数据库系统原理实践报告

项目名称：　数据库系统原理实验报告

姓　　名：

专　　业：

班　　级：

学　　号：

指导教师：

分数	
教师签名	

2020 年 6 月 26 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	
6	

目　录

1 课程任务概述

实验一 软件功能学习

1）练习 SQL Server 或其他某个主流关系数据库管理系统软件的备份方式，要求要有通过数据库的软件功能进行的备份和通过文件形式的脱机备份。

2）练习在新增的数据库上增加用户并配置权限的操作。

实验二 Sql 练习部分

1.建表

2.数据更新

3.数据查询

4.了解系统的查询性能分析功能（选做）

5. DBMS 函数及存储过程和事务（选做）

实验三 数据库应用系统设计

自行选择所擅长的 DBMS 软件以及数据库应用系统（客户端程序或者网站）的程序开发工具，参考后面的题目例子，拟定一个自己感兴趣的数据库应用系统题目，完成该小型数据库应用系统的设计与实现工作。主要内容包括：需求调研与分析、总体设计、数据库设计、详细设计与实现、测试等环节的工作。

2 软件功能学习部分

2.1 任务要求

1）练习 SQL Server 或其他某个主流关系数据库管理系统软件的备份方式，要求要有通过数据库的软件功能进行的备份和通过文件形式的脱机备份。

2）练习在新增的数据库上增加用户并配置权限的操作

2.2 完成过程

实验环境：WINDOWS10

数据库：MYSQL

数据库可视化操作软件：navicat

2.2.1 数据库备份

数据库的备份方式分为冷备份和热备份，冷备份即脱机备份，是在数据库关闭的状况下对数据库进行物理文件级别的备份；热备份需要在联机数据库运行的状态下进行，依赖于数据库自身的相关工具。

冷备份适用于数据规模较大的数据库文件，由于恢复是在文件级别进行的，所以恢复的速度相对较快，但对于备份前后的数据库版本和机器版本有要求。热备份是进行了数据库重新构建的整个过程，恢复速度较慢，但对于恢复的场景没有严格要求，即允许在不同的数据库或主机上进行恢复操作。

下面以 mysql 数据库的两种备份及恢复方式进行说明：

1）数据库软件备份（热备份）：

执行命令：`mysqldump -u root -p train > d:/train_backup.sql`

将 train 数据库备份

接下来删除数据库 train

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| train              |
+--------------------+
5 rows in set (0.00 sec)

mysql> drop database train;
Query OK, 7 rows affected (5.28 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)
```

图 2.1 删除数据库

再导入数据库备份 train_backup.sql

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql> create database train;
Query OK, 1 row affected (0.20 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| train              |
+--------------------+
5 rows in set (0.00 sec)
```

图 2.2 备份数据库

执行：mysql -uroot -p train < d:/train_backup.sql

导入结果：

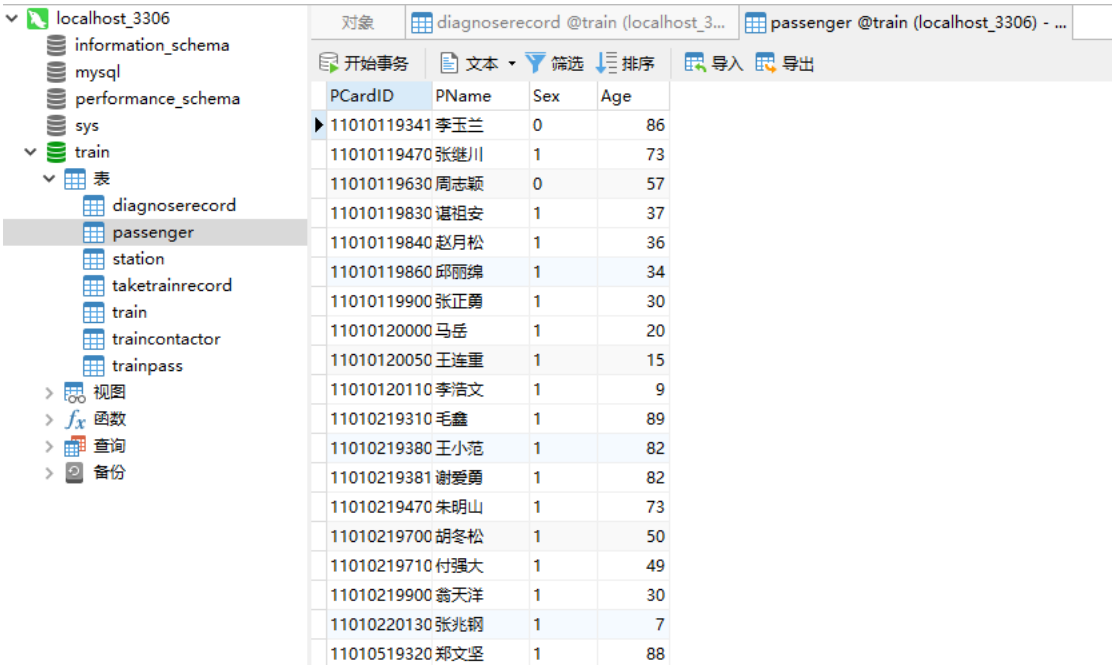

图 2.3 恢复数据库

可见数据库备份恢复成功

2）脱机备份（冷备份）

先停掉 mysql 数据库：

备份需要的数据库目录下的表文件

备份 ib_logfile0，ib_logfile1 配置文件

2.2.2 新增数据库用户并配置权限

查看 mysql 的配置信息：desc mysql.user;

```
mysql> desc mysql.user;
+-----------------------+-----------------------------------+------+-----+---------------
| Field                 | Type                              | Null | Key | Default
+-----------------------+-----------------------------------+------+-----+---------------
| Host                  | char(255)                         | NO   | PRI |
| User                  | char(32)                          | NO   | PRI |
| Select_priv           | enum('N','Y')                     | NO   |     | N
| Insert_priv           | enum('N','Y')                     | NO   |     | N
| Update_priv           | enum('N','Y')                     | NO   |     | N
| Delete_priv           | enum('N','Y')                     | NO   |     | N
| Create_priv           | enum('N','Y')                     | NO   |     | N
| Drop_priv             | enum('N','Y')                     | NO   |     | N
| Reload_priv           | enum('N','Y')                     | NO   |     | N
| Shutdown_priv         | enum('N','Y')                     | NO   |     | N
| Process_priv          | enum('N','Y')                     | NO   |     | N
| File_priv             | enum('N','Y')                     | NO   |     | N
| Grant_priv            | enum('N','Y')                     | NO   |     | N
| References_priv       | enum('N','Y')                     | NO   |     | N
```

图 2.4 配置信息

其中的 Host,User,Password 是登录时候需要输入的参数，列名为*_priv 的列对应的是增、删、改、建表、删表等权限。

本数据库的相关信息：

```
mysql> select host,user,authentication_string from mysql.user;
+-----------+------------------+-------------------------------------------------------------------------+
| host      | user             | authentication_string                                                   |
+-----------+------------------+-------------------------------------------------------------------------+
| localhost | mysql.infoschema | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| localhost | mysql.session    | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| localhost | mysql.sys        | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| localhost | root             | *C8EBD1487A05851EBDFA870FC31CBBB4D536053B                               |
+-----------+------------------+-------------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

图 2.5 数据库信息

创建新用户的步骤：

//只允许指定 ip 连接

create user ’新用户名’@’localhost’ identified by ’密码’;

//允许所有 ip 连接（用通配符%表示）

create user ’新用户名’@’%’ identified by ’密码’;

eg：create user ’testuser’ @ ’%’ identified by ’123456’;

```
mysql> select user,host from mysql.user;
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| testuser         | %         |
| mysql.infoschema | localhost |
| mysql.session    | localhost |
| mysql.sys        | localhost |
| root             | localhost |
+------------------+-----------+
5 rows in set (0.00 sec)
```

图 2.6 查看 user

添加用户 testuser 成功

登录用户 testuser 并查看其权限：

```
C:\Users\朱志成>mysql -u testuser -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16
Server version: 8.0.20 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

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> show grants;
+-------------------------------------+
| Grants for testuser@%               |
+-------------------------------------+
| GRANT USAGE ON *.* TO `testuser`@`%` |
+-------------------------------------+
1 row in set (0.00 sec)
```

图 2.7 查看用户权限

可以看到新创建的 testuser 用户没有其他的权限

为新用户授权基本格式如下：

grant all privileges on 数据库名.表名 to'新用户名'@'指定 ip' identified by '新用户密码' ;

//示例

```
mysql> grant all privileges on train.* to `testuser`@`%` with grant option;
Query OK, 0 rows affected (0.43 sec)

mysql>
```

图 2.8 用户授权

//允许访问所有数据库 train 下的所有表

再次查看权限：

```
mysql> show grants;
+-----------------------------------------------------------------------+
| Grants for testuser@%                                                 |
+-----------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `testuser`@`%`                                  |
| GRANT ALL PRIVILEGES ON `train`.* TO `testuser`@`%` WITH GRANT OPTION |
+-----------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

图 2.9 查看用户权限

设置用户操作权限：

//设置查询权限

```
mysql> grant select on *.* to 'testuser'@'%' with grant option;
Query OK, 0 rows affected (0.42 sec)

mysql>
```

图 2.10 增加查询权限

再次查看权限：

```
mysql> show grants;
+-----------------------------------------------------------------------+
| Grants for testuser@%                                                 |
+-----------------------------------------------------------------------+
| GRANT SELECT ON *.* TO `testuser`@`%` WITH GRANT OPTION               |
| GRANT ALL PRIVILEGES ON `train`.* TO `testuser`@`%` WITH GRANT OPTION |
+-----------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

图 2.11 查看用户权限

//其它操作权限 select 查询 insert 插入 delete 删除 update 修改类似

取消用户查询的查询权限

REVOKE select ON what FROM '新用户名';

```
mysql> revoke select on *.* from 'testuser';
Query OK, 0 rows affected (0.41 sec)
```

图 2.12 移除查询权限

再次查看权限：

```
mysql> show grants;
+-----------------------------------------------------------------------+
| Grants for testuser@%                                                 |
+-----------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `testuser`@`%` WITH GRANT OPTION                |
| GRANT ALL PRIVILEGES ON `train`.* TO `testuser`@`%` WITH GRANT OPTION |
+-----------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

图 2.13 查看用户权限

修改后需要刷新权限：

FLUSH PRIVILEGES;

删除用户：

DROP USER username@localhost;

2.3 任务总结

最初是在 DM 数据库系统上进行的相关实验，但考虑到后续实验三系统开发的相关环境，中途改用了 Mysql 进行相关的实验以提前熟悉数据库的相关操作。

最初对实验要求中的两种备份方式不是很了解，在查阅了相关资料后有了进一步的认识，顺利完成两种备份及恢复。

mysql 的一些操作和 DM 数据库也有些许区别，在刚开始使用的过程中也出现了一些不习惯的地方，后续慢慢适应。

3 Sql 练习部分

3.1 任务要求

1）建表

2）数据更新

3）数据查询

4）了解系统的查询性能分析功能（选做）

5）DBMS 函数及存储过程和事务（选做）

3.2 完成过程

3.2.1 建表

1)定义相关表

(a)车站表【车站编号，车站名，所属城市】

Station (SID int, SName char(20), CityName char(20))

其中，主码为车站编号。

```
create table Station(
    SID int primary key,
    SName char(20) not null,
    CityName char(30)
    );
```

(b)车次表【列车流水号，发车日期，列车名称，起点站编号，终点站编号，开出时刻，终点时刻】

Train (TID int, SDate date, TName char(20), SStationID int, AStationID int, STime datetime, ATime datetime)

其中，TID 为主码，(列车名称，发车日期)为候选码；SStationID 和 AStationID 都来源于车站表的 SID。

（注：车次表中的列车名称是指一般意义上的车次（如 G50），同一个车次在多个不同的日期发车，对应的是不同的记录，这些记录的列车流水号是不一样的）

```
create table Train(
    TID int primary key,
    SDate date not null,
    TName char(20) not null,
    SStationID int ,
    AStationID int ,
    Stime datetime ,
    ATime datetime ,
```

```
    foreign key(SStationID) references Station(SID),
    foreign key(AStationID) references Station(SID),
    unique(SDate,TName)
    );
```

(c)车程表【列车流水号，车站序号，车站编号，到达时刻，离开时刻】

TrainPass (TID int, SNo smallint, SID int, STime datetime, ATime datetime)

其中，主码为(列车编号，车站序号)。SID 来源于车站表的 SID。

```
create table TrainPass(
    TID int not null,
    SNo smallint not null,
    SID int ,
    STime datetime ,
    ATime datetime ,
    primary key(TID,SNo),
    foreign key(SID) references Station(SID),
    foreign key(TID) references Train(TID)
    );
```

(d)乘客表【乘客身份证号，姓名，性别，年龄】

Passenger (PCardID char(18), PName char(20), Sex bit, Age smallint)

其中，主码为乘客身份证号；性别取值为 0/1（“1”表示“男”，“0”表示“女”）。

```
create table Passenger(
    PCardID char(18) not null primary key,
    PName char(20),
    Sex bit,
    Age smallint
);
```

(e)乘车记录表【记录编号，乘客身份证号，列车编号，发车日期，出发站编号，到达站编号，车厢号，席位排号，席位编号，席位状态】

TakeTrainRecord (RID int, PCardID char(18), TID int, SDate date, SStationID int, AStationID int, CarrigeID smallint, SeatRow smallint，SeatNo char(1)，SStatus int)

其中，主码、外码请依据应用背景合理定义。

CarrigeID 若为空，则表示“无座”；

SeatNo 只能取值为’A’、’B’、’C’、’E’、’F’，或为空值；

SStatus 只能取值’0’（退票）、’1’（正常）、’2’（乘客没上车）。

```
create table TakeTrainRecord(
    RID int not null primary key,
    PCardID char(18) ,
```

```
    TID int,
    -- SDate date,
    SStationID int ,
    AStationID int ,
    CarrigeID smallint,
    SeatRow smallint,
    SeatNo char(1) check(SeatNo='A' or SeatNo='B' or SeatNo='C' or SeatNo='D' or
SeatNo='E' or SeatNo is null),
    SStatus int check(SStatus between 0 and 2),
    foreign key(PCardID) references Passenger(PCardID),
    foreign key(TID) references Train(TID),
    foreign key(SStationID) references Station(SID),
    foreign key(AStationID) references Station(SID)
    );
```

(f)诊断表【诊断编号，病人身份证号，诊断日期，诊断结果，发病日期】

DiagnoseRecord (DID int, PCardID char(18), DDay date, DStatus smallint, FDay date)

其中，主码为 DID；DStatus 包括：1：新冠确诊；2：新冠疑似；3：排除新冠

```
create table DiagnoseRecord(
    DID int not null primary key,
    PCardID char(18),
    DDay date,
    DStatus smallint check(DStatus between 1 and 3),
    FDay date,
    foreign key(PCardID) references Passenger(PCardID)
    );
```

(g)乘客紧密接触者表【接触日期, 被接触者身份证号，状态，病患身份证号】

TrainContactor (CDate date, CCardID chsar(18), DStatus smallint, PCardID char(18))

其中，主码为全码。DStatus 包括：1：新冠确诊；2：新冠疑似；3：排除新冠

```
create table Traincontactor(
    CDate date,
    CCardID char(18) references Passenger(PCardID),
    DStatus smallint not null check(DStatus between 1 and 3),
    PCardID char(18) not null,
    primary key(CDate,CCardID,DStatus,PCardID),
    foreign key(CCardID) references Passenger(PCardID),
    foreign key(PCardID) references Passenger(PCardID)
    );
```

2)数据准备

Station 和 passenger 表来自于老师给的数据包，直接导入即可：

对于 Train 表和 TakePass 表，由于车次和车次时刻表中使用的是车站名而非车站 id，所以此处需要对表中的数据进行中转，先建立中转表 train_temp 和 takepass_temp 表，之后利用存储过程生成

```
-- train 表生成
INSERT INTO train SELECT
TID,
@date := DATE_ADD( "2020-01-01", INTERVAL FLOOR( rand()* 30 ) DAY ) AS SDate,
TName,
s1.SID AS SStationID,
s2.SID AS AStationID,
CONCAT( @date, ' ', STime ) AS STime,
CONCAT( @date, ' ', ATime ) AS ATime
FROM
	train_temp,
	station s1,
	station s2
WHERE
	SStationName = s1.SName
	AND AStationName = s2.SName
ORDER BY
	TID;

-- trainpass 表生成
INSERT INTO trainpass SELECT
TID,
SNo,
SID,
CONCAT( SDate, ' ', trainpass_temp.STime ) AS STime,
CONCAT( SDate, ' ', trainpass_temp.ATime ) AS ATime
FROM
	trainpass_temp,
	train,
	station
```

```
WHERE
    trainpass_temp.TName = train.TName
    AND trainpass_temp.SName = station.SName
ORDER BY
    TID,
    SNo;
```

其他三个表先在 excel 中设计好需要的数据导入对应表即可。设计了用于测试查询 sql 的对应数据，冗余数据较少

3)完整性验证

验证在建立外码时是否一定要参考被参照关系的主码

对于 TRAIN 表，其 SSTAITIONID 和 ASTATIONID 均为外码，尝试向其插入一条未参照 STATION 表 STATIONID 的数据：

```
INSERT INTO train.train
VALUES
  ( 1, '2020-5-1', 'insesr_exp', '100000', '30', '2020-5-1 01:53:55', '2020-5-1 03:53:55' );
```

执行结果为：

```
INSERT INTO train.train
VALUES
        ( 1, '2020-5-1', 'insesr_exp', '100000', '30', '2020-5-1 01:53:55', '2020-5-1 03:53:55' )
> 1452 - Cannot add or update a child row: a foreign key constraint fails (`train`.`train`, CONSTRAINT `train_ibfk_1` FOREIGN
KEY (`SStationID`) REFERENCES `station` (`SID`))
> 时间: 0.052s
```

执行失败，因为此处的 SSID 100000 不满足参照性关系，不在 SATATION 表中将 100000 改为 100：

```
INSERT INTO train.train
VALUES
        ( 1, '2020-5-1', 'insesr_exp', '100', '30', '2020-5-1 01:53:55', '2020-5-1 03:53:55' )
> Affected rows: 1
> 时间: 0.41s
```

执行成功：

TID	SDate	TName	SStationID	AStationID	Stime	ATime
1	2020-05-01	insesr_exp	100	30	2020-05-01 01:53:55	2020-05-01 03:53:55

图 3.1 sql 执行结果

实验说明：外码一定是被参照表的主码，当外键不是主键时会触发 error

3.2.2 数据更新

1）分别用一条 sql 语句完成对乘车记录表基本的增、删、改的操作

选择空的 train 表进行操作

增：

```sql
INSERT INTO train.train
VALUES
  ( 1, '2020-5-1', 'exp1', '100', '10', '2020-5-1 01:53:55', '2020-5-1 11:53:55' ),
  ( 2, '2020-5-2', 'exp2', '200', '20', '2020-5-2 02:53:55', '2020-5-2 12:53:55' ),
  ( 3, '2020-5-3', 'exp3', '300', '30', '2020-5-3 03:53:55', '2020-5-3 13:53:55' ),
  ( 4, '2020-5-4', 'exp4', '400', '40', '2020-5-4 04:53:55', '2020-5-4 14:53:55' ),
  ( 5, '2020-5-5', 'exp5', '500', '50', '2020-5-5 05:53:55', '2020-5-5 15:53:55' );
```

图 3.2 sql 执行结果

执行结果：

TID	SDate	TName	SStationID	AStationID	Stime	ATime
1	2020-05-01	exp1	100	10	2020-05-01 01:53:55	2020-05-01 11:53:55
2	2020-05-02	exp2	200	20	2020-05-02 02:53:55	2020-05-02 12:53:55
3	2020-05-03	exp3	300	30	2020-05-03 03:53:55	2020-05-03 13:53:55
4	2020-05-04	exp4	400	40	2020-05-04 04:53:55	2020-05-04 14:53:55
5	2020-05-05	exp5	500	50	2020-05-05 05:53:55	2020-05-05 15:53:55

图 3.3 sql 执行结果

删：

```sql
DELETE FROM train.train WHERE TID = 5
> Affected rows: 1
> 时间: 0.441s
```

执行结果：

TID	SDate	TName	SStationID	AStationID	Stime	ATime
1	2020-05-01	exp1	100	10	2020-05-01 01:53:55	2020-05-01 11:53:55
2	2020-05-02	exp2	200	20	2020-05-02 02:53:55	2020-05-02 12:53:55
3	2020-05-03	exp3	300	30	2020-05-03 03:53:55	2020-05-03 13:53:55
4	2020-05-04	exp4	400	40	2020-05-04 04:53:55	2020-05-04 14:53:55

图 3.4 sql 执行结果

改：

```sql
UPDATE train.train set SStationID = 1000 WHERE TID = 4
> Affected rows: 1
> 时间: 0.147s
```

执行结果：

TID	SDate	TName	SStationID	AStationID	Stime	ATime
1	2020-05-01	exp1	100	10	2020-05-01 01:53:55	2020-05-01 11:53:55
2	2020-05-02	exp2	200	20	2020-05-02 02:53:55	2020-05-02 12:53:55
3	2020-05-03	exp3	300	30	2020-05-03 03:53:55	2020-05-03 13:53:55
4	2020-05-04	exp4	1000	40	2020-05-04 04:53:55	2020-05-04 14:53:55

图 3.5 sql 执行结果

2）批处理操作

将乘车记录表中的从武汉出发的乘客的乘车记录插入到一个新表 WH_TakeTrainRecord 中。

执行 sql：

```
create table WH_TakeTrainRecord(
    RID int not null primary key,
    PCardID char(18) ,
    TID int,
    SStationID int ,
    AStationID int ,
    CarrigeID smallint,
    SeatRow smallint,
    SeatNo char(1) check(SeatNo='A' or SeatNo='B' or SeatNo='C' or SeatNo='D' or SeatNo='E' or SeatNo is null),
    SStatus int check(SStatus between 0 and 2),
    foreign key(PCardID) references Passenger(PCardID),
    foreign key(TID) references Train(TID),
    foreign key(SStationID) references Station(SID),
    foreign key(AStationID) references Station(SID)
    );

insert into WH_TakeTrainRecord
SELECT * from taketrainrecord
where SStationID in (SELECT SID FROM station WHERE CityName = '武汉')
```

执行结果：

RID	PCardID	TID	SStationID	AStationID	CarrigeID	SeatRow	SeatNo	SStatus
1	1506211963021:	4	1599	1625	1	1	A	1
2	4505021960022:	1	1602	1624	2	2	B	1
3	3307821932030:	4	1607	1621	3	3	C	1
4	1303221947063(	2	1607	1621	4	4	D	1

图 3.6 sql 执行结果

3）数据导入导出

通过查阅 DBMS 资料学习数据导入导出功能，并将任务 2.1 所建表格的数据导出到操作系统文件，然后再将这些文件的数据导入到相应空表。

（1）借助 navicat 的导入导出功能即可：

所有数据可以导出到 excel 表中：

（2）借助 mysqldump 导出

mysqldump -u root -p train > train_backup.sql

实验一中已经进行了导入导出的相关实验，所以这里就不再赘述了

4）观察性实验

建立一个关系，但是不设置主码，然后向该关系中插入重复元组，然后观察在图形化交互界面中对已有数据进行删除和修改时所发生的现象。

建立关系 R(A,B)；

```
CREATE TABLE R(
        A int,
        B int)
> OK
> 时间: 0.72s
```

图 3.7 建表

插入数据：

```
INSERT INTO train.R
VALUES
        ( 1, 2 ),
        ( 1, 2 ),
        ( 1, 2 )
> Affected rows: 3
> 时间: 0.54s
```

执行结果：

A	B
1	2
1	2
1	2

图 3.8 插入数据

Navicat 中删除一条记录：

图 3.9 删除数据

执行结果：

A	B
1	2
1	2

图 3.10 删除结果

Navicat 中修改一条记录

A	B
1	3
1	2

图 3.11 修改记录

执行结果：

A	B
1	3
1	2

图 3.12 执行结果

使用说明删除和修改操作只对对应的元组生效。

5）创建视图

创建一个新冠确诊病人的乘火车记录视图，其中的属性包括：身份证号、姓名、年龄、乘坐列车编号、发车日期、车厢号，席位排号，席位编号。按身份证号升序排序，如果身份证号一致，按发车日期降序排序（注意，如果病人买了票但是没坐车，不计入在内）。

```
create view diagnoserecord_info as
select diagnoserecord.PCardID,passenger.PName,passenger.Age,train.TName,
train.SDate,
taketrainrecord.CarrigeID,taketrainrecord.SeatRow,taketrainrecord.SeatNo
from diagnoserecord,passenger,taketrainrecord,train
where diagnoserecord.PCardID = passenger.PCardID
and diagnoserecord.PCardID = taketrainrecord.PCardID
and taketrainrecord.TID = train.TID;
```

结果测试：`select * from diagnoserecord_info;`

PCardID	PName	Age	TName	SDate	CarrigeID	SeatRow	SeatNo
130131200305162576	安相华	17	G4818	2020-01-22	00001	1	C
130131200305162576	安相华	17	K339	2020-01-22	00002	1	C
34040319880709303X	安中一	32	G4818	2020-01-22	00001	2	C
360202200908013015	白翱	11	G531	2020-01-22	00001	1	A
360404194207120139	敖卫	78	G4818	2020-01-22	00001	3	C
370522194808092081	安娜	72	G1007	2020-01-22	00001	1	A
440705200201301351	安炜	18	D4794	2020-01-22	00001	1	B
440705200201301351	安炜	18	D5609	2020-01-22	00002	1	B

图 3.13 执行结果

6）触发器实验

编写一个触发器，用于实现以下完整性控制规则：

1）当新增一个确诊患者时，若该患者在发病前 14 天内有乘车记录，则将其同排及前后排乘客自动加入“乘客紧密接触者表”，其中：接触日期为乘车日期。

```
CREATE TRIGGER trigger_add_diagnose AFTER INSERT ON diagnoserecord FOR EACH ROW
BEGIN
    IF
        (
            new.DStatus = 1
            AND new.PCardID IN (
            SELECT
                taketrainrecord.PCardID
            FROM
                taketrainrecord,
                train
            WHERE
                taketrainrecord.PCardID = new.PCardID
                AND taketrainrecord.TID = train.TID
                AND taketrainrecord.SStatus = 1
                AND datediff( new.FDay, train.SDate ) BETWEEN 0
                AND 14
            )
            ) THEN
            INSERT IGNORE INTO traincontactor SELECT
            train.SDate AS CDate,
            t2.PCardID AS CCardID,
            2 AS DStatus,
            new.PCardID AS PCardID
        FROM
            train,
            taketrainrecord t1,
            taketrainrecord t2
```

```
            WHERE
                new.PCardID = t1.PCardID
                AND t1.TID = train.TID
                AND t1.sstatus = 1
                AND datediff( new.FDay, train.SDate ) BETWEEN 0
                AND 14
                AND t1.TID = t2.TID
                AND t1.CarrigeID = t2.CarrigeID
                AND new.PCardID <> t2.PCardID
                AND abs( t1.SeatRow - t2.SeatRow )<= 1
                AND t2.sstatus = 1;
        END IF;
END
```

执行插入语句：

```
INSERT INTO diagnoserecord
VALUES
  ( '8', '321282197806192885', '2020-01-26', '1', '2020-01-25' );
```

图 3.14 插入 sql

执行结果：takecontactor 中增加了两行，符合预期

2020-01-22	430102201107044038	2	321282197806192885
2020-01-22	51332419700223265X	2	321282197806192885

图 3.15 执行结果

2）当一个紧密接触者被确诊为新冠时，从“乘客紧密接触者表”中修改他的状态为“1”。

```
CREATE TRIGGER trigger_update_diagnose AFTER UPDATE ON diagnoserecord
FOR EACH ROW
BEGIN
    IF
        new.DStatus = 1 THEN
            UPDATE traincontactor
            SET DStatus = 1
        WHERE
            CCardID = new.PCardID;
        END IF;
END
```

插入疑似人员到 diagnoserecord 表中

```
INSERT INTO diagnoserecord
VALUES
  ( 9, '51332419700223265X', '2020-01-27', '2', '2020-01-26' );
```

图 3.16 sql 内容

修改其 DStatus 状态为 1

```
UPDATE diagnoserecord
SET DStatus = 1
WHERE
  PCardID = '51332419700223265X';
```

图 3.17 sql 内容

观察 traincontactor 中 DStatus 状态，与上图对比

2020-01-22	51332419700223265X	1	321282197806192885

图 3.18 执行结果

触发器成功执行

3.2.3 查询

(1)查询确诊者“安娜”的在发病前 14 天内的乘车记录；

```
select * from TakeTrainRecord
    where PCardID in(
        select taketrainrecord.PCardID
        from taketrainrecord, diagnoserecord, passenger,train
        where taketrainrecord.PCardID = diagnoserecord.PCardID
        and taketrainrecord.PCardID = passenger.PCardID
        and taketrainrecord.TID = train.TID
        and Pname = '安娜'
        and DStatus = 1
        and datediff(FDay,SDate) BETWEEN 0 AND 14 );
```

运行结果：

RID	PCardID	TID	SStationID	AStationID	CarrigeID	SeatRow	SeatNo	SStatus
1	370522194808092081	192	1597	471	00001	1	A	

图 3.19 执行结果

(2)查询所有从城市“武汉”出发的乘客乘列车所到达的城市名；

```
select DISTINCT CityName from Station
    where SID in(
        select AStationID
        from Train left join Station on Train.SStationID = Station.SID
        where Station.CityName = '武汉');
```

运行结果：

CityName
咸宁
孝感
黄石
黄冈
十堰
杭州
宁波

图 3.20 执行结果

(3)计算每位新冠患者从发病到确诊的时间间隔（天数）及患者身份信息，并将结果按照发病时间天数的降序排列

```
select * from(
    select PCardID , datediff(DDay,FDay) as day_interval from DiagnoseRecord
    where DStatus = 1 order by day_interval desc) as a
    natural join Passenger;
```

运行结果：

PCardID	day_interval	PName	Sex	Age
370522194808092081	1	安娜	0	72
440705200201301351	1	安炜	1	18
130131200305162576	1	安相华	1	17
34040319880709303X	1	安中一	1	32
360404194207120139	1	敖卫	1	78
360202200908013015	1	白翱	1	11

图 3.21 执行结果

(4)查询“2020-01-22”从“武汉”发出的所有列车

```
select * from Train
    where SStationID IN (select SID from Station where CityName = '武汉')
        and SDate = '2020-01-22';
```

运行结果：

TID	SDate	TName	SStationID	AStationID	Stime	ATime
35	2020-01-22	C5631	1597	1556	2020-01-22 06:53:00	2020-01-22 07:34:00
74	2020-01-22	D3272/D327	1615	231	2020-01-22 12:37:00	2020-01-22 20:23:00
92	2020-01-22	D4794	1615	2020	2020-01-22 09:46:00	2020-01-22 13:07:00
154	2020-01-22	D5941	1615	1661	2020-01-22 07:13:00	2020-01-22 09:08:00
159	2020-01-22	D5951	1615	1540	2020-01-22 14:16:00	2020-01-22 19:36:00
162	2020-01-22	D5957	1615	1661	2020-01-22 13:41:00	2020-01-22 15:43:00
192	2020-01-22	G1007	1597	471	2020-01-22 09:30:00	2020-01-22 14:34:00
314	2020-01-22	G4818	1597	129	2020-01-22 02:54:00	2020-01-22 07:57:00
327	2020-01-22	G518	1615	129	2020-01-22 12:45:00	2020-01-22 17:46:00
348	2020-01-22	G6813	1615	1579	2020-01-22 10:02:00	2020-01-22 12:44:00
351	2020-01-22	G6819	1615	1579	2020-01-22 12:30:00	2020-01-22 15:26:00

图 3.22 执行结果

(5)查询“2020-01-22”途经“武汉”的所有列车；

```
SELECT *
FROM TRAIN
WHERE TID IN(
    SELECT TID
    FROM TRAINPASS,STATION
    WHERE STIME LIKE '2020-01-22%'
    AND TRAINPASS.SID=STATION.SID
    AND STATION.CityName='武汉'
);
```

运行结果：

TID	SDate	TName	SStationID	AStationID	Stime	ATime
55	2020-01-22	D3031/D303	1661	3365	2020-01-22 06:33:00	2020-01-22 14:47:00
70	2020-01-22	D3252/D325	2127	3997	2020-01-22 07:36:00	2020-01-22 17:25:00
74	2020-01-22	D3272/D327	1615	231	2020-01-22 12:37:00	2020-01-22 20:23:00
333	2020-01-22	G531	979	471	2020-01-22 09:14:00	2020-01-22 19:15:00

图 3.23 执行结果

(6)查询“2020-01-22”从武汉离开的所有乘客的身份证号、所到达的城市、到达日期

```
select PCardID,CityName,convert(STime,DATE) from
taketrainrecord,station,trainpass
    where SStationID in (select SID from station where CityName = '武汉')
    and taketrainrecord.TID = trainpass.TID
    and taketrainrecord.AStationID = trainpass.SID
    and taketrainrecord.AStationID = station.SID
    and convert(STime,DATE) = '2020-01-22';
```

运行结果：

(7)统计“2020-01-22” 从武汉离开的所有乘客所到达的城市及达到各个城市的武汉人员数

```
select CityName,count(CityName) from taketrainrecord,station,trainpass
    where SStationID in (select SID from station where CityName = '武汉')
    and taketrainrecord.TID = trainpass.TID
    and taketrainrecord.AStationID = trainpass.SID
    and taketrainrecord.AStationID = station.SID
    and convert(STime,DATE) = '2020-01-22'
    GROUP BY CityName
```

运行结果：

CityName	count(CityName)
深圳	3
南京	1
丰台区	4
郑州	1
石家庄	2

图 3.24 执行结果

(8)查询 2020 年 1 月到达武汉的所有人员

```
select * from
    (select PCardID
    from taketrainrecord, trainpass
    where taketrainrecord.AStationID in (select SID from station where CityName = '
武汉')
    and taketrainrecord.TID = trainpass.TID
    and taketrainrecord.AStationID = trainpass.SID
    and trainpass.STime like '2020-01%') as a
natural join passenger;
```

运行结果：

PCardID	PName	Sex	Age
360681195104064259	巴广玉	1	69
533124199004161279	巴岩	1	30
360202200908013015	白翱	1	11

图 3.25 执行结果

(9)查询 2020 年 1 月乘车途径武汉的外地人员（身份证非“420”开头）

```
select distinct
    passenger.*
from
```

```
    trainpass t,
    taketrainrecord,
    passenger
where
    t.SID in ( select SID from station where CITYNAME = '武汉' )
    and t.TID = TakeTrainRecord.TID
    and t.SNo <= ( select SNO from trainpass where taketrainrecord.ASTATIONID =
trainpass.SID and taketrainrecord.TID = trainpass.TID )
    and t.SNo >= ( select SNO from trainpass where taketrainrecord.SSTATIONID =
trainpass.SID and taketrainrecord.TID = trainpass.TID )
    and TakeTrainRecord.SSTATUS = 1
    and t.STime like '2020-01%'
    and passenger.PCardID = taketrainrecord.PCardID
    and not passenger.PCardID like '420%';
```

运行结果：

PCardID	PName	Sex	Age
62292119641111333X	白福周	1	56
450327194812204606	白光芹	0	72
350802193412192847	白桂	0	86
500111197005171816	白剑	1	50
321282197806192885	于合勤	0	42
51332419700223265X	于和	1	50
430102201107044038	于贺龙	1	9

图 3.26 执行结果

(10)统计“2020-01-22”乘坐过‘G1007’号列车的新冠患者在火车上的密切接触乘客人数（每位新冠患者的同车厢人员都算同车密切接触）

```
select diagnoserecord.PCardID,count(diagnoserecord.PCardID) as contactor_count
    from diagnoserecord, taketrainrecord t1, taketrainrecord t2
    where diagnoserecord.PCardID = t1.PCardID
    and t1.PCardID <> t2.PCardID
    and t1.TID = t2.TID
    and t1.CarrigeID = t2.CarrigeID
    and t1.TID in (select TID from train where SDate = '2020-01-22' and TName =
'G1007')
    group by diagnoserecord.PCardID;
```

运行结果：

PCardID	contactor_count
370522194808092081	2

图 3.27 执行结果

(11)查询一趟列车的一节车厢中有 3 人及以上乘客被确认患上新冠的列车名、出发日期，车厢号

```
select TName, SDate, CarrigeID
    from TakeTrainRecord, DiagnoseRecord, train
    where DStatus = 1
    and taketrainrecord.PCardID = diagnoserecord.PCardID
    and taketrainrecord.TID = train.TID
    group by TName,SDate,CarrigeID
    having count(*) >= 3;
```

运行结果：

TName	SDate	CarrigeID
G4818	2020-01-22	00001

图 3.28 执行结果

(12)查询没有感染任何周边乘客的新冠乘客的身份证号、姓名、乘车日期

```
select a1.PCardID,passenger.PName,train.SDate from
(select diagnoserecord.PCardID,COUNT(diagnoserecord.PCardID) as num
    from diagnoserecord, taketrainrecord t1, taketrainrecord t2
    where diagnoserecord.PCardID = t1.PCardID
    and t1.TID = t2.TID
    and t1.PCardID <> t2.PCardID
    and t1.CarrigeID = t2.CarrigeID
group by diagnoserecord.PCardID ) as a1,
(select diagnoserecord.PCardID,COUNT(diagnoserecord.PCardID) as num
    from diagnoserecord, taketrainrecord t1, taketrainrecord t2
    where diagnoserecord.PCardID = t1.PCardID
    and t1.TID = t2.TID
    and t1.PCardID <> t2.PCardID
    and t1.CarrigeID = t2.CarrigeID
    and t2.PCardID not in (select PCardID from diagnoserecord)
group by diagnoserecord.PCardID ) as a2 , taketrainrecord , passenger , train
where a1.PCardID = a2.PCardID
and a1.num = a2.num
and a1.PCardID = taketrainrecord.PCardID
and a1.PCardID = passenger.PCardID
and train.TID = taketrainrecord.TID
```

运行结果：

PCardID	PName	SDate
360202200908013015	白翎	2020-01-22
370522194808092081	安娜	2020-01-22

图 3.29 执行结果

(13)查询到达 “北京”、或“上海”，或“广州”（即终点站）的列车名，要求 where 子句中除了连接条件只能有一个条件表达式

```
select TName from
Train    join Station on Train.AStationID = Station.SID
where    CityName in ('北京','上海','广州');
```

运行结果：

TName
G4555
G4559
G4651
G4653
G4665
G4671
G4683
G4687
G4689
G4691
G4693
G4697

图 3.30 执行结果

(14)查询“2020-01-22”从“武汉站”出发，然后当天换乘另一趟车的乘客身份证号和首乘车次号，结果按照首乘车次号降序排列，同车次则按照乘客身份证号升序排列

```
select PCardID,TName from
    (select *
    from taketrainrecord
    where SStationID in (select SID from station where CityName = '武汉'))as t1
    join taketrainrecord t2 using(PCardID) join train on t1.TID = train.TID
    where t1.TID <> t2.TID
    and t1.TID in (select DISTINCT TID from trainpass where convert(atime,DATE)
= '2020-01-22')
    order by TName desc, PCardID;
```

运行结果：

PCardID	TName
130131200305162576	G4818
440705200201301351	D4794

图 3.31 执行结果

(15)查询所有新冠患者的身份证号，姓名及其 2020 年以来所乘坐过的列车名、发车日期，要求即使该患者未乘坐过任何列车也要列出来

```
select diagnoserecord.PCardID,PName,TName,SDate
    from diagnoserecord natural left outer join
        (select PCardID,TName,SDate
        from taketrainrecord, train
        where taketrainrecord.TID = train.TID
    ) as a natural join passenger
    where SDate like '2020%' or SDate is null;;
```

运行结果：

PCardID	PName	TName	SDate
110105200604134974	侯保兵	(Null)	(Null)
130131200305162576	安相华	G4818	2020-01-22
34040319880709303X	安中一	G4818	2020-01-22
360202200908013015	白翱	G531	2020-01-22
360404194207120139	敖卫	G4818	2020-01-22
370522194808092081	安娜	G1007	2020-01-22
440705200201301351	安炜	D4794	2020-01-22
440705200201301351	安炜	D5609	2020-01-22

图 3.32 执行结果

(16) 查询所有发病日期相同而且确诊日期相同的病患统计信息，包括：发病日期、确诊日期和患者人数，结果按照发病日期降序排列的前提下再按照确诊日期降序排列。

```
select FDay, DDay, count(*)as ill_num from DiagnoseRecord
    where DStatus = 1
    group by DDay,FDay
    order by FDay desc, DDay desc;
```

运行结果：

FDay	DDay	ill_num
2020-01-24	2020-01-25	6

图 3.33 执行结果

3.2.4 了解系统的查询性能分析功能

对于如下查询语句进行分析：

```
-- 9）查询 2020 年 1 月乘车途径武汉的外地人员（身份证非“420”开头）；
Explain select distinct
    passenger.*
from
    trainpass t,
    taketrainrecord,
    passenger
where
    t.SID in ( select SID from station where CITYNAME = '武汉' )
    and t.TID = TakeTrainRecord.TID
    and t.SNo <= ( select SNO from trainpass where taketrainrecord.ASTATIONID = trainpass.SID and taketrainrecord.TID = trainpass.TID )
    and t.SNo >= ( select SNO from trainpass where taketrainrecord.SSTATIONID = trainpass.SID and taketrainrecord.TID = trainpass.TID )
    and TakeTrainRecord.SSTATUS = 1
    and t.STime like '2020-01%'
    and passenger.PCardID = taketrainrecord.PCardID
    and not passenger.PCardID like '420%';
```

在语句前加上 explain 查看执行计划：

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	taketrainrecord	(Null)	ALL	PCardID,TID	(Null)	(Null)	(Null)	21	10.00	Using where; Using temp
1	PRIMARY	t	(Null)	ref	PRIMARY,SID	PRIMARY	4	train.taketrainrecord.TID	9	11.11	Using where
1	PRIMARY	station	(Null)	eq_ref	PRIMARY	PRIMARY	4	train.t.SID	1	10.00	Using where
1	PRIMARY	passenger	(Null)	eq_ref	PRIMARY	PRIMARY	72	train.taketrainrecord.PCar	1	100.00	Using where
4	DEPENDENT SUBQUERY	trainpass	(Null)	ref	PRIMARY,SID	SID	9	train.taketrainrecord.SSta	1	100.00	Using index
3	DEPENDENT SUBQUERY	trainpass	(Null)	ref	PRIMARY,SID	SID	9	train.taketrainrecord.ASta	1	100.00	Using index

图 3.34 执行计划

expain 出来的信息有 10 列，分别是 id、select_type、table、type、possible_keys、key、key_len、ref、rows、Extra

对这些列的概要描述：

id: 如果相同，可以认为是一组，从上往下顺序执行；在所有组中，id 值越大，优先级越高，越先执行

select_type:表示查询的类型，最外层查询则被标记为主查询 PRIMARY，SUBQUERT 为子查询

table:输出结果集的表

type:表示表的连接类型，all 表示全表扫描；ref 是唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配；eq_ref 是非唯一性索引扫描，返回匹配某个

单独值的所有行

possible_keys: 显示可能应用在这张表中的索引，一个或多个

key: 实际使用到的索引，如果为 NULL，则没有使用索引

key_len: 显示索引的哪一列被使用了，如果可能的话，是一个常数，哪些列或常量被用于查找索引列上的值

ref: 显示索引的哪一列被使用了，如果可能的话，是一个常数，哪些列或常量被用于查找索引列上的值

rows: 根据表统计信息及索引选用情况，大致估算出找到所需的记录所需读取的行数

filtered: 查询的表行占表的百分比

Extra:执行情况的描述和说明，using index 表示相应的 select 操作中使用了覆盖索引，避免访问了表的数据行；

3.2.5 DBMS 函数及存储过程和事务

1）编写一个依据乘客身份证号计算其在指定年乘列车的乘车次数的自定义函数，并利用其查询 2020 年至少乘车过 3 次的乘客。

```
CREATE DEFINER=`root`@`localhost` FUNCTION `getcount`(`ID` char(18),`y` year) RETURNS int
    READS SQL DATA
BEGIN
    #Routine body goes here...
    DECLARE count int;
    select count(*) from taketrainrecord,train
    where taketrainrecord.tid = train.tid
    and YEAR(train.SDate) = y
    and taketrainrecord.PCardID = ID
    group by taketrainrecord.PCardID
    into count;
    RETURN count;
END
```

执行函数测试：

```
insert into taketrainrecord values(30,'370522194808092081',192,1597,471,1,1,'A',1);
insert into taketrainrecord values(31,'370522194808092081',192,1597,471,1,1,'A',1);
insert into taketrainrecord values(32,'370522194808092081',192,1597,471,1,1,'A',1);
insert into taketrainrecord values(33,'370522194808092081',192,1597,471,1,1,'A',1);

select distinct passenger.* from passenger,TakeTrainRecord
where getcount(TakeTrainRecord.PCardID,2020)>=3
and passenger.PCardID = TakeTrainRecord.PCardID;
```

图 3.35 执行结果

测试结果

PCardID	PName	Sex	Age
370522194808092081	安娜	0	72

图 3.36 测试结果

2）尝试编写 DBMS 的存储过程，建立每趟列车的乘坐人数的统计表，并通过存储过程更新该表。

```
CREATE TABLE countpertrain (
TID INT PRIMARY KEY,
count INT,
FOREIGN KEY ( TID ) REFERENCES train ( TID ) );

CREATE PROCEDURE count_train () BEGIN
    INSERT INTO countpertrain SELECT
    TID,
    count(*)
    FROM
        TakeTrainRecord
    GROUP BY
        TakeTrainRecord.TID;
END;
```

执行该存储过程：

Call count_train();

执行结果：

TID	count
92	1
129	1
192	3
284	3
314	7
333	5
397	1

图 3.37 存储过程执行结果

3）尝试在 DBMS 的交互式界面中验证事务机制的执行效果。

mysql 事务具有 ACID 特性：即原子性，一致性，隔离性和持久性

mysql 事务的隔离级别包括：

read uncommitted：读取尚未提交的数据 ：哪个问题都不能解决

read committed：读取已经提交的数据 ：可以解决脏读

repeatable read：重读读取：可以解决脏读 和 不可重复读 ---mysql 默认的

serializable：串行化：可以解决 脏读 不可重复读 和 虚读---相当于锁表

显示当前的隔离级别：

```
mysql> select @@transaction_isolation;
+-------------------------+
| @@transaction_isolation |
+-------------------------+
| REPEATABLE-READ         |
+-------------------------+
1 row in set (0.00 sec)
```

图 3.38 当前隔离级别

可以看到隔离级别为 repeatable read

将隔离级别设置为 read uncommited

此时对于两个不同的事务 A，B：在 A 中进行 update 操作，在 B 中进行 select 操作，当 AB 并行执行时，若 A 由于某些异常情况造成了事务回滚，那么 B 读到的 A 执行 update 以后的数据就属于脏数据。

3.3 任务总结

第二次实验大量练习了 SQL 查询语句的编写，部分查询问题的语义不是很明确，在查询的过程中经过了多次修改，最初检查时未考虑到途径武汉的区间问题，在指出后及时修正了。

比较有难度的部分是几个存储过程的编写，这部分的内容在课堂上并未设计，查阅了一些相关的资料才了解了基本的语法，而且在真正编写的时候还碰到一些不好定位的问题，花费了较多的时间来解决。

选做题在后来进行了尝试，通过查阅资料了解了 mysql 语句查询过程分析中各个项的含义，了解了数据库执行复杂 sql 语句的过程。

总的来说，这次实验大大提高了我的 sql 熟练度，也很好地熟悉了 mysql 的使用，为实验三开发系统软件打下了较好的基础。

4 综合实践任务

4.1 系统设计目标

4.1.1 应用背景

如今，信息资源已经成为一个企业的重要财富，随着企业的规模扩大以及对于信息处理日益增长的需求，拥有一个能够高效管理信息的系统软件已经成为提高企业运作效率的必要条件。而作为计算机领域的基础软件之一，数据库已经越来越体现出它的核心价值，被广泛应用于各种信息处理系统中，数据库的完整设计成了一个系统不可或缺的模块。

本次课程设计选题为常见的企业管理系统，以工资管理为着重点，期望实现对于员工工资状况的高效管理，从而对于企业的管理形成良好反馈，采用 B/S 架构实现。

4.1.2 基本设计目标

系统需求：实现一个工资管理系统，完成工种管理，员工工资管理，员工考勤管理，员工基本信息管理，生成企业工资报表

对管理人员：

1. 添加新员工：即添加新入职员工的基本信息，允许进行修改
2. 删除员工：删除离职员工的相关信息
3. 管理工种情况表：允许对工种，等级，基本工资进行修改
4. 管理员工津贴信息表：查看或修改员工加班类别、加班天数、津贴情况
6. 查看企业工资报表：查看整体的工资情况，以图表的形式进行展示

对员工：

1. 打卡：记录当日工作情况，签到和签退时间
2. 查看个人工资情况
3. 查看个人考勤状况
4. 查看个人津贴情况

4.1.3 追加目标

1.使用 session 区分登录状态

2.数据展示图表添加排序，筛选等功能

4.1.4 安全性目标

1. 实现管理人员和用户人员的权限分离
2. 对数据库的数据定期进行备份，保障数据的可靠性
3. 网页需要进行 SQL 注入攻击的防护

4.2 需求分析

4.2.1 系统功能划分

1.主页：输入网站域名后，直接显示登录页面，跟据输入账号的权限等级跳转到对应的用户界面或管理员界面。

2.用户考勤：进入用户界面后可以考勤，包括签到和签退，记录当前时间到数据库。

3.用户查询：进入用户界面后，用户可以选择查询自己的考勤信息，津贴信息，工资信息。

4.实现管理员对用户的管理：包括管理和个人信息和登录账户信息

5.实现管理员对工种表的管理，允许修改不同工种的基本工资

6.实现管理员查询所有人的考勤记录/津贴情况/工资情况

7.实现管理员对津贴情况添加和更新

8.实现管理员查询并生成企业工资报表，以图表的形式进行多方面的展示

4.2.2 性能需求

系统使用初期数据量较少，对性能无明显要求；当员工数据较多的时候，可能在数据查询上存在一定的时间消耗，这种情况下可以考虑对于基本信息表按照部门进行分表操作以减少查询的时间，还可以考虑建立在需要经常查询的表属性上建立相应的索引。

4.2.3 数据表说明

数据库中应该具有以下的几张表：

员工考勤情况表，反映员工的考勤情况

员工工种情况表，反映员工的工种、等级，基本工资等信息

员工津贴信息表，反映员工的加班时间，加班类别、加班天数、津贴情况等

员工基本信息表，反映员工的基本信息

员工月工资表，反映员工的月工资情况

登陆信息表，用来处理所有的登录账号

4.2.4 数据流图

顶层：

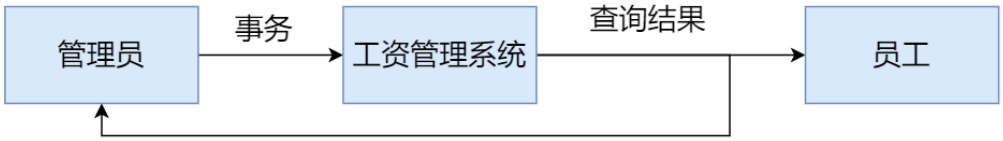

图 4.1 顶层数据流

第二层：

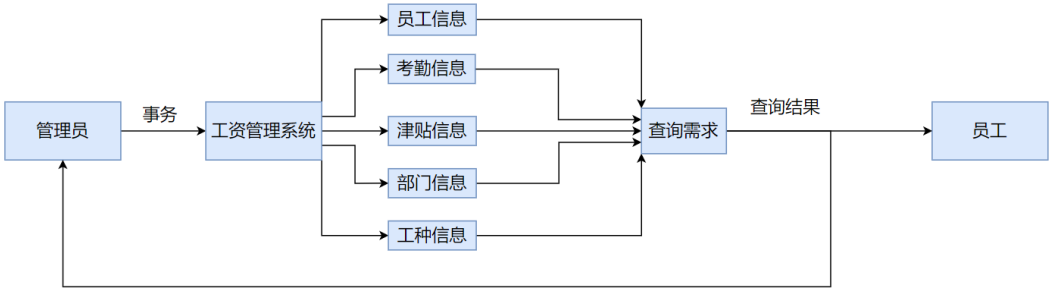

图 4.2 第二层数据流

4.2.5 数据字典

字段名	数据类型	允许为空	字段说明
UID	char(10)	NO	员工 ID
ADate	date	NO	签到日期
ATime	time	YES	签到时间
STime	time	YES	签退时间
BDate	date	NO	津贴日期
BType	char(20)	NO	津贴类型
BDays	smallint(6)	YES	津贴时长
Bonus	int(11)	YES	津贴数目
DID	smallint(6)	NO	部门 ID
DName	char(30)	YES	部门名称
DRef	varchar(255)	YES	部门备注说明
Dnum	smallint(6)	YES	部门人数

UName	char(30)	YES	员工姓名
Age	smallint(6)	YES	年龄
Sex	tinyint(1)	YES	性别，0 为男，1 为女
KID	smallint(6)	YES	工种编号
KName	char(30)	YES	工种名称
Level	smallint(6)	YES	职阶
Base_salary	int(11)	YES	基本薪资
Password	char(30)	YES	登录密码
Authority	smallint(6)	YES	权限等级， 0 为管理员， 1 为一般员工
Month	date	NO	月份
Base_salary	int(11)	YES	基本薪资
attendance_times	smallint(6)	YES	本月出勤次数
attendance_rate	float	YES	本月出勤率
Bonus	int(11)	YES	本月津贴
Total_salary	int(11)	YES	本月总工资
UID	char(10)	NO	员工 ID
UName	char(30)	YES	员工姓名

4.3 总体设计

4.3.1 开发环境

开发环境：python3.6.4
运行环境：Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-178-generic x86_64)

4.3.2 系统架构

依据系统的功能和开发语言的选择，设计系统的总体架构如下图：

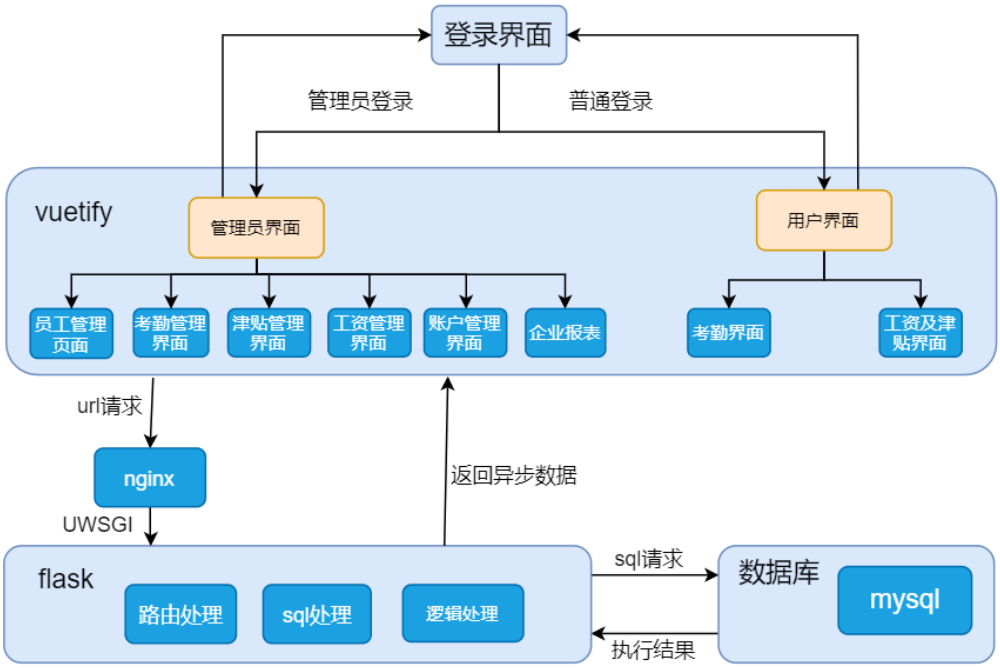

图 4.3 总体架构图

系统前端由 vuetify 实现，后台由 nginx 做路由分发，flask + mysql 做逻辑和数据库处理。各部分的跳转指向如上图，同时对每个页面，区分好用户和管理员的权限。用户在前端操纵网页上触发 url 跳转时，由 flask 的路由层捕获并执行相应处理逻辑后返回数据渲染模板。Linux 云服务器上部署 nginx 服务器接受 url 请求，以 uwsgi 来进行 nginx 和 flask 的 wsgi 接口的通信，通过浏览器根据 IP 或者域名访问该系统。

4.3.3 功能模块

①登录

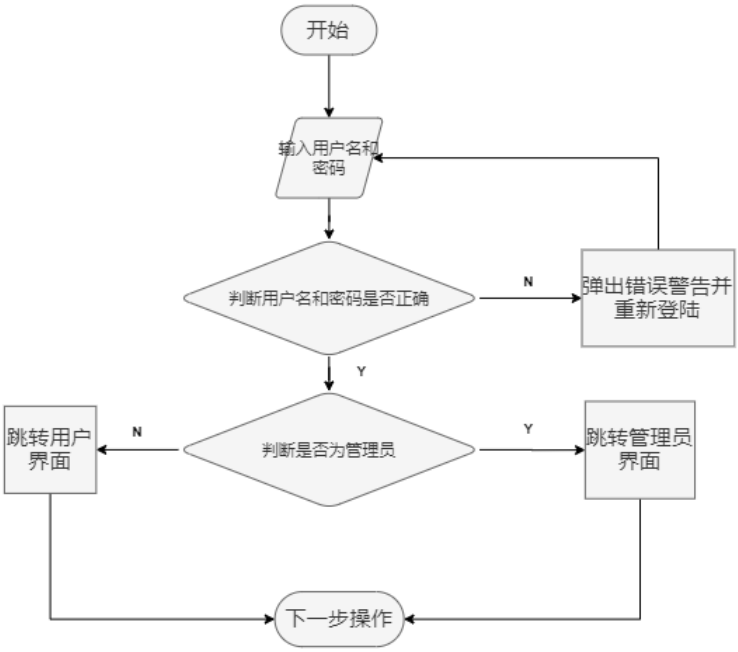

图 4.4 登录模块

②用户考勤

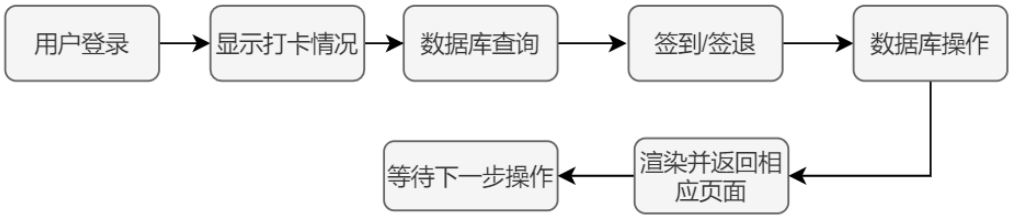

图 4.5 考勤模块

③用户查询个人信息，工资及津贴

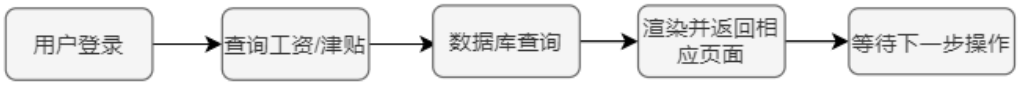

图 4.6 查询模块

④管理员添加新员工

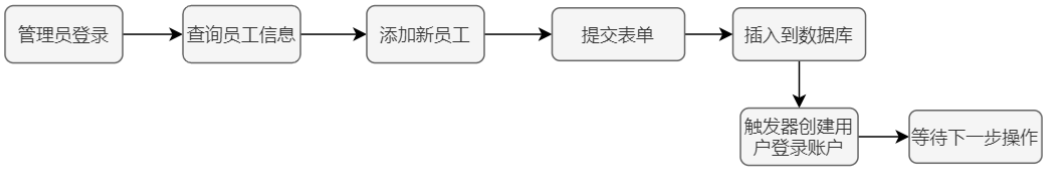

图 4.7 添加模块

⑤管理员修改员工信息/部门信息/工种信息/津贴信息/账号信息

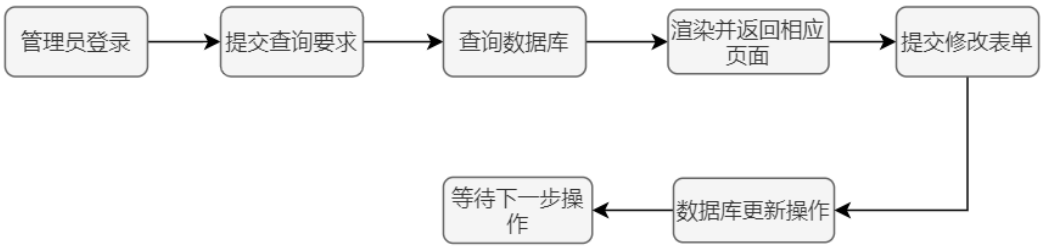

图 4.8 修改模块

⑥管理员查询工资信息和企业报表

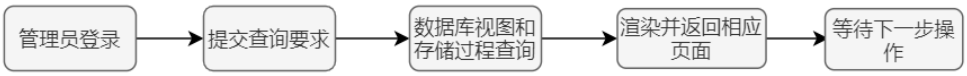

图 4.9 查询报表模块

4.4 数据库设计

4.4.1 E-R 图设计

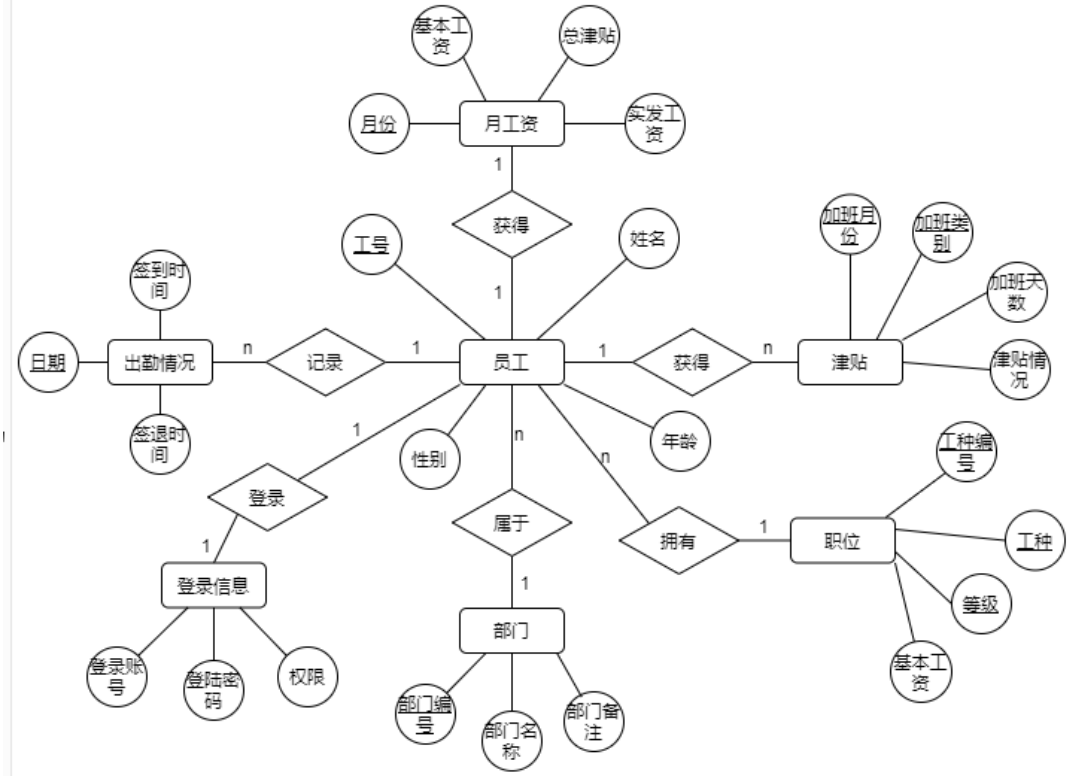

图 4.10 数据库 ER 图设计

4.4.2 关系数据模型

1.员工基本信息表（employee_info）

字段	中文说明	字段类型	完整性约束	备注
UID	工号	char(10)	主码	
UName	姓名	char(30)		
Age	年龄	Samllint		
Sex	性别	{0,1}		{男，女}
KID	工种编号	Smallint	外码	
DID	部门编号	Smallint	外码	

2.员工工种情况表（kind_info）

字段	中文说明	字段类型	完 整 性 约 束	备注

KID	工种编号	Smallint	主码	
KName	工种	char(30)		
Level	等级	Samllint		
Base_salary	基本薪资	int		

3.部门情况表（department_info）

字段	中文说明	字段类型	完整性约束	备注
DID	部门编号	Smallint	主码	
DName	部门名称	Char(30)		
DRef	部门备注	Char(255)		
DNum	部门人数	smallint		

4.员工考勤情况表（attendance_info）

字段	中文说明	字段类型	完整性约束	备注
UID	工号	char(10)	主码，外码	
Date	日期	Date	主码	
ATime	签到时间	time		
STime	签退时间	time		

5.员工津贴信息表（bonus_info）

字段	中文说明	字段类型	完整性约束	备注
UID	工号	char(10)	主码，外码	
BDate	加班月份	Date	主码	
BType	加班类型	char(20)	主码	记录加班原因
BDys	加班天数	Smallint		
Bonus	津贴情况	int		

6.员工月工资表（salary_info）

字段	中文说明	字段类型	完整性约束	备注

UID	工号	char(10)	主码,外码	
Month	月份	Date	主码	
Base_salary	基本工资	int		
attendance_times	出勤次数	smallint		
attendance_rate	出勤率	float		
Bonus	津贴	Int		
Total_salary	总工资	int		

7.登录信息表（log_info）

字段	中文说明	字段类型	完整性约束	备注
UID	工号(登陆账号)	char(10)	主码	
Password	登录密码	Char(100)		
Authority	权限等级	{0,1}		0 为管理员权限

4.4.3 数据库视图

(1) view_get_employee_info：查询用户信息的视图

功能：为后端提供一个查询接口，查询基本的个人信息

具体实现：

```
SELECT
    `employee_info`.`UID` AS `UID`,
    `employee_info`.`UName` AS `UName`,
    `employee_info`.`Age` AS `Age`,
    `employee_info`.`Sex` AS `Sex`,
    `department_info`.`DName` AS `DName`,
    `kind_info`.`KName` AS `KName`,
    `kind_info`.`Level` AS `Level`,
    `kind_info`.`Base_salary` AS `Base_salary`,
    `kind_info`.`KID` AS `KID`,
    `department_info`.`DID` AS `DID`
FROM
    (( `employee_info` JOIN `department_info` ) JOIN `kind_info` )
WHERE
    ((
            `employee_info`.`KID` = `kind_info`.`KID`
        )
    AND ( `employee_info`.`DID` = `department_info`.`DID` ))
```

(2) view_get_salary_report：查询企业工资报表的视图
功能：为后端提供每个员工的年度工资展示接口
具体实现：

```
SELECT
    `employee_info`.`UID` AS `UID`,
    `employee_info`.`UName` AS `UName`,
    `department_info`.`DName` AS `DName`,
    `get_salary_year` ( `employee_info`.`UID`, '2020' ) AS `Salary`,
    `get_year_end_awards_employee` ( `employee_info`.`UID`, '2020' ) AS
`Award`
FROM
    ( `employee_info` JOIN `department_info` )
WHERE
    (
    `employee_info`.`DID` = `department_info`.`DID`)
```

4.4.4 数据库物理设计

使用到索引的地方

Employee_info 中：

INDEX `KID`(`KID`) USING BTREE,

INDEX `DID`(`DID`) USING BTREE,

4.5 详细设计与实现

4.5.1 数据库事务的定义与实现

在本次实验中用到了两个个数据库存储事务。目的是保证一系列操作的原子性，防止在发生意外情况时导致的数据库数据错误。

更新考勤信息：attendance_update (UID CHAR (10), ADate date, ATime time, STime time, type INT)

输入参数：

UID：员工 ID

Adate：考勤日期

ATime：签到时间

STime：签退时间

Type：更新种类，签退还是签到

作用：根据 type 的种类来选择更新 attendance_info 表的方式

具体实现：根据 type 的种类和当前 attendance 表中是否存在签到信息来选择更新的方式，当存在签到信息时不允许再次签到，当存在签到信息时才允许签退

```
CREATE PROCEDURE attendance_update ( UID CHAR ( 10 ), ADate date, ATime time, STime time, type INT ) BEGIN
    DECLARE
        flag INT;
    SELECT
        count(*) INTO flag
    FROM
        attendance_info
    WHERE
        attendance_info.UID = UID
        AND attendance_info.ADate = ADate;
    IF
        type = 0
        AND flag = 0 THEN
            INSERT INTO attendance_info
        VALUES
            ( UID, ADate, ATime, NULL );

        ELSEIF type = 1
        AND flag = 1 THEN
            UPDATE attendance_info
            SET attendance_info.STime = STime
            WHERE
                attendance_info.UID = UID
                AND attendance_info.ADate = ADate;
        END IF;
END;
```

在使用时执行 call attendance_update()即可，用于向后端提供一个处理考勤信息的接口。

插入数据到 salary_info 表：calculate_salary_per_month (UID CHAR (10), whichmonth date)

输入参数：

UID：员工 ID

Date：月份信息

具体实现：首先会获得员工对应工种的基本工资和本月总津贴，然后再 attendance_info 表中获得其再本月的考勤数，根据考勤数计算出其考勤率，将基

本工资乘上考勤率加上津贴作为本月的总工资，最后将上述信息均填入 salary 表中。

```
CREATE PROCEDURE calculate_salary_per_month ( UID CHAR ( 10 ),
whichmonth date ) BEGIN
        DECLARE
            Base_salary INT;
        DECLARE
            Bonus INT;
        DECLARE
            Total_salary INT;
        DECLARE
            attendance_times INT;
        DECLARE
            attendance_rate FLOAT;

        SET Base_salary = get_base_salary ( UID );

        SET Bonus = get_bonus_month ( UID, whichmonth );
        SELECT
            COUNT(*) INTO attendance_times
        FROM
            attendance_info
        WHERE
            attendance_info.UID = UID AND
DATE_FORMAT( attendance_info.ADate, '%Y-%m' ) =
DATE_FORMAT( whichmonth, '%Y-%m' )
            AND TIMESTAMPDIFF( HOUR, attendance_info.ATime,
attendance_info.STime ) >= 8;

        SET attendance_rate = attendance_times / 22;

        SET Total_salary = Bonus + Base_salary * attendance_rate;
        DELETE
        FROM
            salary_info
        WHERE
            salary_info.UID = UID
            AND DATE_FORMAT( salary_info.`Month`, '%Y-%m' ) =
DATE_FORMAT( whichmonth, '%Y-%m' );
        INSERT INTO salary_info
        VALUES
        ( UID, CONCAT( DATE_FORMAT( whichmonth, '%Y-%m' ),
'-01' ),Base_salary, attendance_times, attendance_rate,    Bonus, Total_salary );
    END
```

4.5.2 数据库函数定义与实现

(1) 查询基本工资的函数：get_base_salary()

具体实现：根据员工 ID 查询其对应工种的基本工资

```
CREATE FUNCTION get_base_salary (
    UID CHAR ( 10 )) RETURNS INT BEGIN
    DECLARE
        ret INT;
    SELECT
        Base_salary
    FROM
        employee_info,
        kind_info
    WHERE
        employee_info.UID = UID
        AND employee_info.KID = kind_info.KID INTO ret;
RETURN ret;
END;
```

(2) 查询某个月份所有津贴的函数：get_bonus_month()

具体实现：根据员工 ID 和 date 查询其某个月份的总津贴

```
CREATE FUNCTION get_bonus_month ( UID CHAR ( 10 ), whichmonth date )
RETURNS INT BEGIN
    DECLARE
        total_bonus INT;
    SELECT
        SUM( Bonus )
    FROM
        bonus_info
    WHERE
        bonus_info.UID = UID
        AND DATE_FORMAT( whichmonth, '%Y-%m' ) =
DATE_FORMAT( bonus_info.BDate, '%Y-%m' ) INTO total_bonus;
    RETURN total_bonus;

END;
```

(3) 计算某个 UID 某年的总工资的函数：get_salary_year ()

具体实现：根据员工 ID 和年份记录其获得的总工资

```
CREATE FUNCTION get_salary_year (
```

```
    UID CHAR ( 10 ),
    whichyear CHAR ( 10 )) RETURNS INT BEGIN
    DECLARE
        total INT;
    SELECT
        SUM( salary_info.Total_salary )
    FROM
        salary_info
    WHERE
        salary_info.UID = UID
        AND DATE_FORMAT( salary_info.MONTH, '%Y' ) = whichyear
INTO total;
    RETURN total;
END;
```

(4) 计算某个 UID 某年的年终奖的函数：get_year_end_awards_employee()
具体实现：根据员工 UID 和年份计算其获得的年终奖

```
CREATE FUNCTION get_year_end_awards_employee (
    UID CHAR ( 10 ),
    whichyear CHAR ( 10 )) RETURNS INT BEGIN
        DECLARE
            ret INT;
        SET ret = get_salary_year(UID,whichyear) / 12;
        RETURN ret;

    END;
```

4.5.3 触发器定义与实现

(1) 当对津贴进行改动时重新计算并修改 salary_info 的触发器：
津贴表插入表项时：调用存储过程重新计算插入表项对应的 salary_info
具体实现：

```
CREATE TRIGGER calculate_salary_per_month_when_insert_bonus AFTER
INSERT ON bonus_info FOR EACH ROW
BEGIN
        CALL calculate_salary_per_month ( NEW.UID, NEW.BDate );
END;
```

津贴表删除表项时：调用存储过程重新计算删除表项对应的 salary_info
具体实现：

```
CREATE TRIGGER calculate_salary_per_month_when_delete_bonus AFTER
DELETE ON bonus_info FOR EACH ROW
```

```
BEGIN
        CALL calculate_salary_per_month ( OLD.UID, OLD.BDate );

END;
```

津贴表改动表项时：当新旧表项津贴日期发生变化时，要同时调用存储过程重新计算两个津贴日期对应的 salary_info

具体实现：

```
CREATE TRIGGER calculate_salary_per_month_when_update_bonus AFTER
UPDATE ON bonus_info FOR EACH ROW
BEGIN
        CALL calculate_salary_per_month ( OLD.UID, OLD.BDate );
    IF
        OLD.BDate != NEW.BDate THEN
            CALL calculate_salary_per_month ( NEW.UID, OLD.BDate );

    END IF;
END;
```

(2) 签退时更新当月的薪资情况

考勤表改动表项时：当考勤表插入签退信息时，调用存储过程重新计算考勤日期对应月份的 salary_info

具体实现：

```
CREATE TRIGGER get_salary_permonth_when_update_attendance AFTER
UPDATE ON attendance_info FOR EACH ROW
BEGIN
    IF
        ! isnull( NEW.STime ) THEN
            CALL calculate_salary_per_month ( NEW.UID, NEW.ADate );
    END IF;
END;
```

(3) 员工基本信息表更新时：

添加新员工信息时：为新员工添加登录信息，包括账号和初始密码

具体实现：

```
CREATE TRIGGER add_new_log AFTER INSERT ON employee_info FOR
EACH ROW
BEGIN
        INSERT INTO log_info
    VALUES
    ( new.UID, new.UID, 1 );
END;
```

删除员工时：同时删除其登录账号信息
具体实现：

```
CREATE TRIGGER del_old_log BEFORE DELETE ON employee_info FOR
EACH ROW
BEGIN
    DELETE FROM log_info where log_info.UID=OLD.UID;
END;
```

添加新员工信息时：对应的部门人数自增
具体实现：

```
CREATE TRIGGER department_people_num_plus AFTER INSERT ON
employee_info FOR EACH ROW
BEGIN
        UPDATE department_info
        SET Dnum = IFNULL( Dnum, 0 )+ 1
    WHERE
    DID = new.DID;
END;
```

删除新员工信息时：对应的部门人数自减
具体实现：

```
CREATE TRIGGER department_people_num_minus AFTER DELETE ON
employee_info FOR EACH ROW
BEGIN
        UPDATE department_info
        SET Dnum = Dnum - 1
    WHERE
    DID = old.DID;
END;
```

更新员工部门时：同时对离开的部门和加入的部门人数进行修正
具体实现：

```
CREATE TRIGGER department_people_num_update AFTER UPDATE ON
employee_info FOR EACH ROW
BEGIN
    IF
        ( old.DID != new.DID )
        THEN
            UPDATE department_info
```

```
        SET Dnum = Dnum + 1
    WHERE
        DID = new.DID;
    UPDATE department_info
    SET Dnum = Dnum - 1
    WHERE
        DID = old.DID;
    END IF;
END;
```

4.5.4 主干功能的业务流程图

整个系统的业务流程图如下所示：

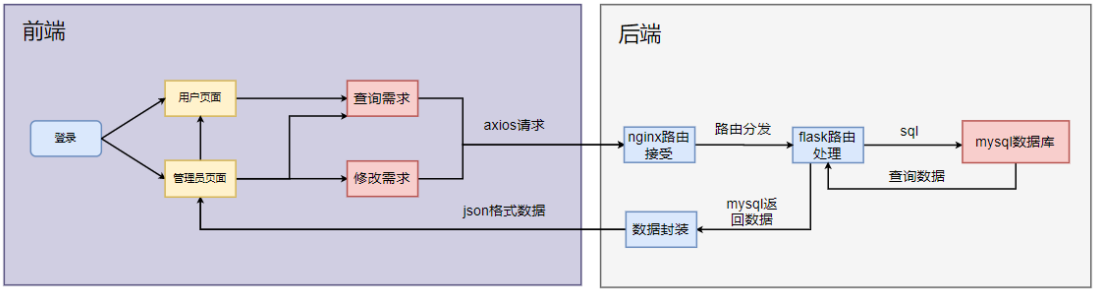

图 4. 11 业务流程图

4.5.5 关键技术说明

下面列举一些系统核心的重要的操作及其算法进行说明。

① 登录的实现

算法设计：

1.首先用户输入所有数据并提交。

2.判断用户提交数据是否有空，若有空则退回 1。

3.执行 SQL 查询，判断用户是否存在且密码是否输入正确，若不正确则退回 1。

4.将用户的登录信息存储到 session 中，并保存其权限等级根据权限等级跳转到对应的主页

```
session['uid'] = uid
session['password'] = password
session['authority'] = authority
session.permanent = True
if(authority == 0):
        return render_template('index.html')
else:
```

```
return render_template('index-user.html')
```

② 用户个人信息展示

算法设计：

1.前端提交 axios 请求

2.后端处理对应路由，根据 session 中存储的 authoriity 权限等级，执行相应的 sql(数据库中已创建好对应的视图)：

```
sql_admin = '''SELECT * from view_get_employee_info'''
sql_user = '''SELECT * from view_get_employee_info where UID = %s'''
```

3.数据库返回查询数据给后端以后，由后端将其封装为 json 格式后返回给 ajax 请求，由前端逻辑进行数据填充

③ 修改和更新请求

算法设计：

1.前端填写表单后发送 axios 请求，附带表单数据

2.后端处理对应路由，先根据不同的路由提取 axios 中的数据，创建相应的 sql 语句，交由数据库处理

例如：对于工种的更新请求，处理方式如下：

```
@app.route("/index/delete/kind_info", methods=["POST"])
 def kind_info_delete():
        KID = request.get_json()['KID']
        sql = '''delete from kind_info where KID = %s'''
        data = db.query(sql,KID)
        return jsonify(data)
```

3.前端点击刷新按钮后可调用②相应的步骤对对应的展示页面予以刷新

部分 sql 语句调用了数据库中的函数和存储过程以简化语句

④ 企业工资报表分析的实现

算法设计：

1.对于企业报表中的每一个部门工资进行累加求和，检测到第一次出现时需要进行初始化

2.将所有部门和对应的总薪资以键值对的形式填充到 echart 中去

3.调用 echart 进行数据展示

4.如果重复点击分析薪资按钮后会回到步骤 1 进行刷新操作

4.5.6 界面设计

前端各个页面的如下述截图所示：

登录页：

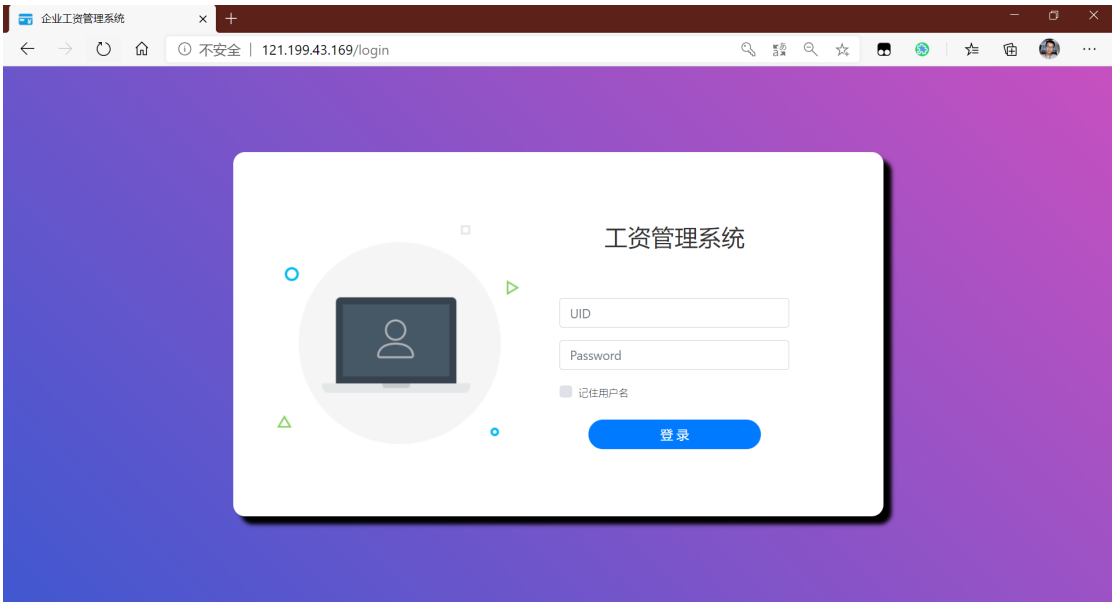

图 4.12 登录页

普通用户登录：

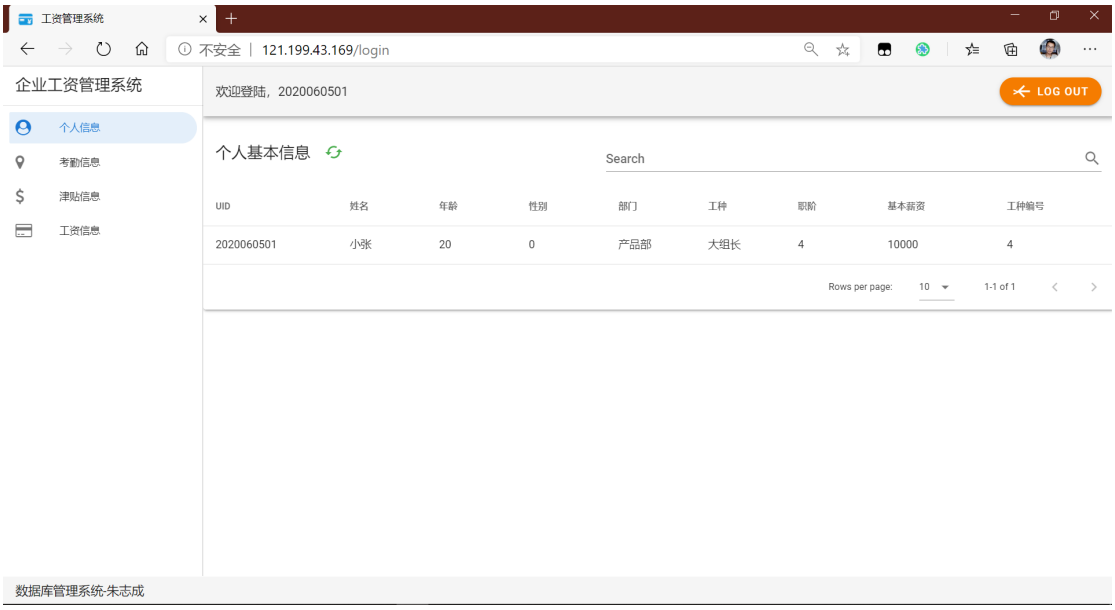

图 4.13 普通用户主页面

考勤页面：

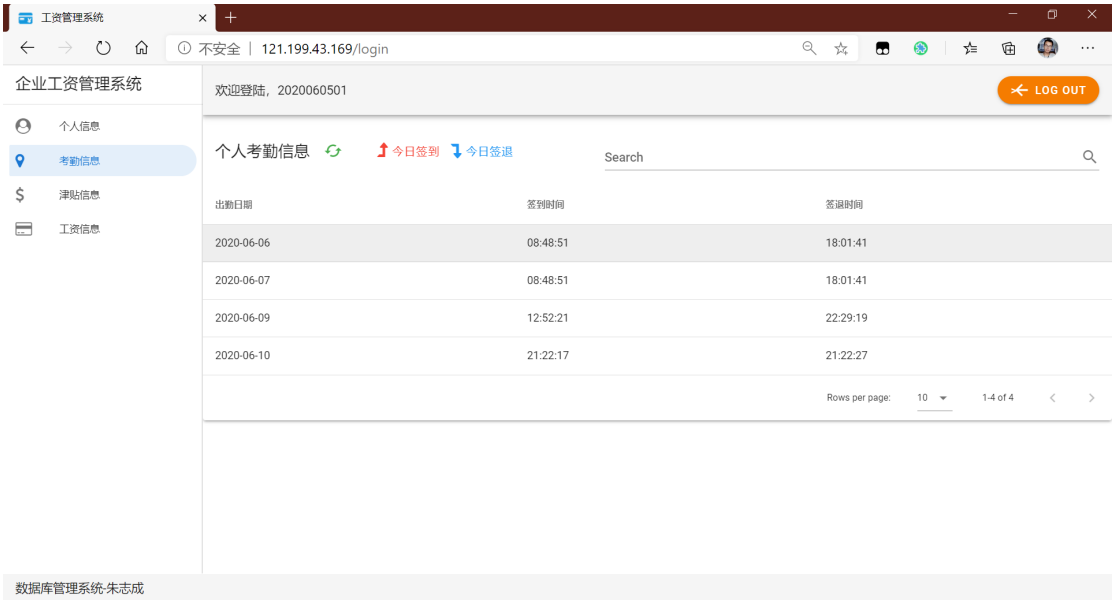

图 4.14 用户考勤页面

津贴信息：

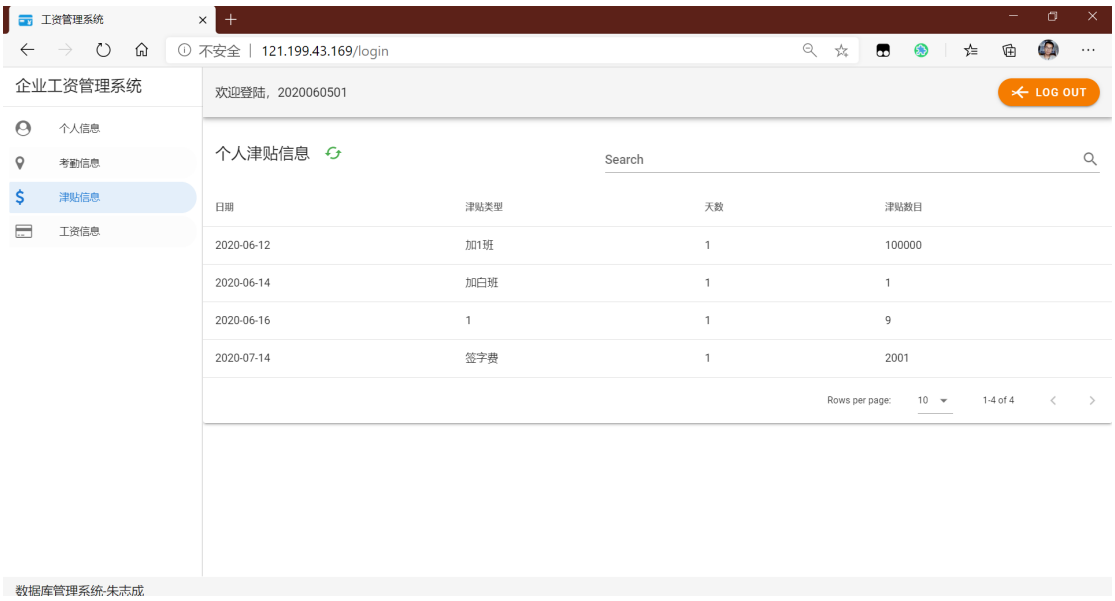

图 4.15 普通用户津贴信息页面

管理员登录：

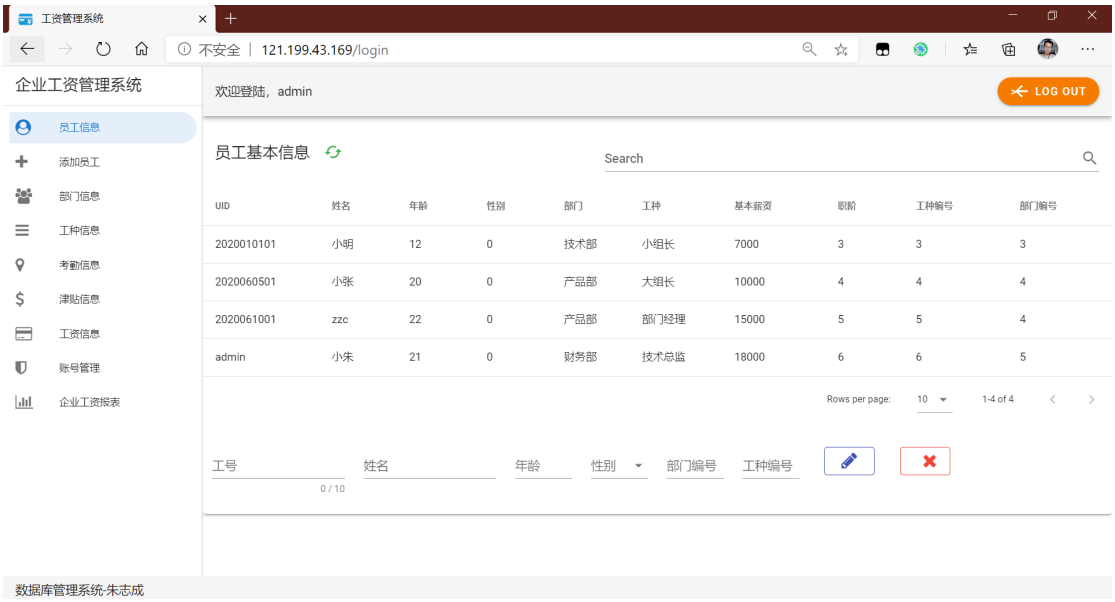

图 4.16 管理员主页面

部门管理及更新页面：

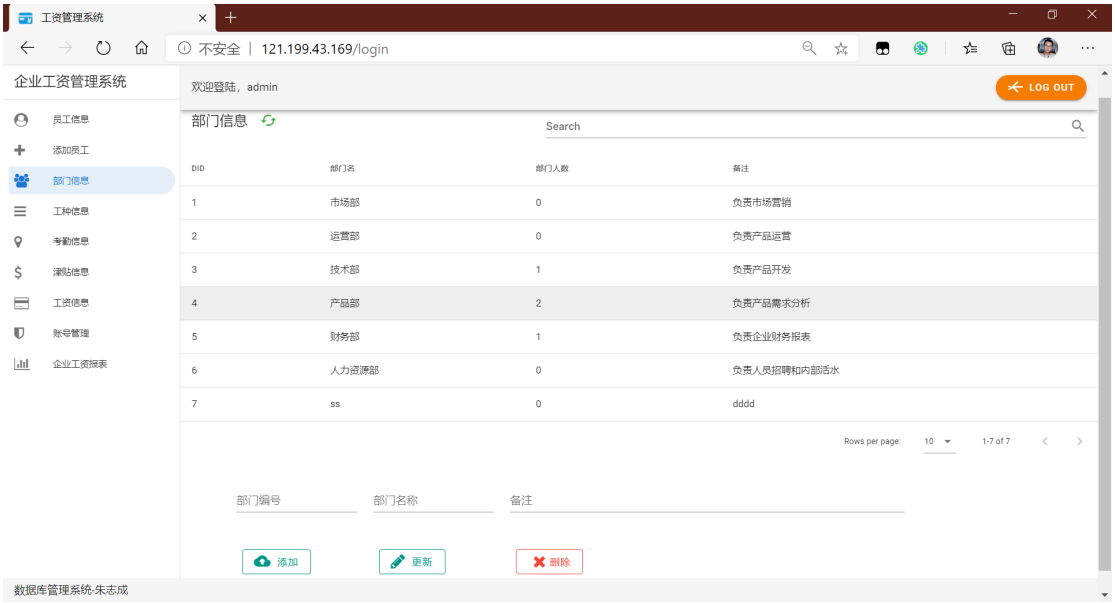

图 4.17 部门信息页面

津贴信息展示及修改页面：

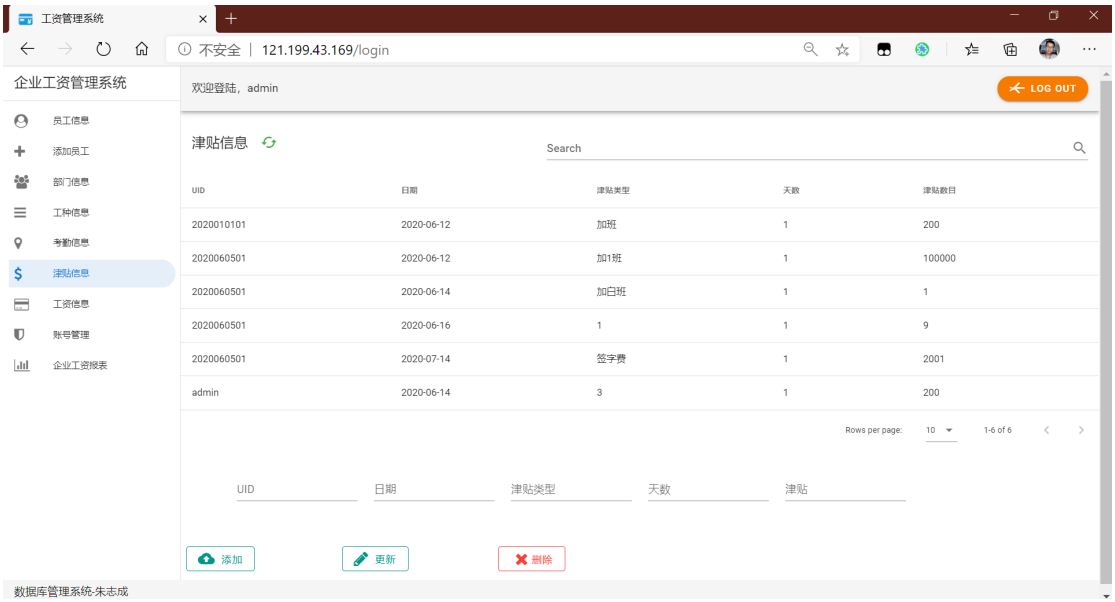

图 4. 18 津贴信息页面

薪资信息展示页面：

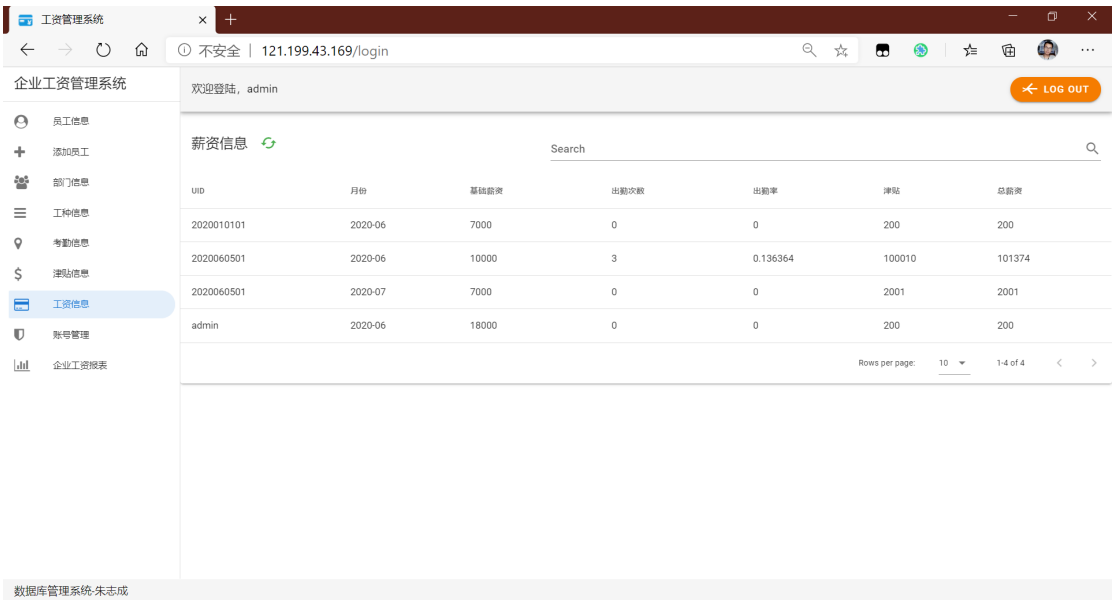

图 4. 19 薪资信息页面

企业工资报表：

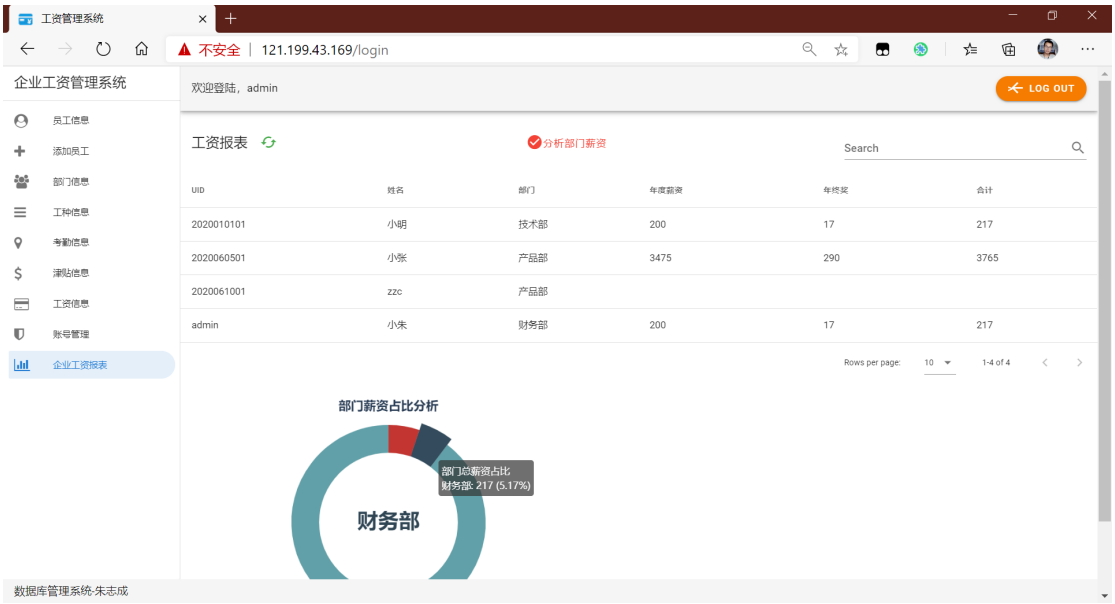

图 4.20 企业工资报表页面

4.6 系统测试

一、登录和注册的数据验证测试

① 用户名或密码错误

当输入错误的用户名和密码时，会刷新登录界面：

图 4.21 登陆页面

点击登录后，由于密码错误(密码为 admin)，会刷新当前登陆页面

图 4.22 返回登陆页面

② 管理员登录测试

输入管理员账号密码进行登录(admin,admin)

图 4.23 登陆页面

成功登录管理员界面

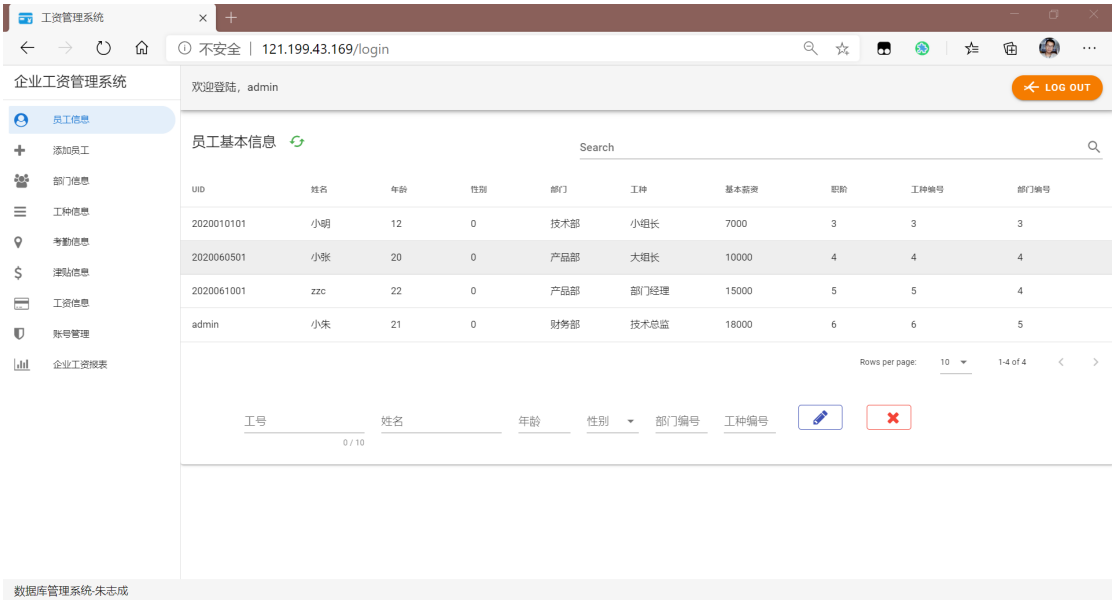

图 4.24 成功登陆页面

③ 登出功能测试

点击管理员界面的 log out

成功清除 session，返回登陆界面

图 4.25 成功登出页面

④ 用户登陆测试

输入用户账号密码进行登录(2020060501, 2020060501)

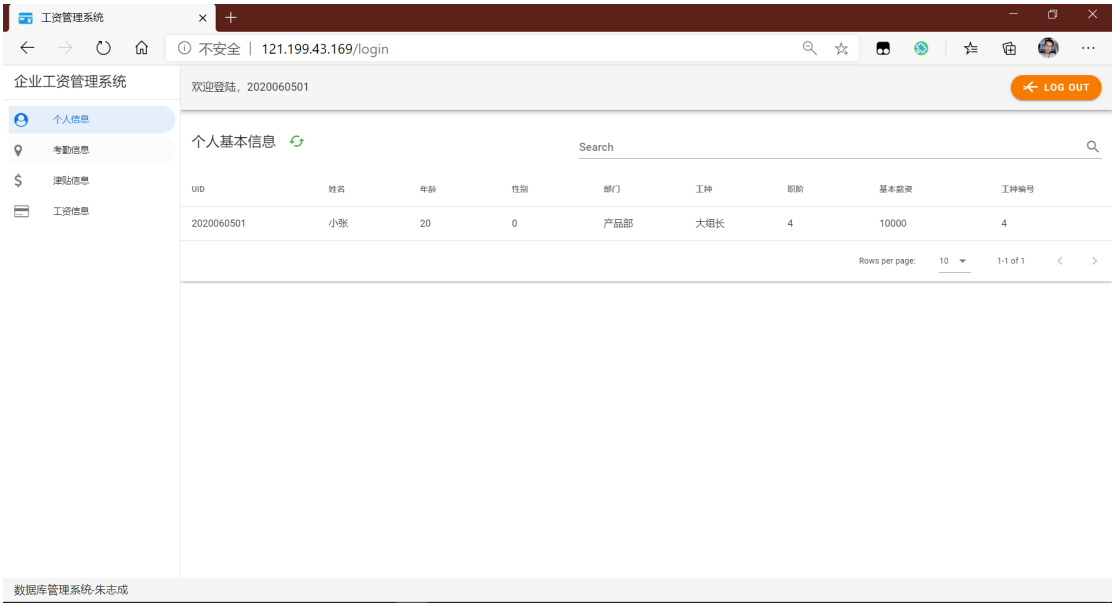

图 4.26 成功登陆用户页面

成功登入用户界面

综上，通过以上的测试，验证登录权限管理功能完善，达到了预期的目标。

二、用户功能测试

① 基本查询功能

登入用户界面后，分别查看个人信息，考勤信息，津贴信息，薪资信息

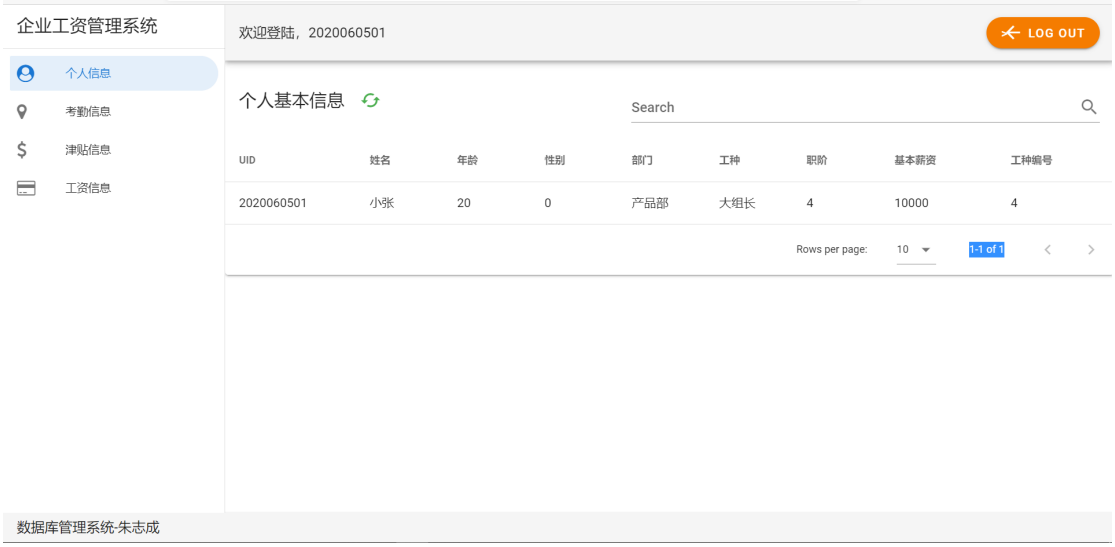

图 4.27 个人信息

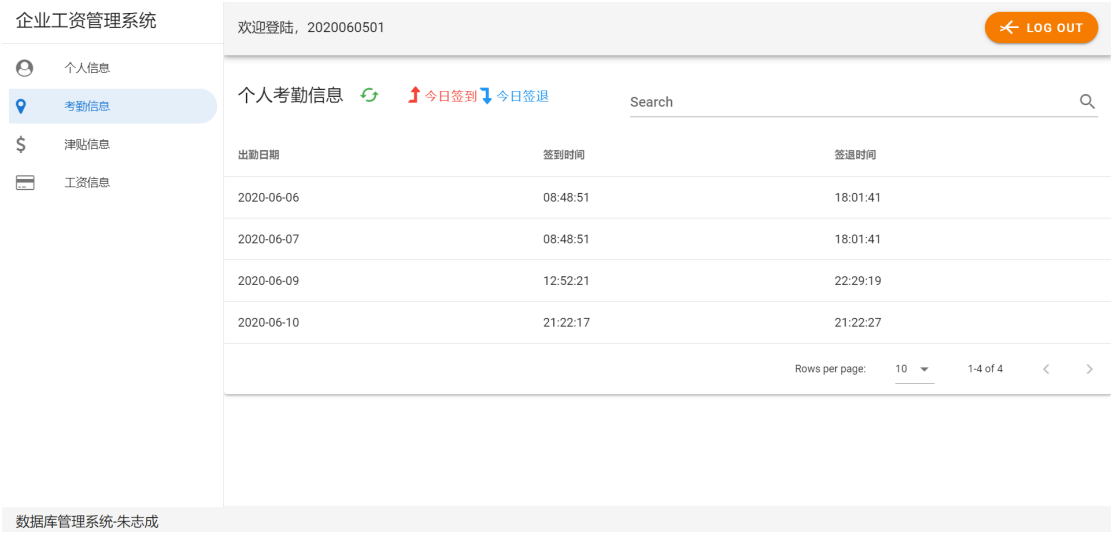

图 4.28 考勤信息

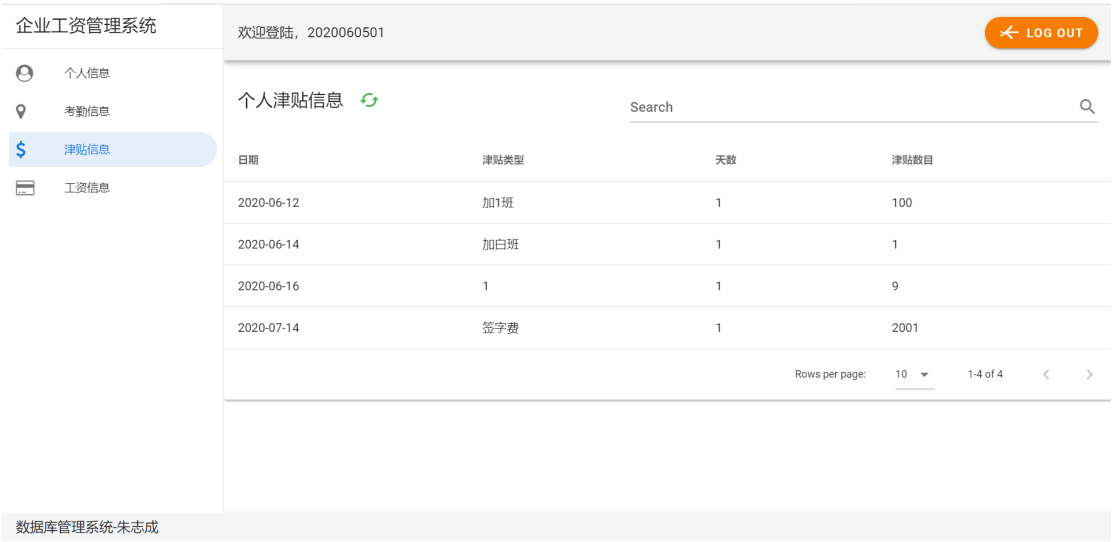

图 4.29 津贴信息

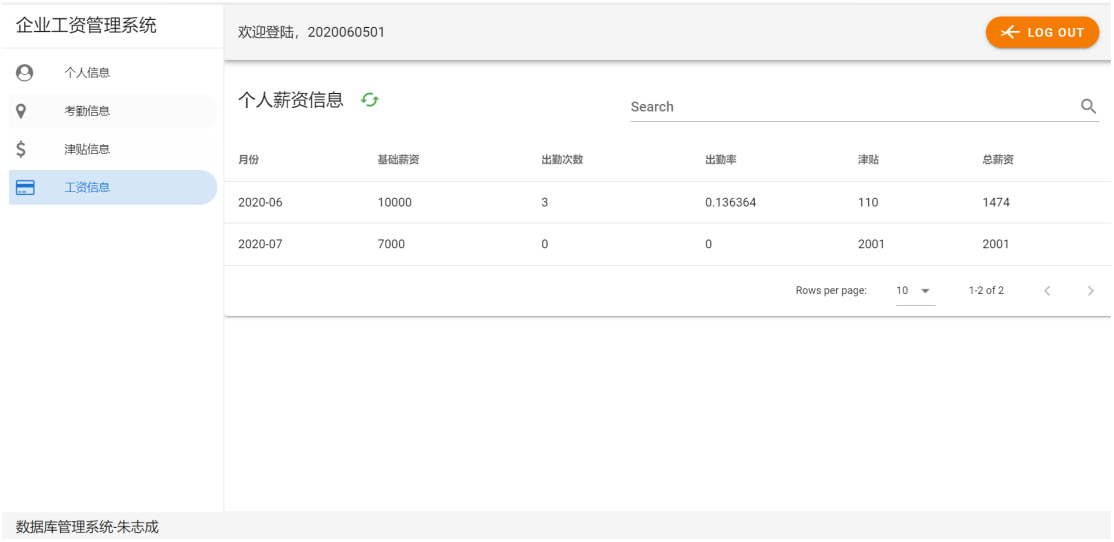

图 4.30 工资信息

② 考勤功能测试

点击签到按钮后刷新考勤表

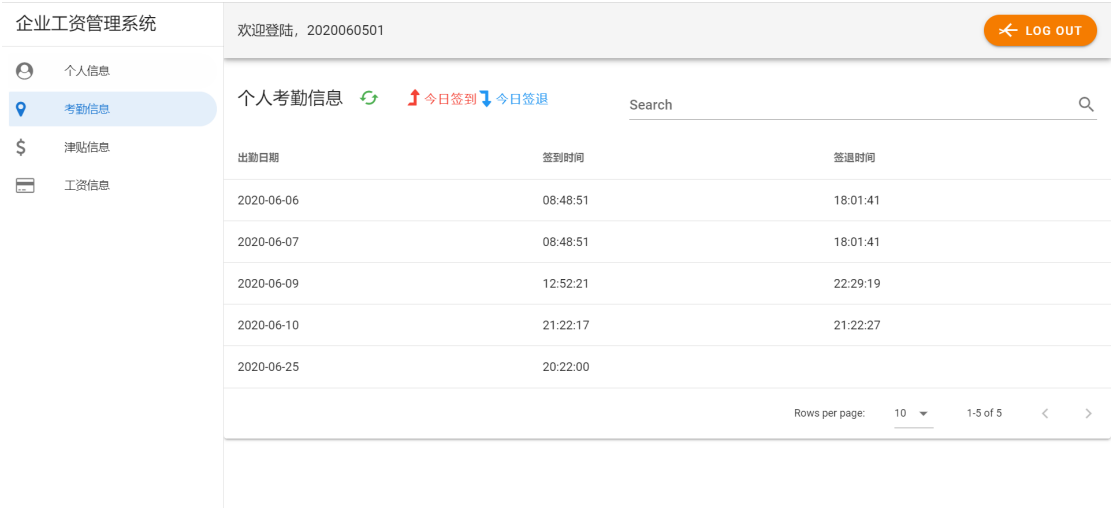

图 4.31 签到功能测试

2020-6-25 成功签到

点击签退按钮后刷新考勤表

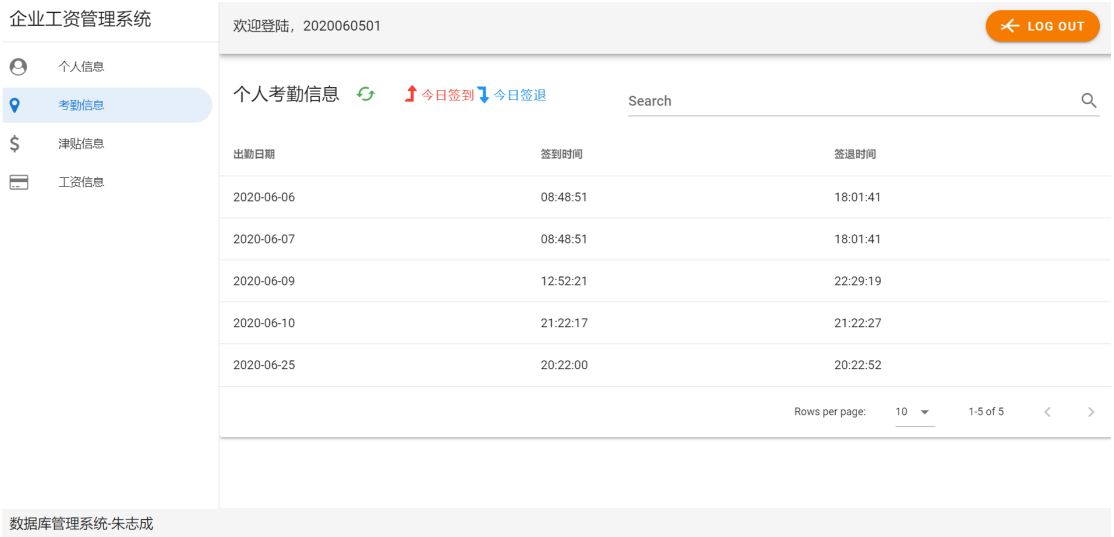

图 4.32 签退功能测试

2020-6-25 成功签退

综上，通过以上的测试，验证系统用户功能完善，达到了预期的目标。

三、管理员功能测试

①添加测试（以员工信息表为例）

初始员工信息表

员工基本信息

Search

UID	姓名	年龄	性别	部门	工种	基本薪资	职阶	工种编号	部门编号
2020010101	小明	12	0	技术部	小组长	7000	3	3	3
2020060501	小张	20	0	产品部	大组长	10000	4	4	4
2020061001	zzc	22	0	产品部	部门经理	15000	5	5	4
admin	小朱	21	0	财务部	技术总监	18000	6	6	5

Rows per page: 10 1-4 of 4

图 4.33 初始员工信息表

添加新员工

添加员工

姓名 小美 2 / 10

年龄 22

性别 1

工号 2020062501

部门编号 2

工种编号 3

RESET SUBMIT

图 4.34 添加新员工

提交结果后刷新基本信息表

员工基本信息

Search

UID	姓名	年龄	性别	部门	工种	基本薪资	职阶	工种编号	部门编号
2020010101	小明	12	0	技术部	小组长	7000	3	3	3
2020060501	小张	20	0	产品部	大组长	10000	4	4	4
2020061001	zzc	22	0	产品部	部门经理	15000	5	5	4
2020062501	小美	22	1	运营部	小组长	7000	3	3	2
admin	小朱	21	0	财务部	技术总监	18000	6	6	5

Rows per page: 10 1-5 of 5

图 4.35 添加结果

刷新登录信息表

账号管理

Search

UID	Password	Authority
2020060501	2020060501	1
2020061001	2020061001	1
2020062501	2020062501	1
admin	admin	0

Rows per page: 10 1-4 of 4

图 4.36 登陆信息表

添加员工信息和登陆账号信息均成功

②修改测试（以工种信息为例）

修改前工种信息

工种信息

Search

KID	工种	职级	基础薪资
1	实习生	1	2000
2	正式工	2	5000
3	小组长	3	7000
4	大组长	4	10000
5	部门经理	5	15000
6	技术总监	6	18000

Rows per page: 10 1-6 of 6

图 4.37 修改前工种信息

修改项

工种编号 1

工种名称 实习生

职阶 1

基本薪资 2001

添加 更新 删除

图 4.38 修改工种信息

更新后刷新工种表

工种信息

Search

KID	工种	职级	基础薪资
1	实习生	1	2001
2	正式工	2	5000
3	小组长	3	7000
4	大组长	4	10000
5	部门经理	5	15000
6	技术总监	6	18000

Rows per page: 10 1-6 of 6

图 4.39 修改后工种信息

KID=2 的基础薪资成功更新为 2001

刷新员工基本信息表

员工基本信息

Search

UID	姓名	年龄	性别	部门	工种	基本薪资	职阶	工种编号	部门编号
2020010101	小明	12	0	技术部	小组长	7000	3	3	3
2020060501	小张	20	0	产品部	大组长	10000	4	4	4
2020061001	zzc	22	0	产品部	部门经理	15000	5	5	4
2020062501	小美	22	1	运营部	实习生	2001	1	1	2
admin	小朱	21	0	财务部	技术总监	18000	6	6	5

Rows per page: 10 1-5 of 5

图 4.40 修改后员工信息

基本信息表也成功更新

③ 企业报表图表分析测试

工资报表情况

工资报表 分析部门薪资 Search

UID	姓名	部门	年度薪资	年终奖	合计
2020010101	小明	技术部	200	17	217
2020060501	小张	产品部	3475	290	3765
2020061001	zzc	产品部			
2020062501	小美	运营部			
admin	小朱	财务部	200	17	217

Rows per page: 10 1-5 of 5

图 4.41 工资报表

点击分析部门薪资

Rows per page: 10 1-5 of 5

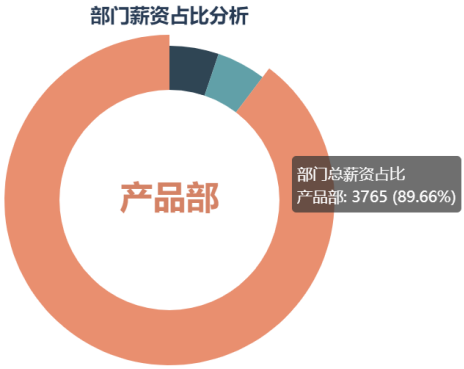

图 4.42 分析部门薪资

分析结果与数据相符

④筛选与搜索功能验证

以管理员的基本员工信息表为例

UID 部分搜索

员工基本信息 202006

UID	姓名	年龄	性别	部门	工种	基本薪资	职阶	工种编号	部门编号
2020060501	小张	20	0	产品部	大组长	10000	4	4	4
2020061001	zzc	22	0	产品部	部门经理	15000	5	5	4
2020062501	小美	22	1	运营部	实习生	2001	1	1	2

Rows per page: 10 1-3 of 3

图 4.43 UID 搜索

姓名搜索

员工基本信息

小明

UID	姓名	年龄	性别	部门	工种	基本薪资	职阶	工种编号	部门编号
2020010101	小明	12	0	技术部	小组长	7000	3	3	3

Rows per page: 10 1-1 of 1

图 4.44 姓名搜索

部门搜索

员工基本信息

产品部

UID	姓名	年龄	性别	部门	工种	基本薪资	职阶	工种编号	部门编号
2020060501	小张	20	0	产品部	大组长	10000	4	4	4
2020061001	zzc	22	0	产品部	部门经理	15000	5	5	4

Rows per page: 10 1-2 of 2

图 4.45 部门搜索

按基本薪资排序

UID	姓名	年龄	性别	部门	工种	基本薪资 ↓	职阶	工种编号	部门编号
admin	小朱	21	0	财务部	技术总监	18000	6	6	5
2020061001	zzc	22	0	产品部	部门经理	15000	5	5	4
2020060501	小张	20	0	产品部	大组长	10000	4	4	4
2020010101	小明	12	0	技术部	小组长	7000	3	3	3
2020062501	小美	22	1	运营部	实习生	2001	1	1	2

图 4.46 排序功能

以上测试说明图表的过滤和搜索功能是完备的

综上，通过以上的测试，验证系统管理员功能完善，达到了预期的目标。

4.7 安全性控制

由于是 web 系统，所以很容易遭到各方面的渗透攻击等，于是在安全性上做

了以下几点完善。

(1) 权限控制

由于 WEB 服务器的特性，所有的页面都可以被用户得到，但对于不同权限的用户，并不是可以访问到服务器上所有的页面，因此对所有的页面都进行了权限控制。当用户登录时，会在 session 中存储 UID 和 authority 权限信息，当管理员登录时，authority 为 0，当一般用户登录时，authority 为 1。退出时，会清除 seeion。所有的页面在加载时，都会对 seesion 值进行检测，不存在则直接跳转到登陆页面，不允许访问。

(2) SQL 防注入

所有的数据获取方式均是以 axsio 的方式实现，数据以 json 格式存储，后端在获取数据时会调用 get_json 函数进行数据赋值，而不是简单的“=”赋值

```
DID = request.get_json()['DID']
DName = request.get_json()['DName']
DRef = request.get_json()['DRef']
```

该操作可以替换掉例如“,”“=”“-”等等敏感的可以利用来进行 SQL 注入的字符串，从而保证了系统的安全性。

(3) 数据库备份要求

数据库支持热备份，编写 shell 脚本部署在服务器上，设置为定时任务，每天特定时间对 mysql 数据库内容进行备份并将备份情况记录到日志中去。当发生意外情况导致数据库崩溃或数据丢失时，可以及时进行数据库的恢复

备份脚本文件内容见附录

4.8 系统设计与实现总结

整个系统的开发历时 10 天

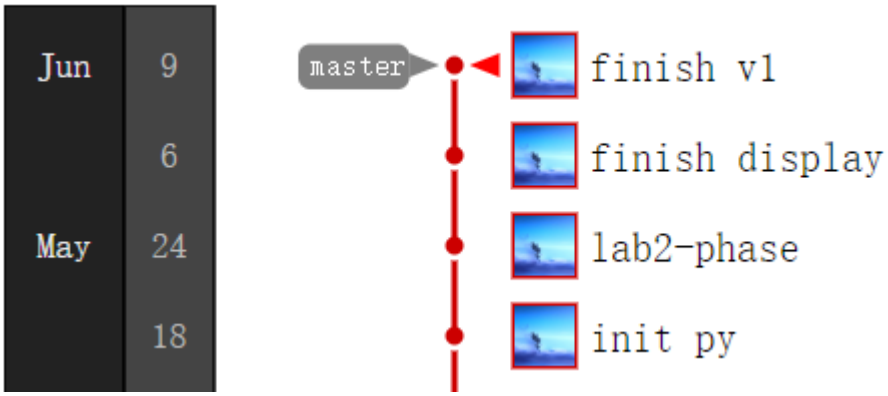

图 4.47 开发历程

整个过程分为以下几个阶段：

① 5 月中下旬开始了解前端和后端的技术框架，B/S 和 C/S 的优劣等，结合项目实际需求，最终选定采用 B/S 架构进行开发

② 确定架构后开始了解对应的技术框架，前端了解的包括原生实现，bootstrap，vue，echart，element-ui，vuetify，后端了解的包括 flask，php，express，数据库包括 mysq

③ 完成数据库设计

撰写文档，设计数据库表类型、ER 图、数据流图等，并在 MySQL 中建立相关的表和建立完整性约束。

④ 初次系统架构设计，开始正式开发

选定实现方案为：原生 html+bootstrap+flask+mysql，设计好了整个系统的流程，规划好了要写哪些模块，哪些文件。

⑤ 界面设计

先选定和设计了基本的界面图，着手实现，同时开始写后端 flask 逻辑

⑥ 重新设计系统架构

到 6 月 6 日做出的 V1 版本和预期有较大差距，重新选定实现方案：在原先的基础上使用了 vue 部分特性，嵌入了 vuetify 图表，后端加上 nginx 做路由分发。

⑦ 第二次实现前端页面，利用 vue 做动态控制

⑧ 后台逻辑完善

编写后台接口，同时和 mysql 数据库交互

⑨ 完善数据库事务，触发器，存储过程，重新设计部分表

⑩ 部署系统到云端，进行系统联调，功能测试与完善

4 课程总结

本次实验在我看来像是一个升级版的软件工程，完整的经历了一个软件项目开发的过程，收获巨大。

之前在软件工程的课程中最后做的也是一个 web 项目，但是当时没有添加数据库，没有后端，只包括了前端和 js 的逻辑项，相比之下，数据库实验实现的系统更加完整，web 的各个方面都有很好的体现。

本次的系统开发过程并不顺利，由于自己对于 web 相关技术不是很了解，在前期调研的时候花费了大量精力，包括对相关技术的了解，不同技术的优劣和适用场景，以及开发初期的试错和项目重构，都花费了大量的时间，导致在最终确定好技术框架后，留给我的时间不是那么充足，最终实现的效果和目标还有些许差距。

本次实验的过程中我收获了许多，在确定好要用 B/S 架构实现以后，我先去复习了一遍 html+css+js 的基础知识，又有了更进一步的认识，并在此基础上学习了 ajax 相关知识。在技术选择的过程中，我对于常见的 web 技术和应用场景都有了基本的认识。通过构建完整系统的过程，我学会了服务器 web 项目的部署，前后端的交互方式，数据库的正确使用，接口的设计规范，项目的迭代和重构等知识，对于自己能力的提高有很大的帮助。

总结整个开发历程，我觉得，自己学会的不仅仅是数据库的知识，更提升了自己的学习能力，解决问题的能力，在这个过程中，看博客，读文档已经成为了常态，我也迅速学会了很多东西。

本次数据库实验中让我觉得较为麻烦的是实验二的数据处理方面，在数据导入的过程中碰到了不少麻烦，不过也学会了存储过程的相关知识（虽然过程体验不是很好）。综合实践很令人满意，我能很明显的感觉出自己的进步，投入的时间对于能力的提升效果明显。

感谢课设，感谢老师的指导，也感谢自己这两个星期的付出。今后我将继续保持在数据库系统开发期间的学习热情，碰到问题积极解决，做到更好。

5 附录

附录 1 数据库备份脚本 backup.sh

```
#!/bin/bash
#功能说明：本功能用于备份 mysql 数据库
#编写日期：2020/06/9
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/local/mysql/bin
export PATH
#数据库用户名
dbuser='user'
#数据库密码
dbpasswd='3323150'
#数据库名,可以定义多个数据库，中间以空格隔开，如：test test1 test2
dbname='test'
#备份时间
backtime=`date +%Y%m%d%H%M%S`
#日志备份路径
logpath='/opt/mysqlbackup/log'
#数据备份路径
datapath='/opt/mysqlbackup'
#日志记录头部
echo ‘”备份时间为${backtime},备份数据库表 ${dbname} 开始” >> ${logpath}/mysqllog.log
#正式备份数据库
for table in $dbname; do
source=`mysqldump -u${dbuser} -p${dbpasswd} --single-transaction ${table}> ${datapath}/${backtime}.sql` 2>> ${logpath}/mysqllog.log;
#备份成功以下操作
if [ "$?" == 0 ];then
cd $datapath
#为节约硬盘空间，将数据库压缩
tar jcf ${table}${backtime}.tar.bz2 ${backtime}.sql > /dev/null
#删除原始文件，只留压缩后文件
rm -f ${datapath}/${backtime}.sql
echo “数据库表 ${dbname} 备份成功!!” >> ${logpath}/mysqllog.log
```

```
else
#备份失败则进行以下操作
echo “数据库表 ${dbname} 备份失败!!” >> ${logpath}/mysqllog.log
fi
done
```

附录 2 数据库表和相关结构创建

```
/*
 Navicat Premium Data Transfer

 Source Server         : aliyun
 Source Server Type    : MySQL
 Source Server Version : 50730
 Source Host           : localhost:3306
 Source Schema         : test

 Target Server Type    : MySQL
 Target Server Version : 50730
 File Encoding         : 65001

 Date: 23/06/2020 23:30:24
*/

SET NAMES utf8mb4;
SET FOREIGN_KEY_CHECKS = 0;

-- ----------------------------
-- Table structure for attendance_info
-- ----------------------------
DROP TABLE IF EXISTS `attendance_info`;
CREATE TABLE `attendance_info`  (
  `UID` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `ADate` date NOT NULL,
  `ATime` time(0) NULL DEFAULT NULL,
  `STime` time(0) NULL DEFAULT NULL,
  PRIMARY KEY (`UID`, `ADate`) USING BTREE,
  CONSTRAINT `attendance_info_ibfk_1` FOREIGN KEY (`UID`) REFERENCES
`employee_info` (`UID`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci
ROW_FORMAT = DYNAMIC;
```

```
-- ----------------------------
-- Table structure for bonus_info
-- ----------------------------
DROP TABLE IF EXISTS `bonus_info`;
CREATE TABLE `bonus_info`  (
  `UID` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `BDate` date NOT NULL,
  `BType` char(20) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
  `BDays` smallint(6) NULL DEFAULT NULL,
  `Bonus` int(11) NULL DEFAULT NULL,
  PRIMARY KEY (`UID`, `BDate`, `BType`) USING BTREE,
  CONSTRAINT `bonus_info_ibfk_1` FOREIGN KEY (`UID`) REFERENCES
`employee_info` (`UID`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci
ROW_FORMAT = DYNAMIC;

-- ----------------------------
-- Table structure for department_info
-- ----------------------------
DROP TABLE IF EXISTS `department_info`;
CREATE TABLE `department_info`  (
  `DID` smallint(6) NOT NULL,
  `DName` char(30) CHARACTER SET utf8 COLLATE utf8_general_ci NULL
DEFAULT NULL,
  `DRef` varchar(255) CHARACTER SET utf8 COLLATE utf8_general_ci NULL
DEFAULT NULL,
  `Dnum` smallint(6) NULL DEFAULT NULL,
  PRIMARY KEY (`DID`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci
ROW_FORMAT = DYNAMIC;

-- ----------------------------
-- Table structure for employee_info
-- ----------------------------
```

```
DROP TABLE IF EXISTS `employee_info`;
CREATE TABLE `employee_info`  (
  `UID` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `UName` char(30) CHARACTER SET utf8 COLLATE utf8_general_ci NULL
DEFAULT NULL,
  `Age` smallint(6) NULL DEFAULT NULL,
  `Sex` tinyint(1) NULL DEFAULT NULL,
  `KID` smallint(6) NULL DEFAULT NULL,
  `DID` smallint(6) NULL DEFAULT NULL,
  PRIMARY KEY (`UID`) USING BTREE,
  INDEX `KID`(`KID`) USING BTREE,
  INDEX `DID`(`DID`) USING BTREE,
  CONSTRAINT `employee_info_ibfk_1` FOREIGN KEY (`KID`) REFERENCES
`kind_info` (`KID`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `employee_info_ibfk_2` FOREIGN KEY (`DID`) REFERENCES
`department_info` (`DID`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci
ROW_FORMAT = DYNAMIC;

-- ----------------------------
-- Table structure for kind_info
-- ----------------------------
DROP TABLE IF EXISTS `kind_info`;
CREATE TABLE `kind_info`  (
  `KID` smallint(6) NOT NULL,
  `KName` char(30) CHARACTER SET utf8 COLLATE utf8_general_ci NULL
DEFAULT NULL,
  `Level` smallint(6) NULL DEFAULT NULL,
  `Base_salary` int(11) NULL DEFAULT NULL,
  PRIMARY KEY (`KID`) USING BTREE
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci
ROW_FORMAT = DYNAMIC;

-- ----------------------------
-- Table structure for log_info
```

```
-- ----------------------------
DROP TABLE IF EXISTS `log_info`;
CREATE TABLE `log_info`  (
  `UID` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `Password` char(30) CHARACTER SET utf8 COLLATE utf8_general_ci NULL
DEFAULT NULL,
  `Authority` smallint(6) NULL DEFAULT NULL,
  PRIMARY KEY (`UID`) USING BTREE,
  CONSTRAINT `log_info_ibfk_1` FOREIGN KEY (`UID`) REFERENCES
`employee_info` (`UID`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci
ROW_FORMAT = DYNAMIC;

-- ----------------------------
-- Table structure for salary_info
-- ----------------------------
DROP TABLE IF EXISTS `salary_info`;
CREATE TABLE `salary_info`  (
  `UID` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `Month` date NOT NULL,
  `Base_salary` int(11) NULL DEFAULT NULL,
  `attendance_times` smallint(6) NULL DEFAULT NULL,
  `attendance_rate` float NULL DEFAULT NULL,
  `Bonus` int(11) NULL DEFAULT NULL,
  `Total_salary` int(11) NULL DEFAULT NULL,
  PRIMARY KEY (`UID`, `Month`) USING BTREE,
  CONSTRAINT `salary_info_ibfk_1` FOREIGN KEY (`UID`) REFERENCES
`employee_info` (`UID`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB CHARACTER SET = utf8 COLLATE = utf8_general_ci
ROW_FORMAT = DYNAMIC;

-- ----------------------------
-- View structure for view_get_employee_info
-- ----------------------------
DROP VIEW IF EXISTS `view_get_employee_info`;
```

```
CREATE ALGORITHM = UNDEFINED SQL SECURITY DEFINER VIEW
`view_get_employee_info` AS select `employee_info`.`UID` AS
`UID`,`employee_info`.`UName` AS `UName`,`employee_info`.`Age` AS
`Age`,`employee_info`.`Sex` AS `Sex`,`department_info`.`DName` AS
`DName`,`kind_info`.`KName` AS `KName`,`kind_info`.`Level` AS
`Level`,`kind_info`.`Base_salary` AS `Base_salary`,`kind_info`.`KID` AS
`KID`,`department_info`.`DID` AS `DID` from ((`employee_info` join
`department_info`) join `kind_info`) where ((`employee_info`.`KID` =
`kind_info`.`KID`) and (`employee_info`.`DID` = `department_info`.`DID`));

-- ----------------------------
-- View structure for view_get_salary_report
-- ----------------------------
DROP VIEW IF EXISTS `view_get_salary_report`;
CREATE ALGORITHM = UNDEFINED SQL SECURITY DEFINER VIEW
`view_get_salary_report` AS select `employee_info`.`UID` AS
`UID`,`employee_info`.`UName` AS `UName`,`department_info`.`DName` AS
`DName`,`get_salary_year`(`employee_info`.`UID`,'2020') AS
`Salary`,`get_year_end_awards_employee`(`employee_info`.`UID`,'2020') AS
`Award` from (`employee_info` join `department_info`) where
(`employee_info`.`DID` = `department_info`.`DID`);

-- ----------------------------
-- Procedure structure for attendance_update
-- ----------------------------
DROP PROCEDURE IF EXISTS `attendance_update`;
delimiter ;;
CREATE PROCEDURE `attendance_update`(UID CHAR ( 10 ), ADate date, ATime
time, STime time, type INT)
BEGIN
    DECLARE
        flag INT;
    SELECT
        count(*) INTO flag
    FROM
```

```
        attendance_info
    WHERE
        attendance_info.UID = UID
        AND attendance_info.ADate = ADate;
    IF
        type = 0
        AND flag = 0 THEN
            INSERT INTO attendance_info
        VALUES
            ( UID, ADate, ATime, NULL );

        ELSEIF type = 1
        AND flag = 1 THEN
            UPDATE attendance_info
            SET attendance_info.STime = STime
            WHERE
                attendance_info.UID = UID
                AND attendance_info.ADate = ADate;

        END IF;
END
;;
delimiter ;

-- ----------------------------
-- Procedure structure for calculate_salary_per_month
-- ----------------------------
DROP PROCEDURE IF EXISTS `calculate_salary_per_month`;
delimiter ;;
CREATE PROCEDURE `calculate_salary_per_month`(UID CHAR ( 10 ),
whichmonth date)
BEGIN
    DECLARE
        Base_salary INT;
    DECLARE
```

```
        Bonus INT;
    DECLARE
        Total_salary INT;
    DECLARE
        attendance_times INT;
    DECLARE
        attendance_rate FLOAT;

    SET Base_salary = get_base_salary ( UID );

    SET Bonus = get_bonus_month ( UID, whichmonth );
    SELECT
        COUNT(*) INTO attendance_times
    FROM
        attendance_info
    WHERE
        attendance_info.UID = UID AND DATE_FORMAT(  attendance_info.ADate,
'%Y-%m' ) = DATE_FORMAT( whichmonth, '%Y-%m' )
        AND TIMESTAMPDIFF( HOUR, attendance_info.ATime,
attendance_info.STime ) >= 8;

    SET attendance_rate = attendance_times / 22;

    SET Total_salary = Bonus + Base_salary * attendance_rate;
    DELETE
    FROM
        salary_info
    WHERE
        salary_info.UID = UID
        AND DATE_FORMAT( salary_info.`Month`, '%Y-%m' ) =
DATE_FORMAT( whichmonth, '%Y-%m' );
    INSERT INTO salary_info
    VALUES
   ( UID, CONCAT( DATE_FORMAT( whichmonth, '%Y-%m' ), '-01' ),Base_salary,
attendance_times, attendance_rate,    Bonus, Total_salary );
```

```
END
;;
delimiter ;

-- ----------------------------
-- Function structure for get_base_salary
-- ----------------------------
DROP FUNCTION IF EXISTS `get_base_salary`;
delimiter ;;
CREATE FUNCTION `get_base_salary`(UID CHAR ( 10 ))
  RETURNS int(11)
BEGIN
      DECLARE
            ret INT;
      SELECT
            Base_salary
      FROM
            employee_info,
            kind_info
      WHERE
            employee_info.UID = UID
            AND employee_info.KID = kind_info.KID INTO ret;
RETURN ret;
END
;;
delimiter ;

-- ----------------------------
-- Function structure for get_bonus_month
-- ----------------------------
DROP FUNCTION IF EXISTS `get_bonus_month`;
delimiter ;;
CREATE FUNCTION `get_bonus_month`(UID CHAR ( 10 ), whichmonth date)
  RETURNS int(11)
BEGIN
```

```
    DECLARE
        total_bonus INT;
    SELECT
        SUM( Bonus )
    FROM
        bonus_info
    WHERE
        bonus_info.UID = UID
        AND DATE_FORMAT( whichmonth, '%Y-%m' ) =
DATE_FORMAT( bonus_info.BDate, '%Y-%m' ) INTO total_bonus;
    RETURN total_bonus;

END
;;
delimiter ;

-- ----------------------------
-- Function structure for get_salary_year
-- ----------------------------
DROP FUNCTION IF EXISTS `get_salary_year`;
delimiter ;;
CREATE FUNCTION `get_salary_year`(UID CHAR ( 10 ),
    whichyear CHAR ( 10 ))
 RETURNS int(11)
BEGIN
    DECLARE
        total INT;
    SELECT
        SUM( salary_info.Total_salary )
    FROM
        salary_info
    WHERE
        salary_info.UID = UID
        AND DATE_FORMAT( salary_info.MONTH, '%Y' ) = whichyear INTO
total;
```

```
RETURN total;
END
;;
delimiter ;

-- ----------------------------
-- Function structure for get_year_end_awards_employee
-- ----------------------------
DROP FUNCTION IF EXISTS `get_year_end_awards_employee`;
delimiter ;;
CREATE FUNCTION `get_year_end_awards_employee`(UID CHAR ( 10 ),
    whichyear CHAR ( 10 ))
 RETURNS int(11)
BEGIN
        DECLARE
            ret INT;
        SET ret = get_salary_year(UID,whichyear) / 12;
        RETURN ret;

    END
;;
delimiter ;

-- ----------------------------
-- Function structure for plus
-- ----------------------------
DROP FUNCTION IF EXISTS `plus`;
delimiter ;;
CREATE FUNCTION `plus`(arg1 INT, arg2 INT)
 RETURNS int(11)
BEGIN
    DECLARE
        total INT;

    SET total = arg1 + arg2;
```

```
RETURN total;
END
;;
delimiter ;

-- ----------------------------
-- Triggers structure for table attendance_info
-- ----------------------------
DROP TRIGGER IF EXISTS `get_salary_permonth_when_update_attendance`;
delimiter ;;
CREATE TRIGGER `get_salary_permonth_when_update_attendance` AFTER
UPDATE ON `attendance_info` FOR EACH ROW BEGIN
    IF
        ! isnull( NEW.STime ) THEN
            CALL calculate_salary_per_month ( NEW.UID, NEW.ADate );

    END IF;
END
;;
delimiter ;

-- ----------------------------
-- Triggers structure for table bonus_info
-- ----------------------------
DROP TRIGGER IF EXISTS `calculate_salary_per_month_when_insert_bonus`;
delimiter ;;
CREATE TRIGGER `calculate_salary_per_month_when_insert_bonus` AFTER
INSERT ON `bonus_info` FOR EACH ROW BEGIN
        CALL calculate_salary_per_month ( NEW.UID, NEW.BDate );

END
;;
delimiter ;

-- ----------------------------
```

```
-- Triggers structure for table bonus_info
-- ----------------------------
DROP TRIGGER IF EXISTS `calculate_salary_per_month_when_update_bonus`;
delimiter ;;
CREATE TRIGGER `calculate_salary_per_month_when_update_bonus` AFTER
UPDATE ON `bonus_info` FOR EACH ROW BEGIN
        CALL calculate_salary_per_month ( OLD.UID, OLD.BDate );
    IF
        OLD.BDate != NEW.BDate THEN
            CALL calculate_salary_per_month ( NEW.UID, OLD.BDate );

    END IF;
END
;;
delimiter ;

-- ----------------------------
-- Triggers structure for table bonus_info
-- ----------------------------
DROP TRIGGER IF EXISTS `calculate_salary_per_month_when_delete_bonus`;
delimiter ;;
CREATE TRIGGER `calculate_salary_per_month_when_delete_bonus` AFTER
DELETE ON `bonus_info` FOR EACH ROW BEGIN
        CALL calculate_salary_per_month ( OLD.UID, OLD.BDate );

END
;;
delimiter ;

-- ----------------------------
-- Triggers structure for table employee_info
-- ----------------------------
DROP TRIGGER IF EXISTS `add_new_log`;
delimiter ;;
CREATE TRIGGER `add_new_log` AFTER INSERT ON `employee_info` FOR
```

```
EACH ROW BEGIN
        INSERT INTO log_info
    VALUES
    ( new.UID, new.UID, 1 );
END
;;
delimiter ;

-- ----------------------------
-- Triggers structure for table employee_info
-- ----------------------------
DROP TRIGGER IF EXISTS `department_people_num_plus`;
delimiter ;;
CREATE TRIGGER `department_people_num_plus` AFTER INSERT ON
`employee_info` FOR EACH ROW BEGIN
        UPDATE department_info
        SET Dnum = IFNULL( Dnum, 0 )+ 1
    WHERE
    DID = new.DID;
END
;;
delimiter ;

-- ----------------------------
-- Triggers structure for table employee_info
-- ----------------------------
DROP TRIGGER IF EXISTS `department_people_num_update`;
delimiter ;;
CREATE TRIGGER `department_people_num_update` AFTER UPDATE ON
`employee_info` FOR EACH ROW BEGIN
    IF
        ( old.DID != new.DID ) THEN
            UPDATE department_info
            SET Dnum = Dnum + 1
        WHERE
```

```
            DID = new.DID;
        UPDATE department_info
        SET Dnum = Dnum - 1
        WHERE
            DID = old.DID;

    END IF;
END
;;
delimiter ;

-- ----------------------------
-- Triggers structure for table employee_info
-- ----------------------------
DROP TRIGGER IF EXISTS `del_old_log`;
delimiter ;;
CREATE TRIGGER `del_old_log` BEFORE DELETE ON `employee_info` FOR
EACH ROW BEGIN
    DELETE FROM log_info where log_info.UID=OLD.UID;
END
;;
delimiter ;

-- ----------------------------
-- Triggers structure for table employee_info
-- ----------------------------
DROP TRIGGER IF EXISTS `department_people_num_minus`;
delimiter ;;
CREATE TRIGGER `department_people_num_minus` AFTER DELETE ON
`employee_info` FOR EACH ROW BEGIN
        UPDATE department_info
        SET Dnum = Dnum - 1
    WHERE
    DID = old.DID;
END
```

```
;;
delimiter ;

SET FOREIGN_KEY_CHECKS = 1;
```

附录 3 后台路由处理主程序 run.py

```
# from method.db_utils import *
import method.db_utils
from flask import Flask, redirect, url_for, request, render_template, jsonify,session
from method.json_utils import tupletojson
import datetime

app = Flask(__name__)

app.secret_key = '168017'

db = method.db_utils.DataBaseHandle('127.0.0.1', 'root', '3323150', 'test', 3306)

@app.route("/")
def login():
    return render_template('login.html')

# @app.route("/login/post", methods=("GET", "POST"))
@app.route("/login", methods=("GET","POST"))
def login_post():
    if request.method=="GET":
        return render_template('login.html')
    if request.method=="POST":
        uid = request.form.get('uid')
        password = request.form.get('password')
        data = db.query('select * from log_info where UID = %s', uid, one=True)
        if (data[1] == password):
            authority = data[2]
            session['uid'] = uid
            session['password'] = password
            session['authority'] = authority
            session.permanent = True
            if(authority == 0):
                return render_template('index.html')
            else:
```

```
                return render_template('index-user.html')
        return render_template('login.html')

@app.route("/logout", methods=["get"])
def logout():
    session.clear()
    return {'res':'sucess'}

@app.context_processor
def my_context_processor():
    user = session.get('uid')
    if user:
        return {'login_user': user}
    return {}

@app.route("/index")
def index():
    if session:
        if session['authority'] == 0:
            return render_template('index.html')
        else:
            return render_template('index-user.html')
    else :
        return render_template('login.html')


@app.route("/index/get/employee_info", methods=["GET"])
def get_employee_info():
    sql = '''SELECT * from view_get_employee_info'''

    sql_user = '''SELECT * from view_get_employee_info where UID = %s'''

    if(session['authority'] == 1):
        data = db.query(sql_user,session['uid'])
    else:
```

```
        data = db.query(sql)
    dict = ('UID', 'UName', 'Age', 'Sex', 'DName', 'KName', 'Level',
'Base_salary','KID','DID')
    res = tupletojson(data, dict)
    return jsonify(res)


@app.route("/index/get/department_info", methods=["GET"])
def get_department_info():
    sql = '''select * from department_info'''
    data = db.query(sql)
    dict = ('DID', 'DName', 'DRef', 'DNum')
    res = tupletojson(data, dict)
    return jsonify(res)


@app.route("/index/get/kind_info", methods=["GET"])
def get_kind_info():
    sql = '''select * from kind_info'''
    data = db.query(sql)
    dict = ('KID', 'KName', 'Level', 'Base_salary')
    res = tupletojson(data, dict)
    return jsonify(res)


@app.route("/index/get/bonus_info", methods=["GET"])
def get_bonus_info():
    db = method.db_utils.DataBaseHandle('127.0.0.1', 'root', '3323150', 'test', 3306)
    sql = '''select UID, DATE_FORMAT(BDate, '%Y-%m-%d') as BDate,
BType,BDays,Bonus from bonus_info'''
    sql_user = '''select UID, DATE_FORMAT(BDate, '%%Y-%%m-%%d') as BDate,
BType,BDays,Bonus from bonus_info where UID = %s'''
    if(session['authority'] == 1):
        data = db.query(sql_user,session['uid'])
    else:
```

```
        data = db.query(sql)
    dict = ('UID', 'BDate', 'BType', 'BDays', 'Bonus')
    res = tupletojson(data, dict)
    return jsonify(res)


@app.route("/index/get/attendance_info", methods=["GET"])
def get_attendance_info():
    sql = '''select UID, DATE_FORMAT(ADate, '%Y-%m-%d') as
ADate ,TIME_FORMAT(ATime,'%T') as ATime,TIME_FORMAT(STime,'%T') as
STime from attendance_info'''
    sql_user = '''select UID,    DATE_FORMAT(ADate, '%%Y-%%m-%%d') as
ADate ,TIME_FORMAT(ATime,'%%T') as ATime,TIME_FORMAT(STime,'%%T')
as STime from attendance_info where UID = %s'''
    if(session['authority'] == 1):
        data = db.query(sql_user,session['uid'])
    else:
        data = db.query(sql)
    dict = ('UID', 'ADate', 'ATime', 'STime')
    res = tupletojson(data, dict)
    return jsonify(res)


@app.route("/index/get/salary_info", methods=["GET"])
def get_salary_info():
    sql = '''select UID, DATE_FORMAT(`Month`,'%Y-%m')`Month`,
Base_salary,attendance_times,attendance_rate ,Bonus, Total_salary from salary_info'''
    sql_user = '''select UID, DATE_FORMAT(`Month`,'%%Y-%%m')`Month`,
Base_salary,attendance_times,attendance_rate ,Bonus, Total_salary from salary_info
where UID = %s'''
    if(session['authority'] == 1):
        data = db.query(sql_user,session['uid'])
    else:
        data = db.query(sql)
    dict = ('UID', 'Month', 'Base_salary','attendance_times','attendance_rate' ,'Bonus',
```

```
'Total_salary')
    res = tupletojson(data, dict)
    return jsonify(res)

@app.route("/index/get/log_info", methods=["GET"])
def get_log_info():
    sql = '''select * from log_info'''
    data = db.query(sql)
    dict = ('UID', 'Password', 'Authority')
    res = tupletojson(data, dict)
    return jsonify(res)

@app.route("/index/get/salary_report", methods=["GET"])
def get_salary_report():
    sql = '''SELECT
    UID,UName,DName,Salary,Award,Salary+Award as Total FROM
view_get_salary_report;'''
    data = db.query(sql)
    dict = ('UID', 'UName', 'DName', 'Salary', 'Award' ,'Total')
    res = tupletojson(data, dict)
    return jsonify(res)

@app.route("/index/add/employee_info", methods=["POST"])
def employee_info_add():
    UID = request.get_json()['UID']
    UName = request.get_json()['UName']
    Age = int(request.get_json()['Age'])
    KID = int(request.get_json()['KID'])
    DID = int(request.get_json()['DID'])
    Sex = int(request.get_json()['Sex'])
    sql = '''insert into employee_info values(%s,%s,%s,%s,%s,%s)'''
    dict = [UID,UName,Age,Sex,KID,DID]
    data = db.query(sql,dict)
    return jsonify(data)
```

```
@app.route("/index/update/employee_info", methods=["POST"])
def employee_info_update():
    UID = request.get_json()['UID']
    UName = request.get_json()['UName']
    Age = int(request.get_json()['Age'])
    KID = int(request.get_json()['KID'])
    DID = int(request.get_json()['DID'])
    Sex = int(request.get_json()['Sex'])
    sql = '''update employee_info set UName= %s , Age= %s ,Sex= %s, KID = %s,
DID = %s where UID = %s'''
    dict = [UName,Age,Sex,KID,DID,UID]
    data = db.query(sql,dict)
    return jsonify(data)

@app.route("/index/delete/employee_info", methods=["POST"])
def employee_info_delete():
    UID = request.get_json()['UID']
    sql = '''delete from employee_info where UID = %s'''
    data = db.query(sql,UID)
    return jsonify(data)


@app.route("/index/add/kind_info", methods=["POST"])
def kind_info_add():
    KID = request.get_json()['KID']
    KName = request.get_json()['KName']
    Level = int(request.get_json()['Level'])
    Base_salary = int(request.get_json()['Base_salary'])
    sql = '''insert into kind_info values(%s,%s,%s,%s)'''
    dict = [KID,KName,Level,Base_salary]
    data = db.query(sql,dict)
    return jsonify(data)

@app.route("/index/update/kind_info", methods=["POST"])
def kind_info_update():
```

```
    KID = request.get_json()['KID']
    KName = request.get_json()['KName']
    Level = int(request.get_json()['Level'])
    Base_salary = int(request.get_json()['Base_salary'])
    sql = '''update kind_info set KName = %s, Level = %s, Base_salary = %s where
KID = %s'''
    dict = [KName,Level,Base_salary,KID]
    data = db.query(sql,dict)
    return jsonify(data)


@app.route("/index/delete/kind_info", methods=["POST"])
def kind_info_delete():
    KID = request.get_json()['KID']
    sql = '''delete from kind_info where KID = %s'''
    data = db.query(sql,KID)
    return jsonify(data)


@app.route("/index/add/bonus_info", methods=["POST"])
def bonus_info_add():
    UID = request.get_json()['UID']
    BDate = datetime.datetime.strptime(    request.get_json()['BDate'],'%Y-%m-%d')
    BType = request.get_json()['BType']
    BDays = int(request.get_json()['BDays'])
    Bonus = int(request.get_json()['Bonus'])
    sql = '''insert into bonus_info values(%s,%s,%s,%s,%s)'''
    dict = [UID,BDate,BType,BDays,Bonus]
    data = db.query(sql,dict)
    return jsonify(data)

@app.route("/index/update/bonus_info", methods=["POST"])
def bonus_info_update():
    UID = request.get_json()['UID']
    BDate = datetime.datetime.strptime(request.get_json()['BDate'],'%Y-%m-%d')
```

```
    BType = request.get_json()['BType']
    BDays = int(request.get_json()['BDays'])
    Bonus = int(request.get_json()['Bonus'])
    sql = '''update bonus_info set BType = %s, Bdays = %s, Bonus = %s where UID
= %s and BDate = %s'''
    dict = [BType,BDays,Bonus,UID,BDate]
    data = db.query(sql,dict)
    return jsonify(data)


@app.route("/index/delete/bonus_info", methods=["POST"])
def bonus_info_delete():
    UID = request.get_json()['UID']
    BDate = datetime.datetime.strptime(request.get_json()['BDate'],'%Y-%m-%d')
    sql = '''delete from bonus_info where UID= %s and BDate = %s'''
    data = db.query(sql,[UID,BDate])
    return jsonify(data)

@app.route("/index/add/department_info", methods=["POST"])
def department_info_add():
    DID = request.get_json()['DID']
    DName = request.get_json()['DName']
    DRef = request.get_json()['DRef']
    sql = '''insert into department_info() values(%s,%s,%s,0)'''
    dict = [DID,DName,DRef]
    data = db.query(sql,dict)
    return jsonify(data)

@app.route("/index/update/department_info", methods=["POST"])
def department_info_update():
    DID = request.get_json()['DID']
    DName = request.get_json()['DName']
    DRef = request.get_json()['DRef']
    sql = '''update department_info set DName = %s, DRef = %s where DID = %s'''
    dict = [DName,DRef,DID]
```

```
    data = db.query(sql,dict)
    return jsonify(data)


@app.route("/index/delete/department_info", methods=["POST"])
def department_info_delete():
    DID = request.get_json()['DID']
    sql = '''delete from department_info where DID = %s'''
    data = db.query(sql,DID)
    return jsonify(data)

@app.route("/index/update/log_info", methods=["POST"])
def log_info_update():
    UID = request.get_json()['UID']
    Password = request.get_json()['Password']
    Authority = request.get_json()['Authority']
    sql = '''update log_info set Password = %s, Authority = %s where UID = %s'''
    dict = [Password,Authority,UID]
    data = db.query(sql,dict)
    return jsonify(data)

@app.route("/index/update/attendance_info", methods=["POST"])
def attendance_info_update():
    UID = session['uid']
    type = request.get_json()['type']
    ADate = request.get_json()['ADate']
    ATime = request.get_json()['ATime']
    STime = request.get_json()['STime']
    sql = '''call attendance_update(%s,%s,%s,%s,%s)'''
    dict = [UID,ADate,ATime,STime,type]
    data = db.query(sql,dict)
    return jsonify(data)


if __name__ == '__main__':
```

```
app.run(host='0.0.0.0')
# app.run()
```

附录 3 管理员前端控制 index.js

```
var vue = new Vue({
    el: "#main",
    vuetify: new Vuetify(),
    data: {
        display: {
            employee_info: true,
            employee_info_add: false,
            bonus_info: false,
            department_info: false,
            kind_info: false,
            log_info: false,
            salary_info: false,
            attendance_info: false,
            salary_report: false
        },

        search: '',

        //employee
        employee_headers: [
            {
                text: 'UID',
                align: 'start',
                sortable: true,
                value: 'UID',
            },
            { text: '姓名', value: 'UName' },
            { text: '年龄', value: 'Age' },
            { text: '性别', value: 'Sex' },
            { text: '部门', value: 'DName' },
            { text: '工种', value: 'KName' },
            { text: '基本薪资', value: 'Base_salary' },
            { text: '职阶', value: 'Level' },
            { text: '工种编号', value: 'KID'    },
```

```
        { text: '部门编号', value: 'DID' }
    ],
    employee_info: [],

    //employee_info_add or update or delete
    valid: false,
    employee_info_add: {
        UID: null,
        UName: null,
        Age: null,
        KID: null,
        DID: null,
        Sex: null,
    },
    Sex_info: [0, 1],


    //department_info
    department_headers: [
        {
            text: 'DID',
            align: 'start',
            sortable: true,
            value: 'DID',
        },
        { text: '部门名', value: 'DName' },
        { text: '部门人数', value: 'DNum' },
        { text: '备注', value: 'DRef' },
    ],
    department_info: [],

    //department_info_add
    department_info_add: {
        DID: null,
        DName: null,
```

```
        DRef: null
    },

    //bonus_info
    bonus_headers: [
        {
            text: 'UID',
            align: 'start',
            sortable: true,
            value: 'UID',
        },
        { text: '日期', value: 'BDate' },
        { text: '津贴类型', value: 'BType' },
        { text: '天数', value: 'BDays' },
        { text: '津贴数目', value: 'Bonus' },
    ],
    bonus_info: [],

    //bonus_info_add
    bonus_info_add: {
        UID: null,
        BDate: null,
        Btype: null,
        BDays: null,
        Bonus: null,
    },

    //salary_info
    salary_headers: [
        {
            text: 'UID',
            align: 'start',
            sortable: true,
            value: 'UID',
        },
```

```
        { text: '月份', value: 'Month' },
        { text: '基础薪资', value: 'Base_salary' },
        { text: '出勤次数', value: 'attendance_times' },
        { text: '出勤率', value: 'attendance_rate' },
        { text: '津贴', value: 'Bonus' },
        { text: '总薪资', value: 'Total_salary' },
    ],


    salary_info: [],

    //attendance_info,
    attendance_headers: [
        {
            text: 'UID',
            align: 'start',
            sortable: true,
            value: 'UID',
        },
        { text: '出勤日期', value: 'ADate' },
        { text: '签到时间', value: 'ATime' },
        { text: '签退时间', value: 'STime' },
    ],
    attendance_info: [],

    //kind_info
    kind_headers: [
        {
            text: 'KID',
            align: 'start',
            sortable: true,
            value: 'KID',
        },
        { text: '工种', value: 'KName' },
        { text: '职级', value: 'Level' },
```

```
        { text: '基础薪资', value: 'Base_salary' },
    ],
    kind_info: [],

    //kind_info_add
    kind_info_add: {
        KID: null,
        KName: null,
        Level: null,
        Base_salary: null
    },


    //log_info
    log_headers: [
        {
            text: 'UID',
            align: 'start',
            sortable: true,
            value: 'UID',
        },
        { text: 'Password', value: 'Password' },
        { text: 'Authority', value: 'Authority' },
    ],
    log_info: [],

    //kind_info_add
    log_info_add: {
        UID: null,
        Password: null,
        Authoruty: null,
    },
    log_authority_items:[0,1],
```

```
        //salary_report
        salary_report_headers: [
            {
                text: 'UID',
                align: 'start',
                sortable: true,
                value: 'UID',
            },
            { text: '姓名', value: 'UName' },
            { text: '部门', value: 'DName' },
            { text: '年度薪资', value: 'Salary' },
            { text: '年终奖', value: 'Award' },
            { text: '合计', value: 'Total' }
        ],
        salary_report: [],


    },

    methods: {
        show: function (target) {
            for (each in this.display) {
                if (target === each)
                    this.display[each] = true;
                else
                    this.display[each] = false;
            }
        },

        logout:function(){
            axios.get('/logout')
            .then(function (res) {
                window.location.href="login"
            }).catch(function (error) {
```

```
            alert(error);
        });
    },

    info_refresh: function (tar) {
        var _this = this;
        axios.get('/index/get/' + tar)
            .then(function (res) {
                _this[tar] = [];
                for (var i of res.data) {
                    _this[tar].push(i);
                }
            }).catch(function (error) {
                alert(error);
            });
    },

    get_department: function () {
        this.show('department_info');
        if (this.department_info.length === 0)
            this.info_refresh('department_info');
    },

    get_kind: function () {
        this.show('kind_info');
        if (this.kind_info.length === 0)
            this.info_refresh('kind_info');
    },

    get_bonus: function () {
        this.show('bonus_info');
        if (this.bonus_info.length === 0)
            this.info_refresh('bonus_info');
    },
```

```
get_salary: function () {
        this.show('salary_info');
        if (this.salary_info.length === 0)
                this.info_refresh('salary_info');
},

get_attendance: function () {
        this.show('attendance_info');
        if (this.attendance_info.length === 0)
                this.info_refresh('attendance_info');
},

get_log: function () {
        this.show('log_info');
        if (this.log_info.length === 0)
                this.info_refresh('log_info');
},

get_salary_report: function () {
        this.show('salary_report');
        if (this.salary_report.length === 0)
                this.info_refresh('salary_report');
},

employee_info_reset:function(){
        for (each in this.employee_info_add) {
                this.employee_info_add[each] = null;
        }
},

employee_info_submit: function () {
        var _this = this;
        axios.post('/index/add/employee_info', _this.employee_info_add)
                .then(function (res) {
                        for (each in _this.employee_info_add) {
```

```
                _this.employee_info_add[each] = null;
            }
            console.log(res);
        }).catch(function (error) {
            alert(error);
        });
},

employee_info_update: function () {
    var _this = this;
    axios.post('/index/update/employee_info', _this.employee_info_add)
        .then(function (res) {
            for (each in _this.employee_info_add) {
                _this.employee_info_add[each] = null;
            }
            console.log(res);
        }).catch(function (error) {
            alert(error);
        });
},

employee_info_delete: function () {
    var _this = this;
    axios.post('/index/delete/employee_info', _this.employee_info_add)
        .then(function (res) {
            for (each in _this.employee_info_add) {
                _this.employee_info_add[each] = null;
            }
            console.log(res);
        }).catch(function (error) {
            alert(error);
        });
},

kind_info_submit: function () {
```

```
        var _this = this;
        axios.post('/index/add/kind_info', _this.kind_info_add)
            .then(function (res) {
                for (each in _this.kind_info_add) {
                    _this.kind_info_add[each] = null;
                }
                console.log(res);
            }).catch(function (error) {
                alert(error);
            });
    },

    kind_info_update: function () {
        var _this = this;
        axios.post('/index/update/kind_info', _this.kind_info_add)
            .then(function (res) {
                for (each in _this.kind_info_add) {
                    _this.kind_info_add[each] = null;
                }
                console.log(res);
            }).catch(function (error) {
                alert(error);
            });
    },

    kind_info_delete: function () {
        var _this = this;
        axios.post('/index/delete/kind_info', _this.kind_info_add)
            .then(function (res) {
                for (each in _this.kind_info_add) {
                    _this.kind_info_add[each] = null;
                }
                console.log(res);
            }).catch(function (error) {
                alert(error);
```

```
            });
        },

        bonus_info_submit: function () {
            var _this = this;
            axios.post('/index/add/bonus_info', _this.bonus_info_add)
                .then(function (res) {
                    for (each in _this.bonus_info_add) {
                        _this.bonus_info_add[each] = null;
                    }
                    console.log(res);
                }).catch(function (error) {
                    alert(error);
                });
        },

        bonus_info_update: function () {
            var _this = this;
            axios.post('/index/update/bonus_info', _this.bonus_info_add)
                .then(function (res) {
                    for (each in _this.bonus_info_add) {
                        _this.bonus_info_add[each] = null;
                    }
                    console.log(res);
                }).catch(function (error) {
                    alert(error);
                });
        },

        bonus_info_delete: function () {
            var _this = this;
            axios.post('/index/delete/bonus_info', _this.bonus_info_add)
                .then(function (res) {
                    for (each in _this.bonus_info_add) {
                        _this.bonus_info_add[each] = null;
```

```
                }
                console.log(res);
            }).catch(function (error) {
                alert(error);
            });
        },

        department_info_submit: function () {
            var _this = this;
            axios.post('/index/add/department_info', _this.department_info_add)
                .then(function (res) {
                    for (each in _this.department_info_add) {
                        _this.department_info_add[each] = null;
                    }
                    console.log(res);
                }).catch(function (error) {
                    alert(error);
                });
        },

        department_info_update: function () {
            var _this = this;
            axios.post('/index/update/department_info',
_this.department_info_add)
                .then(function (res) {
                    for (each in _this.department_info_add) {
                        _this.department_info_add[each] = null;
                    }
                    console.log(res);
                }).catch(function (error) {
                    alert(error);
                });
        },

        department_info_delete: function () {
```

```
        var _this = this;
        axios.post('/index/delete/department_info', _this.department_info_add)
            .then(function (res) {
                for (each in _this.department_info_add) {
                    _this.department_info_add[each] = null;
                }
                console.log(res);
            }).catch(function (error) {
                alert(error);
            });
    },

    log_info_update: function () {
        var _this = this;
        axios.post('/index/update/log_info', _this.log_info_add)
            .then(function (res) {
                for (each in _this.log_info_add) {
                    _this.log_info_add[each] = null;
                }
                console.log(res);
            }).catch(function (error) {
                alert(error);
            });
    },

    analysis:function(){
        var option = {
            tooltip: {
                trigger: 'item',
                formatter: '{a} <br/>{b}: {c} ({d}%)'
            },
            title: {
                text: '部门薪资占比分析',
                left: 'center',
                top: 20,
```

```
            textStyle: {
                color: '#293c55'
            }
        },
        series: [
            {
                name: '部门总薪资占比',
                type: 'pie',
                radius: ['50%', '70%'],
                avoidLabelOverlap: false,
                label: {
                    show: false,
                    position: 'center'
                },
                emphasis: {
                    label: {
                        show: true,
                        fontSize: '30',
                        fontWeight: 'bold'
                    }
                },
                labelLine: {
                    show: false
                },
                data: [
                ]
            }
        ]
    };

    this.get_salary_report();

    var total = [];

    for (each in this.salary_report){
```

```
                if (total[this.salary_report[each].DName]==undefined)
                    total[this.salary_report[each].DName] =
this.salary_report[each].Total;
                else
                    total[this.salary_report[each].DName] +=
this.salary_report[each].Total;
            }

            // console.log(total);

            for(each in total){
                option.series[0].data.push({ value: total[each], name: each });
            }

            option.series[0].data.sort(function (a, b) { return a.value - b.value; });
            var myChart = echarts.init(document.getElementById("mycharts"));
            myChart.setOption(option, true);
        }
    },
    mounted: function () {
        var _this = this;
        axios.get('/index/get/employee_info')
            .then(function (res) {
                for (var i of res.data) {
                    _this.employee_info.push(i);
                }
            }).catch(function (error) {
                alert(error);
            });
    }
});
```