



华中科技大学

数据库系统原理实践报告

项目名称： 电影院信息管理系统
姓 名： 练炳诚
专 业： 计算机科学与技术
班 级： CS1705 班
学 号： U201714707
指导教师： 左琼

分数	
教师签名	

2020 年 6 月 22 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
2 软件功能学习部分	2
2.1 任务要求	2
2.2 完成过程	2
2.3 任务总结	5
3 SQL 练习部分	6
3.1 任务要求	6
3.2 完成过程	9
3.3 任务总结	28
4 综合实践任务	29
4.1 系统设计目标	29
4.2 需求分析	29
4.3 总体设计	31
4.4 数据库设计	33
4.5 详细设计与实现	36
4.6 系统测试	46
4.7 系统设计与实现总结	56
4 课程总结	58
附录	59

1 课程任务概述

1. 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。
2. 熟练掌握 SQL 的数据定义、数据操纵和数据控制语言的运用。
3. 熟悉数据库应用系统的设计方法和开发过程。

2 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，要求要有通过数据库的软件功能进行的备份和通过文件形式的脱机备份。
- 2) 练习在新增的数据库上增加用户并配置权限的操作。

2.2 完成过程

2.2.1 练习 DBMS 的备份方式

本次实验使用的是达梦 DBMS。

数据库实例中，模式 PERSON 中存在表 ADDRESS_TYPE，如图 2.1 所示。

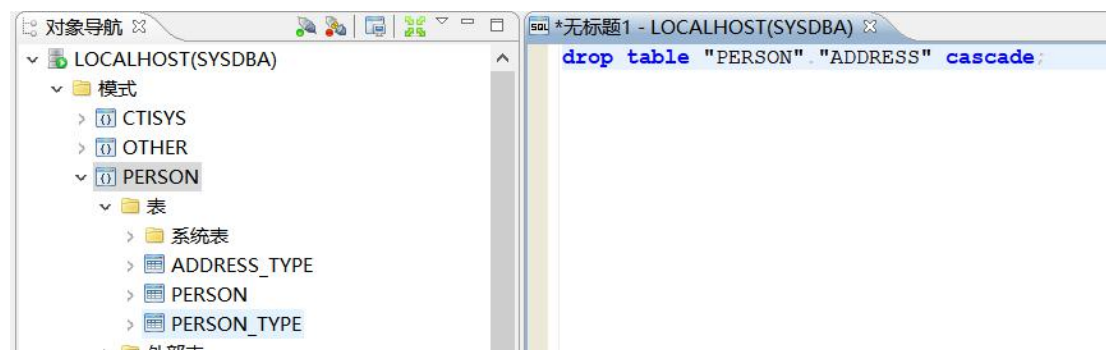


图 2.1 数据库实例中的表 ADDRESS_TYPE

通过 DM 的 tool 软件来备份数据库，首先通过 DM 服务查看器关闭 DmServiceServer，同时启动 DmAPService。如图 2.2 所示。

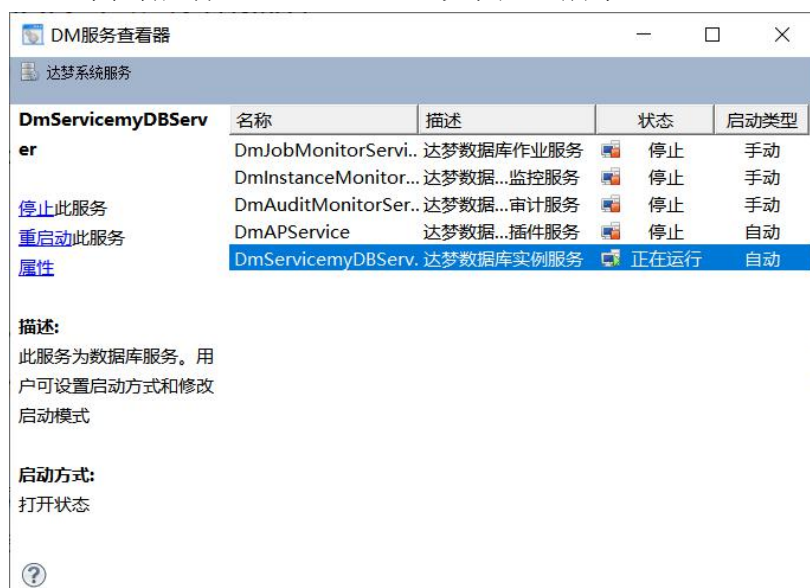


图 2.2 DM 服务查看器

然后，通过 DM 控制台工具进行数据库备份，点击新建备份，在 dmdbms/d
ata/bak/目录下新建备份集 bak1_by_software，备份集名相同，选择完全备份，如
图 2.3 所示。

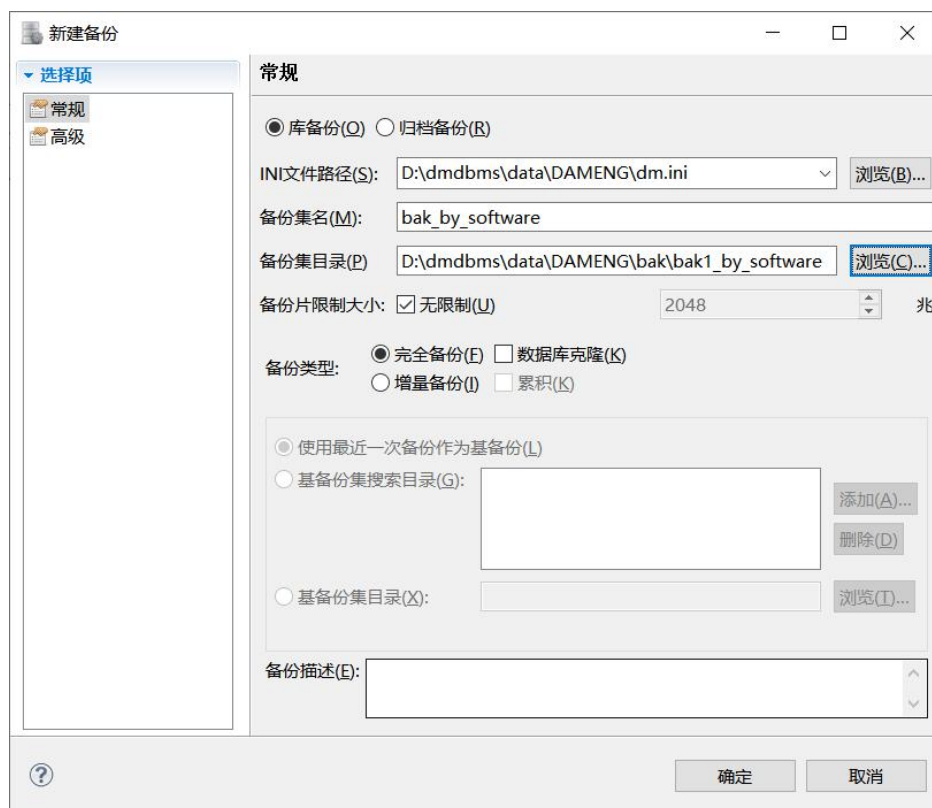


图 2.3 新建备份

创建备份后，为验证备份恢复是否成功，重新启动 DmServiceServer，在 DM
管理工具中级联删除表 ADDRESS_TYPE，删除后如图 2.4 所示。

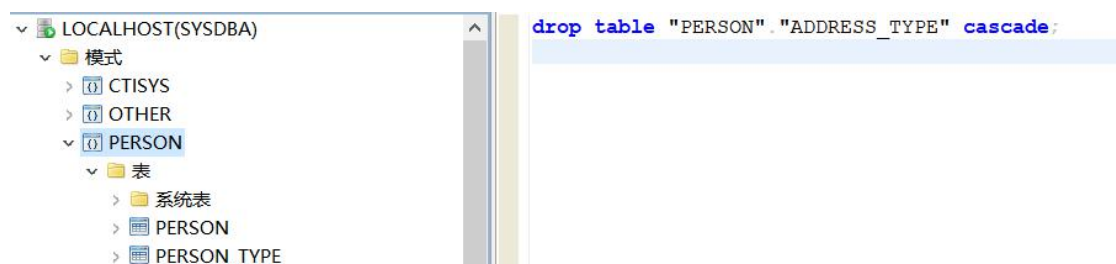


图 2.4 删除表 ADDRESS_TYPE

关闭 DmServiceServer，在 DM 控制台工具中，选择刚刚创建的备份文件，
对数据库进行还原、恢复。完成后，启动 DmServiceServer，并在 DM 管理工具
中查看 PERSON 模式下的表，如图 2.5 所示。

ADDRESS_T...	NAME
1	发货地址
2	送货地址
3	家庭地址
4	公司地址
*	<!NOT NULL, <!NOT NULL

图 2.5 恢复备份后的表

可以看到，恢复备份后 PERSON 模式下的表 ADDRESS_TYPE 复原了，说明恢复备份成功。

通过文件的形式进行脱机备份，先停止数据库服务，然后拷贝\dmdbms\data\目录下所有文件(包括日志文件和数据文件)进行脱机备份，拷贝成功后删除文件夹 data，并将拷贝的文件重命名为 data，重新启动数据库服务，在 DM 管理工具中连接数据库，如图 2.6 所示。

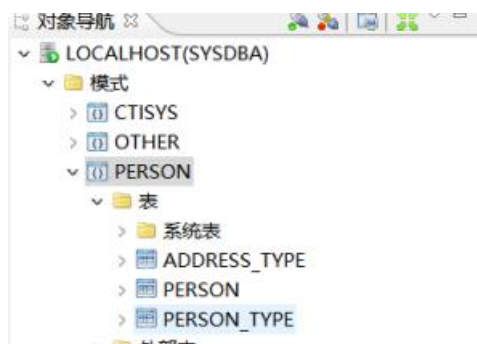


图 2.6 脱机备份结果

结果表明，通过文件形式的脱机备份(完全备份)一样能达到备份效果。

2.2.2 练习增加用户并配置权限的操作

在 DM 管理工具中，对数据库实例创建一个新的管理用户 NEWUSER，将 RESOURCE 角色赋予该用户，不允许该用户转授权限，SQL 语句如下：

```
create user "NEWUSER" identified by "qq2244444"
grant "RESOURCE" to "NEWUSER";
```

创建该角色后，在 DM 管理工具中查看该数据库的管理用户，如图 2.7 所示。

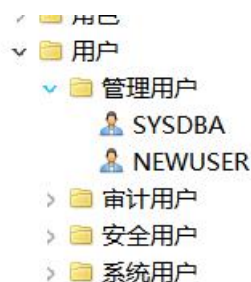


图 2.7 数据库管理用户列表

由于用户 NEWUSER 没有创建用户的权限，故切换至 NEWUSER 登陆数据库时，尝试创建用户 NEXTUSER，结果如图 2.8 所示。

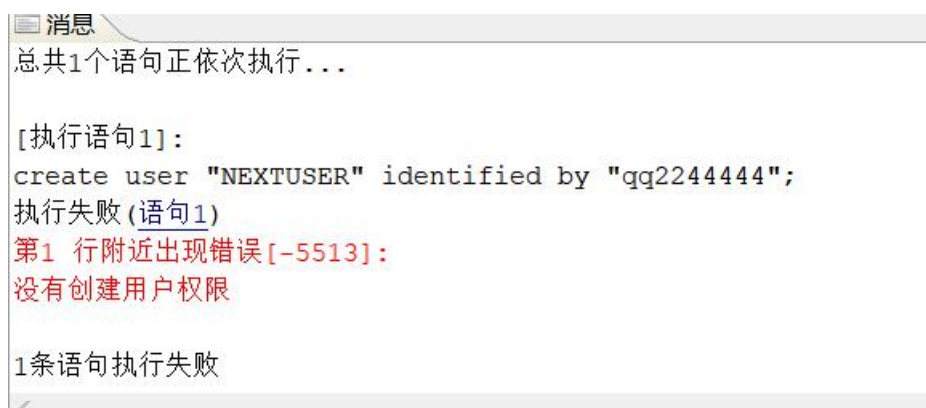


图 2.8 用户 NEWUSER 尝试创建新用户

而 RESOURCE 角色具有创建表的权限，用户 NEWUSER 创建表结果如图 2.9 所示。

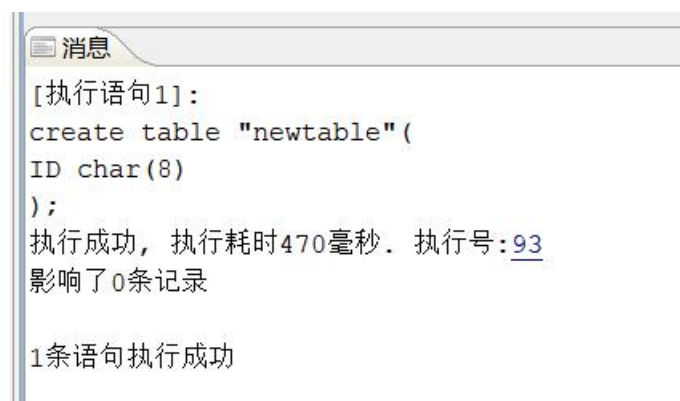


图 2.9 用户 NEWUSER 尝试创建表

说明用户权限授予正确。

2.3 任务总结

本次实验的任务主要是帮助熟悉数据库软件的使用，包括基本的数据库备份方式和角色创建方式，完成较为顺利，其中遇到的问题是第一次创建备份的时候失败了，后来在网上查询后才知道在使用软件功能进行备份时，由于达梦数据库默认需要开启 DmAP 服务，且进行备份时需要关闭数据库服务，而创建库后数据库服务是默认开启的，需要关闭服务。另一个问题是通过备份还原数据库时需要先还原，再恢复，仅还原而没有恢复时连接数据库会报错：数据库已还原，还应当被恢复。还原与恢复的区别是还原是将备份集中的有效数据页重新写入目标数据文件的过程。恢复则是指通过重做归档日志，将数据库状态恢复到备份结束时的状态。

3 SQL 练习部分

3.1 任务要求

3.1.1 建表

1) 创建下列跟“疫期乘坐列车”相关的关系，包括主码和外码的说明

车站表【车站编号，车站名，所属城市】

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

其中，主码为车站编号。

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

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

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

车程表【列车流水号，车站序号，车站编号，到达时刻，离开时刻】

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

其中，主码为(TID,SNo)。SID 来源于车站表的 SID。

乘客表【乘客身份证号，姓名，性别，年龄】

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

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

乘车记录表【记录编号，乘客身份证号，列车流水号，出发站编号，到达站编号，车厢号，席位排号，席位编号，席位状态】

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

其中，主码、外码请依据应用背景合理定义。

CarrigeID 若为空，则表示“无座”；

SeatNo 只能取值为‘A’、‘B’、‘C’、‘E’、‘F’，或为空值；

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

诊断表【诊断编号，病人身份证号，诊断日期，诊断结果，发病日期】

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

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

乘客紧密接触者表【接触日期，被接触者身份证号，状态，病患身份证号】

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

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

2) 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，并在实验报告中简述过程和结果。

3) 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。

3.1.2 数据更新

1) 分别用一条 sql 语句完成对乘车记录表基本的增、删、改的操作；

2) 批处理操作

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

3) 数据导入导出

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

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。

5) 创建视图

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

但是没坐车，不计入在内）。

6) 触发器实验

编写一个触发器，用于实现以下完整性控制规则：

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

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

3.1.3 查询

请分别用一条 SQL 语句完成下列各个小题的查询需求：

- 1) 查询确诊者“张三”的在发病前 14 天内的乘车记录；
- 2) 查询所有从城市“武汉”出发的乘客乘列车所到达的城市名；
- 3) 计算每位新冠患者从发病到确诊的时间间隔（天数）及患者身份信息，并将结果按照发病时间天数的降序排列；
- 4) 查询“2020-01-22”从“武汉”发出的所有列车；
- 5) 查询“2020-01-22”途经“武汉”的所有列车；
- 6) 查询“2020-01-22”从武汉离开的所有乘客的身份证号、所到达的城市、到达日期；
- 7) 统计“2020-01-22”从武汉离开的所有乘客所到达的城市及达到各个城市的武汉人员数。
- 8) 查询 2020 年 1 月到达武汉的所有人员；
- 9) 查询 2020 年 1 月乘车途径武汉的外地人员（身份证非“420”开头）；
- 10) 统计“2020-01-22”乘坐过‘G007’号列车的新冠患者在火车上的密切接触乘客人数（每位新冠患者的同车厢人员都算同车密切接触）。
- 11) 查询一趟列车的一节车厢中有 3 人及以上乘客被确认患上新冠的列车名、出发日期，车厢号；
- 12) 查询没有感染任何周边乘客的新冠乘客的身份证号、姓名、乘车日期；
- 13) 查询到达“北京”、或“上海”，或“广州”（即终点站）的列车名，要求 where 子句中除了连接条件只能有一个条件表达式；
- 14) 查询“2020-01-22”从“武汉站”出发，然后当天换乘另一趟车的乘客身份证号和首乘车次号，结果按照首乘车次号降序排列，同车次则按照乘客身份证号升序排列；
- 15) 查询所有新冠患者的身份证号，姓名及其 2020 年以来所乘坐过的列车名、发车日期，要求即使该患者未乘坐过任何列车也要列出来；
- 16) 查询所有发病日期相同而且确诊日期相同的病患统计信息，包括：发病

日期、确诊日期和患者人数，结果按照发病日期降序排列的前提下再按照确诊日期降序排列。

3.1.4 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。

3.1.5 DBMS 函数及存储过程和事务（选做）

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

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

3) 尝试在 DBMS 的交互式界面中验证事务机制的执行效果。

3.2 完成过程

以下步骤均在达梦 DBMS 下完成。

3.2.1 建表

1) 建表

按题目要求，建立所需的 7 个表，其中，对乘车记录表 TakeTrainRecord，设定主码为记录编号。创建表的 sql 语句如下：

```
CREATE TABLE "L714707"."Passenger"
(
  "PCardID" CHAR(18) NOT NULL,
  "PName" CHAR(20) NOT NULL,
  "Sex" BIT NOT NULL,
  "Age" SMALLINT NOT NULL,
  CLUSTER PRIMARY KEY("PCardID")) STORAGE(ON "MAIN", CLUSTERBTR);

CREATE TABLE "L714707"."Station"
(
  "SID" INT NOT NULL,
  "SName" CHAR(20) NOT NULL,
  "CityName" CHAR(20) NOT NULL,
  CLUSTER PRIMARY KEY("SID"),
  UNIQUE("SID")) STORAGE(ON "MAIN", CLUSTERBTR);

CREATE TABLE "L714707"."Train"
(
  "TID" INT NOT NULL,
  "SDate" DATE NOT NULL,
  "TName" CHAR(20) NOT NULL,
  "SStationID" INT,
  "AStationID" INT,
  "STime" DATETIME(6),
  "ATime" DATETIME(6),
  CLUSTER PRIMARY KEY("TID"),
  FOREIGN KEY("SStationID") REFERENCES "L714707"."Station"("SID"),
  FOREIGN KEY("AStationID") REFERENCES "L714707"."Station"("SID")) STORAGE(ON "MAIN", CLUSTERBTR);
```

```

CREATE TABLE "L714707"."TrainPass"
(
  "TID" INT NOT NULL,
  "SNo" SMALLINT NOT NULL,
  "SID" INT,
  "STime" DATETIME(6),
  "ATime" DATETIME(6),
  CLUSTER PRIMARY KEY("TID", "SNo"),
  FOREIGN KEY("SID") REFERENCES "L714707"."Station"("SID") STORAGE(ON "MAIN",
  CLUSTERBTR) ;

CREATE TABLE "L714707"."TakeTrainRecord"
(
  "RID" INT NOT NULL,
  "PCardID" CHAR(18),
  "TID" INT,
  "SStationID" INT,
  "AStationID" INT,
  "CarrigeID" SMALLINT,
  "SeatRow" SMALLINT,
  "SeatNo" CHAR(1),
  "SStatus" INT NOT NULL,
  CLUSTER PRIMARY KEY("RID"),
  UNIQUE("RID"),
  FOREIGN KEY("PCardID") REFERENCES "L714707"."Passenger"("PCardID"),
  UNIQUE("PCardID"),
  FOREIGN KEY("SStationID") REFERENCES "L714707"."Station"("SID"),
  FOREIGN KEY("AStationID") REFERENCES "L714707"."Station"("SID"),
  CHECK("SeatNo" IN ('A', 'B', 'C', 'D', 'E', 'F')),
  CHECK("SStatus" IN ('0', '1', '2')) STORAGE(ON "MAIN", CLUSTERBTR) ;

CREATE TABLE "L714707"."DiagnoseRecord"
(
  "DID" INT NOT NULL,
  "PCardID" CHAR(18),
  "DDay" DATE,
  "DStatus" SMALLINT,
  "FDay" DATE,
  CLUSTER PRIMARY KEY("DID"),
  FOREIGN KEY("PCardID") REFERENCES "L714707"."Passenger"("PCardID"),
  CHECK("DStatus" IN ('1', '2', '3')) STORAGE(ON "MAIN", CLUSTERBTR) ;

CREATE TABLE "L714707"."TrainContactor"
(
  "CDate" DATE NOT NULL,
  "CCardID" CHAR(18) NOT NULL,
  "DStatus" SMALLINT NOT NULL,
  "PCardID" CHAR(18) NOT NULL,
  CLUSTER PRIMARY KEY("CDate", "CCardID", "DStatus", "PCardID"),
  CHECK("DStatus" IN ('1', '2', '3')) STORAGE(ON "MAIN", CLUSTERBTR) ;

CREATE TABLE "L714707"."DiagnoseRecord"
(
  "DID" INT NOT NULL,
  "PCardID" CHAR(18),
  "DDay" DATE,
  "DStatus" SMALLINT,
  "FDay" DATE,
  CLUSTER PRIMARY KEY("DID"),
  CHECK("DStatus" IN ('1', '2', '3')) STORAGE(ON "MAIN", CLUSTERBTR) ;

```

创建完毕后，DM 管理工具中查看表如图 3.1 所示。

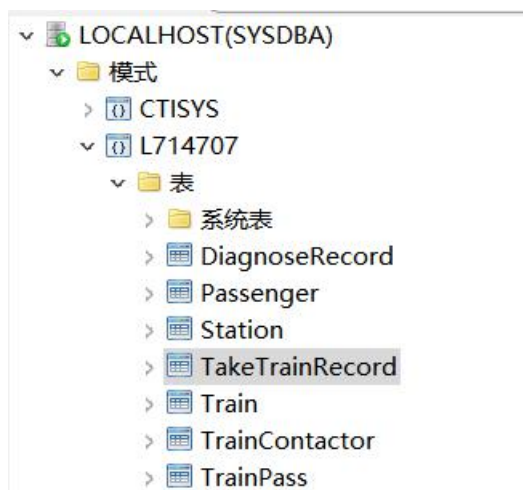


图 3.1 创建表结果

可以看到表创建在模式 L714707 下。

2) 观察性实验

当创建表时，测试外码未参照被参考关系的主码，分两种情况，一是声明该码是外码，但未声明参考的关系；另一种情况是声明该码是外码，参考了另一个关系的非主码，两种情况的创建结果分别如图 3.2、3.2 所示。

```
CREATE TABLE "L714707"."Train2"
(
  "TID" INT NOT NULL,
  "SDate" DATE NOT NULL,
  "TName" CHAR(20) NOT NULL,
  "SStationID" INT,
  "AStationID" INT,
  "STime" DATETIME(6),
  "ATime" DATETIME(6),
  CLUSTER PRIMARY KEY("TID"),
  FOREIGN KEY("SStationID")
  STORAGE (ON "MAIN", CLUSTERBTR) ;
执行失败 (语句1)
第 12 行, 第 8 列[STORAGE]附近出现错误[-2007]:
语法分析出错

1条语句执行失败
```

图 3.2 外码参考验证结果 (1)

```
CREATE TABLE "L714707"."Train4"
(
  "TID" INT NOT NULL,
  "SDate" DATE NOT NULL,
  "TName" CHAR(20) NOT NULL,
  "SStationID" INT,
  "AStationID" INT,
  "STime" DATETIME(6),
  "ATime" DATETIME(6),
  CLUSTER PRIMARY KEY("TID"),
  FOREIGN KEY("SStationID") REFERENCES "L714707"."Train"("SStationID"))
STORAGE(ON "MAIN", CLUSTERBTR) ;
执行失败 (语句1)
第12 行附近出现错误[-2707]:
被引用表[Train]引用索引不存在

1条语句执行失败
```

图 3.3 外码参考验证结果（2）

结果说明在建立外码时，必须要参考被参考关系的主码，否则将报错。

3) 数据准备

根据提供的参考数据，根据查询验证实验的要求，向各表中录入一定量的数据，由于子任务查询中会用到，故此处不再展示。

3.2.2 数据更新

1) 增、删、改

向表 TakeTrainRecord 中插入一条记录，并查询该表，sql 语句如下：

```
insert into "L714707"."TakeTrainRecord"("RID", "PCardID", "TID", "SStationID", "AStationID",
"CarrigeID", "SeatRow", "SeatNo", "SStatus")
VALUES(1,110101193412052028,1,null,null,1,1,'E',1);
```

结果如图 3.4 所示。

	RID	PCardID	TID	SStati...	AStati...	Carrig...	SeatRow	SeatNo	SStatus
	INT	CHAR(18)	INT	INT	INT	SMALLINT	SMALLINT	CHAR(1)	INT
1	1	110101193412052028	1	NULL	NULL	1	1	E	1

图 3.4 数据插入结果

修改其中的列 SeatNo 为 F 和列 SStatus 为 2，sql 语句如下：

```
insert into "L714707"."TakeTrainRecord"("RID", "PCardID", "TID", "SStationID", "AStationID",
"CarrigeID", "SeatRow", "SeatNo", "SStatus")
VALUES(1,110101193412052028,1,null,null,1,1,'E',1);
```

结果如图 3.5 所示。

RID	PCardID	TID	SStati...	AStati...	Carrig...	SeatRow	SeatNo	SStatus
INT	CHAR(18)	INT	INT	INT	SMALLINT	SMALLINT	CHAR(1)	INT
1	110101193412052028	1	NULL	NULL	1	1	F	2

图 3.5 数据修改结果

删除该条记录，sql 语句如下：

```
delete from "L714707"."TakeTrainRecord"
where "RID" = 1;
```

结果如图 3.6 所示。

RID	PCardID	TID	SStati...	AStati...	Carrig...	SeatRow	SeatNo	SStatus
INT	CHAR(18)	INT	INT	INT	SMALLINT	SMALLINT	CHAR(1)	INT

图 3.6 数据删除结果

2) 批处理操作

基于建表时导入的部分数据，sql 语句如下：

```
CREATE TABLE "L714707"."WH_TakeTrainRecord"
(
  "RID" INT NOT NULL,
  "PCardID" CHAR(18),
  "TID" INT,
  "SStationID" INT,
  "AStationID" INT,
  "CarrigeID" SMALLINT,
  "SeatRow" SMALLINT,
  "SeatNo" CHAR(1),
  "SStatus" INT NOT NULL,
  CLUSTER PRIMARY KEY("RID"),
  UNIQUE("RID"),
  FOREIGN KEY("PCardID") REFERENCES "L714707"."Passenger"("PCardID"),
  UNIQUE("PCardID"),
  FOREIGN KEY("SStationID") REFERENCES "L714707"."Station"("SID"),
  FOREIGN KEY("AStationID") REFERENCES "L714707"."Station"("SID"),
  CHECK("SeatNo" IN ('A', 'B', 'C', 'D', 'E', 'F')),
  CHECK("SStatus" IN ('0', '1', '2')) STORAGE(ON "MAIN", CLUSTERBTR);

insert into "L714707"."WH_TakeTrainRecord"
select "RID", "PCardID", "TID", "SStationID", "AStationID", "CarrigeID", "SeatRow", "SeatNo", "SStatus"
from "L714707"."TakeTrainRecord", "L714707"."Station"
where "SStationID" = "SID" and "CityName" = '武汉';
```

查询表 WH_TakeTrainRecord，结果如图 3.7 所示。

	RID	PCardID	TID	SStationID	AStationID	CarrigeID	SeatRow	SeatNo	SStatus
	INT	CHAR(18)	INT	INT	INT	SMALLINT	SMALLINT	CHAR(1)	INT
1	1	320115195511102613	1	1610	5	1	3	A	1
2	17	520422195504012051	4	1610	6	1	13	B	1
3	31	450224197105030482	7	1610	2	1	5	A	1
4	68	450222193204084936	14	1610	8	1	5	C	1
5	102	330305194210290919	20	1610	13	1	9	B	1

图 3.7 批处理结果

结果说明插入正确。

3) 数据导入导出

将上一步创建的表 WH_TakeTrainRecord 导出到操作系统文件，基于 DM 管理工具图形化界面操作，如图 3.8 所示。

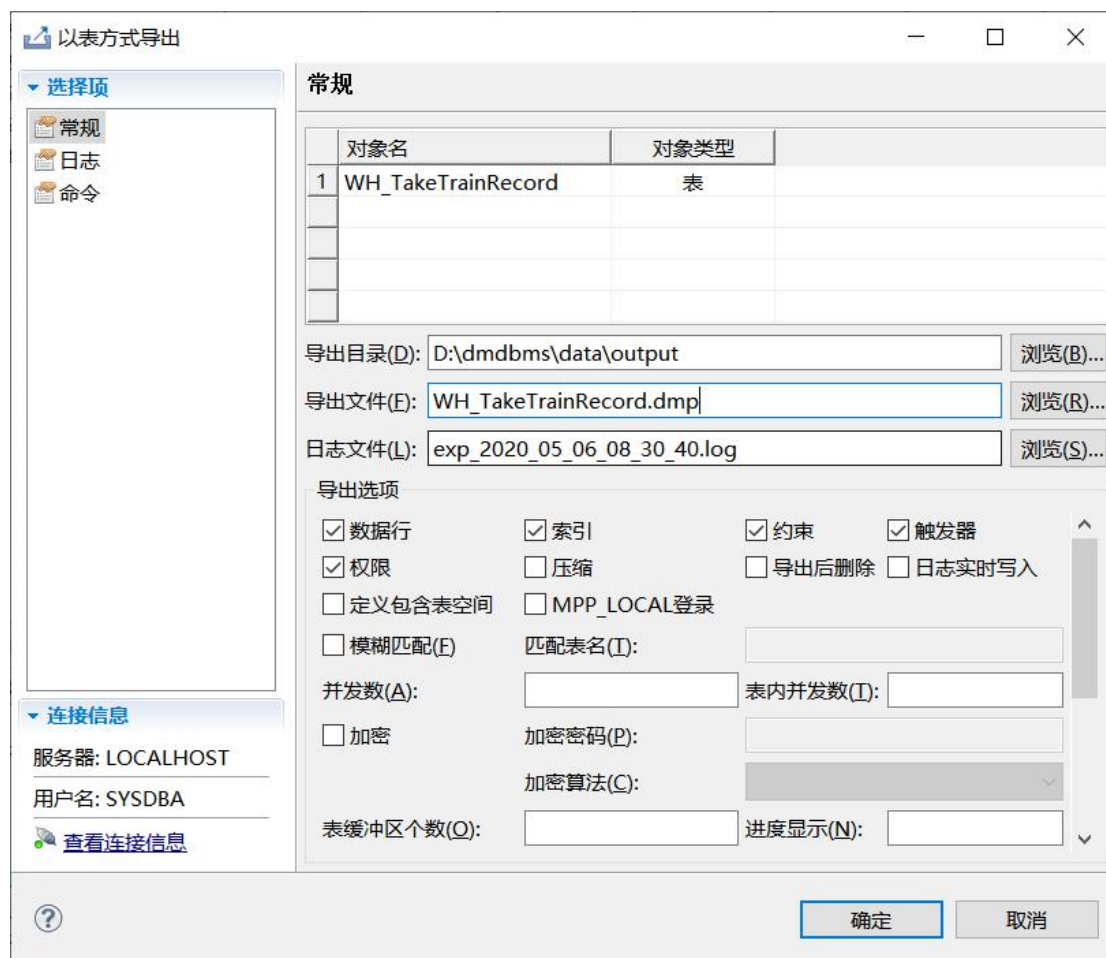


图 3.8 数据导出操作

导出后，用 `delete` 语句删除表中的数据，并将导出的数据导入表，在导入选项时，由于只需导入数据，故导入选项只勾选数据行，并在表存在时的操作选择 APPEND 表示将数据添加到空表 WH_TakeTrainRecord，如图 3.9 所示。

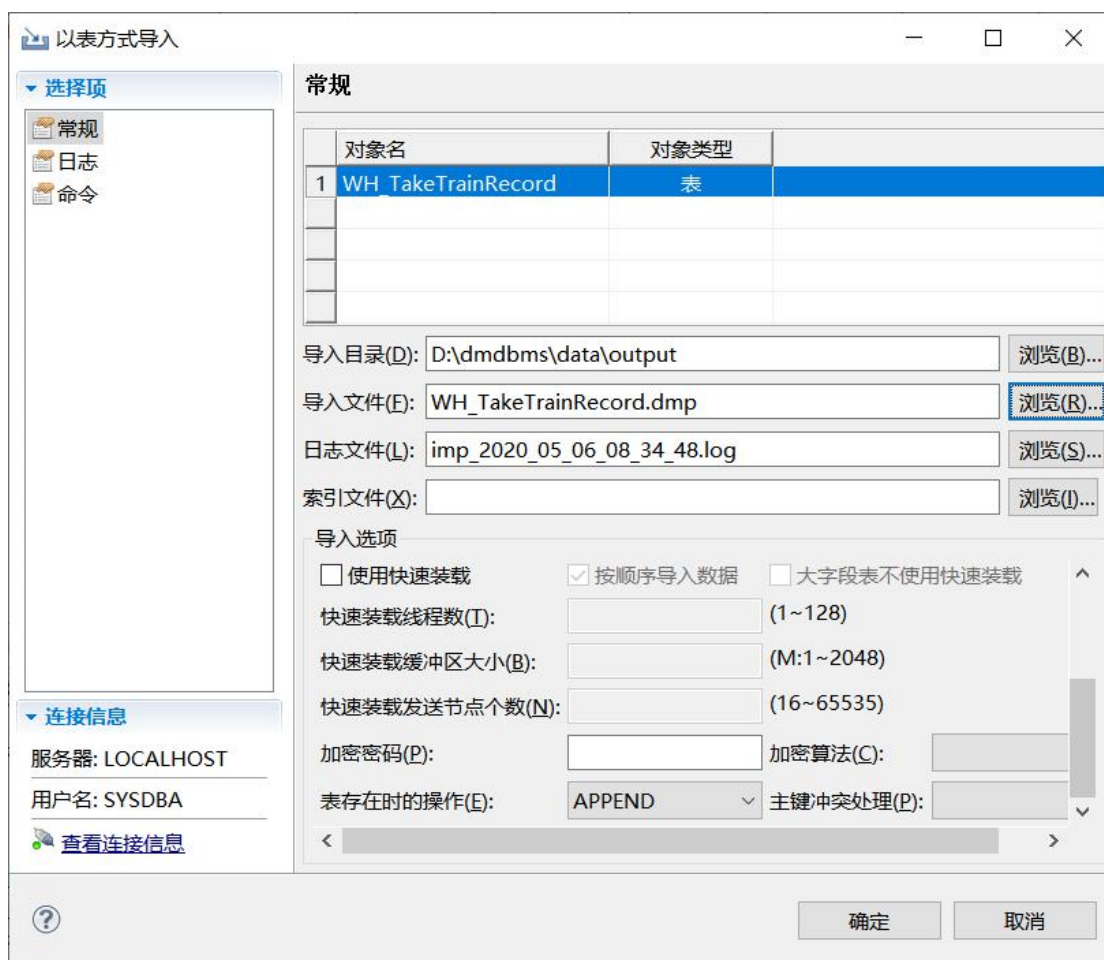


图 3.9 数据导入操作

经过验证，导入的数据与导出前一致，说明数据导出导入成功。

4) 观察性实验

创建一个关系 test，属性列为 A、B、C，sql 语句如下：

```
CREATE TABLE "L714707"."test"
(
  "A" CHAR(10),
  "B" CHAR(10),
  "C" CHAR(10)) STORAGE(ON "MAIN", CLUSTERBTR);
```

向关系中重复插入元组('a','b','c')三次，插入后 test 关系的数据如图 3.10 所示。

	A CHAR(10)	B CHAR(10)	C CHAR(10)
1	a	b	c
2	a	b	c
3	a	b	c

图 3.10 向关系 test 插入重复元组

对该关系进行修改操作，将列 A 属性值为 a 的元组对应属性修改为 A，修改后结果如图 3.11 所示。

	A CHAR(10)	B CHAR(10)	C CHAR(10)
1	A	b	c
2	A	b	c
3	A	b	c

图 3.11 修改元组结果

将列 A 属性值为 A 的元组删除，删除后结果如图 3.12 所示。

	A CHAR(10)	B CHAR(10)	C CHAR(10)
*	<!NULL>	<!NULL>	<!NULL>

图 3.12 删除元组结果

结果表明，对未设定主码的关系，可以插入重复的元组，且对该关系的实例进行修改、删除时，对满足目标的所有重复元组会执行相同操作。

5) 创建视图

根据要求，设计创建视图 sql 语句如下：

```
CREATE OR REPLACE VIEW "L714707"."XG_VIEW" as
Select
"L714707"."Passenger"."PCardID","L714707"."Passenger"."PName",
"L714707"."Passenger"."Age","L714707"."TakeTrainRecord"."TID",
"L714707"."TakeTrainRecord"."CarrigeID","L714707"."TakeTrainRecord"."SeatRow",
"L714707"."TakeTrainRecord"."SeatNo","L714707"."Train"."SDate"
from
"L714707"."Passenger","L714707"."TakeTrainRecord",
"L714707"."Train","L714707"."DiagnoseRecord"
where
"L714707"."DiagnoseRecord"."DStatus" = '1'
and "L714707"."DiagnoseRecord"."PCardID" = "L714707"."TakeTrainRecord"."PCardID"
and "L714707"."TakeTrainRecord"."PCardID" = "L714707"."Passenger"."PCardID"
and "L714707"."TakeTrainRecord"."TID" = "L714707"."Train"."TID"
and "L714707"."TakeTrainRecord"."SStatus" = '1'
order by
"L714707"."Passenger"."PCardID",
"L714707"."Train"."SDate" desc;
```

为验证视图正确性，设计插入数据 sql 语句如下：

```
insert into "L714707"."Station" values(1597,'武汉','湖北武汉');
insert into "L714707"."Station" values(1615,'汉口','湖北武汉');
insert into "L714707"."Station" values(128,'北京南','北京丰台区');
insert into "L714707"."Station" values(129,'北京西','北京丰台区');
insert into "L714707"."Station" values(34,'合肥南','安徽合肥');

insert into "L714707"."Passenger" values('150621196302132688','艾朝瑛',0.57);
insert into "L714707"."Passenger" values('330402199505011250','安东',1.25);
insert into "L714707"."Passenger" values('230125201305192995','白巨星',1.7);
insert into "L714707"."Passenger" values('320925194103132161','包琴',0.79);

insert into "L714707"."Train" values(1,'2020-01-22','G1',1597,129,'2020-01-22 09:10:00','2020-01-22 13:25:10');
insert into "L714707"."Train" values(2,'2020-01-22','G2',1615,129,'2020-01-22 12:05:00','2020-01-22 17:04:52');
insert into "L714707"."Train" values(3,'2020-01-22','G3',34,128,'2020-01-22 13:27:00','2020-01-22 17:29:56');
insert into "L714707"."Train" values(4,'2020-01-23','G4',129,1597,'2020-01-23 09:11:00','2020-01-23 13:21:10');

insert into "L714707"."TakeTrainRecord" values(1,'330402199505011250',1,1597,129,6,8,'F',1);
insert into "L714707"."TakeTrainRecord" values(2,'150621196302132688',1,1597,129,10,7,'E',1);
insert into "L714707"."TakeTrainRecord" values(3,'230125201305192995',2,1615,129,3,5,'F',1);
insert into "L714707"."TakeTrainRecord" values(4,'320925194103132161',3,34,128,13,9,'C',1);
insert into "L714707"."TakeTrainRecord" values(5,'330402199505011250',4,129,1597,1,4,'A',1);
insert into "L714707"."TakeTrainRecord" values(6,'330402199505011250',4,129,1597,1,5,'B',2);

insert into "L714707"."DiagnoseRecord" values(1,'330402199505011250','2020-01-23',1,'2020-1-25');
insert into "L714707"."DiagnoseRecord" values(2,'150621196302132688','2020-01-23',3,null);
insert into "L714707"."DiagnoseRecord" values(3,'320925194103132161','2020-01-23',1,'2020-1-25');
```

其中，乘客安东和包琴分别是确诊患者和康复患者，其余乘客在诊断表中没有记录或者是疑似确诊。视图应该查看出这两位乘客的具体信息，查询结果如图 3.13 所示。

XG_VIEW 视图查询数据.sql - LOCALHOST(SYSDBA)								
	PCardID CHAR(18)	PName CHAR(20)	Age SMALLINT	TID INT	CarrigeID SMALLINT	SeatRow SMALLINT	SeatNo CHAR(1)	SDate DATE
1	320925194103132161	包琴	79	3	13	9	C	2020-01-22
2	330402199505011250	安东	25	4	1	4	A	2020-01-23
3	330402199505011250	安东	25	1	6	8	F	2020-01-22

图 3.13 视图查询结果

查询结果按身份证号升序排列，且安东的两条乘车记录按发车日期降序排列，满足要求。

6) 触发器实验

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

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

触发器 1 的 sql 语句如下：

```

create or replace trigger "L714707"."AUTO_UPDATE"
after INSERT or UPDATE
on "L714707"."DiagnoseRecord"
referencing OLD ROW AS "OLD" NEW ROW AS "NEW"
for each row
BEGIN
    if(new."DStatus" = 1)
    then insert into
"L714707"."TrainContactor" ("CDate", "CCardID", "DStatus", "PCardID")
    select distinct
to_date(x."ATime", 'yyyy-mm-dd'), n."PCardID", '2', new."PCardID"
    from "L714707"."TakeTrainRecord" m, "L714707"."TakeTrainRecord"
n, "L714707"."TrainPass" x, "L714707"."TrainPass" y, "L714707"."TrainPass" z
    where m."PCardID" = new."PCardID" and
m."SStationID" = x."SID" and
m."TID" = y."TID" and
m."AStationID" = y."SID" and
n."TID" = z."TID" and
n."TID" = m."TID" and
z."SNo" >= x."SNo" and
z."SNo" <= y."SNo" and
m."CarrigeID" = n."CarrigeID" and
n."SeatRow" >= (m."SeatRow"-1) and
n."SeatRow" <= (m."SeatRow"+1) and
m."PCardID" != n."PCardID" and
m."SStatus" = '1' and
n."SStatus" = '1' and
DATEDIFF(day, to_date(x."ATime", 'yyyy-mm-dd'), new."FDay") <= 14;
update "L714707"."TrainContactor"
set "DStatus" = 1
where "L714707"."TrainContactor"."CCardID" in(
    select "PCardID"
    from "L714707"."DiagnoseRecord" d
    where d."DStatus" = 1
);
end if;
END;

```

实现思路是根据患者的乘车记录表中的起点站和终点站信息，查找跟其乘坐同一列车、同一车厢、前后排及邻座的人，通过车程表中患者乘坐列车起点站的SNo 筛选在其发病前 14 天的密切接触者，先将这些密切接触者信息以状态 2(疑似)加入，后面更新其状态信息，查诊断表里面这些密切接触者是否本身已确诊，若是则将其状态更新为 1 表示确诊。

为验证触发器功能的正确性，设计下列数据，图 3.14 是 TakeTrainRecord 中的数据，查询密切接触者表内容如图 3.15 所示。

10	10	440703200808303668	7	403	1599	8	10	F	1
11	11	350628196910241396	7	403	1599	8	9	A	1
12	12	654003198410044971	7	403	1599	8	1	C	1

图 3.14 触发器 1 验证时乘车记录表内容

CDate	CCardID	DStatus	PCardID
DATE	CHAR (18)	SMALLINT	CHAR (18)

图 3.15 更新患者信息表时密切接触表为空

现将身份证 350 开头的人确诊，再查询密切接触者表内容，如图 3.16 所示。

	CDate DATE	CCardID CHAR(18)	DStatus SMALLINT	PCardID CHAR(18)
1	2020-01-22	340711199102281656	2	350628196910241396
2	2020-01-22	440703200808303668	2	350628196910241396

图 3.16 经触发器更新的密切接触表

结果表明触发器生效，将与该人密切接触的两位乘客加入密切接触表。

创建触发器 2 的 sql 语句如下：

```
create or replace trigger "L714707"."COMFIRMED"
after INSERT or UPDATE of "DStatus"
on "L714707"."DiagnoseRecord"
referencing OLD ROW AS "OLD" NEW ROW AS "NEW"
for each row
BEGIN
    /*触发器体*/
    IF(new."DStatus" = 1)
    THEN UPDATE "L714707"."TrainContactor"
    set "DStatus" = 1
    where "CCardID" = new."PCardID";
    END IF;
END;
```

实现思路是对诊断表的每一行，若更新后为确诊状态，则更新密切接触表，使其状态变为确诊。

延续上一步的密切接触表，将身份证 340 开头的人设定为确诊，查询密切接触表得到的结果如图 3.17 所示。

	CDate DATE	CCardID CHAR(18)	DStatus SMALLINT	PCardID CHAR(18)
1	2020-01-22	340711199102281656	1	350628196910241396
2	2020-01-22	350628196910241396	1	340711199102281656
3	2020-01-22	440703200808303668	2	340711199102281656
4	2020-01-22	440703200808303668	2	350628196910241396

图 3.17 更新确诊信息后的密切接触表

可以看到，身份证 340 开头的乘客的状态变为 1，同时，触发了第一个触发器，将另外两个密切接触者加入该表，其中身份证 440 开头的乘客与 350 开头的乘客均被加入密切接触表，且 350 开头的乘客由于已经是确诊者，则其状态为 1，进一步验证了触发器 1 的正确性。

3.2.3 查询

1) 查询确诊者“张三”的在发病前 14 天内的乘车记录。

采用多表连接查询方案，根据获取确诊者张三的身份证号及发病日期，在乘车记录表中查询日期在发病日期 14 天内的张三的乘车信息，sql 语句如下：


```

select distinct ttr."TID"
from "L714707"."Train" t,"L714707"."TrainPass"
tp,"L714707"."TakeTrainRecord" ttr,"L714707"."DiagnoseRecord"
dr,"L714707"."Passenger" p
where ttr."TID" = t."TID" and tp."TID" = t."TID"
and p."PName" = '张三'
and p."PCardID" = dr."PCardID" and dr."PCardID" = ttr."PCardID"
and (tp."STime" IS NOT NULL and
days_between(add_days(tp."STime",14),dr."FDay") <=14)
or (tp."STime" IS NULL and
days_between(add_days(tp."ATime",14),dr."FDay") <=14);

```

查询结果如下：



TID
1

图 3.18 查询 1 结果

数据集中，张三乘坐的车有 1 号和 8 号，而张三是 1 月 22 日发病，列车 1 是 1 月 21 日发车，列车 8 是 2 月 1 日发车，满足要求。

2) 查询所有从城市“武汉”出发的乘客乘列车所到达的城市名；

sql 语句如下：

```

select distinct q."CityName"
from "L714707"."Station" p,"L714707"."Station"
q,"L714707"."TakeTrainRecord" ttr
where p."SID" = ttr."SStationID" and p."CityName" = '武汉' and q."SID" =
ttr."AStationID" and "SStatus" = 1

```

实现思路是多表连接查询，选择出发车站所属城市为武汉的列车到达的城市名，查询结果如下：



CityName
丰台区
长沙
黄石

图 3.19 查询 2 结果

列车表中，从武汉出发的列车有 3 列，分别到达北京西站、长沙南站、大冶北站，所属城市分别是丰台区、长沙、黄石，满足要求。

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

sql 语句如下：

```

select days_between("DDay","FDay"), "PCardID"
from "L714707"."DiagnoseRecord" d
where d."DStatus" = 1
order by days_between("DDay","FDay") desc

```

即查询诊断表中 DStatus 为 1 的，按发病天数降序排列。查询结果如下：

	DAYS_BETWEEN("DDay", "FDay") INTEGER	PCardID CHAR(18)
1	24	222222222222222222
2	3	350628196910241396
3	3	123456789987654321
4	1	340711199102281656

图 3.20 查询 3 结果

4) 查询“2020-01-22”从“武汉”发出的所有列车；

sql 语句如下：

```
select "TID", "TName"
from "L714707"."Train" t, "L714707"."Station" s
where t."SStationID" = s."SID" and s."CityName" = '武汉' and t."SDate" <
to_date('2020-1-23', 'yyyy-mm-dd')
```

查询结果如下：

	TID INT	TName CHAR(20)
1	2	列车6 ...
2	3	列车1 ...
3	4	列车2 ...
4	5	列车5 ...

图 3.21 查询 4 结果

经过比对，查询结果均为当日从武汉出发的列车。

5) 查询“2020-01-22”途经“武汉”的所有列车；

sql 语句如下：

```
select t."TID", "TName"
from "L714707"."Train" t, "L714707"."TrainPass" tp, "L714707"."Station" s
where t."TID" = tp."TID" and tp."SID" = s."SID" and s."CityName" = '武汉' and
t."SDate" < to_date('2020-1-23', 'yyyy-mm-dd')
```

查询结果如下：

	TID INT	TName CHAR(20)
1	1	列车4 ...
2	2	列车6 ...
3	3	列车1 ...
4	4	列车2 ...
5	5	列车5 ...
6	6	列车3 ...
7	7	G007 ...

图 3.21 查询 5 结果

经过比对，仅列车 8、9 不是 1-22 日出发的，查寻结果正确。

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

sql 语句如下：

```
select distinct "PCardID", q."CityName", tp."ATime"
from "L714707"."Station" p, "L714707"."Station"
q, "L714707"."TakeTrainRecord" ttr, "L714707"."TrainPass" tp
where ttr."SStationID" = p."SID" and p."CityName" = '武汉' and ttr."TID" =
tp."TID" and
ttr."SStationID" = tp."SID" and ttr."AStationID" = q."SID" and
ttr."SStatus" = 1 and
((tp."STime" IS NOT NULL and '2020-1-22' =
to_date(tp."STime", 'yyyy-mm-dd')) or
(tp."STime" IS NULL and '2020-1-22' =
to_date(tp."ATime", 'yyyy-mm-dd')));
```

实现思路是查询所有从武汉离开的乘客，且其乘车时间是 1 月 22 日，为了区别起点站和中间站，需判断 STime 是否为 NULL，查询结果如下：

	PCardID CHAR(18)	CityName CHAR(30)	ATime DATETIME(6)
1	370522194808092081	长沙	2020-01-22 06:45:00.000000
2	430381193112280763	黄石	2020-01-22 07:43:00.000000
3	230125199406192556	黄石	2020-01-22 07:43:00.000000

图 3.22 查询 6 结果

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

sql 语句如下：

```
select q."CityName", COUNT(*)
from "L714707"."Station" p, "L714707"."Station"
q, "L714707"."TakeTrainRecord" ttr, "L714707"."TrainPass" tp
where ttr."SStationID" = p."SID" and
p."CityName" = '武汉' and
ttr."TID" = tp."TID" and
ttr."AStationID" = tp."SID" and
ttr."AStationID" = q."SID" and
ttr."SStatus" = 1 and
to_date(tp."STime", 'yyyy-mm-dd') = '2020-1-22'
group by q."CityName"
```

实现思路是多表连接查询，查询起点站在武汉的乘客其终点站城市，统计结果通过 group by 语句合并，查询结果如下：

	CityName CHAR(30)	COUNT(*) BIGINT
1	长沙	1
2	黄石	2

图 3.23 查询 7 结果

与查询 6 结果对比说明查询正确。

8) 查询 2020 年 1 月到达武汉的所有人员;

sql 语句如下:

```
select "PName",p."PCardID"
from "L714707"."Passenger" p,"L714707"."TakeTrainRecord"
ttr,"L714707"."Station" s,"L714707"."TrainPass" tp
where ttr."ASStationID" = s."SID" and
      s."CityName" = '武汉' and
      p."PCardID" = ttr."PCardID" and
      tp."TID" = ttr."TID" and
      tp."SID" = ttr."ASStationID" and
      to_date(tp."STime",'yyyy-mm-dd') < '2020-2-24';
```

查询结果如下:

	PName CHAR(20)	PCardID CHAR(18)
1	蔡鹏程	340711199102281656
2	廖舒波	654003198410044971
3	廖金金	350628196910241396
4	顾少莹	440703200808303668

图 3.24 查询 8 结果

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

Sql 语句如下:

```
select distinct "PName",p."PCardID"
from "L714707"."Passenger" p,"L714707"."TakeTrainRecord"
ttr,"L714707"."TrainPass" x,"L714707"."TrainPass" y,"L714707"."TrainPass"
z,"L714707"."Station" s
where ttr."PCardID" not like '420%' and
      ttr."SStatus" = '1' and
      ttr."TID" = x."TID" and
      ttr."TID" = y."TID" and
      ttr."SStationID" = x."SID" and
      ttr."ASStationID" = y."SID" and
      z."TID" = ttr."TID" and
      z."SID" = s."SID" and
      s."CityName" = '武汉' and
      z."SNo" >= x."SNo" and
      z."SNo" <= y."SNo" and
      (to_char(z."STime",'yyyy-mm') = '2020-01' or
      to_char(z."ATime",'yyyy-mm') = '2020-01') and
      p."PCardID" = ttr."PCardID";
```

实现思路是多表连接查询, 其中 x、y 表分别是起点站编号和终点站编号, 根据 SNo 范围搜索 2020 年 1 月乘车 SNo 在范围内且该站在武汉的, 身份证不以 420 开头的人员, 为区分起点站、终点站和中间站, 对时间的判断通过 or 连接, 查询结果如下:

	PName CHAR(20)	PCardID CHAR(18)
1	张三	123456789987654321
2	蔡鹏程	340711199102281656
3	廖金金	350628196910241396
4	鲍顽园	44162219690811478X
5	顾少莹	440703200808303668
6	廖舒波	654003198410044971
7	安娜	370522194808092081
8	卞云开	230125199406192556
9	毕亚男	430381193112280763

图 3.25 查询 9 结果

10) 统计“2020-01-22”乘坐过‘G007’号列车的新冠患者在火车上的密切接触乘客人数（每位新冠患者的同车厢人员都算同车密切接触）。

sql 语句如下：

```
select count(distinct n."PCardID")
from "L714707"."TakeTrainRecord" m, "L714707"."TakeTrainRecord" n,
     "L714707"."TrainPass" a, "L714707"."TrainPass" b, "L714707"."TrainPass"
tp1, "L714707"."TrainPass" tp2,
     "L714707"."Train" t, "L714707"."DiagnoseRecord" d
where m."TID" = t."TID" and
      t."TName" = 'G007' and
      m."SStatus" = 1 and
      m."PCardID" = d."PCardID" and
      d."DStatus" = 1 and
      d."FDay" <= '2020-01-22' and
      --以上表示乘坐 G007 的确诊患者的乘车记录表 m
      (m."TID" = a."TID" and m."SStationID" = a."SID" and
to_date(a."ATime", 'yyyy-mm-dd') = '2020-01-22' or
      m."TID" = b."TID" and m."AStationID" = b."SID" and
to_date(b."STime", 'yyyy-mm-dd') = '2020-01-22') and
      --以上表示 a 为表 m 的起始站表, b 为表 m 的终点站表, 时间在 2020-1-22
      n."TID" = m."TID" and
      n."CarrigeID" = m."CarrigeID" and
      n."PCardID" != m."PCardID" and
      n."SStatus" = 1 and
      --以上表示与确诊患者乘坐同一列车同一车厢的旅客列表, 不一定有重叠时间段
      n."TID" = tp1."TID" and
      n."SStationID" = tp1."SID" and
      n."TID" = tp2."TID" and
      n."AStationID" = tp2."SID" and
      (tp1."ATime" >= a."ATime" and tp1."ATime" <= b."STime" or tp2."STime" >=
a."ATime" and tp2."STime" <= b."STime");
      --以上表示这些旅客和患者共处同一车厢
```

实现思路是多表连接查询，在 where 子句中先写入乘坐 G007 列车的确诊患者的选择条件，再写出选择其于 2020 年 1 月 22 日的乘车起始站和终点站条件，再然后写出选择与其同日同车厢的旅客条件，最后写出这些旅客与确诊患者共处该车厢的条件，查询结果如下：

COUNT(...)	BIGINT
1	4

图 3.26 查询 10 结果

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

sql 语句如下：

```
select distinct t."TName",t."SDate",ttr."CarrigeID"
from "L714707"."Train" t,"L714707"."TakeTrainRecord" ttr
where (t."TID",ttr."CarrigeID") in(
    select ttr."TID",ttr."CarrigeID"
    from "L714707"."TakeTrainRecord" ttr,"L714707"."DiagnoseRecord" d
    where ttr."PCardID" = d."PCardID" and d."DStatus" = 1
    group by(ttr."TID",ttr."CarrigeID")
    having COUNT(*) >= 3);
```

实现思路是嵌套子查询，将一趟列车的一节车厢有 3 人及以上旅客确诊的列车编号、车厢号分组，再查询该列车编号对应的列车名及出发日期，查询结果如下：

TName	SDate	Carrig...
CHAR(20)	DATE	SMALLINT
1 G007	2020-01-22	8

图 3.27 查询 11 结果

12) 查询没有感染任何周边乘客的新冠乘客的身份证号、姓名、乘车日期；

sql 语句如下：

```
select distinct p."PCardID",p."PName",tp."ATime"
from "L714707"."Passenger" p,"L714707"."TakeTrainRecord"
ttr,"L714707"."TrainPass" tp,"L714707"."DiagnoseRecord" d
where p."PCardID" = d."PCardID" and
    d."DStatus" = 1 and
    p."PCardID" = ttr."PCardID" and
    ttr."TID" = tp."TID" and
    ttr."SStationID" = tp."SID" and
    not exists(
        select *
        from "L714707"."TrainContactor" x
        where x."PCardID" = p."PCardID" and x."DStatus" = 1);
```

实现思路是先查询新冠患者及其乘坐的列车，然后通过 not exists 判断该型患者的密切接触者是否确诊，若返回 true 则该患者未感染任何周边乘客。

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

sql 语句如下:


```
select distinct "TName"
from "L714707"."Train"
where "AStationID" in (select SID
from "L714707"."Station"
where "CityName" in ('北京','上海','广州'));
```

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

sql 语句如下:

```
select distinct x."PCardID",m."TID"
from "L714707"."Station" s,"L714707"."TakeTrainRecord"
x,"L714707"."TakeTrainRecord" y,"L714707"."TrainPass"
m,"L714707"."TrainPass" n
where x."SStationID" = s."SID" and
x."SStatus" = '1' and
s."SName"='武汉' and
--x 为从武汉站出发的乘车记录表
x."TID" = m."TID" and
x."SStationID" = m."SID" and
to_date(m."ATime",'yyyy-mm-dd')='2020-01-22' and
--m 为 2020-1-22 途径武汉站的车程表
x."PCardID" = y."PCardID" and
x."RID" != y."RID" and
y."SStatus" = '1' and
--y 为同一人从武汉站出发后换乘的乘车记录表
y."TID" = n."TID" and
to_date(n."ATime",'yyyy-mm-dd')='2020-01-22' and
--n 为 y 相应的 2020-1-22 的车程表
n."ATime" > m."ATime"
order by m."TID" desc,x."PCardID";
```

实现思路是先得到从武汉站出发的乘车记录表,再得到 1 月 22 日途径武汉的车程表,再得到这类乘客的换乘记录表,再得到相应的车程表,通过后一个车程表中到达时间大于前一个车程表筛选出首乘车次号,查询结果如下:



	PCardID CHAR(18)	TID INT
1	430381193112280763	4

图 3.28 查询 14 结果

其中,该乘客的乘车记录如下:

7	7	430381193112280763	4	1597	1567	6	2	B	1
8	8	230125199406192556	4	1597	1567	3	7	F	1
9	9	340711199102281656	7	403	1599	8	10	E	1
10	10	440703200808303668	7	403	1599	8	10	F	1
11	11	350628196910241396	7	403	1599	8	9	A	1
12	12	654003198410044971	7	403	1599	8	1	C	1
13	13	110109196508260440	7	403	1599	8	8	A	1
14	14	430381193112280763	6	1668	3663	1	1	A	1

图 3.29 查询 14 的乘客乘车记录

可以看到该乘客首乘列车为 4,然后换成 6 号,查询正确。

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

sql 语句如下:

```
select p."PCardID", p."PName", t."TName", t."SDate"
from "L714707"."DiagnoseRecord" d left outer join
("L714707"."TakeTrainRecord" ttr left outer join "L714707"."Train" t
on (ttr."TID" = t."TID"))
on (d."PCardID" = ttr."PCardID"), "L714707"."Passenger" p
where p."PCardID" = d."PCardID" and
d."DStatus" = 1 and
(t."SDate" is not null and to_char(t."SDate", 'yyyy') = '2020' or
t."SDate" is null);
```

实现思路是多表连接查询, 由于要求患者即使未坐过车也要列出来, 则采用左外连接, 查询结果如下:

	PCardID CHAR(18)	PName CHAR(20)	TName CHAR(20)	SDate DATE
1	666666666666666666	666	.. NULL	NULL
2	123456789987654321	张三	.. 列车4	... 2020-01-21
3	123456789987654321	张三	.. 列车7	... 2020-02-01
4	222222222222222222	222	.. NULL	NULL
5	350628196910241396	廖金金	.. G007	... 2020-01-22
6	340711199102281656	蔡鹏程	.. G007	... 2020-01-22
7	110109196508260440	冯琼玲	.. G007	... 2020-01-22

图 3.30 查询 15 结果

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

sql 语句如下:

```
select "DDay", "FDay", count(*)
from "L714707"."DiagnoseRecord"
where "DStatus" = 1
group by "DDay", "FDay"
having count(*) > 1
order by "FDay" desc, "DDay" desc;
```

实现思路是直接使用 group by 将发病日期和确诊日期相同的病患分组, 分组条件是计数大于 1。查询结果如下:

	DDay DATE	FDay DATE	COUNT(*) BIGINT
1	2020-01-23	2020-01-21	2
2	2020-01-23	2020-01-20	2
3	2020-01-25	2020-01-01	2
4	2020-01-24	2020-01-01	2

图 3.31 查询 16 结果

确诊表内容如下:

	DID INT	PCardID CHAR(18)	DDay DATE	DStatus SMALLINT	FDay DATE
1	0	666666666666666666	2020-01-23	1	2020-01-21
2	1	123456789987654321	2020-01-25	1	2020-01-22
3	2	111111111111111111	NULL	3	NULL
4	3	222222222222222222	2020-01-25	1	2020-01-01
5	4	350628196910241396	2020-01-23	1	2020-01-20
6	5	340711199102281656	2020-01-22	1	2020-01-21
7	6	110109196508260440	2020-02-10	1	2020-02-02
8	7	333333333333333333	2020-01-25	1	2020-01-01
9	8	444444444444444444	2020-01-23	1	2020-01-20
10	9	555555555555555555	2020-01-23	1	2020-01-21
11	10	666666666666666666	2020-01-24	1	2020-01-01
12	11	777777777777777777	2020-01-24	1	2020-01-01
13	12	888888888888888888	2020-01-24	2	2020-01-01

图 3.32 查询 16 确诊表内容

说明查询查询结果正确。

3.3 任务总结

通过本次实验，掌握了 sql 语句操作数据库的整体流程，包括创建表、更新表、删除表、创建视图、创建触发器，以及对数据的增删改查，感觉本次任务难度较大，花费时间也很多，要自己去设计数据验证 sql 语句的正确性，且查询任务的部分语句如 9、10、14、15，还有创建触发器的第一个任务都很困难，花费了很多精力。但总地来说，现在写 sql 语句比之前熟练了不少。

4 综合实践任务

4.1 系统设计目标

4.1.1 应用背景

随着人们生活水平的提高，去电影院观看电影的人数也日渐增多，各电影院的规模也不断变大，提供各类电影和不同规格的影厅供消费者选择，如何方便地对电影院的各类信息进行管理成了必须解决的问题，一个电影院信息管理系统能够帮助电影院工作人员解决该问题。

4.1.2 总体目标

电影院信息管理系统应用于各电影院，售票员能通过该系统进行售票、退票的操作，管理员能够通过该系统进行增删电影、电影排挡、更换影厅的操作，超级管理员能够通过该系统添加管理员、售票员账户。

4.2 需求分析

4.2.1 功能需求

由于不同类别的人员管理的信息不同，故系统需具有登录功能，系统提供不同类别账号的登录功能，根据用户选择的具体类别和输入的账号密码，查询数据库进行登录判断。账号类型主要分为超级管理员、管理员、售票员。

超级管理员登录后，管理系统应提供对管理员、售票员账号的管理功能，可增加/删除账号，以及对某个账号的具体信息的修改，账号的信息有：姓名、性别、联系方式。

管理员账号登录后，可对影院的电影档期进行规划，如增加/删除电影，修改电影排挡信息（排档时间、播放影厅、票价）。

售票员账号登录后，可选择电影档期内的电影进行售票/退票，售票时需选择电影编号、票种类型（学生票、成人票）、坐席等信息；需要进行退票操作时，根据电影票编号进行退票操作。

系统功能需求见表 4.1。

表 4.1 系统功能需求汇总表

功能编号	功能名称	功能描述	用户类别
1	增加账号	添加一个管理员或售票员账号	超级管理员
2	删除账号	删除一个管理员或售票员账号	超级管理员
3	修改账号信息	修改管理员或售票员账号信息	超级管理员
4	增加电影	添加一个电影及其排挡信息	管理员

续表 4.1

功能编号	功能名称	功能描述	用户类别
5	删除电影	删除一个电影及其排挡信息，若电影已排入档期，则删除相应档期信息	管理员
6	上映/下架电影	对指定电影进行上映/下架，若下架电影已排入档期，则删除相应档期信息	管理员
7	添加影厅	添加一条影厅信息	管理员
8	删除影厅	删除一条影厅信息，若影厅已排入档期，则删除相应档期信息	管理员
9	启用/停用影厅	对指定影厅进行启用/停用，若停用影厅已排入档期，则删除相应档期信息	管理员
10	排挡	对指定日期进行排挡，能选择的电影和影厅均是上映和启用的，且排挡只能对当前日期及后七天进行	管理员
11	售票	选择具体一个档期及指定座位进行售票	售票员
12	退票	若电影票未使用，可进行退票	售票员

4.2.2 性能需求

该信息管理系统是面向单个电影院设计的，针对的用户群体是电影院工作人员，对于超级管理员的账号增删改功能，考虑到该功能实际情况下使用概率较小，无吞吐量及并发量要求；对管理员的电影档期信息增删改功能，通常情况下每天也只需操作几次，同样无吞吐量和并发量需求；对售票员的功能，考虑到一般电影院的售票窗口只有几个，买票时的需要进行排队，故对吞吐量也无需求，但对不同售票窗口有并发需求，系统需保证并发操作的可靠执行。

另外，系统需要高速响应不同的事件。

综上，系统的性能需求为响应速度快、高可靠性。

4.2.3 数据完整性需求

对账号信息，每个账号应该是独一无二的，且密码非空，长度不小于 6 位；对电影排挡信息，电影编号参照电影信息的编号，播放影厅编号参考影厅信息的

编号；售票信息中，售票编号为主码，电影编号、影厅编号参考排挡信息中的相应编号，坐席编号满足：对同一场电影，坐席编号唯一当且仅当该票有效（无退票）。

4.2.4 数据流图

根据功能需求分析，超级管理员产生账号信息数据，通过账号管理子系统传递给账号信息表，同时可以通过子系统读取某一账号信息。

管理员对电影排挡的功能划分为两个子系统实现，通过电影管理系统对影院放映的电影进行增删改，通过电影排挡子系统对电影排挡信息进行增删，同时可以通过该子系统读取某一条电影排挡信息，除此之外，管理员通过影厅管理子系统对影厅的信息进行修改。

售票员通过售票系统获取影厅信息和电影排挡信息，产生售票记录并计入售票信息表，退票时读取售票信息表，对该售票信息进行修改。

系统顶层数据流图如图 4.1 所示。

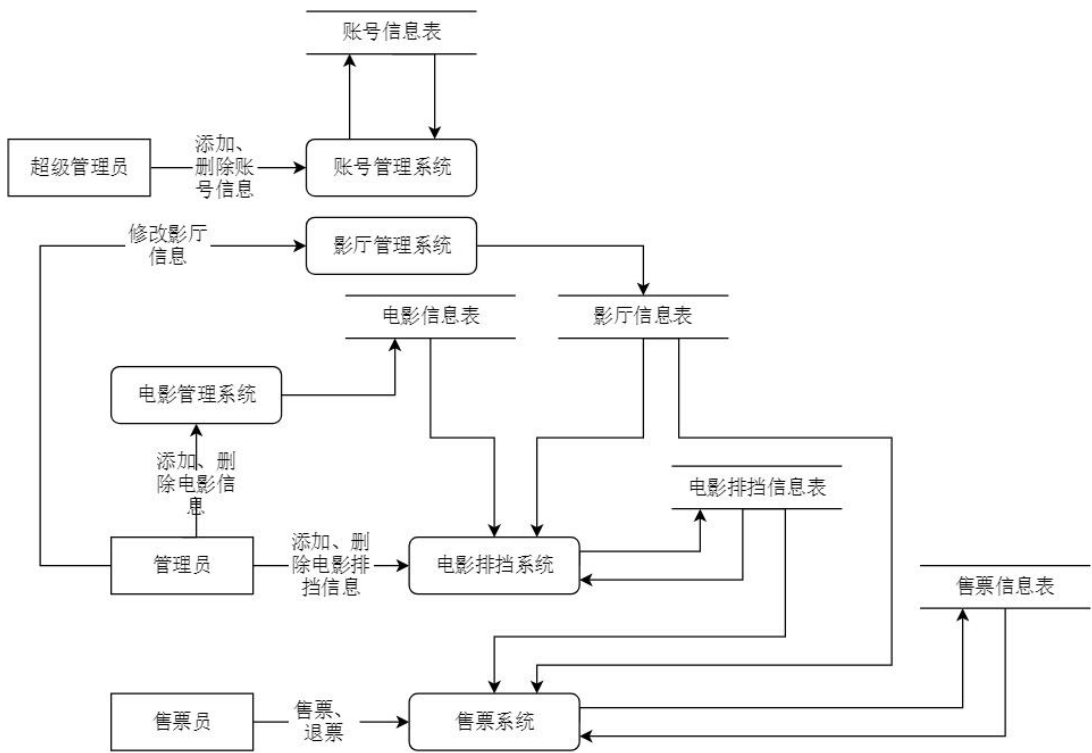


图 4.1 电影信息管理系统顶层数据流图

4.3 总体设计

考虑到管理系统主要由影院工作人员使用，面向 3 类用户（超级管理员、管理员、售票员），故考虑系统可采用 C/S 架构实现，客户端对 3 种类别用户采用通用的架构，根据登录用户类型选择展示不同的子系统，服务器端采用 MySQL 数据库及 DBMS。采用 JavaFX 作为界面开发库，Sence-Builder 作为辅助开发工

具，基于 MVC 开发模式实现，使用 IDEA 集成开发环境。

通过需求分析，设计系统架构图如图 4.2 所示。

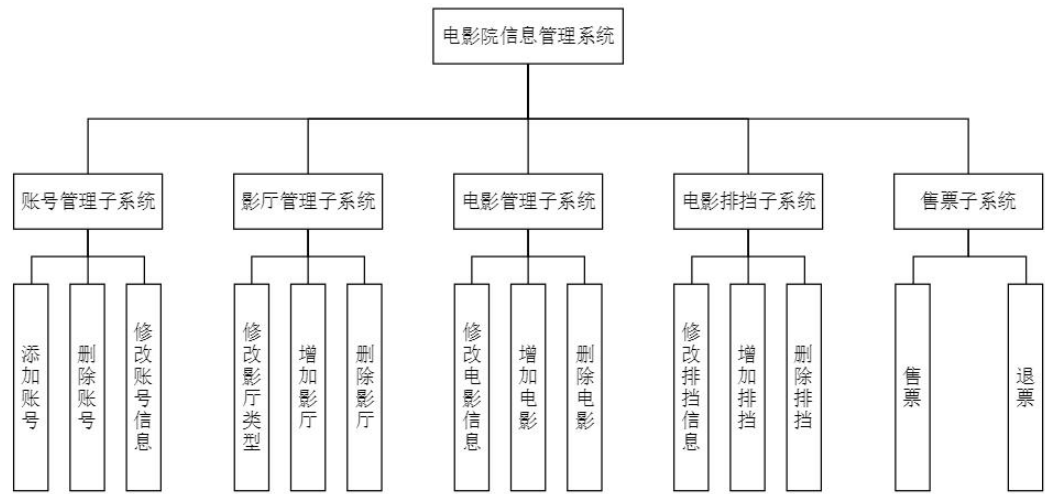


图 4.2 系统架构图

系统的总体流程是，首先进行登录判断，根据账号类型和输入的密码，若判断登录成功，则进入相应类别账号的功能模块；否则，提示账号或密码错误。

超级管理员登录后，可查看账号信息，增加账号，删除账号；管理员登录后，可增加增加、删除电影信息，增删改影厅信息，增删改电影排挡信息；售票员登录后，可增、改售票信息。

系统总体流程图如图 4.3 所示。

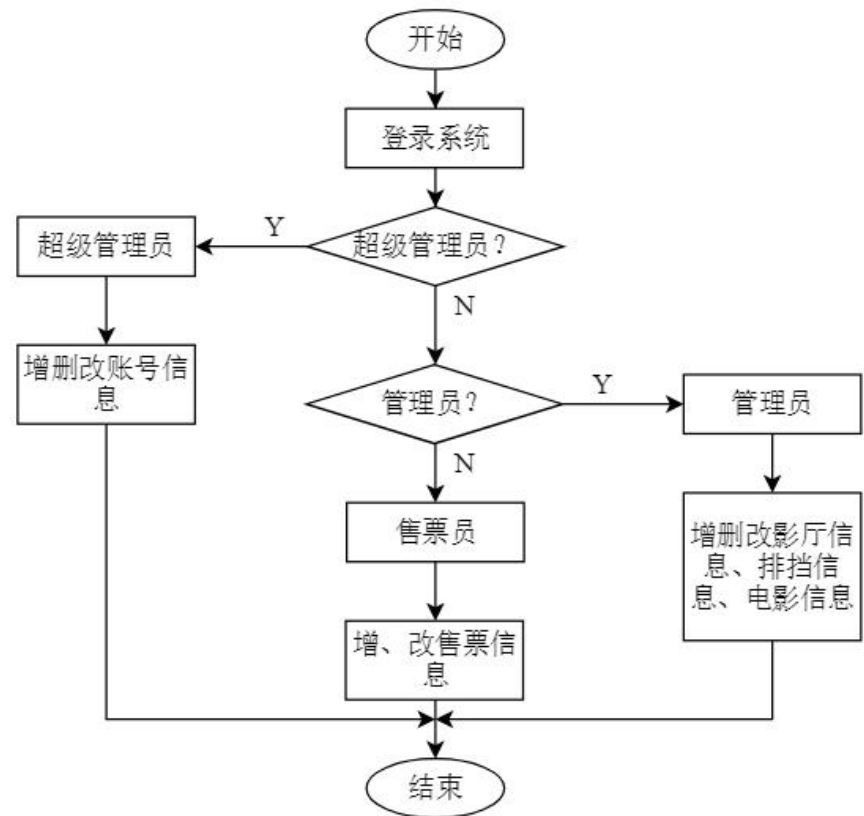


图 4.3 系统总体流程图

4.4 数据库设计

系统的全局 ER 图如图 4.4 所示。

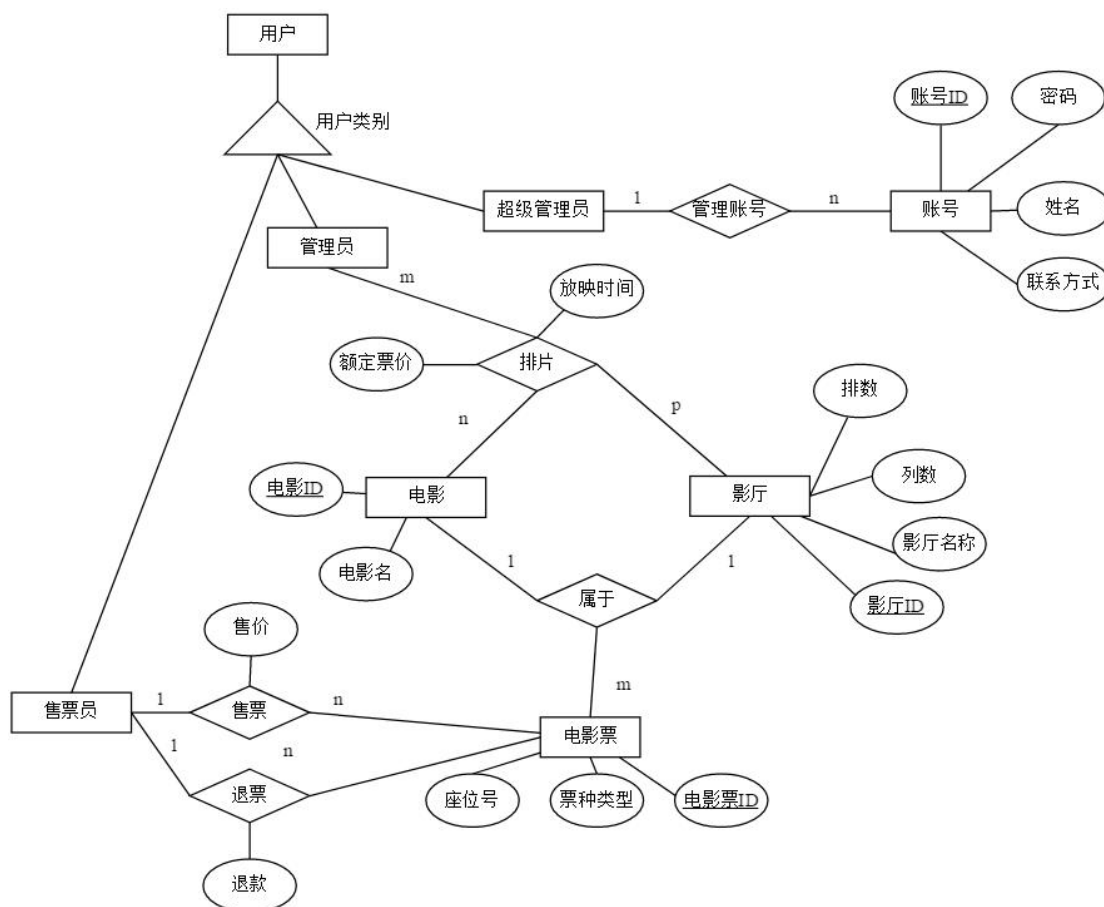


图 4.4 全局 ER 图

其中，用户类别有超级管理员、管理员、售票员，超级管理员管理账号信息，账号的属于有：账号 ID，密码，姓名，联系方式，主码是账号 ID，即用户登录时的账号；管理员管理影厅，影厅的属性有：影厅 ID，影厅名称，排数，列数，主码是影厅 ID，管理员管理电影，电影的属性有：电影 ID，电影名称，电影时长，主码是电影 ID，管理员管理档期信息，档期信息的属性有：档期 ID，电影 ID，影厅 ID，播出时间，额定票价，主码是档期 ID；售票员售票产生售票记录，售票记录的属性有：电影票 ID，档期 ID，票种类型，退票标志，主码是电影票 ID，售票员退票更新售票记录，退票的属性是电影票 ID。

转换成关系数据模型如下：

账号(账号 ID，密码，姓名，联系方式)

管理账号(超级管理员 ID，账号 ID，操作日期)

影厅(影厅 ID，影厅名称，排数，列数)

电影(电影 ID，电影名称，电影时长)

排片(影厅 ID，电影 ID，放映时间，额定票价)

电影票(电影票 ID, 票种类型, 排, 列)

属于(电影票 ID, 电影 ID, 影厅 ID)

售票(售票员 ID, 电影票 ID, 售价)

退票(售票员 ID, 电影票 ID)

建立账号信息表 tb_account, 影厅信息表 tb_movie_room, 电影信息表 tb_movie, 档期信息表 tb_schedule, 售票信息表 tb_ticket, 其内容及各字段说明分别见表 4.2-表 4.6。

表 4.2 账号信息表 tb_account

属性名	英文名	数据类型 (精度)	主码/ 外码	是否为空	允许重复	描述
账号	account	char(10)	主码	否	否	登录账号
密码	pwd	char(20)		否	是	登录密码
姓名	uname	char(10)		否	是	用户姓名
联系方式	phone	char(11)		是	是	用户手机号
账户类型	idtype	int(2)		否	是	0: 超级管理员 1: 管理员 2: 售票员
最后登录 时间	lastlogintime	datetime		是	是	账号的最后登录 日期时间

表 4.3 影厅信息表 tb_movie_room

属性名	英文名	数据类型 (精度)	主码/ 外码	是否为空	允许重复	描述
影厅 ID	roomid	int(10)	主码	否	否	影厅 ID
影厅名称	rname	char(20)		否	否	影厅名称
排数	row	smallint(3)		否	是	坐席排数
列数	column	smallint(3)		否	是	坐席列数

表 4.4 电影信息表 tb_movie

属性名	英文名	数据类型 (精度)	主码/ 外码	是否为空	允许重复	描述
电影 ID	movieid	int(10)	主码	否	否	电影 ID
电影名	mname	char(20)		否	否	电影名称
电影时长	mtime	time		否	是	电影的时长
是否上映	screen	boolean		否	是	0 表示电影下架, 1 表示上映

表 4.5 档期信息表 tb_schedule

属性名	英文名	数据类型 (精度)	主码/ 外码	是否为 空	允许重复	描述
档期 ID	schedule_id	int(10)	主码	否	否	档期 ID
电影 ID	movie_id	int(10)	外码	否	是	参照表 tb_movie
影厅 ID	room_id	int(10)	外码	否	是	参照表 tb_movie_room
开始时间	show_time	datetime		否	否	电影播出时间
结束时间	end_time	datetime		否	否	电影结束时间
播日期	show_date	date		否	否	电影播出日期
额定票价	normal_price	number(3,1)				额定票价

表 4.6 电影票信息表 tb_ticket

属性名	英文名	数据类型 (精度)	主码/ 外码	是否为空	允许重复	描述
票号	tid	int(10)	主码	否	否	标识电影票
行号	seat_row	int(10)		否	是	坐席行号
列号	seat_column	int(10)		否	是	坐席列号
档期 ID	schedule_id	int(10)	外码	否	是	参照表 tb_schedule
票种类型	type	bool		否	是	0 表示全票, 1 表示学生票
使用标志	use_flag	bool		否	是	0 表示未使用, 1 表示使用
实际售价	ap	number(3,1)				由票种类型和额定售价决定的实际售价

4.5 详细设计与实现

4.5.1 开发环境说明

系统采用 java 开发，使用 MySQL 本地数据库，开发环境如下：

操作系统：Windows 10，16G 内存，64 位

JDK：java-jdk-13.02

IDE：IntelliJ-IDEA-2019.3.4 x64

数据库：mysql-8.0.20

UI 开发：javafx-sdk-11.0.2

数据库连接驱动：mysql-connector-java-8.0.20

其中，MySQL 数据库的账号为 root，密码为 root，创建的数据库实例名为：L714707，使用默认端口 localhost:3306。

4.5.2 登录子系统

在登录子系统中，当用户选择登录时，判断账户输入框和密码输入框中是否均有内容，若没有输入则进行提示；否则，根据用户输入的账户密码以及选择的账户类型，查询账户表，若匹配其中一个元组，则根据其账户类型进入相应的功能界面，并更新其最近登录时间。为防止恶意试探账户表中的账户信息，对所有不匹配的登录操作均提示“账户或密码错误”（即使账户存在）。

根据需求分析，对用户类别进行了功能分层，超级管理员只管理账户信息，不能对电影进行排挡或者售票，管理员只进行电影管理、影厅管理和排挡，售票员只进行售票、退票。

在登录子系统中，涉及对表 tb_account 的查询、更新操作，相关 sql 语句的 java 代码实现见附录。

登录子系统的流程如图 4.5 所示。

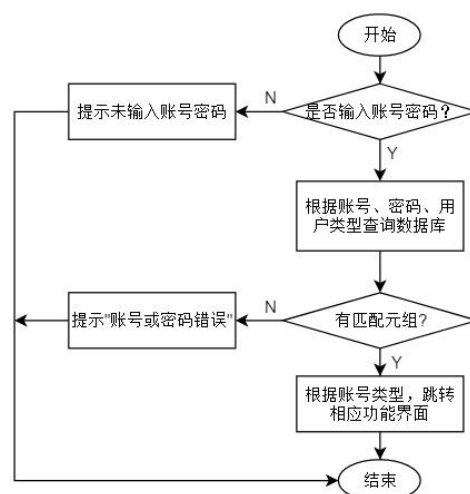


图 4.5 登录子系统流程图

登录子系统 UI 设计如图 4.6 所示。



图 4.6 登录子系统 UI 设计

4.5.3 账户管理子系统

账户管理子系统的业务流程为：超级管理员登录后，查询账号信息表，将所有账号信息显示到界面表格中，超级管理员可根据账号来进行账号信息的增删改。

增加账号时，需输入要添加账号的账号、密码、姓名等 3 个非空属性，并选择账号类型，根据输入信息构造账号信息元组尝试插入账号信息表，若插入失败（违反约束或数据库访问出错），则提示用户；否则，提示插入成功，并刷新账户列表界面。

修改账户时，首先需要输入指定账号，然后根据账户查询账户信息表，获取该账户的具体信息显示到文本框中，超级管理员可选择文本框中的具体一项内容进行修改，修改完毕后点击提交，此时系统会构造该账户信息元组，并在表上尝试更新操作，若更新失败（违反约束或数据库访问出错）则提示用户；否则，提示更新成功，并刷新账户列表界面。

删除账户时，根据输入的账号（主码）查表进行删除即可，删除失败（主码不存在或数据库访问异常）进行提示，否则提示删除成功，并刷新账户列表界面。

在账户管理子系统中，涉及对表 `tb_account` 的查询、插入、更新、删除操作，相关 `sql` 语句的 `java` 代码实现见附录。

账户管理子系统流程图如图 4.7 所示。

在对电影进行删除或下架时，考虑到指定的电影有可能已经排入档期列表，故在执行电影表的删除和下架后，需对档期表中包含指定电影的记录进行删除，根据表结构的定义，设计触发器如下：

```
-- 触发器 1：下架电影时，删除档期表对应的档期信息
CREATE TRIGGER trigger_unscreen_movie after update on tb_movie for each row
begin
    declare new_movie_id int;
    declare new_screen boolean;
    set new_movie_id = new.movieid;
    set new_screen = new.screen;
    if new_screen = 0 then
        delete from tb_schedule where movie_id = new_movie_id;
    end if;
end;

-- 触发器 2：删除电影时，删除档期表对应的档期信息
CREATE TRIGGER trigger_delete_movie after delete on tb_movie for each row
begin
    declare old_movie_id int;
    set old_movie_id = old.movieid;
    delete from tb_schedule where movie_id = old_movie_id;
end;
```

类似地，对影厅进行删除或停用时，若该影厅已排入档期，也需要删除包含该影厅的档期记录，设计触发器如下：

```
-- 触发器 3：停用影厅时，删除档期表对应的档期信息
CREATE TRIGGER trigger_stop_movie_room after update on tb_movie_room for each row
begin
    declare new_room_id int;
    declare new_useflag int;
    set new_room_id = new.roomid;
    set new_useflag = new.useflag;
    if new_useflag = 0 then
        delete from tb_schedule where room_id = new_room_id;
    end if;
end;

-- 触发器 4：删除影厅时，删除档期表对应的档期信息
CREATE TRIGGER trigger_delete_movie_room after delete on tb_movie_room for each row
begin
    declare old_room_id int;
    set old_room_id = old.roomid;
    delete from tb_schedule where room_id = old_room_id;
end;
```

这样设计的目的是考虑到实际应用背景，在之后的档期管理子系统中，可以限定能够选择的电影和影厅一定是上映和启用的，但若某部电影因演员犯法或其他特殊原因需要紧急下架时，或者影厅因为设备故障需要紧急停用时，就可直接对相应电影或影厅进行下架和停用操作，不用在档期表中一条条地删除记录。需要说明的是，影厅的增加和删除功能在实际应用背景中基本不会用到，因为一个电影院在建成时影厅数量已经固定，这里设计进去只是单纯地展示对表

4.5.5 档期管理子系统

档期管理子系统主要有添加档期、删除档期、复用档期 3 个功能。

在添加档期功能中，管理员首先选择需要排挡的日期，该日期的选择范围限定为当前系统日期到之后的七天以内，这样设计的原因是对过去的日期进行排挡是不合理的，由于影院每隔一段时间有新的电影引入，故对超过 7 天后的日期进行排挡也是不合理的。

选择完日期后，系统查询数据库，获取上映的电影列表和启用的影厅列表，管理员可选择一部电影、一个影厅，开始时间段，设定额定票价，选择完毕后系统构造一条档期信息记录，记录中结束时间由电影时长和开始时间自动计算产生。接着系统查询表 `tb_schedule` 中日期为选定日期、影厅 ID 分为选定影厅的 ID 的所有记录（即该影厅在当天的所有安排），进行时间冲突检测，若待插入档期信息的时间段与该影厅已安排的时间段有冲突，则提示用户，不进行插入；否则，将记录插入表 `tb_schedule`。

在时间冲突检测时，考虑到实际应用背景下，一部电影播放完毕后，会预留一段时间进行人员散场、卫生打扫，本系统中设定该预留时间为 5 分钟，即某影厅在播放完一部电影后，需间隔至少 5 分钟才能播放下一部电影。

时间冲突检测算法的原理为：当已知的时间段无冲突时，将待插入时间段的开始时间和结束时间与这些无冲突时间段的开始时间和结束时间分为数组 `BEGIN`、`END`，并进行升序排序，接着从 `BEGIN` 数组的第 $i+1$ 个元素开始（ i 从 1 开始，直到数组长度 `length`），与 `END` 数组的第 i 个元素比较，若 `BEGIN[i+1] >= END[i]`，则存在时间段冲突，算法返回 `false`；比较完毕后，算法返回 `true`。

时间冲突检测算法的伪代码如下：

```
initial:   begin[1..length] 为所有时间段的开始时间，end[1..length] 为所有时间段的结束时间，并进行升序排序
           normal_time = 5//预留时间，本系统中设定为 5 分钟
begin
for i = 1 to length - 1
    if begin[i+1] >= end[i] + normal_time
        return false;
return true;
end;
```

添加档期功能的流程图如下图 4.11 所示。

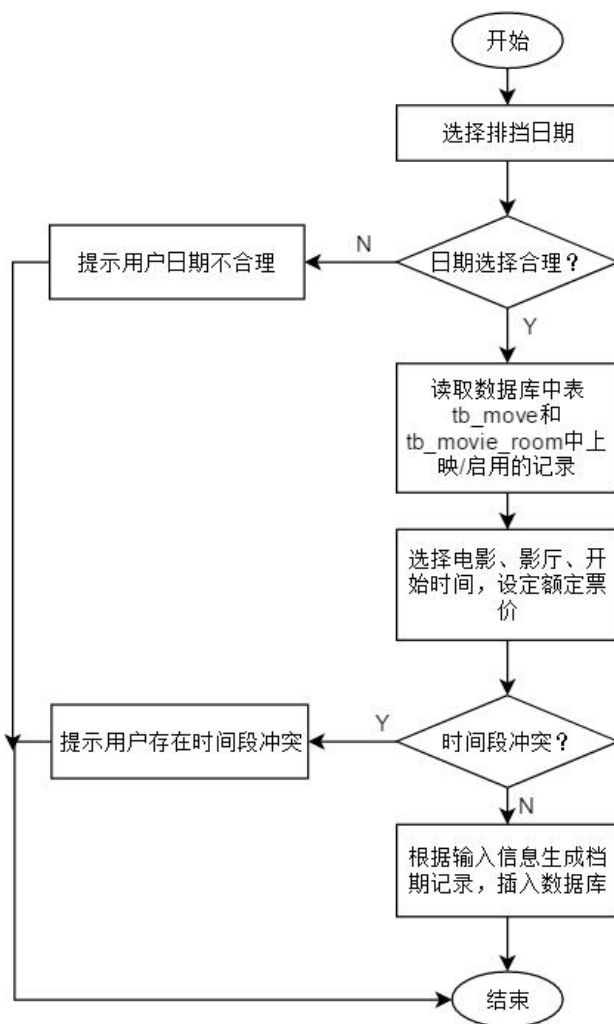


图 4.11 添加档期功能流程图

对于删除档期的功能，管理员输入档期 ID，系统首先判断输入是否是数字，若不是则提示用户；否则，根据输入的 ID 查询表 `tb_schedule`，删除指定档期 ID 的记录，删除失败则提示用户。

对于复用档期的功能，考虑如下应用背景：影院在一段时间内，由于没有新的电影上映或者下架，故连续几天的档期安排可能是相同的，仅仅是日期不同，这时管理员可直接选择某一天的档期安排进行复用，系统获取指定日期的所有档期安排记录，将日期替换为需要安排的档期日期，批量生成档期记录并插入表 `tb_shcedule`，而无需一条一条地插入档期记录。

档期管理子系统中，要显示电影名称以及获取电影的时长，故涉及对表 `tb_movie` 的查询操作，要添加/删除档期记录，复用档期记录，故涉及对表 `tb_scheduled` 的查询、添加、删除、更新操作，相应 sql 语句的 java 代码实现见附录。

档期管理子系统的 UI 设计如图 4.12 所示。



图 4.12 档期管理子系统 UI 设计

4.5.6 售票及退票子系统

为保证电影票的唯一性（即同一场次下同一坐席号只能出现一次），以及考虑道售票记录表的记录数量较大，故在表 `tb_ticket` 的属性组 `(schedule_id,seat_row,seat_column)`上建立了索引，这样可以保证数据的唯一性，同时也可在退票时提高查询电影票信息的效率。

对于售票功能，售票员首先选择售票的日期（即该电影票的使用日期），确定后，系统查询表 `tb_schedule`，获取该日期下安排的电影 ID 列表，并查表 `tb_movie` 获取电影名称进行显示。紧接着选择要观看的电影，确定后，系统查询表 `tb_schedule`，获取该电影在该日期下的安排信息，即时间段、影厅号。然后选择观影时间段，这里需要对信息进行过滤，系统根据当前系统时间和选择的目标时间进行比较，若目标时间已过期，则不显示该时间段信息。再然后根据用户选择的时间段，过滤影厅选项供用户选择。

在用户选择影厅号后，就已经确定了电影的场次（观影日期、电影、时间段、影厅），为了方便售票员根据用户需求选择坐席，系统会根据电影的场次信息查询表 `tb_movie_room` 和表 `tb_ticket`，动态渲染坐席，首先根据表 `tb_movie_room` 中查询到的影厅的排、列数动态添加坐席组件，然后根据表 `tb_ticket` 中同一电影场次下的电影票信息，将售出的坐席渲染为红色和不可选定状态。

渲染完坐席后，根据用户选择的坐席号序列，生成批量的电影票信息，用户选择票种类型后，根据生成电影票信息的数量（即购票数量）、该场次的额定票

价计算总价格（学生票打五折）。

购票功能的流程如图 4.13 所示。

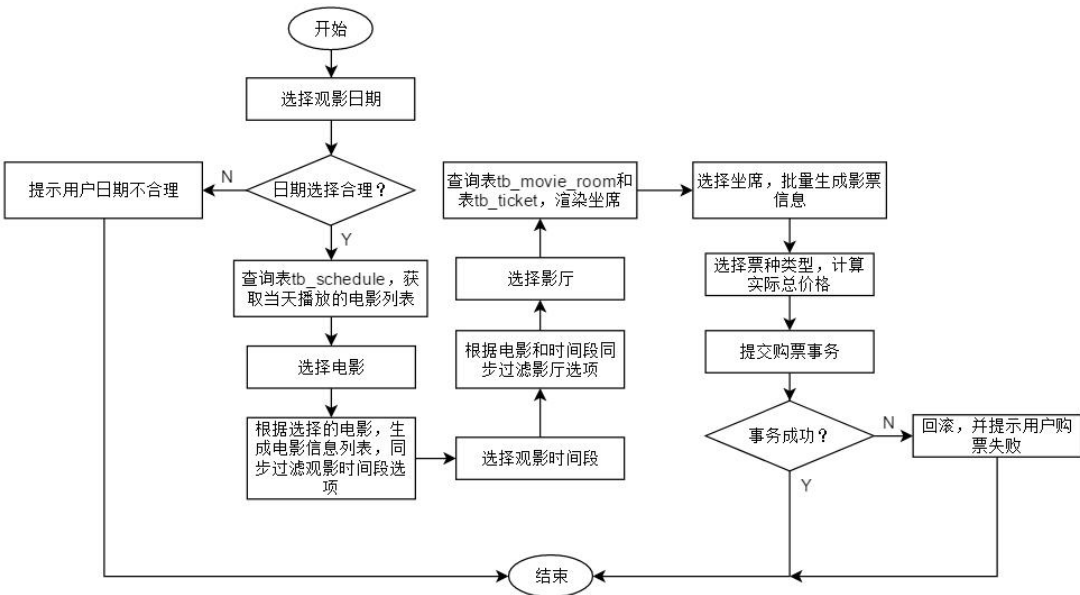


图 4.13 购票功能流程图

考虑实际应用背景下，电影院前台有多个售票窗口，购票行为有并发性，即当前窗口购票的一批票中不能在其他窗口被售出，否则不允许此次购票行为执行。系统需保证购票功能的正确执行，故此处将购票设定为事务执行，当提交购票事务时，系统将此前批量生成的电影票信息逐条按 sql 语句插入表 `tb_ticket`，一旦中间插入失败，则回滚事务并提示用户购票失败；否则，提交事务，提示用户购票成功。

购票事务的 java 语言定义如下：

```

try {
    //关闭自动提交，进行事务处理
    connection.setAutoCommit(false);
    Statement statement = connection.createStatement();
    //插入每一张购票信息，出错则回滚
    for (TicketInfo ticketInfo : ticketInfos) {
        String sql = String.format("insert into tb_ticket
values(null,%d,%d,%d,%d,%d,%d,%3.1f)",
                                ticketInfo.getSchedule_id(),
                                ticketInfo.getSeat_row(),
                                ticketInfo.getSeat_column(),
                                ticketInfo.getTicket_type(),
                                ticketInfo.isUse_flag()?1:0,
                                ticketInfo.getAp());
        statement.addBatch(sql); //将本条 sql 语句加入执行器
    }
    statement.executeBatch(); //执行所有 sql 语句
    connection.commit(); //提交事务
    connection.setAutoCommit(true); //开启自动提交
    return true;
} catch (SQLException e) { //sql 语句执行异常捕获
    e.printStackTrace();
    try {
        connection.rollback(); //回滚事务
        connection.setAutoCommit(true); //开启自动提交
        return false;
    } catch (SQLException ex) {
        ex.printStackTrace();
        return false;
    }
}
}

```

对于退票功能，根据影票 ID 查询表 tb_ticket，若有对应记录，且属性 use_flag 的值为 0（表示电影票未使用），则进行退票；否则不予退票。

售票及退票子系统涉及对表 tb_movie、tb_movie_room、tb_shedule 的查询操作，以及对表 tb_ticket 的查询、添加、删除、更新操作，相应 sql 语句的 java 代码实现见附录。

售票及退票子系统 UI 设计如图 4.14 所示。

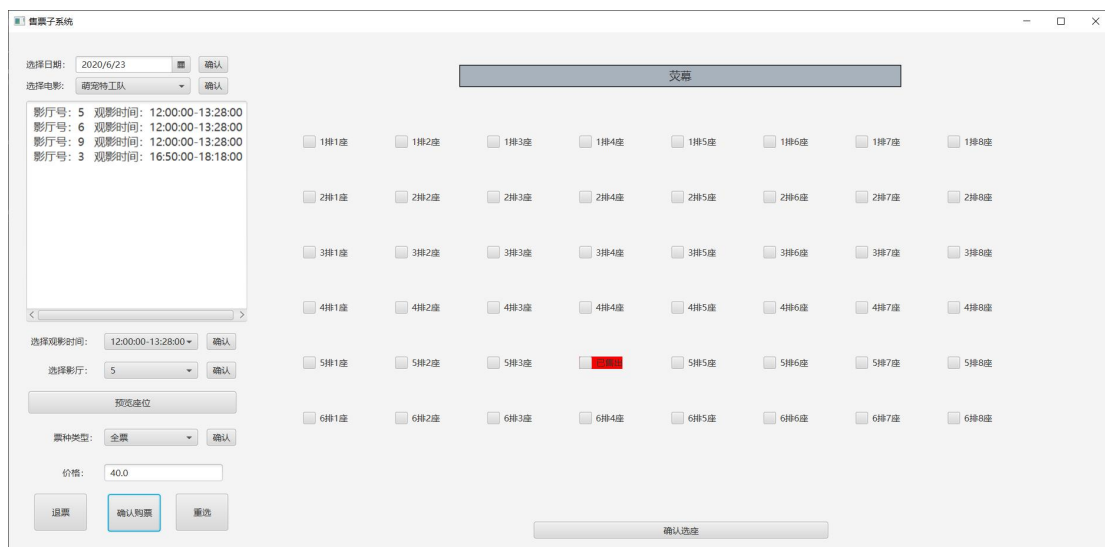


图 4.14 售票及退票子系统 UI 设计

4.6 系统测试

创建表的 sql 语句和插入测试数据的 sql 语句已随本报告打包，分别为建表.sql 文件和测试数据.sql 文件，测试数据中插入了 4 条账号信息，覆盖了 3 个账户类型，两个售票员账户用于测试并发购票操作；电影表中共 4 部电影信息，影厅表中共 9 条影厅信息，这些影厅具有不同规格的坐席数量，用于测试售票子系统中坐席渲染的正确性。

整个测试流程为：首先测试登录子系统，接下来对账户管理子系统进行测试，然后测试电影管理子系统、影厅管理子系统、档期管理子系统，最后测试售票及退票子系统。

需要说明的是，对于所有子系统公共异常情况（例如文本框未输入内容、输入 ID 时输入非数字字符串等），由于系统开发时对此类异常采用了统一的识别处理方式，故这类公共异常只在首次出现时测试。

首先测试登录子系统，直接登录时应该提示账号或密码为空，结构如图 4.15 所示。



图 4.15 直接登录结果

输入账号 root 和错误的密码 roo，登录，应该提示账号或密码错误，结构如图 4.16 所示。

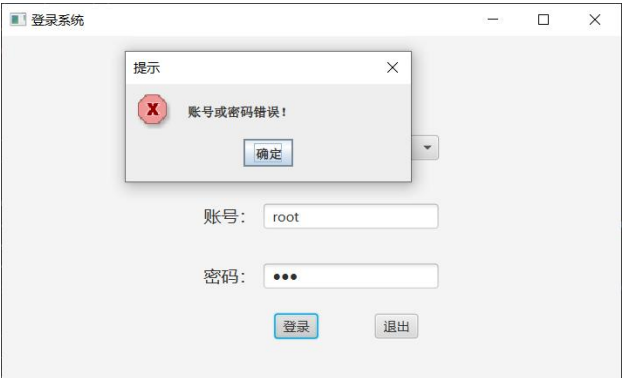


图 4.16 输入错误密码时登录结果

输入超级管理员 root 账号及密码，登录后，跳转至账号管理子系统界面，

结果如图 4.17 所示。



图 4.17 超级管理员登录结果

点击添加，什么都不输入直接添加，应该提示未输入账号，结果如图 4.18 所示。

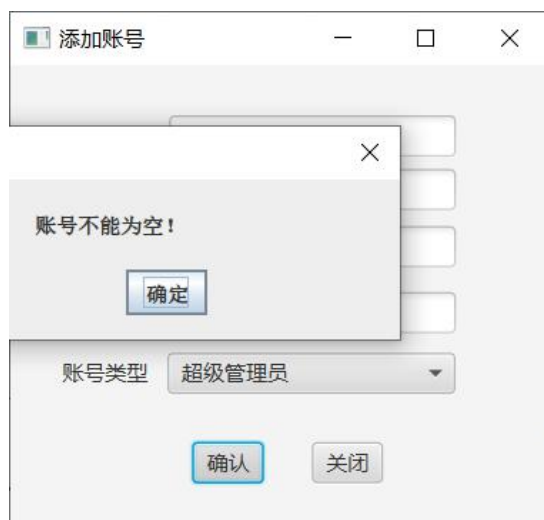


图 4.18 未输入账号添加结果

输入表中定义的非空属性账号、密码、姓名后，选择账号类型为管理员，添加结果如图 4.19 所示。

账号	密码	姓名	电话	账号类型	上
admin	admin	李四	87541111	1	2020-06-22 1
admin2	admin2	张三		1	

图 4.19 添加管理员账户结果

可以看到表中刷新了添加进去的账户信息。然后修改这个账户的姓名为李四，将其账号类型改为售票员，如图 4.20 所示。



修改账号信息对话框，包含以下字段和按钮：

- 账号：admin2
- 密码：admin2
- 姓名：李四
- 电话：（空）
- 账号类型：售票员
- 按钮：确定、修改、修改、修改、确认、关闭

图 4.20 修改账户界面

修改结果见图 4.21。

账号	密码	姓名	电话	账号类型
admin	admin	李四	87541111	1
admin2	admin2	李四		2

图 4.21 修改账户结果

然后测试删除账户功能，从输入异常情况此前已测试过，这里输入一个不存在的账户，点击删除，结果如图 4.22 所示。



删除账号对话框，包含以下元素：

- 提示：请输入要删除的账号：
- 输入框：abc123
- 提示框：账号不存在！

图 4.22 输入非数字串结果

然后，删除此前添加的账号 admin2，删除结果如图 4.23 所示。



账号管理子系统窗口，显示账号信息表：

账号	密码	姓名	电话	账号类型	上次登录时间
admin	admin	李四	87541111	1	2020-06-22 16:06:48
root	root	张三	87540000	0	2020-06-22 20:23:42
seller1	seller1	王五	87542222	2	2020-06-22 16:25:43
seller2	seller2	赵六	87543333	2	2020-06-17 15:27:02

图 4.23 删除账户 admin2 结果

可以看到账户被成功删除。

接着退出账号，登录管理员账号，如图 4.24 所示。



图 4.24 管理员账号登录界面

选择添加电影，此前已测试过空输入异常情况，此处不再测试，输入电影名称“测试1”，时长选择1时0分0秒，上映，添加界面和结果如图 4.25、4.26 所示。

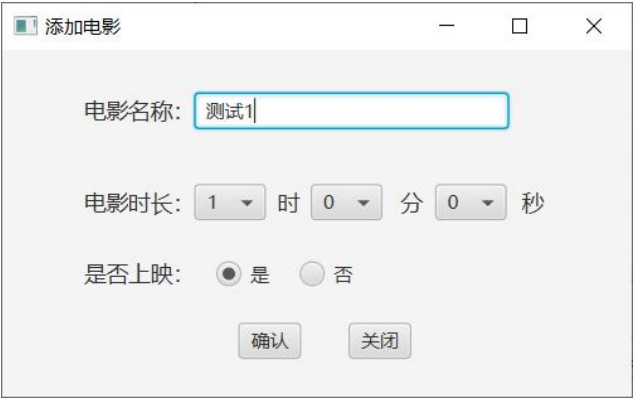


图 4.25 添加电影界面

电影ID	电影名	电影时长	是否上映
1	一呼一吸	02:18:48	true
2	我们的父辈	01:41:23	true
3	拯救大兵瑞恩	02:48:21	true
4	萌宠特工队	01:28:28	true
11	测试1	01:00:00	true

图 4.26 添加结果

然后测试电影的上映/下架功能，此前已测试过空输入处理，此处测试输入非数字串时程序的响应，结果如图 4.27 所示。



图 4.27 输入非数字串 ID 结果

将此前添加的电影“测试 1”下架，结果如图 4.28 所示。

4	明龙特工队	01:28:28	true
11	测试1	01:00:00	false

图 4.28 下架电影测试结果

然后测试删除电影功能，将电影“测试 1”删除，结果如图 4.29 所示。

电影ID	电影名	电影时长	是否上映
1	一呼一吸	02:18:48	true
2	我们的父辈	01:41:23	true
3	拯救大兵瑞恩	02:48:21	true
4	萌宠特工队	01:28:28	true

图 4.29 删除电影测试结果

结果说明删除成功。由于此前已测试过输入 ID 不存在的异常处理情况，此处不再测试。

接下来测试对影厅的添加、启用/停用、删除功能，由于影厅管理子系统的管理模式于电影管理模式完全相同，故此处不再测试异常情况。首先，添加影厅“测试 1”，选择坐席行数和列数均为 10，添加界面和添加结果分别如图 4.30 和 4.31 所示。

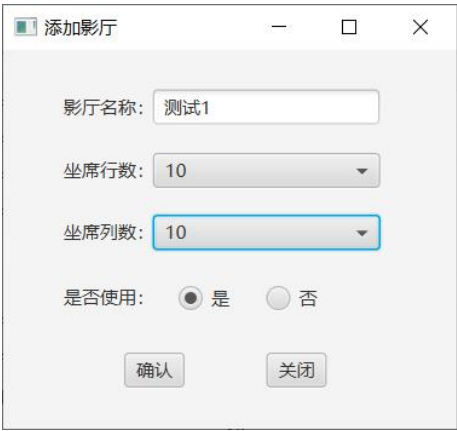


图 4.30 添加影厅界面

影厅号	影厅名称	坐席行数	坐席列数	是否使用
1	标准厅	6	8	true
2	标准厅	6	8	true
3	标准厅	6	8	true
4	标准厅	6	8	true
5	3D厅	6	8	true
6	3D厅	6	8	true
7	巨幕厅	10	12	true
8	巨幕厅	10	12	true
9	IMAX厅	12	14	true
14	测试1	10	10	true

图 4.31 添加影厅测试结果

然后停用影厅“测试1”，结果如图 4.32 所示。

14	测试1	10	10	false
----	-----	----	----	-------

图 4.32 停用影厅测试结果

然后删除影厅“测试1”，结果如图 4.33 所示。

影厅号	影厅名称	坐席行数	坐席列数	是否使用
1	标准厅	6	8	true
2	标准厅	6	8	true
3	标准厅	6	8	true
4	标准厅	6	8	true
5	3D厅	6	8	true
6	3D厅	6	8	true
7	巨幕厅	10	12	true
8	巨幕厅	10	12	true
9	IMAX厅	12	14	true

图 4.33 删除影厅测试结果

结果显示，影厅成功删除。

然后测试档期管理子系统，首先是选择日期，测试日期是 2020-6-22，这里选择 2020-6-22，确定时，结果如图 4.34 所示。

档期ID	影厅号	电影名称	开始时间	结束时间	档期日期	禁止显示
99	1	拯救大兵瑞恩	16:50:00	19:38:00	2020-06-22	30.0
100	1	拯救大兵瑞恩	19:50:00	22:38:00	2020-06-22	30.0
101	2	拯救大兵瑞恩	16:50:00	19:38:00	2020-06-22	30.0
102	2	我们的父辈	22:10:00	23:51:00	2020-06-22	30.0
103	3	萌宠特工队	16:50:00	18:18:00	2020-06-22	30.0
104	3	拯救大兵瑞恩	18:30:00	21:18:00	2020-06-22	30.0
105	4	一呼一吸	00:00:00	02:18:00	2020-06-22	10.0
94	5	萌宠特工队	12:00:00	13:28:00	2020-06-22	40.0
95	6	萌宠特工队	12:00:00	13:28:00	2020-06-22	40.0
96	7	我们的父辈	12:00:00	13:41:00	2020-06-22	35.0
97	8	我们的父辈	12:00:00	13:41:00	2020-06-22	35.0
98	9	萌宠特工队	12:00:00	13:28:00	2020-06-22	50.0

图 4.34 档期管理界面

测试添加档期功能的时间冲突检测算法，对 1 号影厅，在 19:45 插入一个播放电影一呼一吸的档期，结果如图 4.35 所示。

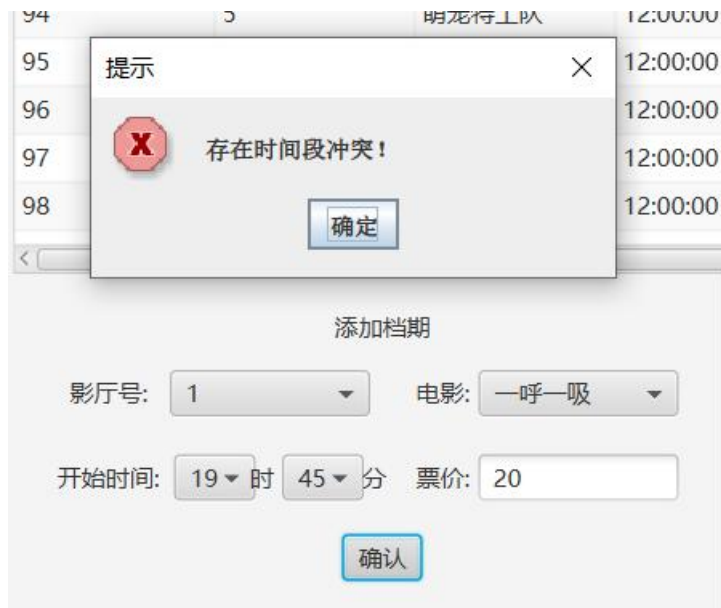


图 4.35 时间冲突检测算法测试

结果说明时间冲突检测算法成功检测到时间冲突，并阻止了该记录的插入。然后进行正常插入，影厅 1 在 10:00 开始播放一呼一吸，测试几个如图 4.36 所示。

118	1	一呼一吸	10:00:00	12:18:00	2020-06-22	30.0
-----	---	------	----------	----------	------------	------

图 4.36 档期插入测试结果

接着将这条档期记录删除，空输入异常和非数字串输入异常此前已测试，此处不再测试，结果如图 4.37 所示。

档期ID	影厅号	电影名称	开始时间	结束时间	播放日期	额定
99	1	拯救大兵瑞恩	16:50:00	19:38:00	2020-06-22	30.0
100	1	拯救大兵瑞恩	19:50:00	22:38:00	2020-06-22	30.0
101	2	拯救大兵瑞恩	16:50:00	19:38:00	2020-06-22	30.0
102	2	我们的父辈	22:10:00	23:51:00	2020-06-22	30.0

图 4.37 删除档期测试结果

影厅 1 的档期里已删除该条记录。

接着测试档期复用功能，选择未安排档期的日期 2020-6-25，如图 4.38 所示。

增加档期

删除档期

查询档期安排

档期ID

影厅号

电影名称

开始时间

结束时间

播放日期

额定

连续日期:

5050\6\52

确定

增加条

删除条

档期保留条

图 4.38 2020-6-25 档期安排

选择 2020-6-22 的档期进行复用，如图 4.39 所示。

复用档期安排

复用日期：

2020/6/22

确认

图 4.39 档期复用界面

复用结果如图 4.40 所示。

选择日期：2020/6/25

确认

档期ID	影厅号	电影名称	开始时间	结束时间	播放日期	额定票价
119	5	萌宠特工队	12:00:00	13:28:00	2020-06-25	40.0
120	6	萌宠特工队	12:00:00	13:28:00	2020-06-25	40.0
121	7	我们的父辈	12:00:00	13:41:00	2020-06-25	35.0
122	8	我们的父辈	12:00:00	13:41:00	2020-06-25	35.0
123	9	萌宠特工队	12:00:00	13:28:00	2020-06-25	50.0
124	1	拯救大兵瑞恩	16:50:00	19:38:00	2020-06-25	30.0
125	1	拯救大兵瑞恩	19:50:00	22:38:00	2020-06-25	30.0
126	2	拯救大兵瑞恩	16:50:00	19:38:00	2020-06-25	30.0
127	2	我们的父辈	22:10:00	23:51:00	2020-06-25	30.0
128	3	萌宠特工队	16:50:00	18:18:00	2020-06-25	30.0
129	3	拯救大兵瑞恩	18:30:00	21:18:00	2020-06-25	30.0
130	4	一呼一吸	00:00:00	02:18:00	2020-06-25	10.0

图 4.40 档期复用测试结果

可以发现完全复用了 6 月 22 日的档期安排，播放日期修改为 6 月 25 日。
最后测试售票子系统，切换至售票员 1 的账号登录，如图 4.41 所示。

售票子系统

选择日期：

确认

选择电影：

确认

选择观影时间：

确认

选择影厅：

确认

预览座位

票种类型：

全票

确认

价格：

退票 确认购票 重试

确认选座

图 4.41 售票员登录界面

售票界面的操作逻辑是：选择观影日期，点击确认，然后选择电影，点击确认，然后选择观影时间，点击确认，然后选择影厅，点击确认，然后预览座位，通过右侧的座位选择，点击确认选座，然后选座票种类型，点击确认，产生价格，最后选择确认购票。

违反该操作流程均为异常情况，经过系列测试，系统能够应对异常操作次序并做出提示，此处不再展示，仅展示购票时的最终结果如图 4.42、4.43 所示。

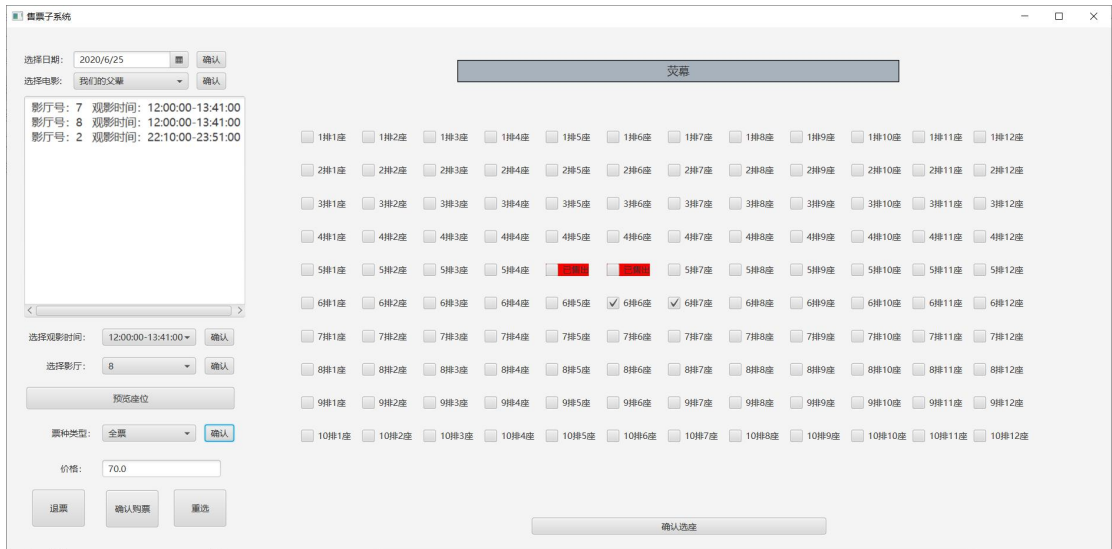


图 4.42 售票功能测试结果(1)



图 4.43 售票功能测试结果(2)

另外登录售票员 2 的账号，打开另一个售票窗口，测试并发购票时系统的响应，在其中第一个售票窗口中，选座如图 4.44 所示的两个坐席，确认后生成电影票信息，在点击购票开始购票事务前，在另一个窗口购买掉图 4.44 选择的两个坐席中的一个坐席，如图 4.45 所示，回到第一个窗口，点击购票，结果如图 4.46 所示。



图 4.44 售票窗口 1 选择坐席

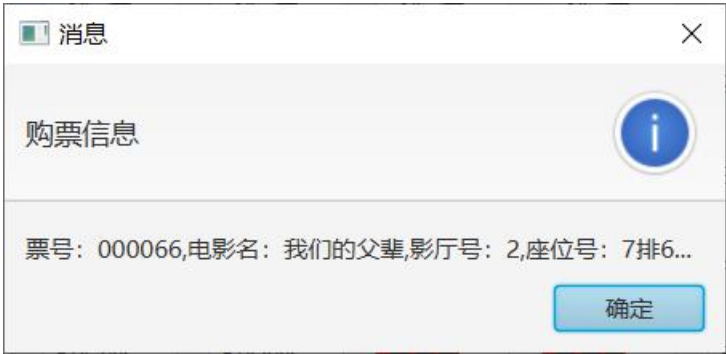


图 4.45 售票窗口 2 中售卖掉 7 排 6 座



图 4.46 售票窗口 1 购票结果

系统提示购票事务失败，并刷新坐席列表，显示该坐席已售出。

接着测试退票功能，由于售票员界面没有显示电影票列表，故查询数据库得到如图 4.47 所示的售票列表。

	tid	schedule_id	seat_row	seat_column	ticket_type	use_flag	ap
1	1	24	1	1	0	1	40.0
2	2	24	1	2	0	1	40.0
3	3	24	1	3	0	1	40.0
4	4	24	1	4	0	1	40.0
5	6	24	1	5	0	1	40.0
6	9	24	3	5	0	1	40.0
7	16	24	2	1	0	1	40.0
8	17	24	2	2	0	1	40.0
9	18	24	2	3	0	1	40.0
10	19	24	2	4	0	1	40.0
11	20	24	2	5	0	1	40.0
12	34	24	3	4	0	1	40.0
13	35	24	3	3	0	1	40.0
14	36	24	4	3	0	0	40.0
15	37	24	4	4	0	0	40.0
16	38	24	4	5	0	0	40.0
17	39	24	4	2	0	0	40.0
18	40	24	5	2	0	0	40.0
19	42	24	5	4	1	0	20.0
20	43	24	5	5	1	0	20.0

图 4.47 售票列表

在退票界面，空输入和数字串输入异常此前已测试，此处不再赘述。退掉 ID 为 1 的票，结果如图 4.48 所示。

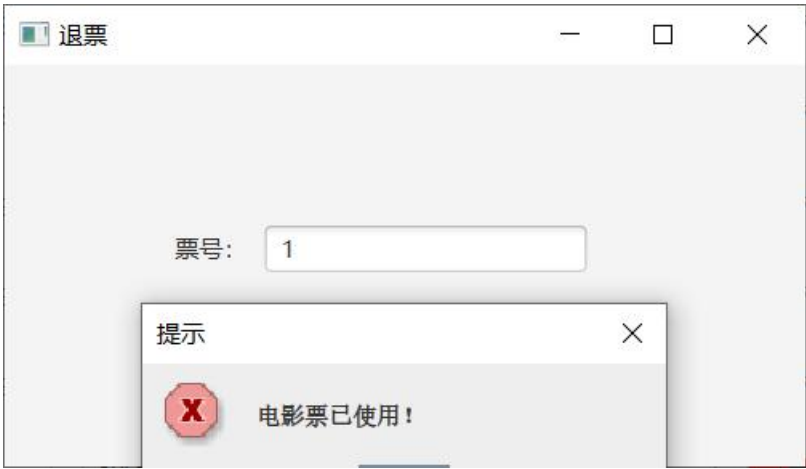


图 4.48 退票号为 1 的票测试结果

由于该电影票 use_flag 为 1，表示已经使用，故不允许退票。接着推掉 ID 为 36 的票，结果如图 4.49 所示。



图 4.49 退票号为 364 的票测试结果

查询数据库，发现该电影票记录已被删除。

至此，测试流程结束，程序功能能正常实现，对异常操作有响应，程序能正常对数据库中的表进行查询、插入、删除、更新等操作。

4.7 系统设计与实现总结

在系统设计与实现中，主要完成了以下工作：

1. 针对电影院信息管理的需求，详细分析了系统的功能需求、性能需求、数据完整性需求，并制定了数据流图。
2. 对系统的整体架构进行了设计并通过系统架构图体现，通过系统总体业务流程图分析了系统的业务流程。

3. 在数据库设计中，通过 E-R 图构建了系统中的实体关系模型，并将其转换为关系数据模型，最终构造了系统所需的五个表。
4. 使用 javaFX 作为 UI 开发，基于 MVC 开发模式，实现了一套良好的用户操作界面，程序能够应对各类异常情况（空输入、输入不合理、操作不合理）。
5. 对系统中各类与数据库交互的代码，集成在了 DAO(Data Access Object) 中，使业务逻辑代码和数据交互代码分离。
6. 对电影的下架/删除和影厅的停用/删除操作，定义了触发器，在数据库层面保证了数据库中档期信息的正确性（详细内容见 4.5.4）。
7. 对档期信息的插入，设计了时间段冲突检测算法进行冲突检测，在应用程序层面保证了数据库中档期信息的正确性（详细内容见 4.5.5）。
8. 在档期管理功能中设计了档期信息的复用功能，能够批量拷贝指定日期的档期安排，方便了管理员进行排挡（详细内容见 4.5.5）。
9. 在售票子系统中，为方便售票员直观选择售票，基于监听器模式设计了一系列可自过滤信息的操作流程，并可在界面动态渲染坐席界面，可点击多个坐席进行批量购票，方便了购票操作（详细内容见 4.5.6）。
10. 在售票子系统中，定义了购票事务，保证数据库的正确性。
11. 考虑到电影票喜喜表记录数量较大，在表 tb_ticket 的属性组 (schedule_id,seat_row,seat_column) 上建立了索引，保证了数据的唯一性，提高了查询记录的性能。

5 课程总结

在课程实践环节中，首先熟悉了达梦 DBMS，通过其管理软件实现了数据库的创建、销毁、备份及恢复，实现了权限的授予，这一部分的问题可以通过网上查询和查询使用手册解决，较为简单。

接下来是练习各类 SQL 语句，这一部分是我认为本次实践中最难的部分，提供的数据结合了当时的疫情背景，需实现的功能较为复杂，需要多表连接查询、派生表、嵌套查询等操作，每个功能的 sql 语句都写得比平时课程学习的复杂得多，还要自己构造数据验证 sql 语句的正确性，创建触发器的语句也很复杂。当时在做这部分实践时课程还未学习到数据库事务和存储过程，故这一部分的内容未实践，算是 sql 练习部分的有待完善的工作。总之通过 sql 语句练习的实践，对书写 sql 语句起到了非常大的帮助，使我在接下来的综合实践中能够熟练地写出各种 sql 语句。

综合实践是我认为本次课程实践中最有意义和最感兴趣的部分，课程提供了几个可选项做参考，同时也没限制选题，完全可以自由发挥，在综合实践这一部分，从提出系统的功能需求，到做需求分析，再到数据库设计，然后到详细设计和系统实现，是大学课程的所有实践课程中系统开发流程最为完整的。我选择了自选题目电影院信息管理系统，实现了账号管理、电影管理、影厅管理、档期管理、售票和退票等功能，在这些功能中，我认为做得好的地方在档期管理中的复用档期功能和售票功能中的坐席渲染功能，档期复用功能使得管理员能够轻松地进行重复档期安排工作，坐席渲染功能使得售票员可以方便地批量选择坐席进行售票。在整个系统开发中，我考虑了许多异常情况并对此进行处理，比如在添加档期时设定了时间冲突检查算法，防止一个影厅在一天内出现冲突的时间安排，在下架电影和停用影厅时，定义了触发器同步删除档期表中的相应记录，进行购票时，定义了购票事务保证了数据的正确性，还有很多人机交互逻辑上的处理，使得用户能够根据软件界面引导使用系统。在完成系统的代码实现时，一共写了 4262 行 java 代码，是目前实现的系统中代码量最大的。

当然，由于是自选题目，且对实际应用背景不是特别了解，在开发完这个系统后回顾了一下，发现还是有许多可以改进的地方，比如为实现用户自助购票的会员账号子系统，统计资金流水、电影票房和管理员工工资的财政子系统，这些都是电影院信息管理系统具有实际需求的功能。

总之，在今年这个疫情特殊年，在家里完成了本课程的实践环节，由于缺少了线下和老师面对面交流的机会，对这次综合实践还是有些许影响，在此感谢老师细心的远程指导，为我纠正了在数据库设计上的一些错误，让我能顺利地完成任务。

附录

```
/**
 * 获取账号信息，根据账号查询账户信息表，返回查询结果集
 * 由于账号是主码，故结果集最多只有一个元组
 * @param account 账户字符串
 * @return 账户信息的结果集
 */
public ResultSet getAccountInfo(String account) {
    try {
        Statement statement = connection.createStatement();
        String sql = String.format("select * from tb_account where account =
        \"%s\"", account);
        return statement.executeQuery(sql);
    } catch (SQLException e) {
        return null; // 返回 null，到上层 Controller 处理该异常
    }
}

/**
 * 更新账户的最后登录时间，抛出异常由上层处理
 * @param account 账号
 * @param time 时间字符串，格式 yyyy-mm-dd hh:mm:ss
 */
public void updateLoginTime(String account, String time) {
    try {
        Statement statement = connection.createStatement();
        String sql = String.format("update tb_account set last_login_datetime
        = \"%s\" where account = \"%s\"", time, account);
        statement.executeUpdate(sql);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * 获取账号信息列表，抛出异常由上层处理
 * @return 账号对象的列表
 */
public ObservableList<AccountInfo> getAccountInfoList() {
    ObservableList<AccountInfo> list = FXCollections.observableArrayList();
    try {
        Statement statement = connection.createStatement();
```

```

        String sql = "select * from tb_account";
        ResultSet re = statement.executeQuery(sql);
        while (re.next()) {
            String account = re.getString("account");
            String pwd = re.getString("pwd");
            String uname = re.getString("uname");
            String phone = re.getString("phone");
            Integer idtype = re.getInt("idtype");
            Timestamp last_login_datetime =
re.getTimestamp("last_login_datetime");
            String datetime = null;
            if (last_login_datetime != null) {
                datetime = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss").format(last_login_datetime);
            }
            list.add(new AccountInfo(account, pwd, uname, phone, idtype,
datetime));
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;//返回 null， 到上层 Controller 处理该异常
    }
    return list;
}

public boolean insertAccountInfo(AccountInfo accountInfo) {
    if (accountInfo == null) {
        return false;
    }
    try {
        Statement statement = connection.createStatement();
        String sql = String.format("insert into
tb_account(account,pwd,uname,phone,idtype,last_login_datetime)
VALUES('%s','%s','%s','%s',%d,null)",
accountInfo.getAccount(),accountInfo.getPwd(),accountInfo.getName(),accountInfo.g
etPhone(),accountInfo.getUserType());
        statement.executeUpdate(sql);
        return true;
    } catch (SQLException e) {
        return false;//返回 false， 到上层 Controller 处理该异常
    }
}

/**

```

```

    * 删除账号信息表中指定账号信息，其他的修改由触发器自动完成
    * @param account 要删除的账号
    * @return 成功删除则返回 true，否则返回 false，由上层处理
    */
    public boolean deleteAccountInfo(String account) {
        try {
            Statement statement = connection.createStatement();
            String sql = String.format("delete from tb_account where account = '%s'",account);
            statement.executeUpdate(sql);
            return true;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;//返回 false，到上层 Controller 处理该异常
        }
    }

    /**
     * 修改账号信息表中指定账户的信息，其他的修改由触发器自动完成
     * @param accountInfo 要修改的账号数据对象
     * @return 修改成功返回 true，否则返回 false，由上层处理
     */
    public boolean updateAccountInfo(AccountInfo accountInfo) {
        if (accountInfo == null) {
            return false;
        }
        try {
            Statement statement = connection.createStatement();
            String sql = String.format("update tb_account set pwd = '%s',uname = '%s',phone='%s',idtype = '%s'where account = '%s'",accountInfo.getPwd(),accountInfo.getName(),accountInfo.getPhone(),accountInfo.getUserType(),accountInfo.getAccount());
            statement.executeUpdate(sql);
            return true;
        } catch (SQLException e) {
            return false;//返回 false，由上层 Controller 处理异常
        }
    }

    public ResultSet getMovieInfo(Integer movieid) {
        try {
            Statement statement = connection.createStatement();
            String sql = String.format("select * from tb_movie where movieid = %d",movieid);

```



```

        return statement.executeQuery(sql);
    } catch (SQLException e) {
        return null;//返回 null， 到上层 Controller 处理该异常
    }
}

/**
 * 获取电影信息列表， 异常由上层处理
 * @return 电影信息列表
 */
public ObservableList<MovieInfo> getMovieInfoList() {
    ObservableList<MovieInfo> list = FXCollections.observableArrayList();
    try {
        Statement statement = connection.createStatement();
        String sql = "select * from tb_movie";
        ResultSet re = statement.executeQuery(sql);
        while (re.next()) {
            Integer mid = re.getInt("movieid");
            String mname = re.getString("mname");
            String mtime = re.getTime("mtime").toString();
            boolean screen = re.getBoolean("screen");
            list.add(new MovieInfo(mid, mname, mtime, screen));
        }
    } catch (SQLException e) {
        return null;//返回 null， 到上层 Controller 处理该异常
    }
    return list;
}

/**
 * 获取电影的数量， 用来产生下一个插入的电影 ID
 * @return 电影数量
 */
public Integer getSumOfMovie() {
    try {
        Statement statement = connection.createStatement();
        String sql = "select count(*) as sum_of_movie from tb_movie";
        ResultSet re = statement.executeQuery(sql);
        if (re.next()) {
            return re.getInt("sum_of_movie");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

```

        return -1;//返回 false， 到上层 Controller 处理该异常
    }

    /**
     * 向电影表中添加电影信息，成功返回 true，失败返回 false
     * @param movieInfo 电影信息数据对象
     * @return 成功返回 true，失败返回 false
     */
    public boolean insertMovieInfo(MovieInfo movieInfo) {
        if (movieInfo == null) {
            return false;
        }
        try {
            Statement statement = connection.createStatement();
            String sql = String.format("insert into tb_movie
VALUES(null,'%s','%s\\',%d)",
movieInfo.getMname(),movieInfo.getMtime(),movieInfo.isScreen()?1:0);
            statement.executeUpdate(sql);
            return true;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;//返回 false， 到上层 Controller 处理该异常
        }
    }

    /**
     * 删除指定 ID 的电影，删除成功返回 true，失败返回 false
     * @param movieid 电影 ID
     * @return 删除成功返回 true，失败返回 false
     */
    public boolean deleteMovieInfo(Integer movieid) {
        try{
            Statement statement = connection.createStatement();
            String sql = String.format("delete from tb_movie where movieid
= %d",movieid);
            statement.executeUpdate(sql);
            return true;
        } catch (SQLException e) {
            return false;//返回 false， 到上层 Controller 处理该异常
        }
    }

    /**
     * 对指定电影 ID 进行上映/下架

```

```

    * @param movieid 电影 ID
    * @param screen true 表示上映, false 表示下架
    * @return 成功返回 true, 失败返回 false
    */
    public boolean screenMovie(Integer movieid, boolean screen) {
        try{
            Statement statement = connection.createStatement();
            String sql = String.format("update tb_movie set screen = %d where
movieid = %d",screen?1:0, movieid);
            statement.executeUpdate(sql);
            return true;
        } catch (SQLException e) {
            return false;//返回 false, 到上层 Controller 处理该异常
        }
    }

    /**
    * 获取指定影厅的信息
    * @param roomid 影厅 ID
    * @return 指定影厅的查询结果
    */
    public ResultSet getMovieRoomInfo(Integer roomid) {
        try{
            Statement statement = connection.createStatement();
            String sql = String.format("select * from tb_movie_room where
roomid = %d", roomid);
            return statement.executeQuery(sql);
        } catch (SQLException e){
            e.printStackTrace();
            return null;//返回 null, 到上层 Controller 处理该异常
        }
    }

    /**
    * 获取影厅信息列表
    * @return 影厅信息列表
    */
    public ObservableList<MovieRoomInfo> getMovieRoomInfoList() {
        ObservableList<MovieRoomInfo> list =
FXCollections.observableArrayList();
        try {
            Statement statement = connection.createStatement();
            String sql = "select * from tb_movie_room";
            ResultSet re = statement.executeQuery(sql);

```

```

        while (re.next()) {
            Integer roomid = re.getInt("roomid");
            String rname = re.getString("rname");
            Integer row_num = re.getInt("row_num");
            Integer column_num = re.getInt("column_num");
            boolean useflag = re.getBoolean("useflag");
            list.add(new MovieRoomInfo(roomid, rname, row_num,
column_num, useflag));
        }
    } catch (SQLException e) {
        return null;//返回 null，到上层 Controller 处理该异常
    }
    return list;
}

```

```

/**
 * 获取电影的数量，用来产生下一个插入的电影 ID
 * @return 电影数量
 */
public Integer getSumOfMovieRoom() {
    try {
        Statement statement = connection.createStatement();
        String sql = "select count(*) as sum_of_movie_room from
tb_movie_room";
        ResultSet re = statement.executeQuery(sql);
        if (re.next()) {
            return re.getInt("sum_of_movie_room");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1;//返回 false，到上层 Controller 处理该异常
}

```

```

/**
 * 添加影厅信息，添加成功返回 true，失败返回 false
 * @param movieRoomInfo 影厅信息对象
 * @return 成功返回 true，失败返回 false
 */
public boolean insertMovieRoomInfo(MovieRoomInfo movieRoomInfo) {
    if (movieRoomInfo == null) {
        return false;
    }
    try {

```

```

        Statement statement = connection.createStatement();
        String sql = String.format("insert into tb_movie_room
VALUES(null,'%s',%d,%d,%d)",
movieRoomInfo.getRname(),movieRoomInfo.getRow_num(),movieRoomInfo.getCol
umn_num(),movieRoomInfo.isUseflag()?1:0);
        statement.executeUpdate(sql);
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;//返回 false， 到上层 Controller 处理该异常
    }
}

```

```

/**
 * 根据 useflag 更新影厅的使用情况
 * @param roomid 影厅 id
 * @param useflag 使用情况，true 表示使用，false 表示停用
 * @return 修改成功返回 true，失败返回 false
 */
public boolean useMovieRoom(Integer roomid, boolean useflag) {
    if (roomid == null) {
        return false;
    }
    try {
        Statement statement = connection.createStatement();
        String sql = String.format("update tb_movie_room set useflag = %d
where roomid = %d",useflag?1:0, roomid);
        statement.executeUpdate(sql);
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;//返回 false， 由上层 Contoller 处理异常
    }
}

```

```

/**
 * 删除指定的影厅信息
 * @param roomid 影厅 id
 * @return 删除成功返回 true，失败返回 false
 */
public boolean deleteMovieRoomInfo(int roomid) {
    try{
        Statement statement = connection.createStatement();
        String sql = String.format("delete from tb_movie_room where roomid

```

```

        = "%d",roomid);
        statement.executeUpdate(sql);
        return true;
    } catch (SQLException e) {
        return false;//返回 false， 到上层 Controller 处理该异常
    }
}

/**
 * 根据指定日期，获取当日档期表的 list，注意档期表中存放的是电影 ID
 * 需转查电影表换为电影名称
 * @param date 指定日期字符串
 * @return 指定日期的档期表 list，异常返回 null
 */
public ObservableList<ScheduleInfo> getScheduleInfoListByDate(String date) {
    ObservableList<ScheduleInfo> list = FXCollections.observableArrayList();
    try {
        Statement statement = connection.createStatement();
        String sql = String.format("select * from tb_schedule where show_date
= '%s'", date);
        ResultSet re = statement.executeQuery(sql);
        while (re.next()) {
            Integer schedule_id = re.getInt("schedule_id");
            ResultSet resultSetOfMovie =
getMovieInfo(re.getInt("movie_id"));
            String movie_name = null;
            if (resultSetOfMovie.next()) {
                movie_name = resultSetOfMovie.getString("mname");
            }
            Integer movie_id = re.getInt("movie_id");
            Integer room_id = re.getInt("room_id");
            String show_time = re.getTime("show_time").toString();
            String end_time = re.getTime("end_time").toString();
            String show_date = re.getDate("show_date").toString();
            Double normal_price = re.getDouble("normal_price");
            list.add(new ScheduleInfo(schedule_id, movie_id, movie_name,
room_id, show_time, end_time, show_date, normal_price));
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;//返回 null， 到上层 Controller 处理该异常
    }
    return list;
}

```

```

/**
 * 获取影厅 ID 列表，排挡管理界面中初始化添加档期的影厅选项使用
 * 注意这里是选取的使用中的影厅
 * @return 影厅 ID 列表
 */
public ArrayList<Integer> getMovieRoomIDList() {
    ArrayList<Integer> list = new ArrayList<>();
    try {
        Statement statement = connection.createStatement();
        String sql = "select roomid from tb_movie_room where useflag = 1";
        ResultSet re = statement.executeQuery(sql);
        while (re.next()) {
            list.add(re.getInt(1));
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;//返回 null，到上层 controller 处理该异常
    }
    return list;
}

/**
 * 获取电影名称列表，排挡管理界面中初始化添加档期的电影选项使用
 * 注意这里是选取的上映中的电影
 * @return 电影名称列表
 */
public ArrayList<String> getMovieNameList() {
    ArrayList<String> list = new ArrayList<>();
    try {
        Statement statement = connection.createStatement();
        String sql = "select mname from tb_movie where screen = 1";
        ResultSet re = statement.executeQuery(sql);
        while (re.next()) {
            list.add(re.getString(1));
        }
    } catch (SQLException e) {
        e.printStackTrace();
        return null;//返回 null，到上层 controller 处理该异常
    }
    return list;
}

/**

```

```

    * 获取指定电影名称的电影时长
    * @param movieName 电影名称
    * @return 该电影时长的 Time 对象
    */
    public Time getMovieTime(String movieName) {
        try {
            Statement statement = connection.createStatement();
            String sql = "select mtime from tb_movie where mname = " + "'" + movieName + "'";
            ResultSet re = statement.executeQuery(sql);
            if (re.next()) {
                return re.getTime(1);
            }
        } catch (SQLException e) {
            e.printStackTrace();
            return null; // 返回 null，到上层 controller 处理该异常
        }
        return null;
    }

    /**
     * 获取指定日期下指定影厅的时间序列二元组，用于时间冲突检测
     * @param room_id 影厅 ID
     * @param show_date 选定的播放日期
     * @return 时间序列二元组
     */
    public ArrayList<Tuple<Integer,Integer>> getTimeTupleFromMovieRoom(int room_id, String show_date) {
        ArrayList<Tuple<Integer,Integer>> timeList = new ArrayList<>();
        try {
            Statement statement = connection.createStatement();
            String sql = String.format("select show_time, end_time from tb_schedule where room_id = %d and show_date = '%s'", room_id, show_date);
            ResultSet re = statement.executeQuery(sql);
            while (re.next()) {
                Integer beginTime = Integer.parseInt(re.getTime(1).toString().substring(0,2)) * 60 + Integer.parseInt(re.getTime(1).toString().substring(3,5));
                Integer endTime = Integer.parseInt(re.getTime(2).toString().substring(0,2)) * 60 + Integer.parseInt(re.getTime(2).toString().substring(3,5));
                timeList.add(new Tuple<>(beginTime, endTime));
            }
        } catch (SQLException e) {

```



```

        e.printStackTrace();
        return null;//返回 null， 到上层 controller 处理该异常
    }
    return timeList;
}

/**
 * 插入档期信息，注意插入的档期一定是不存在时间冲突的合法信息
 * 在 Controller 中已经通过冲突检测，此处直接插入表即可
 * @param scheduleInfo 档期信息对象
 * @return 插入成功返回 true，失败返回 false
 */
public boolean insertScheduleInfo(ScheduleInfo scheduleInfo) {
    if (scheduleInfo == null) {
        return false;
    }
    try {
        Statement statement = connection.createStatement();
        String sql = String.format("insert into tb_schedule
VALUES(null,%d,%d,'%s','%s','%s',%3.1f)",scheduleInfo.getMovie_id(),scheduleInf
o.getRoom_id(),scheduleInfo.getShow_time(),scheduleInfo.getEnd_time(),scheduleIn
fo.getShow_date(),scheduleInfo.getNormal_price());
        statement.executeUpdate(sql);
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;//返回 false， 到上层 Controller 处理该异常
    }
}

/**
 * 根据电影名称获取电影 ID
 * @param mname 电影名称
 * @return 若存在则返回电影 ID，若不存在则返回-1
 */
public Integer getMovieIDByName(String mname) {
    try {
        Statement statement = connection.createStatement();
        String sql = "select movieid from tb_movie where mname = '" +
mname + "'";
        ResultSet re = statement.executeQuery(sql);
        if (re.next()) {
            return re.getInt(1);
        }
    }
}

```

```

        } catch (SQLException e) {
            e.printStackTrace();
            return -1;
        }
        return -1;
    }

/**
 * 根据档期 ID 获取档期信息
 * @param scheduleID 档期 ID
 * @return 成功返回结果集，失败返回 null
 */
public ResultSet getScheduleInfo(int scheduleID) {
    try {
        Statement statement = connection.createStatement();
        String sql = "select * from tb_schedule where schedule_id = " +
scheduleID;
        return statement.executeQuery(sql);
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * 根据档期 ID 删除档期信息，注意此处档期 ID 一定存在，
 * Controoler 层删除前已经判断过
 * @param scheduleID 档期 ID
 * @return 删除成功返回 true，失败返回 false
 */
public boolean deleteScheduleInfo(int scheduleID) {
    try {
        Statement statement = connection.createStatement();
        String sql = "delete from tb_schedule where schedule_id = " +
scheduleID;
        statement.executeUpdate(sql);
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
        return false;
    }
}

/**

```

```

    * 通过 ScheduleList 对 Schedule 表进行更新，
    * 用于复用档期功能
    * @param list scheduleList, 里面是相同日期的所有档期
    * @return 更新成功返回 true, 失败返回 false
    */
    public boolean
    updateScheduleInfoByScheduleInfoList(ObservableList<ScheduleInfo> list) {
        try {
            Statement statement = connection.createStatement();
            String sql = "select schedule_id from tb_schedule where show_date = "
            + list.get(0).getShow_date() +"";
            ResultSet re = statement.executeQuery(sql);
            //先删除当前日期的档期安排
            while (re.next()) {
                if (!deleteScheduleInfo(re.getInt(1))) { //若删除失败则返回 false
                    return false;
                }
            }
            for (ScheduleInfo scheduleInfo : list) { //插入每一条档期记录
                sql = String.format("insert into tb_schedule
VALUES(null,%d,%d,'%s','%s','%s','%3.1f",
                                scheduleInfo.getMovie_id(),
                                scheduleInfo.getRoom_id(),
                                scheduleInfo.getShow_time(),
                                scheduleInfo.getEnd_time(),
                                scheduleInfo.getShow_date(),
                                scheduleInfo.getNormal_price());
                statement.executeUpdate(sql);
            }
            return true;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    public ArrayList<String> getMovieNameByWatchDate(String date) {
        try {
            ArrayList<String> list = new ArrayList<>();
            Statement statement = connection.createStatement();
            String sql = "select distinct mname from tb_movie,tb_schedule where
tb_movie.movieid = tb_schedule.movie_id and tb_schedule.show_date = " + date
            +"";

            ResultSet re = statement.executeQuery(sql);

```

```

        while (re.next()) {
            list.add(re.getString(1));
        }
        return list;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

public ObservableList<MovieScheduleInfo> getMovieScheduleInfo(int
movie_id,String show_date) {
    ObservableList<MovieScheduleInfo> list =
FXCollections.observableArrayList();
    try {
        Statement statement = connection.createStatement();
        String sql = "select room_id,show_time,end_time from tb_schedule
where movie_id = " + movie_id + " and show_date = " + show_date +"";
        ResultSet re = statement.executeQuery(sql);
        while (re.next()) {
            list.add(new MovieScheduleInfo(re.getInt(1),new
Tuple<>(re.getString(2),re.getString(3))));
        }
        return list;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

public ArrayList<Integer> getMovieRoomIDListBySelectInfo(int
movie_id,String show_time,String end_time,String show_date) {
    ArrayList<Integer> list = new ArrayList<>();
    try {
        Statement statement = connection.createStatement();
        String sql = "select distinct room_id from tb_schedule where movie_id
= " + movie_id + " and show_time = " + show_time + " and end_time = " +
end_time + " and show_date = " + show_date +"";
        ResultSet re = statement.executeQuery(sql);
        while (re.next()) {
            list.add(re.getInt(1));
        }
        return list;
    } catch (SQLException e){

```

```

        e.printStackTrace();
        return null;
    }
}

public Tuple<Integer,Integer> getMovieRoomSeatTuple(int room_id) {
    Tuple<Integer,Integer> seatTuple = new Tuple<>(0,0);
    try {
        Statement statement = connection.createStatement();
        String sql = "select row_num,column_num from tb_movie_room
where roomid = " + room_id;
        ResultSet re = statement.executeQuery(sql);
        if (re.next()) {
            seatTuple.setP1(re.getInt(1));
            seatTuple.setP2(re.getInt(2));
        }
        return seatTuple;
    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

public int getScheduleIDBySelectInfo(int movie_id,int room_id,String
show_time,String end_time,String show_date) {
    try {
        Statement statement = connection.createStatement();
        String sql = "select schedule_id from tb_schedule where movie_id = "
+ movie_id +
        " and room_id = " + room_id +
        " and show_time = '" + show_time + "'" +
        " and end_time = '" + end_time + "'" +
        " and show_date = '" + show_date + "'";
        ResultSet re = statement.executeQuery(sql);
        if (re.next()) {
            return re.getInt(1);
        }
        return -1;
    } catch (SQLException e) {
        e.printStackTrace();
        return -1;
    }
}

```

```

    public      ObservableList<Tuple<Integer,Integer>>      getSoldSeatInfoList(int
schedule_id) {
        ObservableList<Tuple<Integer,Integer>>      list      =
FXCollections.observableArrayList();
        try {
            Statement statement = connection.createStatement();
            String sql ="select  seat_row,seat_column  from  tb_ticket  where
schedule_id = " + schedule_id;
            ResultSet re = statement.executeQuery(sql);
            while (re.next()) {
                list.add(new Tuple<>(re.getInt(1),re.getInt(2)));
            }
            return list;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }
}

```

```

    public double getNormalPrice(int schedule_id) {
        try {
            Statement statement = connection.createStatement();
            String sql ="select normal_price from tb_schedule where schedule_id
= " + schedule_id;
            ResultSet re = statement.executeQuery(sql);
            if (re.next()) {
                return  re.getDouble(1);
            }
            return -1;
        } catch (SQLException e) {
            e.printStackTrace();
            return -1;
        }
    }
}

```

```

    public boolean insertTicketInfos(ObservableList<TicketInfo> ticketInfos) {
        if (ticketInfos == null) {
            return false;
        }
        try {
            //关闭自动提交，进行事务处理
            connection.setAutoCommit(false);
            Statement statement = connection.createStatement();
            //插入每一张购票信息，出错则回滚

```

```

        for (TicketInfo ticketInfo : ticketInfos) {
            String sql = String.format("insert into tb_ticket
values(null,%d,%d,%d,%d,%d,%3.1f)",
                                ticketInfo.getSchedule_id(),
                                ticketInfo.getSeat_row(),
                                ticketInfo.getSeat_column(),
                                ticketInfo.getTicket_type(),
                                ticketInfo.isUse_flag()?1:0,
                                ticketInfo.getAp());

            statement.addBatch(sql);
        }
        statement.executeBatch();
        connection.commit();
        connection.setAutoCommit(true);
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
        try {
            connection.rollback();
            connection.setAutoCommit(true);
            return false;
        } catch (SQLException ex) {
            ex.printStackTrace();
            return false;
        }
    }
}

```

```

public int getTicketID(int schedule_id,int row_num,int column_num) {
    try {
        Statement statement = connection.createStatement();
        String sql = "select tid from tb_ticket where schedule_id = " +
schedule_id +
                                " and seat_row = " + row_num +
                                " and seat_column = " + column_num;
        ResultSet re = statement.executeQuery(sql);
        if (re.next()) {
            return re.getInt(1);
        }
        return -1;
    } catch (SQLException e) {
        e.printStackTrace();
        return -1;
    }
}

```

```

    }

    public boolean isTicket(int ticket_id) {
        try {
            Statement statement = connection.createStatement();
            String sql = "select use_flag from tb_ticket where tid = " + ticket_id;
            ResultSet re = statement.executeQuery(sql);
            if (re.next()) {
                return true;
            }
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return false;
    }

    public boolean isUsedTicket(int ticket_id) {
        try {
            Statement statement = connection.createStatement();
            String sql = "select use_flag from tb_ticket where tid = " + ticket_id;
            ResultSet re = statement.executeQuery(sql);
            if (re.next()) {
                return re.getBoolean(1);
            }
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return false;
    }

    public boolean deleteTicketInfo(int ticket_id) {
        try {
            Statement statement = connection.createStatement();
            String sql = "delete from tb_ticket where tid = " + ticket_id;
            statement.executeUpdate(sql);
            return true;
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }
}

```