

**数据库系统原理实践报告**

|  |  |
| --- | --- |
| 项目名称： | 机票预订和管理系统 |
| 姓 名： | 彭子晨 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1804 |
| 学 号： | U201814755 |
| 指导教师： | 袁平鹏 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2021 年 6 月 30 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 1](#_Toc75997182)

[2 软件功能学习部分 2](#_Toc75997183)

[2.1 任务要求 2](#_Toc75997184)

[2.2 完成过程 2](#_Toc75997185)

[2.3任务总结 5](#_Toc75997186)

[3 SQL练习部分 6](#_Toc75997187)

[3.1 任务要求 6](#_Toc75997188)

[3.2 完成过程 6](#_Toc75997189)

[3.3任务总结 28](#_Toc75997190)

[4 综合实践任务 30](#_Toc75997191)

[4.1系统设计目标 30](#_Toc75997192)

[4.2 需求分析 30](#_Toc75997193)

[4.3 总体设计 33](#_Toc75997194)

[4.4 数据库设计 36](#_Toc75997195)

[4.5 详细设计与实现 42](#_Toc75997196)

[4.6 系统测试 49](#_Toc75997197)

[4.7 系统设计与实现总结 55](#_Toc75997198)

[5 课程总结 56](#_Toc75997199)

[附录 57](#_Toc75997200)

[SQL练习部分代码 57](#_Toc75997201)

[综合实现任务核心代码 66](#_Toc75997202)

# 1 课程任务概述

在课设规定的时间内，撰写并完成课程实践报告。

实践报告由5章组成，依次对应下列内容：

**1 课程任务概述**

简要陈述介绍本实践课程的各项任务要求。

**2 软件功能学习**

阐述第1部分任务的完成过程。

**3 SQL练习**

阐述第2部分的完成过程。

**4 应用系统设计**

阐述第3部分的完成过程。

**5 课程总结**

逐条概括、总结此次课程实践的主要工作，阐述此次课程实践的心得体会，展望此次课程实践的有待改进和完善的工作。

# 软件功能学习部分

## 2.1 任务要求

完成下列1~2题。

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

实验环境如下：

操作系统：Windows 10

数据库：MySQL

图形化界面：Navicat for MySQL --版本15.0.25

### 2.2.1备份练习

（1）导入数据

首先将老师准备的excel数据源导入到数据库中。

第一步：在Navicat中选择“文件”>“导入”>选择对应文件夹中的数据源文件。如图 2‑1所示。

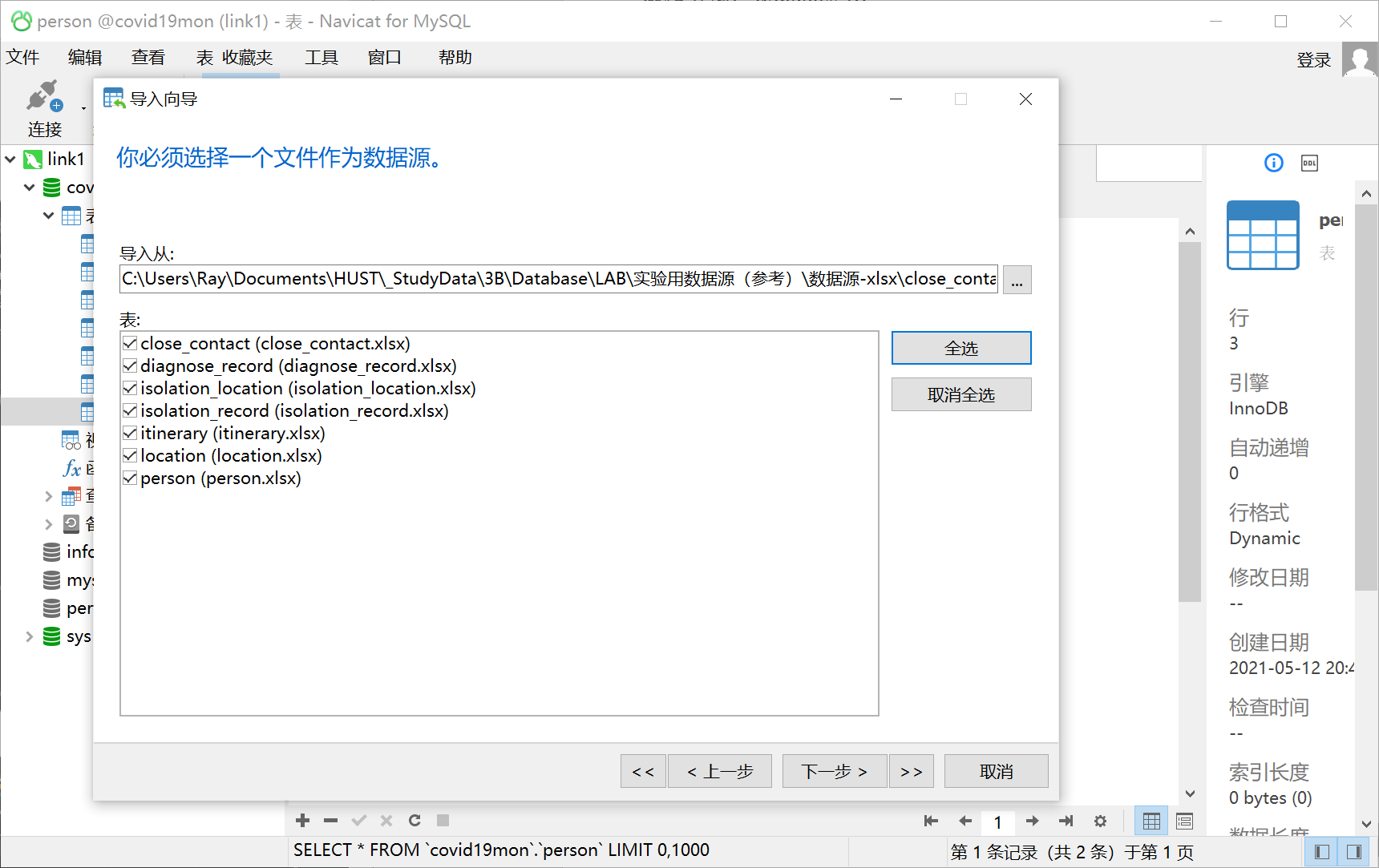


图 2‑1 选择数据源

选择现有的表或者新建表。由于这里我是在做完建表练习后做的备份实验，所以不勾选新建表，直接选择已有的表进行数据导入。如图 2‑2所示。

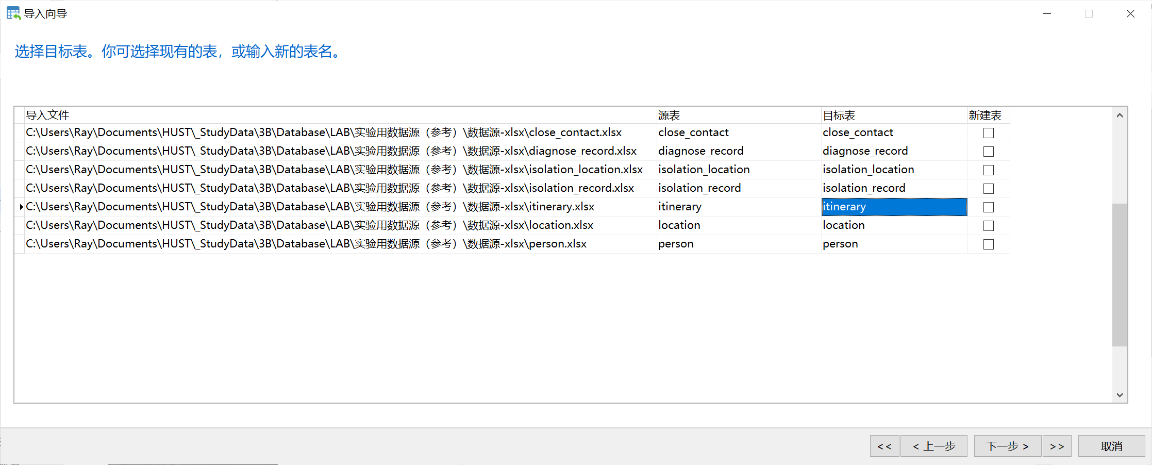


图 2‑2 选择表和新建表

数据导入成功后如图 2‑3所示。

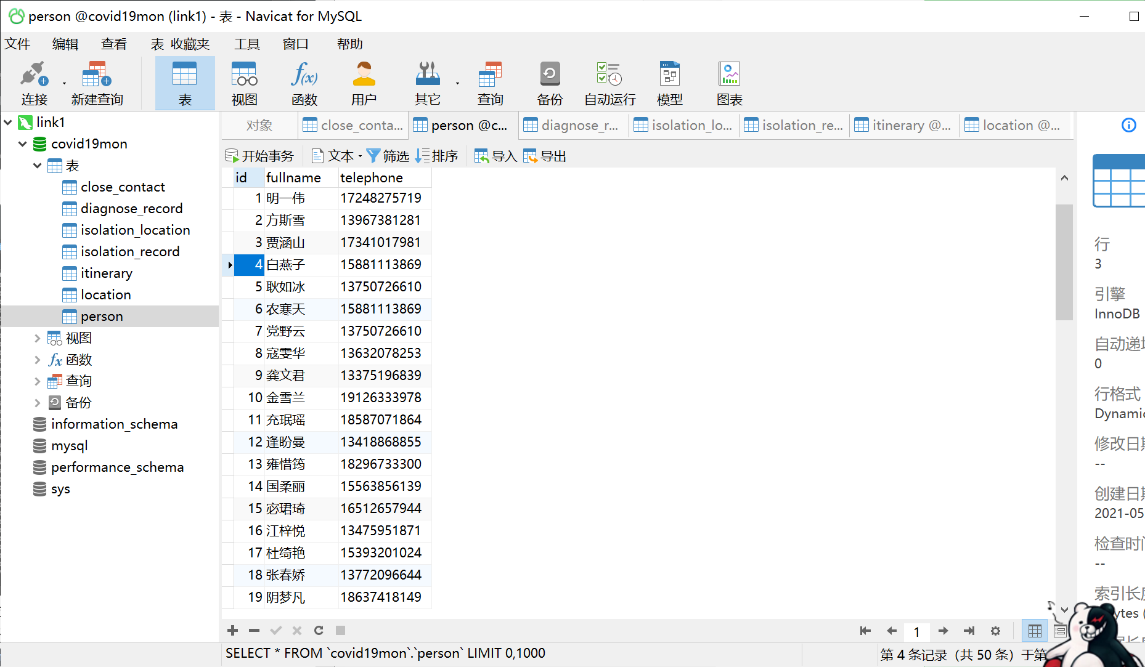


图 2‑3 导入成功

（2）数据备份

Navicat为数据备份提供了图形化操作按键。点击“备份”>“新建备份”>输入注释>“备份”,即可完成数据备份。如图 2‑4所示。

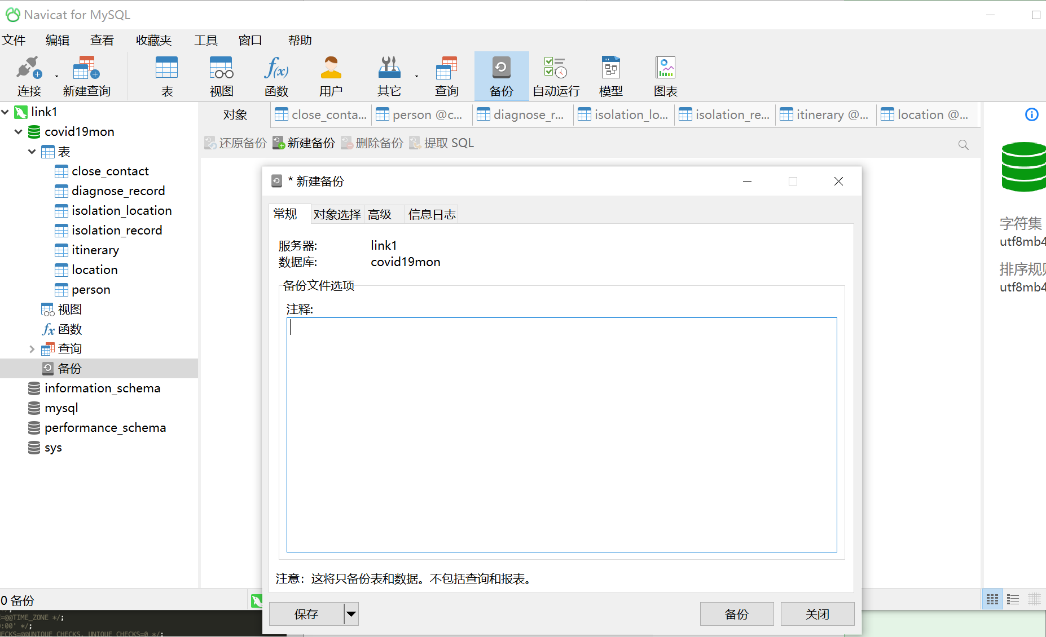


图 2‑4 建立备份

建立备份后，选择备份后选择“还原备份”即可进行备份的还原，还可以进行提取SQL代码等操作。如图 2‑5所示。

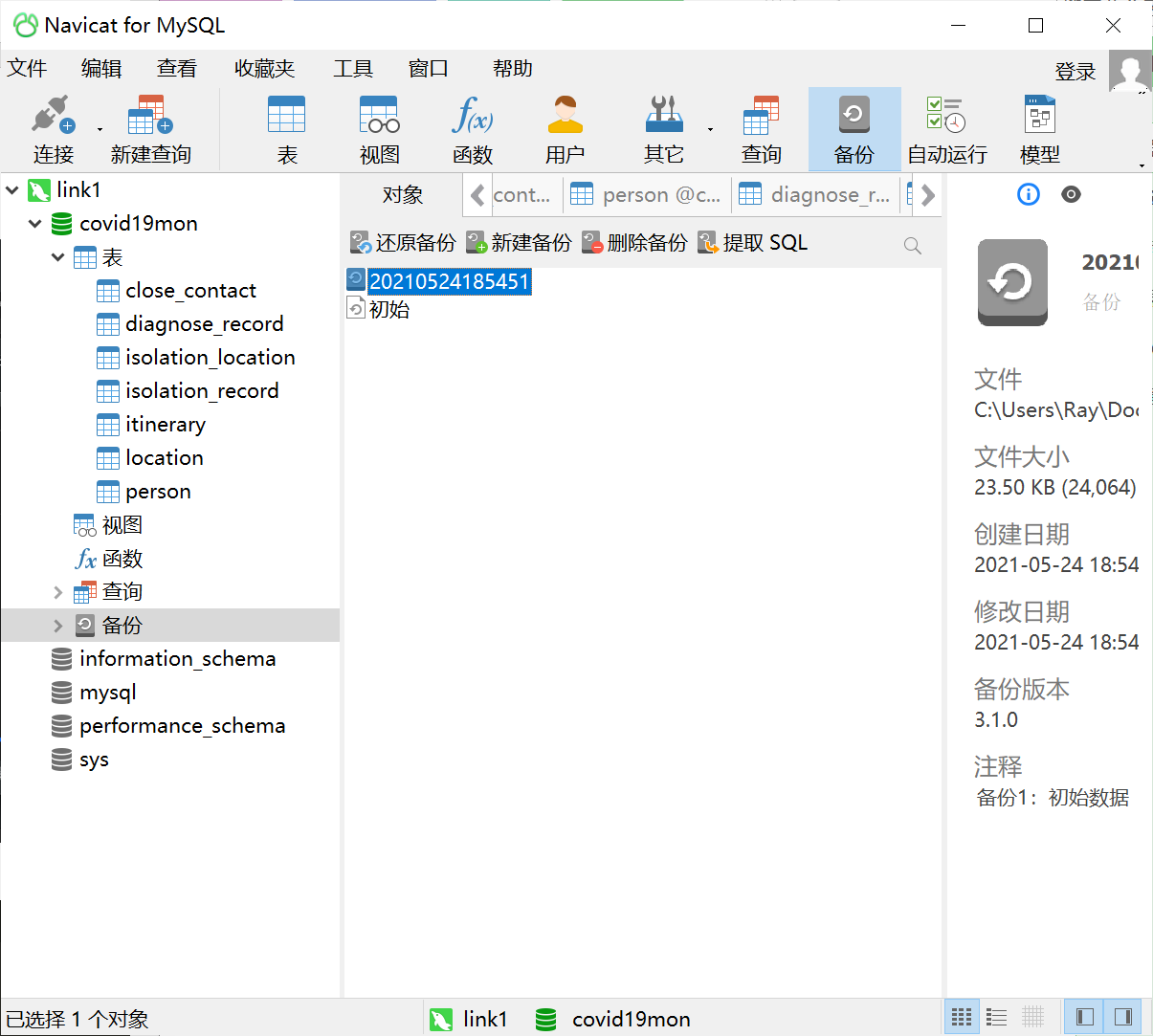


图 2‑5 还原备份

### 2.2.2权限配置

在Navicat的“用户”>“新建用户”可以新增数据库用户并授予相关权限。自定义的内容包括但不限于：

* 用户名
* 主机
* 密码
* 加密方式
* 授予权限
* 设置成员归属
* 自动生成SQL代码
* …

界面如图 2‑6所示。

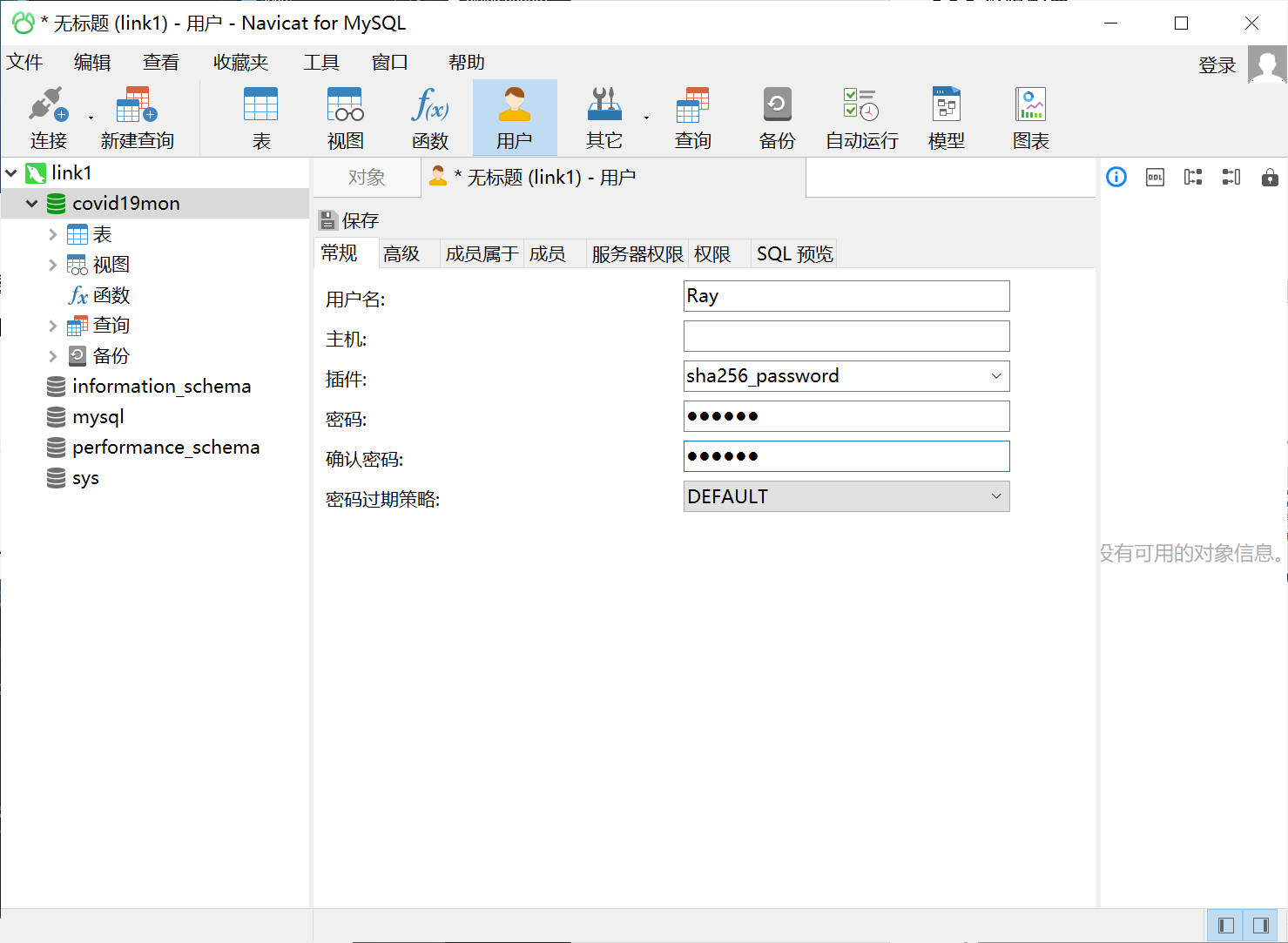


图 2‑6 用户设置

## 2.3任务总结

第一个任务十分简单，主要就是数据库平台的选择和安装，以及一些基础功能的了解学习。由于MySQL整体更轻便，而且可以使用Navicat等软件进行数据库图形化的支持，数据备份、用户设置、权限配置等一些功能都集成地十分良好，所以这次实验选择了MySQL。

就整体的体验而言，Navicat的界面和功能都很直观易懂，唯一的问题是只有14天的试用期，可能之后做第三个实验会更换软件，不过这也对整体的学习影响不大。

# SQL练习部分

## 3.1 任务要求

假设在某个区域内的所有地点都存储在地点表中。该地区中的所有人员信息存储在人员表中。根据收集到的人员行程建立了疫情期间某个地区的人员行程表。

该地区的部分人员进行了核酸检测，检测结果保存在诊断表中。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中，隔离信息存储到隔离表中。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中。

## 3.2 完成过程

### 3.2.1 建表

1. **建表**

根据以上背景，请在DBMS中创建这些关系，包括主码和外码的说明，并写出指定关系的建表SQL语句。建表的语句不是特别难，主要区分好主键、外键，以及非空之类的特殊条件。由于实际写SQL语句时并不是一遍过，所以在反复测试过程中每次都要记得Drop之前建立的表，以防出错。

建表内容已通过了educoder平台的测试，具体语句如下所示：

|  |
| --- |
| create database if not exists covid19mon;  use covid19mon;  -- 请在以下适当的空白位置填写SQL语句完成任务书的要求。空白位置不够的话，可以通过回车换行增加。  -- 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));  -- 代码结束  /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/ |

1. **观察性实验**

通过观察上述表格可知：在设置外码时，外码一定要是被参照表的主码。当外键不是参照表的主键时会报错。报错信息如图 3‑1所示。

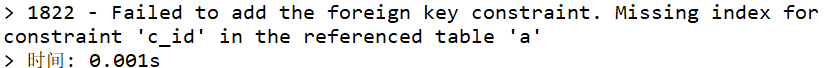


图 3‑1 非主键报错

1. **数据准备**

依据后续实验的要求，向上述表格中录入适当数量的实验数据，从而对相关的实验任务能够起到验证的作用。通过数据包导入实验数据，如图 3‑2所示。

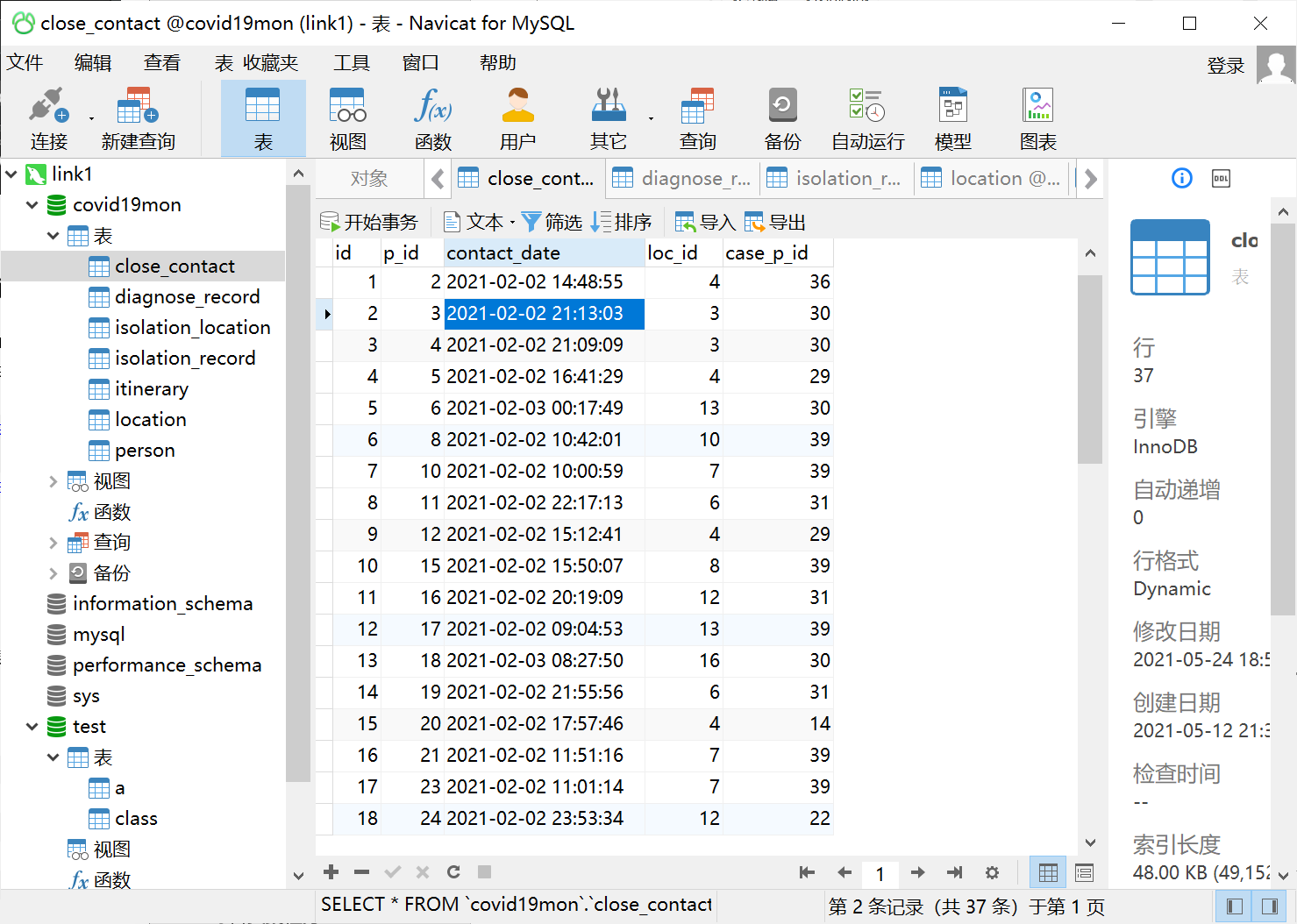


图 3‑2 导入数据

### 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；**

使用INSERT、DELETE、UPDATE指令进行增、删、改操作。

|  |
| --- |
| -- 插入三条数据  INSERT INTO person(id,fullname,telephone)  VALUES(1,"张小敏","13907110001"),(2,"李大锤","18907110002"),(3,"孙二娘","13307100003");  -- 删除2  DELETE FROM person WHERE id = 2  -- 更改电话  UPDATE person SET telephone = "13607176668" WHERE id = 1 |

1. **批处理操作**

将行程表中所有到达地点2的记录插入到新表location\_record\_2中。

|  |
| --- |
| create table location\_record\_2 AS(  SELECT \* FROM itinerary  WHERE loc\_id = 2  ); |

结果如图 3‑3所示：

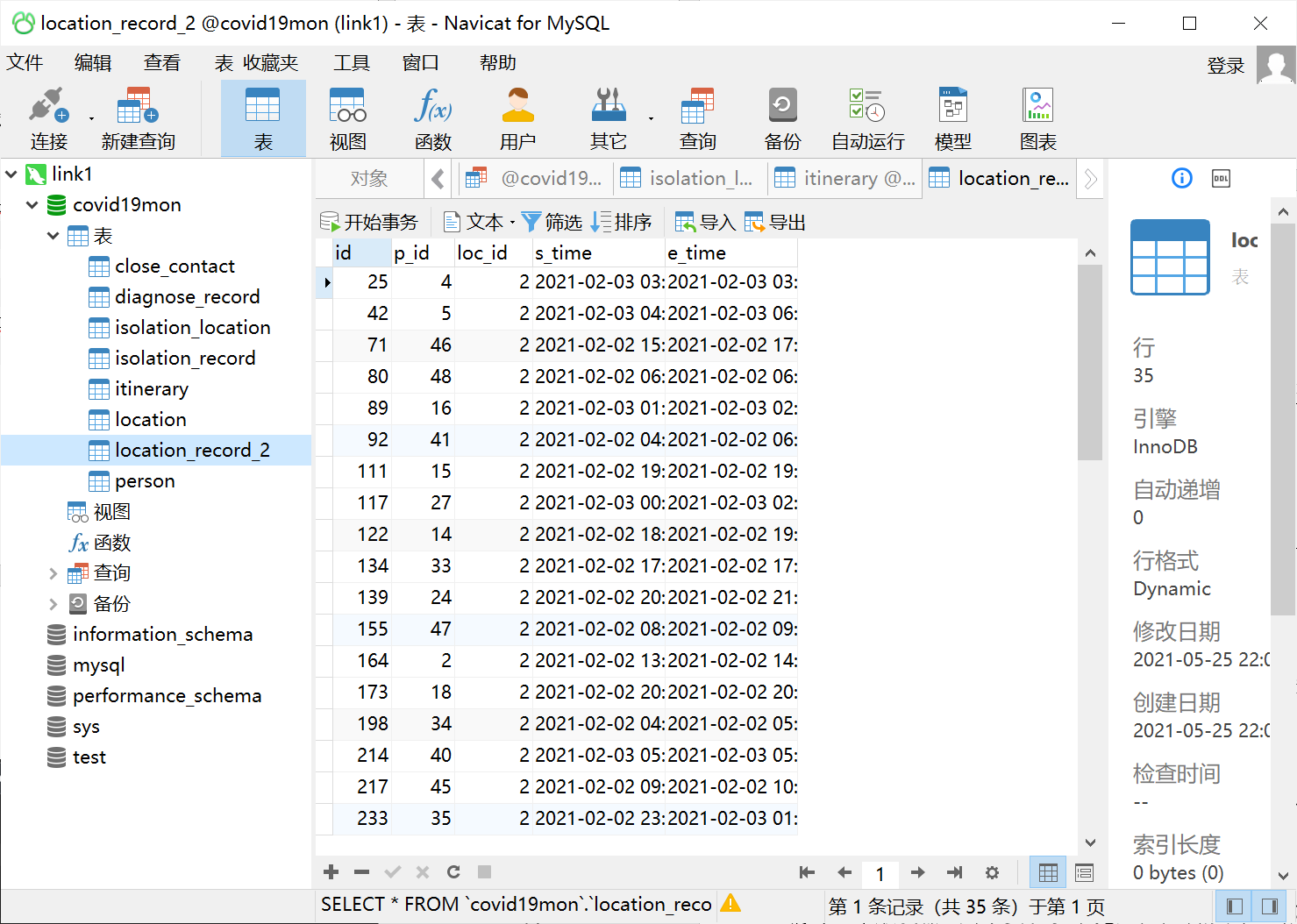


图 3‑3 新表

1. **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。实验1中已经进行过导入操作，所以这里仅测试导出操作。

选择“文件”>“导出”>“全部记录”>选择导出文件类型>进行一些自定义设置>“开始”。即可导出文件。步骤如图 3‑4所示。

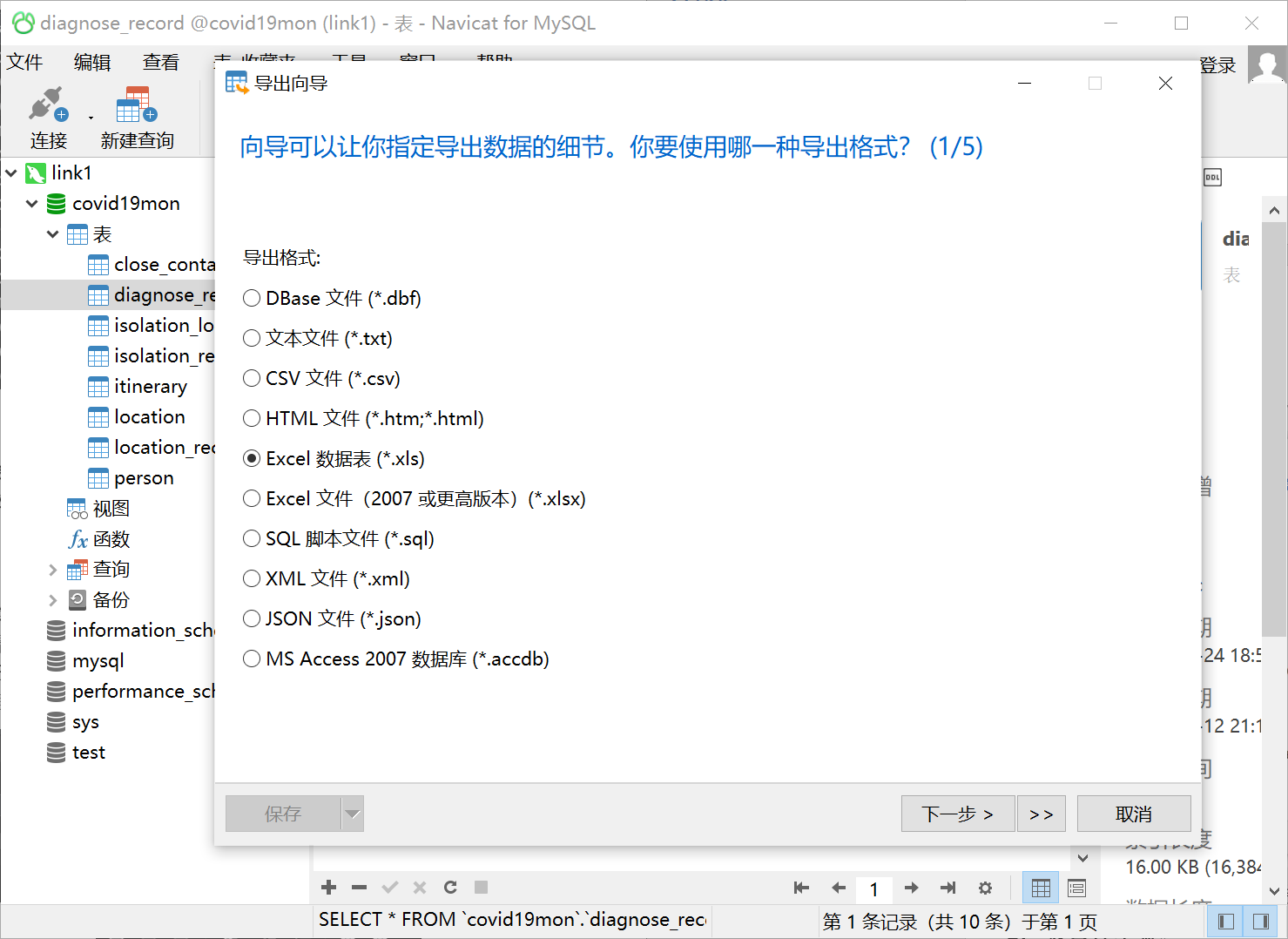


图 3‑4 导出文件

导出后的Excel文件如图 3‑5所示。



图 3‑5 导出后的Excel文件

1. **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  CREATE table nomainkey(  id INT,  str CHAR(10)  )  -- 插入数据  INSERT INTO nomainkey(id,str)  VALUES (1,"aaa"),(1,"aaa"),(1,"aaa"),(2,"bb"),(2,"bb"),(3,"c");  -- 删除数据  DELETE \* FROM nomainkey WHERE id = 1;  -- 修改数据  UPDATE nomainkey SET str = "abcd" WHERE id = 2; |

插入数据不会报错，插入后的表内数据如图 3‑6所示。

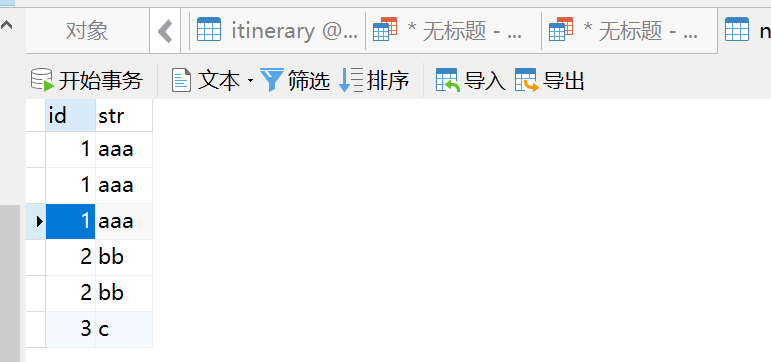


图 3‑6 插入重复数据

使用DELETE语句删除重复的(1,”aaa”)数据，会报错，如图 3‑6所示。

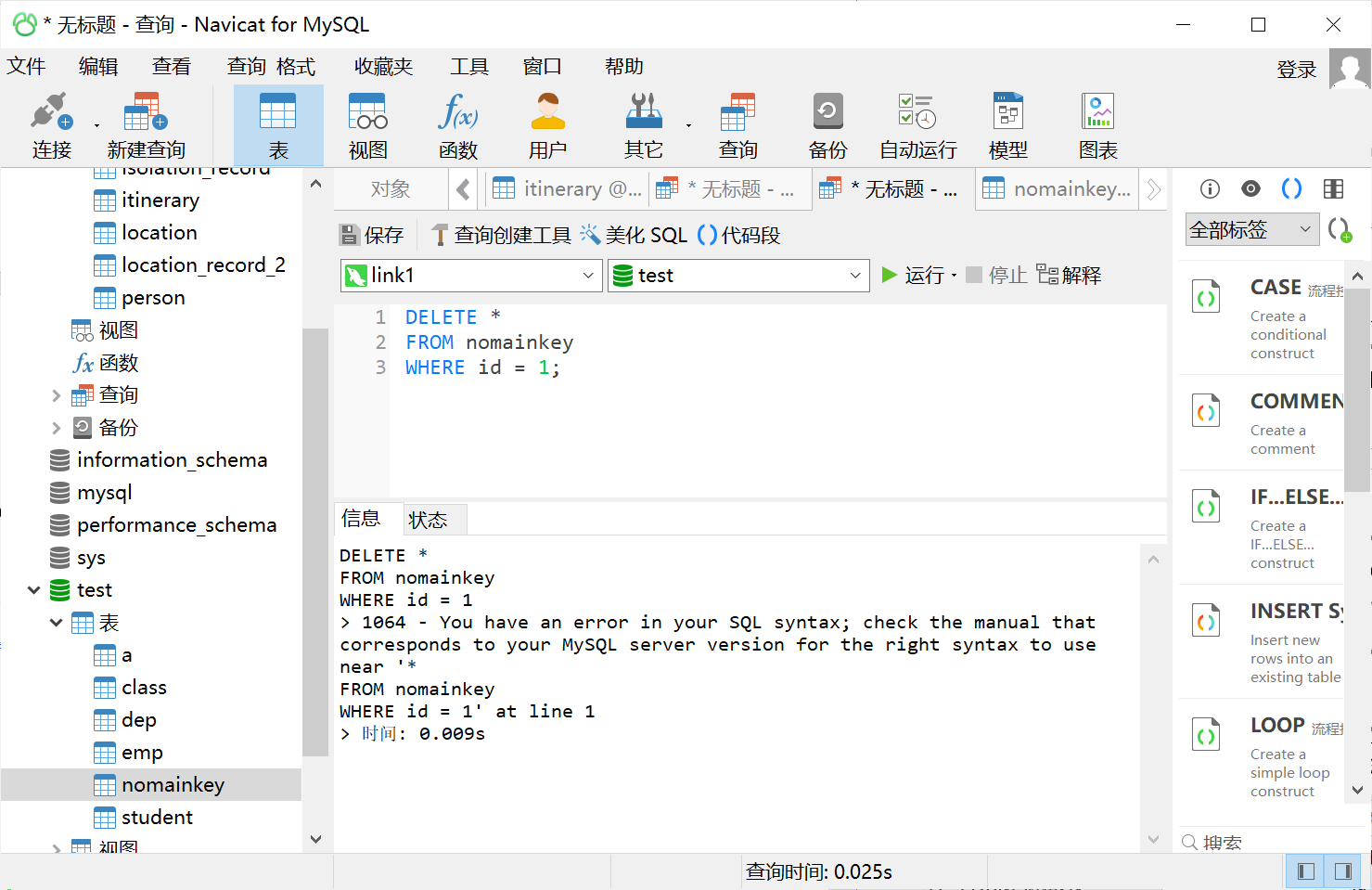


图 3‑7 删除重复语句报错

使用UPDATE修改数据可以成功执行，执行的结果如图 3‑8所示。

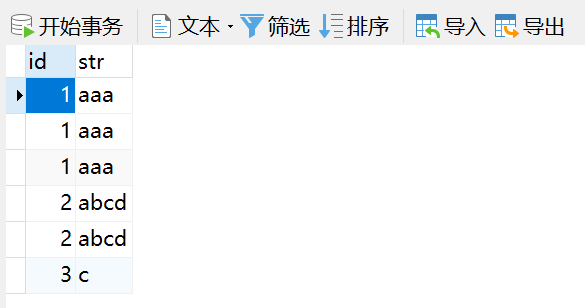


图 3‑8 修改语句成功执行

1. **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：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”。

编写触发器要注意MySQL一次只能定义一个事件驱动的触发器。这个实验就需要同时触发update和insert，所以需要写两个触发器。而且MySQL需要先用“DELIMITER 界符”语句指定触发器定义语句的界符。这个实验中是使用了“;;”(即两个连续的分号)作触发器定义语句的界符，并且在触发器定义结束后重新指定语句的界符，即恢复为单个分号。详细SQL语句如下：

|  |
| --- |
| -- 触发器  DELIMITER ;;  CREATE TRIGGER change\_isolation\_state\_insert AFTER INSERT  ON diagnose\_record FOR EACH ROW  BEGIN  IF NEW.result = 1 THEN  UPDATE isolation\_record SET state = 3    WHERE NEW.p\_id = p\_id;  END IF;  END;;  DELIMITER ;  DELIMITER ;;  CREATE TRIGGER change\_isolation\_state\_update AFTER UPDATE  ON diagnose\_record FOR EACH ROW  BEGIN  IF NEW.result = 1 THEN  UPDATE isolation\_record SET state = 3    WHERE NEW.p\_id = p\_id;  END IF;  END;;  DELIMITER ; |

### 3.2.3 查询

分别用一条SQL语句完成下列各个小题的需求：

1. 查询截至目前的人流量大于30的地点，并按照从高到低排序。

主要利用GROUP BY在行程表中根据地点序号loc\_id分类，通过COUNT获得每个地点的人流量，使用HAVING筛选出大于30的地点，最后排序显示。SQL语句如下，结果图 3‑9如所示。

|  |
| --- |
| SELECT location\_name,COUNT(loc\_id) 'visitors'  FROM location,itinerary  WHERE location.id = loc\_id  GROUP BY loc\_id  HAVING COUNT(\*)>30  ORDER BY visitors DESC,location\_name ASC; |

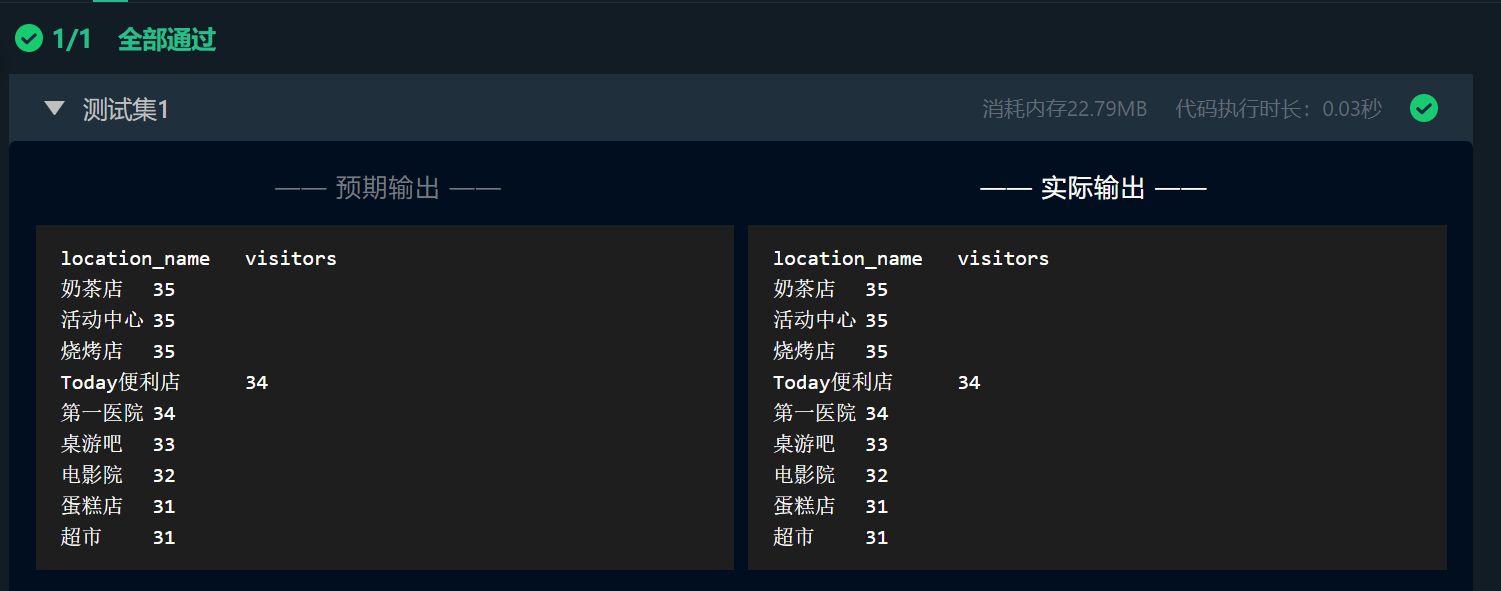


图 3‑9 查询任务1成功

1. 查询出每个隔离地中正在进行隔离的人数，并按数量由多到少排序。

对隔离地、隔离记录两表进行查询，判断state=1的人，使用GROUP BY按照隔离地id进行分组，对分组重命名为“number”，最后使用ORDER BY根据人数（number）进行排序显示。SQL语句如下，结果如图 3‑10所示。

|  |
| --- |
| SELECT location\_name, COUNT(isol\_loc\_id) 'number'  FROM isolation\_record,isolation\_location  WHERE isolation\_location.id = isol\_loc\_id AND state = 1  GROUP BY isol\_loc\_id  ORDER BY number DESC, location\_name ASC; |



图 3‑10 查询任务2成功

1. 查询行程表中连续行程中的一对行程地点（例如，方斯雪，在2021-02-02 03:02:12至2021-02-02 03:51:12 在博物馆中，2021-02-02 03:51:12 至 2021-02-02 04:23:12出现超市中，输出【姓名，重合时间，起始地点id，起始地点，结束地点id，结束地点】，查询结果如下图所示）**考虑行程相邻，即下一行程的开始时间与前一行程的结束时间无别的行程。**

通过建立行程表a,b,地点表c,d分别筛选两个行程，通过题目可知判断条件有1.a行程的结束时间=b行程的起始时间；2.a,b行程是同一个人；再加上地点表获取地点名字即可。SQL语句如下，结果如图 3‑11所示。

|  |
| --- |
| SELECT person.id,fullname,telephone,a.e\_time 'reclosing\_time',a.loc\_id 'loc1',c.location\_name 'address1',b.loc\_id 'loc2',d.location\_name 'address2'  FROM person,itinerary a,itinerary b,location c,location d  WHERE a.e\_time = b.s\_time AND a.p\_id = b.p\_id AND a.p\_id = person.id      AND a.loc\_id = c.id AND b.loc\_id = d.id  ORDER BY person.id,reclosing\_time; |



图 3‑11 查询任务3成功

1. 查询贾涵山的行程情况。

本来这个任务很简单，只需要通过人物表、行程表两表自然联系然后判断一下姓名即可。但是由于Educoder平台任务有特殊要求（在没有一条行程的情况下也要输出姓名、电话、NULL），所以使用LEFT JOIN可以保证没有行程的情况。SQL语句如下，结果如图 3‑12所示。

|  |
| --- |
| SELECT fullname,telephone,location\_name,s\_time,e\_time  FROM person LEFT JOIN  (SELECT location\_name, p\_id, s\_time, e\_time   FROM location, itinerary   WHERE loc\_id = location.id   ) a  ON (person.id = a.p\_id)  WHERE (person.fullname = '贾涵山')  ORDER BY person.id DESC, s\_time; |

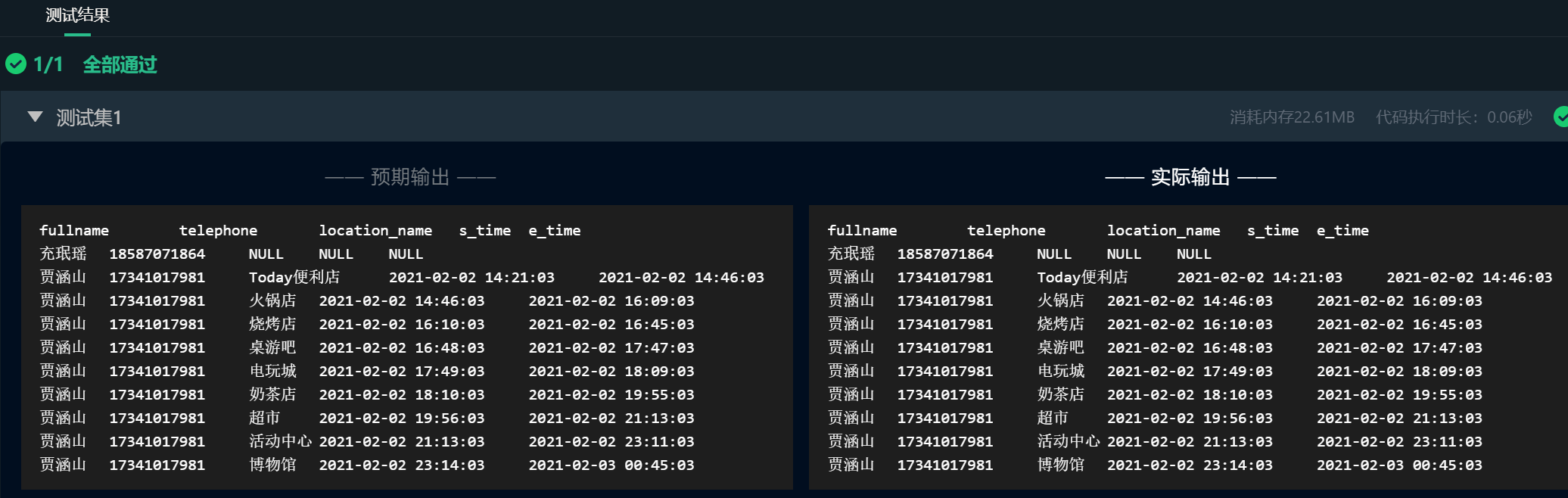


图 3‑12 查询任务4成功

1. 查询地名中带有‘店’字的地点名称。

通过LIKE判断带有‘店’字的地点名称。SQL语句如下，结果如图 3‑13所示。

|  |
| --- |
| SELECT id,location\_name  FROM location  WHERE location\_name LIKE '%店%'  ORDER BY id; |



图 3‑13 查询任务5成功

1. 新发现一位确诊者，已知他在2021.2.2日20:05:40到21:25:40之间在“活动中心”，查询他接触到的人员的名字

这道题主要需要判断时间的交集，所以条件上只要确认接触人到达地点的时间在确诊者离开时间之前，且接触人离开时间在确诊者到达时间之后，即可确定两个时间有交集。SQL语句如下，结果如图 3‑14所示。

|  |
| --- |
| SELECT fullname  FROM person, itinerary, location  WHERE person.id = itinerary.p\_id AND itinerary.loc\_id = location.id      AND location\_name = '活动中心'      AND (itinerary.s\_time < '2021-02-02 21:25:40')  AND (itinerary.e\_time > '2021-02-02 20:05:40')  ORDER BY fullname; |



图 3‑14 查询任务6成功

1. 查询正在使用的隔离区名（使用DISTINCT关键字）

通过隔离记录表判断state=1的记录，使用GROUP BY根据隔离地点序号分组，在隔离地点表中得到正在使用的隔离区名。SQL语句如下，结果如图 3‑15所示。

|  |
| --- |
| SELECT DISTINC location\_name  FROM isolation\_location, isolation\_record  WHERE isol\_loc\_id = isolation\_location.id      AND isolation\_record.state = 1  GROUP BY isol\_loc\_id  ORDER BY isol\_loc\_id; |



图 3‑15 查询任务7成功

1. 查询有出行记录的人员（使用EXISTS关键字）

使用EXISTS关键字在行程表中寻找人员表中的id，在行程表中的p\_id对应的人员就是有出行记录的人员。SQL语句如下，结果如图 3‑16所示。

|  |
| --- |
| SELECT fullname,telephone  FROM person  WHERE EXISTS      ( SELECT p\_id          FROM itinerary          WHERE p\_id = person.id          ORDER BY p\_id) |

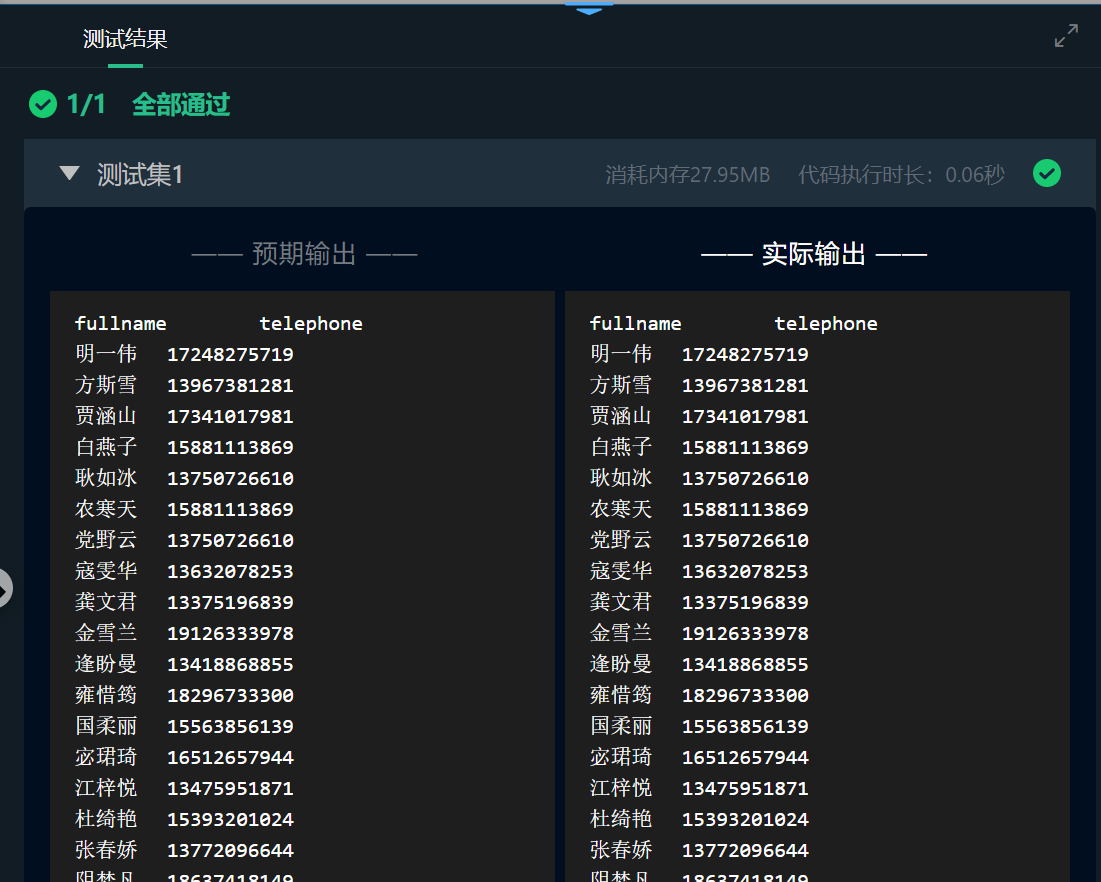


图 3‑16 查询任务8成功

1. 查询没有达到地点“Today便利店”的人数（使用NOT EXISTS 关键字）

使用NOT EXISTS关键字查询行程表、地点表的自然联合中，去“Today便利店”的行程中，人员表不包含的人员ID，使用COUNT计算人数。SQL语句如下，结果如图 3‑17所示。

|  |
| --- |
| SELECT COUNT(id) number  FROM person  WHERE NOT EXISTS   (SELECT \*    FROM itinerary,location    WHERE p\_id = person.id AND itinerary.loc\_id = location.id AND location\_name = 'Today便利店'); |

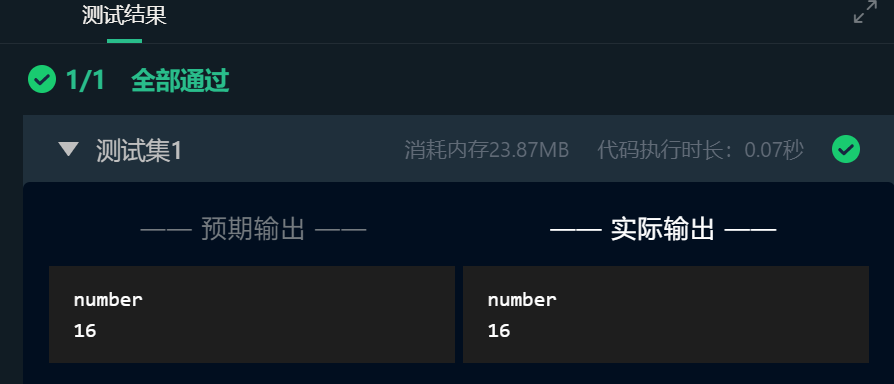


图 3‑17 查询任务9成功

1. 查询去过所有地点的人员

使用两个NOT EXISTS完成任务，将题目重新解释为一个双重否定的形式： “不存在这样的人员，有未去过的地点”。所以通过两层NOT EXISTS嵌套完成了“不存在这样的人员，有这样的地点”以及“不存在这样的地点，在行程表中出现过”。SQL语句如下，结果如图 3‑18所示。

|  |
| --- |
| 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; |

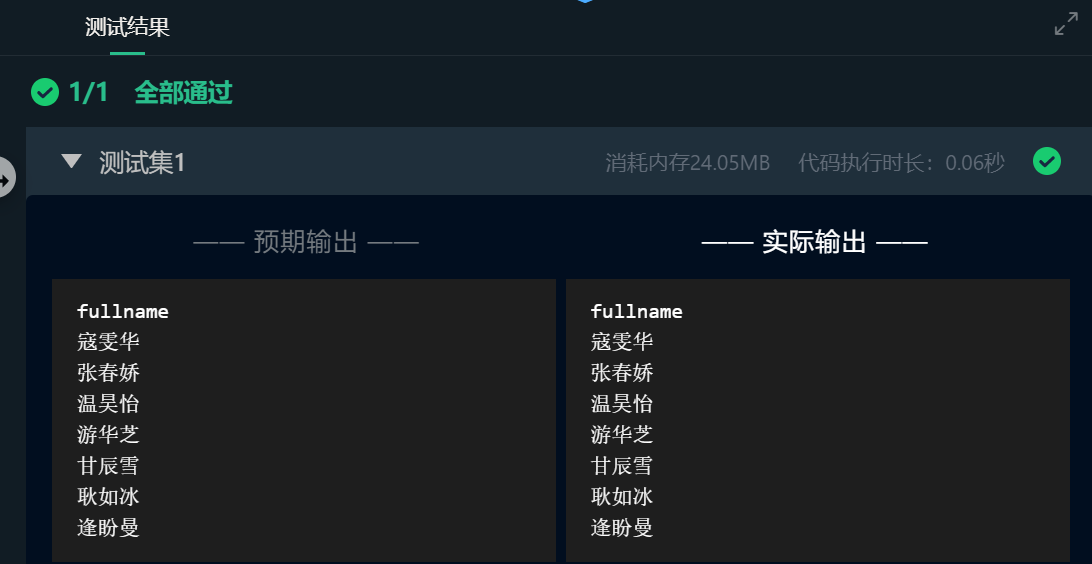


图 3‑18 查询任务10完成

1. 新建一个视图，里面有每个隔离点的id,名称，已用房间，容量。

新建视图，从隔离记录表中找到正在隔离的人员，其中已占用房间通过GROUP BY地点序号分组后，COUNT计算得到。SQL语句如下，结果如图 3‑19所示。

|  |
| --- |
| CREATE VIEW isolation\_location\_status (id, location\_name, capacity, occupied)  AS  SELECT isolation\_location.id,location\_name,capacity,COUNT(case when state = 1 then 1 else null end) 'occupied'  FROM isolation\_location LEFT JOIN  (SELECT \*   FROM isolation\_record   WHERE state = 1) a  ON (isolation\_location.id = a.isol\_loc\_id)  GROUP BY isolation\_location.id; |

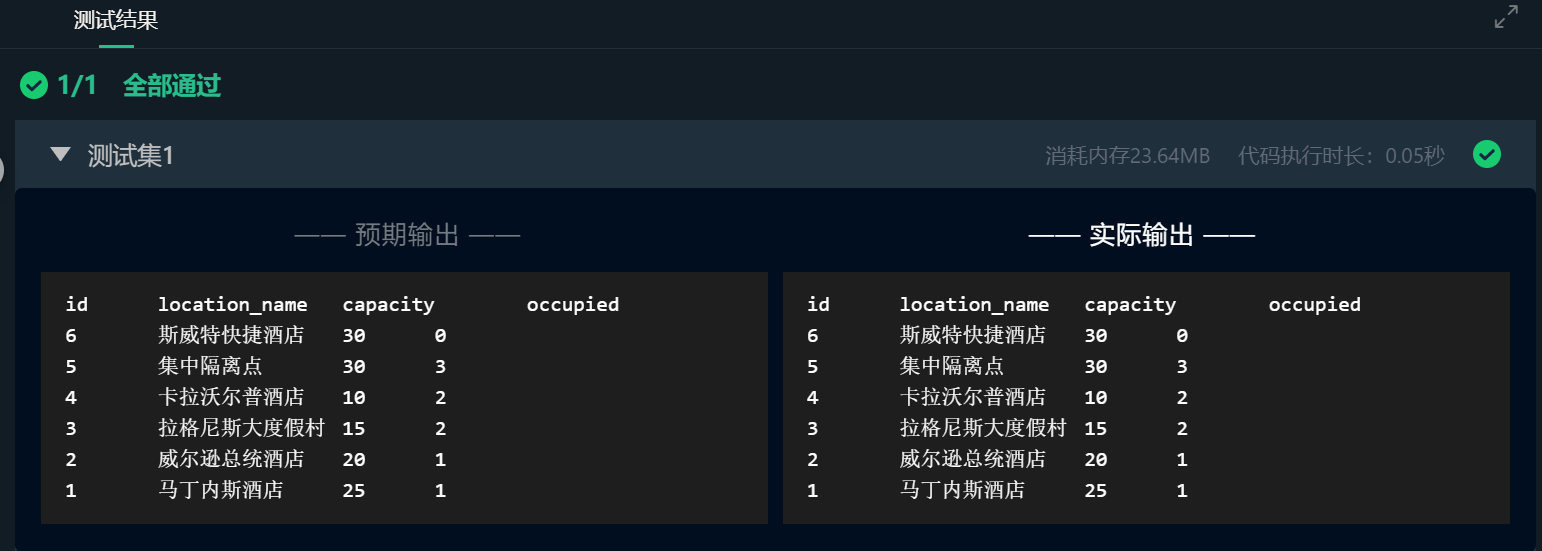


图 3‑19 查询任务11成功

1. 从视图中查询还有空房间的隔离点的剩余房间数目。

从视图中SELECT对应列，记得重命名剩余房间数目列。SQL语句如下，结果如图 3‑20所示。

|  |
| --- |
| SELECT location\_name, capacity-occupied 'available\_rooms'  FROM isolation\_location\_status  ORDER BY id; |



图 3‑20 查询任务12成功

1. 查询靳宛儿传染（接触）的人有哪些。

判断每一条行程的地点序号与感染者的行程地点相同，且确保接触者的到达时间在感染者离开时间之前、接触者的离开时间在感染者到达时间之后。SQL语句如下，结果如图 3‑21所示。

|  |
| --- |
| SELECT fullname, telephone  FROM itinerary a, person  WHERE person.id = a.p\_id      AND a.loc\_id IN (SELECT loc\_id FROM itinerary b, person c WHERE c.fullname = '靳宛儿' AND c.id = b.p\_id)      AND a.s\_time < (SELECT e\_time FROM person p, itinerary i WHERE p.fullname = '靳宛儿' AND a.loc\_id=i.loc\_id AND i.p\_id = p.id)      AND a.e\_time > (SELECT s\_time FROM person p, itinerary i WHERE p.fullname = '靳宛儿' AND a.loc\_id=i.loc\_id AND i.p\_id = p.id)      AND person.fullname!='靳宛儿'  ORDER BY fullname; |

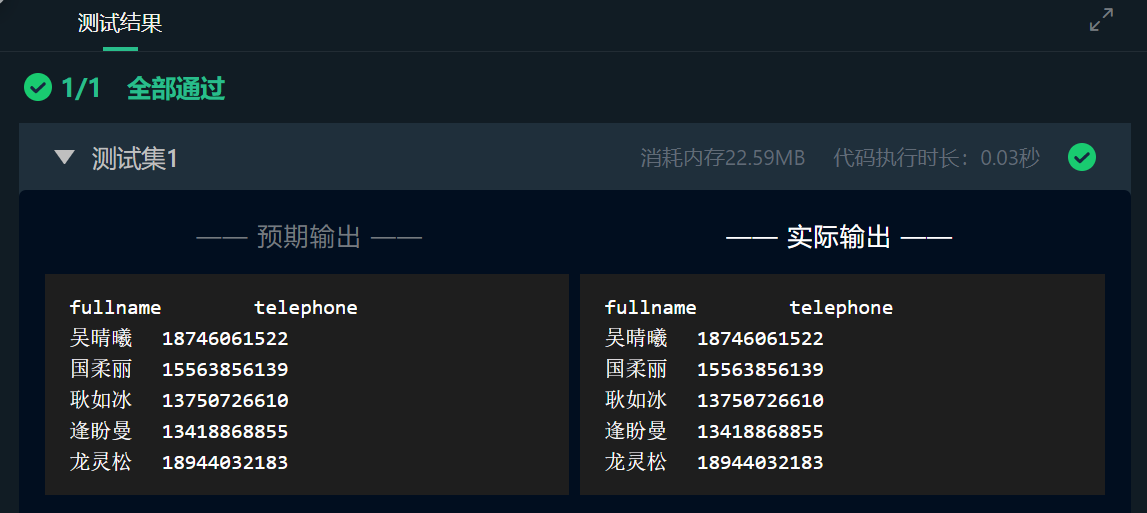


图 3‑21 查询任务13成功

1. 查询每个地点的密切接触者的数量。

对密切接触表、地点表进行自然联合，使用GROUP BY根据地点序号分组，使用COUNT统计每个地点的密切接触者数量，最后根据该数排序显示。SQL语句如下，结果如图 3‑22所示。

|  |
| --- |
| SELECT location\_name, COUNT(\*) 'close\_contact\_number'  FROM close\_contact, location  WHERE location.id = loc\_id  GROUP BY loc\_id  ORDER BY close\_contact\_number DESC, location\_name; |



图 3‑22 查询任务14成功

1. 查询密接/感染人数最多的用户的名字和感染人数

根据密切接触次数从大到小排序，使用LIMIT 1获得次数最多的用户。SQL语句如下，结果如图 3‑23所示。

|  |
| --- |
| SELECT case\_p\_id, fullname, COUNT(\*) 'infected\_number'  FROM close\_contact, person  WHERE person.id = case\_p\_id  GROUP BY case\_p\_id  ORDER BY infected\_number DESC  LIMIT 1; |

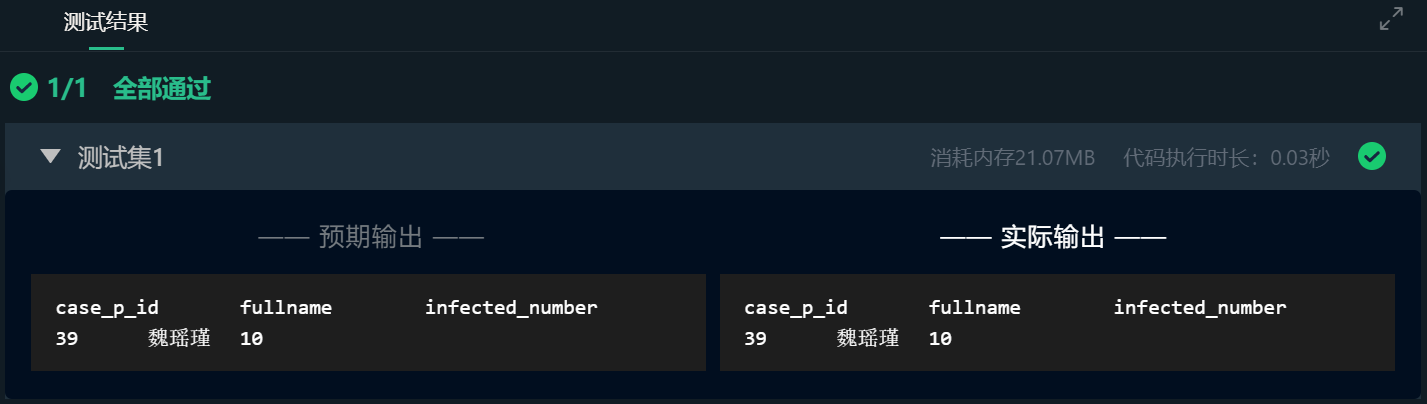


图 3‑23 查询任务15成功

1. 查询2021-02-02 10:00:00到14:00:00，行程记录最频繁的五个人及行程记录数目。

对行程表进行筛选，选择到达时间和离开时间均在2021-02-02 10:00:00到14:00:00之间的行程，根据GROUP BY按照人员分组并按照COUNT次数从大到小排序，最后使用LIMIT 5筛选出前五名（行程记录最频繁的五个人）。SQL语句如下，结果如图 3‑24所示。

|  |
| --- |
| SELECT fullname, COUNT(\*) 'record\_number'  FROM itinerary,person  WHERE p\_id = person.id      AND ((s\_time >= '2021-02-02 10:00:00' AND s\_time < '2021-02-02 14:00:00')      OR (e\_time > '2021-02-02 10:00:00' AND e\_time <= '2021-02-02 14:00:00'))  GROUP BY p\_id  ORDER BY record\_number DESC, fullname ASC  LIMIT 5; |



图 3‑24 查询任务16成功

### 3.2.4 了解系统的查询性能分析功能（选做）

选择上述2.3任务中某些较为复杂的SQL语句，查看其执行之前系统给出的分析计划和实际的执行计划，记录观察的结果，并对其进行简单的分析。

### 3.2.5 DBMS函数及存储过程（选做）

1. 编写一个依据人员编号计算其达到所有地点的次数的自定义函数，并利用其查询至少到达过3个地点的人员。

|  |
| --- |
| set global log\_bin\_trust\_function\_creators=TRUE;  -- 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  DELIMITER ;;  CREATE FUNCTION Count\_Records(tmp\_id INT)  RETURNS INT  BEGIN      RETURN (SELECT COUNT(\*) FROM itinerary WHERE p\_id = tmp\_id);  END;;  DELIMITER ;  -- 利用创建的函数，仅用一条SQL语句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 person.id 'id', fullname, telephone  FROM person, itinerary  WHERE person.id = p\_id AND Count\_Records(person.id)>=3  GROUP BY person.id  ORDER BY person.id ASC; |

函数测试结果如图 3‑25所示。

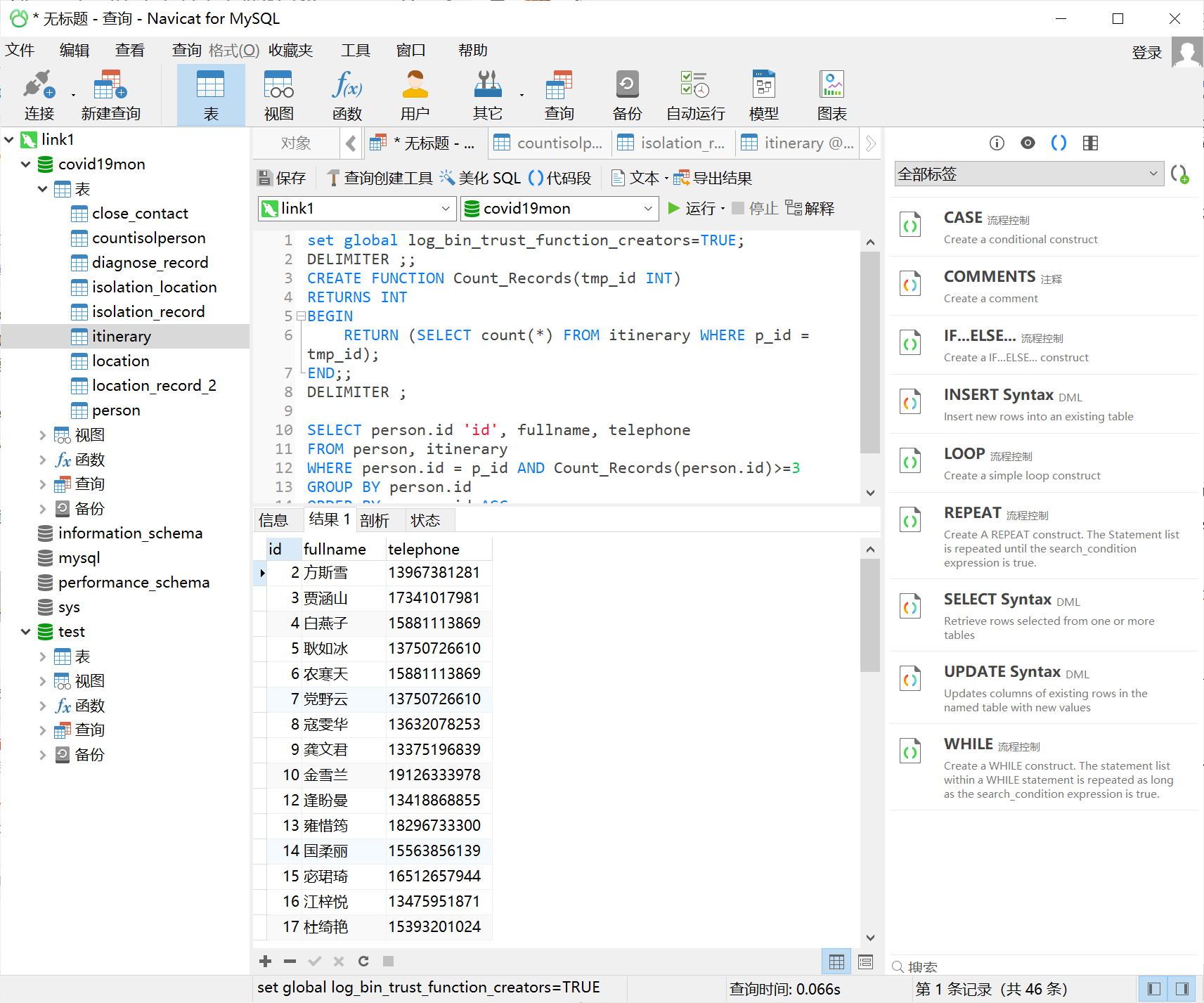


图 3‑25 函数查询

1. 尝试编写DBMS的存储过程，建立每个隔离点的人数统计表，并通过存储过程更新该表。

|  |
| --- |
| CREATE TABLE countIsolPerson(  loc\_id INT,  num INT,  CONSTRAINT pk\_isol\_loc\_id PRIMARY KEY(loc\_id)  );  DELIMITER ;;  CREATE PROCEDURE sumIsolPerson()  BEGIN  INSERT INTO countIsolPerson(loc\_id,num)  SELECT isol\_loc\_id,count(\*)  FROM isolation\_record  WHERE state = 1  GROUP BY isol\_loc\_id;  END;;  DELIMITER ;  CALL sumIsolPerson(); |

结果如图 3‑26所示：

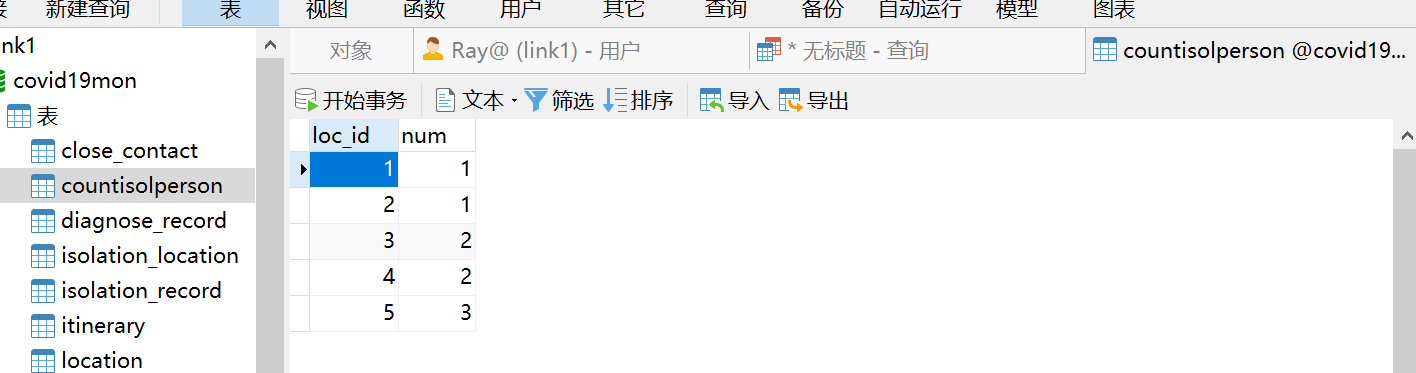


图 3‑26 更新表成功

## 3.3任务总结

sql练习任务主要是帮助我们熟悉sql语句的各种操作。

首先是建表部分。建表部分并不困难，但由于实验是在educoder平台上进行的，而当时的建表说明中表6的隔离地点表、表7的隔离表是反的，所以一开始建表没有注意顺序，导致报了很多错。由于隔离表中有需要引用隔离地点表的foreign\_key，所以必须要按照顺序建表。

然后是数据更新部分。增删改部分都比较基础，但是触发器的实验卡了一段时间。由于MySQL一次只能定义一个事件（insert,delete,update三选一）驱动的触发器。这意味着，如果希望多个事件均触发完整性检查，得创建多个触发器。这个实验就需要同时触发更新（update）和插入（insert），所以需要写两个触发器。而且MySQL需要先用“DELIMITER 界符”语句指定触发器定义语句的界符（即结束符)。这个实验中是使用了“;;”(即两个连续的分号)作触发器定义语句的界符，并且在在触发器定义结束后重新指定语句的界符(恢复为单个分号)。

然后是任务量很大的数据查询部分。这个部分需要仔细推敲题目条件，而且educoder平台上可以看到正确输出，就会更容易意识到要求。其中印象最深的是第4题查询某个人的行程，由于这题有个特殊的要求，就是如果这个人没有任何行程，但查询结果依然要显示出来，只是姓名之后的内容都以“NULL”显示。这道题困了我一段时间，后来通过仔细查看书本会PPT回顾知识点，才使用“LEFT JOIN”解决了问题。

最后还有附加选做的函数任务。通过这个任务我基本了解了如何创建函数、使用函数。在测试函数的时候Navicat报了“This function has none of DETERMINISTIC,NO SQL, or READS SQL DATA in….”错误，通过上网搜索发现这是我们开启了bin-log, 我们就必须指定我们的函数是否是DETERMINISTIC 确定性的、READS SQL DATA 、只读数据等等。最后的解决方法是在首行加上：set global log\_bin\_trust\_function\_creators=TRUE;

总体来说，SQL练习任务十分显著地提高了我对数据库的理解，也让我对一些指令、在什么场合使用更加熟练。为之后的综合实践任务打下了良好的基础。

# 综合实践任务

## 4.1系统设计目标

随着社会发展的不断进步，出行选择飞机的消费者也越来越多，为了满足人们的出行需求，方便旅客购票以及机场管理，实现航空公司的现代化管理，进一步提高工作效率，需要开发一个售票及管理系统。该系统需要具有完整的存储，查询，管理，核对等功能。用户可以通过该系统进行机票的查询预订操作，机场可以对航班班次、用户订单等信息进行查询和管理。

该C/S系统分别从管理员和用户两个角色出发，目标实现了航班管理、预订机票两大主要功能模块。管理员可以管理航班、管理班次、查看用户订单；用户可以查询航班、预订机票、退订机票。通过数据库和前后端代码的连接，实现了机票预订系统的核心逻辑和必须功能。

## 4.2 需求分析

* + 1. **实验要求**

**1．系统功能的基本要求**

1. 每个航班信息的输入；
2. 每个航班的坐位信息的输入；
3. 当旅客进行机票预定时，输入旅客基本信息，系统为旅客安排航班，打印取票通知和帐单；
4. 旅客在飞机起飞前一天凭取票通知交款取票；
5. 旅客能够退订机票；
6. 能够查询每个航班的预定情况、计算航班的满座率。
7. 包含事务（包含commit，rollback），存储过程/触发器，视图，函数。
8. 在程序中需要体现SQL和编程语言的结合。

**2．数据库要求：**

在数据库中至少应该包含下列数据表：

1. 航班信息表；
2. 航班坐位情况表；
3. 旅客订票信息表；
4. 取票通知表；
   * 1. **需求分析**

根据要求以及现有的产品参考进行需求分析。首先构想的是做一个C/S系统，分别从用户端、管理员端两个角度出发进行功能设计。其中两边功能主要有如下几点：

用户端主要功能：

1. 登录注册：每个用户可以注册登陆账号，进行机票的预订退订操作；
2. 查询航班：通过输入起飞地点、到达地点、起飞时间搜寻，航班信息还要包括航班号、起飞降落的时间地点、票价；
3. 预订航班：输入旅客基本信息以及航班座位，打印取票通知和帐单；
4. 退订机票：旅客在飞机起飞前、交付账单之前都可以退订机票；

管理端主要功能：

1. 管理航班班次：可以增加航班，增加、删除、查询航班班次；
2. 管理订单：可以查询用户订单信息；

可以做的附加功能：

1. 可以支持用户选座（靠窗、过道选择）；
2. 可以设置飞机延时，对用户进行通知；

整体需求的主要模块如下图 4‑1所示。

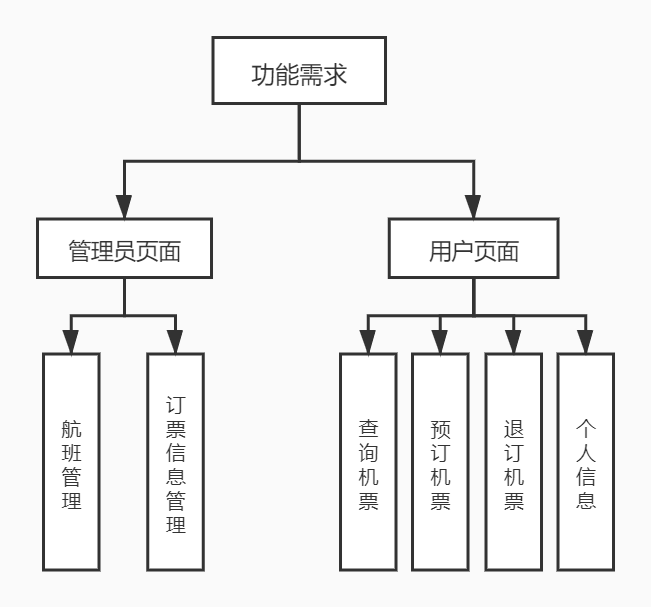


图 4‑1 需求图

* + 1. **数据流图**

根据大致需求，进行详细拆分后可以得到数据流图，如下图 4‑2所示。

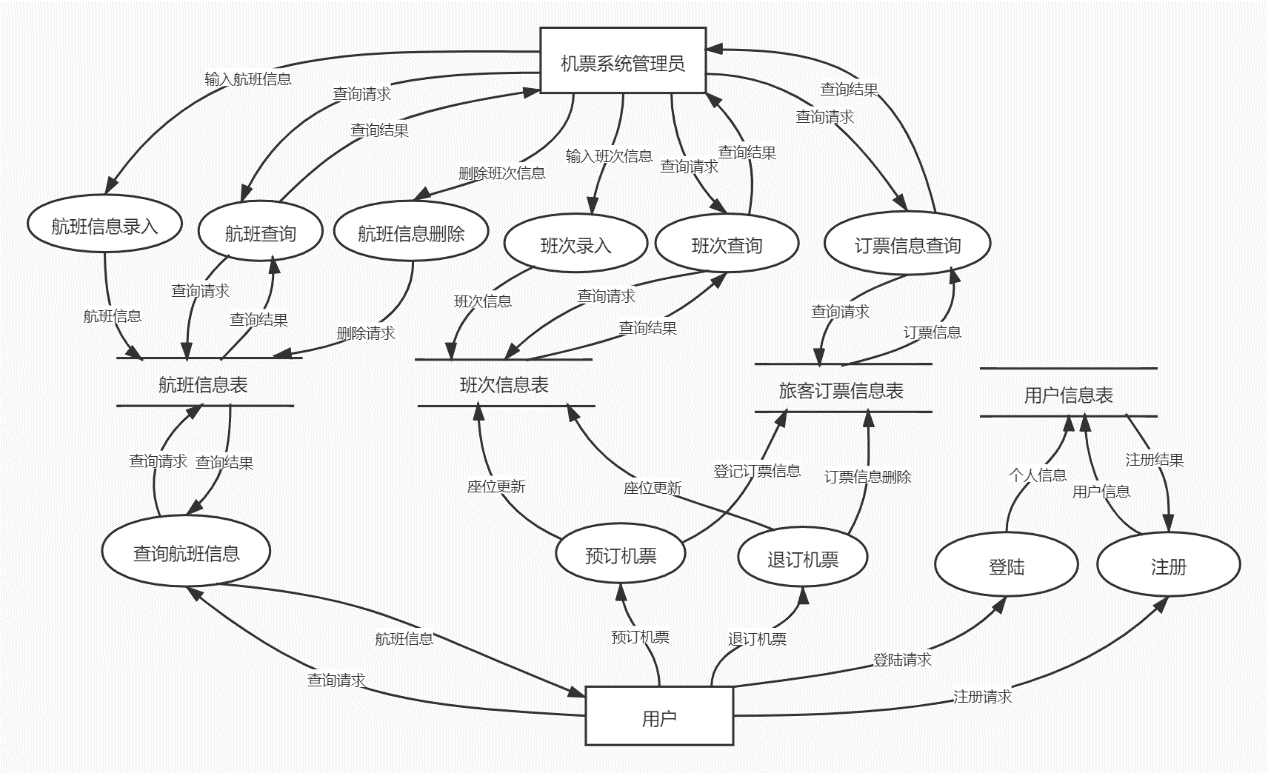


图 4‑2 数据流图

* + 1. **系统的数据需求**

机场售票和管理系统的数据需求应该包括如下几点:

（1）数据录入和处理的准确性和实时性

数据的输入是否准确是数据处理的前提，错误的输入会导致系统输出的不正确和不可用从而使系统的工作失去意义。数据的输入来源是手工输入，手工输入要通过系统面上的安排系统具有容错性，并且对操作人员要进行系统的培训。在系统中，数据输入往往是大量的，因此系统要有一定的处理能力，以保证迅速的处理数据。

（2）数据的一致性与完整性

由于系统的数据是共享的，在不同的航空公司中，机票航班等是共享数据，所以如何保证这些数据的一致性，是系统必须解决的问题。要解决这一问题，要有一定的人员维护数据的一致性，在数据录入处控制数据的去向，并且要求对数据库的数据完整性进行严格的约束。对于输入的数据，要为其定义完整性规则，如果不能符合完整性约束，系統应拒绝该数据。

（3）数据的共享与独立性

整个机场售票和管理系统的数据是共享的。然而，从系统开发的角度上看，共享会给设计和调试带来困难。因此，应该提供灵活的配置，使各个分系统能够独立运行，而通过人工干预的手段进行系统数据的交换。这样，也能提供系统的强壮性。

## 4.3 总体设计

系统的设计主要为功能设计、页面设计、数据库设计三大块，也就是后端、前端、和数据存储。其中前两大数据模块在下方分点阐述。

* + 1. **功能设计**

系统的功能模块组成主要为两部分：用户模块、管理员模块。用户模块需要实现的两大功能就是预订机票、退订机票，为了识别不同用户进行的操作，还需要登录注册系统。管理员模块要实现的功能简单来说就是增删改查，考虑到实际操作的真实性，这里设计的管理员有权利对航班和班次进行查询、增加、删除操作，对用户的订单只有查询操作。

两个模块的功能逻辑设计和大致流程如下图 4‑3、图 4‑4所示。

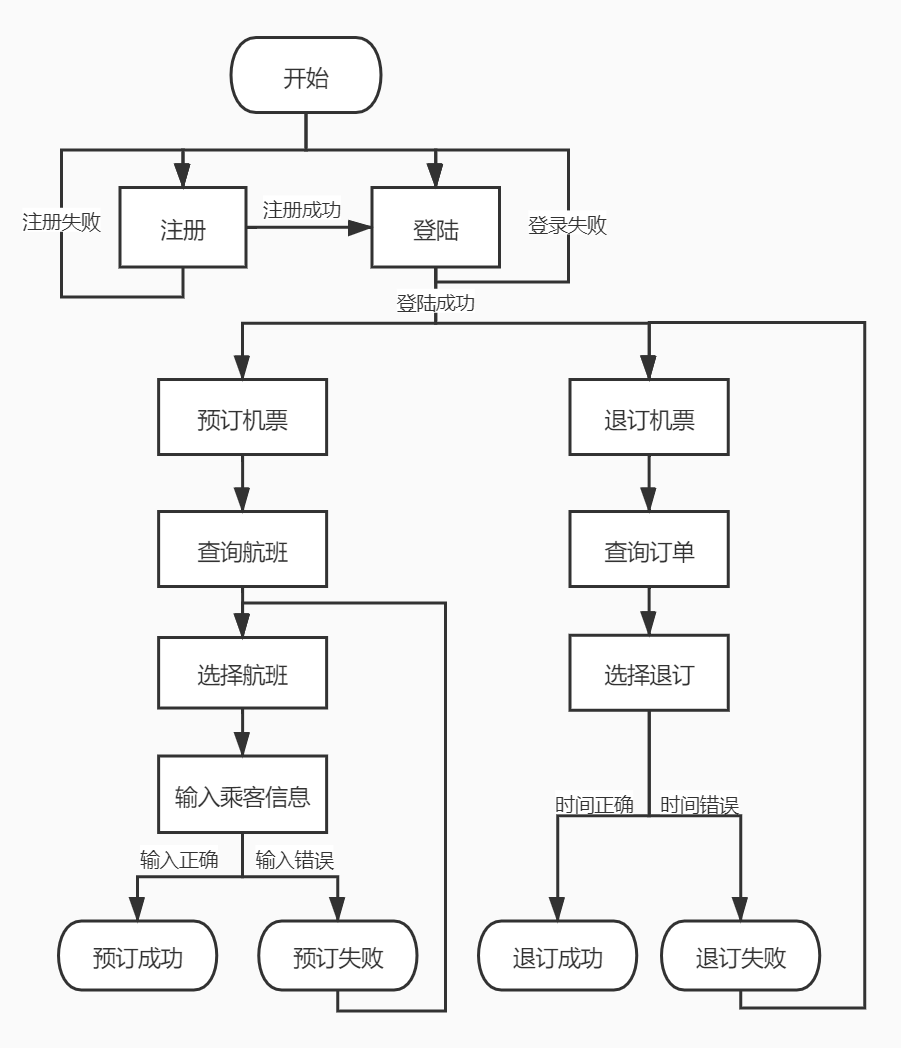


图 4‑3 用户模块

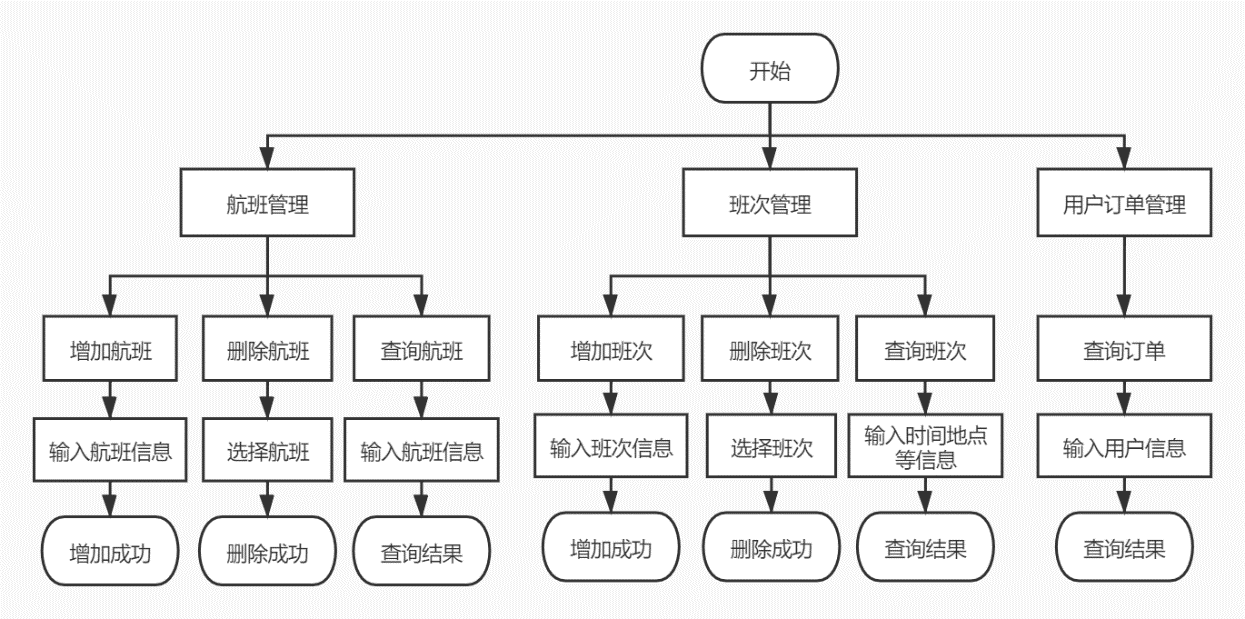


图 4‑4 管理员模块

* + 1. **页面设计**

页面设计如下所示：

1. 管理员

- 班次管理

- 查看班次页面

- 查看订单页面

- 增加班次页面

- 删除班次页面

- 航班管理

- 增加航班页面

- 删除航班（未完成）

2. 用户

- 登录注册

- 登录页面

- 注册页面

- 预定机票

- 航班查询页面

- 输入信息预定页面

- 退订机票

- 查询订单并删除页面

- 个人信息（未完成）

其中部分原型页面设计如下图 4‑5 机票查询图、 4‑6 信息输入、图 4‑7 班次查询所示。



图 4‑5 机票查询



图 4‑6 信息输入



图 4‑7 班次查询

## 4.4 数据库设计

* + 1. **ER图设计**

ER图设计如下图 4‑8所示。

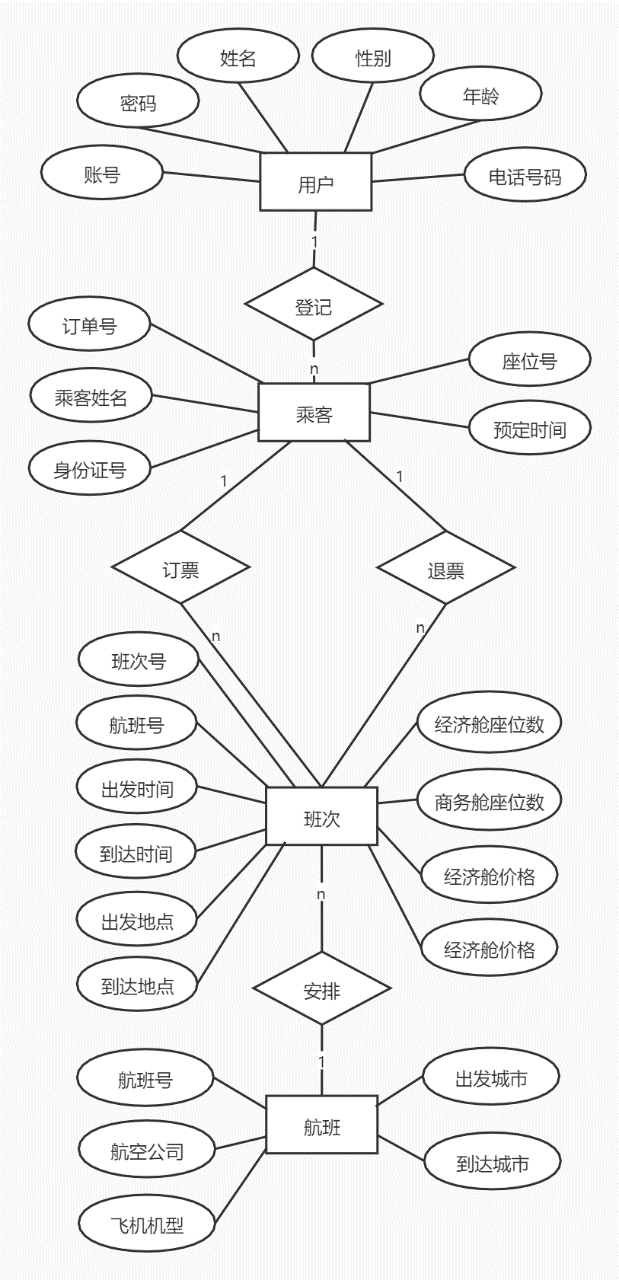


图 4‑8 ER图

根据ER图分析，可以根据三个原则进行表的设计。

1. 根据实体内容建表；（用户、乘客、班次、航班）

2. 根据联系建表；（登记、订票、退票、安排）

3. 对上述表进行合并简化。

最初设计过于冗余，在功能的开发过程中不断修改打磨，更改结构设计。因此在最终设计中，总共设计了四个表，涵盖了所需要的所有信息，分别是：

航班信息表：（航班号，公司类型，机型，出发城市，到达城市）

班次信息表：（班次号，航班号，出发城市，出发时间，到达城市，到达时间，商务舱座位数，经济舱座位数，商务舱价格，经济舱价格）

用户信息表：（账号，密码，姓名，年龄，性别，电话号码）

乘客信息表：（订单号，账号，班次号，乘客名，身份证号，舱位类型，座位号，订单时间）

* + 1. **逻辑设计**

表的详细设计及字段内容如下。

1. 航班信息表（flight）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 含义 | 类型 | 备注 |
| flight\_id | 航班号 | varchar(20) | 主码 |
| company\_name | 公司名 | varchar(20) |  |
| plane\_type | 机型 | varchar(20) |  |
| departure\_city | 出发城市 | varchar(20) |  |
| arrival\_city | 到达城市 | varchar(20) |  |

1. 班次信息表（classes）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 含义 | 类型 | 备注 |
| class\_id | 班次号 | varchar(20) | 主码 |
| departure\_time | 起飞时间 | datetime |  |
| departure\_airport | 起飞机场 | varchar(20) |  |
| arrival\_time | 到达时间 | datetime |  |
| arrival\_airport | 到达机场 | varchar(20) |  |
| a\_number | 商务舱座位 | int(11) |  |
| b\_number | 经济舱座位 | int(11) |  |
| a\_price | 商务舱价格 | int(11) |  |
| b\_price | 经济舱价格 | int(11) |  |
| flight\_id | 航班号 | varchar(20) | 外码，约束名为classes\_flight\_fid\_fk |

1. 用户信息表（user）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 含义 | 类型 | 备注 |
| account | 账号 | varchar(20) | 主码 |
| password | 密码 | varchar(20) |  |
| name | 用户名 | varchar(20) |  |
| sex | 性别 | varchar(10) |  |
| age | 年龄 | int |  |
| tel | 电话号码 | varchar(20) |  |

1. 乘客信息表（passengers）

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 含义 | 类型 | 备注 |
| order\_id | 订单号 | int | 主码 |
| name | 乘客名 | varchar(20) |  |
| account | 账号 | varchar(20) | 外码，约束名为passengers\_users\_account\_fk |
| id | 身份证号 | varchar(20) |  |
| order\_time | 订单时间 | datetime |  |
| seat | 座位号 | varchar(5) |  |
| degree | 舱位类型 | varchar(2) |  |
| class\_id | 班次号 | varchar(20) | 外码，约束名为passengers\_classes\_cid\_fk |

建表SQL语句如下

|  |
| --- |
| -- 表1 航班信息表(flight) 记录航班信息  -- 航班号flight\_id，公司名company\_name，飞机类型plane\_type  -- 出发城市departure\_city，到达城市arrival\_city  DROP TABLE IF EXISTS flight;  CREATE TABLE flight (  flight\_id varchar(20) NOT NULL PRIMARY KEY,  company\_name varchar(20) NOT NULL,  plane\_type varchar(20) NOT NULL,  departure\_city varchar(20) NOT NULL,  arrival\_city varchar(10) NOT NULL  );  -- 表2 班次信息表(class) 记录班次信息  -- 航班号id，航班名name，起飞时间departure\_time，到达时间arrival\_time  -- 起飞地点departure\_airport，到达地点arrival\_airport  -- 座位数a\_number,b\_number, 票价a\_price，b\_price  DROP TABLE IF EXISTS classes;  CREATE TABLE classes (  class\_id varchar(20) NOT NULL PRIMARY KEY,  departure\_time datetime NOT NULL,  departure\_airport varchar(20) NOT NULL,  arrival\_time datetime NOT NULL,  arrival\_airport varchar(20) NOT NULL,  a\_number int(11) NOT NULL,  b\_number int(11) NOT NULL,  a\_price int(11) NULL,  b\_price int(11) NULL,  flight\_id varchar(20) NULL,  constraint classes\_flight\_fid\_fk FOREIGN KEY (flight\_id)  references flight(flight\_id)  on delete cascade on update cascade  );  -- 表3 用户信息表(users)记录用户信息  -- 账号account, 密码password，用户姓名name，性别sex  -- 年龄age，电话tel  DROP TABLE IF EXISTS users;  CREATE TABLE users (  account varchar(20) NOT NULL PRIMARY KEY,  password varchar(20) NOT NULL,  name varchar(20) NOT NULL,  sex varchar(10) null,  age int null,  tel varchar(20) null  );  -- 表4 旅客订票信息表  -- 序号order\_id, 身份证号id, 座位号seat, 舱位degree  DROP TABLE IF EXISTS passengers;  CREATE TABLE passengers (  order\_id int NOT NULL PRIMARY KEY,  name varchar(20) NOT NULL,  account varchar(20) NOT NULL,  id varchar(20) NOT NULL,  order\_time datetime NOT NULL,  seat varchar(5) NOT NULL,  degree varchar(2) NOT NULL,  class\_id varchar(20) NOT NULL,  constraint passengers\_classes\_cid\_fk FOREIGN KEY (class\_id)  references classes(class\_id)  on delete cascade on update cascade,  constraint passengers\_users\_account\_fk FOREIGN KEY (account)  references users(account)  on delete cascade on update cascade  ); |

* + 1. **物理设计**

本系统主要通过MySQL+Navicat实现了数据库的功能，通过java的DBHelper设置对应数据库的连接，使得程序可以成功执行。

数据库及表单设计如图 4‑9所示。

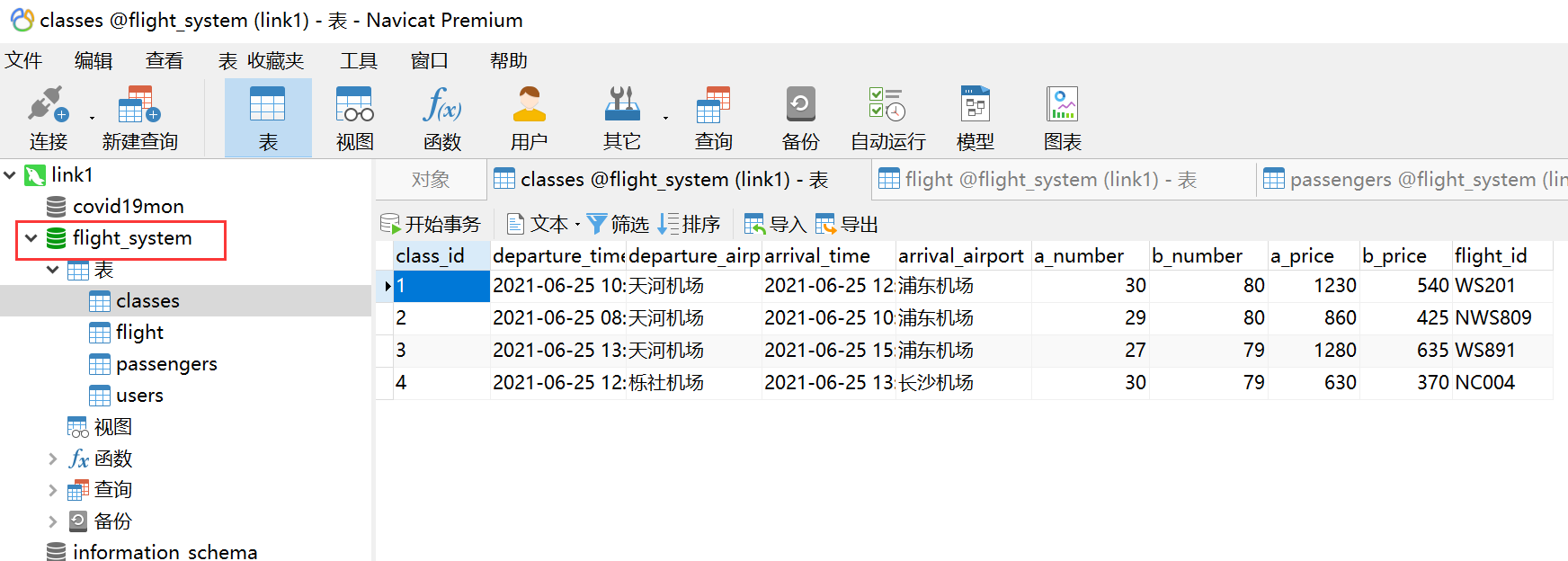


图 4‑9 数据库和表

数据库的连接配置文件如图 4‑10所示。

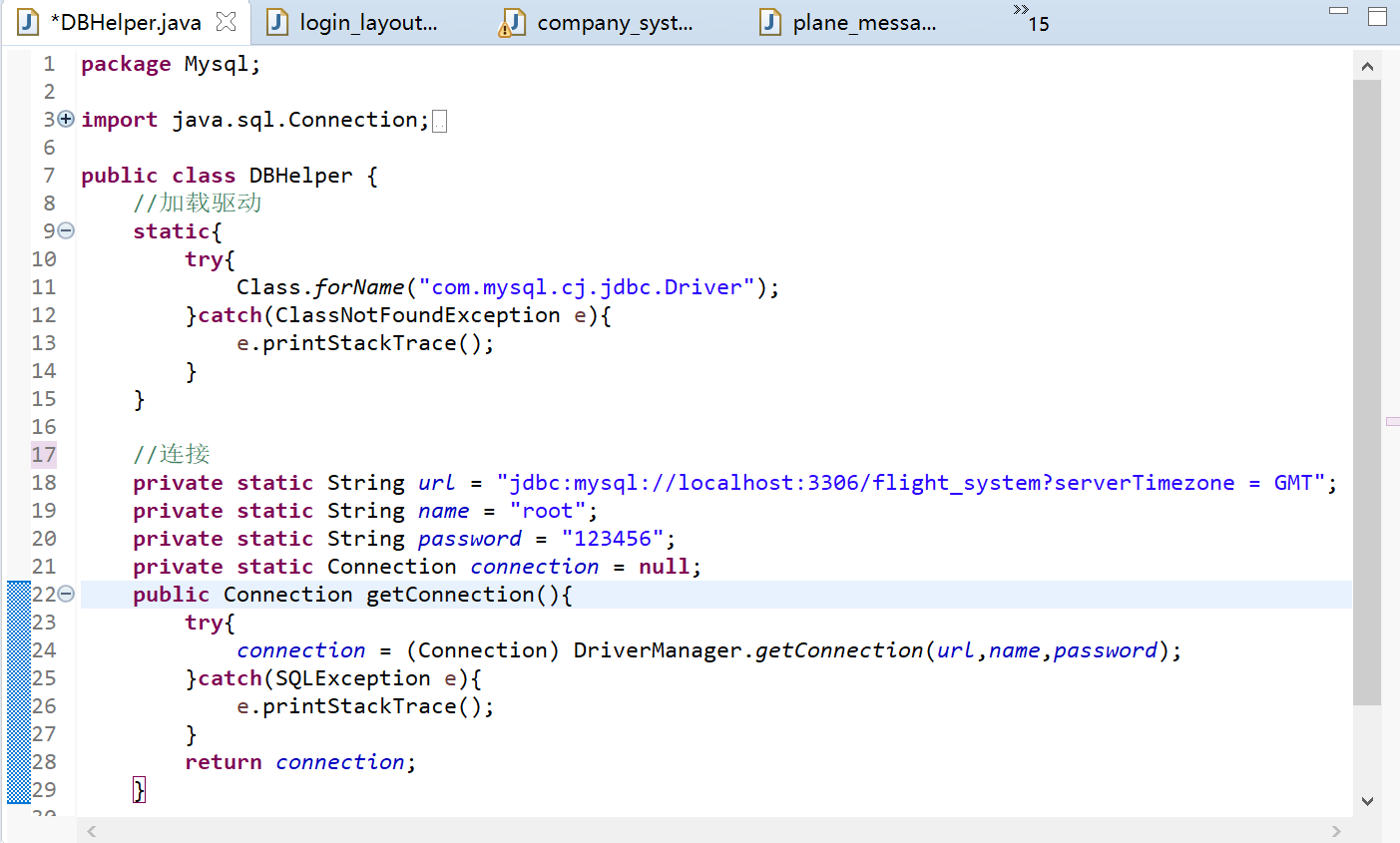


图 4‑10 数据库连接

## 4.5 详细设计与实现

* + 1. **开发环境**

数据库：MySQL+Navicat

IDE：Eclipse

语言：Java

界面UI：Java Swing

* + 1. **功能详细设计**

1. 登录功能

用户在登录界面输入账号和密码，若账号存在，且密码与数据库中储存的密码一致时，登录成功，不一致则出现弹框显示账号不存在/密码错误。在程序中通过select语句实现了账号密码的查询，如下图 4‑11所示。

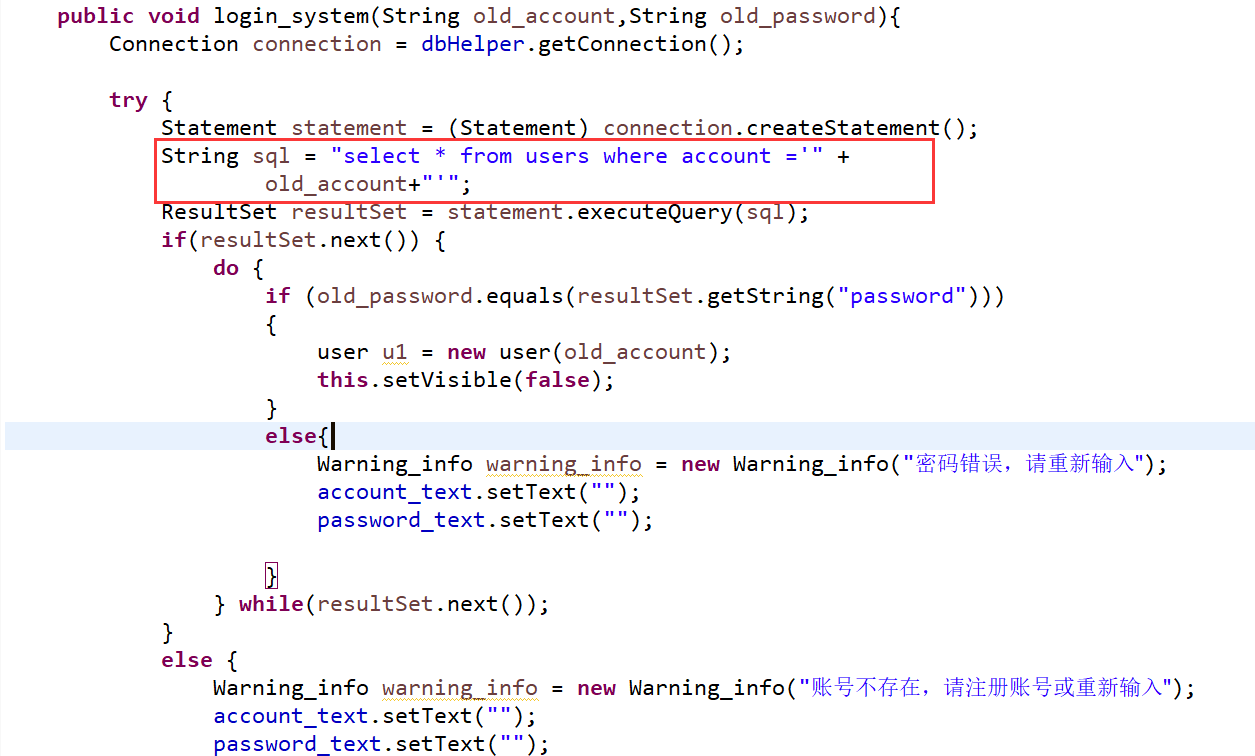


图 4‑11 登录查询账号密码

1. 注册功能

注册新用户，用户在输入框输入个人信息，其中账号、密码、姓名是必填项，按下提交后校验发现没有填的话则弹出弹框提示用户填写。数据填写正确后通过语句检查数据库中是否已存在该账号，若存在，则提示用户重新输入账号。账号无重复后资料输入无误后，账号创建成功，数据库中插入新用户信息，并跳回登录界面。

其中插入新用户主要使用了Insert语句，如图 4‑12所示。



图 4‑12 插入新用户信息

1. 航班查询功能

用户登录成功后，进入机票查询界面，在对应位置输入出发城市、到达城市、出发时间、到达时间，还可通过公司和舱位进行筛选，最后在右侧显示筛选后的班次信息，班次信息可以看到出发时间、到达时间、出发机场、到达机场、票价等详细信息。用户可以通过“预订”按键预订机票。

查询筛选班次主要使用了Select语句，出发城市、到达城市、出发时间、到达时间这四个为必选项，所以写在了基础语句之中，其它筛选项通过if语句判断是否选择后再加入SQL语句。主要语句内容如图 4‑13所示。



图 4‑13 查询筛选班次信息

1. 订票功能

选择想要预定的班次，点击“预订”按钮后，会出现一个填写乘客信息的弹框，正确输入乘客基本信息后则预定成功，对应班次的余票减一处理。如果同一班次的座位号已被预订，则会弹框提示“座位号已被预订，预定失败”，否则弹框显示“预订成功”。

订单号是逐个递增的，由于考虑到可能在有乘客退订的情况下订单号不连续，所以在插入新订单的时候，新order\_id的获取是先通过一个select max(order\_id)获取原本订单号的最大值，最后插入的时候加一。更新票数也是根据班次号、舱位类型进行判断，再使用update语句更新对应班次对应舱位的座位数为（原来的值-1）。主要部分代码如下图 4‑14所示。



图 4‑14 机票预订

1. 退订功能

用户可以通过左上角的菜单进入“退订机票”页面。通过选择框用户可以看到自己账号下的订单号并进行选择。对应订单号对应的数据会显示页面中，在确认订单信息无误后用户可以点击“确认取消订单”按钮取消该订单。

在取消订单后要进行两个操作，一个是使用delete语句删除乘客表中对应的订单数据，另一个是对对应的座位进行加一操作。这个地方的设计本来可以更简便，让座位数为(初始值-count(对应订单数))的设计，然后利用触发器来实现。这也算是一个实现不太完美的点。目前系统的具体实现如下图 4‑15所示。



图 4‑15 退订机票

1. 班次订单查询功能

左上角菜单栏可以选择查询班次信息还是订单信息，通过输入框输入一些对应信息后，点击查询按钮就可显示筛选后对应的班次、订单信息，可供管理员对信息进行检查核对。信息以表格形式呈现。

其中订单查询的语句如下图 4‑16所示。



图 4‑16 订单查询

1. 增加航班功能

输入航班各个信息后，点击确认，当航班号重复时，会提示航班已存在，重新添加航班号。信息无误后点击确认，则航班添加成功。

其中增加航班的主要语句如图 4‑17所示。



图 4‑17 增加航班

1. 增加班次功能

选择增加班次，添加班次的基本信息后，确认添加，旁边列表会显示对应班次已有的起飞降落时间，如有冲突则添加失败，如果信息填写正确且没有冲突班次则添加班次成功。

1. 删除班次功能

选择取消班次，通过选择框管理员可以看到所有的班次号并进行选择。对应班次号对应的数据会显示在页面中，检查无误后，点击确认删除，该班次则取消成功。

其中取消班次的主要语句如图 4‑18所示。

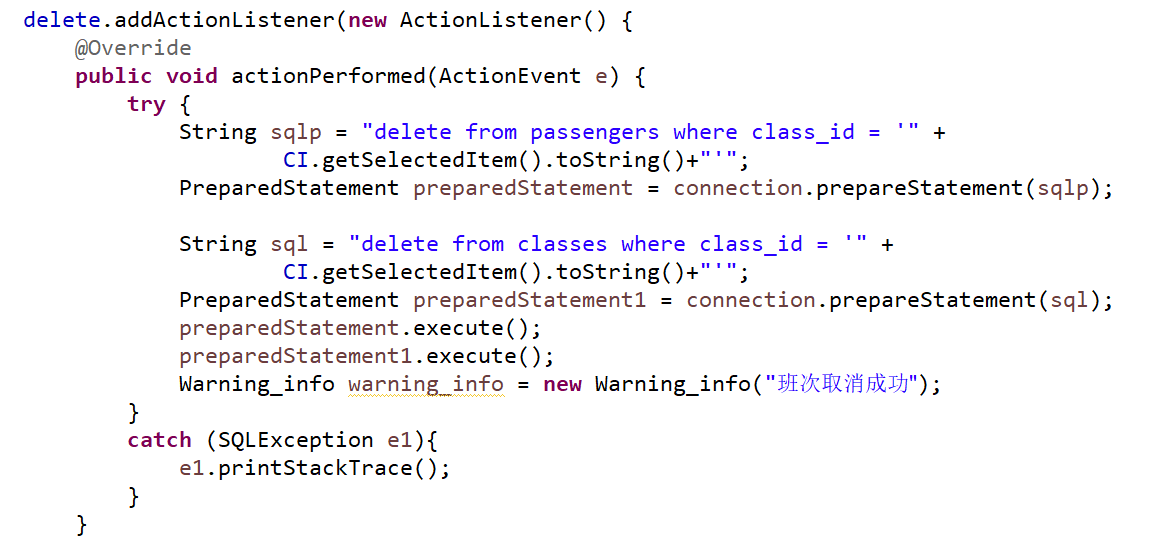


图 4‑18 取消班次

* + 1. **包模块设计**

该系统一共有三个包，分别为机场（管理班次&查询订单）、公司（管理航班）、用户（登录注册、预订退订）。

|  |  |  |
| --- | --- | --- |
| package | .java | 备注 |
| Airport | flight | 增加班次、删除班次 |
| main\_system | 入口文件 |
| research | 查询订单、查询班次 |
| Company | addflight | 增加航班、删除航班（未完成） |
| company\_system | 入口文件 |
| Customer | plane\_message | 入口文件 |
| login\_layout | 登陆页 |
| register\_layout | 注册页 |
| user | 预订/退订菜单选择页 |
| plane\_message\_layout | 预订机票页 |
| unsubscribe | 退订机票页 |
| order\_layout | 填写乘客信息弹框 |
| payment\_layout | 账单、取票详情页（未完成） |
| buttonUI | 各种按钮的显示样式 |
| Warning\_info | 提示弹框 |

## 4.6 系统测试

系统页面展示如下。

用户页面入口文件进入就是登陆页，输入账号密码正确，选择登陆可以进入机票预订/退订页，选择注册可以进入注册页。如图 4‑19所示。



图 4‑19 登录页面

注册页面，输入账号、密码、姓名三个必填项，电话、年龄两个选填项，以及一个性别选择框。点击“注册”按钮，如果没有重复的账号，且输入数据符合规范就会跳转到登录页面，否则会出现注册失败弹框。如图 4‑20所示。

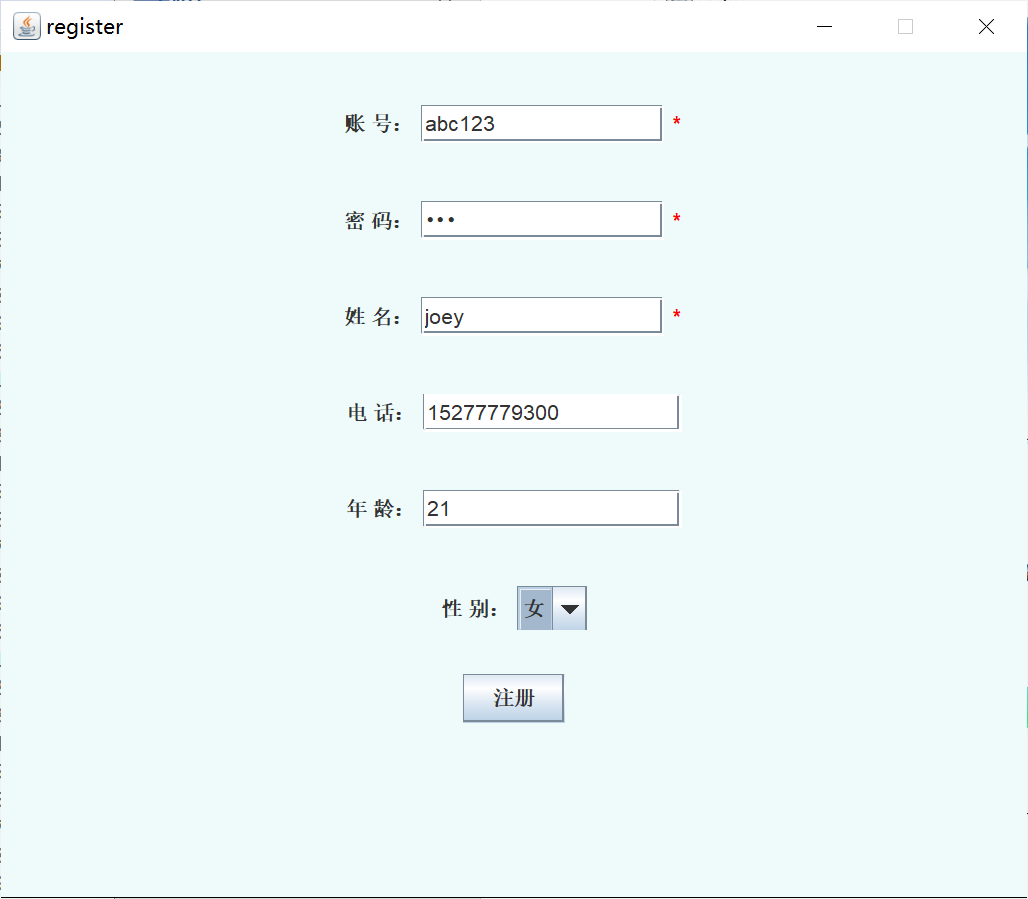


图 4‑20 注册页面

机票查询页面，左边的输入框可以输入想要查询的航班班次信息。点击“搜索”在右侧会出现查询结果。如图 4‑21所示。



图 4‑21 机票查询

得到搜索结果的查询页面。每一个符合要求的班次都有两个舱位：商务舱和经济舱，可以选择需要的进行机票预订操作。如图 4‑22所示。



图 4‑22 筛选结果

选择预订机票后弹框要求输入信息，包括乘客证件号、乘客姓名以及要预约的座位号。如果座位号已被预订则预定失败，否则添加成功。如图 4‑23所示。



图 4‑23 机票预订

通过左上角的菜单栏可以切换到退订机票页面。如图 4‑24所示。



图 4‑24 机票退订

选择当前账号已预订的订单号可以查看详细信息。确认无误后点击“确定取消该订单”删除订单。如图 4‑25所示。



图 4‑25 选择订单

班次信息查询页面，上面输入信息点击“搜索”可以筛选班次。默认显示所有班次信息。如图 4‑26所示。



图 4‑26 班次查询

订单信息查询页面，上面输入信息点击“搜索”可以筛选订单。默认显示所有订单信息。如图 4‑27所示。



图 4‑27 订单信息查询

安排航班班次页面。可以选择对应的航班号，输入信息则可安排航班班次。尤其需要注意输入的机场和起飞时间信息，会在右边的两个表格显示已占用的时间段，不能相互冲突，否则会添加班次失败。如图 4‑28所示。



图 4‑28 安排班次

选择当前班次号可以查看班次信息。确认无误后点击“确定取消航班”删除班次。如图 4‑29所示。



图 4‑29 取消班次

增加航班页面，可以输入信息增加航班。如图 4‑30所示。



图 4‑30 增加航班

## 4.7 系统设计与实现总结

1. 设计了系统ER图、数据流图；
2. 实现了数据库的表创建；
3. 实现了数据的增删改查；
4. 实现了系统完整的功能流程；
5. 实现了系统和数据库的连接；
6. 实现了页面UI；

# 5 课程总结

通过本次实验，我感觉到了自己能力综合性的成长。本次实验共分成了三个部分，软件功能学习、SQL练习、综合实践，三个实验由浅及深，由易到难。

第一个实验虽然没有太多可言的，但是在选择环境的时候，我认真对比了SQL Server和MySQL之间的优劣，最后选择了更加轻便的MySQL，并且下载了Navicat作为图形化界面工具来简化一些操作。

第二个实验题量很多，不过也涉及到了数据操作的各个方面，通过复杂的条件设置来引导我去活学活用课堂中的理论知识，将各种查询、插入、删除语句的使用牢牢记住。还有触发器、函数等在实际开发很实用的学习内容。在解题的过程中，会碰到很多问题，比如说“查询去过所有地点的人”，一开始看到这道题毫无头绪，后来从书中看到了类似的例题，通过两个not exists将题意衍生成了“不存在这样的地方，这个人没有去过”从而巧妙的解决了本来可能要写很长的题目。从中我学习到的是宝贵的思想。

第三个实验是一个范围很广，上限很高的实验。由于一个完整的系统不仅仅需要数据库相关的实现，还需要前端的页面、后端的逻辑实现，才能呈现出一个完整的功能系统。这个实验对我来说最困难的部分是前后端的实现，数据库的内容反而在其次了。由于之前没有尝试过独立制作包含前后端的系统，所以在开发环境的选择上就浪费了很多时间。最开始想使用的是React+Antd作为前端，SpringBoot作为后端，但是在联调的时候总是有很多问题。最后决定“返璞归真”，还是使用Java写逻辑，展示界面就不那么严格，使用Java Swing的默认组件来写。这才开始动工，总共开发时长也就一个星期左右。

现在回想起自己的开发历程，还是有很多问题和收获。问题在于最初的设计不够完备。在开发过程中经常写着写着功能就发现不合理，然后回头去更改数据表的结构，这样费时费力，所以在最开始就应该思考清楚、设计完备。其次感到不足的是座位数量的设计，现在的设计是根据操作响应，也就是“预订”这个操作会触发条件，有SQL语句操作对应座位数量减少，“退订”反之。但是这样的实现其实并不优雅。更好的实现是让座位数本身关联订单数。这样无论是预订还是退订，在更新数据的同时就能使得数据同步，也更不易出错。

由于时间比较赶，还缺少了一个取票通知页面未完成。但是总体系统已经实现了所有必须的功能，所以有一点不满希望下次能继续进步。

感谢指导老师在检查时指出我ER图存在的问题，让我对数据库知识有了更深的了解，这次实验让我收获很大。

# 附录

SQL练习部分代码

|  |
| --- |
| create database if not exists covid19mon;  use covid19mon;  -- 请在以下适当的空白位置填写SQL语句完成任务书的要求。空白位置不够的话，可以通过回车换行增加。  -- 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));  -- 代码结束  /\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/  -- 插入三条数据  INSERT INTO person(id,fullname,telephone)  VALUES(1,"张小敏","13907110001"),(2,"李大锤","18907110002"),(3,"孙二娘","13307100003");  -- 删除2  DELETE FROM person WHERE id = 2  -- 更改电话  UPDATE person SET telephone = "13607176668" WHERE id = 1  -- 触发器  -- SQL Server一次可以定义多个事件(insert,delete,upate三者的任意组合)驱动的触发器，但MySQL一次只能定义一个事件(insert,delte,update三选一)驱动的触发器。这意味着，如果你希望多个事件均触发完整性检查，你得创建多个触发器。\*/  -- MySQL虽然允许你随时定义触发器，但要求你用“DELIMITER 界符”语句指定触发器定义语句的界符（即结束符)。我们推荐你使用“;;”(即两个连续的分号)作触发器定义语句的界符。请在触发器定义结束后重新指定语句的界符(恢复为单个分号)。\*/  DELIMITER ;;  CREATE TRIGGER change\_isolation\_state\_insert AFTER INSERT  ON diagnose\_record FOR EACH ROW  BEGIN  IF NEW.result = 1 THEN  UPDATE isolation\_record SET state = 3  WHERE NEW.p\_id = p\_id;  END IF;  END;;  DELIMITER ;  DELIMITER ;;  CREATE TRIGGER change\_isolation\_state\_update AFTER UPDATE  ON diagnose\_record FOR EACH ROW  BEGIN  IF NEW.result = 1 THEN  UPDATE isolation\_record SET state = 3  WHERE NEW.p\_id = p\_id;  END IF;  END;;  DELIMITER ;  -- 1 人流量>30的地点&人流量  SELECT location\_name,COUNT(loc\_id) 'visitors'  FROM location,itinerary  WHERE location.id = loc\_id  GROUP BY loc\_id  HAVING COUNT(\*)>30  ORDER BY visitors DESC,location\_name ASC;  -- 2 查询每个隔离地中正在进行隔离的人数，并按数量由多到少排序  SELECT location\_name, COUNT(isol\_loc\_id) 'number'  FROM isolation\_record,isolation\_location  WHERE isolation\_location.id = isol\_loc\_id AND state = 1  GROUP BY isol\_loc\_id  ORDER BY number DESC, location\_name ASC;  -- 3 查询行程表中所有属于同一个人的接续行程信息  SELECT person.id,fullname,telephone,a.e\_time 'reclosing\_time',a.loc\_id 'loc1',c.location\_name 'address1',b.loc\_id 'loc2',d.location\_name 'address2'  FROM person,itinerary a,itinerary b,location c,location d  WHERE a.e\_time = b.s\_time AND a.p\_id = b.p\_id AND a.p\_id = person.id  AND a.loc\_id = c.id AND b.loc\_id = d.id AND person.id > 30  ORDER BY person.id,reclosing\_time;  -- 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、到过什么地方（地名），何时到达，何时离开 。 列名原样列出即可，不必用别名。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,s\_time,e\_time  FROM person LEFT JOIN  (SELECT location\_name, p\_id, s\_time, e\_time  FROM location, itinerary  WHERE loc\_id = location.id  ) a  ON (person.id = a.p\_id)  WHERE (person.fullname = '充珉瑶' OR person.fullname = '贾涵山')  ORDER BY person.id DESC, s\_time;  -- 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 '%店%'  ORDER BY id;  \*\*\*  -- 6 新发现一位确诊者，已知他在2021.2.2日20:05:40到21:25:40之间在“活动中心”逗留，凡在此间在同一地点逗留过的，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 fullname  FROM person, itinerary, location  WHERE person.id = itinerary.p\_id AND itinerary.loc\_id = location.id  AND location\_name = '活动中心'  AND (itinerary.s\_time < '2021-02-02 21:25:40')  AND (itinerary.e\_time > '2021-02-02 20:05:40')  ORDER BY fullname;  -- 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, isolation\_record  WHERE isol\_loc\_id = isolation\_location.id  AND isolation\_record.state = 1  GROUP BY isol\_loc\_id  ORDER BY isol\_loc\_id;  -- 8 (带exists关键字)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 fullname,telephone  FROM person  WHERE EXISTS  ( SELECT p\_id  FROM itinerary  WHERE p\_id = person.id  ORDER BY p\_id)  LIMIT 30;  -- 9 写一条带NOT EXISTS 子查询的SQL语句实现下述查询要求：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(id) number  FROM person  WHERE NOT EXISTS  (SELECT \*  FROM itinerary,location  WHERE p\_id = person.id AND itinerary.loc\_id = location.id AND location\_name = 'Today便利店');  -- 10 (带not exists)查询人员表去过所有地点的人员姓名。查询结果依人员姓名顺序排序。  -- 含义转换->不存在这样的地点y,人员x没有去过  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;  -- 11 写一条SQL语句创建能反映所有隔离点现状的视图：isolation\_location\_status的创建  CREATE VIEW isolation\_location\_status (id, location\_name, capacity, occupied)  AS  SELECT isolation\_location.id,location\_name,capacity,COUNT(case when state = 1 then 1 else null end) 'occupied'  FROM isolation\_location LEFT JOIN  (SELECT \*  FROM isolation\_record  WHERE state = 1) a  ON (isolation\_location.id = a.isol\_loc\_id)  GROUP BY isolation\_location.id;  -- 12 从视图isolation\_location\_status中查询各隔离点的剩余空房间的数目。  SELECT location\_name, capacity-occupied 'available\_rooms'  FROM isolation\_location\_status  ORDER BY id;  -- 13 查询靳宛儿接触者的姓名和电话。与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 itinerary a, person  WHERE person.id = a.p\_id  AND a.loc\_id IN (SELECT loc\_id FROM itinerary b, person c WHERE c.fullname = '靳宛儿' AND c.id = b.p\_id)  AND a.s\_time < (SELECT e\_time FROM person p, itinerary i WHERE p.fullname = '靳宛儿' AND a.loc\_id=i.loc\_id AND i.p\_id = p.id)  AND a.e\_time > (SELECT s\_time FROM person p, itinerary i WHERE p.fullname = '靳宛儿' AND a.loc\_id=i.loc\_id AND i.p\_id = p.id)  AND person.fullname!='靳宛儿'  ORDER BY fullname;  -- 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。  SELECT location\_name, COUNT(\*) 'close\_contact\_number'  FROM close\_contact, location  WHERE location.id = loc\_id  GROUP BY loc\_id  ORDER BY close\_contact\_number DESC, location\_name;  -- 15 查询感染人数最多的人员编号，姓名，和被其感染的人数。  SELECT case\_p\_id, fullname, COUNT(\*) 'infected\_number'  FROM close\_contact, person  WHERE person.id = case\_p\_id  GROUP BY case\_p\_id  ORDER BY infected\_number DESC  LIMIT 1;  -- 16 查询2021-02-02 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。  SELECT fullname, COUNT(\*) 'record\_number'  FROM itinerary,person  WHERE p\_id = person.id  AND ((s\_time >= '2021-02-02 10:00:00' AND s\_time < '2021-02-02 14:00:00')  OR (e\_time > '2021-02-02 10:00:00' AND e\_time <= '2021-02-02 14:00:00'))  GROUP BY p\_id  ORDER BY record\_number DESC, fullname ASC  LIMIT 3;  -- 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 MAX(capacity)  FROM isolation\_location  WHERE capacity NOT IN  (SELECT MAX(capacity)  FROM isolation\_location)  );  -- 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  DELIMITER ;;  CREATE FUNCTION Count\_Records(tmp\_id INT)  RETURNS INT  BEGIN  RETURN (SELECT COUNT(\*) FROM itinerary WHERE p\_id = tmp\_id);  END;;  DELIMITER ;  -- 利用创建的函数，仅用一条SQL语句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 person.id 'id', fullname, telephone  FROM person, itinerary  WHERE person.id = p\_id AND Count\_Records(person.id)>=3  GROUP BY person.id  ORDER BY person.id ASC; |

综合实现任务核心代码

|  |
| --- |
| //------------------------登陆  public void login\_system(String old\_account,String old\_password){  Connection connection = dbHelper.getConnection();    try {  Statement statement = (Statement) connection.createStatement();  String sql = "select \* from users where account ='" +  old\_account+"'";  ResultSet resultSet = statement.executeQuery(sql);  if(resultSet.next()) {  do {  if (old\_password.equals(resultSet.getString("password")))  {  user u1 = new user(old\_account);  this.setVisible(false);  }  else{  Warning\_info warning\_info = new Warning\_info("密码错误，请重新输入");  account\_text.setText("");  password\_text.setText("");    }  } while(resultSet.next());  }  else {  Warning\_info warning\_info = new Warning\_info("账号不存在，请注册账号或重新输入");  account\_text.setText("");  password\_text.setText("");  }  resultSet.close();  }catch (SQLException e){  e.printStackTrace();  }finally{    }  }  }  //------------------退订机票  delete = new JButton("确定取消该订单");  delete.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  try {    String sql = "select a\_number, b\_number from classes where class\_id = '"+ClassId+ "'";  Statement statement = (Statement) connection.createStatement();  ResultSet resultSet = statement.executeQuery(sql);  while (resultSet.next()){  a\_number = resultSet.getInt("a\_number");  b\_number = resultSet.getInt("b\_number");  }  resultSet.close();    String sql1 = "delete from passengers where order\_id = '"+ OrderId.getSelectedItem().toString()+"'";  PreparedStatement preparedStatement1 = connection.prepareStatement(sql1);    String sql2;  if(Degree.getText() == "A") {  int cur\_number = a\_number+1;  sql2 = "update classes set a\_number ='"+cur\_number+"' where class\_id='"+ClassId+"'";  }  else {  int cur\_number = b\_number+1;  sql2 = "update classes set b\_number ='"+cur\_number+"' where class\_id='"+ClassId+"'";  }    PreparedStatement preparedStatement2 = connection.prepareStatement(sql2);  preparedStatement1.execute();  preparedStatement2.execute();  Warning\_info warning\_info = new Warning\_info("班次取消成功");    updateUI();  }  catch (SQLException e1){  e1.printStackTrace();  Warning\_info warning\_info = new Warning\_info("出现错误");  }  }  });  //-------------------预订机票  book1.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  order\_layout o = new order\_layout(a\_price,ticketType1,data,departPlace,ArrivalPlace);    o.OK.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  String name = o.name\_text.getText();  String id = o.id\_text.getText();  String seat = o.seat\_text.getText();  Date date = new Date();  SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");  try{  String sql1 = "select order\_id from passengers where seat='"+seat+"' and class\_id='"+class  Statement statement1 = (Statement) connection.createStatement();  ResultSet resultSet1 = statement1.executeQuery(sql1);  if(resultSet1.next()) {  Warning\_info warning\_info = new Warning\_info("输入的座位号已被占用,请重新购票");  }  else {  String sql2 = "select max(order\_id) as number from passengers";  String sql3 = "insert into passengers values (?,?,?,?,?,?,?,?)";  Statement statement2 = (Statement) connection.createStatement();  ResultSet resultSet2 = statement2.executeQuery(sql2);  while(resultSet2.next()){  int n = resultSet2.getInt("number");  PreparedStatement preparedStatement = connection.prepareStatement(sql3);  preparedStatement.setInt(1,n+1);  preparedStatement.setString(2,name);  preparedStatement.setString(3,account);  preparedStatement.setString(4,id);  preparedStatement.setString(5,dateFormat.format(date));  preparedStatement.setString(6,seat);  preparedStatement.setString(7,"A");  preparedStatement.setString(8,class\_id);  preparedStatement.execute();  preparedStatement.close();  }  o.setVisible(true);  Warning\_info warning\_info = new Warning\_info("预定成功");  resultSet2.close();    //票数减一  int cur\_number = a\_number-1;  String sql4 = "update classes set a\_number ='" +cur\_number +"' where class\_id='"+class  PreparedStatement preparedStatement = connection.prepareStatement(sql4);  preparedStatement.execute();  }  resultSet1.close();    o.setVisible(false);  }  catch (SQLException e1){  e1.printStackTrace();  }  }  });  }  }); |