华中科技大学

数据库系统原理实践报告

项目名称：　　工资管理系统

姓　　名：　　李响

专　　业：　　计算机科学与技术

班　　级：　　CS1802

学　　号：　　U201814531

指导教师：　　赵小松

分数	
教师签名	

2021 年 7 月 2 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

数据库课程实验的主要目的是帮助学生掌握主流数据库软件的基本使用方法，并增强对 SQL 语言的掌握。

通过本课程实验，应该达到以下目的：熟练掌握 SQL 的数据定义、数据操纵和数据控制语言的运用；熟悉数据库应用系统的设计方法和开发过程；通过上机实践，熟悉一种大型数据库管理系统，了解 DBMS 的体系结构。

本实验的具体的任务要求如下：

（1）学习并掌握一款主流数据库管理系统软件，如 SQL Server、MySQL。练习并掌握该 DBMS 的基本备份方式，包括数据和日志文件的脱机备份，以及联机的系统备份功能。练习在新增的数据库上增加用户并配置权限的操作，并通过新创建的用户登录数据库，并且执行未经授权的 SQL 语句，以验证自己的权限配置的正确性；

（2）掌握基本的 SQL 语句编写方法，并在 EduCoder 平台通过各项练习与测试，包括数据库的创建，表的创建，数据的插入、删除、更新操作，数据的查询操作，触发器的创建，函数的创建与使用等；

（3）了解嵌入式 SQL 语言的使用方法，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，拟定一个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 实验一 学习软件功能

2.1 任务要求

完成下列 1~2 题，并在实践报告中叙述过程，可适当辅以插图：

（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：数据和日志文件的脱机备份、系统的备份功能；

（2）练习在新增的数据库上增加用户并配置权限的操作，通过用创建的用户登录数据库并且执行未经授权的 SQL 语句验证自己的权限配置是否成功。

2.2 完成过程

本次实验的需要掌握一款主流的关系数据库软件的基本使用方法，通过比对后选择了 SQL Server 作为学习和掌握的软件，实验环境如下：

操作系统：Windows 10 家庭中文版

数据库软件：Microsoft SQL Server 2019

数据库管理软件：Microsoft SQL Server management Studio 18

2.2.1 练习数据备份

根据数据库是否脱机，可以将数据库系统的备份方式分为冷备份以及热备份两种，其中，冷备份是指数据库在脱机状态下进行的备份，在进行冷备份时，需要停止数据库的所有服务并将数据库进行脱机处理；热备份是指在数据库运行的情况下进行数据备份，进行热备份时可能会导致 I/O 速度下降，但是仍可以进行正常的服务，具体备份过程如下：

1. 数据库冷备份（脱机备份）

（1）选中需要进行备份的数据库，关闭所有正在查询的服务，选中需要进行脱机处理的数据库 covid19mon，如下图 2.1 所示，选中“**任务**”的子项“**脱机**”将数据库进行脱机处理；

图 2.1 数据库脱机处理

（2）进入数据库文件位置（软件安装时选择的位置，也可以如图 2.2 所示，通过数据库的“**属性**”的子项“**文件**”查看数据库文件位置），复制数据库文件（.mdf 和.ldf 文件，分别为数据库文件和日志文件），保存到目标位置。

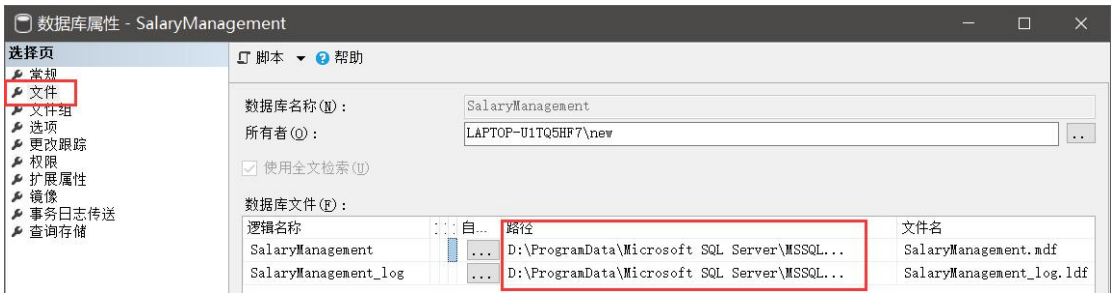


图 2.2 数据库文件位置

（3）使用备份文件恢复数据库。由于冷备份是将整个数据库文件进行复制保存，当原系统中存在该数据库时，直接使用冷备份数据恢复数据库会显示错误，需要将原数据库删除后才可以使用数据库文件恢复。

冷备份数据库文件恢复本质上是将**数据库文件添加到数据库管理系统**中，不是传统意义上的数据恢复，如图 2.3 所示，使用数据库的“附加”功能添加数据库文件，即可实现冷备份数据库文件的恢复。

需要注意的是，如果出现附加失败的情况，一般是由于当前系统中存在同名的数据库，或者是**文件操作权限有问题，需要设置文件为所有人可控制**。

图 2.3 冷备份数据库文件恢复

2. 数据库热备份

Sql Server 的联机备份分为全量备份（Full Backup）、差异备份（Differential Backup）以及事务日志备份（Transaction Log Backup）。

其中，全量备份是数据库的完整拷贝，耗时长且比较占用磁盘空间；差异备份只备份当前数据库与上一次全量备份的差异部分；事务日志备份则不备份数据库的数据，仅对日志文件的内容进行备份。

通常数据库会在较长时间定时做全量备份，以分钟为间隔进行事务日志备份，日常定期做差异备份，以确保数据库的可恢复性，备份方式如下：

（1）以全量备份为例，如图 2.4 所示，选中需要进行备份的数据库，选择“任务”的子项“备份”对数据进行备份。备份类型选择完整备份，点击确认即可完成全量备份。

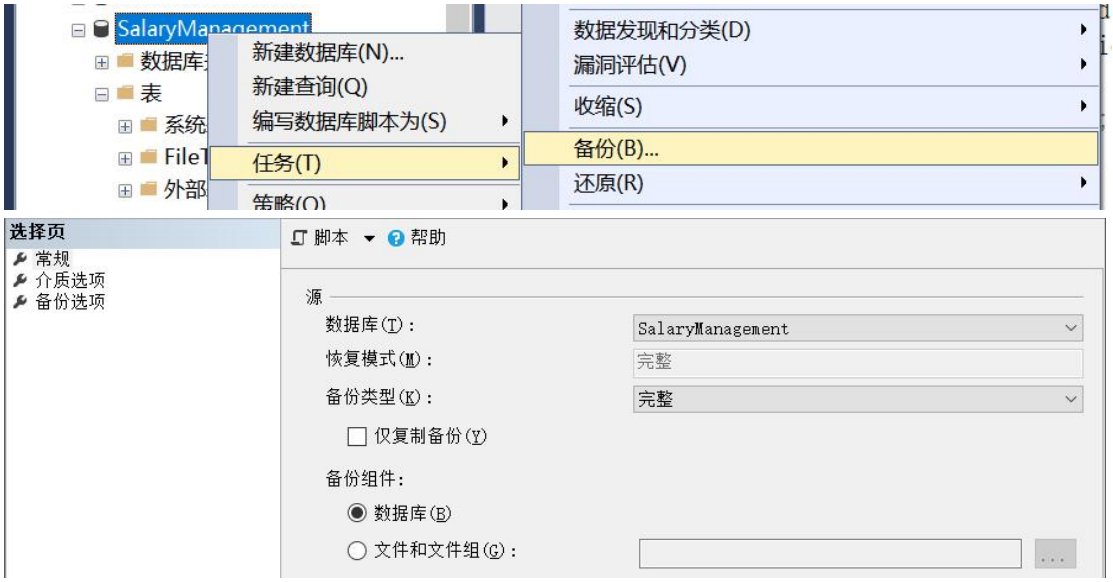


图 2.4 数据库全量备份

（2）数据库恢复，如图 2.5 选中需要进行恢复的数据库，选择“任务”的子项“还原”对数据进行还原。还原类型可以选择数据库或文件等，需要根据需求确定。

图 2.5 数据库还原

2.2.2 练习创建用户与权限配置

使用超级用户（或其他具有创建用户权限的用户）或 windows 身份用户进行登录连接，创建数据库系统的**登录名**以及需要访问的数据库的**用户名**，并将设计的数据库操作**权限授予创建的用户**，使用新创建的登录名登录并进行操作，具体的创建过程如下：

（1）创建登录名与用户名

首先，使用 T-SQL 语句创建登录名“lx”；然后，在数据库“covid19mon”上创建同名用户“lx”，将该用户的登录名设置为刚创建的登录名“lx”；最后，将 person 表的查询权限赋予该用户。

```
use covid19mon;
create login lx with password = '123456';
create user lx for login lx;
grant select on person to lx;
```

（2）登录并验证权限

使用新创建的用户名登录，并进行权限的验证，如图 2.6 所示，由于只赋予了 person 表的 select 权限，因此在数据库“covid19mon”中只能看见 person 表，其余表皆不可见，也无法查询；当进行插入等操作时，会显示拒绝操作。

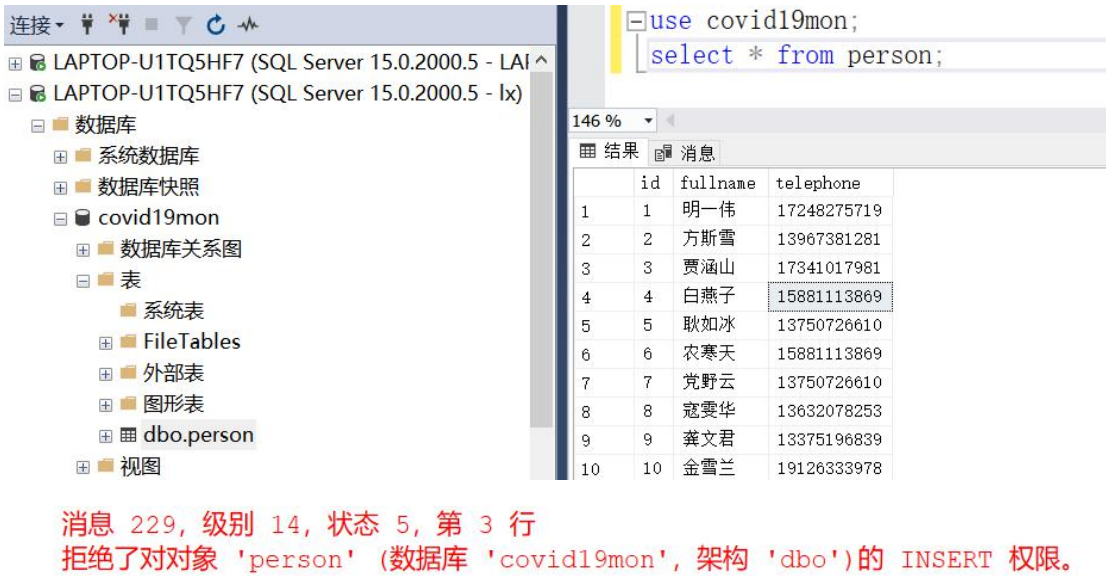


图 2.6 用户登录与权限验证

2.3 任务总结

考虑到 Sql Server 所用的 T-SQL 语言语法较 MySQL 更为严格，比较适合于初学者使用，因此本此选择了 Sql Server 作为实验软件平台。

本次任务偏向于软件的熟悉与基本使用方法的掌握，在实验过程中，我确实遇到了不少问题，大部分问题通过熟悉软件和 Sql 语言的语法就可以快速解决，单仍有少数问题迫使我查阅相关的资料。

对于数据的备份，我查阅了相关的资料，深入了解了 Sql Server 三种主要备份方式的运行原理，掌握了冷备份与热备份的差异和适用场景。同时，我尝试创建用户并赋予权限，实现了从理论到实践的跃进。

通过本次实验，我掌握了 Sql Server 软件的基本使用方法，也熟悉了使用 Sql 语言编写查询文件进行系统操作。

3 实验二 SQL 练习

3.1 任务要求

使用基本的 SQL 语句创建一个新冠肺炎信息管理的数据库，完成相关的数据操作，并在 EduCoder 平台测试，数据操作包括：数据库的创建，表的创建，数据的插入、删除、更新操作，数据的查询操作，触发器的创建，函数的创建与使用等，该数据库系统的具体的要求如下：

（1）假设在某个区域内的所有地点都存储在**地点表**（location）中；

（2）该地区中的所有人员信息存储在**人员表**（person）中；

（3）根据人员行程建立了疫情期间某个地区的**人员行程表**（itinerary）；

（4）部分人员进行核酸检测，结果保存在**诊断表**（diagnose_record）中；

（5）根据诊断表中的检测结果，对新冠确诊和无症状感染者的密切接触者进行隔离，密切接触者的信息存储到**密切接触表**（close_contact）中，隔离信息存储到**隔离表**（isolation_record）中；

（6）隔离地点共设置了 4 个，并且每个隔离地点都有容量限制，隔离点的信息存储在**隔离地点表**（isolation_location）中。

3.2 完成过程

3.2.1 数据库与基本表的创建

创建数据库与基本表，并建立相对应的主键约束与外键约束，使用 constraint 子句对约束进行命名，具体的创建过程如下：

1. 创建数据库

使用 Sql 语句，创建用于管理新冠疫情防控的数据库 covid19mon，并将当前可用数据库设置为 covid19mon 数据库。

```
create database covid19mon;
use covid19mon;
```

2. 创建基本表

按照要求创建新冠疫情防控信息的 7 个表，并为表创建主码与外码，同时，定义主外码约束，并按照要求为约束命名；注册并登录 EduCoder 平台对建表语句进行测试，建表的详细要求参见任务书，创建代码如下：

```
-- 表 1 人员表(person)
 create table person(
       id int, fullname char(20) not null,
       telephone char(11) not null,
       constraint pk_person primary key(id)
 );
```

```
-- 表 2 地点表(location)
create table location(
        id int, location_name char(20) not null,
        constraint pk_location primary key(id)
);

-- 表 3 行程表（itinerary）
create table itinerary(
      id int, p_id int, loc_id int,
      s_time datetime, e_time datetime,
      constraint pk_itinerary primary key(id),
      constraint fk_itinerary_pid foreign key(p_id) references person(id),
      constraint fk_itinerary_lid foreign key(loc_id) references location(id)
);

-- 表 4 诊断表（diagnose_record）
create table diagnose_record(
      id int, p_id int, diagnose_date datetime, result int,
      constraint pk_diagnose_record primary key(id),
      constraint fk_diagnose_pid foreign key(p_id) references person(id)
);

-- 表 5 密切接触者表（close_contact）
create table close_contact(
      id int, p_id int, contact_date datetime, loc_id int, case_p_id int,
      constraint pk_close_contact primary key(id),
      constraint fk_contact_pid foreign key(p_id) references person(id),
      constraint fk_contact_lid foreign key(loc_id) references location(id),
      constraint fk_contact_caseid foreign key(case_p_id) references person(id)
);

-- 表 6 隔离地点表（isolation_location）
create table isolation_location(
      id int, location_name char(20), capacity int,
      constraint pk_isolation_loc primary key(id),
);

-- 表 7 隔离表（isolation_record）
create table isolation_record(
      id int, p_id int, s_date datetime, e_date datetime, isol_loc_id int, state int,
      constraint pk_isolation primary key(id),
      constraint fk_isolation_pid foreign key(p_id) references person(id),
      constraint fk_isolation_lid foreign key(isol_loc_id) references isolation_location(id)
);
```

3. 观察性实验

验证在建立外码时是否一定要参考被参照关系的主码，通过分析可以得到，外码一定是被参照表的主码，所以当外键不是主键时会报错，如图 3.1 所示，当被参照列为非候选码时，无法创建约束。

```
在被引用表 'diagnose_record' 中没有与外键 'FK__closed__case_p_i__4CA06362' 中的引用列列表匹配的主键或候选键。
消息 1750，级别 16，状态 1，第 1 行
无法创建约束或索引。请参阅前面的错误。
```

图 3.1 外键参考关系验证

4. 数据准备

从群聊中下载数据插入 sql 文件“SQL Server 实训数据批量插入.sql”，执行当前插入数据文件即可实现数据的插入操作。

3.2.2 数据更新

1. 分别用一条 sql 语句完成对人员表基本的增、删、改的操作

（1）向人员表 person 表中插入如下数据(1, 张小敏, 13907110001)，(2, 李大锤, 18907110002)，(3, 孙二娘, 13307100003)，共 3 条数据：

```
insert into person(id, fullname, telephone) values(1, N'张小敏', N'13907110001');
insert into person(id, fullname, telephone) values(2, N'李大锤', N'18907110002');
insert into person(id, fullname, telephone) values(3, N'孙二娘', N'13307100003');
```

（2）从人员表 person 中删除(2, 李大锤, 18907110002)：

```
delete from person
   where id = 2;
```

（3）请将人员表 person 中 1 号人员的电话更改成 13607176668：

```
update person
set telephone = N'13607176668'
   where id = 1;
```

2. 将行程表中所有到达地点 2 的记录插入到新表 location_record_2 中

首先，创建表 location_record_2，其属性列和行程表完全相同；然后，从行程表中查询出到达地点 2 的记录插入到 location_record_2 中，代码如下：

```
create table location_record_2(
   id int primary key,
   p_id int,
   loc_id int,
   s_time datetime,
   e_time datetime,
   foreign key(p_id) references person(id),
   foreign key(loc_id) references location(id)
);
insert into location_record_2 select * from itinerary where loc_id = 2;
select * from location_record_2;
```

执行以上代码后，执行结果如图 3.2 所示，可以发现到达地点为 2 的记录已经顺利插入新创建的 location_record_2 表中。

结果 消息

	id	p_id	loc_id	s_time	e_time
1	25	4	2	2021-02-03 03:05:09.000	2021-02-03 03:57:09.000
2	42	5	2	2021-02-03 04:48:29.000	2021-02-03 06:22:29.000
3	71	46	2	2021-02-02 15:46:04.000	2021-02-02 17:05:04.000
4	80	48	2	2021-02-02 06:54:09.000	2021-02-02 06:55:09.000
5	89	16	2	2021-02-03 01:28:09.000	2021-02-03 02:21:09.000
6	92	41	2	2021-02-02 04:45:47.000	2021-02-02 06:24:47.000
7	111	15	2	2021-02-02 19:07:07.000	2021-02-02 19:31:07.000
8	117	27	2	2021-02-03 00:20:07.000	2021-02-03 02:14:07.000
9	122	14	2	2021-02-02 18:02:58.000	2021-02-02 19:32:58.000
10	134	33	2	2021-02-02 17:29:33.000	2021-02-02 17:41:33.000

图 3.2 基本表 location_record_2 内容截图

3. 数据导入导出

通过查阅 DBMS 资料学习数据导入导出功能，将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，具体过程如下：

（1）数据导出

右键点击数据库 covid19mon，选择“任务”，再选择“导出数据”，进入数据导出界面；然后，点击 next，进入选择数据源界面，如图 3.3 所示，数据源选择“SQL Server Native Client 11.0”，即直接从数据库中导出数据，数据库选择需要导出的数据库“covid19mon”。

图 3.3 选择导出数据源界面

继续点击 next，如图 3.4 所示，选择数据输出的目标类型、文件类型以及文件路径，本次数据导出选择“Flat File Destination”，即选择输出到文件，文件名为 person.txt；输出的源表选择 person 表，点击完成即可实现数据的输出。

图 3.4 选择输出文件界面

（2）数据导入

将之前导出的数据导入数据库中，如图 3.5 所示，首先选择数据源，导出的文件“person.txt”，之后依照流程，选择数据库，导入的表，并导入即可。

图 3.5 选择导入数据源界面

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，具体步骤如下

所示，首先，创建基本表 test，不设置主键；然后，插入重复元组，观察现象；最后在图形化界面修改或删除相同元组中的一个，观察现象。

当插入相同元组时，数据库不会报错，会显示正常插入；但是，当删除其中一个元组时，如图 3.6 所示，系统会显示删除的行值改变了多个行，无法删除。

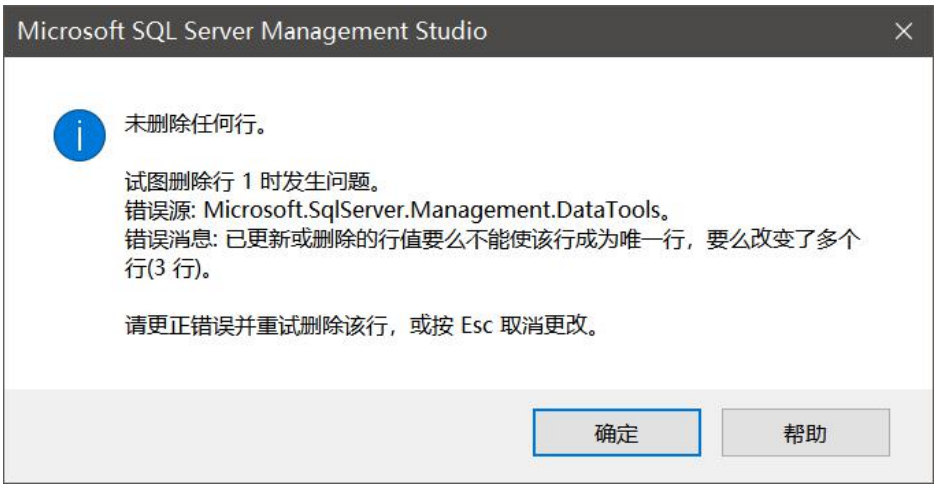


图 3.6 观察性实验删除失败界面

5. 触发器实验

编写一个触发器，用于实现完整性控制规则，当隔离表中的某位隔离人员在诊断表中的诊断结果为“1：新冠确诊”，将隔离状态从“1”改成“3”，代码实现如下：

```
create trigger trans_patients on diagnose_record
    after insert, update
as update isolation_record
    set state = 3
    where p_id in (select p_id from inserted where result = 1);
```

当某位隔离人员的诊断状态发生变化时，即当有新的记录插入或对原有数据进行修改时都需要触发该触发器，因此触发器应建立在诊断表（diagnose_record）上，并在插入（insert）或更新（update）时触发。当诊断表新更新的数据属性中，诊断结果为 1 时需要将隔离状态中的状态修改为 3。

3.2.3 数据查询

请分别用一条 SQL 语句完成下列各个小题的需求：

1. 查询累计人流量大于 30 的地点名称和累计人流量，请用 visitors 作累积人流量的标题名称。查询结果按照人流量从高到低排序，人流量相同时依地点名称的字典顺序排序。

```
select location_name, count(itinerary.id) as visitors
    from location, itinerary
    where location.id = itinerary.loc_id
```

```
    group by itinerary.loc_id, location_name having count(itinerary.id) > 30
    order by visitors desc, location_name asc;
```

解题思路：地点名称存储在地点信息表 location 中，同一地点的累计人流量可以通过行程表 itinerary 进行分组并计算得出，因此采用两表连接的方式，并使用 group 子句将行程按照地点分组，并使用聚集函数 count 统计每一组的累计人数，即可得出对应的结果，最后使用 order 子句进行排序即可，代码如下，测试结果如图 3.7 所示。

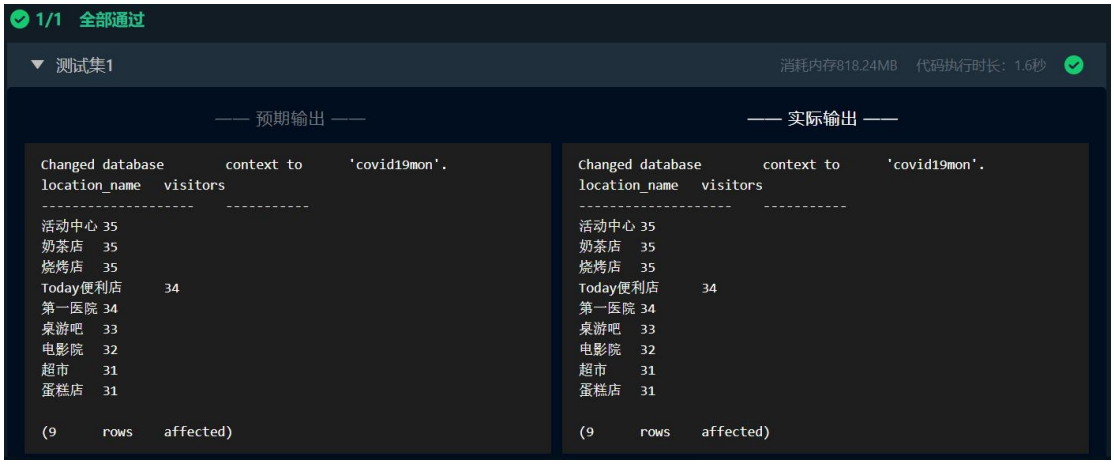


图 3.7 查询累计人流量测试结果图

2. 查询每个隔离地及该地正在隔离的人数，以 number 为隔离人数标题，查询结果依隔离人数由降序排序，隔离人数相同时依隔离点名称字典序排序。

```
select isolation_location.location_name, count(distinct isolation_record.p_id) as number
    from isolation_location, isolation_record
    where isolation_location.id = isolation_record.isol_loc_id
        and isolation_record.state = 1
    group by location_name, isol_loc_id
    order by number desc, location_name asc
```

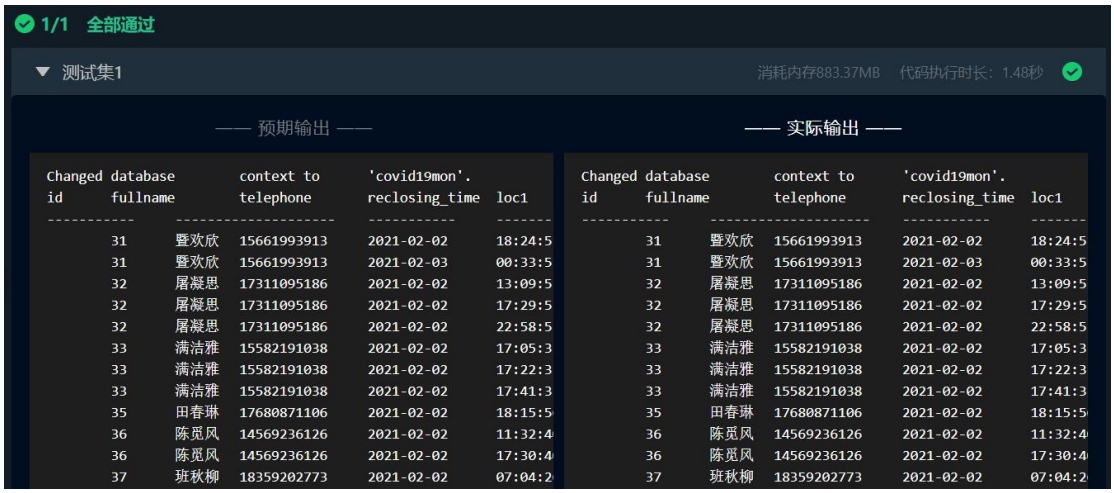
解题思路：思路与上一道题类似，使用两表连接的方式将数据进行连接合并，使用 group 子句进行分组，聚集函数 count 进行统计，最后使用 order 进行排序，测试结果如图 3.8 所示。

图 3.8 查询隔离人数结果图

3. 查询行程表中同一个人接续行程的时间和地点信息，所谓接续行程指同一个人第一段行程的结束时间与第二段行程的开始时间重合。查询人员编号大于30的接续行程，输出信息包括人员编号、姓名、重合时间、起始地点id、起始地点、结束地点id以及结束地点，查询结果依人员编号排序。

```
select person.id, fullname, telephone, first.e_time as reclosing_time,
   l1.id as loc1, l1.location_name as address1, l2.id as loc2, l2.location_name as address2
      from itinerary as first, itinerary as second, person, location as l1, location as l2
      where person.id = first.p_id and first.p_id > 30 and first.p_id = second.p_id
            and first.loc_id = l1.id and second.loc_id = l2.id
            and first.e_time = second.s_time
      order by person.id asc, first.e_time asc
```

解题思路：本题的输出信息较为复杂，需要进行多个表的连接，本题最重要的信息是接续行程的定义（第一段结束时间与第二段开始时间重合），其他信息使用where进行列举即可，并使用order子句进行排序，执行结果如图3.9所示。



Changed id	database fullname	context to telephone	'covid19mon'. reclosing_time	loc1
31	暨欢欣	15661993913	2021-02-02	18:24:5
31	暨欢欣	15661993913	2021-02-03	00:33:5
32	屠凝思	17311095186	2021-02-02	13:09:5
32	屠凝思	17311095186	2021-02-02	17:29:5
32	屠凝思	17311095186	2021-02-02	22:58:5
33	满洁雅	15582191038	2021-02-02	17:05:3
33	满洁雅	15582191038	2021-02-02	17:22:3
33	满洁雅	15582191038	2021-02-02	17:41:3
35	田春琳	17680871106	2021-02-02	18:15:5
36	陈觅风	14569236126	2021-02-02	11:32:4
36	陈觅风	14569236126	2021-02-02	17:30:4
37	班秋柳	18359202773	2021-02-02	07:04:2

Changed id	database fullname	context to telephone	'covid19mon'. reclosing_time	loc1
31	暨欢欣	15661993913	2021-02-02	18:24:5
31	暨欢欣	15661993913	2021-02-03	00:33:5
32	屠凝思	17311095186	2021-02-02	13:09:5
32	屠凝思	17311095186	2021-02-02	17:29:5
32	屠凝思	17311095186	2021-02-02	22:58:5
33	满洁雅	15582191038	2021-02-02	17:05:3
33	满洁雅	15582191038	2021-02-02	17:22:3
33	满洁雅	15582191038	2021-02-02	17:41:3
35	田春琳	17680871106	2021-02-02	18:15:5
36	陈觅风	14569236126	2021-02-02	11:32:4
36	陈觅风	14569236126	2021-02-02	17:30:4
37	班秋柳	18359202773	2021-02-02	07:04:2

图 3.9 查询接续行程结果图

4. 查询充珉瑶和贾涵山的行程情况，查询结果包括：姓名、电话、地名、到达时间以及离开时间，查询结果依人员编号降序排序，同一人员行程依行程开始时间顺序排列。

```
select fullname, telephone, location_name,
            convert(char(19),s_time,20) as s_time,
            convert(char(19),e_time,20) as e_time
      from person
         left outer join itinerary on person.id = itinerary.p_id
         left outer join location on itinerary.loc_id = location.id
      where person.fullname in (N'充珉瑶',N'贾涵山')
      order by person.id desc, s_time
```

解题思路：本题存在一个陷阱，如果查询人员不存在行程记录时，需要将行程相关的属性置为NULL，也就是说需要进行外连接，若将person表作为左表则需要进行左外连接（left outer join），然后使用in进行集合运算并使用order子

句进行排序，查询结果如图 3.10 所示。

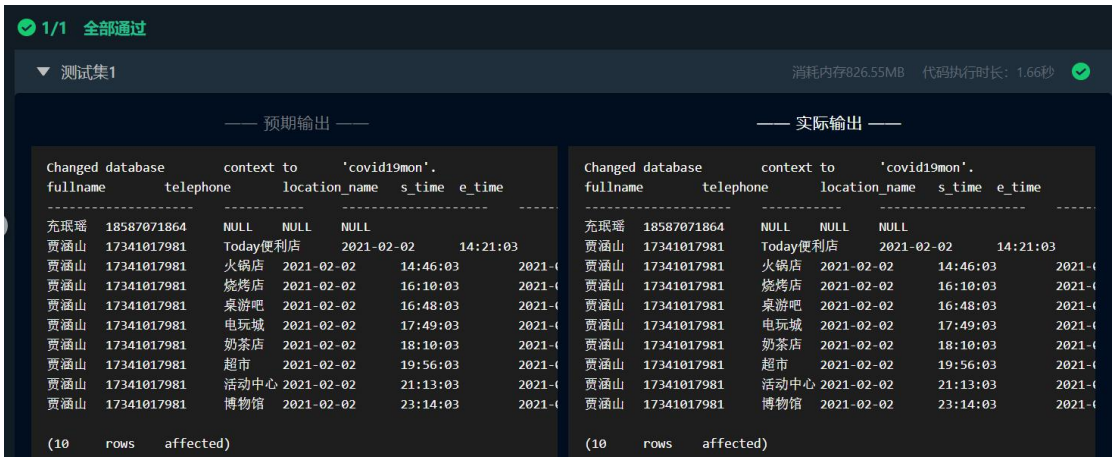


图 3.10 查询充珉瑶和贾涵山的行程情况结果图

5. 查询地名中带有‘店’字的地点编号和名称，查询结果按地点编号排序。

```
select id, location_name
    from location
    where location_name like N'%店%'
    order by id
```

解题思路：本题的考察要点是 LIKE 语句的使用，使用“%”进行通配，查询其中包含“店”字的记录，查询结果如图 3.11 所示。

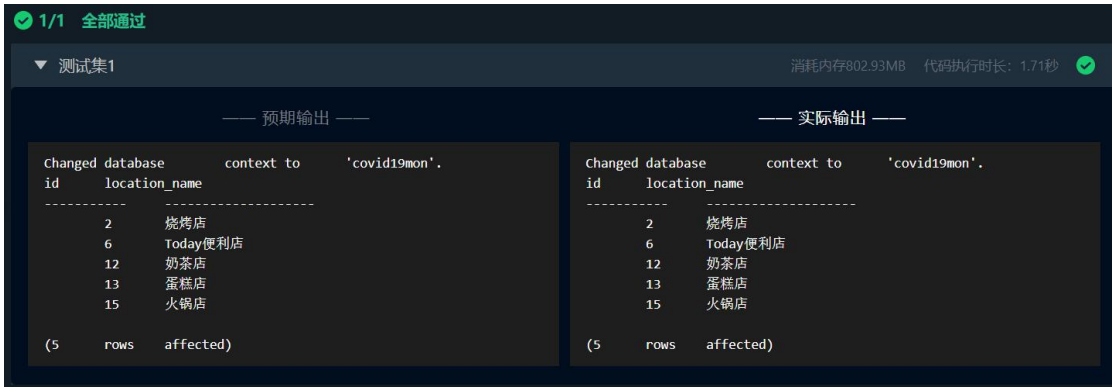


图 3.11 查询地名中带有店字记录结果图

6. 新发现一位确诊者，已知他在 2021.2.2 当天 20:05:40 到 21:25:40 之间在“活动中心”逗留，凡在此时间段在此地点逗留过的，视为接触者，请查询接触者的姓名和电话，查询结果按姓名排序。

```
select distinct fullname, telephone
    from person, itinerary, location
    where itinerary.p_id = person.id and itinerary.loc_id = location.id
    and location_name = N'活动中心'
    and s_time < '2021-02-02 21:25:40.000'
    and e_time > '2021-02-02 20:05:40.000'
    order by fullname
```

解题思路：如果行程记录中某条记录的停留时间与确诊者的逗留时间存在重

合就说明是接触者，即当开始时间小于 21:25:40 且结束时间大于 20:05:40，具体的查询结果如图 3.12 所示。

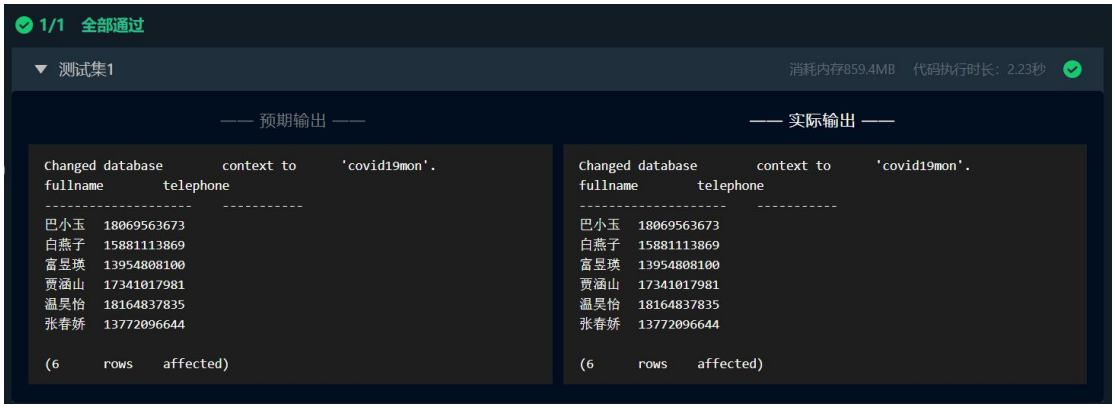


图 3.12 查询接触者结果图

7. 查询仍在使用的隔离点名称，注意，隔离记录里如果只有隔离结束或确诊转院的记录，表明该隔离点已暂时停用，只要还有一个人在此处隔离，表明该隔离点仍在使用，查询结果按隔离点编号排序。

```
select location_name
    from isolation_location
    where id in (select id from isolation_location
        where exists(select * from isolation_record
            where isolation_record.isol_loc_id = isolation_location.id
              and isolation_record.state = 1 )
          )
    order by isolation_location.id
```

解题思路：第一层为隔离地点信息表，由于输出的数据为隔离地点而不是隔离地点编号，因此需要多加这一层；后两层使用相关嵌套查询的方式进行查询，隔离记录中某个地点的存在隔离记录，则表示当前隔离点仍在使用，具体的查询结果如图 3.13 所示。

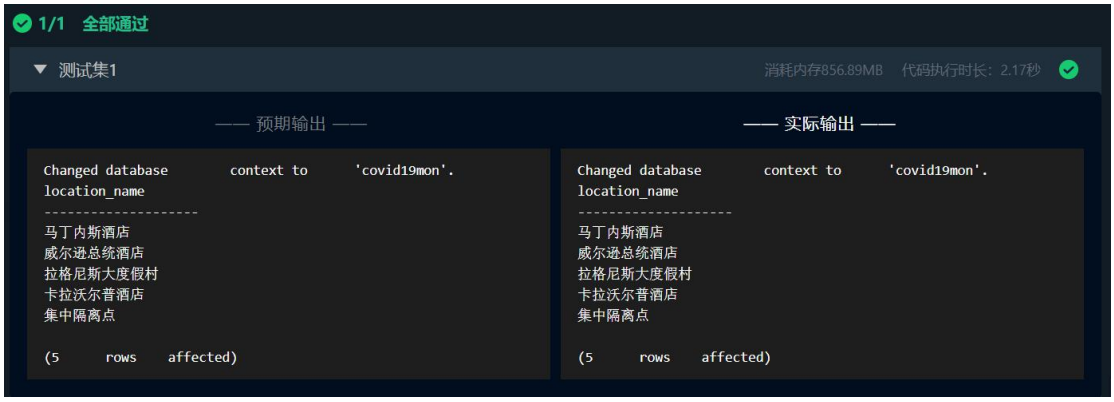


图 3.13 查询仍在使用的隔离点结果图

8. 用一条带 exists 关键字的 SQL 语句，查询前 30 位有出行记录的人员姓名和电话，查询结果按照人员编号排序。

```
select top 30 fullname, telephone from person
    where id in(select id from person
        where exists(select * from itinerary
        where itinerary.p_id = person.id)
    )
    order by id asc
```

解题思路：使用 Sql Server 提供的 top 函数可以快速解决该问题，使用相关嵌套查询以及 exists 子句，查询人员是否在行程表中有行程记录，具体的查询结果如图 3.14 所示。

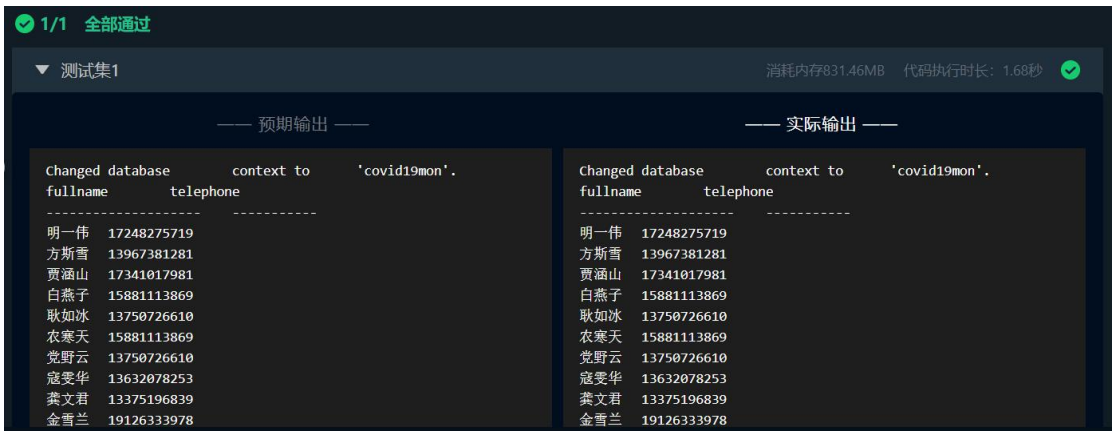


图 3.14 查询前 30 位有出行记录的人员结果图

9. 使用带 not exists 子查询的 SQL 语句，查询人员表中**没有**去过地点“Today 便利店”的人数，请给统计出的人数命名为 number。

```
select count(*) as number from person
    where not exists(select * from itinerary, location
        where person.id = itinerary.p_id
          and itinerary.loc_id = location.id
          and location_name = N'Today 便利店'
        )
```

解题思路：本题要查询人员表中未去过一个地点的人员树，需要使用相关嵌套查询以及 not exists 子句进行查询，最后使用聚集函数 count 进行人数统计即可，具体的查询结果如图 3.15 所示。

图 3.15 查询没有去过地点 Today 便利店的人数结果图

10. 使用一条 SQL 语句查询去过所有地点的人员姓名，查询结果依人员姓名的字典顺序排序。

```
select fullname from person
    where not exists(select * from location
        where not exists(select * from itinerary
            where location.id = loc_id and person.id = p_id)
    )order by fullname asc

select fullname from person
    where not exists(select * from location
        where location.id not in(select loc_id from itinerary
            where person.id = p_id)
    )order by fullname asc
```

解题思路：本题需要使用 exists 存在谓词实现全称谓词的逻辑，将去过所有地点的人员进行翻译，即是对于某个人员不存在地点表中地点没有在行程表中出现，该逻辑可以使用 not exists 以及 not in 两种方式实现，具体的实现方式如上，查询结果如图 3.16 所示。

图 3.16 查询去过所有地点的人员结果图

11. 创建能反映所有隔离点现状的视图 isolation_location_status 其内容包括：地点编号（id）、隔离地点名（location_name）、房间容量（capacity）以及已占用量（occupied）；注意，正在隔离的人占用着隔离点的位置，隔离结束或已转院的人不占用位置。

```
create view isolation_location_status(id, location_name, capacity, occupied)
as select isolation_location.id, location_name, capacity, count(isol_loc_id) as occupied
    from isolation_location
      left outer join isolation_record
        on isol_loc_id = isolation_location.id and isolation_record.state = 1
  group by isolation_location.id, location_name, capacity
```

解题思路：由于可能存在隔离点不存在隔离者的情况，为防止由于自然连接产生的部分可能的地点的丢失，**因此外用外连接的方式进行连接**，使用创建视图

语句按照要求创建视图即可，查询结果如图 3.17 所示。

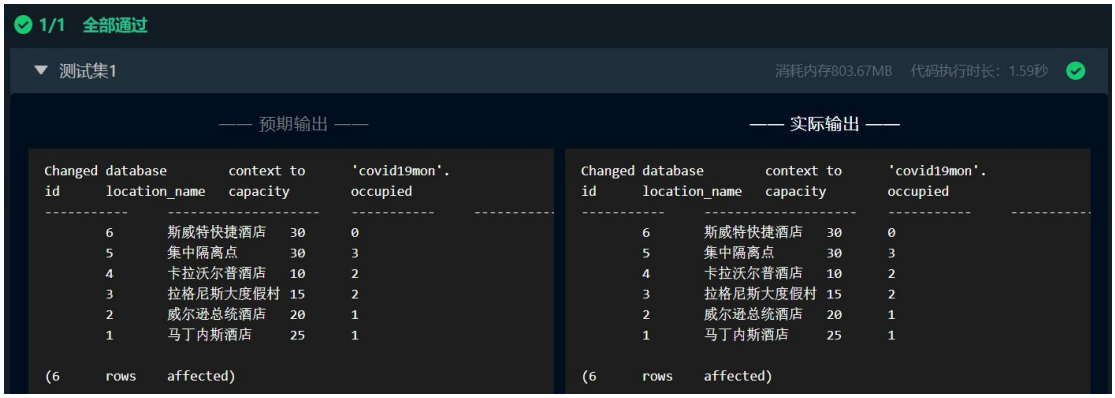


图 3.17 建立反映所有隔离点现状视图结果图

12. 从视图 isolation_location_status 中查询各隔离点的剩余空房间的数目。需要列出的数据项为：隔离点名称与剩余房间数，其中剩余房间数为计算得出，请给该列命名为 available_rooms，查询结果依隔离点编号排序。

```
select location_name, (capacity - occupied) as available_rooms
    from isolation_location_status
    order by id
```

解题思路：查询视图与查询基本表的方式是相同的，available_rooms 空余数使用容量（capacity）与占用量（occupied）计算即可，查询结果如图 3.18 所示。

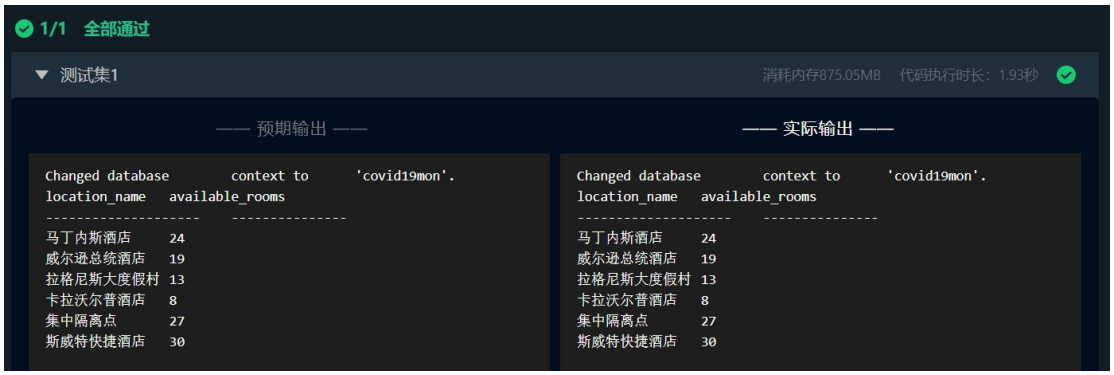


图 3.18 从视图查询各隔离点的剩余空房间结果图

13. 筛查发现，靳宛儿为无症状感染者，因此需要查询靳宛儿接触者的姓名和电话，以便通知并安排隔离。其中，靳宛儿在同一地点逗留时间有交集的均为其接触者，查询结果按照人员姓名排序。

```
select fullname, telephone from person
  where fullname != N'靳宛儿' and id in (select itinerary.p_id from itinerary
    where exists(select * from person, itinerary as temp
      where temp.p_id = person.id and fullname = N'靳宛儿'
        and itinerary.loc_id = temp.loc_id
        and itinerary.e_time - temp.s_time > 0
        and temp.e_time - itinerary.s_time > 0)
    )order by fullname
```

解题思路：本题思路与第 6 题的要求相似，因此解题思路也相似，但是由于查询信息为接触者的姓名，因此需要在一般的嵌套查询外增加一层查询，中间两层的查询与之前类似，不赘述，需要注意的是，查询出来的人员中可能包含靳宛儿本人，需要进行剔除，具体的查询结果如图 3.19 所示。

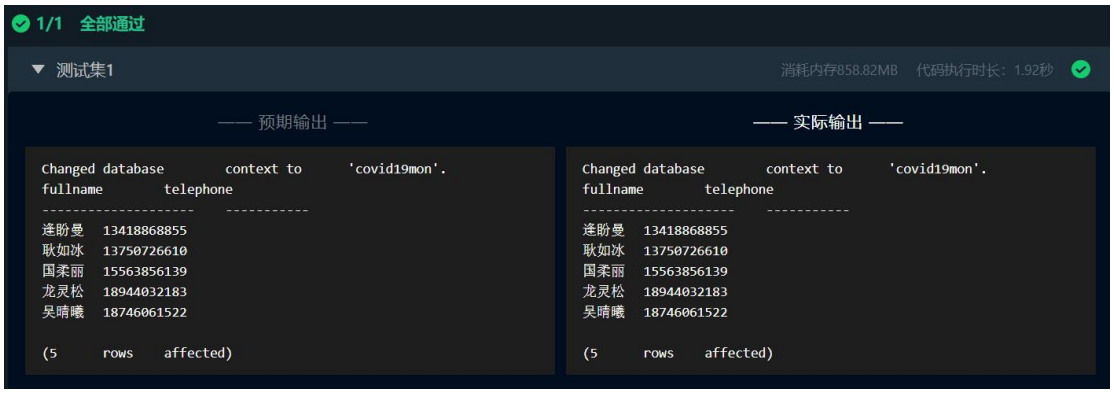


图 3.19 查询靳宛儿接触者结果图

14. 依据密切接触表的内容查询每个地点的密切接触者的数量，列出内容包括：地点名称，密接者人数。人数由统计获得列名命名为 close_contact_number，查询结果依密接者人数降序排列，密接者人数相同时依地点名称排序。

```
select location_name, count(*) as close_contact_number
    from close_contact, location
    where loc_id = location.id
    group by location_name
    order by close_contact_number desc, location_name asc
```

解题思路：简单的两表连接并分组查询，最后使用聚集函数 count 对人数进行统计即可实现此功能，具体的查询结果如图 3.20 所示。

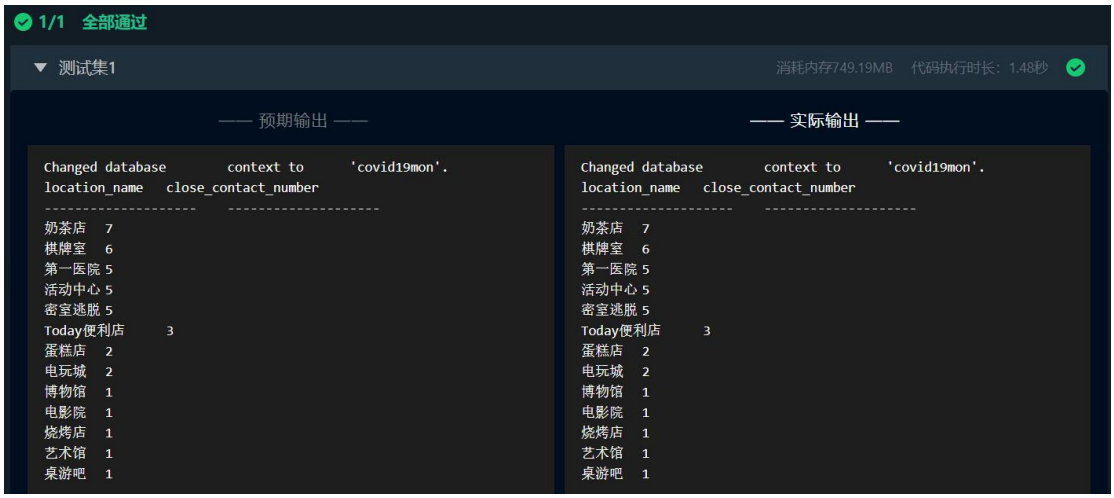


图 3.20 查询每个地点的密切接触者的数量结果图

15. 查询感染人数最多的人员编号，姓名和被其感染的人数,感染人数由统计所得，命名为 infected_number，注意，为确保其他测试程序的正确性并简化题目，此题暂简化为被密接者就是感染者。

```
select top 1 person.id as case_p_id, fullname, count(*) as infected_number
        from person, close_contact
        where person.id = case_p_id
        group by person.id, fullname
        order by infected_number desc
```

解题思路：简单的两表连接并分组查询，最后使用聚集函数 count 对人数进行统计，最后经过降序排序后，使用 top 函数取出最顶端的一条记录即是需要查询的数据，具体的查询结果如图 3.21 所示。

图 3.21 查询感染人数最多的人员结果图

16. 查询 2021-02-02 当天 10:00:00 到 14:00:00 期间，行程记录最频繁的三个人的姓名及行程记录条数，记录条数命名为 record_number，记录数并列的按姓名顺序排列。

```
select top 3 fullname, count(*) as record_number
      from person, itinerary
      where person.id = p_id and (
         s_time between '2021-02-02 10:00:00.000' and '2021-02-02 14:00:00.000'
         or e_time between '2021-02-02 10:00:00.000' and '2021-02-02 14:00:00.000'
      )
      group by fullname
      order by record_number desc, fullname
```

解题思路：简单的两表连接并分组查询，最后使用聚集函数 count 对人数进行统计；最后，经过降序排序后，并使用 top 函数取出最顶端的 3 条记录即是需要查询的人员记录数据。

注意，在此间的出行记录的变动都应计算，查询结果如图 3.22 所示。

图 3.22 查询行程记录最频繁的三个人员结果图

17. 从隔离点中，查询房间数(capacity)居第二多的隔离点名称及其房间数。

```
select location_name, capacity
    from isolation_location
    where capacity in (select top 1 capacity
        from (select top 2 capacity from isolation_location
            group by capacity
            order by capacity desc)as temp
        order by capacity asc
    )
```

解题思路：本题需要查询房间数第二多，房间数第二多的隔离点可能会超过一个，因此需要将各地点的房间数取出、去重并进行排序（使用两次 top 函数），取得第二多的房间数，最后用房间数找到对应的信息，查询结果如图 3.23 所示。

图 3.23 查询房间数第 2 多的结果图

3.2.4 创建并使用函数

1. 用 create function 语句创建符合以下要求的函数：

依据人员编号计算其到达所有地点的次数（即行程表中的记录数），函数名为 Count_Records，函数的参数名可以自己命名。

```
create function Count_Records(@person_id int)
returns int
begin
    declare @temp int;
    select @temp = count(*) from itinerary where p_id=@person_id;
    return @temp;
end
```

解题思路：本题的要点是熟悉 T-Sql 的语法，首先需要将返回值声明一个变量@temp，查询后赋值给该值并将改制返回即可。

2. 利用创建的函数，查询在行程表中至少有 3 条行程记录的人员信息，查询结果依人员编号排序。

```
select * from person
where dbo.Count_Records(id) >= 3
order by id;
```

解题思路：使用刚创建的函数进行查询测试，查询结果如图 3.24 所示。

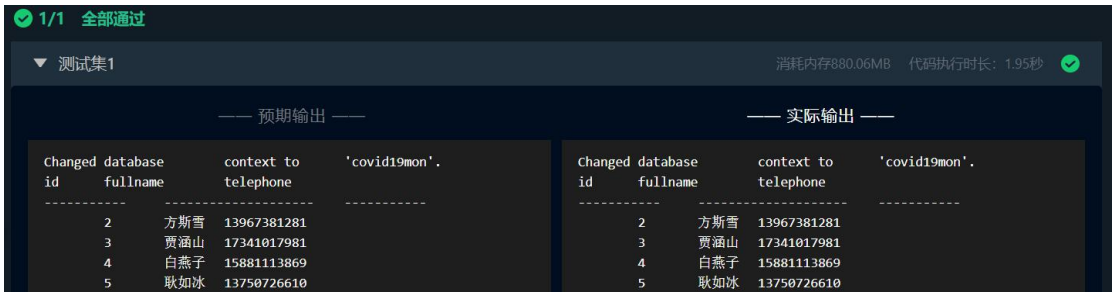


图 3.24 查询至少有 3 条行程记录人员结果图

3.3 任务总结

本次实验的主要任务的目的是帮助我们熟练掌握 SQL 语言的基本使用方法，包括数据库的创建、基本表创建以及表的插入、更新与删除操作，以及最为重要的数据查询操作。

在本次实验中，我基本掌握了 SQL 语言的各种基本语法，掌握了包括数据定义、数据控制以及数据查询相关的各类语法，掌握了数据查询的一些基本方式与技巧，可以灵活使用嵌套查询、连接查询、分组查询等查询方式进行数据的查询与修改等操作。同时，也尝试接触和使用一些 Sql Server 的其他的工具以及相关的工具，如触发器、函数以及视图等。

4 综合实践任务

4.1 系统设计目标

4.1.1 应用背景

本次选择的数据库选题是工资管理系统，工资管理系统是公司等需要发放工资的部门、企业系统中重要也是不可或缺的一部分，这个系统与公司中其他的系统，如考勤系统、加班系统以及员工的管理系统等存在比较紧密的关联性，实际情况较为复杂。

4.1.2 总体目标

本次数据库系统的设计实验不仅需要设计一个具有良好性能的数据库，还需要设计配套的客户端以便员工进行操作。因此，本次实验采用 C/S 模式进行系统设计，将使用 SQL Server 数据库作为数据库平台，QT 来设计 GUI 界面。

为避免出现数据不一致并减少冗余，需要设计一个高内聚、高共享以及低冗余的数据库系统。该数据库需要实现工种信息，部门信息，人员信息，工资信息等的集中管理；同时，对于不同权限的人有着不同的功能；此外，为使得工资系统具有灵活性，需要实现考勤、加班记录的增删改查，并能自动修改与这些记录相关的工资记录。

4.2 需求分析

工资管理系统需要实现工资信息相关的查询操作，此外由于工种信息以及考勤、加班记录与工资情况高度相关，所以此系统也应具有修改考勤、加班记录的功能，并可自动修改与之相关的工资数据。安全性方面需要注意，不同职级的人具有不同的权限，不同权限的可执行的操作不同。

在企业的工资管理系统中，该企业具有多个部门，每个部门具有多个员工，每个员工属于且只属于一个部门。每一个员工属于一个工种，该企业具有多个工种，每种工种具有基本工资信息，员工的基本工资取决于工种。

员工上班会产生考勤记录，加班会产生加班记录，考勤记录的状态以及加班记录的类型与时长等会影响当月的工资。

同时为最大程度还原系统，需要提供修改工种基本工资（不可删除增加工种，此功能不是工资管理的功能，应该是人事管理的功能），以及修改加班记录以及考勤记录的功能（具有权限限制），具体功能如下：

（1）设置与修改工种的基本工资

（2）考勤记录管理，根据考勤记录修改当月工资；

（3）加班津贴管理，根据加班时间和类型给予不同的加班津贴；

（4）员工能查看自身的基本信息，查看包括月收入、年收入、年终奖金以及自身考勤记录与加班记录在内的个人信息；

（5）部门管理者可查看本部门的工资情况，查看本部门的员工工资、考勤与加班情况，并提供修改考勤、加班记录的功能（同时自动修改工资）；

（6）企业管理者可查看企业的工资情况，查看单个员工的工资情况、每个部门的工资情况，并可查看并修改单个员工的考勤、加班记录

4.3 总体设计

4.3.1 系统 C/S 架构设计

本系统采用 C/S 模式，其中客户端使用 QT 实现，服务器端使用 SQL Server 实现。如图 4.1 所示，客户端向服务器端发送查询或修改请求，服务器端对此进行响应，返回查询数据或其他提示信息，客户端将获取的信息进行显示。

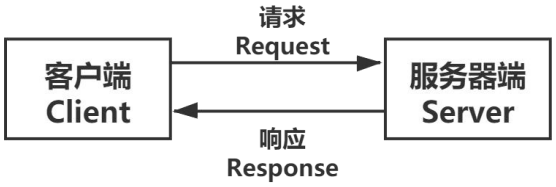


图 4.1 系统 C/S 架构图

4.3.2 功能模块设计

本系统分为客户端与服务器端，其中服务器端使用 SQL Server 实现，主要是数据库的设计，详见 4.4 数据库设计，本节主要说明客户端的设计。

客户端主要分为 5 个大模块，如图 4.2 所示，分别是登录模块、员工信息模块、部门管理模块、企业管理模块以及系统维护模块，每个大模块下又可以细分为若干小模块，具体模块以及其功能说明划分如下：

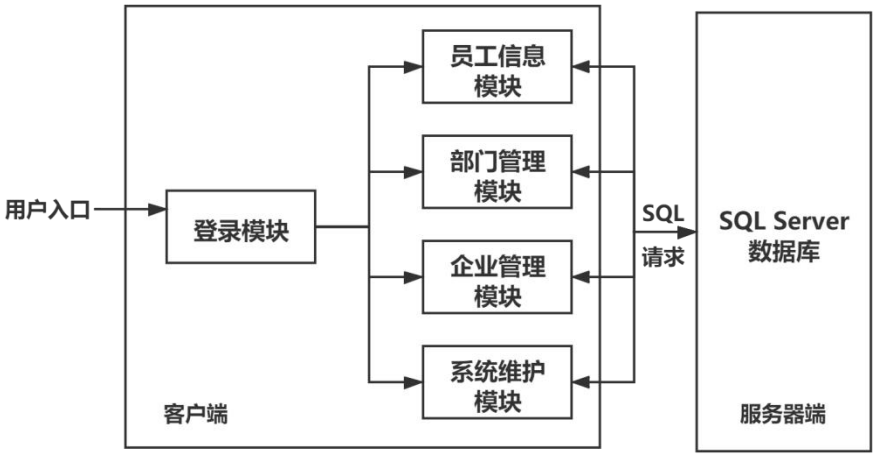


图 4.2 系统客户端总体架构图

1. 登录模块

当用户进入系统后，需要先进行用户身份的验证，也即用户登录，该模块负责用户身份的验证并读取其操作权限，生成用户对象用于之后模块的操作。当用户登录后，进入操作界面，可以依照**自身权限**查看自身信息，进行部门信息的查看与管理，进行企业信息的查看与管理以及进行系统维护。

2. 员工信息模块

该模块为员工个人信息的显示模块，可以显示当前登录员工的个人信息，所有权限都可以访问此功能。

如图 4.3 所示，当前模块分为若干子模块，这些子模块是并行的，子模块包括员工**个人基本信息的显示模块**、员工**月工资查看模块**、员工**年工资查看模块**、**考勤记录查看模块**、**加班记录查看模块**以及**密码修改模块**。

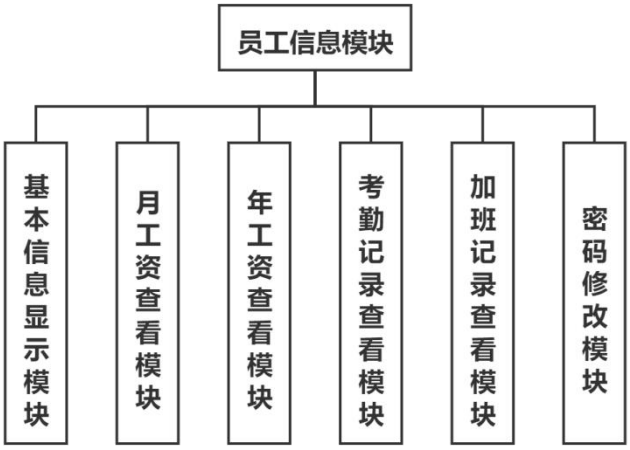


图 4.3 员工信息模块图

3. 部门管理模块

该模块为部门管理的模块，可以查看当前该部门管理者所在部门的部门工资报表，并以 Excel 文件的形式进行导出，也可以查看当前部门的员工的工资、考勤等信息并可以修改员工的考勤和加班记录。

当前模块的使用要求用户至少拥有**部门主管级的权限**，当前模块可以划分为若干子模块，包括**部门工资报表模块**、**员工信息查看模块**、**考勤记录修改模块**以及**加班记录修改模块**等。

4. 企业管理模块

该模块与部门管理模块功能类似，显示的内容相似，仅是显示的范围由单个部门向整个企业转变，其他模块和功能与部门管理相同。本模块的使用要求用户至少拥有企业主管级权限。

5. 系统维护模块

该模块的功能只有基本工资的调整，其他功能可以根据需求添加，本次为简化系统的实现，仅实现工种基本工资修改功能，本模块需用户拥有最高权限。

4.3.3 系统总体业务流程

如图 4.4 所示，用户进入系统后，首先在登录界面进行身份验证，身份验证成功即可进入系统，失败则继续停在登录界面；进入系统后，按照提示进行功能选择，选定功能需要进行权限验证，若权限验证通过则进行功能的执行，若验证失败则但会重新进行功能的选择。

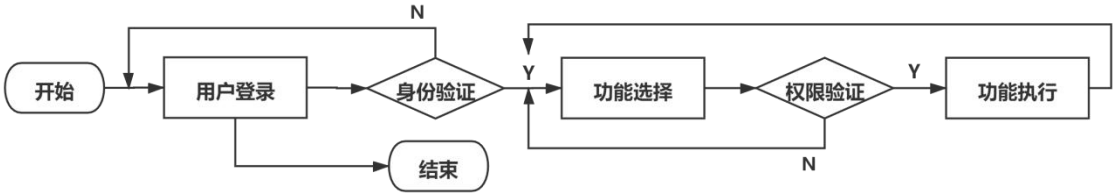


图 4.4 总体业务流程图

4.4 数据库设计

4.4.1 E-R 图设计

通过分析确定系统的 E-R 图如图 4.5 所示，实体包括员工、工种、部门、收入记录、考勤记录以及加班记录，具体的属性参见 E-R 图。

员工属于某一工种，员工任职于某一部门，某一部门包含若干员工；每个员工具有多条加班记录以及考勤记录；考勤记录与加班记录影响月工资；高权限员工可以修改加班记录以及考勤记录。

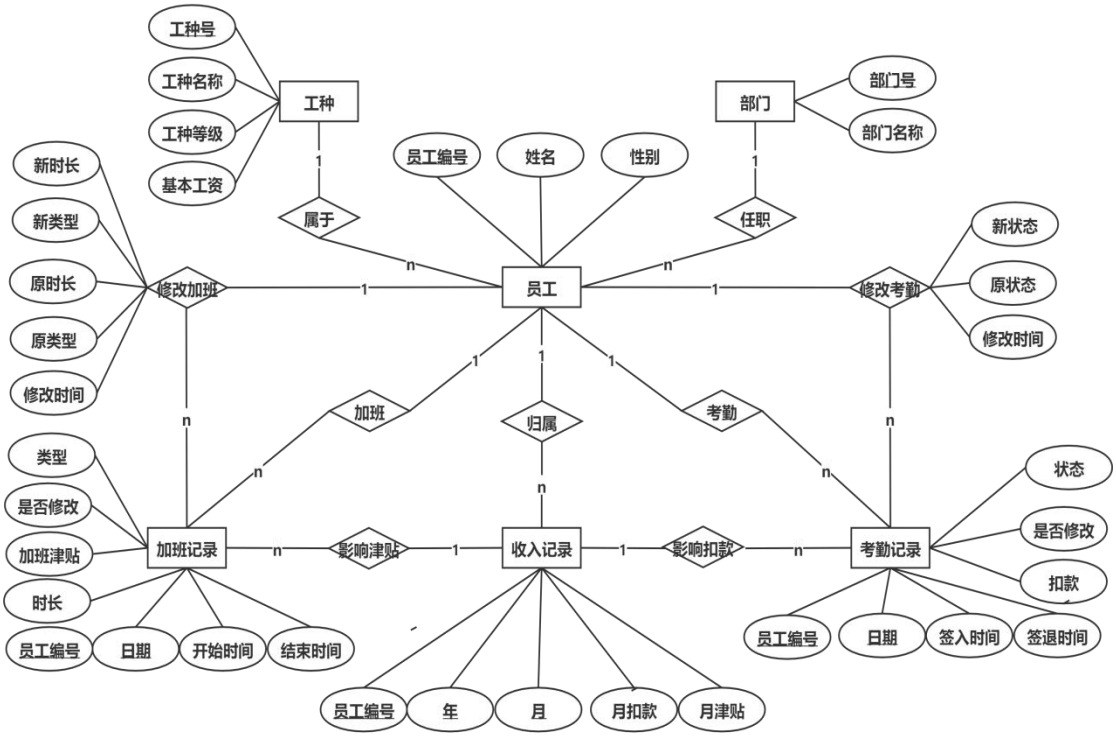


图 4.5 工资管理系统 E-R 图

4.4.2 数据库表设计

1. 工种信息表（workerType）

字段名	中文说明	字段类型	完整性约束
t_id	工种编号	int	primary key
t_name	工种名称	nchar(20)	not null
t_rank	工种等级	int	not null
base_salary	基本工资	money	not null

2. 部门信息表（department）

字段名	中文说明	字段类型	完整性约束
d_id	部门编号	int	primary key
d_name	部门名称	nchar(20)	not null

3. 员工信息表（workerInfo）

字段名	中文说明	字段类型	完整性约束
w_id	员工编号	int	primary key
w_name	员工姓名	nchar(20)	not null
w_sex	员工性别	nchar(1)	check(w_sex in(N'男', N'女')
w_telephone	员工电话	nchar(11)	not null
d_id	所属部门	int	foreign key department(d_id)
t_id	所属工种	int	foreign key workerType(t_id)
is_leave	是否离职	bit	

4. 考勤表（attendance）

属性 a_status 表示考勤状态，使用 int 型的数字表示，1 表示正常，2 表示迟到，3 表示缺勤，4 表示请假，5 表示早退。

字段名	中文说明	字段类型	完整性约束
w_id	员工编号	int	primary key foreign key workerInfo(w_id)
a_date	考勤日期	date	primary key
begin_time	签入时间	smalldatetime	not null
end_time	签退时间	smalldatetime	not null
a_status	考勤状态	int	check(a_status>0 and a_status<=5)
e_status	是否修改	bit	
payment	扣款	money	

5. 加班表（extraWork）

字段名	中文说明	字段类型	完整性约束
w_id	员工编号	int	primary key foreign key workerInfo(w_id)
e_date	加班日期	date	primary key
begin_time	开始时间	smalldatetime	not null
end_time	结束时间	smalldatetime	
duration	加班时长	int	not null
type	加班类型	int	not null
e_status	是否修改	bit	
payment	加班津贴	money	

6. 员工收入表（workerSalary）

字段名	中文说明	字段类型	完整性约束
w_id	员工编号	int	primary key foreign key workerInfo(w_id)
s_year	年	int	primary key
s_month	月	int	primary key check(s_month between 1 and 12)
base_pay	基本工资	money	not null
merit_pay	绩效工资	money	not null
extra_pay	加班津贴	money	not null

7. 系统用户表（dbuser）

用于用户登录，属性 permission 表示权限等级，使用 int 型的数字表示共分为 4 级，1 表示普通用户，2 表示部门主管，3 表示企业主管，4 表示系统管理员。

字段名	中文说明	字段类型	完整性约束
w_id	员工编号	int	primary key foreign key workerInfo(w_id)
pwd	密码	nvarchar(50)	not null
permission	权限等级	int	check(permission between 1 and 4)

8. 加班修改情况表（extraWork_editRecord）

字段名	中文说明	字段类型	完整性约束
w_id	员工编号	int	primary key foreign key workerInfo(w_id)
edit_time	修改时间	datetime	primary key
old_duration	旧加班时长	int	not null

old_type	旧加班类型	int	not null
new_duration	新加班时长	int	
new_type	新加班类型	int	
r_w_id	被改加班记录员工号	int	primary key foreign key extraWork(w_id)
r_date	被改加班记录日期	date	primary key foreign key extraWork(e_date)

9. 考勤修改情况表（attendance_editRecord）

字段名	中文说明	字段类型	完整性约束
w_id	员工编号	int	primary key foreign key workerInfo(w_id)
edit_time	修改时间	datetime	primary key
old_status	旧考勤状态	int	not null
new_status	新考勤状态	int	not null
r_w_id	被改考勤记录员工号	int	primary key foreign key attendance(w_id)
r_date	被改考勤记录日期	date	primary key foreign key attendance(e_date)

4.4.3 数据库视图设计

1. 月薪水视图（month_salary）

字段名	中文说明	字段类型	备注
w_id	员工编号	int	
w_name	员工姓名	nchar(20)	
d_id	部门编号	int	
d_name	部门名称	nchar(20)	
t_name	工种名称	nchar(20)	
s_year	年	int	
s_month	月	int	
base_pay	基本工资	money	
merit_pay	绩效工资	money	
extra_pay	加班津贴	money	

2. 年薪水视图（month_salary）

字段名	中文说明	字段类型	备注
w_id	员工编号	int	
w_name	员工姓名	nchar(20)	

d_id	部门编号	int	
d_name	部门名称	nchar(20)	
t_name	工种名称	nchar(20)	
s_year	年	int	
year_pay	年工资	money	年工资=基本工资+绩效工资+津贴
year_award	年终奖	money	年终奖=年工资/12

4.5 详细设计与实现

在 SQL Server 上使用建表语句以及建视图语句创建基本表和视图，使用 QT 设计客户端界面以及查询等逻辑，详细的设计与实现流程如下：

4.5.1 数据库触发器定义与实现

为简化数据存储，减少冗余信息，使用触发器修改月工资记录。分别在考勤记录表（attendance）以及加班记录表（extraWork）建立一个触发器，当对考勤记录/加班记录进行更新或插入新的考勤记录/加班记录时，需要根据新插入的记录信息修改月工资的数值，以实现加班/考勤对月工资的约束。

这两个触发器的实现过程类似，具体源码详见附录，以考勤表上的触发器 merit_salary 为例，使用游标 cursor 读取更新后（inserted）元组的某些属性值。

首先，根据更新后的考勤状态修改扣款属性，若新状态为 1（正常）或 4（请假），则将扣款置为 0；若新状态为 2（迟到）或 5（早退），则将扣款置为当天基本工资的 20%；若新状态为 3（缺勤），则将扣款置为当天全部基本工资。

```
if @newstatus = 1 set @newpayment = 0
if @newstatus = 2 set @newpayment = 0 - @base_salary / 21.75 * 0.2
if @newstatus = 3 set @newpayment = 0 - @base_salary / 21.75
if @newstatus = 4 set @newpayment = 0
if @newstatus = 5 set @newpayment = 0 - @base_salary / 21.75 * 0.2
```

然后，修改收入表中的绩效工资属性，若当月未创建工资记录，则先创建记录；若已存在记录，则将原绩效工资减去原扣款，加上新扣款即为新绩效工资。

```
-- 修改收入表中的月绩效工资
if not exists (select * from workerSalary where w_id = @newWid
   and s_year=convert(int,year(@newdate)) and s_month=convert(int,month(@newdate)))
      insert into workerSalary values(@newWid,convert(int,year(@newdate)),
            convert(int,month(@newdate)),@base_salary, 0 , 0);
update workerSalary
set merit_pay = merit_pay - @oldpayment + @newpayment
where w_id = @newWid and s_year = convert(int,year(@newdate))
      and s_month = convert(int,month(@newdate));
```

4.5.2 数据库事务定义与实现

由于数据库查询过程通常为单一语句，QT 可以自动将其定义为事务进行处理，因此不对查询过程进行显式的事务定义。

本系统中存在两处需要显式定义事务，分别是对考勤记录以及加班记录进行修改的两处代码逻辑。

以修改考勤记录为例，因为修改考勤记录必须同时修改考勤记录表（attendance）以及考勤修改情况表（attendance_editRecord），因此一旦发生异常需要将两张表同时进行回滚（Rollback）操作，因此需要显式定义事务进行处理，以修改考勤记录为例，具体事务定义如下：

利用 QT 封装的 QSqlQuery 模块，将事务的 SQL 代码合理拆分为嵌入式 SQL 的形式进行执行，首选执行“start transaction”表示事务开始，然后分别执行考勤记录表（attendance）的更新操作以及考勤修改情况表（attendance_editRecord）的插入操作，若两者均正常执行则用“commit”进行提交，否则用“rollback”进行回滚。

```
QSqlQuery transaction_start, transaction_commit, transaction_rollback;
QSqlQuery query_insert, query_update;

transaction_start.exec("start transaction");
bool ok1=query_update.exec("update attendance set a_status="+new_status+", e_status=1"
                           "where w_id = " + id + " and a_date = '" + date + "';");
bool ok2 = query_insert.exec("insert into attendance_editRecord values "
                           "(" + w_id +", getdate(), " + old_status +", "
                           + new_status + ", " + id + ", '" + date +"');");
if(ok1 && ok2) transaction_commit.exec("commit");
else transaction_rollback.exec("rollback");
```

修改加班记录的流程与之类似，不在赘述，详见源码。

4.5.3 主要模块设计与实现

客户端主要分为 5 个大模块，分别是登录模块、员工信息模块、部门管理模块、企业管理模块以及系统维护模块。

其中，部门管理模块与企业管理模块共用一份代码，通过访问的权限来判断当前进行的管理模式，因此本处仅对登录模块、员工信息模块、部门管理模块以及系统维护模块 4 大模块的设计与实现过程进行描述，具体过程如下：

1. 登录模块

进入系统后首先打开的就是登录窗口，如图 4.6 所示，用户需要输入员工编号以及密码进行身份验证，验证成功后即可进入正式的业务操作界面，初始密码统一为“123456”；若验证失败则留在此界面进行等待下次输入。

图 4.6 登录界面示意图

当用户输入员工编号和密码之后，点击登录按钮进入登录判断逻辑。

验证流程如图 4.7 所示，首先，判断员工编号与密码是否为空，若其中一个为空则验证失败；若两者都不为空，则获取用户输入的信息，然后通过 select 语句在 SQL 数据库中查询用户名对应的密码。如果没有查询到该用户名，说明用户不存在；否则将查询到的密码与用户输入的密码进行比对，如果密码一致则登录成功，否则登陆失败。

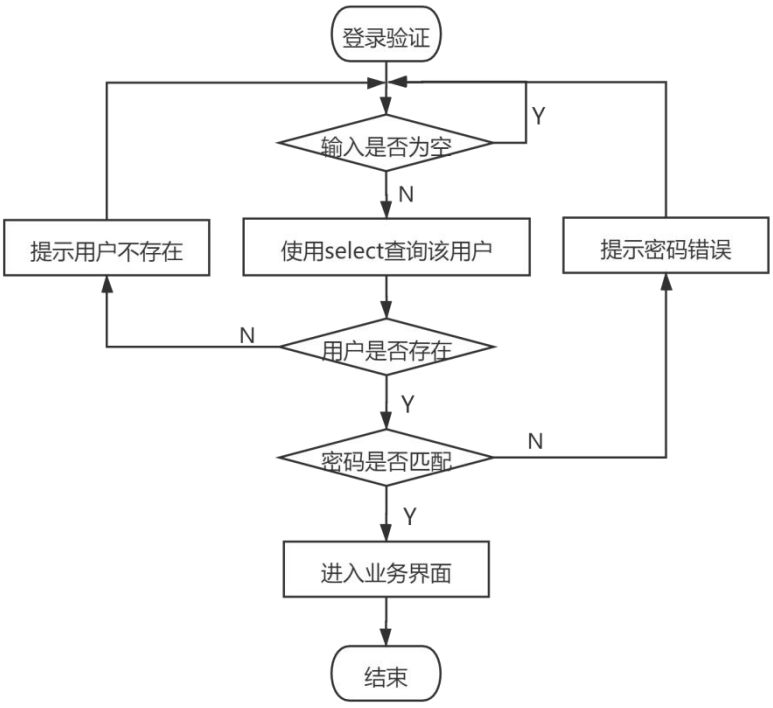


图 4.7 登录验证逻辑流程图

如图 4.8 所示，当用户进入业务界面后，可以依照**自身权限**进行相应的操作，高级用户拥有低级用户的一切操作权限，可以进行低级用户的所有操作。

对于普通员工权限（1 级）用户，只能进行自身信息的查看；对于部门主管级权限（2 级）用户可以进行部门信息的查看与管理；对于企业高管级权限（3 级）用户，可以进行企业信息的查看与管理；对于超级管理员权限（4 级）用户

可以额外进行系统维护。

图 4.8 业务选择界面示意图

选择业务后，首先需要进行权限验证，若权限不匹配，则系统会弹窗提示“抱歉，您的权限过低！”，拒绝当前操作并返回业务选择逻辑界面；如果权限验证通过，则会进入对应的操作页面。

2. 员工信息模块

该模块的功能是显示员工的各类信息，包括个人的基本资料、月收入、年收入以及考勤、加班记录情况，同时，该系统也具有修改密码的功能。各个模块使用按钮进行访问，各个模块之间不存在交集，业务逻辑是完全并行的。

对于信息查看类模块（除修改密码模块），都是使用 QT 的 SQL 查找模块 QSqlQuery 以及 QSqlQueryModel 进行查找并使用 tableView 插件进行显示即可，显示流程一致，具体参见**源代码**，在此不就此进行叙述。

对于密码修改模块，操作流程与登录的流程相似，按照提示两次输入新密码，若两次输入存在空，则直接提示输入为空，若两次输入不相同则提示错误；若两次相同，则用 update 语句对用户表进行更新，密码修改界面如图 4.9 所示。

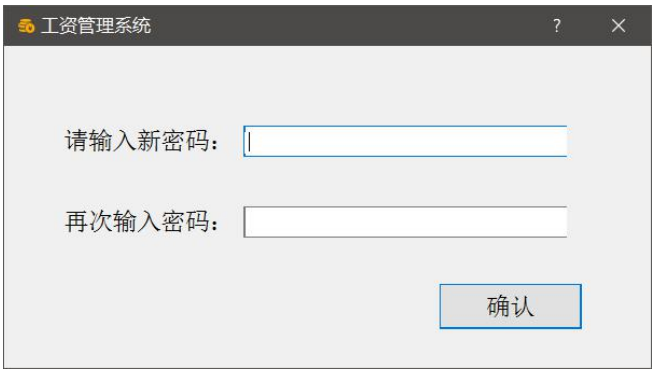


图 4.9 密码修改界面示意图

3. 部门管理模块

部门管理界面如图 4.10 所示，在此界面可以查看当前该部门管理者所在部门的部门工资报表，并以 Excel 文件的形式进行导出，也可以查看当前部门的员

工的工资、考勤等信息并可以修改员工的考勤和加班记录。这些功能都使用按钮进行访问，不同模块之间的业务是完全并行的，不存在交集。

图 4.10 部门管理界面示意图

当按下“查看工资报表”按钮即可查看部门的工资报表，可以按照需求选择按月查看或按年查看，并可以选择导出为 Excel 文件，如图 4.11 为按月查看的工资报表格式，使用 QSqlQueryModel 输入对应的 select 语句进行查找并使用 tableView 插件进行显示。

```
QSqlQueryModel *model = new QSqlQueryModel(ui->tableView);
QString str = "select d_name as '部门名称',"
              "s_year as '年',"
              "sum(base_pay + merit_pay + extra_pay) as '年部门工资支出' "
              "from month_salary ";
if(is_depart) str = str + " where d_id = " + d_id ;
str = str + " group by d_id, d_name, s_year order by s_year;";
model->setQuery(str);
ui->tableView->setModel(model);
```

图 4.11 按月查看部门工资报表界面示意图

当按下“查询员工工资”按钮，可进入员工工资查询逻辑，首先界面会显示可以查看到的员工信息（部门管理可看到本部门员工信息，企业管理可看到所有员工信息）。

输入员工编号并点击查询即可查看对应员工的工资情况，如图 4.12 所示，可查看 1 号员工的工资情况。实现逻辑与之前查询的逻辑类似，不赘述。

图 4.12 查询员工工资界面示意图

修改员工考勤状态与修改加班状态的实现方式类似。

以修改员工考勤状态为例，当按下“修改员工考勤状态”按钮，进入修改考勤状态界面如图 4.13 所示，输入相对应的员工编号，考勤日期以及修改后的新状态，点击确认修改即可修改状态，如果输入的新状态与原状态相等，则会提示与原状态相同，拒绝修改。

若与原状态不同则会进行修改，**由于考虑到每次修改都需要有记录保留以供后续进行复盘**，因此需要同时修改两张表考勤表（attendance）以及考勤修改情况表（attendance_editRecord），考勤修改表会记录修改人的编号，修改时间以

及修改前后的状态。这两张表的修改时必须同时进行不可分割的，因此需要显式定义为事务 transaction，事务定义参见 **4.5.2**，在此不赘述。

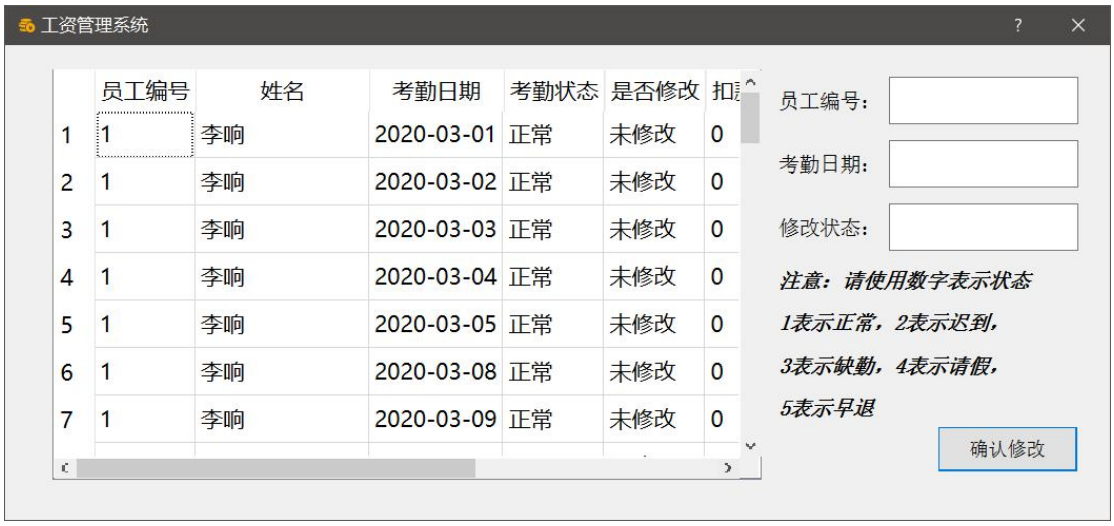


图 4.13 修改考勤状态界面示意图

4. 系统维护模块

该模块的功能只有基本工资的调整，其他功能可以根据需求添加，基本工资修改界面如图 4.14 所示，工种基本工资的修改与修改考勤记录也是类似，输入工种编号并输入新基本工资即可，需要注意的是修改完成的基本工资需要到下月生效，当月不生效。

图 4.14 修改基本工资界面示意图

4.6 系统测试

配置数据库，在 SQL Server 端，首先使用建表与建视图语句创建基本表与视图，然后创建触发器，最后插入测试数据；在客户端需要创建 ODBC 数据源，并使用 QT 进行连接，开始测试。

为了能够合适地测试系统功能，本次实验测试设计了专门的测试样例。

首先，设计了 3 个不同的部门，包括 1 号部门（**销售部**）、2 号部门（**产品部**）以及 3 号部门（**技术部**）；设计 3 个不同的工种，包括 1 号工种（**工程师**）、2 号工种（**销售**）以及 3 号工种（**产品经理**）。

其中，技术部插入 1 名员工，具有超级用户权限；产品部与销售部均插入两名员工，并授予对应的权限，每个人的初始密码置为 123456。最后，插入相对应的考勤记录以及加班记录，**详细测试数据见附录**，具体测试流程如下所示：

1. 测试环境

（1）操作系统：Windows10 家庭版

（2）开发语言：C++、SQL

（3）客户端开发：QT

（4）数据库开发：Microsoft SQL Server 2019

2. 用户登录测试

首先，打开应用进入登录界面，用 1 号用户登录，其权限为超级权限。

如图 4.15 所示，输入对应的员工编号与密码，密码会以密文形式显示，正常登录会进入如下所示的业务选择界面。

当登录验证异常会弹窗显示错误信息，如图 4.15 所示，若用户名或密码为空则提示“请输入用户名后登录”或“请输入密码后登录”；若当前用户不存在会提示“用户不存在”，若密码错误则提示密码错误。

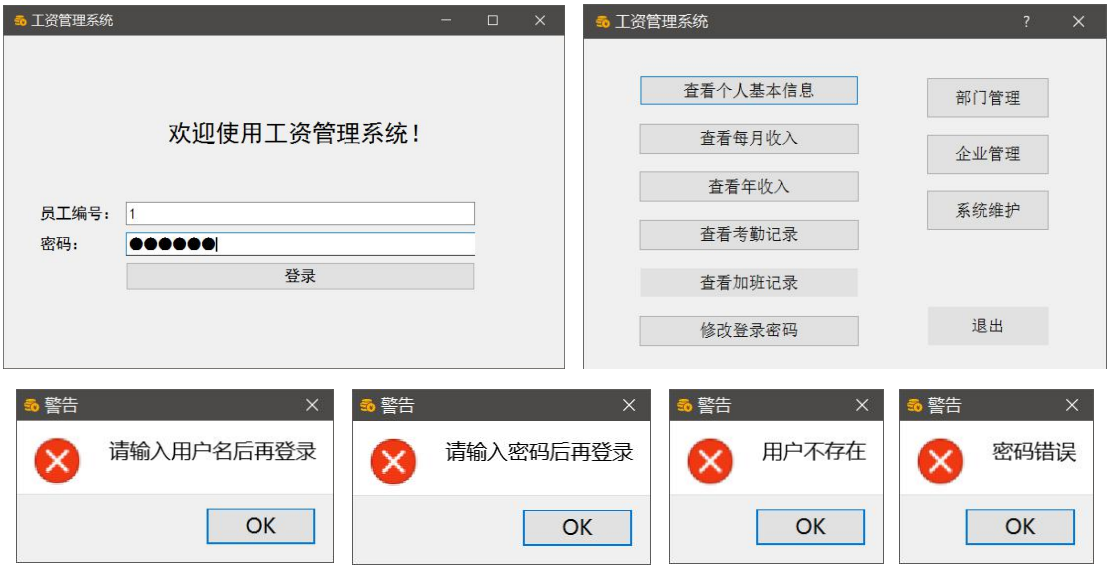


图 4.15 登录界面以及错误信息提示图

如图 4.15 所示，进入业务选择界面后，可以看到大量的按钮，通过点击不同的按钮即可进行入不同的业务中。

左侧按钮为员工信息显示模块（包括登录密码的修改）的业务，右侧可以选择进入部门管理、企业管理以及系统维护的相关业务界面，右下角为退出按钮，

可以返回登录界面，点击左侧的按钮可以进行个人信息的查看，具体测试见 3。

3. 查看个人信息测试

点击不同的按钮可以查看不同的信息，点击“查看个人基本信息”按钮，会弹窗显示个人基本信息，如图 4.16 所示，可以看到本人的各项基本信息，包括工号、姓名、性别、电话、所属部门以及所属工种。

图 4.16 个人基本信息图

点击“查看月收入”以及“查看年收入”按钮，会弹窗显示工资信息，如图 4.17 所示，可以看到员工的月工资与年工资，其中月工资的分类较为详细，收入的各项组成（基本工资、扣款以及加班津贴）都有所显示；年工资栏除了显示年工资，同时会显示年终奖金，该值为年工资的 12 分之一。

图 4.17 员工个人月工资与年工资情况图

点击“查看考勤记录”以及“查看加班记录”按钮，会弹窗显示考勤与加班信息，如图 4.18 所示，考勤记录与加班记录的条目比较多，员工仅仅可以查看

这些记录，不能对这些记录进行任何形式的修改。

图 4.18 员工考勤与加班情况图

4. 密码修改测试

点击“修改登录密码”按钮，进入密码修改界面，如图 4.19 所示，按照提示重复输入新密码即可完成修改，密码以密文的形式显示，可以方式泄密，当修改成功系统弹出提示“密码修改成功”。

但是，如果密码长度小于 6，系统会提示“密码长度不得小于 6”；若两次输入不相同，则系统提示两次输入不相同。

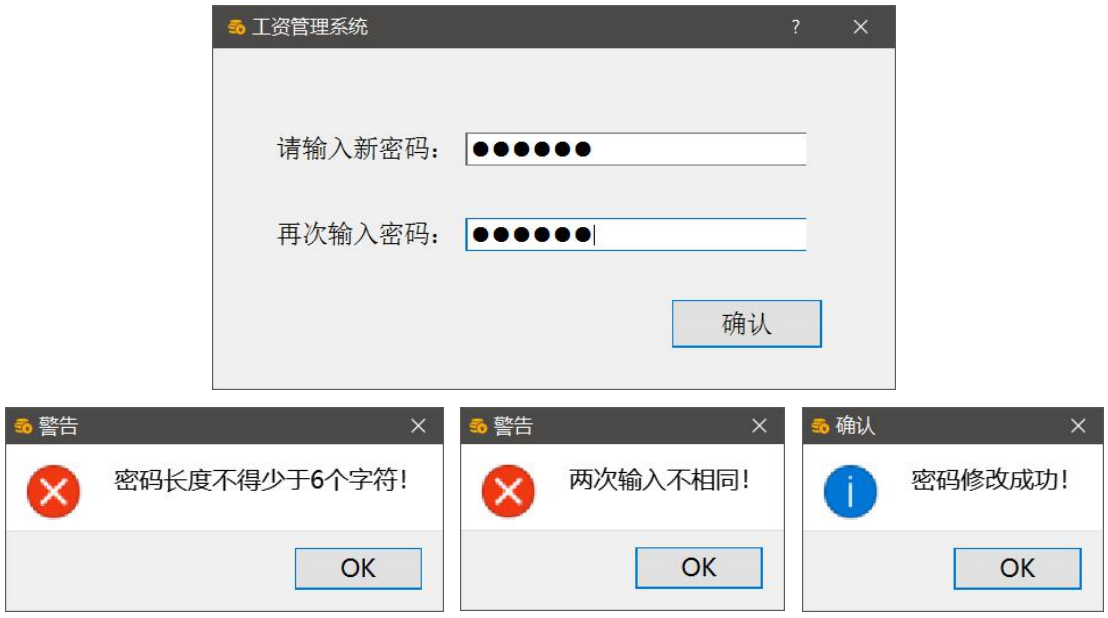


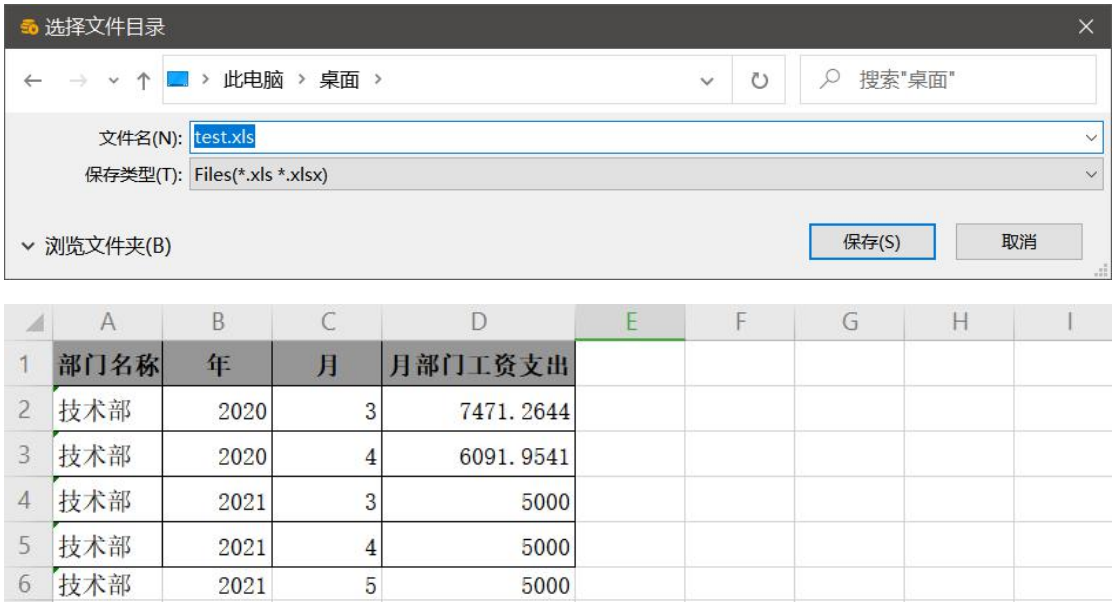
图 4.19 密码修改界面与提示图

5. 查看工资报表测试

点击“查看工资报表”按钮，如图 4.20 所示，进入工资报表显示界面，点击按钮“按月查看”或“按年查看”可以在年工资报表与月工资报表间进行切换。

同时点击“导出为 Excel”按钮可以实现数据的导出功能，如图 4.21 所示，先选择合适的文件路径与文件名，点击保存，即可自动生成对应的 Excel 文件，文件效果如图 4.21 所示。

图 4.20 按月查看与按年查看部门工资报表示意图



	A	B	C	D	E	F	G	H	I
1	部门名称	年	月	月部门工资支出					
2	技术部	2020	3	7471.2644					
3	技术部	2020	4	6091.9541					
4	技术部	2021	3	5000					
5	技术部	2021	4	5000					
6	技术部	2021	5	5000					

图 4.21 导出为 Excel 功能示意图

6. 查询员工工资测试

点击“查询员工工资”按钮，如图 4.22 所示，进入查询员工工资界面。首先，界面上会显示该部门或企业的所有员工的基本信息；然后，在员工编号输入框中输入需要查询的员工的编号，点击“查询”，如图 4.23 所示，可以查询该员工的工资记录，同时可以使用导出功能将需要的数据进行导出。

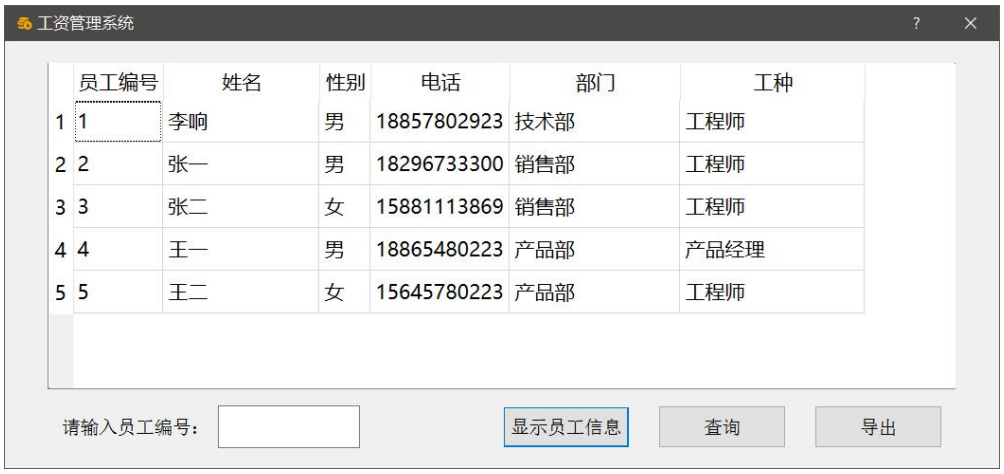


图 4.22 企业所有员工信息查询示意图

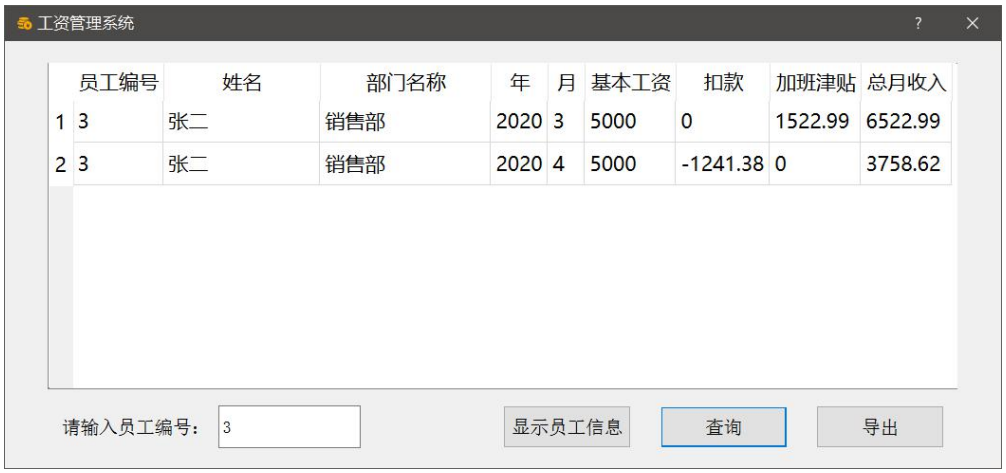


图 4.23 查询单个员工工资情况示意图

7. 修改员工考勤测试

点击“修改员工考勤状态”按钮，如图 4.25 所示，找到需要修改的记录，输入对应的员工编号、考勤日期以及修改后的新状态号（1 表示正常，2 表示迟到，3 表示缺勤，4 表示请假，5 表示早退），最后点击“确认修改”。

若修改后的状态与原状态相同，如图 4.24 所示系统报错，弹窗显示错误信息“更新状态与原状态相同，修改失败”；若修改后的状态与原先不相同，如图 4.25 所示，修改后会重新显示考勤记录，考勤状态会发生改变，是否修改属性变为“修改”表示受到修改，需要注意，同时扣款也会发生变化。

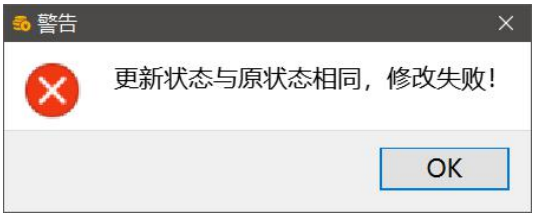


图 4.24 修改考勤状态错误提示图

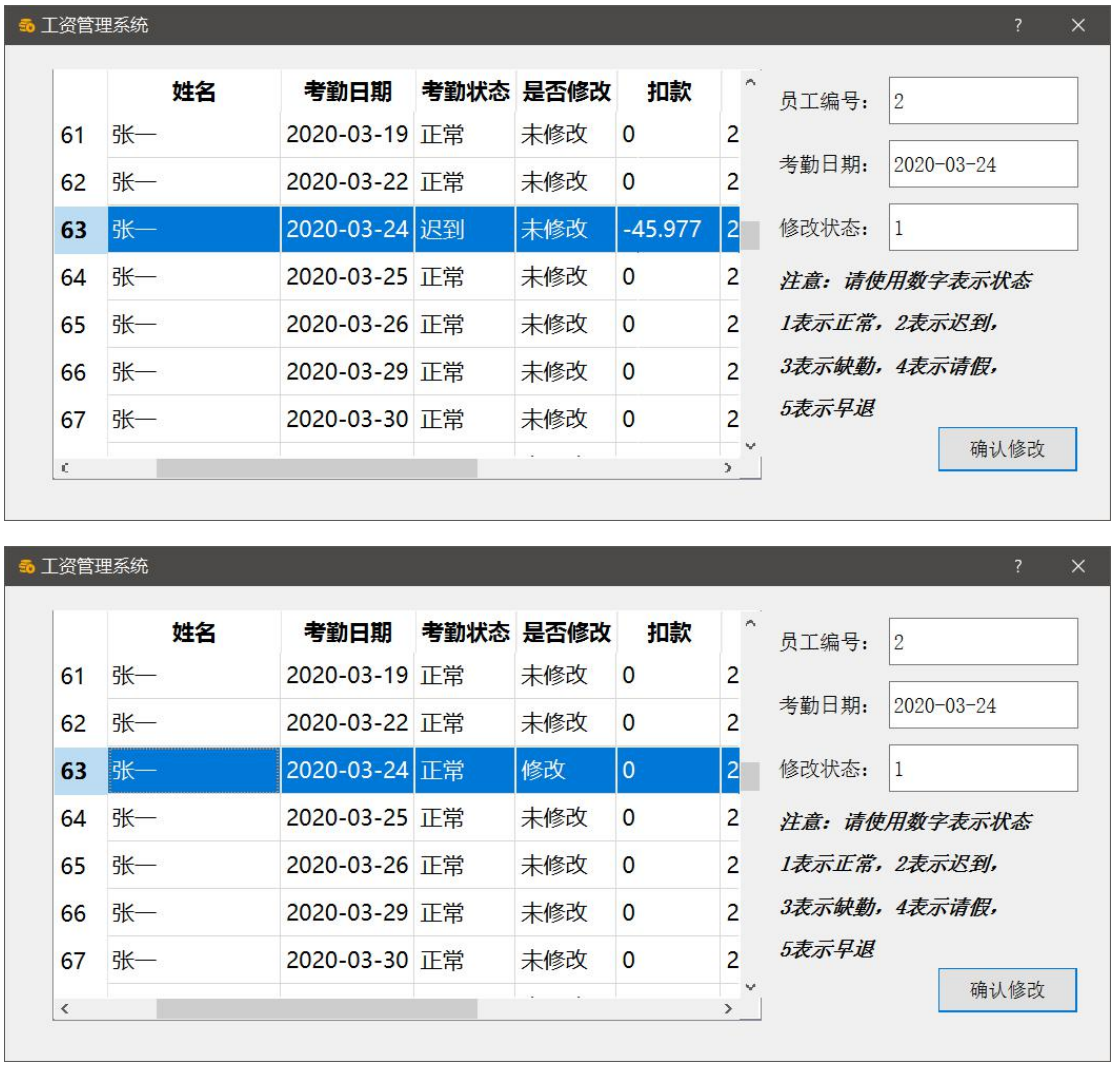


图 4.25 修改员工考勤状态前后示意图

为提高系统的可靠性，**对于考勤记录的修改都需要进行留档记录**，用于检查复盘之用。

如图 4.26 所示，每一条修改记录都包含：修改者员工编号、修改时间、被修改的考勤记录的主键（同时作为外键参照考勤表）以及修改前后的状态值信息，这些信息可以有效防止某些人员恶意修改考勤表。

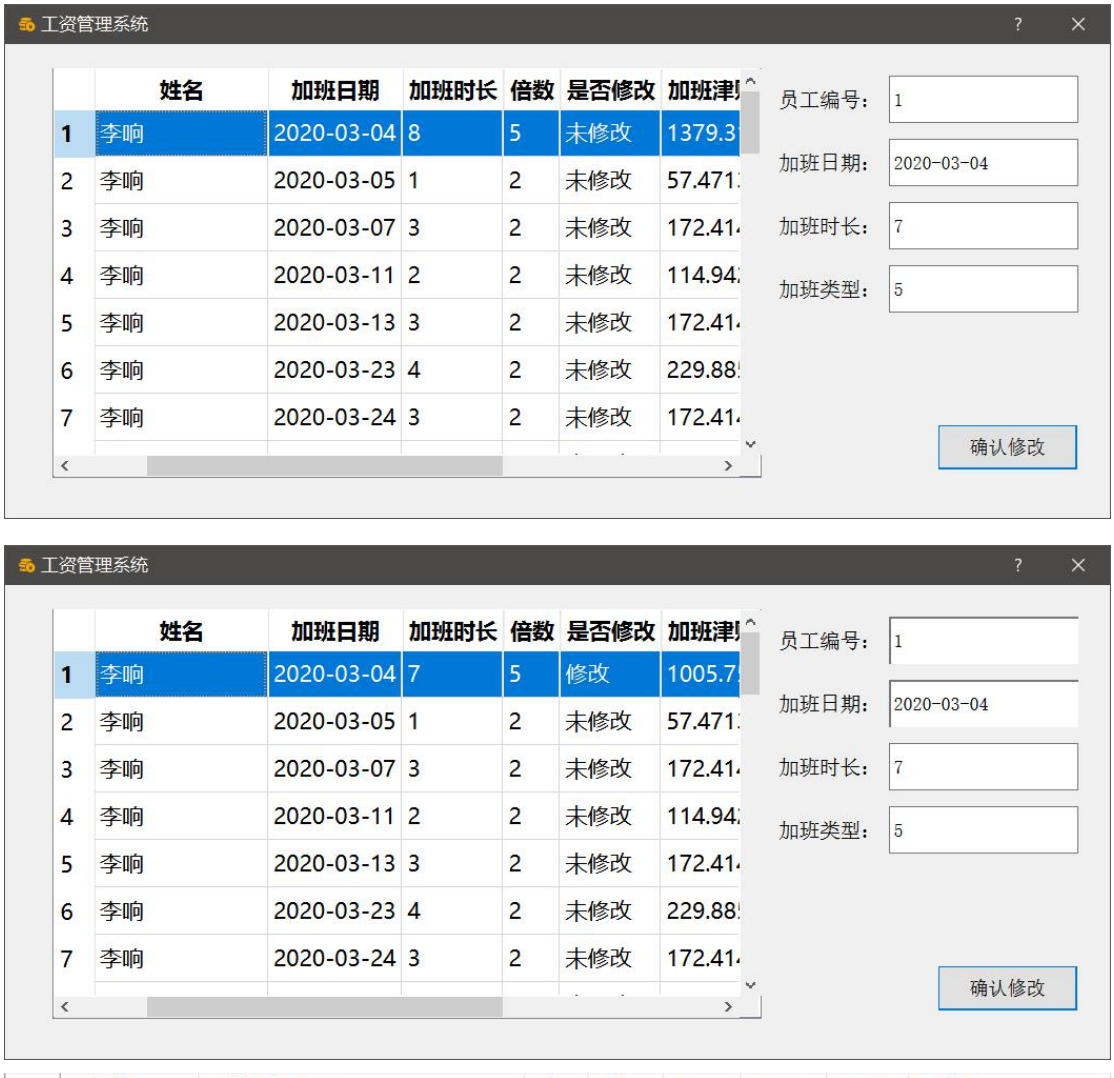
	w id	edit time	old status	new status	r w id	r date
▶	1	2021-07-08 12:36:44.250	2	1	2	2020-03-24
*	NULL	NULL	NULL	NULL	NULL	NULL

图 4.26 考勤修改表内容截图

8. 修改员工加班测试

修改员工加班记录的实现逻辑与考勤记录极为相似，测试方式也十分相似，具体的测试结果如图 4.27 所示，具体不在此进行赘述，修改加班时长与类型（为简化设计加班类型被简化为倍数，也即是可以获得多少倍与基本工资的加班津

贴）后，会自动修改加班津贴数额，并修改“是否修改”属性。



	w id	edit time	ol...	ol...	ne...	new...	r w...	r date
▶	1	2021-07-08 12:15:07.403	3	2	5	3	1	2020-03-04
	1	2021-07-08 12:15:34.910	5	3	7	5	1	2020-03-04
*	NULL	NULL	N...	NU...	NULL	NULL	NULL	NULL

图 4.27 修改员工加班记录结果测试图

9. 修改工种基本工资测试

进入系统维护界面后，选择进入“修改基本工资”按钮进入修改基本工资界面，将需要修改基本工资的工种编号以及基本工资数输入文本框中，点击“确认修改”即可实现修改，修改效果如图 4.28 所示。

图 4.28 修改工种基本工资测试图

4.7 系统设计与实现总结

在本次工资管理系统的应用系统设计与实现中，我围绕工资管理系统的基本需求进行分析和设计，实现了一个基本功能完善，业务逻辑清晰的系统，完成了数据库服务器与客户端的合理互动，具体完成的工作如下：

（1）设计了系统的总体框架，包括模块的划分以及各模块之间的联系，系统整体使用C/S模式进行设计，使用QT设计客户端以及UI界面，使用SQL Server实现服务器端设计；

（2）设计了数据库的概念模型（E-R图），并将E-R图分解为合适基本表结构，设计并实现了数据库的模式与外模式，创建了基本表与视图，定义了完善的完整性约束条件，并使用触发器实现工资和奖金的处理；

（3）实现了用户登录时的身份验证功能；

（4）实现权限控制，共包含4级权限，包括1级权限（员工权限）、2级权限（部门主管权限）、3级权限（企业高管权限）以及4级权限（超级权限）；

（5）实现了员工信息查看功能；

（6）实现了部门管理以及企业管理功能，这两大功能包含四个子功能，包括查看工资报表、查询员工工资、修改考勤记录以及修改加班记录的功能（**修改考勤和加班记录会有修改记录留存，修改记录不可修改**）；

（7）实现了工种基本工资的修改功能；

（8）实现了数据库事务，以保证数据库的一致性。

5 课程总结

本次数据库实验分为三个部分，其中第一与第二部分是关于 SQL 软件的基本使用方法以及 SQL 语言的基本语法，第三部分需要设计一个业务逻辑清晰且可靠的数据库应用系统，在本次实验中我完成了很多工作，具体如下：

（1）通过实验 1，掌握了 SQL Server 以及对应的数据库管理软件的基本功能和使用方法，掌握了数据备份与恢复的基本流程，了解并学习了安全性控制相关的授权与回收操作，同时接触了数据的导出与导入；

（2）通过实验 2，充分练习并掌握了 SQL 语言的基本语法，包括创建数据库、创建表、创建视图、添加完整性约束、数据插入、数据删除、数据更新、数据查询、创建触发器、创建并使用函数等 SQL 语言流程；

（3）通过实验 3，学习并掌握了 SQL Server 与 QT 联合编程的流程与 QT 中嵌入式 SQL 语言的使用方式，了解了 QSqlQuery 相关模块的使用方式；完成了工资管理系统的设计与实现，包括需求分析、总体设计、数据库设计、详细设计与测试环节，基本掌握了基于 SQL Server 与 QT 的简易数据库应用系统的开发流程和基本方法。

本次实验从易到难，分模块对我的数据库管理和使用能力进行了训练和检验，通过本次课程，我不仅掌握了 SQL Server 的基本使用方法，也学会了 SQL 语言的基本语法，这对之后接触其他的关系数据库有着极强的帮助作用。

此外，此次实践也让我独立地开发了一个简易可用的数据库应用，在此过程中，我体会到了数据库管理系统的强大功能，也了解到了开发的基本流程。

在此次实践中，我所开发的应用程序符合任务要求，功能完备业务的逻辑清晰，且系统稳定性高，异常处理能力强。

附录

Create.sql 用于创建基本表与视图：

```
/* ********************************************************** */
-- 表 1  工种信息表(workerType)
 create table workerType(
       t_id int, t_name nchar(20) not null,
       t_rank int not null, base_salary money not null,
       constraint pk_workerType primary key(t_id)
 );
 -- 表 2  部门信息表(department)
 create table department(
       d_id int, d_name nchar(20) not null,
       constraint pk_department primary key(d_id)
 );
-- 表 3  员工信息表(workerInfo)
 create table workerInfo(
       w_id int, w_name nchar(20) not null,
       w_sex nchar(1) check(w_sex in(N'男',N'女')),
       w_telephone nchar(11) not null,
       d_id int not null, t_id int not null, is_leave bit,
       constraint pk_workerInfo primary key(w_id),
       constraint fk_workerInfo_department foreign key(d_id) references department(d_id),
       constraint fk_workerInfo_workerType foreign key(t_id) references workerType(t_id)
 );
-- 表 4  考勤表(attendance)
-- 根据考勤情况设置奖惩，1 表示正常，2 表示迟到，3 表示缺勤，4 表示请假，5 表示早退
create table attendance(
       w_id int not null, a_date date not null,
       begin_time smalldatetime not null, end_time smalldatetime,
       a_status int not null check(a_status > 0 and a_status <= 5), e_status bit, payment money,
       constraint pk_attendance primary key(w_id, a_date),
       constraint fk_attendance_workerInfo foreign key(w_id) references workerInfo(w_id)
);
-- 表 5  加班表(extraWork)
create table extraWork(
       w_id int not null, e_date date not null,
       begin_time smalldatetime not null, end_time smalldatetime,
       duration int not null, type int not null, e_status bit, payment money,
       constraint pk_extraWork primary key(w_id, e_date),
       constraint fk_extraWork_workerInfo foreign key(w_id) references workerInfo(w_id)
);
-- 表 6  员工收入表(workerSalary)
```

```
create table workerSalary(
        w_id int not null, s_year int not null,
        s_month int not null check( s_month between 1 and 12 ),
        base_pay money not null, merit_pay money not null, extra_pay money not null,
        constraint pk_workerSalary primary key(w_id, s_year, s_month),
        constraint fk_workerSalary_workerInfo foreign key(w_id) references workerInfo(w_id)
);
-- 表 7  系统用户表(dbuser)
create table dbuser(
        w_id int not null, pwd nvarchar(50) not null,
        permission int not null check(permission >= 1 and permission < 5),
        constraint pk_dbuser primary key(w_id),
        constraint fk_dbuser_workerInfo foreign key(w_id) references workerInfo(w_id)
);
-- 表 8  加班修改情况表(extraWork_editRecord)
create table extraWork_editRecord(
        w_id int not null, edit_time datetime not null,
        old_duration int not null, old_type int not null, new_duration int, new_type int,
        r_w_id int not null, r_date date not null,
        constraint pk_extraWork_editRecord primary key(w_id, edit_time, r_w_id, r_date),
        constraint fk_extraWork_editRecord_extraWork foreign key(r_w_id, r_date)
            references extraWork(w_id, e_date),
        constraint fk_extraWork_editRecord_workerInfo foreign key(w_id)
            references workerInfo(w_id)
);
-- 表 9  考勤修改情况表(attendance_editRecord)
create table attendance_editRecord(
        w_id int not null, edit_time datetime not null,
        old_status int not null, new_status int not null,
        r_w_id int not null, r_date date not null,
        constraint pk_attendance_editRecord primary key(w_id, edit_time, r_w_id, r_date),
        constraint fk_attendance_editRecord_attendance foreign key(r_w_id, r_date)
            references attendance(w_id, a_date),
        constraint fk_attendance_editRecord_workerInfo foreign key(w_id)
            references workerInfo(w_id)
);
-- 创建月薪水表（包括月基本工资）
 create view month_salary(w_id, w_name, d_id, d_name, t_name, s_year,
                                  s_month, base_pay, merit_pay, extra_pay)
 as select workerInfo.w_id, w_name, department.d_id, d_name, t_name,
                                   s_year, s_month, base_pay, merit_pay, extra_pay
      from workerSalary, workerInfo, department, workerType
      where workerSalary.w_id = workerInfo.w_id and
            workerInfo.t_id = workerType.t_id and workerInfo.d_id = department.d_id;
```

```
go
-- 创建年薪水表（包括年终奖）
 create view year_salary(w_id, w_name, d_id, d_name, t_name, s_year, year_pay, year_award)
 as select workerInfo.w_id, w_name, department.d_id, d_name, t_name, s_year,
      sum(base_pay + merit_pay + extra_pay), sum(base_pay + merit_pay + extra_pay)/12
      from workerSalary, workerInfo, workerType, department
      where workerSalary.w_id = workerInfo.w_id and
           workerInfo.t_id = workerType.t_id and workerInfo.d_id = department.d_id
      group by workerInfo.w_id, w_name, department.d_id, d_name, t_name, s_year
-- 代码结束
/* ************************************************************ */
```

Trigger.sql 触发器创建：

```
/* ************************************************************ */
create trigger merit_salary on attendance
      after insert, update
as if update (a_status)
begin
      declare my_cursor cursor for select w_id, a_date, a_status from inserted;
      declare @newWid int; declare @newdate date; declare @newstatus int;
      open my_cursor;
      fetch next from my_cursor into @newWid, @newdate, @newstatus;
      while @@FETCH_STATUS = 0
            begin
                  declare @base_salary money; declare @newpayment money;
                  declare @oldpayment money;

                  select @base_salary = base_salary from workerInfo, workerType
                        where workerInfo.w_id = @newWid and workerInfo.t_id = workerType.t_id;

                  set @oldpayment = 0;
                  if exists (select payment from deleted
                                          where w_id = @newWid and a_date = @newdate)
                        select @oldpayment=payment from deleted
                              where w_id = @newWid and a_date = @newdate

                  -- 根据考勤情况设置奖惩，1 表示正常，2 表示迟到，
                  --                          3 表示缺勤，4 表示请假，5 表示早退
                  if @newstatus = 1 set @newpayment = 0
                  if @newstatus = 2 set @newpayment = 0 - @base_salary / 21.75 * 0.2
                  if @newstatus = 3 set @newpayment = 0 - @base_salary / 21.75
                  if @newstatus = 4 set @newpayment = 0
                  if @newstatus = 5 set @newpayment = 0 - @base_salary / 21.75 * 0.2
```

```
            -- 修改 payment
            update attendance
                set payment = @newpayment
                where w_id = @newWid and a_date = @newdate;

            -- 修改薪水表中的月绩效工资
            if not exists (select * from workerSalary
                where w_id = @newWid and s_year = convert(int,year(@newdate))
                    and s_month = convert(int,month(@newdate)))
                insert into workerSalary
                    values(@newWid,convert(int,year(@newdate)),
                    convert(int,month(@newdate)),@base_salary, 0 , 0);

            update workerSalary
                set merit_pay = merit_pay - @oldpayment + @newpayment
                where w_id = @newWid and s_year = convert(int,year(@newdate))
                    and s_month = convert(int,month(@newdate));

            fetch next from my_cursor into @newWid, @newdate, @newstatus;
        end
    close my_cursor;
end
go

create trigger extra_salary on extraWork
    after insert, update
as if update (type) or update (duration)
begin
    declare my_cursor cursor for select w_id, e_date, type, duration from inserted;
    declare @newWid int;declare @newdate date;
    declare @newtype int;declare @newduration int;
    open my_cursor;
    fetch next from my_cursor into @newWid, @newdate, @newtype, @newduration;
    while @@FETCH_STATUS = 0
        begin
            declare @base_salary money;
            declare @newpayment money;
            declare @oldpayment money;

            select @base_salary = base_salary from workerInfo, workerType
                where workerInfo.w_id = @newWid and workerInfo.t_id = workerType.t_id;

            set @oldpayment = 0;
            if not exists (select payment from deleted
```

```
                    where w_id = @newWid and e_date = @newdate)
                select @oldpayment=payment from deleted
                    where w_id = @newWid and e_date = @newdate;

            -- 根据加班情况设置津贴，type 表示支付津贴为几倍工资
            set @newpayment = @base_salary / 21.75 / 8 * @newtype * @newduration;

            -- 修改 payment
            update extraWork
                set payment = @newpayment
                where w_id = @newWid and e_date = @newdate;

            -- 修改薪水表中的月津贴
            if not exists (select * from workerSalary
                where w_id = @newWid and s_year = convert(int,year(@newdate))
                    and s_month = convert(int,month(@newdate)))
                insert into workerSalary
                    values(@newWid,convert(int,year(@newdate)),
                    convert(int,month(@newdate)),@base_salary, 0, 0);

            update workerSalary
                set extra_pay = extra_pay - @oldpayment + @newpayment
                where w_id = @newWid and s_year = convert(int,year(@newdate))
                    and s_month = convert(int,month(@newdate));

            fetch next from my_cursor into @newWid, @newdate, @newtype, @newduration;
        end
    close my_cursor;
end
/* ********************************************************** */
```

Data.sql 测试数据插入：

```
/* ********************************************************** */
begin transaction

-- department 数据:
insert into department values (1,N'销售部'),
    (2,N'产品部'),
    (3,N'技术部');

-- workerType 数据:
insert into workerType values (1,N'工程师',2,5000),
    (2,N'销售',1,3000),
    (3,N'产品经理',3,8000);
```

```
-- workerInfo 数据:
insert into workerInfo values (1, N'李响',N'男',N'18857802923',3,1,0),
	(2, N'张一',N'男',N'18296733300',1,1,0),
	(3, N'张二',N'女',N'15881113869',1,1,0),
	(4, N'王一',N'男',N'18865480223',2,3,0),
	(5, N'王二',N'女',N'15645780223',2,1,0);

-- dbuser 数据:
insert into dbuser values (1,N'123456',4),
	(2,N'123456',2),
	(3, N'123456',1),
	(4, N'123456',2),
	(5, N'123456',1);

-- attendance 数据:
insert into attendance values
	(1,'2021-03-01','2020-03-01 08:54:21','2020-03-01 17:23:56',1,0,0),
	(1,'2021-04-01','2020-03-01 08:54:21','2020-03-01 17:23:56',1,0,0),
	(1,'2021-05-01','2020-03-01 08:54:21','2020-03-01 17:23:56',1,0,0),
	(1,'2020-03-01','2020-03-01 08:54:21','2020-03-01 17:23:56',1,0,0),
	(1,'2020-03-02','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-03','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-04','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-05','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-08','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-09','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-10','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
	(1,'2020-03-11','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-12','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-15','2021-05-10 08:54:21','2021-05-10 17:23:56',4,0,0),
	(1,'2020-03-16','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-17','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-18','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-19','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-22','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-24','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
	(1,'2020-03-25','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-26','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-29','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-30','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-03-31','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-04-01','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
	(1,'2020-04-02','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
```

(1,'2020-04-03','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-04','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-05','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-08','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-09','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-10','2021-05-10 08:54:21','2021-05-10 17:23:56',4,0,0),
(1,'2020-04-11','2021-05-10 08:54:21','2021-05-10 17:23:56',4,0,0),
(1,'2020-04-12','2021-05-10 08:54:21','2021-05-10 17:23:56',4,0,0),
(1,'2020-04-15','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-16','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-17','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-18','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-19','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-22','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-24','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-25','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-26','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-29','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(1,'2020-04-30','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-01','2020-03-01 08:54:21','2020-03-01 17:23:56',1,0,0),
(2,'2020-03-02','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-03','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-04','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-05','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-08','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-09','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-10','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(2,'2020-03-11','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-12','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-15','2021-05-10 08:54:21','2021-05-10 17:23:56',4,0,0),
(2,'2020-03-16','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-17','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-18','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-19','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-22','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-24','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(2,'2020-03-25','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-26','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-29','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-30','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-03-31','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-01','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-02','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-03','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),

(2,'2020-04-04','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-05','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-08','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-09','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-10','2021-05-10 08:54:21','2021-05-10 17:23:56',5,0,0),
(2,'2020-04-11','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-12','2021-05-10 08:54:21','2021-05-10 17:23:56',4,0,0),
(2,'2020-04-15','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-16','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(2,'2020-04-17','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-18','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-19','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(2,'2020-04-22','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-24','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(2,'2020-04-25','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-26','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(2,'2020-04-29','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(2,'2020-04-30','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(3,'2020-04-01','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(3,'2020-04-02','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(3,'2020-04-03','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(3,'2020-04-04','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(3,'2020-04-05','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(3,'2020-04-08','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(3,'2020-04-09','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(3,'2020-04-10','2021-05-10 08:54:21','2021-05-10 17:23:56',5,0,0),
(3,'2020-04-11','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(3,'2020-04-12','2021-05-10 08:54:21','2021-05-10 17:23:56',4,0,0),
(3,'2020-04-15','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(3,'2020-04-16','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(3,'2020-04-17','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(3,'2020-04-18','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(3,'2020-04-19','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(3,'2020-04-22','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(3,'2020-04-24','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(3,'2020-04-25','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(3,'2020-04-26','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(3,'2020-04-29','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(3,'2020-04-30','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(4,'2020-04-01','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-02','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-03','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-04','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-05','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),

```
(4,'2020-04-08','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-09','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-10','2021-05-10 08:54:21','2021-05-10 17:23:56',5,0,0),
(4,'2020-04-11','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-12','2021-05-10 08:54:21','2021-05-10 17:23:56',4,0,0),
(4,'2020-04-15','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-16','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(4,'2020-04-17','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-18','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-19','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(4,'2020-04-22','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-24','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(4,'2020-04-25','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-26','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(4,'2020-04-29','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(4,'2020-04-30','2021-05-10 08:54:21','2021-05-10 17:23:56',3,0,0),
(5,'2020-03-01','2020-03-01 08:54:21','2020-03-01 17:23:56',1,0,0),
(5,'2020-03-02','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-03','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-04','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-05','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-08','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-09','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-10','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(5,'2020-03-11','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-12','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-15','2021-05-10 08:54:21','2021-05-10 17:23:56',4,0,0),
(5,'2020-03-16','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-17','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-18','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-19','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-22','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-24','2021-05-10 08:54:21','2021-05-10 17:23:56',2,0,0),
(5,'2020-03-25','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-26','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-29','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-30','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0),
(5,'2020-03-31','2021-05-10 08:54:21','2021-05-10 17:23:56',1,0,0);

-- extraWork 数据:
insert into extraWork values
(1,'2020-03-04','2021-03-10 19:05:21','2021-03-10 22:12:56',3,2,0,0),
(1,'2020-03-05','2021-03-10 19:05:21','2021-03-10 22:12:56',1,2,0,0),
(1,'2020-03-07','2021-03-10 19:05:21','2021-03-10 22:12:56',3,2,0,0),
```

```
(1,'2020-03-11','2021-03-10 19:05:21','2021-03-10 22:12:56',2,2,0,0),
(1,'2020-03-13','2021-03-10 19:05:21','2021-03-10 22:12:56',3,2,0,0),
(1,'2020-03-23','2021-03-10 19:05:21','2021-03-10 22:12:56',4,2,0,0),
(1,'2020-03-24','2021-03-10 19:05:21','2021-03-10 22:12:56',3,2,0,0),
(1,'2020-04-04','2021-03-10 19:05:21','2021-03-10 22:12:56',3,2,0,0),
(1,'2020-04-05','2021-03-10 19:05:21','2021-03-10 22:12:56',1,2,0,0),
(1,'2020-04-07','2021-03-10 19:05:21','2021-03-10 22:12:56',3,2,0,0),
(1,'2020-04-11','2021-03-10 19:05:21','2021-03-10 22:12:56',2,2,0,0),
(1,'2020-04-13','2021-03-10 19:05:21','2021-03-10 22:12:56',3,2,0,0),
(1,'2020-04-23','2021-03-10 19:05:21','2021-03-10 22:12:56',4,2,0,0),
(1,'2020-04-24','2021-03-10 19:05:21','2021-03-10 22:12:56',3,2,0,0),
(2,'2020-03-10','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(2,'2020-03-11','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(2,'2020-03-15','2021-05-10 09:05:21','2021-05-10 16:12:56',7,2,0,0),
(2,'2020-03-16','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(2,'2020-03-21','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(2,'2020-03-25','2021-05-10 09:05:21','2021-05-10 16:12:56',5,3,0,0),
(3,'2020-03-10','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(3,'2020-03-11','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(3,'2020-03-15','2021-05-10 09:05:21','2021-05-10 16:12:56',7,2,0,0),
(3,'2020-03-16','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(3,'2020-03-21','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(3,'2020-03-25','2021-05-10 09:05:21','2021-05-10 16:12:56',5,3,0,0),
(4,'2020-03-10','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(4,'2020-03-11','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(4,'2020-03-15','2021-05-10 09:05:21','2021-05-10 16:12:56',7,2,0,0),
(4,'2020-03-16','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(4,'2020-03-21','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(4,'2020-03-25','2021-05-10 09:05:21','2021-05-10 16:12:56',5,3,0,0),
(5,'2020-03-10','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(5,'2020-03-11','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(5,'2020-03-15','2021-05-10 09:05:21','2021-05-10 16:12:56',7,2,0,0),
(5,'2020-03-16','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(5,'2020-03-21','2021-05-10 19:05:21','2021-05-10 22:12:56',3,2,0,0),
(5,'2020-03-25','2021-05-10 09:05:21','2021-05-10 16:12:56',5,3,0,0);
commit
/* ********************************************************** */
```