INTO a;
    IF a = 0 THEN
        INSERT ...
        SET sta = 1;
    ELSE
        SET sta = -3; -- Error code -3: employee already exists
    END IF;
END //
delimiter ;

-- A procedure that deletes an employee.
DROP PROCEDURE if exists delete_employee;
delimiter //
CREATE PROCEDURE delete_employee(IN del_id VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the employee exsit
    SELECT count(*) FROM employee WHERE id = del_id INTO a;
    IF a = 1 THEN
        -- 删除该员工
        ...
    ELSE
        SET sta = -2; -- Error code -2: employee does not exist
    END IF;
```

```
END //
delimiter ;
```

## 3.10 部门管理

从主页进入部门管理，可以对部门信息进行增删改查。一旦创建了一个新部门，就会在`employee_department`表中自动增加一条记录，内容为“新部门领导ID—新部门名”。同样地，一旦对某个部门进行修改，涉及到领导ID的变更或删除也会自动同步在`employee_department`表中。这是通过触发器Trigger实现的，将会在后面详细绍。

来源：`main.py`

```
@app.route('/homepage/department', methods=['GET', 'POST'])
def department():
    if request.method == 'GET':
        return render_template('department.html')
    else:
        # 如果 session 中没有连接，返回登录页面
        if 'username' not in session:
            return redirect(url_for('login'))
        # 如果 session 中有连接，从 session 中获取连接，然后执行后续操作
        username = session['username']
        password = session['password']
        conn = db_login(user=username, password=password)
        if 'SEARCH' in request.form:
            ...
        elif 'ADD' in request.form:
            ...
        elif 'DELETE' in request.form:
            ...
        elif 'UPDATE' in request.form:
            ...
```

需要说明的是到这里支持对主键或者主键外的任何属性的任何组合的查询，但代码实现上并无特殊之处，只是新增了很多if-else，故不在此处站时代码。调用的与数据库交互的函数定义如下：

来源：`db.py`

```
# ************** Department **************
def department_search(conn, search_text, search_type):
    ...

def department_add(conn, add_id, add_bank, add_depart, add_leader):
    ...

def department_delete(conn, delete_id):
    ...

def department_update(conn, old_id, new_id, new_bank, new_department, new_leader):
    ...
```

增、删、改用到的MySQL存储过程如下：

来源：`procedures.py`

```
-- A procedure that changes the name of a department.
DROP PROCEDURE if exists change_department;
delimiter //
CREATE PROCEDURE change_department(IN old_department_id VARCHAR(50), IN
new_department_id VARCHAR(50), IN new_bank_name VARCHAR(50), IN new_department_name
VARCHAR(50), IN new_leader_id VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE s INT DEFAULT 0;
    DECLARE a INT;
    DECLARE continue HANDLER FOR SQLEXCEPTION SET s = 1;
    SET sta = 0;

    -- Check whether the new leader exsit
   ...
    -- Check whether the new bank exsit
    ...
    -- Check whether the old department exsit
    ...

    IF sta = 0 THEN
        IF new_department_id = old_department_id THEN
            UPDATE ...
        ELSE
            -- Check whether the new department exsit
            ...
            IF a = 1 THEN
                SET sta = -6; -- Error code -6: New department already exists
            ELSE
                -- Delete the old employee_department relationships
                ...
                -- Insert the new department_name
                ...
                -- Update the bank_name and department_name of all
employee_department relationships
                ...
                -- Delete the old department
                ...
            END IF;
        END IF;
    END IF;

    -- Process errors
    ...

END //
delimiter ;


-- A procedure that creates a new department.
DROP PROCEDURE if exists create_department;
delimiter //
```

```
CREATE PROCEDURE create_department(IN add_department_id VARCHAR(50), IN
add_bank_name VARCHAR(50), IN add_department_name VARCHAR(50), IN add_leader_id
VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE s INT DEFAULT 0;
    DECLARE a INT;
    DECLARE continue HANDLER FOR SQLEXCEPTION SET s = 1;

    START TRANSACTION;
    SET sta = 0;
    -- Check whether the department exsit
    ...
    -- Check whether the bank exsit
    ...
    -- Check whether the leader exsit
    ...
    -- No problem, insert the new department
    IF sta = 0 THEN
        INSERT ...
    END IF;

    -- Process errors
    ...

END //
delimiter ;

-- A procedure that deletes a department.
DROP PROCEDURE if exists delete_department;
delimiter //
CREATE PROCEDURE delete_department(IN delete_department_id VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;
    SET sta = 0;

    -- Check whether the department exsit
    SELECT count(*) FROM department WHERE department_id = delete_department_id INTO
a;
    IF a = 1 THEN
        -- 删除该部门
        ...
    ELSE
        IF sta = 0 THEN
            SET sta = -2; -- Error code -2: department does not exist
        END IF;
    END IF;
END //
delimiter ;
```

下面介绍与`employee_department`表同步更新所使用到的触发器Trigger：

来源：`procedures.sql`

```
-- Trigger

-- A trigger that create a new employee_department relationship when a new
department is created.
DROP TRIGGER if exists create_employee_department_trigger;
delimiter //
CREATE TRIGGER create_employee_department_trigger
AFTER INSERT ON department
FOR EACH ROW
BEGIN
    INSERT INTO employee_department (employee_id, department_id) VALUES
(NEW.leader_id, NEW.department_id);
END //
delimiter ;


-- A trigger that change the employee_department relationships when a department is
changed.
DROP TRIGGER if exists change_department_trigger;
delimiter //
CREATE TRIGGER change_department_trigger
AFTER UPDATE ON department
FOR EACH ROW
BEGIN
    UPDATE employee_department SET department_id = NEW.department_id, employee_id =
NEW.leader_id WHERE department_id = OLD.department_id;
END //
delimiter ;
```

由于删除部门的时候连带删除了“领导—部门”的记录，所以只对更新和插入操作定义了触发器。针对插入操作，在`employee_department`表中同步插入一条即可。对更新操作，在`employee_department`表中找到department和领导ID相同的记录进行更新。

## 3.11 员工—部门

从主页进入“员工—部门”管理员工和部门的归属。由于一个员工可能同时在多个部门任职/兼职，一个部门也会有多个员工，所以单独建立一个新表`employee_department`来管理员工的归属问题。

支持查找某个部门的所有员工和某个员工的所有任职部门。增、改、删和其余部分的管理相似。

来源：`main.py`

```
@app.route('/homepage/employee_department', methods=['GET', 'POST'])
def employee_department():
    if request.method == 'GET':
        return render_template('employee_department.html')
    else:
        # 如果 session 中没有连接，返回登录页面
        if 'username' not in session:
            return redirect(url_for('login'))
        # 如果 session 中有连接，从 session 中获取连接，然后执行后续操作
        username = session['username']
```

```
        password = session['password']
        conn = db_login(user=username, password=password)
        if 'SEARCH' in request.form:
            ...
        elif 'ADD' in request.form:
            ...
        elif 'DELETE' in request.form:
            ...
        elif 'UPDATE' in request.form:
            ...
```

调用的与数据库交互的函数定义如下：

来源：`db.py`

```
def employee_department_search(conn, search_text, search_type):
    ...

def employee_department_add(conn, add_employee_id, add_department_id):
    ...

def employee_department_delete(conn, delete_employee_id, delete_department_id):
    ...

def employee_department_update(conn, old_employee_id, old_department_id,
new_employee_id, new_department_id):
    ...
```

增、删、改用到的MySQL存储过程如下：

来源：`procedures.py`

```
-- A procedure that changes the employee_department relationship.
DROP PROCEDURE if exists change_employee_department;
delimiter //
CREATE PROCEDURE change_employee_department(IN old_employee_id VARCHAR(50), IN
old_department_id VARCHAR(50), IN new_employee_id VARCHAR(50), IN new_department_id
VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE s INT DEFAULT 0;
    DECLARE a INT;
    DECLARE continue HANDLER FOR SQLEXCEPTION SET s = 1;
    SET sta = 0;

    -- Check whether the new department exsit
    ...
    -- Check whether the new employee exsit
    ...
    -- Check whether the old employee_department exsit
    ...
    -- Check whether the old department exsit
    ...
    -- Check whether the old employee exsit
```

```
    ...
    -- Check whether the old employee is the leader of the old department
    ...

    IF sta = 0 THEN
        IF new_employee_id != old_employee_id OR new_department_id !=
old_department_id THEN
            -- Check whether the new employee_department exsit
            ...
            ELSE
                -- Update the employee_department relationship
                UPDATE ...
            END IF;
        END IF;
    END IF;

    -- Process errors
    ...

END //
delimiter ;

-- A procedure that creates a new employee_department relationship.
DROP PROCEDURE if exists create_employee_department;
delimiter //
CREATE PROCEDURE create_employee_department(IN add_employee_id VARCHAR(50), IN
add_department_id VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE s INT DEFAULT 0;
    DECLARE a INT;
    DECLARE continue HANDLER FOR SQLEXCEPTION SET s = 1;
    SET sta = 0;

    START TRANSACTION;
    -- Check whether the employee_department exsit
    ...
    -- Check whether the department exsit
    ...
    -- Check whether the employee exsit
    ...

    -- No problem, insert the new department
    IF sta = 0 THEN
        INSERT ...
    END IF;

    -- Process errors
    ...
END //
delimiter ;

-- A procedure that deletes an employee_department relationship.
```

```
DROP PROCEDURE if exists delete_employee_department;
delimiter //
CREATE PROCEDURE delete_employee_department(IN delete_employee_id VARCHAR(50), IN
delete_department_id VARCHAR(50), OUT sta INT)
BEGIN
    DECLARE a INT;

    -- Check whether the employee_department relationship exsit
    SELECT count(*) FROM employee_department WHERE employee_id = delete_employee_id
AND department_id = delete_department_id INTO a;
    IF a = 1 THEN
        -- Check whether the employee is the leader of the department
        ...
        ELSE
            -- 删除该员工-部门关系
            ...
        END IF;
    ELSE
        SET sta = -5; -- Error code -5: employee_department does not exist
    END IF;
END //
delimiter ;
```

# 4 实验与测试

## 4.1 依赖

需要使用以下Python包：

- `flask`：Web应用框架
- `mysql-connector-python`：连接MySQL数据库
- `os`：与文件系统交互
- `werkzeug`：利用`werkzeug.utils.secure_filename`检查文件名是否正确

## 4.2 部署

从GitHub上git-clone该项目

```
git clone https://github.com/psycho-xiong/ustc-database-lab2-2024
```

在MySQL中依次运行`mysql`中的`procedures.sql`和`table_init.sql`。然后在`main.py`中修改`conn = db_login(user=user, password=password)`中传入的参数。

然后运行下面的指令

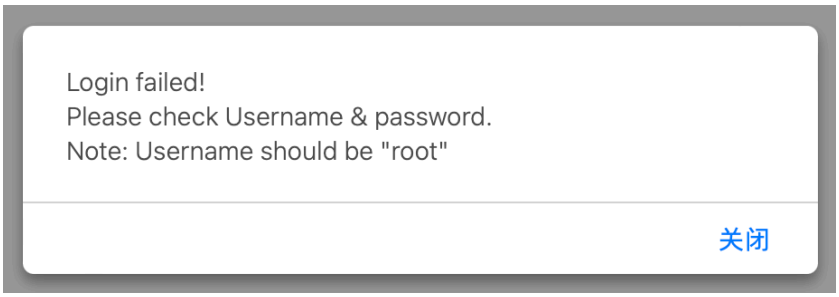
```
cd ustc-database-lab2-2024
python3 main.py
```

然后从浏览器进入http://127.0.0.1:5000即可。

## 4.3 登录

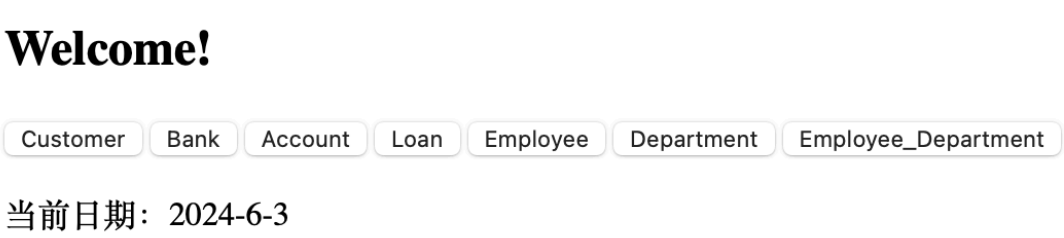
登录界面为：


如果用户/密码错误，弹出下面的弹框




用户/密码正确，进入主页。

## 4.4 主页




点击进入相关管理。

## 4.5 客户管理

返回主页

# 客户管理

## 查询客户

查询内容： 留空显示所有客户 ID 查询

## 增加客户

ID： 姓名： 增加

## 修改客户

原ID： 新ID： 新姓名： 修改

## 删除客户

ID： 删除

## 查询结果

| ID | 姓名 | 总资产 |
| --- | --- | --- |
| 001 | xxq | 10800.00 |
| 002 | lsr | 2000.00 |
| 003 | qwe | 33000.00 |
| 004 | asd | 300.00 |

**查询**：如果查询条件为空，那么将显示所有记录，这一点在所有管理中都实现，之后不再赘述。

查询可以选择按照ID或者姓名查询：


**增加客户**：

| ID | 姓名 | 总资产 |
| --- | --- | --- |
| 001 | xxq | 10800.00 |
| 002 | lsr | 2000.00 |
| 003 | qwe | 33000.00 |
| 004 | asd | 300.00 |
| 005 | jrr | None |

这里因为刚增加的客户还没有账户，所以总资产是“None”。

**修改客户**：


| ID | 姓名 | 总资产 |
| --- | --- | --- |
| 001 | xxq | 10800.00 |
| 002 | lsr | 2000.00 |
| 003 | qwe | 33000.00 |
| 004 | asd | 300.00 |
| 005 | huu | None |

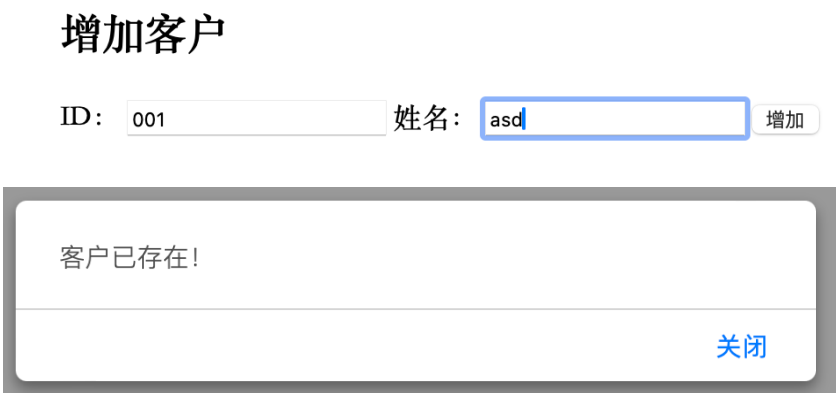
**删除客户**：

# 查询结果

| ID | 姓名 | 总资产 |
| --- | --- | --- |
| 001 | xxq | 10800.00 |
| 002 | lsr | 2000.00 |
| 003 | qwe | 33000.00 |
| 004 | asd | 300.00 |

**错误的操作**都有相应的弹窗：

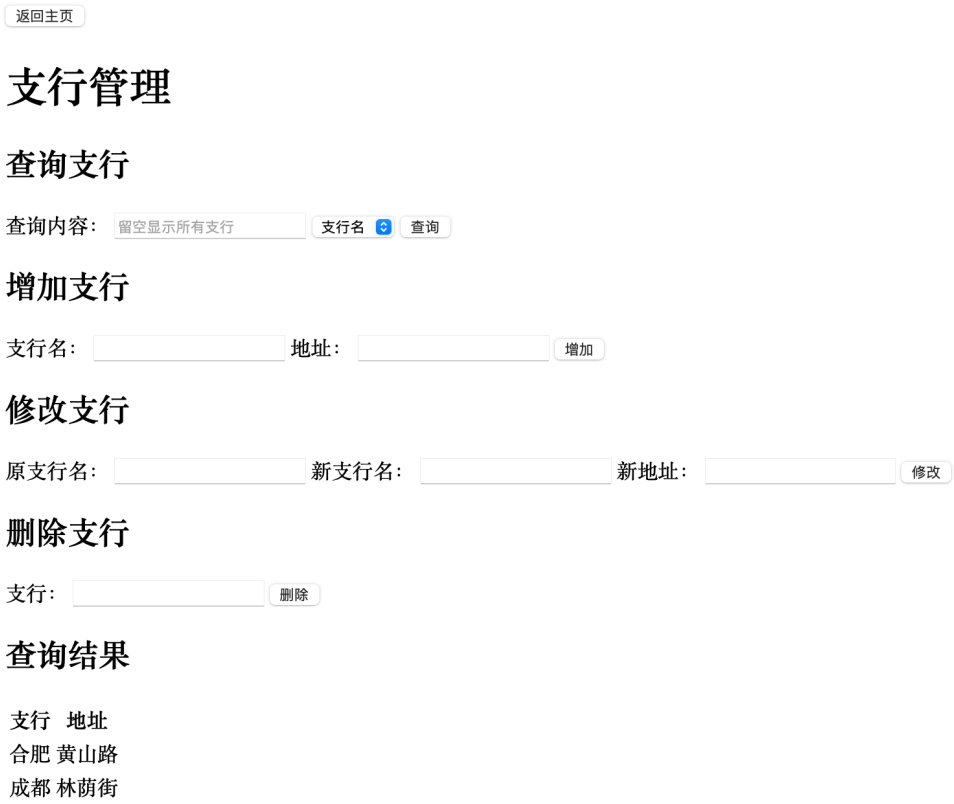
- 新建客户已存在



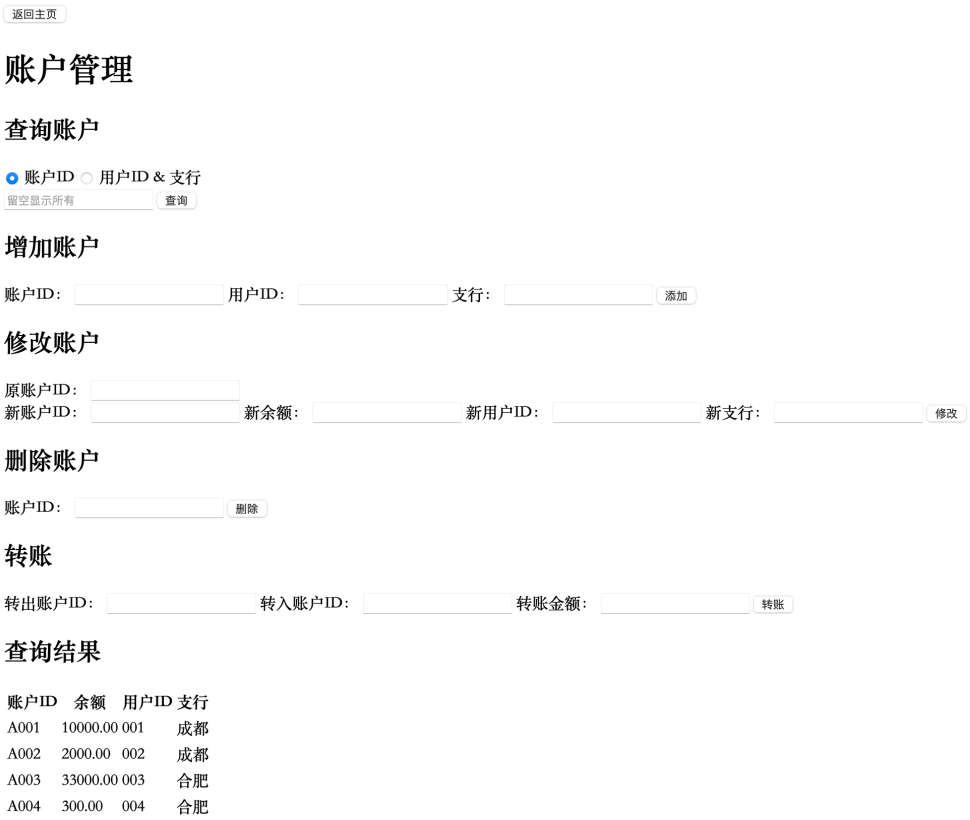

- 修改/删除的客户不存在


其余管理界面与客户管理相同，操作也相同。之后仅展示界面，**如无不同，不再做操作演示。**

## 4.6 支行管理




## 4.7 账户管理




这里的查询与之前的又一个不同之处。查询的键为账户ID（主键），或者另两个的任意组合。其余操作完全相同。

这里如果余额不足，会弹窗提醒：

## 转账

转出账户ID： A001 转入账户ID： A002 转账金额： 1000000 转账


## 4.8 贷款管理

返回主页

### 贷款管理

#### 查询贷款

贷款ID 用户ID & 支行

留空显示所有 查询

#### 增加贷款

贷款ID： 贷款额度 用户ID： 支行： 添加

#### 修改贷款

原贷款ID：

新贷款ID： 新贷款额度： 新未还款额度： 新用户ID： 新支行： 修改

#### 删除贷款

贷款ID： 删除

#### 还款

贷款ID： 还款账户ID： 还款金额： 还款

#### 查询结果

| 贷款ID | 贷款额度 | 未还款金额 | 用户ID | 支行 |
|---|---|---|---|---|
| L001 | 100000.00 | 97000.00 | 001 | 合肥 |
| L002 | 150000.00 | 140000.00 | 002 | 合肥 |

这里如果修改贷款中未还款额度大于总额度的话，会出现报错：

#### 修改贷款

原贷款ID： L001

新贷款ID： L001 新贷款额度： 100000 新未还款额度： 110000 新用户ID： 001 新支行： 合肥 修改


如果输入的额度不是数字的话，也会出现报错：

#### 修改贷款

原贷款ID： L001

新贷款ID： L001 新贷款额度： 100000kk 新未还款额度： 97000 新用户ID： 001 新支行： 合肥 修改


## 4.9 员工管理

返回主页

## 员工管理

### 查询员工

查询内容： 留空显示所有员工 ID 查询

### 增加员工

ID： 姓名： 照片： 选取文件 未选择文件 添加

### 修改员工

原ID： 新ID： 新姓名： 新照片： 选取文件 未选择文件 修改

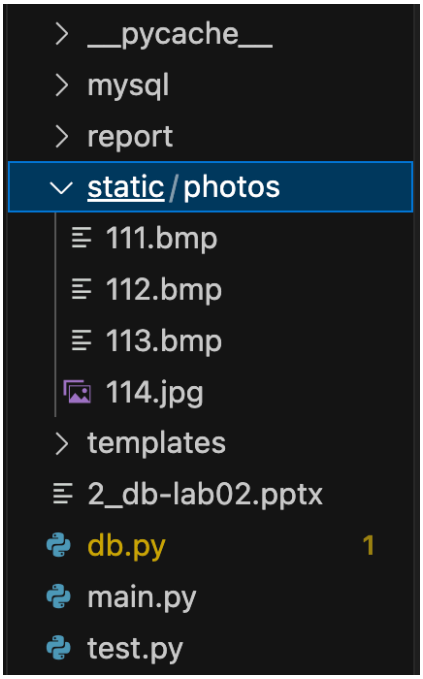
### 删除员工

ID： 删除

### 查询结果

| ID | 姓名 | 照片 |
| --- | --- | --- |
| 111 | xxc |  |
| 112 | lsr |  |

这里可以验证文件系统中存在对应的员工照片 `111.bmp`，`112.bmp` 和 `113.bmp`。

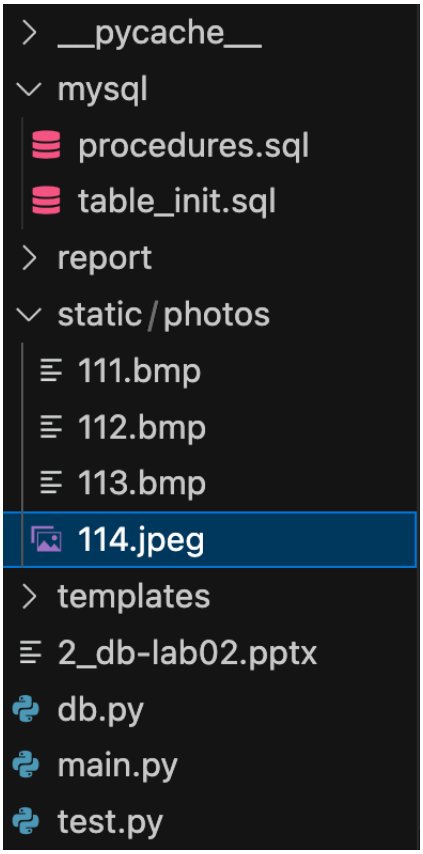



这时添加一个新员工

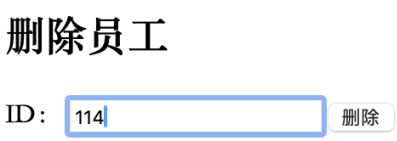
## 增加员工

ID： 114 姓名： qweasd 照片： 选取文件 Tomie.jpeg 添加

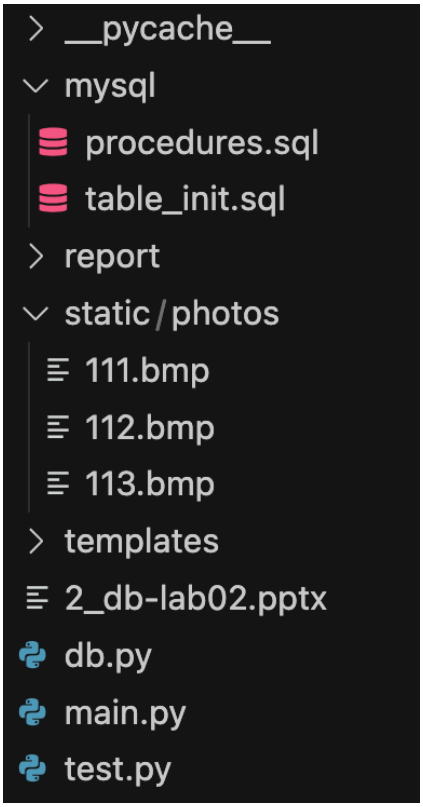
这时文件系统中已经有了新增的员工的照片




然后删除这个员工




这时文件系统中该员工的照片已经被删除



## 4.10 部门管理


这里如果对部门领导ID进行修改，会在“员工—部门”中自动修改，比如：


提交后查看“员工—部门”有：

| 员工ID | 部门ID |
| --- | --- |
| 111 | D2 |

即同步成功。

## 4.11 员工—部门

返回主页

# 员工-部门管理

## 查询员工-部门

查询内容： 留空显示所有 员工ID： 查询

## 增加员工-部门

员工ID： 部门ID： 添加

## 修改员工-部门

原员工ID： 原部门ID：
新员工ID： 新部门ID： 修改

## 删除员工-部门

员工： 部门： 删除

## 查询结果

| 员工ID | 部门ID |
| --- | --- |
| 111 | D2 |
| 113 | D2 |
| 112 | D3 |

这里的设计是不能直接修改领导在其当任领导的部门的“员工-部门”关系：

## 修改员工-部门

原员工ID： 111 原部门ID： D2
新员工ID： 113 新部门ID： D2 修改


所以如果要修改部门的领导，只能在部门管理中修改。

同理不能直接删除领导在其当任领导的部门的“员工-部门”关系：

## 删除员工-部门

员工： 111 部门： D2 删除


即一个部门不能直接删除其领导的“员工-部门”关系。

# 5 总结

通过完成数据库实验，我收获了许多宝贵的经验和知识。在这个实验中，我掌握了数据库的设计、创建、查询和管理等技能。以下是我总结的一些收获：

1. 数据库设计要遵循一定的规范和原则，如规范化、ER图等，以确保数据的一致性和完整性。
2. 在实验过程中，通过解决遇到的问题和错误，提高了自己的调试和排错能力。
3. 学会了用Python等编程语言连接数据库，并进行了数据的增删改查操作，熟悉数据库编程的基本思路和方法。
4. 学习了HTML、CSS和JavaScript等前端基础技术。