

# 《数据库课程设计》

## 实验报告

实验名称： 工资管理系统

班 级： XXXXXXXXXX

组 员： XXXXXXXXXXXXXXXXXX

XXXXXXXXXXXXXXXXXX

指导老师： XXXX

日 期： 2022 年 6 月 15 日

评阅成绩：

教师签名：



# 目 录

引 言.....	1
1 数据库概念模型.....	2
1.1 需求分析.....	2
1.2 概念结构设计.....	2
1.2.1 抽象出实体.....	2
1.2.2 局部 E-R 图.....	3
1.2.3 整体 E-R 图.....	4
2 数据库逻辑模型.....	4
2.1 将 E-R 图转化为关系模式.....	4
2.2 用户子模式建立.....	5
3 数据库物理设计与实施.....	5
3.1 创建数据库.....	5
3.1.1 利用控制命令行创建数据库.....	5
3.1.2 利用客户端程序创建数据库.....	6
3.2 创建和管理基本表.....	7
3.2.1 创建基本表.....	7
3.2.2 管理基本表.....	8
3.2.3 访问数据库.....	11
4 程序设计.....	12
4.1 创建应用程序.....	12
4.2 连接数据库.....	14
4.3 设计应用程序.....	15
5 测试用例.....	15
6 作业总结.....	17
附 录.....	17



# 引 言

## 背景资料

- (1) 某单位现有 1000 名员工，其中有管理人员、财务人员、技术人员和销售人员。
- (2) 该单位下设 4 个科室，即经理室、财务科、技术科和销售科。
- (3) 工资由基本工资、福利补贴和奖励工资构成、失业保险和住房公积金在工资中扣除。
- (4) 每个员工的基本资料有姓名、性别、年龄、单位和职业（如经理、工程师、销售员等）。
- (5) 每月个人的最高工资不超过 3000 元。工资按月发放，实际发放的工资金额为工资减去扣除。

## 基本要求

能实现以下主要功能：

- (1) 实现按照科室录入、修改个人的基本资料、工资和扣除金额的数据。
- (2) 计算个人的实际发放工资。
- (3) 按照课时、职业分类统计人数和工资金额。
- (4) 实现分类查询。
- (5) 能够删除辞职人员的数据。

## 选题意义

为了方便操作管理员或者相关部门负责人对全体员工进行信息管理和工资结算，开发一个可交互的应用程序可以快捷、直观地展现指定员工的个人信息和工资情况，对于人员的调动和信息的修改都可以比较方便地在应用程序上操作，从而间接使用数据库。

# 1 数据库概念模型

## 1.1 需求分析

### 1.1.1 相关管理人员

管理人员的需求：

- (1) 查询员工的个人信息包括员工的工资详情，能够按照指定条件查询指定员工的信息，员工的信息包括姓名、性别、年龄、单位、职位。
- (2) 修改员工的信息，对于单位和职位可以自行添加新栏目；对于员工的工资，能够自己计算工资剩余情况。
- (3) 删除指定员工的信息。
- (4) 添加新员工的个人信息，对于单位和职位，可以供操作者选择，如若有新栏目能够添加新栏目。

## 1.2 概念结构设计

### 1.2.1 抽象出实体

根据需求分析，整个系统设计只需满足管理员即可，我们需要将问题简化为管理员工的个人信息和工资情况两个方向，则需要以下两个实体：

- (1) 员工：属性有姓名、性别、年龄、单位、职位、最终工资，还应该加上员工号这个属性用来唯一标识每个员工的信息。
- (2) 工资信息：属性有基本工资、福利补贴、奖金、失业保险、住房公积金、最终工资，同理还应该加上员工号来唯一标识每个员工的信息。

此外，还应该将两个实体的另一个共同属性—最终工资给联系起来，即在个人信息这个实体中设置一个外键用来参考工资信息实体中的最终工资信息，使其保持一致。

### 1.2.2 局部 E-R 图

根据分析，工资管理系统包含**员工**和**工资信息**两个实体，图中的主键以下划线来表示，对应的 E-R 图如下：

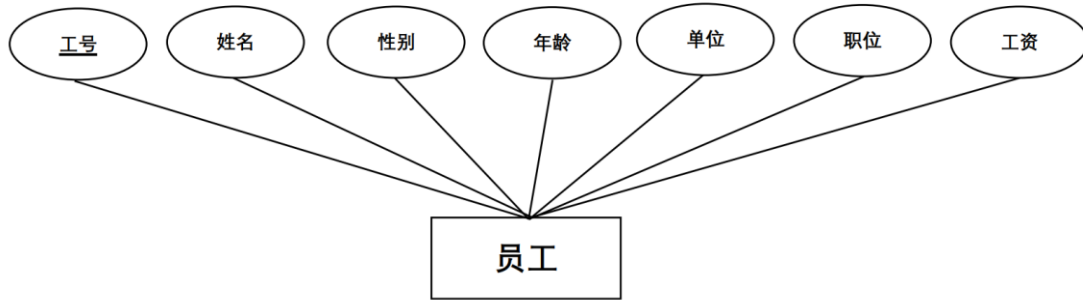


图 1-1 员工实体及其属性

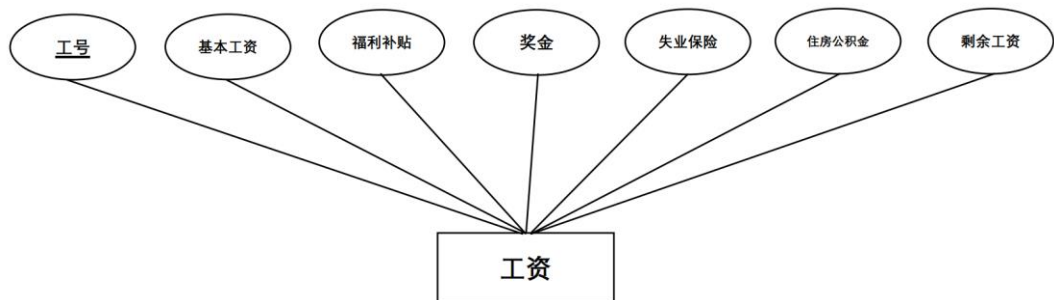


图 1-2 工资实体及其属性

### 1.2.3 整体 E-R 图

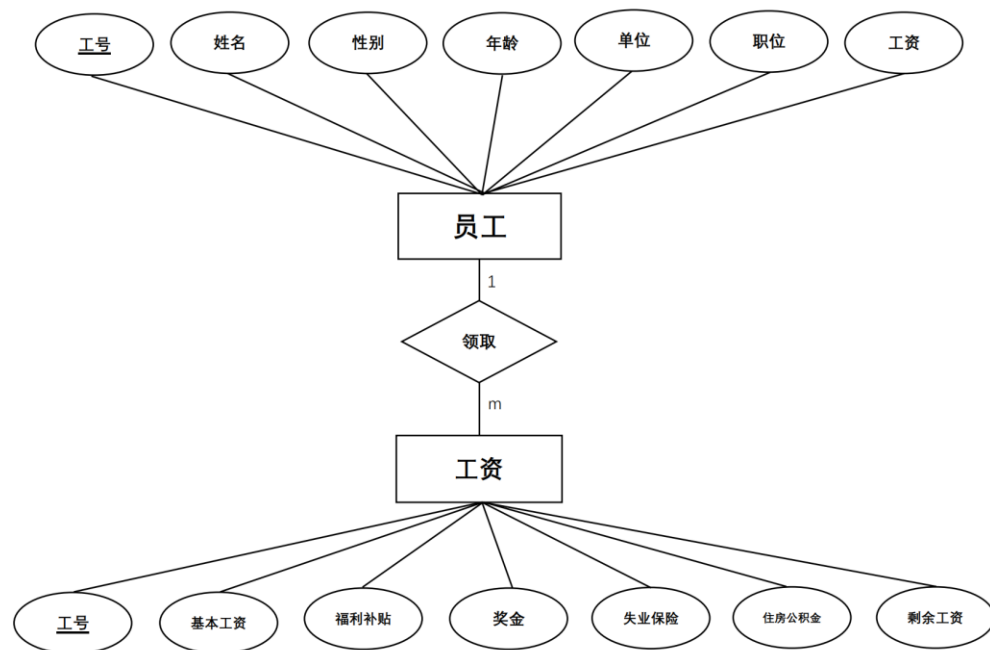


图 2-3 整体 E-R 图

## 2 数据库逻辑模型

### 2.1 将 E-R 图转化为关系模式

根据 E-R 图得出关系模式，其中下划线表示主键，波浪线表示外键：

员工（员工号，姓名，性别，年龄，单位，职位，工资）

工资（员工号，基本工资，福利补贴，奖金，失业保险，住房公积金，剩余工资）



## 2.2 用户子模式建立

为了便于操作者对员工数据进行查询，可以创建一个视图将两个实体表连接起来，即该视图包含员工的所有信息：

员工总信息（员工号，姓名，性别，年龄，单位，职位，基本工资，福利补贴，奖金，失业保险，住房公积金，剩余工资）

这样如果后续需要查看员工所有信息，则不需要频繁利用表间连接，可以提高查询效率。

## 3 数据库物理设计与实施

### 3.1 创建数据库

#### 3.1.1 利用控制命令行创建数据库

在控制命令行页面，输入 `mysql -u [用户名] -p`，然后按照提示输入密码即可进入 MySQL 的控制页面：

```
C:\Users\fish>mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 78
Server version: 8.0.27 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

图 3-1 控制命令行

在 MySQL 控制页则可以使用相关语句进行数据库操作，例如我们创建数据库，利用查询语句 `create database [数据库]` 来创建数据库。

### 3.1.2 利用客户端程序创建数据库

这里利用 MySQL 的管理软件 SQLyog 来演示，官方下载地址：  
<https://sqlyog.en.softonic.com/>。

进入客户端，首先是连接你已经配置好的 MySQL 数据库，如果是本地连接，即可在连接页面 MySQL 地址填写 `localhost`，填写你配置 MySQL 时的用户名和密码，点击连接即可进入主页面。



图 3-2 SQLyog 建立连接

进入主页面，在主页面新建查询，即可在查询语句窗口键入查询语句：

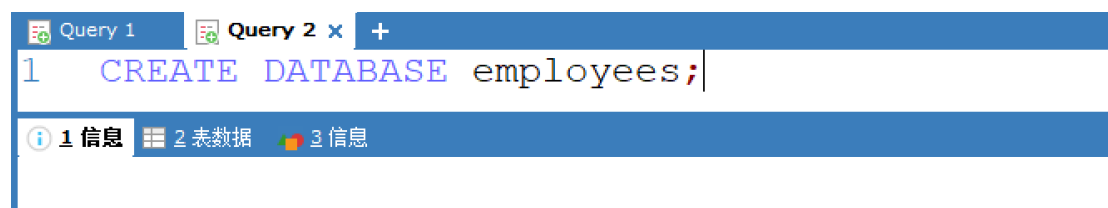


图 3-3 查询窗口

后续的创建过程都在 SQLyog 中进行操作，也在 cmd 控制命令行利用查询语句完成相关操作，原理相同。

## 3.2 创建和管理基本表

### 3.2.1 创建基本表

在客户端里可以快速方便创建基本表，即在右侧栏目打开你创建好的数据库，右键数据库内的表，按照提示即可创建基本表。下面提供利用查询语句来创建基本表，语句如下：

```
CREATE TABLE employees.employee_info (
    employee_id INT NOT NULL,
    name VARCHAR(10),
    sex INT,
    age VARCHAR(10),
    unit VARCHAR(10),
    position VARCHAR(10),
    salary FLOAT DEFAULT 0,
    PRIMARY KEY (employee_id)
);
```

成功创建如图所示，可以执行语句 `desc employee_info`，查看表结构：

1 DESC employee\_info;

Field	Type	Null	Key	Default	Extra
employee_id	int 3B	NO	PRI	(NULL)	OK auto_increment
name	var... 11B	YES		(NULL)	OK
sex	var... 11B	YES		(NULL)	OK
age	int 3B	YES		(NULL)	OK
unit	var... 11B	YES		(NULL)	OK
position	var... 11B	YES		(NULL)	OK
salary	float 5B	NO	PRI	0	1B

图 3-4 查看表结构

在创建基本表之后，我们就可以利用相关语句进行表的增删查改功能，例如查询数据 `SELECT * FROM employee_info` 查询表内所有数据，利用 `INSERT INTO <`

表> VALUES (<值 1>,<值 2>,...)来插入数据等等。

## 3.2.2 管理基本表

### 3.2.2.1 创建表属性的约束

根据需求分析，我们除了要设置主键以外，还得利用一个外键将两个表的工资联系起来，实现语句如下：

```
ALTER TABLE employees.employee_info /*创建外键语句*/  
ADD CONSTRAINT salary_foreignkey FOREIGN KEY (salary) REFERENCES  
employees.salary_info(salary) ON UPDATE CASCADE ON DELETE CASCADE;  
  
ALTER TABLE employees.employee_info /*创建主键语句*/  
ADD PRIMARY KEY (employee_id);
```

这里要注意，在进行外键的创建时，要保证被引用的属性为主键，其他属性的主键、外键的创建与上语句代码类似，实现效果如图 3-4 所示。

### 3.2.2.2 创建存储过程

在进行管理员进入我们的工资管理系统时，我们应该进行必要的登录验证，以保证我们数据的安全性。在进行用户名、密码验证时，我们可以利用数据库的存储过程来验证登录的合法性，这样在代码实现相关程序的时候就可直接调用存储过程，实现效果如下：



图 3-5 存储过程实现效果

因为在此之前，该表中已存在 `[Administor,admin]` 和 `[test,test]` 两个数据项，在我们调用存储过程时即可返回数字 1，如果不匹配则返回数字 0。

对应的实现语句如下：

```
DELIMITER $$
USE employees$$
DROP PROCEDURE IF EXISTS account_judge$$
CREATE DEFINER=root@% PROCEDURE account_judge(IN account VARCHAR(20), IN
password VARCHAR(20))
BEGIN
    DECLARE result INT;
    SELECT COUNT(*) INTO result
    FROM admin
    WHERE admin.account = account
    AND admin.password = password;
    SELECT IF(result>0,1,0);
END$$
DELIMITER ;
```

### 3.2.2.3 创建函数

为了便于操作者在更新数据后统计员工总数，我们可以利用一个函数来实现统计功能，数据库的函数实现与存储过程相类似，效果及实现语句如下所示：



图 3-5 函数实现效果

```

DELIMITER $$
USE employees$$
DROP FUNCTION IF EXISTS total_em$$

CREATE DEFINER=root@% FUNCTION total_em() RETURNS INT
BEGIN
    DECLARE sum INT DEFAULT 0;
    SELECT COUNT(*) INTO sum
    FROM employee_info;
    RETURN sum;
END$$
DELIMITER ;

```

### 3.2.2.4 创建触发器

因为要将两个实体表联系起来，即在对一个表进行更新操作时，如果另外一个表有相同属性应该进行相应更新，这时候就可以使用触发器来实现数据同步的操作，这里以添加信息时的触发器为例，实现语句如下：

```

DELIMITER $$
USE employees$$
DROP TRIGGER /*!50032 IF EXISTS */ data_input$$
CREATE
    /*!50017 DEFINER = 'root'@'localhost' */
    TRIGGER data_input AFTER INSERT ON employee_info
    FOR EACH ROW begin
        insert into salary_info(employee_id,salary)
        values (new.employee_id,new.salary);
end;$$
DELIMITER ;

```

这样，我们在对 **employee\_info** 表进行添加数据时，相应的工资表也将对应新建数据，从而保证数据的统一。

### 3.2.3 访问数据库

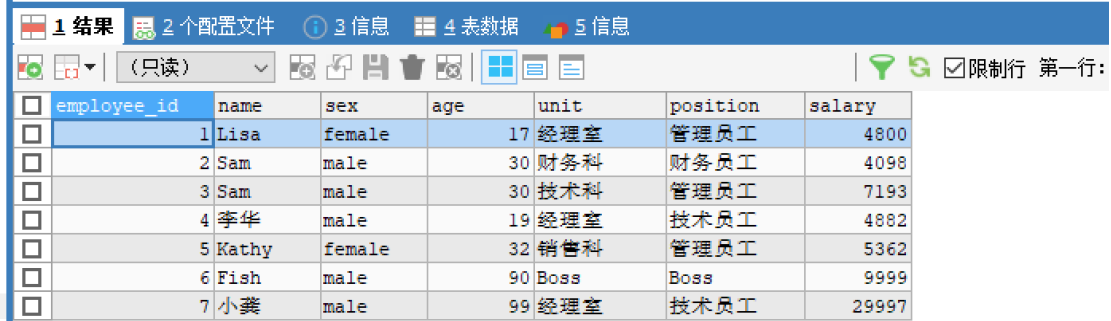
（在 DBMS 上实现对数据库的增、删、查、改操作，过程和结果截图）

至此，我们已经拥有了一个较为完整的数据库，在创建应用程序之前，我们在 SQLyog 上测试相关语句，查看功能是否按照要求实现，测试如下：

#### （1）查询

利用查询语句 `SELECT * FROM employee_info;` 查询表内所有数据：

```
2  SELECT *
3  FROM employee_info;
```



employee_id	name	sex	age	unit	position	salary
1	Lisa	female	17	经理室	管理员工	4800
2	Sam	male	30	财务科	财务员工	4098
3	Sam	male	30	技术科	管理员工	7193
4	李华	male	19	经理室	技术员工	4882
5	Kathy	female	32	销售科	管理员工	5362
6	Fish	male	90	Boss	Boss	9999
7	小葵	male	99	经理室	技术员工	29997

图 3-6-1 查询语句

#### （2）删除

利用控制语句 `DELETE FROM <表名> WHERE <条件>` 完成指定数据项的删除：

```
2  DELETE FROM employee_info
3  WHERE `name`='test';
```



employee_id	name	sex	age	unit	position	salary
8	test		0			0

图 3-6-2 删除语句

执行该语句后，对应的数据项将从实体表中删除，因为我们创建了触发器，对应的工资表相应数据也随之删除。

#### （3）添加

利用控制语句 `INSERT INTO <表名>(col_1,col_2,...) VALUES ('值 1','值 2',...)`

在指定表添加数据:

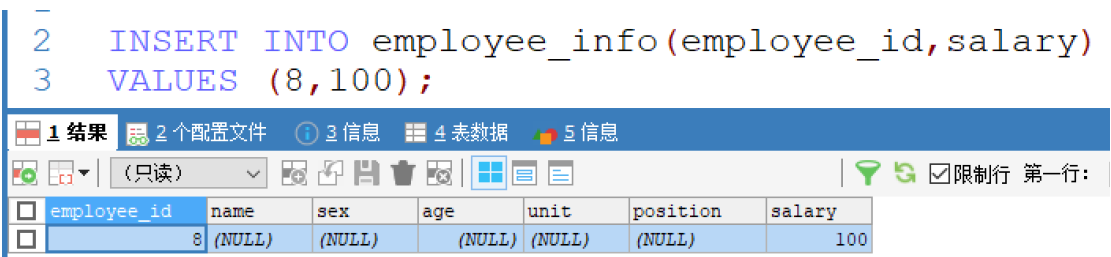


图 3-6-3 添加语句

(4) 更新

利用控制语句 `UPDATE <表名>... SET <col>=<value>... WHERE <条件>` 来对指定表的指定列进行数据更新:

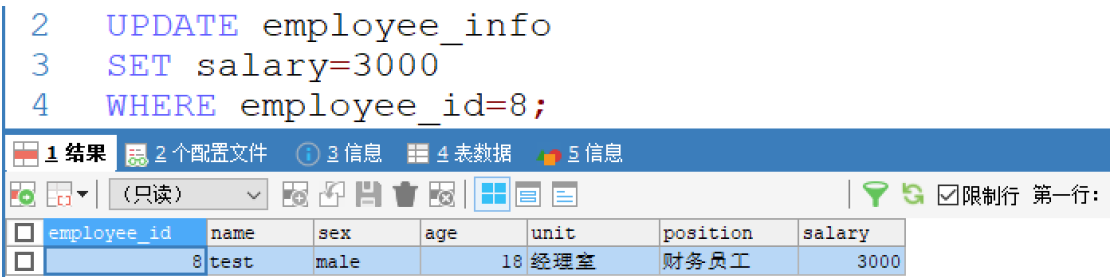


图 3-6-4 更新语句

# 4 程序设计

## 4.1 创建应用程序

为了快速创建一个交互应用程序, 本案例使用了 Python 的 Tkinter 库来创建基本的程序界面窗口, 此外我们还应该创建一个登录窗口保证用户信息只能管理员才能进行更改操作, 实现效果如下:



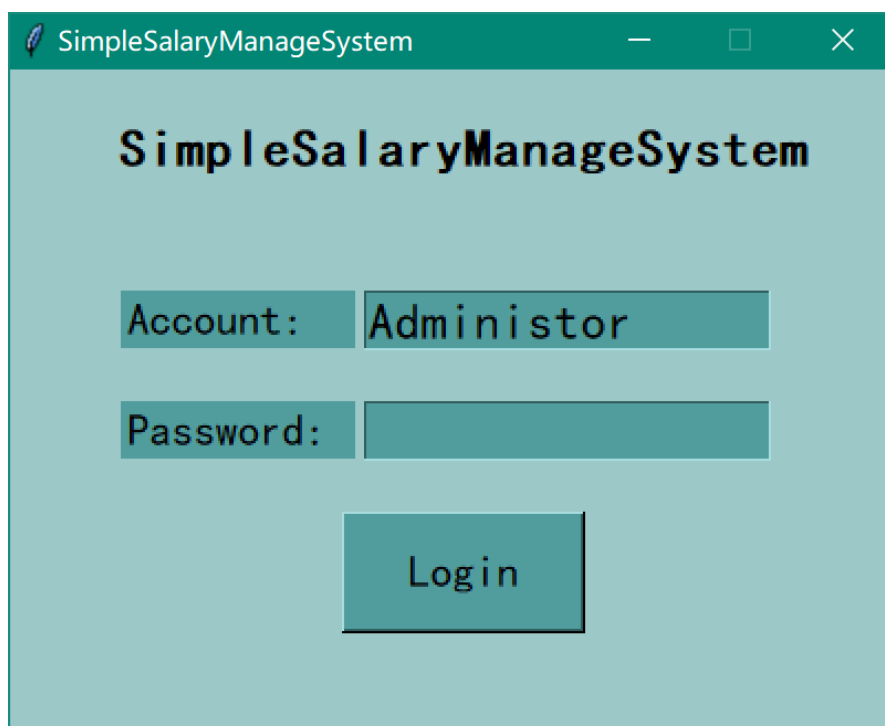


图 4-1-1 登录界面

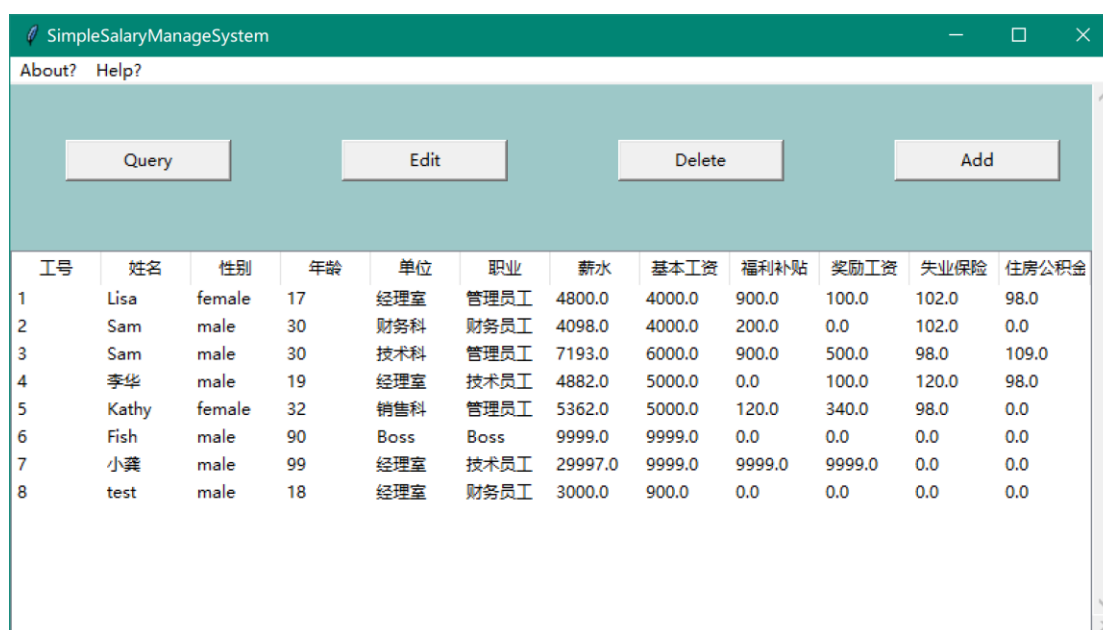


图 4-1-2 功能界面

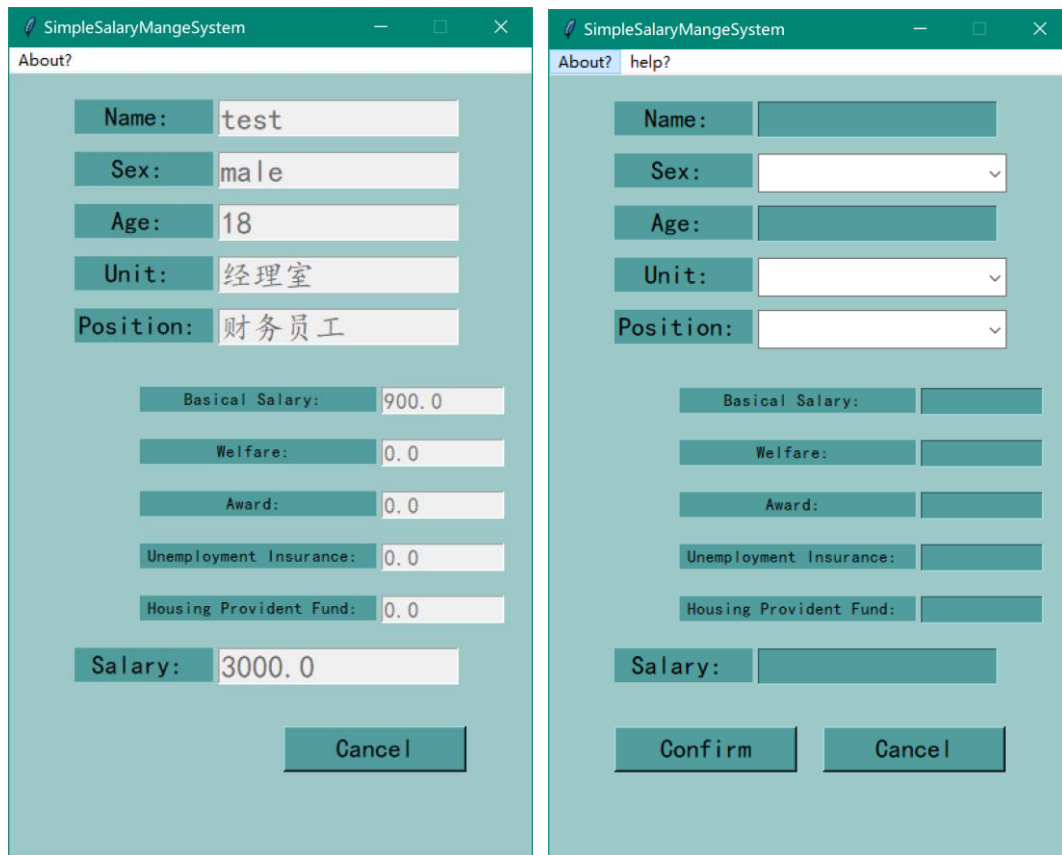


图 4-1-3 员工详情界面和添加界面

## 4.2 连接数据库

Python 程序连接 MySQL 数据库，需要借助第三方库 pyMySQL。安装则需要  
在 python 的安装目录下 bin 文件夹内输入命令 `pip install pyMySQL`，后续则可用 `import pymysql` 导入第三方库，具体的连接过程代码如下：

```
con = pymysql.connect(host=em_host, user=em_user, password=em_password,
database=em_database)
cur = con.cursor()
sql = "<sql 语句>"
cur.execute(sql)
```

这样我们就已经将数据库连接好，后续我们在程序设计过程中对数据库进行交互  
只需要利用游标对象调用 `execute` 方法执行 SQL 语句，然后利用游标对象 `cur` 调用其 `fetchone` 和 `fetchall` 方法获取表中的数据。

## 4.3 设计应用程序

利用编辑器 PyCharm 进行交互界面设计以及相关功能的实现，操作界面效果已在 4-1 进行展示，实现的代码在附录。

## 5 测试用例

测试主要从对员工信息的增、删、查、改方面进行，主要测试过程如下：

(1) 查询功能

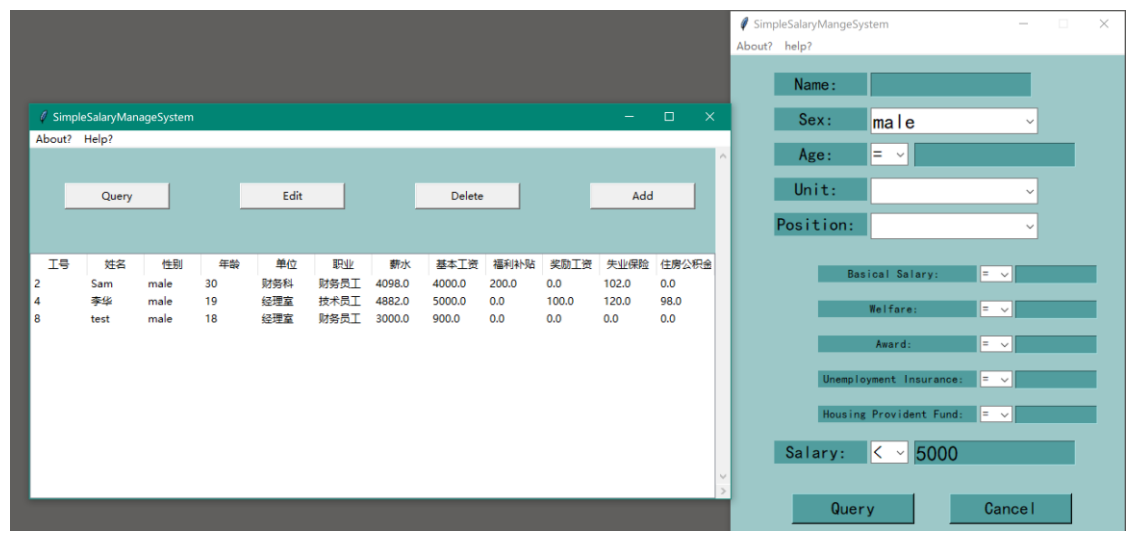
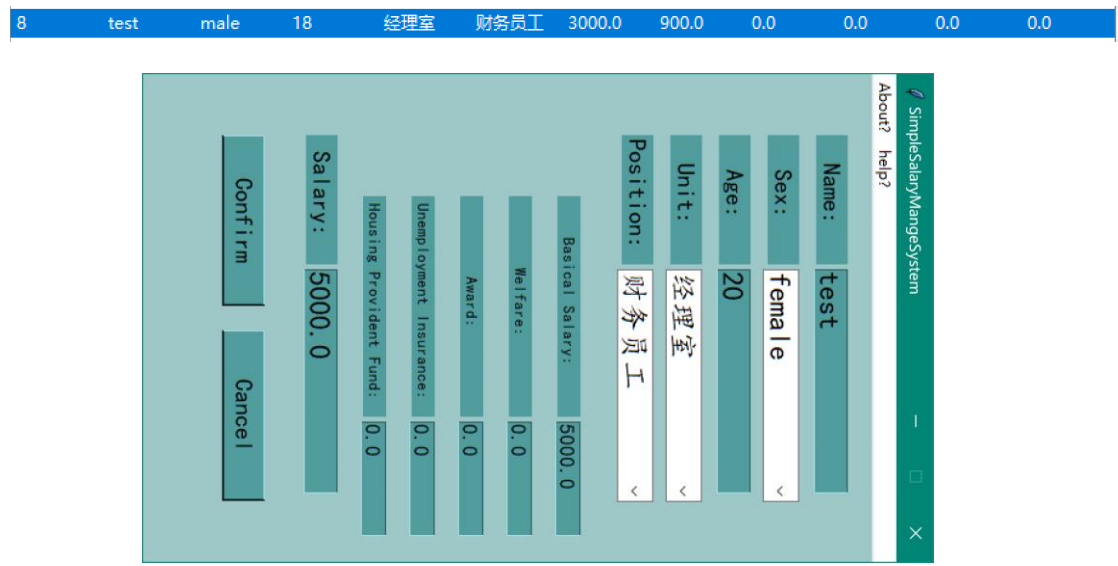


图 5-1 查询功能

(2) 编辑功能



8	test	female	20	经理室	财务员工	5000.0	5000.0	0.0	0.0	0.0	0.0
---	------	--------	----	-----	------	--------	--------	-----	-----	-----	-----

图 5-2 编辑功能

(3) 添加功能

图 5-3 添加功能

(4) 删除功能

工号	姓名	性别	年龄	基本工资	福利补贴	奖励工资	失业保险	住房公积金		
1	Lisa	female	17	0	900.0	100.0	102.0	98.0		
2	Sam	male	30	0	200.0	0.0	102.0	0.0		
3	Sam	male	30	0	900.0	500.0	98.0	109.0		
4	李华	male	19	0	0.0	100.0	120.0	98.0		
5	Kathy	female	32	0	120.0	340.0	98.0	0.0		
6	Fish	male	90	0	0.0	0.0	0.0	0.0		
7	小龚	male	99	0	9999.0	9999.0	0.0	0.0		
8	test	female	20	0	0.0	0.0	0.0	0.0		
9	example	female	21	技术科	技术员工	100.0	100.0	0.0	0.0	0.0

图 5-4 删除功能

## 6 作业总结

纵观本案例的整个流程，主要是在问题中提取出数据库模型，根据需求分析抽象出实体表，在据此设计出整个数据库。然后就是对已学知识进行灵活运用，包括数据库的查询语句的设计、表的设计、各类约束条件、主键和索引、触发器的设计、存储过程、函数等等。然后就是对已学编程知识的运用，将各类学科提及的设计方法、思想融会贯通。

## 附 录

```
#删除功能按钮
def fun_delete(self):
    print("delete employee")
    item = ""
    str = ""
    for str in self.table.selection():
        item = self.table.item(str)
        print(item["values"])
    print("delete id:")
    print(item["values"][0])
    result_ack = messagebox.askyesno(title="确认",message="是否删除 {} 所有信息?".format(item["values"][1]))
    if result_ack == True:
        con = pymysql.connect(host=em_host, user=em_user, password=em_password,
database=em_database)
        cur = con.cursor()
        sql = "delete from employee_info where employee_id = {}".format(item["values"][0])
        cur.execute(sql)
        con.commit()
        cur.close()
        con.close()
        messagebox.showinfo(title="提示",message="删除成功! ")
        self.table.delete(str)
    else :
        pass
```

#编辑按钮的核心功能

```
        con = pymysql.connect(host=em_host, user=em_user,
                                password=em_password, database=em_database)
        cur = con.cursor()

        sql = """update employee_info set
name="{0}",sex="{0}",age={0},unit="{0}",position="{0}",salary={0} where employee_id={0};""".format(
            edit_data[0],\
            edit_data[1],\
            edit_data[2],\
            edit_data[3],\
            edit_data[4],\
            edit_data[10],\
            item[0])
        # ----
        print(sql)
        cur.execute(sql)
        sql = """update salary_info
set
basic_salary={0},welfare={0},award={0},unemployment_insurance={0},housing_provident_fund=
{0},salary={0} where employee_id={0};""".format(\
            edit_data[5],\
            edit_data[6],\
            edit_data[7],\
            edit_data[8],\
            edit_data[9],\
            edit_data[10],\
            item[0])
        # ---
        cur.execute(sql)
        con.commit()
        print(sql)
        cur.close()
        con.close()
```

#添加按钮的核心功能

```
con = pymysql.connect(host=em_host, user=em_user,
password=em_password, database=em_database)
cur = con.cursor()
sql = "select employee_id from employee_info order by employee_id desc
limit 1;"
cur.execute(sql)
final_id = cur.fetchall()[0][0]
print(type(final_id))
final_id += 1
sql = """insert into
employee_info(employee_id,name,sex,age,unit,position,salary)
values({}, '{}', '{}', {}, '{}', '{}', {});""".format(
    final_id, \
    add_data[0], \
    add_data[1], \
    add_data[2], \
    add_data[3], \
    add_data[4], \
    add_data[10])
print(sql)
cur.execute(sql)
sql = """update salary_info
set
basical_salary={},welfare={},award={},unemployment_insurance={},housing_provident_fund=
{},salary={} where employee_id={};""".format(
    add_data[5], \
    add_data[6], \
    add_data[7], \
    add_data[8], \
    add_data[9], \
    add_data[10], \
    final_id)
cur.execute(sql)
con.commit()
print(sql)
cur.close()
con.close()
```

#查询按钮核心功能

#解析下拉框符号

def get\_cmd(cmd\_combobox):

cmd = cmd\_combobox.get()

if cmd=='=':

return '='

elif cmd=='-':

return '-'

elif cmd=='>':

return '>'

elif cmd=='>=':

return '>='

elif cmd=='<':

return '<'

elif cmd=='<=':

return '<='

elif cmd=='=':

return '!='

#根据输入内容修改sql语句 解析输入框内容

cnt = 0

if query\_data[0] != '':

cnt+=1

sql = sql + 'name={}{}' and '.format(query\_data[0])

if query\_data[1] != '':

cnt += 1

sql = sql + 'sex={}{}' and '.format(query\_data[1])

if query\_data[2] != '':

cnt += 1

cmd = get\_cmd(query\_em.combobox\_cmd\_age)

if cmd != '-':

sql = sql + 'age{}{}' and '.format(cmd,query\_data[2])

else:

num = query\_data[2].split('-')

sql = sql + 'age between {} and {} and

'.format(num[0],num[1])

if query\_data[3] != '':

cnt += 1

sql = sql + 'unit={}{}' and '.format(query\_data[3])

if query\_data[4] != '':

cnt += 1

sql = sql + 'position={}{}' and '.format(query\_data[4])

if query\_data[5] != '':

cnt += 1

cmd = get\_cmd(query\_em.combobox\_cmd\_basis)

if cmd != '-':

sql = sql + 'basical\_salary{}{}' and

'.format(cmd,query\_data[5])

else:

num = query\_data[5].split('-')

sql = sql + 'basical\_salary between {} and {} and

'.format(num[0], num[1])

if query\_data[6] != '':

cnt += 1

cmd = get\_cmd(query\_em.combobox\_cmd\_welfare)

if cmd != '-':

sql = sql + 'welfare{}{}' and '.format(cmd,query\_data[6])

else:

num = query\_data[6].split('-')

sql = sql + 'welfare between {} and {} and '.format(num[0],

num[1])

if query\_data[7] != '':

cnt += 1

cmd = get\_cmd(query\_em.combobox\_cmd\_award)

if cmd != '-':

sql = sql + 'award{}{}' and '.format(cmd, query\_data[7])

else:

num = query\_data[7].split('-')

sql = sql + 'award between {} and {} and '.format(num[0],

num[1])

if query\_data[8] != '':

cnt += 1

cmd = get\_cmd(query\_em.combobox\_cmd\_insurance)

if cmd != '-':

sql = sql + 'unemployment\_insurance{}{}' and '.format(cmd,

query\_data[8])

else:

num = query\_data[8].split('-')

sql = sql + 'unemployment\_insurance between {} and {} and

'.format(num[0], num[1])

if query\_data[9] != '':

cnt += 1

cmd = get\_cmd(query\_em.combobox\_cmd\_fund)

if cmd != '-':

sql = sql + 'housing\_provident\_fund{}{}' and '.format(cmd,

query\_data[9])

else:

num = query\_data[9].split('-')

sql = sql + 'housing\_provident\_fund between {} and {} and

'.format(num[0], num[1])

if query\_data[10] != '':

cnt += 1

cmd = get\_cmd(query\_em.combobox\_cmd\_salary)

if cmd != '-':

sql = sql + 'salary{}{}' and '.format(cmd, query\_data[10])

else:

num = query\_data[10].split('-')

sql = sql + 'salary between {} and {} and '.format(num[0],

num[1])

if cnt==0:

sql = sql.rstrip(' where ')

sql = sql + ';

print(sql)

else:

sql = sql.rstrip(' and ')

sql = sql + ';

con = pymysql.connect(host=em\_host, user=em\_user,

password=em\_password, database=em\_database)

cur = con.cursor()

cur.execute(sql)

cur.close()

con.close()



```

/*存储过程：验证用户名、密码*/
DELIMITER $$
USE employees$$
DROP PROCEDURE IF EXISTS account_judge$$
CREATE PROCEDURE account_judge(IN account VARCHAR(20), IN password VARCHAR(20))
BEGIN
    DECLARE result INT;
    SELECT COUNT(*) INTO result
    FROM admin
    WHERE admin.account = account
    AND admin.password = password;
    SELECT IF(result>0,1,0);
END$$
DELIMITER ;

```

```

/*函数：统计员工总数*/
DELIMITER $$
USE employees$$
DROP FUNCTION IF EXISTS total_em$$
CREATE FUNCTION total_em() RETURNS int
begin
    declare sum int default 0;
    select count(*) into sum
    from employee_info;
    return sum;
end$$
DELIMITER ;

```

```

/*触发器 1：删除员工信息实现同步删除*/
DELIMITER $$
USE employees$$
DROP TRIGGER /*!50032 IF EXISTS */ data_delete$$
CREATE
    TRIGGER data_delete AFTER DELETE ON employee_info
    FOR EACH ROW BEGIN
        DELETE FROM salary_info
        WHERE employee_id = old.employee_id;
END;$$
DELIMITER ;

```

/\*触发器 2：在员工表输入数据，实现工资表同步添加工号和工资\*/

DELIMITER \$\$

USE employees\$\$

DROP TRIGGER /\*!50032 IF EXISTS \*/ data\_input\$\$

CREATE

TRIGGER data\_input AFTER INSERT ON employee\_info

FOR EACH ROW BEGIN

INSERT INTO salary\_info(employee\_id,salary)

VALUES (new.employee\_id,new.salary);

END;\$\$

DELIMITER ;

/\*触发器 3：实现员工表和工资表地同步更新\*/

DELIMITER \$\$

USE employees\$\$

DROP TRIGGER /\*!50032 IF EXISTS \*/ data\_update\$\$

CREATE

TRIGGER data\_update AFTER UPDATE ON employee\_info

FOR EACH ROW begin

update salary\_info

set salary\_info.salary = new.salary

where salary\_info.employee\_id = new.employee\_id;

end;\$\$

DELIMITER ;

