

任务书

1 软件功能学习部分（必做题）

完成下列 1~2 题，并在实践报告中叙述过程，可适当辅以插图（控制在 A4 三页篇幅以内）

- 1) 练习 sqlserver 的两种完全备份方式：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 Sql 练习部分（必做题）

2.1 建表

1) 假设某微博平台的数据库中有下列关系，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：

用户【用户 ID，姓名，性别，出生年份，所在城市】记录所有注册用户的基本信息，英文表名和字段名如下：

USER(UID 整型, NAME 字符串, SEX 一位汉字, BYEAR 整型, CITY 字符串);

分类【分类 ID，分类名称】记录微博平台中所有可能涉及的微博的类型，例如文学、艺术、军事等，英文表名和字段名如下：

LABEL(LID 整型, LNAME 字符串);

博文【博文 ID，标题，用户 ID，年，月，日，正文】记录每一篇微博的基本信息，英文表名和字段名如下：

MBLOG(BID 整型, TITLE 字符串, UID 整型, PYEAR 整型, PMONTH 整型, PDAY 整型, CONT 字符串),

写出该关系的建表 SQL 语句；

博文标注【博文 ID，分类 ID】记录每一篇微博的作者给该微博贴上的分类标签，一篇微博可以涉及不止一种分类，英文表名和字段名如下：

B_L(BID 整型, LID 整型);

关注【用户 ID，被关注用户 ID】记录每位用户关注的其他用户，每位用户可关注多人，英文表名和字段名如下：

FOLLOW(UID 整型, UIDFLED 整型);

好友【用户 ID， 好友 ID】记录每位用户的好友（可多个），英文表名和字段名如下：

FRIENDS(*UID* 整型, *FUID* 整型);

订阅【用户 ID, 订阅分类 ID】记录用户订阅的每一种分类，英文表名和字段名如下：

SUB(*UID* 整型, *LID* 整型);

点赞【用户 ID, 博文 ID】记录用户点赞的每一篇微博，英文表名和字段名如下：

THUMB(*UID* 整型, *BID* 整型),

写出该关系的建表 SQL 语句；

头条【年，月，日，博文 ID, 顺序号】记录每一天的热度排名前十的博文 ID 号以及该博文在热度前十名中的排名，英文表名和字段名如下：

TOPDAY(*TYEAR* 整型, *TMONTH* 整型, *TDAY* 整型, *BID* 整型, *TNO* 整型)。

2) 观察性实验

用户在订阅分类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。

2.2 数据更新

1) 分别用一条 sql 语句完成对博文表基本的增、删、改的操作；

2) 批处理操作

将关注 3 号用户的用户信息插入到一个自定义的新表 FANS_3 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。

在后续的上机实验环节，通过导入导出或者备份机制实现前次上机环节的数据恢复。

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) 触发器实验

编写一个触发器，用于实现对点赞表的完整性控制规则：当插入或者被点赞博文时，如果博文作者就是点赞者本人，则拒绝执行。

2.3 查询

请分别用一条 SQL 语句完成下列各个小题的需求：

- 1) 查询“张三”用户关注的所有用户的 ID 号、姓名、性别、出生年份、所在城市，并且按照出生年份的降序排列，同一个年份的则按照用户 ID 号升序排列。
- 2) 查找没有被任何人点赞的博文 ID、标题以及发表者姓名，并将结果按照标题字符顺序排列。
- 3) 查找 2000 年以后出生的武汉市用户发表的进入过头条的博文 ID；
- 4) 查找订阅了所有分类的用户 ID；
- 5) 查找出生年份小于 1970 年或者大于 2010 年的用户 ID、出生年份、所在城市，要求 where 子句中只能有一个条件表达式；
- 6) 统计每个城市的用户数；
- 7) 统计每个城市的每个出生年份的用户数，并将结果按照城市的升序排列，同一个城市按照出生用户数的降序排列其相应的年份；
- 8) 查找被点赞数超过 10 的博文 ID 号；
- 9) 查找被 2000 年后出生的用户点赞数超过 10 的博文 ID 号；
- 10) 查找被 2000 年后出生的用户点赞数超过 10 的每篇博文的进入头条的次数；
- 11) 查找订阅了文学、艺术、哲学、音乐中至少一种分类的用户 ID，要求不能使用嵌套查询，且 where 子句中最多只能包含两个条件；
- 12) 查找标题中包含了“最多地铁站”和“_华中科技大学”两个词的博文基本信息；
- 13) 查找所有相互关注的用户对的两个 ID 号，要求不能使用嵌套查询；
- 14) 查找好友圈包含了 5 号用户好友圈的用户 ID；
- 15) 查找 2019 年 4 月 20 日每一篇头条博文的 ID 号、标题以及该博文的每一个分类 ID，要求即使该博文没有任何分类 ID 也要输出其 ID 号、标题；
- 16) 查找至少有 3 名共同好友的所有用户对的两个 ID 号。
- 17) 创建视图：查阅 DBMS 内部函数，创建一个显示当日热度排名前十的微博信息的视图，其中的属性包括：博文 ID、博文标题、发表者 ID、发表者姓名、被点赞数。

2.4 了解系统的查询性能分析功能（选做，各班指导教师可适当调整）

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。

2.5 DBMS 函数及存储过程和事务（选做，各班指导教师可适当调整）

- 1) 编写一个依据用户 ID 号计算其发表的博文进入头条的累计天数的 DBMS 自定义函数，并利用其查询 2000 年后出生的上述头条累计天数达到 100 天的所有用户 ID。
- 2) 建立关系“点赞排行榜【博文 ID，当天点赞人数】”，里面存储系统当天点

赞数前十名的博文 ID 及其点赞人数，尝试编写一个 DBMS 的存储过程，通过该存储过程更新该表。

3) 尝试在 DBMS 的交互式界面中验证事务机制的执行效果。

3 数据库应用系统设计（必做）

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

下列题目作为选题背景参考，也可依据这些题目拟定一个自己感兴趣的具有类似工作量和复杂程度的课题。

题目 1：电信收费管理系统

采用 B/S 或 C/S 模式实现一个电信收费管理系统软件。实现电信套餐种类、用户信息、客服代表、收款员等信息的管理。

要求：

- 1) 实现不同权限的浏览和更新。
- 2) 实现用户扣、缴费情况及帐户余额的查询。
- 3) 实现欠款用户使用状态的自动改变。
- 4) 实现客服代表的业绩统计功能。
- 5) 提供至少两种风格的查询报表。

题目 2：员工培训管理系统

采用 B/S 或 C/S 模式实现一个员工培训管理系统软件。完成培训计划制定、培训导师安排、学员分批次注册、学员培训期间的考勤、考核与工资等信息的管理。

要求：

- 1) 培训计划应包括一系列具有先后依赖关系的课程，且培训计划可更新。
- 2) 实现不同权限的浏览和更新。
- 3) 实现考勤、考核、计算工资的功能。
- 4) 能够综合评价培训导师和学员的业绩。
- 5) 实现员工培训不合格后的再次培训管理。

题目 3：汽车租赁信息系统

采用 B/S 或 C/S 模式实现一个汽车租赁信息系统。完成用户、车辆、经手员工、租借情况、车辆损毁情况、交通违规罚款等信息的管理。

要求：

- 1) 实现不同权限的浏览和更新。
- 2) 能够根据车辆使用情况计算押金退还金额。
- 3) 能查询客户的租借历史记录, 并进行信誉度评价, 进行会员制和非会员制的客户管理。
- 4) 能够管理车辆报修信息;
- 5) 能够生成租借公司的日、月、季度、年财务报表。

题目 4: 医院管理系统

采用 B/S 或 C/S 模式实现一个医院管理系统。完成药品、诊疗、医师、病人、病房等信息的管理。

要求:

- 1) 提供面向公众的导医和收费标准明细查询的功能。
- 2) 挂号、收费、诊疗人员等具有不同的查询和修改权限。
- 3) 按照看病的基本流程(例如: 预约——挂号——门诊——检查——复诊——住院治疗——出院结算)进行信息管理。
- 4) 提供病人收费汇总清单, 提供各种药品或检查项目的使用情况汇总;
- 5) 提供医院各部门财务报表及医院整体财务报表, 并且分日明细表和月、年汇总表。

题目 5: 田径运动会管理系统

采用 B/S 或 C/S 模式实现一个田径运动会管理系统。完成参赛单位(国际比赛的单位为国家, 国内比赛的单位为省份)、运动员、裁判、比赛项目、比赛成绩的信息管理。

要求:

- 1) 提供不同权限的录入、查询界面;
- 2) 比赛采用分组晋级制度, 例如 A 组、B 组、1/8、1/4、半决赛、决赛。能够维护、查询赛事日程表;
- 3) 能够查询每项比赛的世界记录、本赛事历史记录;
- 4) 能够查询每次小组赛或者半决赛、决赛的运动员个人信息及上一轮成绩;
- 5) 能够统计全能赛项的个人成绩和名次;
- 6) 能够统计各参赛单位的整体情况。

题目 6: 机票预定系统

1、系统功能的基本要求:

- 每个航班信息的输入。
- 每个航班的座位信息的输入;

- 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；
- 旅客在飞机起飞前一天凭取票通知交款取票；
- 旅客能够退订机票；
- 能够查询每个航班的预定情况、计算航班的满座率。

2、数据库要求：在数据库中至少应该包含下列数据表：

- 航班信息表；
- 航班座位情况表；
- 旅客订票信息表；
- 取票通知表；
- 帐单。

设计一个 B/S 或 C/S 模式的系统实现上述功能。

题目 7：工资管理系统

1、系统功能的基本要求：

- 员工每个工种基本工资的设置
- 加班津贴管理，根据加班时间和类型给予不同的加班津贴；
- 按照不同工种的基本工资情况、员工的考勤情况产生员工的每月的月工资；
- 员工年终奖金的生成，员工的年终奖金计算公式 = (员工本年度的工资总和 + 津贴的总和) / 12；
- 企业工资报表。能够查询单个员工的工资情况、每个部门的工资情况、按月的工资统计，并能够打印；

2、数据库要求：在数据库中至少应该包含下列数据表：

- 员工考勤情况表；
- 员工工种情况表，反映员工的工种、等级，基本工资等信息；
- 员工津贴信息表，反映员工的加班时间，加班类别、加班天数、津贴情况等；
- 员工基本信息表
- 员工月工资表。

设计一个 B/S 或 C/S 模式的系统实现上述功能。

题目 8：网上销售系统

网上销售系统要求提供包括商品信息管理、查询、订购、销售等功能的网上交易平台，对客户和商店管理员应提供不同的操作界面和使用权限。具体功能包括：

- 商品信息管理：商店可以对商品信息进行管理，包括商品的类别、名称、描述信息、售价、图片、折扣等。
- 客户信息管理：客户可以自助注册并管理自己的个人信息。
- 商品查询：客户可以通过多种方式查询并且挑选网上商店出售的商品，通过在网上填写并确认订单的方式来购买商品。

- 订单查询：客户可以管理自己的订单信息，查询订单的处理情况。
- 订单处理：商店可以对客户的订单信息、汇款单信息进行审核，以确定是否发货，并修改订单状态。

设计一个 B/S 模式的系统实现上述功能。

题目 9：仓储管理系统

实现一个仓库库存货品信息管理系统软件。仓库的日常工作包括货品的入库和出库。入库要由采购人员提供进货单，进货单经过审核人员审核验收后方能进行货品入库。出库要由销售人员提供出货单，经过审核人员审核批准后才能提货。当销售人员需要提货而货品的库存量不足时可先进行缺货登记，当有相应货品入库时，按缺货登记时间顺序处理出货请求。要求：

- 能实现库存货品信息的管理，货品的相关信息包括：货品号、货品名、存放地、货品库存量、生产厂家等。
- 实现进货单的填写、修改、审核和查询等功能。
- 实现出货单的填写、修改、审核和查询等功能。
- 实现缺货登记、查询和处理。
- 根据不同用户身份提供不同的操作权限和界面。

设计一个 B/S 或 C/S 模式的系统实现上述功能。

题目 10：图书管理系统

假设图书馆的工作人员要处理下列日常工作：

- 借书：核实读者身份并检查是否存在下述情况：
 - 该读者借书的数额超标；
 - 该读者所借的书逾期未还；
 - 该读者曾因借书过期被罚款而未交；
 如不存在上述情况，则登记借书信息；
- 还书：检查所还图书是否损坏或过期，是则登记罚单信息并打印罚单，在交纳罚金前，不允许该读者继续借书。若图书损坏，注销该图书信息，否则进行还书登记。
- 罚款：根据罚单收取罚金，同时取消该读者的借书限制。
- 图书信息维护：新书上架、旧书下架及图书信息查询。
- 读者信息维护：录入、注销、修改及查询读者信息。

此外，图书馆还应向读者提供下列基本功能：

- 查询图书信息；
- 查询自己的基本信息和借书记录；
- 续借；

设计一个 B/S 或 C/S 模式的系统实现上述功能。

题目 11：超市收银系统

假设一家小型超市的收银台（前台）要完成下列日常工作：

- 收银：收银员输入顾客的会员卡卡号（若有卡）、所购商品的货号等信息，系统根据这些信息获取相应的价格信息并计算应收取的总金额。完成收银后，记录交易信息，修改有关种类商品的剩余量以及该持卡顾客的消费情况。
- 发卡：顾客可交纳一定的费用（如 50 元）办理一张会员卡，以后在该商场购物可凭卡享受 9 折优惠。如果一个未持卡顾客一次购物满 1000 元，可为其免费发放一张会员卡，每张卡的优惠期为一年，一年内消费达到一定金额的可继续享受下一年的优惠。
- 款项盘存：收银员下班或交接班前对本收银台中本班次收取的款额进行盘存，明确责任。

此外，还应提供下列后台功能：

- 商品信息的录入、修改、删除和查询等。
- 收银员身份及口令管理。

设计一个 C/S 模式的系统实现上述功能。

目 录

1	课程任务概述.....	1
2	软件功能学习.....	2
	2.1 任务要求.....	2
	2.2 完成过程.....	2
	2.3 任务总结.....	4
3	SQL 练习	5
	3.1 任务要求.....	5
	3.2 完成过程.....	5
	3.3 任务总结.....	33
4	综合实践任务.....	35
	4.1 系统设计目标.....	35
	4.2 需求分析.....	35
	4.3 总体设计.....	37
	4.4 数据库设计.....	38
	4.5 详细设计与实现.....	41
	4.6 系统测试.....	49
	4.7 系统设计与实现总结.....	55
5	课程总结.....	56
6	附录.....	58

1 课程任务概述

本次数据库系统原理实践实验有三部分，第一部分是软件功能学习部分，第二部分是 SQL 练习部分。这两个部分是数据库系统原理实践的基础部分，大部分操作都是有关概念的操作，比如触发器的创建，函数与存储过程的创建，事务的运用等。

1. 软件功能学习部分

练习 sqlserver 的两种完全备份方式：数据和日志文件的脱机备份、系统的备份功能。练习在新增的数据库上增加用户并配置权限的操作，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。

通过这两个实验，熟悉并掌握 SQL Server 的基本功能，为之后运用数据的恢复，用户的创建打下基础。

2. SQL 练习部分

根据任务要求，在 DBMS 中创建相应的数据库关系，包括主码和外码的说明。并且通过一条 SQL 语句检查外键设置是否正确，并向表中插入适量数据，方便后续实验。

熟悉并掌握数据更新，分别用一条 sql 语句完成对博文表基本的增、删、改的操作。掌握批处理操作以及数据导入导出操作。了解当删除或修改重复元组时发生的现象。编写触发器实现对某张表的完整性控制规则。

熟练掌握数据库查询语句，完成所有的练习。

了解系统的查询性能分析功能，并给出简单的分析。

编写用户自定义函数，存储过程，并验证事务机制的执行效果。

2 软件功能学习

2.1 任务要求

练习 sqlserver 的两种完全备份方式：数据和日志文件的脱机备份、系统的备份功能。练习在新增的数据库上增加用户并配置权限的操作，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。

通过这两个实验，熟悉并掌握 SQL Server 的基本功能，为之后运用数据的恢复，用户的创建打下基础。

2.2 完成过程

2.2.1 sqlserver 备份

在需要备份的数据库上点击右键，找到任务栏，之后点击备份，如图 2-1。



图 2-1 备份功能位置

1. 数据库完整备份

如图 2-2，备份类型选择为完整，选择备份文件，确定即可。

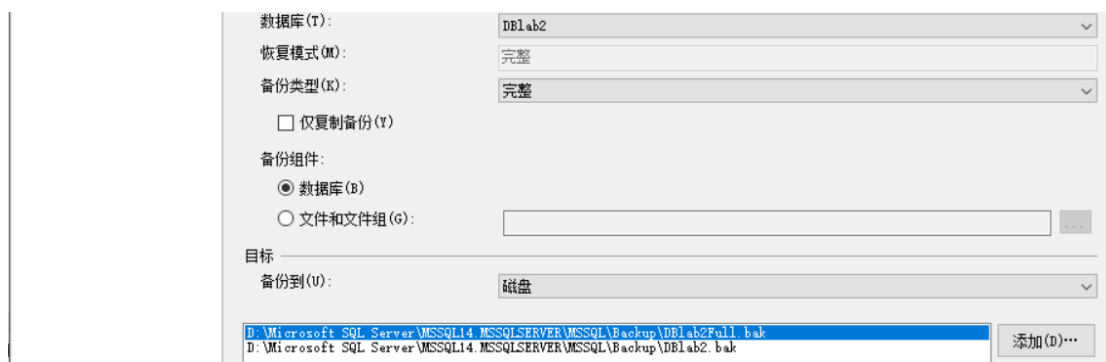


图 2-2 完整备份

2. 日志文件备份

如图 2-3，添加备份文件，后缀名为 trn。然后选择备份类型为事务日志，选择刚刚创建的备份文件 DBlab2Log.trn 为备份路径，确定即可。



图 2-3 事务日志备份

2.2.2 新增用户并配置权限

如图 2-4，在登录名上右键，选择新建登录名。



图 2-4 新建登录名位置

如图 2-5，输入登录名，选择 SQL Server 身份验证，默认数据库选择 DBlab2，然后点击确定即可。

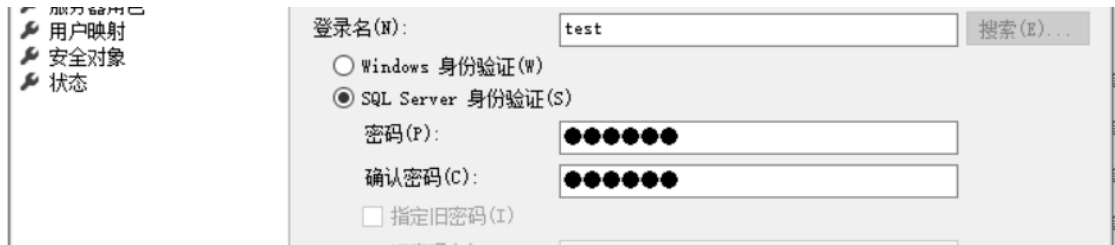


图 2-5 新建登录名

然后在用户映射选项卡里面，选择映射到数据库 DBlab2。如图 2-6。



图 2-6 建立用户映射

然后在需要设置权限的表上右键，选择属性，然后在权限里面选择用户 test，并选择相应的权限，如图 2-7，图 2-8。

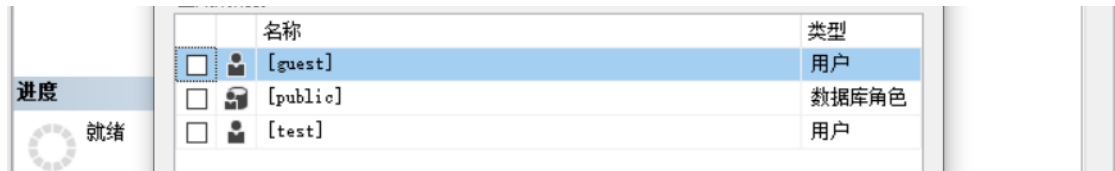


图 2-7 设置权限



图 2-8 设置权限

最后使用 test 用户登录，发现只有之前选择的表能够显示出来，然后使用 select 与 insert 语句，发现都无法成功执行。如图 2-9，图 2-10。

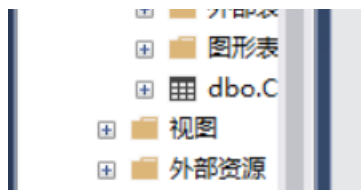


图 2-9 测试用户权限

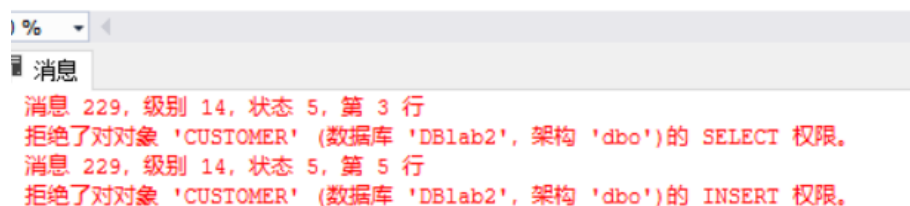
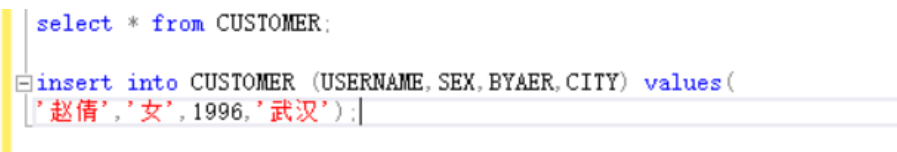


图 2-10 测试用户权限

2.3 任务总结

实验主要是通过查阅网上的资料，结合自己的经验完成的。在数据库备份方式的实验中，了解到了数据库备份方式的不同，完整数据库备份固然是很好，但是一旦数据量过大，这种备份方式花费的时间就变得很长，取而代之的是事务日志的备份方式，这种方式极大地提高了备份效率。

在创建用户的实验中，遇到的问题是，建立了用户之后不能登录，通过上网查阅资料，发现是因为自己的某些服务没有打开，比如 SQL Server 代理。完成了这个实验，知道了用户与数据库之间存在某种关系，而有些权限是只有数据库管理员才拥有的，一般用户的权限被限制。

3 SQL 练习

3.1 任务要求

根据任务要求，在 DBMS 中创建相应的数据库关系，包括主码和外码的说明。并且通过一条 SQL 语句检查外键设置是否正确，并向表中插入适量数据，方便后续实验。

熟悉并掌握数据更新，分别用一条 sql 语句完成对博文表基本的增、删、改的操作。掌握批处理操作以及数据导入导出操作。了解当删除或修改重复元组时发生的现象。编写触发器实现对某张表的完整性控制规则。

熟练掌握数据库查询语句，完成所有的练习。

了解系统的查询性能分析功能，并给出简单的分析。

编写用户自定义函数，存储过程，并验证事务机制的执行效果。

3.2 完成过程

3.2.1 数据定义

1. 建立表

用户【用户 ID，姓名，性别，出生年份，所在城市】记录所有注册用户的基本信息，英文表名和字段名如下：

USER(*UID* 整型, *NAME* 字符串, *SEX* 一位汉字, *BYEAR* 整型, *CITY* 字符串);

分类【分类 ID，分类名称】记录微博平台中所有可能涉及的微博的类型，例如文学、艺术、军事等，英文表名和字段名如下：

LABEL(*LID* 整型, *LNAME* 字符串);

博文【博文 ID，标题，用户 ID，年，月，日，正文】记录每一篇微博的基本信息，英文表名和字段名如下：

MBLOG(*BID* 整型, *TITLE* 字符串, *UID* 整型, *PYEAR* 整型, *PMONTH* 整型, *PDAY* 整型, *CONT* 字符串);

写出该关系的建表 SQL 语句；

博文标注【博文 ID，分类 ID】记录每一篇微博的作者给该微博贴上的分类标签，一篇微博可以涉及不止一种分类，英文表名和字段名如下：

B_L(*BID* 整型, *LID* 整型);

关注【用户 ID，被关注用户 ID】记录每位用户关注的其他用户，每位用户可关注多人，英文表名和字段名如下：

FOLLOW(*UID* 整型, *UIDFLED* 整型);

好友【用户 ID, 好友 ID】记录每位用户的好友（可多个），英文表名和字段名如下：

FRIENDS(*UID* 整型, *FUID* 整型);

订阅【用户 ID, 订阅分类 ID】记录用户订阅的每一种分类，英文表名和字段名如下：

SUB(*UID* 整型, *LID* 整型);

点赞【用户 ID, 博文 ID】记录用户点赞的每一篇微博，英文表名和字段名如下：

THUMB(*UID* 整型, *BID* 整型)。

头条【年, 月, 日, 博文 ID, 顺序号】记录每一天的热度排名前十的博文 ID 号以及该博文在热度前十名中的排名，英文表名和字段名如下：

TOPDAY(*TYEAR* 整型, *TMONTH* 整型, *TDAY* 整型, *BID* 整型, *TNO* 整型)。

根据以上要求，创建对应的表。注意主键和外键的设置。SQL 语句如下：

```
1. CREATE TABLE CUSTOMER(  
2.   UID int NOT NULL PRIMARY KEY IDENTITY,  
3.   USERNAME char(20) NOT NULL,  
4.   SEX char(2),  
5.   BYAER int NOT NULL,  
6.   CITY char(20));  
7.  
8. CREATE TABLE LABEL(  
9.   LID int NOT NULL PRIMARY KEY IDENTITY,  
10.  LNAME char(20) NOT NULL);  
11.  
12. CREATE TABLE MBLOG(  
13.  BID int NOT NULL PRIMARY KEY IDENTITY,  
14.  TITLE char(30) NOT NULL,  
15.  UID int NOT NULL,  
16.  PYEAR int NOT NULL,  
17.  PMONTH int NOT NULL,  
18.  PDAY int NOT NULL,  
19.  CONT char(200),  
20.  FOREIGN KEY (UID) REFERENCES CUSTOMER(UID));  
21.  
22. CREATE TABLE B_L(  
23.  BID int NOT NULL ,  
24.  LID int NOT NULL ,  
25.  PRIMARY KEY (BID , LID),  
26.  FOREIGN KEY (BID) REFERENCES MBLOG(BID),  
27.  FOREIGN KEY (LID) REFERENCES LABEL(LID));  
28.  
29. CREATE TABLE FOLLOW(  
30.  UID int NOT NULL ,  
31.  UIDFLED int NOT NULL,  
32.  PRIMARY KEY (UID, UIDFLED),
```

```

33. FOREIGN KEY (UID) REFERENCES CUSTOMER (UID));
34.
35. CREATE TABLE FRIENDS (
36.   UID INT NOT NULL,
37.   FUID INT NOT NULL,
38.   PRIMARY KEY(UID,FUID),
39.   FOREIGN KEY (UID) REFERENCES CUSTOMER (UID));
40.
41. CREATE TABLE SUB (
42.   UID INT NOT NULL,
43.   LID INT NOT NULL,
44.   PRIMARY KEY (UID, LID),
45.   FOREIGN KEY (UID) REFERENCES CUSTOMER (UID),
46.   FOREIGN KEY (LID) REFERENCES LABEL (LID));
47.
48. CREATE TABLE THUMB (
49.   UID INT NOT NULL,
50.   BID INT NOT NULL,
51.   PRIMARY KEY (UID, BID),
52.   FOREIGN KEY (UID) REFERENCES CUSTOMER (UID),
53.   FOREIGN KEY (BID) REFERENCES MBLOG (BID));
54.
55. CREATE TABLE toptoday(
56.   TYEAR INT NOT NULL,
57.   TMONTH INT NOT NULL,
58.   TDAY INT NOT NULL,
59.   BID INT NOT NULL,
60.   TNO INT NOT NULL,
61.   PRIMARY KEY (TYEAR, TMONTH, TDAY, BID),
62.   FOREIGN KEY (BID) REFERENCES MBLOG(BID));

```

需要注意的问题时一开始我把 toptoday 的主键设置为了 BID，后来发现这是不行的，因为一条微博可能不同的时间上了头条，因此主键不能为 BID。创建完表格后，查看对象资源管理器中的表可以看到已经创建了所有表。

2. 观察性实验

如图 3-1，向 sub 里插入 (1,12)，发现语句终止。因为在 LABEL 表里，LID 没有 12，而 sub 的 LID 是外键，因此语句出错。

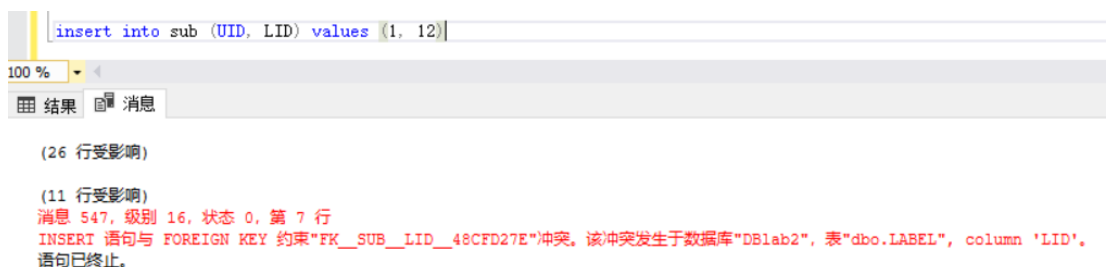


图 3-1 建表观察性实验

3. 数据准备

向表中插入适当数据，为了报告的紧凑性，在此仅给出部分插入语句。

```

1. insert into CUSTOMER (USERNAME,SEX,BYAE,CITY) values(
2.   '赵倩','女',1996,'武汉'),(
3.   '黄超','男',1986,'武汉'),(
4.   '安城','女',1993,'北京'),(
5.   '李源','女',1989,'上海'),(
6.   '赵齐','男',1991,'郑州'),(

```



```

7. '李旺','男',1992,'上海'),(
8. '陈锋','男',1996,'天津'),(
9. '陈冕','女',1999,'济南'),(
10. '黄玲玲','女',1985,'成都'),(
11. '孙中飞','男',1988,'成都'),(
12. '钱婷婷','女',1999,'重庆'),(
13. '李雯','女',2002,'西安');
14.
15. insert into label (LNAME) values (
16. '搞笑'),(
17. '社会'),(
18. '时尚'),(
19. '电影'),(
20. '美女'),(
21. '体育'),(
22. '动漫');
23.
24. insert into MBLOG (TITLE, UID, PYEAR, PMONTH, PDAY, CONT) values(
25. '校长带全校看复联 4', 1, 2019, 4, 26, '复联 4 上映, 校长包场'),(
26. '生活大爆炸结局', 3, 2019, 4, 25, '生活大爆炸最后一次录制'),(
27. '风筝', 5, 2019, 4, 20, '竟然有人放航母风筝'),(
28. '复联 4 彩蛋', 2, 2019, 4, 24, '复联 4 没有彩蛋'),(
29. '复联 4 剧透', 2, 2019, 4, 24, '灭霸死了'),(
30. '谢文骏 110 米跨栏夺冠', 8, 2019, 4, 25, '谢文骏打破记录'),(
31. '林俊杰圣所', 2, 2019, 4, 26, '要去石家庄了'),(
32. '邓超盘腿', 1, 2019, 4, 26, '邓超搞笑自拍');

```

3.2.2 数据更新

1. 分别用一条 sql 语句完成对博文表基本的增、删、改的操作；

```

1. --insert into mblog
2. insert into MBLOG (TITLE, UID, PYEAR, PMONTH, PDAY, CONT) values(
3. '李敏镐退伍', 1, 2019, 4, 25, '李敏镐退伍');
4.
5. --delete a row from mblog
6. delete from MBLOG
7. where bid = 9;
8.
9. --update a row from mblog
10. update MBLOG
11. set CONT = '李敏镐退伍啦！'
12. where bid = 9;

```

以上三条 SQL 语句完成了任务要求。

2. 批处理操作

将关注 3 号用户的用户信息插入到一个自定义的新表 FANS_3 中。SQL 语句如下：

```

1. CREATE TABLE fans_3(UID int NOT NULL PRIMARY KEY, USERNAME char(20) NOT NULL
, SEX char(2), BYAER int NOT NULL, CITY char(20));
2.
3. insert into fans_3 (UID, USERNAME, SEX, BYAER, CITY)
4. select UID, USERNAME, SEX, BYAER, CITY from CUSTOMER
5. where uid in (select uid from Follow
6. where UIDFLED = 3);
7.

```

```
8. select * from fans_3;
```

批处理插入就是 insert 后面不跟 values，而是直接跟 select，不过要保证的是插入的列数要和后面 select 的列数相同，不然会出错。结果如图 3-2，可以看到批处理操作正确执行。

结果		消息	
	UID	UIDFLED	
1	1	2	
2	1	3	
3	1	5	
4	2	1	
5	2	7	
6	3	2	
7	3	9	
8	4	8	
9	5	9	
10	6	2	
11	13	1	
12	13	2	
13	13	3	
14	13	5	
15	13	7	

	UID	USERNAME	SEX	BYAER	CITY
1	1	赵倩	女	1996	武汉
2	13	张三	男	1993	南京

图 3-2 批处理操作

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。

在需要导出数据的数据库上右键，选择任务→导出数据。数据源选择如图 3-3，

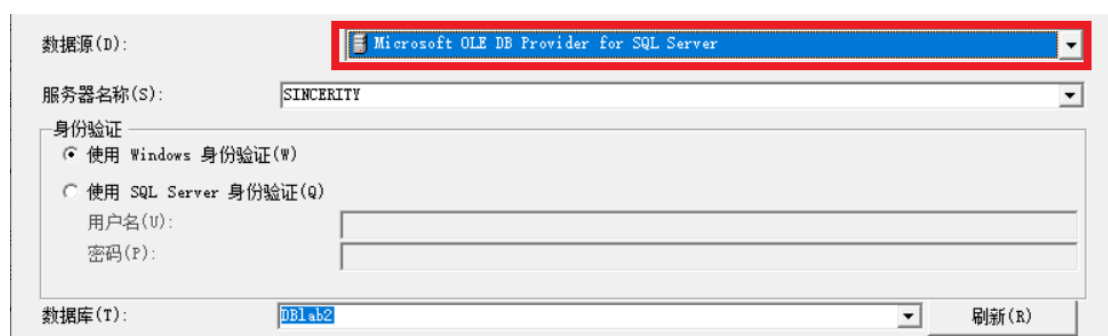


图 3-3 数据源选择

导出目标选择 Excel 格式，如图 3-4。

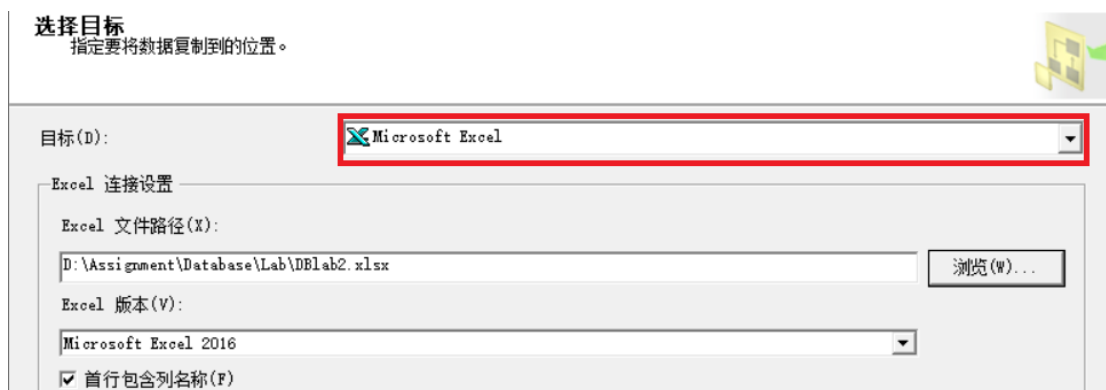


图 3-4 导出格式

之后按照步骤，选择默认设置即可。在选择表和视图的哪一步选择需要导出的数据即可。

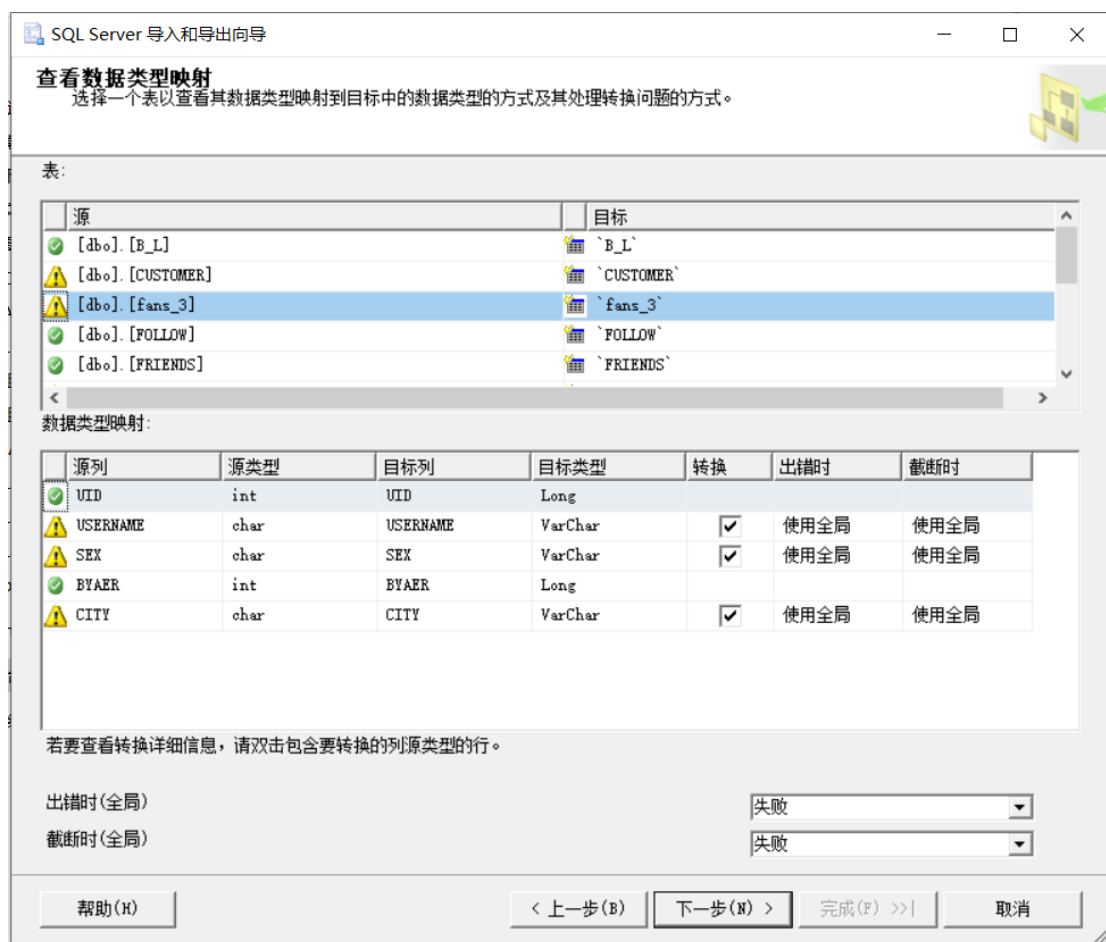


图 3-5 导出数据类型映射

最后完成数据导出，如图 3-6。

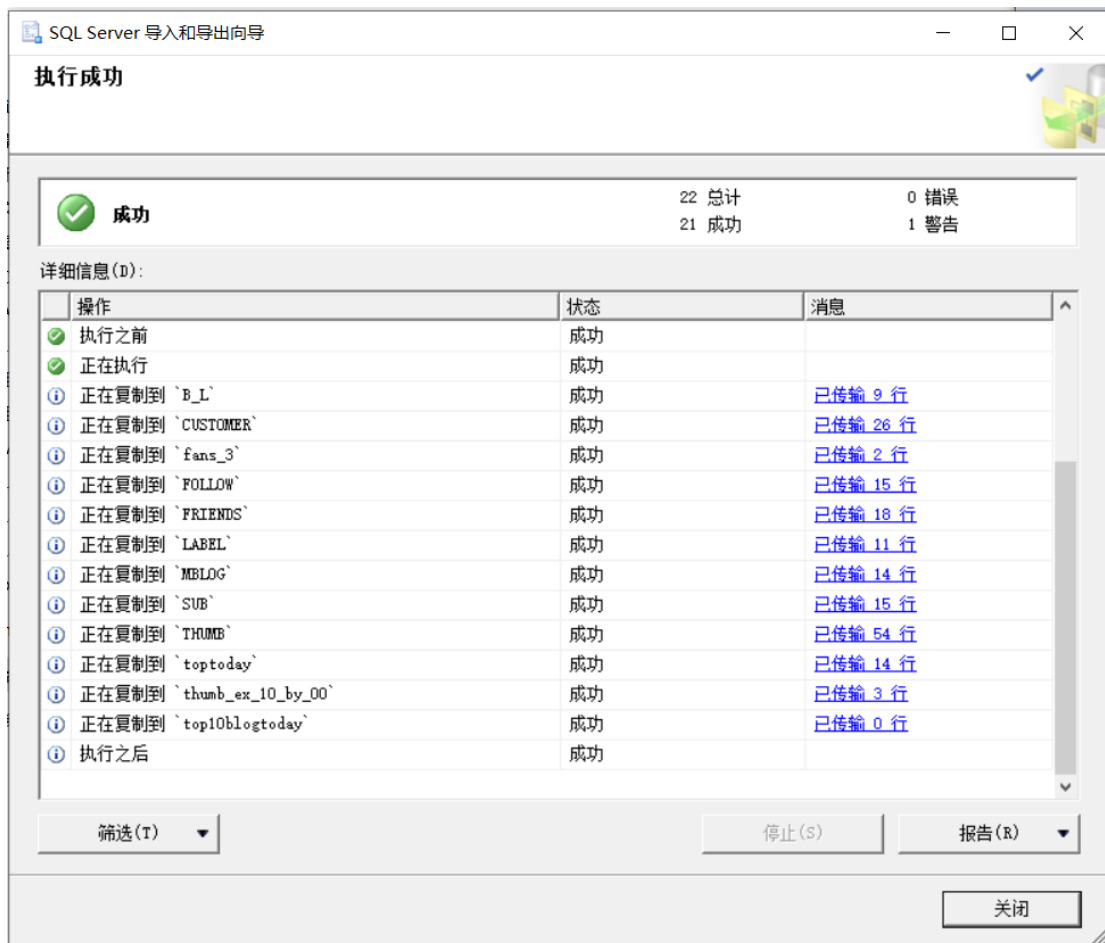


图 3-6 导出完成

导出完成后，会发现在目标位置生成了 excel 文档，这就是导出的数据，接下来就是新建测试数据库，将 excel 文档导入。在需要导入数据的数据库上右键，选择任务→导入数据。选择数据源如图 3-7，数据目标源如图 3-8。



图 3-7 导入数据源

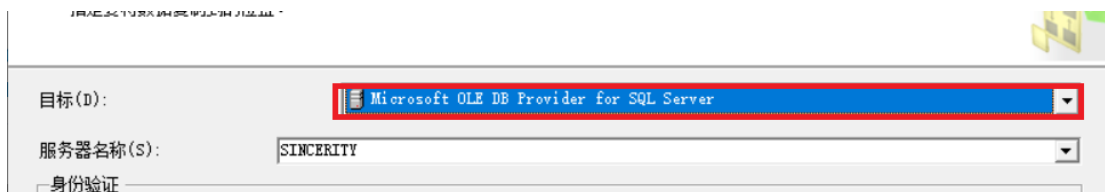


图 3-8 导入目标源

接着按照步骤，选择默认设置即可。在选择导入数据时，选择需要的数据，如图 3-9。

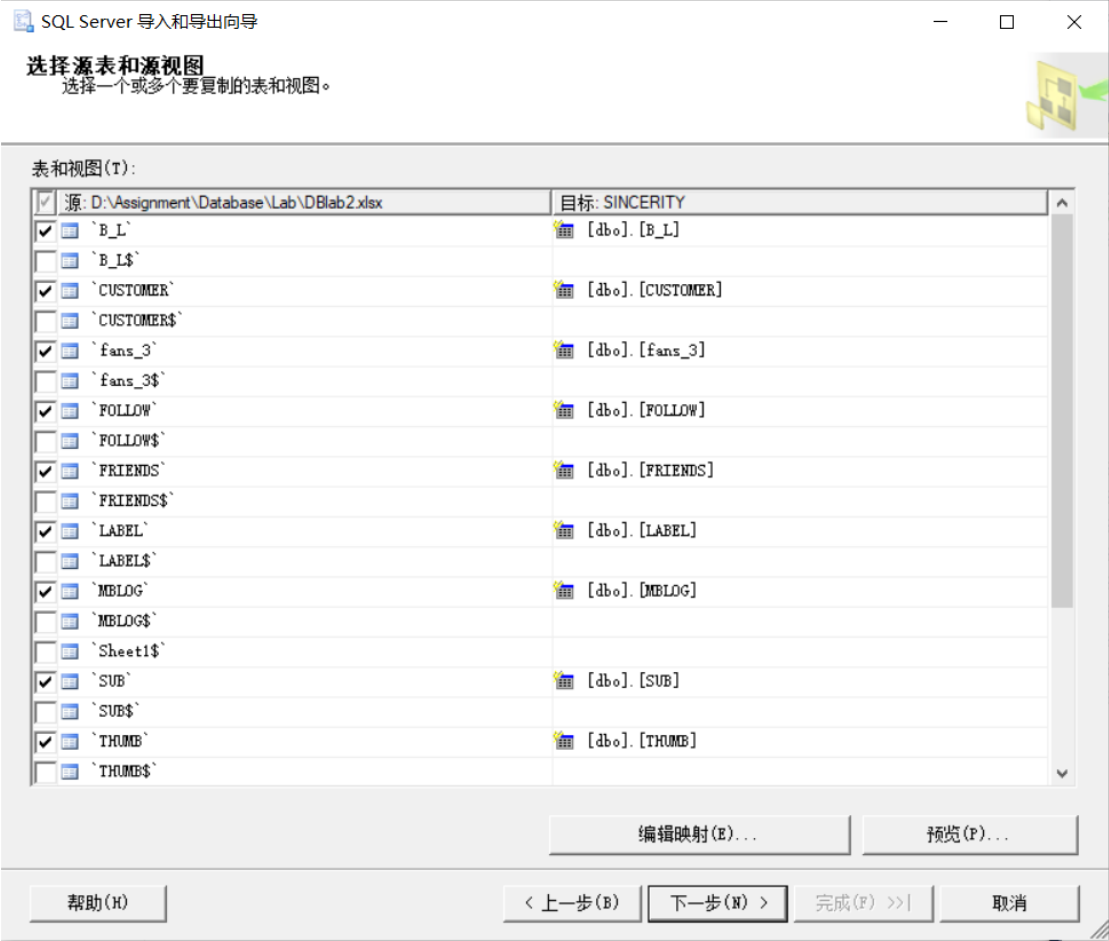


图 3-9 选择导入的表和视图

数据导入完成，如图 3-10。

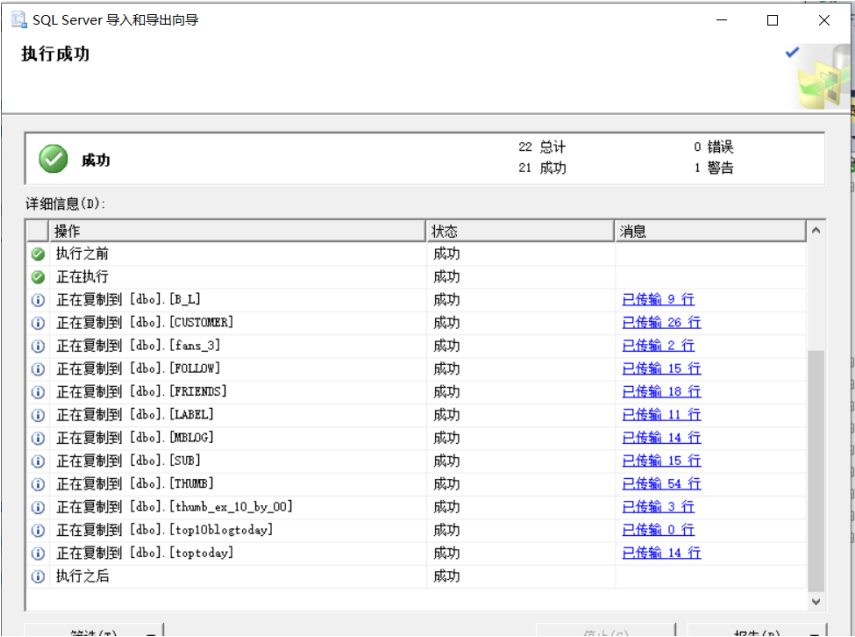


图 3-10 导入完成

测试新数据库，观察数据是否正确导入。



The screenshot shows a database management interface. At the top, there's a command window with the text 'use DBlab2test;' and 'select * from CUSTOMER;'. Below this, a tab labeled '结果' (Results) is active, displaying a table of 24 rows. The table has columns: UID, USERNAME, SEX, BYAER, and CITY. The first row is highlighted. At the bottom, a green status bar indicates '查询已成功执行。' (Query executed successfully).

	UID	USERNAME	SEX	BYAER	CITY
1	1	赵倩	女	1996	武汉
2	2	黄超	男	1986	武汉
3	3	安城	女	1993	北京
4	4	李源	女	1989	上海
5	5	赵齐	男	1991	郑州
6	6	李旺	男	1992	上海
7	7	陈锋	男	1996	天津
8	8	陈冕	女	1999	济南
9	9	黄玲玲	女	1985	成都
10	10	孙中飞	男	1988	成都
11	11	钱婷婷	女	1999	重庆
12	12	李雯	女	2002	西安
13	13	张三	男	1993	南京
14	14	李春生	女	1966	长春
15	15	李昂	男	2012	昆明
16	16	李武	男	1986	武汉
17	17	林兴钰	男	2001	武汉
18	18	林龙	男	2002	上海
19	19	林芯诗	女	2003	北京
20	20	林蕊玲	女	2004	深圳
21	21	谢灵彤	女	2005	广州
22	22	谢凤丹	女	2006	成都
23	23	潘辰光	男	2002	长沙
24	24	潘东	男	2003	长沙

图 3-11 测试导入是否正常

如图 3-11，导入正常。

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。

如图 3-12 为 SQL 语句执行之前，在表 test2 中插入重复元组（1,1）后表中的数据情况。在执行语句之后，在此查看表 test2 中情况，发现为空表。说明一条 SQL 语句会对重复元组同时进行修改，如图 3-13。

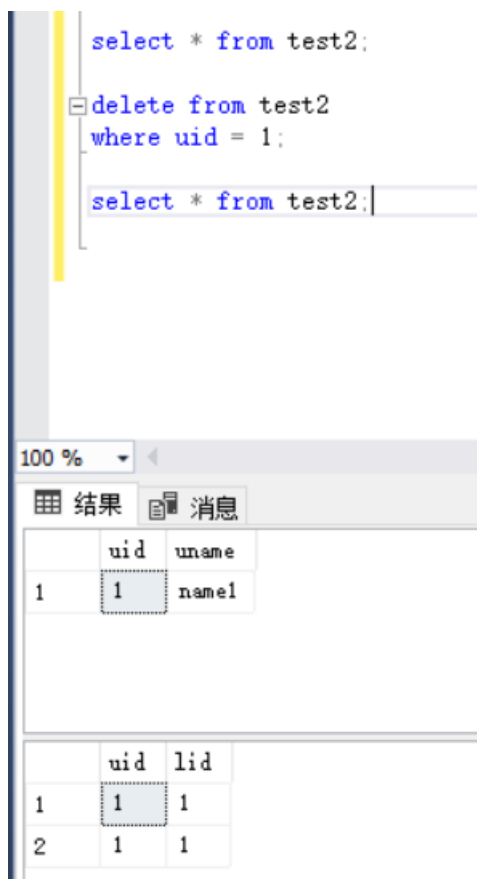


图 3-12 数据更新观察

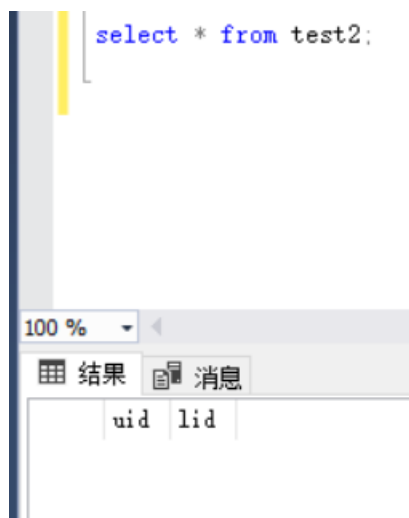


图 3-13 数据更新观察

同理，执行 update 语句也会同时对重复元组进行修改。

5. 触发器实验

编写一个触发器，用于实现对点赞表的完整性控制规则：当插入或者更新点赞关系时，如果博文作者就是点赞者本人，则拒绝执行。

我的触发器设置为在 insert 语句之后触发，如果当前表中存在作者对自己的博文点赞的情况，那么就回退一步（即此次 insert 无效）。创建触发器的 SQL 语句如下：

```

1. go
2. create trigger thumb_control
3. on thumb
4. after insert
5. as
6.     if exists(select thumb.UID from THUMB,MBLOG where THUMB.BID = MBLOG.BID
7. and THUMB.UID = MBLOG.UID)
8.     begin
9.         rollback transaction
10.    end
11. go

```

之后进行测试，如图 3-14，1 号博文由 1 号用户发布，因此当 1 号用户试图点赞 1 号博文时，触发了触发器，导致插入失败。

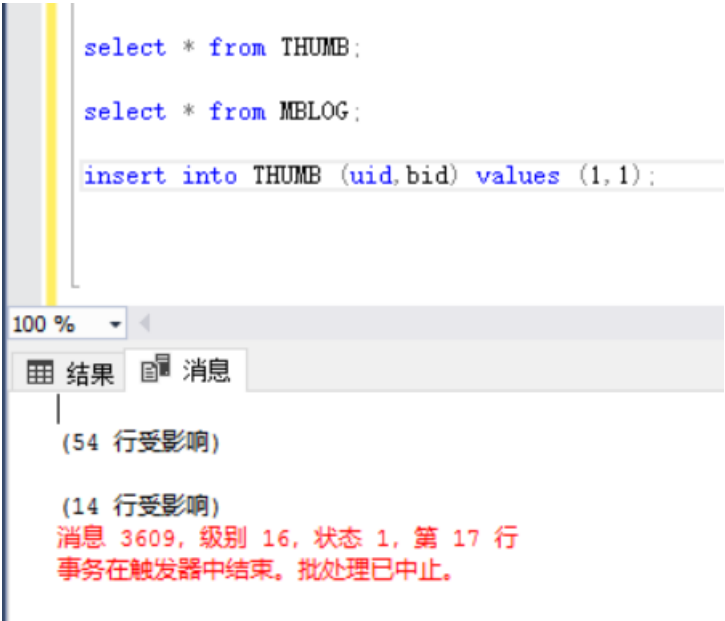


图 3-14 触发器测试

结果		消息	
	UID	BID	
1	1	2	
2	1	3	
3	1	8	
4	2	7	
5	2	8	
6	3	6	
7	3	8	
8	4	6	

	BID	TITLE	UID	PYEAR	PMONTH	PDAY	CONT
1	1	校长带全校看复联4	1	2019	4	26	复联4上映，校长包场
2	2	生活大爆炸结局	3	2019	4	25	生活大爆炸最后一次录制
3	3	风筝	5	2019	4	20	竟然有人偷航母风筝

图 3-15 点赞表与博文用户对应表

在新的任务书中，还需要考虑更新的操作，当更新后出现了自己点赞自己的

情况后，也需要撤回这次操作。

```
1. CREATE TRIGGER [dbo].[thumb_update]
2. ON [dbo].[THUMB]
3. AFTER update
4. AS
5. BEGIN
6.     begin tran
7.     if exists(select thumb.UID from THUMB,MBLOG where THUMB.BID = MBLOG.BID
8.     and THUMB.UID = MBLOG.UID)
9.     begin
10.         print'点赞失败，不能给自己点赞！'
11.         rollback transaction
12.     end
13.     else
14.     begin
15.         print'点赞成功！'
16.         commit tran
17.     end
18. END
```

之后进行测试，如图 3-16。

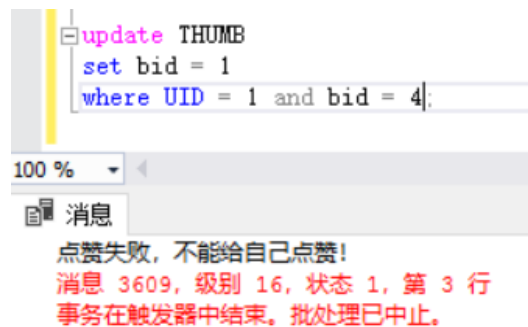


图 3-16 更新触发器测试

由上图看出，更新触发器测试完成。

3.2.3 查询

1. 查询“张三”用户关注的所有用户的 ID 号、姓名、性别、出生年份，所在城市，并且按照出生年份的降序排列，同一个年份的则按照用户 ID 号升序排列。

```
1. select CUSTOMER.UID, CUSTOMER.USERNAME, CUSTOMER.SEX, CUSTOMER.BYAER, CUSTOMER.CITY
2. from CUSTOMER inner join FOLLOW
3. on FOLLOW.UID = 13 and FOLLOW.UIDFLED = CUSTOMER.UID
4. Group by CUSTOMER.UID, CUSTOMER.USERNAME, CUSTOMER.SEX, CUSTOMER.BYAER, CUSTOMER.CITY
5. order by CUSTOMER.BYAER DESC, CUSTOMER.UID
```

通过 FOLLOW 表可以获取的张三用户关注的用户 ID 号，然后通过 FOLLOW 表与 CUSTOMER 表联结即可获取用户信息。

通过 select * from FOLLOW where UID = 13;可以验证获取的用户信息 ID 是否 FOLLOW 表中 ID 相同。如图 3-17。

	UID	USERNAME	SEX	BYAER	CITY
1	1	赵倩	女	1996	武汉
2	7	陈锋	男	1996	天津
3	3	安城	女	1993	北京
4	5	赵齐	男	1991	郑州
5	2	黄超	男	1986	武汉

	UID	UIDFLED
1	13	1
2	13	2
3	13	3
4	13	5
5	13	7

图 3-17 查询 task1

2. 查找没有被任何人点赞的博文 ID、标题以及发表者姓名，并将结果按照标题字符顺序排列。

```

1. select MBLOG.BID, MBLOG.TITLE, CUSTOMER.USERNAME
2. from MBLOG , CUSTOMER , thumb as p
3. where MBLOG.BID not in (select thumb.BID from thumb) and MBLOG.UID = CUSTOMER.UID
4. Group by MBLOG.BID, MBLOG.TITLE, CUSTOMER.USERNAME
5. order by MBLOG.TITLE

```

首先从 thumb 表里获取被赞过的博文 ID，使用 not in 来查找没有被赞过的博文 ID。之后将所需信息所在的表联结即可。

测试结果如图 3-18。

	BID	TITLE	USERNAME
1	8	邓超盘腿	赵倩
2	4	复联4彩蛋	黄超
3	5	复联4剧透	黄超
4	10	李敏镐退伍	赵倩
5	11	李敏镐退伍	赵倩

	UID	BID
1	1	2
2	1	3
3	2	7
4	3	6
5	4	6
6	5	6
7	6	1

	BID	TITLE	UID	PYEAR	PMONTH	PDAY	CONT
1	1	校长带全校看复联4	1	2019	4	26	复联4上映，校长包场
2	2	生活大爆炸结局	3	2019	4	25	生活大爆炸最后一次录制
3	3	风筝	5	2019	4	20	竟然有人放航母风筝
4	4	复联4彩蛋	2	2019	4	24	复联4没有彩蛋
5	5	复联4剧透	2	2019	4	24	灭霸死了
6	6	谢文骏110米跨...	8	2019	4	25	谢文骏打破记录
7	7	林俊杰圣所	2	2019	4	26	要去石家庄了
8	8	邓超盘腿	1	2019	4	26	邓超搞笑自拍
9	10	李敏镐退伍	1	2019	4	25	李敏镐退伍啦！
10	11	李敏镐退伍	1	2019	4	25	李敏镐退伍

点赞表

图 3-18 查询 task2

3. 查找 2000 年以后出生的武汉市用户发表的进入过头条的博文 ID;

```

1. select MBLOG.BID
2. from MBLOG,CUSTOMER,toptoday
3. where CUSTOMER.BYAE >= 2000 and MBLOG.UID = CUSTOMER.UID and MBLOG.BID in (
   select toptoday.BID from toptoday)
4. group by MBLOG.BID;
```

查询条件为 2000 年以后出生，武汉市用户，并且发表的博文进入过头条。

查询结果如图 3-19。（之前没有看到是武汉市，为了适应题目要求，特地将 12 号的城市修改为武汉，原来为西安）

结果						消息	
	TYEAR	TMONTH	TDAY	BID	TNO		
1	2019	4	26	1	2		
2	2019	4	26	2	3		
3	2019	4	26	12	1		

	BID	TITLE	UID	PYEAR	PMONTH	PDAY	CONT
8	8	邓超盘腿	1	2019	4	26	邓超搞笑自拍
9	10	李敏镐退伍	1	2019	4	25	李敏镐退伍啦!
10	11	李敏镐退伍	1	2019	4	25	李敏镐退伍
11	12	复联4上映, 校...	12	2019	4	26	复联4上映, 校长包场

	UID	USERNAME	SEX	BYAER	CITY
11	11	钱婷婷	女	1999	重庆
12	12	李雯	女	2002	武汉
13	13	张三	男	1993	南京
14	14	李春生	女	1966	长春

	BID
1	12

图 3-19 查询 task3

4. 查找订阅了所有分类的用户 ID;

```

1. select sub.UID
2. from sub
3. group by sub.UID
4. having count(sub.LID) = (select count(*) from LABEL);

```

查询条件就是订阅了所有分类。因此只需要 count 用户订阅的分类，并且与所有的分类数目相比较。

结果如图 3-20。

结果			消息	
	UID	LID		
1	1	1		
2	1	2		
3	1	3		
4	1	4		
5	1	5		
6	1	6		
7	1	7		
8	2	1		
9	3	2		

	UID
1	1

图 3-20 查询 task4

5. 查找出生年份小于 1970 年或者大于 2010 年的用户 ID、出生年份、所在城市，要求 where 子句中只能有一个条件表达式；

```
1. select CUSTOMER.UID, CUSTOMER.BYAER, CUSTOMER.CITY
2. from CUSTOMER
3. where CUSTOMER.BYAER not between 1970 and 2010;
```

要求 where 子句只能有一个条件表达式，因此想到了 between，前面加一个 not 就可以满足题目要求的小于 1970 年，或者大于 2010 年。

结果		消息			
	UID	USERNAME	SEX	BYAER	CITY
6	6	李旺	男	1992	上海
7	7	陈锋	男	1996	天津
8	8	陈冕	女	1999	济南
9	9	黄玲玲	女	1985	成都
10	10	孙中飞	男	1988	成都
11	11	钱婷婷	女	1999	重庆
12	12	李雯	女	2002	西安
13	13	张三	男	1993	南京
14	14	李春生	女	1966	长春
15	15	李昂	男	2012	昆明

	UID	BYAER	CITY
1	14	1966	长春
2	15	2012	昆明

图 3-21 查询 task5

6. 统计每个城市的用户数；

```
1. select count(CUSTOMER.UID) as population, CUSTOMER.CITY
2. from CUSTOMER
3. Group by CUSTOMER.CITY;
```

直接利用 count 以及 group by 即可统计每个城市用户数。
测试结果如图 3-22。

结果		消息			
	UID	USERNAME	SEX	BYAER	CITY
1	1	赵倩	女	1996	武汉
2	2	黄超	男	1986	武汉
3	3	安城	女	1993	北京
4	4	李源	女	1989	上海
5	5	赵齐	男	1991	郑州
6	6	李旺	男	1992	上海
7	7	陈锋	男	1996	天津
8	8	陈冕	女	1999	济南
9	9	黄玲玲	女	1985	成都
10	10	孙中飞	男	1988	成都
11	11	钱婷婷	女	1999	重庆
12	12	李雯	女	2002	西安
13	13	张三	男	1993	南京

	population	CITY
1	1	北京
2	2	成都
3	1	济南
4	1	南京
5	2	上海
6	1	天津
7	2	武汉
8	1	西安
9	1	郑州
10	1	重庆

图 3-22 查询 task6

7. 统计每个城市的每个出生年份的用户数，并将结果按照城市的升序排列，同一个城市按照出生用户数的降序排列其相应的年份；

```

1. select CUSTOMER.CITY, count(CUSTOMER.UID) as population, CUSTOMER.BYAER
2. from CUSTOMER
3. Group by CUSTOMER.CITY , CUSTOMER.BYAER
4. order by CUSTOMER.CITY ,count(CUSTOMER.UID) desc;

```

order by 默认按照升序排列，加上 desc 参数则按照降序排列。

结果如图 3-23。

结果		消息			
	UID	USERNAME	SEX	BYAER	CITY
9	9	黄玲玲	女	1985	成都
10	10	孙中飞	男	1988	成都
11	11	钱婷婷	女	1999	重庆
12	12	李雯	女	2002	西安
13	13	张三	男	1993	南京
14	14	李春生	女	1966	长春
15	15	李昂	男	2012	昆明
16	16	李武	男	1986	武汉

	CITY	population	BYAER
1	北京	1	1993
2	长春	1	1966
3	成都	1	1985
4	成都	1	1988
5	济南	1	1999
6	昆明	1	2012
7	南京	1	1993
8	上海	1	1989
9	上海	1	1992
10	天津	1	1996
11	武汉	2	1986
12	武汉	1	1996
13	西安	1	2002
14	郑州	1	1991
15	重庆	1	1999

图 3-23 查询 task7

8. 查找被点赞数超过 10 的博文 ID 号；

```

1. select thumb.BID
2. from thumb
3. group by thumb.BID
4. having count(thumb.BID) > 10;

```

直接使用 count 判断点赞数是否超过 10。

结果如图 3-24。

结果		消息
	UID	BID
1	1	2
2	1	3
3	1	8
4	2	7
5	2	8
6	3	6
7	3	8
8	4	6
9	4	8
10	5	6
11	5	8
12	6	1
13	6	8
14	7	8
15	8	8
16	9	8
17	10	8
18	11	8

	BID
1	8

图 3-24 查询 task8

9. 查找被 2000 年后出生的用户点赞数超过 10 的博文 ID 号；

```

1. go
2. CREATE VIEW thumb_ex_10_by_00 AS
3. select thumb.BID
4. from thumb,CUSTOMER
5. where CUSTOMER.BYAE >= 2000 and CUSTOMER.UID = THUMB.UID
6. group by thumb.BID
7. having count(thumb.UID) > 10;
8. go
9.
10. select BID from thumb_ex_10_by_00;

```

在这里其实没有必要使用视图，但是为了后面题目 10 的方便，在这里先建立了视图。主要就是将 thumb 表和 customer 表联结，条件为 2000 年后出生的用户，点赞数超过 10。

结果如图 3-25，点赞表数据太多，在此不便展示。

	BID
1	1
2	2
3	8

图 3-25 查询 task9

10. 查找被 2000 年后出生的用户点赞数超过 10 的每篇博文的进入头条的次数;

```
1. go
2. CREATE VIEW thumb_ex_10_by_00 AS
3. select thumb.BID
4. from thumb,CUSTOMER
5. where CUSTOMER.BYAEER >= 2000 and CUSTOMER.UID = THUMB.UID
6. group by thumb.BID
7. having count(thumb.UID) > 10;
8. go
9.
10. select toptoday.BID ,count(*) as topnums
11. from toptoday
12. where toptoday.BID in (select thumb_ex_10_by_00.BID from thumb_ex_10_by_00)
13. group by toptoday.BID;
```

如果在 task9 中创建了视图，那么在此就不用重复创建视图。使用 count 计算次数，条件为头条里的 bid 在视图中所有 bid 的集合内。

结果如图 3-26。

	BID	topnums
1	1	1
2	2	1

图 3-26 查询 task10

11. 查找订阅了文学、艺术、哲学、音乐中至少一种分类的用户 ID，要求不能使用嵌套查询，且 where 子句中最多只能包含两个条件;

```
1. select CUSTOMER.UID
2. from CUSTOMER,sub
3. where CUSTOMER.UID = sub.UID and sub.LID in (8,9,10,11)
4. group by CUSTOMER.UID;
```

题目要求不能使用嵌套查询，并且 where 子句最多只能包含两个条件。因此第一个条件 customer.uid = sub.uid 用于筛选，后面一个条件用于判断订阅的标签是否是文学、艺术、哲学、音乐中的一种。

结果如图 3-27。

	UID
1	1
2	2
3	7
4	9
5	11

图 3-27 查询 task11

12. 查找标题中包含了“最多地铁站”和“华中科技大学”两个词的博文基本信

息;

```
1. select * from MBLOG
2. where MBLOG.TITLE like '%最多地铁站%' and MBLOG.TITLE like '%华中科技大学%';
```

使用通配符%，直接查找标题中是否包含了“最多地铁站”和“华中科技大学”两个词。

结果如图 3-28。

13	14	华中科技大学最多...	1	2019	4	20	华中科技大学在2号线...
14	15	华中科技大学开学	1	2019	4	20	华中科技大学在2号线...

	BID	TITLE	UID	PYEAR	PMONTH	PDAY	CONT
1	14	华中科技大学最多地铁站	1	2019	4	20	华中科技大学在2号线上拥有最多地铁站

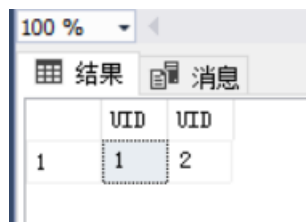
图 3-28 查询 task12

13. 查找所有相互关注的用户对的两个 ID 号，要求不能使用嵌套查询；

```
1. select FOLLOW.UID, FOLLOW1.UID
2. from FOLLOW,FOLLOW1 as FOLLOW1
3. where FOLLOW.UID = FOLLOW1.UIDFLED and FOLLOW.UIDFLED = FOLLOW1.UID and FOLLOW.UID < FOLLOW1.UID;
```

使用两个表自联结。条件为 FOLLOW 的 uid 等于 FOLLOW1 的 uidfled，FOLLOW 的 uidfled 等于 FOLLOW1 的 uid。这样就可以判断两个用户是不是互相关注了。

结果如图 3-29。



	UID	UID
1	1	2

图 3-29 查询 task13

14. 查找朋友圈包含了 5 号用户朋友圈的用户 ID；

```
1. select uid
2. from FRIENDS
3. where FRIENDS.FUID in (select fuid from FRIENDS where FRIENDS.UID = 5) and FRIENDS.uid != 5
4. group by FRIENDS.uid
5. having count(FRIENDS.UID) = (select count(FRIENDS.UID) from FRIENDS where FRIENDS.UID = 5);
```

首先查询 5 号用户的所有好友，并且查询条件是 friends 表中的 FUID 在 5 号用户的好友集里面。并且后面的 having 子句保证了包含 5 号用户的所有好友，因为只需要计算选择出来的好友数是否与 5 号用户的好友数相同即可。

结果如图 3-30。

结果			消息	
	UID	FUID		
1	1	2		
2	1	3		
3	1	4		
4	1	6		
5	1	7		
6	2	1		
7	2	3		
8	3	1		
9	3	2		
10	3	5		
11	4	1		
12	5	3		
13	5	6		
14	5	7		
15	6	1		
16	6	5		
17	7	1		
18	7	5		

	uid
1	1

图 3-30 查询 task14

15. 查找 2019 年 4 月 20 日每一篇头条博文的 ID 号、标题以及该博文的每一个分类 ID，要求即使该博文没有任何分类 ID 也要输出其 ID 号、标题；

```

1. select MBLOG.BID, MBLOG.TITLE,
2.     case when MBLOG.BID in (select B_L.BID from B_L) then (select B_L.LID fr
   om B_L where MBLOG.BID = B_L.BID)
3.     else NULL
4.     end BlogLid
5. from MBLOG, B_L
6. where MBLOG.PYEAR = 2019 AND MBLOG.PMONTH = 4 AND MBLOG.PDAY = 20
7. group by MBLOG.BID, MBLOG.TITLE;

```

首先选择 2019 年 4 月 20 日的每一篇博文的 ID 号，标题。然后如果有分类，则输出分类，如果没有，则为 NULL。这里使用 case 可以实现。

结果如图 3-31。

结果		消息	
	BID	LID	
1	1	2	
2	1	4	
3	2	4	
4	3	1	
5	4	4	
6	5	4	
7	6	6	
8	7	1	
9	7	3	

	BID	TITLE	BlogLid
1	3	风筝	1
2	14	华中科技大学最多地铁站	NULL
3	15	华中科技大学开学	NULL

图 3-31 查询 task15

16. 查找至少有 3 名共同好友的所有用户对的两个 ID 号。

```

1. select FRIENDS.uid, FRIENDS1.uid --, count(FRIENDS.FUID) as num
2. from FRIENDS, FRIENDS as FRIENDS1
3. where FRIENDS.UID != FRIENDS1.UID and FRIENDS.FUID = FRIENDS1.FUID and FRIENDS.UID < FRIENDS1.UID
4. group by FRIENDS.UID, FRIENDS1.uid
5. having count(FRIENDS.FUID) >=3

```

使用两个 friends 表进行自联结。条件为 `FRIENDS.UID != FRIENDS1.UID` and `FRIENDS.FUID = FRIENDS1.FUID` 这样可以保证找到每个 uid 的共同好友，后面的 `FRIENDS.UID < FRIENDS1.UID` 是为了保证好友对只出现一次（比如 1,2 和 2,1）。后面只要 count 后大于等于三即可。

结果如图 3-32。

结果		消息	
	UID	FUID	
1	1	2	
2	1	3	
3	1	4	
4	1	6	
5	1	7	
6	2	1	
7	2	3	
8	3	1	
9	3	2	
10	3	5	
11	4	1	
12	5	3	

	uid	uid
1	1	5

图 3-32 查询 task16

17. 创建视图：查阅 DBMS 内部函数，创建一个显示当日热度排名前十的微博信息的视图，其中的属性包括：博文 ID、博文标题、发表者 ID、发表者姓名、被点赞数。

```

1. go
2. CREATE VIEW top10blogtoday AS
3. select MBLOG.BID, MBLOG.TITLE, MBLOG.UID, CUSTOMER.USERNAME,
4.      case when (select COUNT(thumb.uid) from THUMB where toptoday.BID = T
HUMB.BID) >=0 then (select COUNT(thumb.uid) from THUMB where toptoday.BID =
THUMB.BID)
5.      else 0
6.      end thumbs
7. from toptoday,MBLOG,CUSTOMER,THUMB
8. where toptoday.TYEAR=year(GETDATE()) and toptoday.TMONTH=month(GETDATE()) an
d toptoday.TDAY=day(GETDATE()) and toptoday.BID=MBLOG.BID and MBLOG.UID = CU
STOMER.UID and toptoday.TNO<=10
9. group by MBLOG.BID, MBLOG.TITLE, MBLOG.UID, CUSTOMER.USERNAME, toptoday.BID,
toptoday.TNO
10. go
11.
12. select * from top10blogtoday;

```

创建视图，包含的信息有博文 ID、博文标题、发表者 ID、发表者姓名、被点赞数。前面四个只需要将 toptoday,mblog,customer,thumb 联结即可。当前日期可以用 GETDATE() 内置函数获得，通过 year ()，month ()，day () 即可获得对应的年月日。点赞数的实现可以参照 task15，新建一个列，如果有点赞则显示点赞数，如果没有，则显示 0。也是用 xase 实现的。

结果如图 3-33。

结果					
	TYEAR	TMONTH	TDAY	BID	TNO
4	2019	4	27	1	1
5	2019	4	27	2	2
6	2019	4	27	3	3
7	2019	4	27	4	4
8	2019	4	27	5	5
9	2019	4	27	6	6
10	2019	4	27	7	7
11	2019	4	27	8	8
12	2019	4	27	10	9
13	2019	4	27	11	10
14	2019	4	27	12	11

	BID	TITLE	UID	USERNAME	thumbs
1	1	校长带全校看复联4	1	赵倩	13
2	2	生活大爆炸结局	3	安城	13
3	3	风筝	5	赵齐	1
4	4	复联4彩蛋	2	黄超	0
5	5	复联4剧透	2	黄超	0
6	6	谢文骏110米跨...	8	陈冕	3
7	7	林俊杰圣所	2	黄超	1
8	8	邓超盘腿	1	赵倩	23
9	10	李敏镐退伍	1	赵倩	0
10	11	李敏镐退伍	1	赵倩	0

图 3-33 查询 task17

3.2.4 了解系统的查询性能分析功能

分析前面查询任务的 14-17 条 SQL 语句。

图 3-34 图 3-35 是它们的预估执行计划。

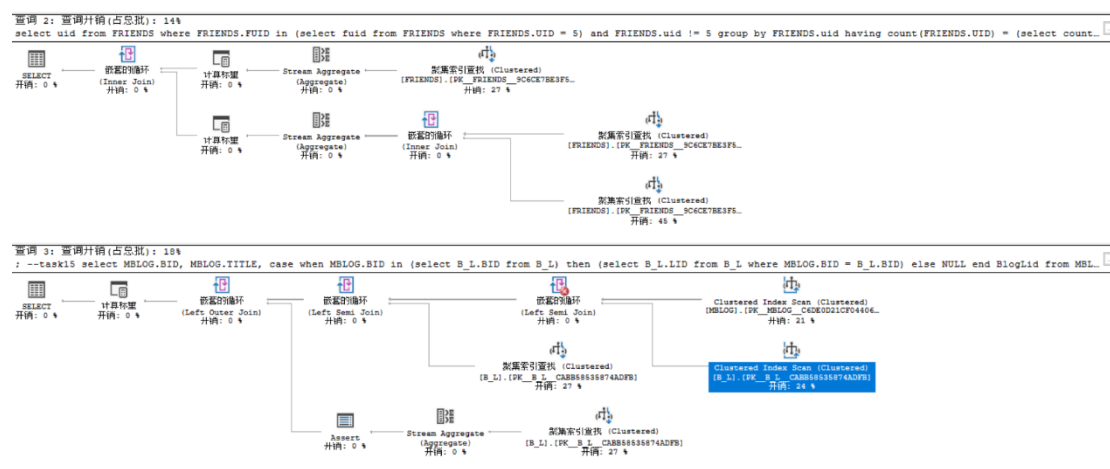


图 3-34 预估执行计划

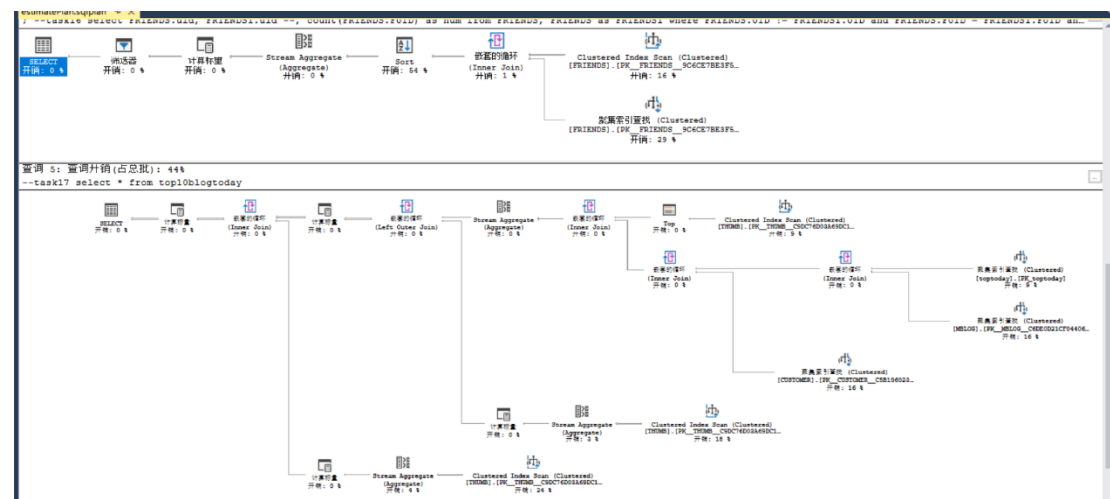


图 3-35 预估执行计划

由上图可知，查询语句 14 与 15 中，聚集索引查找时最消耗资源的部分，几乎所有的开销都在这个上面。而且由上图看出，一般来说，嵌套的循环都会有两部分开销。

而在查询语句 16 中，占开销最大的居然是 sort，这是令人没有想到的，因为在语句 16 中，根本没有用到 order by 这样的语句，但是最后却是调用了排序。猜想可能是要求大于等于三，SQL Server 直接用排序来实现了。而查询语句 17 就跟前面的 14,15 相同，开销都花在聚集索引查找上了。

图 3-36 图 3-37 是它们的实际执行计划。

可以看到，实际执行计划和它们的预估执行计划完全相同。

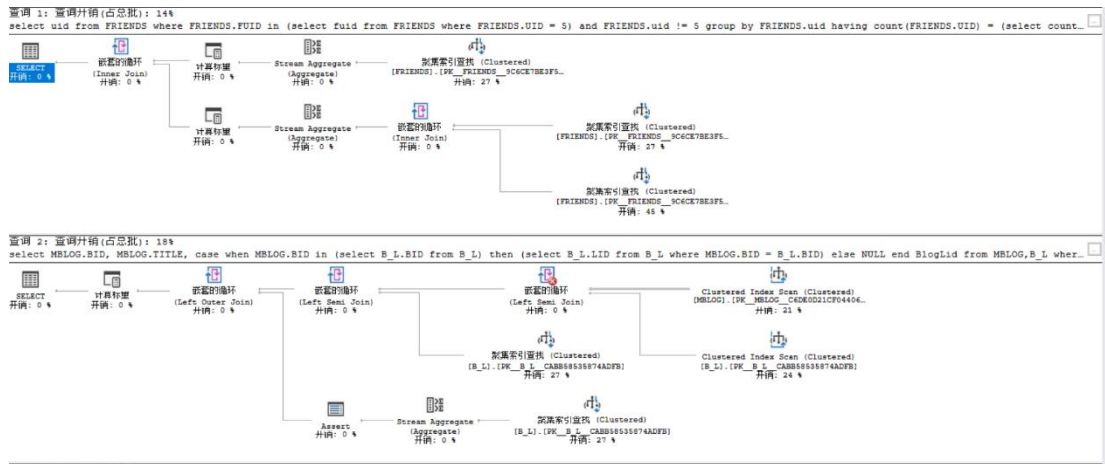


图 3-36 实际执行计划

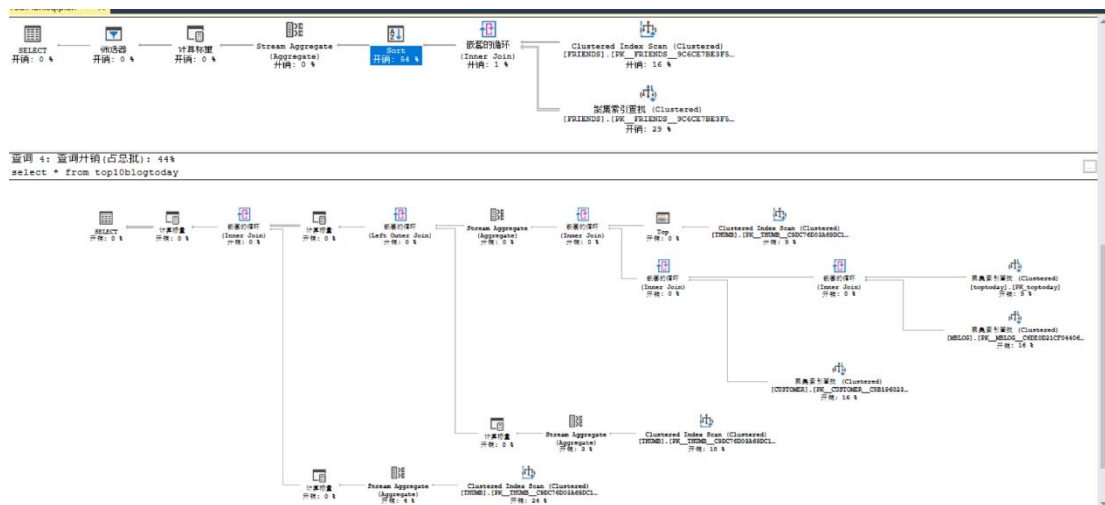


图 3-37 实际执行计划

相同可能是因为数据量比较小，这样查询比较节省开销，也有可能是这就是最佳方案了。

同时也可以看出，最后一条查询语句的开销最大，占了4条语句开销中的44%。

3.2.5 DBMS 函数及存储过程和事务

1. 编写一个依据用户 ID 号计算其发表的博文进入头条的累计天数的 DBMS 自定义函数，并利用其查询 2000 年后出生的上述头条累计天数达到 100 天的所有用户 ID。

```

1. CREATE FUNCTION [dbo].[toptimes100]
2. (
3.     -- Add the parameters for the function here
4.     @topdays int = 100
5. )
6. RETURNS TABLE
7. AS
8. RETURN
9. (
10.    -- Add the SELECT statement with parameter references here
11.    SELECT MBLOG.UID,toptoday.BID
12.    from MBLOG,toptoday,CUSTOMER

```

```

13.     where MBLOG.BID = toptoday.BID and MBLOG.UID = CUSTOMER.UID and CUSTOMER
        .BYAER >= 2000
14.     group by MBLOG.UID,toptoday.BID
15.     having count(toptoday.BID) >= @topdays
16. )

```

topdays 是参数，默认值为 100.由于实际数据库没有那么多数据，因此实际执行时可以根据具体情况改变参数。

执行 select * from toptimes100(3)结果如下。

	TYEAR	TMONTH	TDAY	BID	TNO
1	2019	4	26	1	2
2	2019	4	26	2	3
3	2019	4	26	12	1
4	2019	4	27	1	1
5	2019	4	27	2	2
6	2019	4	27	3	3
7	2019	4	27	4	4
8	2019	4	27	5	5
9	2019	4	27	6	6
10	2019	4	27	7	7
11	2019	4	27	8	8
12	2019	4	27	10	9
13	2019	4	27	11	10
14	2019	4	27	12	11
15	2019	4	29	1	1
16	2019	4	29	2	2
17	2019	4	29	3	3
18	2019	4	29	4	4
19	2019	4	29	5	5
20	2019	4	29	6	6

	UID	BID
1	1	1

图 3-38 函数测试

在这里为了特地适应 2000 年这个情况，将 1 号的出生年份改为了 2000。可以看到上面虽然有 2 条博文进了 3 次，但是最后只有用户 1 的博文被挑选出来。

2. 建立关系“点赞排行榜【博文 ID，当天点赞人数】”，里面存储系统当天点赞数前十名的博文 ID 及其点赞人数，尝试编写一个 DBMS 的存储过程，通过该存储过程更新该表。

```
1. go
2. create procedure updatethumbtoday
3. as
4. begin
5.     truncate table thumbtoday;
6.     insert into thumbtoday (bid, thumbnums)
7.     select top 10 MBLOG.BID, count(THUMB.UID)
8.     from MBLOG, THUMB
9.     where MBLOG.PYEAR = year(GETDATE()) and MBLOG.PMONTH = month(GETDATE())
      and MBLOG.PDAY = day(GETDATE()) and MBLOG.BID = THUMB.BID
10.    group by MBLOG.BID
11.    order by count(THUMB.UID);
12. end
```

获取当前时间在前面查询 SQL 语句已经写过了，在此不再赘述。主要需要注意的就是前 10，这里我先排了序，然后取 top10 就行了。

执行 exec updatethumbtoday 之后，结果如图 3-39。

结果			消息
	BID	thumbnums	
1	1031	3	
2	1032	3	
3	1033	3	
4	1034	3	
5	1035	3	
6	1036	3	
7	1037	3	
8	1038	3	
9	1039	3	
10	1030	1	

图 3-39 存储过程

可以看到，执行非常顺利。

3. 尝试在 DBMS 的交互式界面中验证事务机制的执行效果。

```
1. begin tran
2.     insert into THUMB (UID, BID) values (1, 7);
3.     insert into THUMB (UID, BID) values (1, 1);
4. if @@ERROR > 0)
5.     begin rollback tran end
6. else
7.     begin commit tran end
```

当给自己点赞时，会出错，并且回滚事务，即回到事务开始前的状态。由于事务的原子性，这两条语句都被回滚了，因此在点赞表里不会出现 (1,7)，尽管

(1,7) 可以插入点赞表。

测试结果如图 3-40 图 3-41。

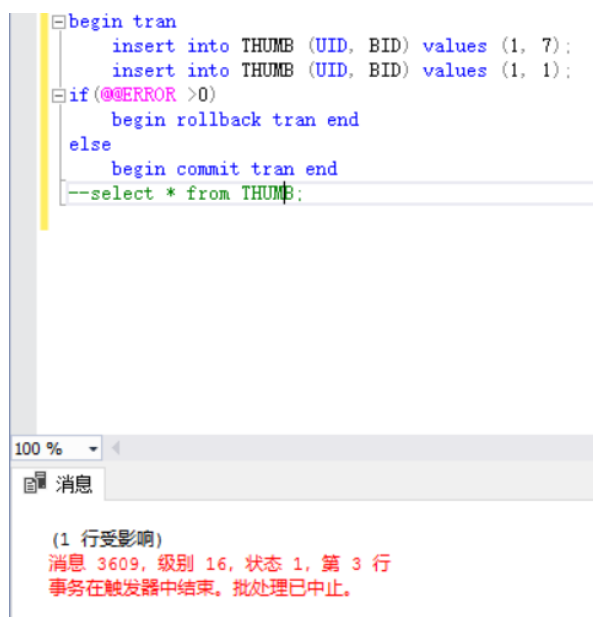


图 3-40 事务机制

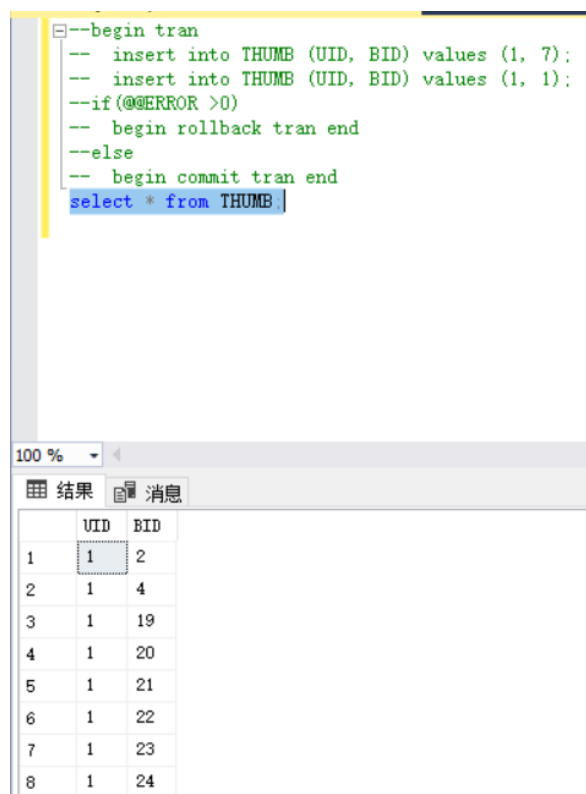


图 3-41 事务机制

3.3 任务总结

在这次实验中遇到的问题主要就是查询语句掌握还不够熟练，遇到复杂的查询语句时，不能很好地将其拆分为几个步骤，一个一个地去完成它，最后拼凑起来完成最终的语句。

其次就是，在做查询的时候，要想到能够如何使我的查询而更加便捷，比如在上面的查询语句中，有一个没有要求要用到视图，但是为了后面的方便，我还是创建了视图，即使看上去用不用视图没有什么关系，但是后面那个查询语句，用了之前就创建好的视图的话，会更加快速地完成。因此，我明白了创建一个通用的视图是很重要的，视图并不能有过多的限制，最好是能够运用很多次，比如可以将商品的价格排序作为一个视图，后面不管想找什么商品的价格话，只需要使用这个视图就可以完成了。

这次的很多实验都是自己上网查阅资料，结合课堂所学知识完成的。比如说触发器的创建，一开始不知道 SQL SERVER 触发器的语法规则，就不知道从哪里下手，后来知道了就很快完成了。触发器一般是用于出错检查，完整性控制，以后设计数据库的时候，要考虑完整性，一旦某个操作是错误的，不能让它继续执行，要及时终止掉它。

查询开销主要就是了解它，要知道优化该在哪里进行。

最后就是函数，存储过程和事务。这些都是 SQL SERVER 很强大的部分，函数，存储过程能够有效地提高效率，将要做的操作封装起来，提供给用户的就是一个调用规则，而不是让用户自行访问数据库。事务机制的出现也很有效地防止了错误操作，比如转账失败时，就要回滚事务，而不是让它继续执行，否则就会出现比较严重的错误。

4 综合实践任务

4.1 系统设计目标

在上世纪，航空公司机票预订主要是用手动系统，这个系统由一群人组成，而他们手里的卡片就代表一次预订或者飞机上的某个位置。然而，在上世纪 50 年代后期，人们要求能够有一个预订系统能够实时显示所有航班信息，并且实现机票预订的自动化。由于这一需求，早期人们推出了电子预订系统 Magnetronic Reservisor。

但是随着飞机旅客人数的增加，航空公司多样化，早期简单的系统已经不足以满足各类人群的需求以及高并发量，还考虑到界面操作的人性化等因素，设计并实现一个界面友好，操作便捷的机票预订系统尤为重要。

系统的总体目标是方便旅客出行，信息显示完善，有一定的容错能力。旅客可以根据出发地和目的地随时查询当前所有航班信息，选择适合自己行程的航班。

4.2 需求分析

采用 B/S 或 C/S 模式实现一个机票预订系统。系统功能的基本要求有：

- 每个航班信息的输入。
- 每个航班的座位信息的输入；
- 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；
- 旅客在飞机起飞前一天凭取票通知交款取票；
- 旅客能够退订机票；
- 能够查询每个航班的预定情况、计算航班的满座率。
- 实现月度、季度、年度航空公司财务报表。

数据库要求有：在数据库中至少应该包含下列数据表：

- 航班信息表；
- 航班座位情况表；
- 旅客订票信息表；
- 取票通知表；
- 帐单。

在性能上，机票预订系统需要能够同时处理大量请求，并且不出故障。在负荷过大时能够将请求缓存在队列中。必要时，如果实在不能处理请求，应该向用户反馈信息，通知系统繁忙。除此之外，系统还应该快速实现订票功能，节省用户的时间。

在数据完整性方面，需要实现航空公司与客户之间收支平衡，在航班预订满

了之后，系统应该取消该航班的显示。进一步为了更好的出行，系统应该取消在当前时间几十分钟的航班的预订权限，这样是为了更加方便的管理以及让客户有充足的时间来准备这次行程。

同时也要考虑系统的安全性，系统需要保证每个客户只对自己的信息有访问权限，他不能访问全部的数据库信息，只能调用系统授予的服务。这样做很有必要，否则一些不法分子可能会通过这一漏洞窃取私密信息以及大量资金。

顶层数据流图如图 4-42。

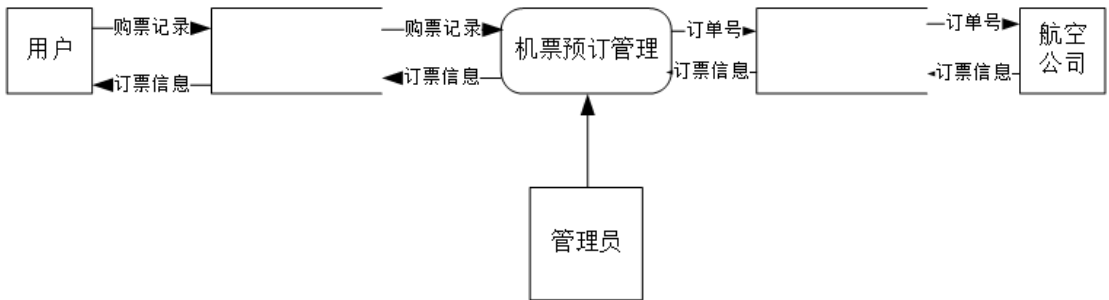


图 4-42 顶层数据流图

数据字典如下：

数据流名称	购票记录	标识符	
描述	用户购票信息		
数据流来源	用户	数据流去向	数据 库
数据结构			
名称	类型	描述	
PID	INT	乘客 ID	
PName	Varchar（9）	乘客姓名	
FIID	INT	订票 ID	
BookingDate	DateTime	订票时间	
DepartureDate	DateTime	出发时间	
Flight	INT	航班号，可获取航班 信息	
Charges	INT	费用	

订票信息的数据字典和购票信息的数据字典相同，在此就不再赘述了。

4.3 总体设计

系统采用 B/S 模式，总体上包含用户端，管理端，数据库。用户端和管理端有不同的访问数据库的权限。这样做能够限制客户端的一些功能，使这个系统变得更加安全。如图 4-43。

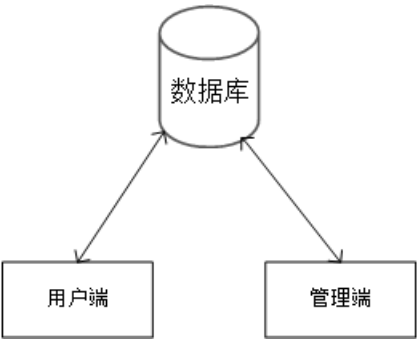


图 4-43 B/S 架构图

功能模块如下：

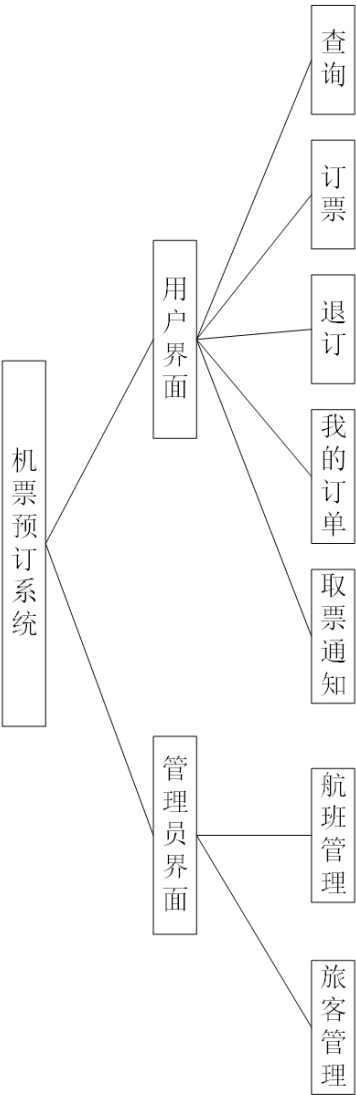


图 4-44 功能模块图

功能主要分为用户界面和管理员界面。用户界面主要包含查询，订票，退订，

个人中心等模块。管理员界面主要包括航班管理和旅客管理等模块。有了这些功能，应该能基本上满足系统的要求了。

在这里列出主要的业务处理流程图，详细的设计在之后会进一步讲解。

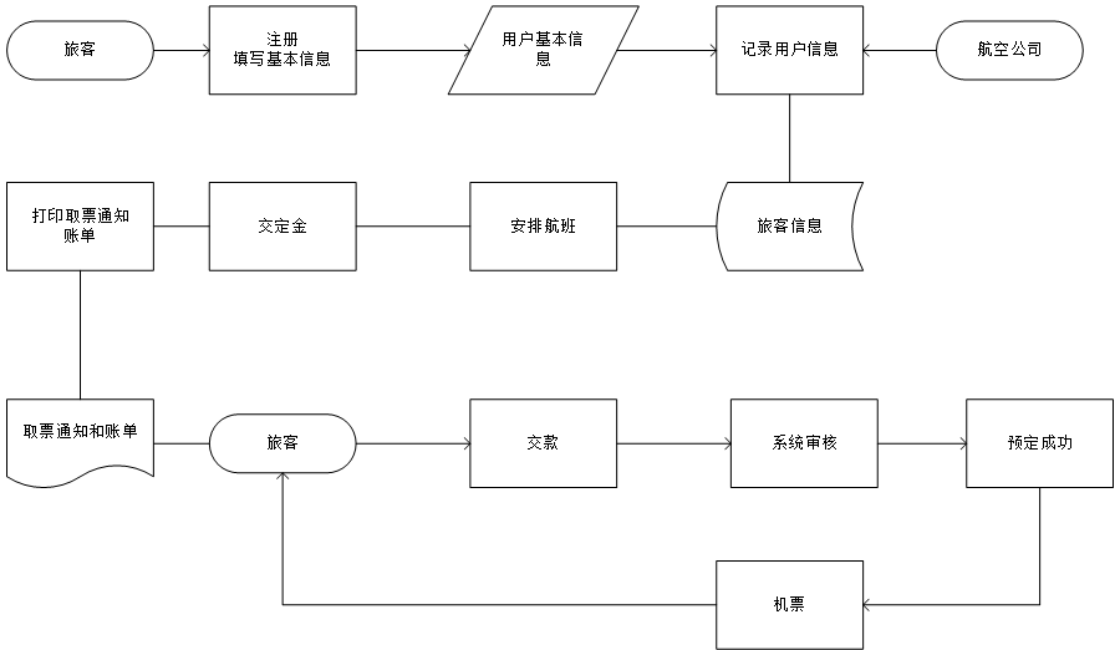


图 4-45 业务处理流程图

4.4 数据库设计

根据数据库的设计要求，设计了 ER 图，表会在下面详细说明。如图 4-46。

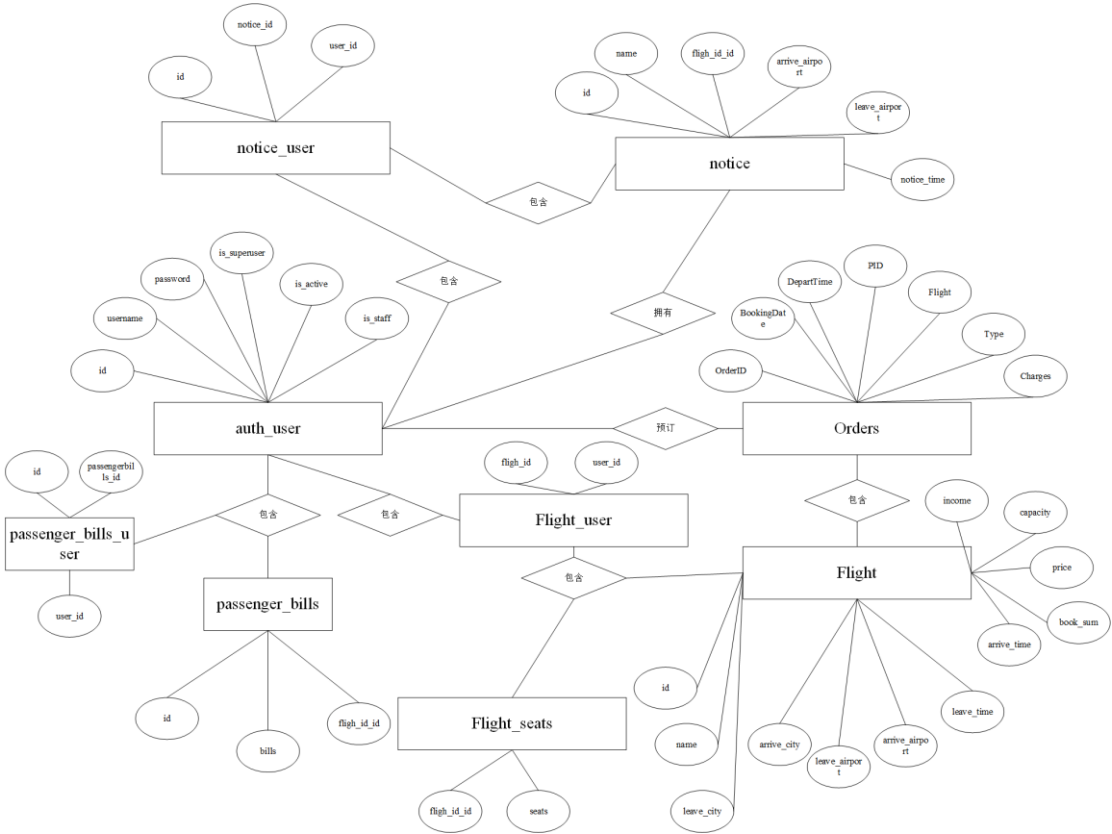


图 4-46 E-R 图

为了更加清晰的表示数据库中表的情况，表 4-1 显示了数据库中含有的表。

表 4-1 数据库中的表

表名	说明
auth_user	旅客信息表
flight	航班信息表
flight_user	航班用户联系表
flightseats	座位信息表
passengerbills	账单表
passengerbills_user	账单用户联系表
ticketnotice	取票通知表
tichetnotice_user	取票通知用户联系表

表 4-2 旅客信息表的字段类型说明以及约束。

表 4-2 旅客信息表

字段	类型	说明	约束
id	INT	乘客 ID	Primary Key
username	Varchar(9)	乘客姓名	NOT NULL
Password	Varchar(9)	乘客密码	NOT NULL
is_superuser	INT	是否是超级用户	NOT NULL
is_staff	INT	是否职工	NOT NULL
is_active	INT	是否被禁	NOT NULL

表 4-3 航班信息表的字段类型说明以及约束。

表 4-3 航班信息表

字段	类型	说明	约束
id	INT	航班编号	Primary Key
name	Varchar(9)	航班名字	NOT NULL
leave_city	Varchar(9)	航班起飞城市	NOT NULL
arrive_city	Varchar(9)	航班到达城市	NOT NULL
leave_airport	Varchar(9)	航班起飞机场	NOT NULL
arrive_airport	Varchar(9)	航班到达机场	NOT NULL
leave_time	DATETIME	航班起飞时间	NOT NULL
arrive_time	DATETIME	航班到达时间	NOT NULL
capacity	INT	航班座位数	NOT NULL

price	FLOAT	航班价格	NOT NULL
book_sum	INT	航班订购数	NOT NULL
income	FLOAT	航班收入	NOT NULL

表 4-4 航班用户表的字段类型说明以及约束。

表 4-4 航班用户表

字段	类型	说明	约束
id	INT	表 ID	Primary Key
flight_id	INT	航班 id	Foreign Key
user_id	INT	用户 id	Foreign Key

表 4-5 航班座位表的字段类型说明以及约束。

表 4-5 航班座位表

字段	类型	说明	约束
id	INT	表 ID	Primary Key
seats	INT	座位数	Foreign Key
flight_id_id	INT	航班 ID	Foreign Key

表 4-6 账单表的字段类型说明以及约束。

表 4-6 账单表

字段	类型	说明	约束
id	INT	表 ID	Primary Key
bills	FLOAT	账单费用	Foreign Key
flight_id_id	INT	航班 ID	Foreign Key

表 4-7 账单用户表的字段类型说明以及约束。

表 4-7 账单用户表

字段	类型	说明	约束
id	INT	表 ID	Primary Key
passingbills_id	INT	账单 ID	Foreign Key
user_id	INT	用户 ID	Foreign Key

表 4-8 取票通知表的字段类型说明以及约束。

表 4-8 取票通知表

字段	类型	说明	约束
id	INT	表 id	Primary Key
notice_time	DATETIME	通知时间	NOT NULL
name	Varchar(9)	航班名称	NOT NULL
leave_airport	Varchar(9)	起飞机场	NOT NULL
arrive_airport	Varchar(9)	离开机场	NOT NULL
flight_id_id	INT	航班 ID	Foreign Kry

表 4-9 的字段类型说明以及约束。

表 4-9 取票通知用户表

字段	类型	说明	约束
id	INT	表 ID	Primary Key
ticketnotice_id	INT	取票通知 ID	Foreign Key
user_id	INT	用户 ID	Foreign Key

4.5 详细设计与实现

1. 设置

采用 Django 开发。首先需要在 `url.py` 中加入各个页面的地址，然后在设置中（`settings.py`）添加自己的 app，在 `database` 中设置默认的数据库。

2. 网页设计

本次课程设计网页借鉴了一些模板，航班显示仿照携程的布局。

在本次设计中网页不是重点范围，因此在此不作重点描述。

3. 表创建

数据库需要在 `models.py` 里面定义类。成员就相当于字段，具体的表在设计中已经给出，在此不再赘述。当设定好了类时，只需要在命令行中输入 `python manage.py makemigrations` 和 `python manage.py migrate` 即可完成表的迁移。（创建更新）。

4. 后台开发

后台开发都在 `views.py` 中实现。

1) 显示用户订单

通过 `request` 可以得到用户是不是管理员，如果用户是管理员并且与我指定的 `id` 相同，那么就可以显示公司财务报表了。

如果不是管理员，那么就要显示用户的订单，使用 `filter` 可以过滤出用户的订单，只需要把 `request.user` 传进去就可以了。然后用 `list` 保存传入 `html`，在网页上显示就可以了。

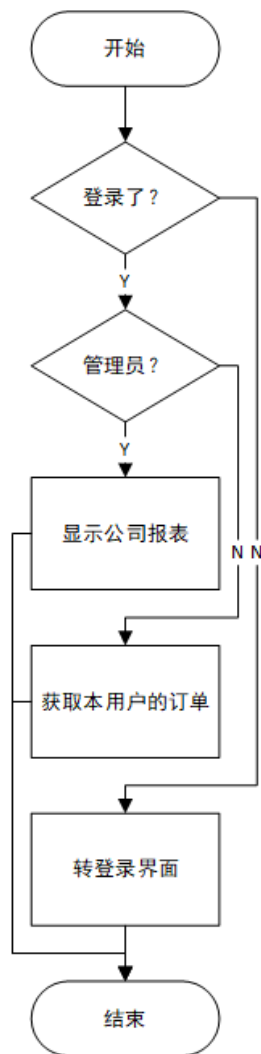


图 4-47 显示用户订单流程图

2) 显示用户取票通知

与显示用户订单相同，只是需要从取票通知表中读取数据，流程和显示订单流程图相同，在此就不做赘述。

3) 显示主页

一开始进入需要一个欢迎界面用于搜索，这个界面就写的美观了一点，同时获取这个页面上的输入信息，跳转到结果页面。

4) 登录

登录模块需要做的就是验证用户名与密码是否能匹配到数据库中的数据，如果登录成功，就加载订票的页面（这里分为管理员和不是管理员的情况，如果是管理员，就需要现实公司的财务报表）。如果登录失败就还是在这个界面。

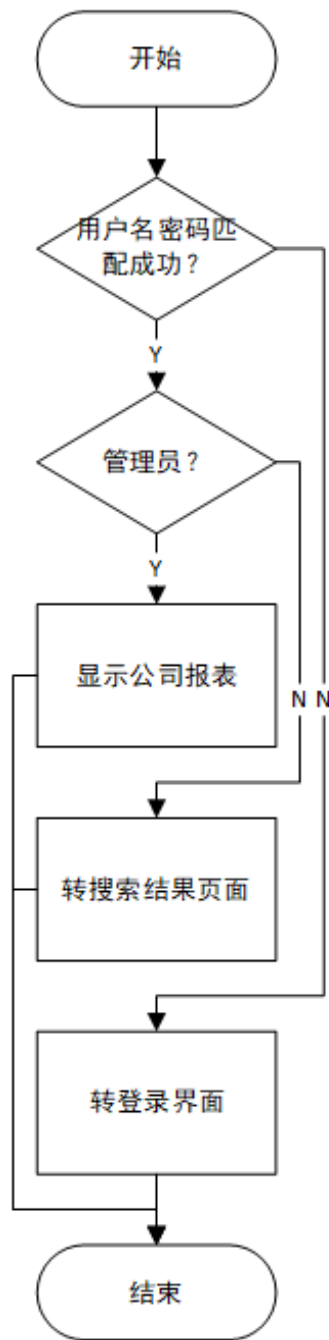


图 4-48 登录流程图

5) 注册

注册在注册界面上进行，获取用户的输入，存入数据库。Django 有一套自己的 API，与数据库进行交互。只需修改成员属性，最后 save 就行了。如果已经注册成功，跳转到搜索界面。

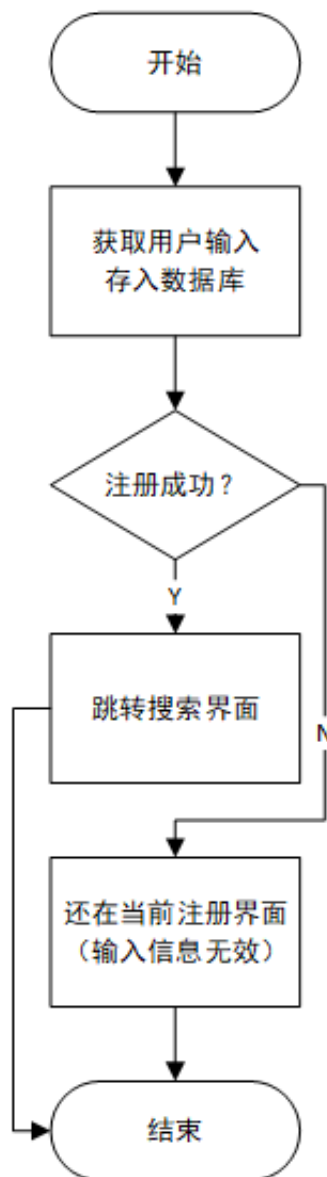


图 4-49 注册流程图

6) 退出登录

退出登录实现起来简单，就是跳转到登录界面就可以了。

7) 搜索结果

获取搜索结果算是一个比较复杂的功能了。首先需要获得用户在网页上输入的数据，如果输入的数据有效则搜索对应的航班；否则还是停留在当前界面。同理搜索航班也要用到 **filter**，只需要将用户输入的信息作为条件即可。同时为了增强用户体验，需要实现能够按照时间，价格排序的功能。搜索完之后，将数据传给 **html** 显示就好。

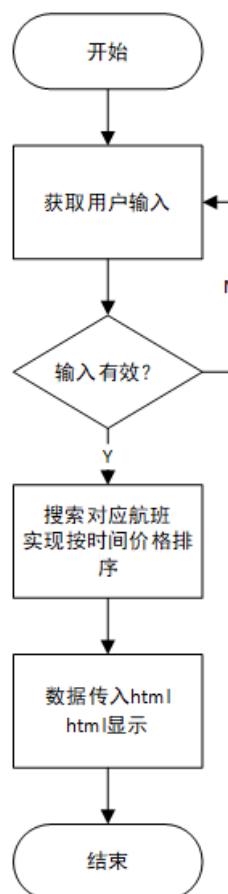


图 4-50 搜索结果流程图

8) 订票

订票是一个核心模块了，所以它的实现也较复杂。

首先检查用户是否登录了，如果用户没有登录，需要转到登录界面。否则就要进行与数据库的交互了。首先获得预订的航班 id, 在航班座位表, 用户账单表, 取票通知表中创建对应的记录（如果有就是获取了），在这里可以使用 `get_or_create` 方法，可以避免报错或者重复插入的问题。然后获取用户已订购的航班的信息，如果用户已经预定过了，需要转到提示界面，提示不能重复购买。如果能够预订，就需要进行数据的插入更新。Django 使用自己的 API，因此只要将一个对象的成员进行改变，然后使用 `save()` 方法，就可以保存到数据库中了。在这里加入用户 id 并不是使用简单的 INT 类型的变量，而是在航班对象中又加入了一个 `user` 对象，直接使用 `user.add` 方法加入 `user`，就会两张表的联系。类似地，需要对航班座位表，账单表，取票通知表进行修改，写入数据库。完成这些操作之后，需要将航班信息传入 `html`，跳转到预定成功的界面。

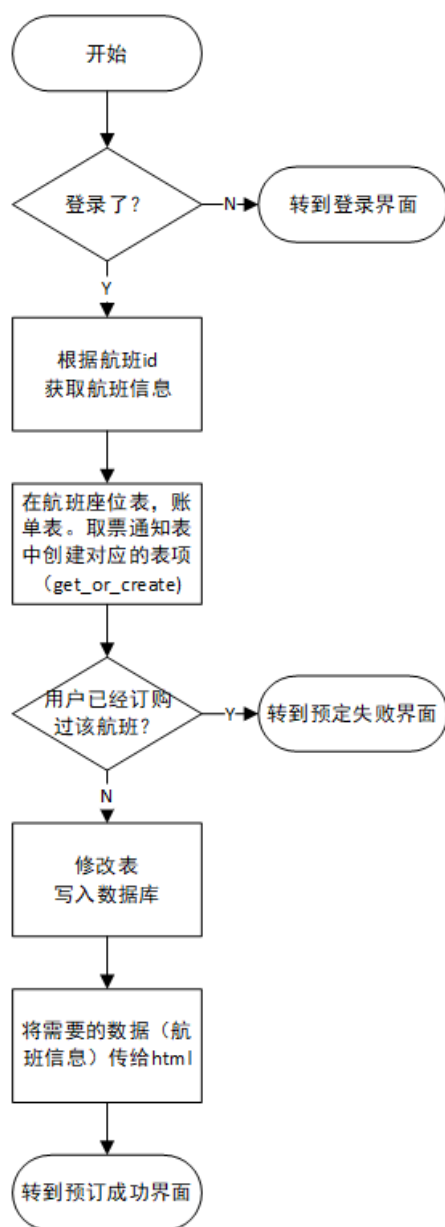


图 4-51 订票流程图

9) 退票

退票相比订票没有那么复杂，也是需要和数据库进行交互，方式和订票相同。只不过逻辑上可能有点差别，比如收入需要减少等等。在此就不做过多的描述了。

10) 财务报表

财务报表的实现也是有点复杂的。首先需要将航班信息都去出来存在一个列表中。要求实现月度，季度，年度的功能。这里的逻辑是通过航班的时间获取月度，年度收入。季度收入就在月度收入表基础上合并一下（三个月为一个季度）。这样就可以实现了。具体是通过航班的起飞时间获取月份，年份。然后将这个信息存入到相应的变量中，与收入组成一个元组存入一个列表中。循环这个列表，根据时间进行分类，将时间相同的元组的收入加起来，就形成了月度和年度报表。季度报表需要在月度报表的基础上实现，在月度列表里循环，如果月份符合要求

（比如第一季度是 1,2,3 月等）就将其收入加起来，与季度信息一起存入一个列表中。

图 4-52 月度财务报表流程图，图 4-53 季度报表流程图。在这里展示两个流程图，年度的没有给出是因为年度的流程与月度相同只不过把月份换成了年份而已。

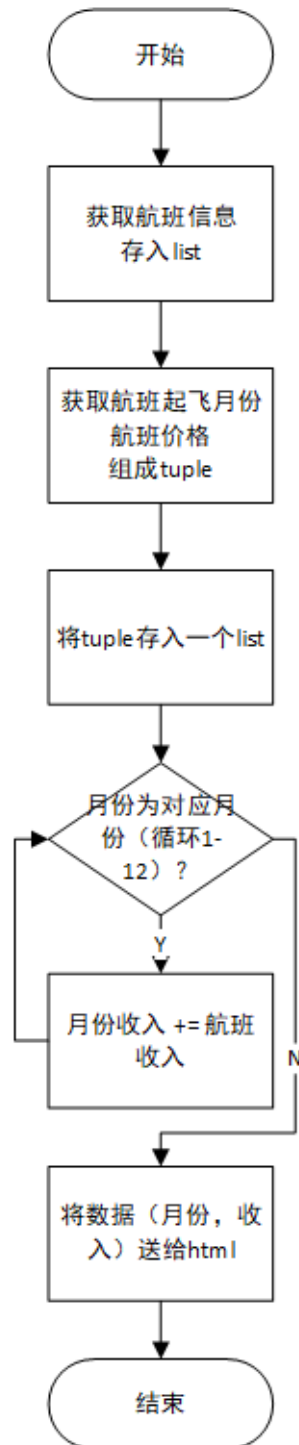


图 4-52 月度财务报表流程图

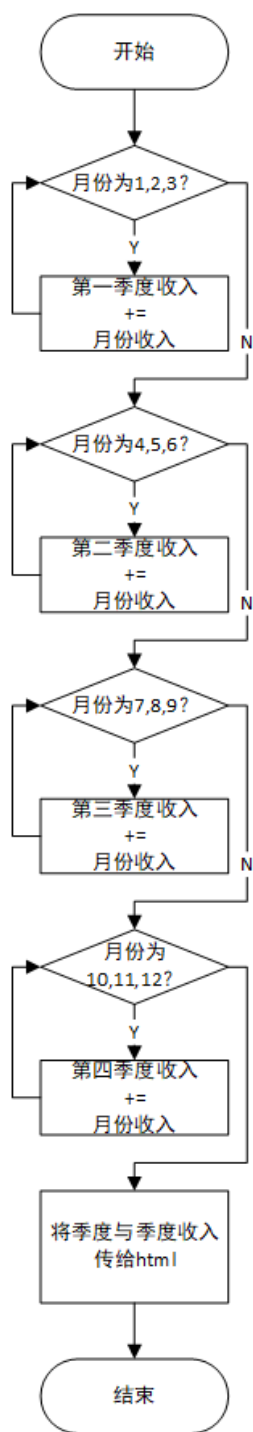


图 4-53 季度报表流程图

5. 后台管理

后台管理 Django 提供了一个强大的后台，我们只需要传入需要显示的数据就可以完成这部分的工作了。

需要使用超级用户的身份进行登录才可进入后台管理。可以进行添加航班信息，删除用户等操作。

4.6 系统测试

首先是主页，因为是一开始的界面，所以就设计的好看了一点。

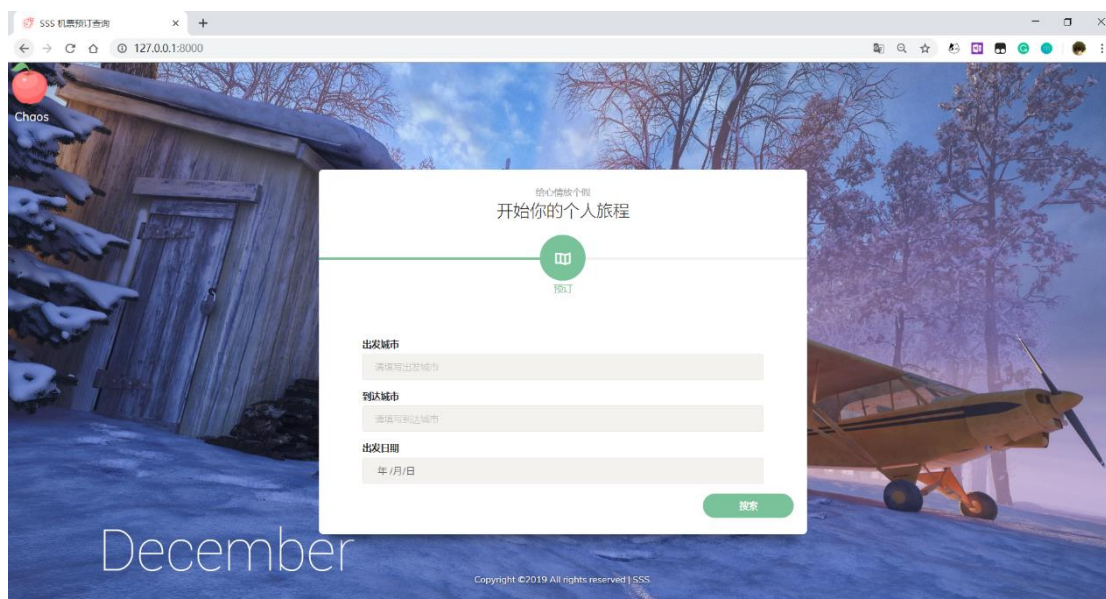


图 4-54 主页

可以输入出发城市，到达城市以及出发日期。其中出发城市与到达城市需要手动输入，这里我没有太多的数据，做成可用鼠标选择的也没有什么意义。下面的出发日期可以手动输入也可以进行选择。

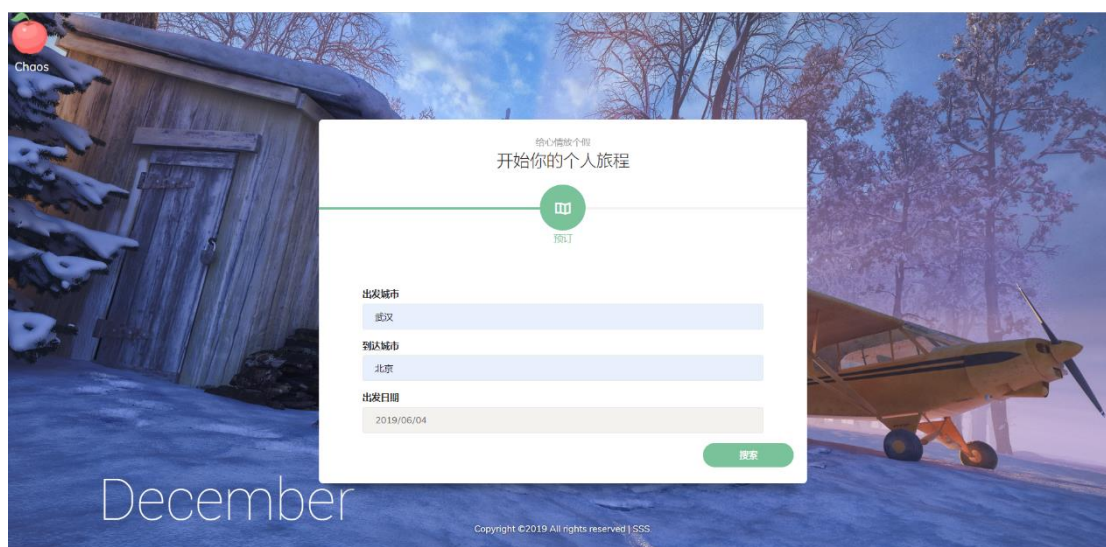


图 4-55 主页输入信息

输入信息，点击搜索后会跳转进入另一个界面。

武汉	北京	2019/06/04	重新搜索
航班信息	起飞时间	到达时间	价格
南方航空CZ3117	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 0/100 ¥ 1610.0 订票
厦门航空MF1291	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 0/100 ¥ 1610.0 订票
东方航空MU2451	08:20 天河国际机场T3	10:20 首都国际机场T2	座位数 0/100 ¥ 1610.0 订票
东方航空MU2455	17:55 天河国际机场T3	19:55 首都国际机场T2	座位数 0/100 ¥ 1610.0 订票
中国国航CA8201	08:10 天河国际机场T3	10:10 首都国际机场T2	座位数 0/100 ¥ 1620.0 订票
中国国航CA8209	19:05 天河国际机场T3	21:05 首都国际机场T2	座位数 0/100 ¥ 1620.0 订票
中国国航CA8215	19:50 天河国际机场T3	23:35 首都国际机场T2	座位数 0/100 ¥ 1620.0 订票

图 4-56 搜索结果图

搜索结果页面仿照携程，减少了它的准点率这个表项。航班都是在携程上寻找对应航班，自己录入。可以发现结果正确。

现在订购南方航空 CZ3117。如下图。

您好

请确认您的姓名: cordelia ,

请确认您的 取票通知及账单 ,

姓名	航班	起飞机场	到达机场	起飞时间	到达时间	价格
cordelia	南方航空CZ3117	武汉天河国际机场T3	北京首都国际机场T2	2019-06-04 07:50:00	2019-06-04 09:55:00	¥ 1610.0

请在 2019-06-04 前一天 % 取票通知 取票。

预订 返回

图 4-57 预订机票

第一行会让旅客确认姓名，然后下方会打印出取票通知以及账单。

最下面一行会提醒客人哪天凭取票通知取票。如果确认信息无误的话，点击预订即可。

此时重新搜索，会发现南方航空 CZ3117 座位数变成了 1/100.

武汉	北京	2019/06/04	重新搜索
航班信息	起飞时间	到达时间	价格
南方航空CZ3117	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 1/100 ¥ 1610.0 订票
厦门航空MF1291	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 0/100 ¥ 1610.0 订票

图 4-58 预订成功

在导航栏点击我的订单，会显示订单信息。

航班信息	起飞机场	到达机场	起飞时间	到达时间	价格
南方航空CZ3117	武汉天河国际机场T3	北京首都国际机场T2	2019-06-04 07:50:00	2019-06-04 09:55:00	¥ 1610.0 退票

返回

图 4-59 我的订单

会显示订单的基本信息，为了区分，表头都设置了颜色。并且有一个退票的按钮。

点击导航栏的取票通知，会显示出所有的取票通知。可以发现取票时间确实是起飞前一天。

姓名	航班信息	起飞机场	到达机场	取票时间
cordelia	南方航空CZ3117	天河国际机场T3	首都国际机场T2	2019-06-03

返回

图 4-60 取票通知

如果退票，取票通知也会一起消失。同时南航 CZ3117 座位数也会变成 0/100.

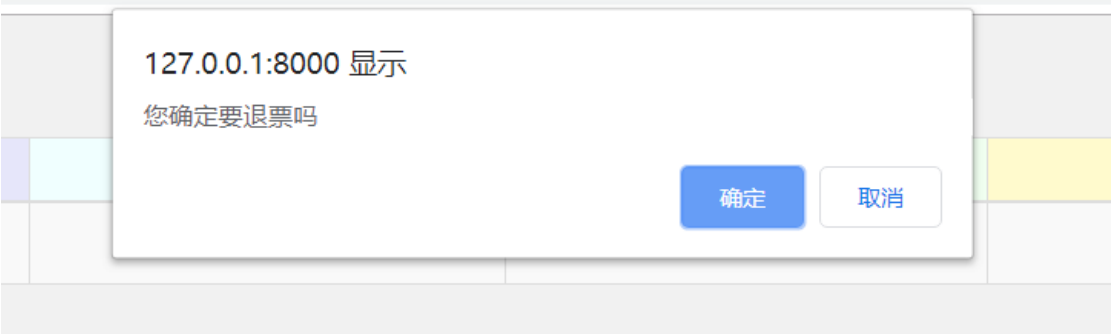


图 4-61 退票

航班信息	起飞机场	到达机场	起飞时间	到达时间	价格
------	------	------	------	------	----

返回

图 4-62 我的账单

姓名	航班信息	起飞机场	到达机场	取票时间
----	------	------	------	------

返回

图 4-63 取票通知

南方航空CZ3117	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 0/100	¥ 1610.0	订票
------------	-------------------	-------------------	--------------	----------	----

图 4-64 搜索结果（座位数改变）

如果想换账号，需要点击导航栏的退出，这时会转到登录界面。

登录账号

用户名:

密码:

登录

还没有账号? [点击此处注册。](#)

图 4-65 登录界面

如果没有账号，则需要注册。

注册账号

Username:

testtest

Email address:

Password:

····

注册

已有账户? [点击此处](#)登录。

图 4-66 注册界面

您好, testtest

图 4-67 成功登录

除此之外，还可以按照起飞时间升序，到达时间降序，价格升序排列。

起飞时间升序，到达时间降序是为了有的人喜欢早点出发，而有的人时间来不及了，就有可能查哪一班航班最晚，符合自己的时间。

航班信息	起飞时间	到达时间	价格
南方航空CZ3117	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 0/100 ¥ 1610.0 订票
厦门航空MF1291	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 0/100 ¥ 1610.0 订票
中国国航CA8201	08:10 天河国际机场T3	10:10 首都国际机场T2	座位数 0/100 ¥ 1620.0 订票
东方航空MU2451	08:20 天河国际机场T3	10:20 首都国际机场T2	座位数 0/100 ¥ 1610.0 订票
东方航空MU2455	17:55 天河国际机场T3	19:55 首都国际机场T2	座位数 0/100 ¥ 1610.0 订票
中国国航CA8209	19:05 天河国际机场T3	21:05 首都国际机场T2	座位数 0/100 ¥ 1620.0 订票
中国国航CA8215	19:50 天河国际机场T3	23:35 首都国际机场T2	座位数 0/100 ¥ 1620.0 订票

图 4-68 起飞时间升序

航班信息	起飞时间	到达时间	价格
中国国航CA8215	19:50 天河国际机场T3	23:35 首都国际机场T2	座位数 8/100 ¥ 1620.0 订票
中国国航CA8209	19:05 天河国际机场T3	21:05 首都国际机场T2	座位数 8/100 ¥ 1620.0 订票
东方航空MU2455	17:55 天河国际机场T3	19:55 首都国际机场T2	座位数 8/100 ¥ 1610.0 订票
东方航空MU2451	08:20 天河国际机场T3	10:20 首都国际机场T2	座位数 8/100 ¥ 1610.0 订票
中国国航CA8201	08:10 天河国际机场T3	10:10 首都国际机场T2	座位数 8/100 ¥ 1620.0 订票
南方航空CZ3117	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 8/100 ¥ 1610.0 订票
厦门航空MF1291	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 8/100 ¥ 1610.0 订票

图 4-69 到达时间降序

航班信息	起飞时间	到达时间	价格
南方航空CZ3117	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 8/100 ¥ 1610.0 订票
厦门航空MF1291	07:50 天河国际机场T3	09:55 首都国际机场T2	座位数 8/100 ¥ 1610.0 订票
东方航空MU2451	08:20 天河国际机场T3	10:20 首都国际机场T2	座位数 8/100 ¥ 1610.0 订票
东方航空MU2455	17:55 天河国际机场T3	19:55 首都国际机场T2	座位数 8/100 ¥ 1610.0 订票
中国国航CA8201	08:10 天河国际机场T3	10:10 首都国际机场T2	座位数 8/100 ¥ 1620.0 订票
中国国航CA8209	19:05 天河国际机场T3	21:05 首都国际机场T2	座位数 8/100 ¥ 1620.0 订票
中国国航CA8215	19:50 天河国际机场T3	23:35 首都国际机场T2	座位数 8/100 ¥ 1620.0 订票

图 4-70 价格升序

如果是指定的用户（管理员），则可以看到财务报表。

月度收入	
月份	收入
06	820.0
季度收入	
季度	收入
01	0
02	820.0
03	0
04	0
年度收入	
年份	收入
2019	820.0

图 4-71 财务报表

Django 自带了一个强大的后台，如图。

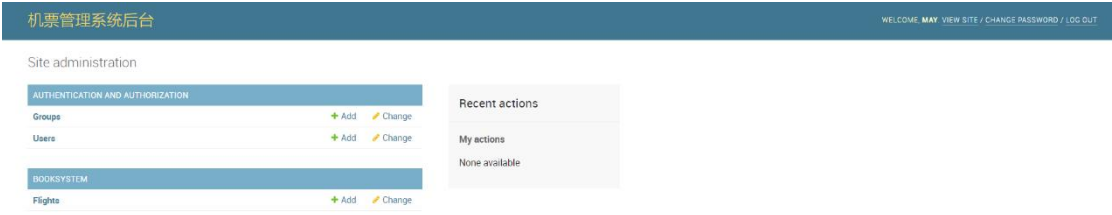


图 4-72 后台管理

如下图，可以看到用户信息和航班信息。

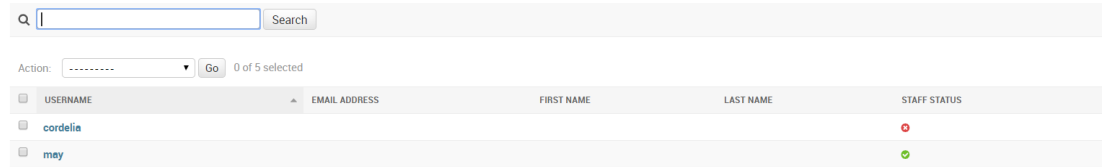


图 4-73 用户信息

<input type="checkbox"/>	国泰港龙航空	武汉	香港	天河国际机场	香港国际机场T1	June 4, 2019, 8:50 p.m.	June 4, 2019, 10:50 p.m.	80	883.0	0	0.0
<input type="checkbox"/>	南方航空CZ3967	长沙	上海	黄花国际机场T2	浦东国际机场T2	June 4, 2019, 12:40 p.m.	June 4, 2019, 3 p.m.	100	400.0	0	0.0
<input type="checkbox"/>	东方航空MU2533	武汉	上海	天河国际机场T3	浦东国际机场T1	June 4, 2019, 3:05 p.m.	June 4, 2019, 4:40 p.m.	89	430.0	1	430.0
<input type="checkbox"/>	南方航空CZ3579	武汉	上海	天河国际机场T3	浦东国际机场T2	June 4, 2019, 2:10 p.m.	June 4, 2019, 3:45 p.m.	80	430.0	0	0.0
<input type="checkbox"/>	东方航空MU2543	武汉	上海	天河国际机场T3	浦东国际机场T1	June 4, 2019, 12:20 p.m.	June 4, 2019, 1:50 p.m.	100	430.0	0	0.0
<input type="checkbox"/>	南方航空CZ6197	武汉	上海	天河国际机场T3	浦东国际机场T2	June 4, 2019, 8:30 a.m.	June 4, 2019, 10 a.m.	99	390.0	1	390.0
<input type="checkbox"/>	南方航空CZ6171	武汉	上海	天河国际机场T3	浦东国际机场T2	June 4, 2019, 10:40 p.m.	June 5, 2019, 12:10 a.m.	80	340.0	0	0.0
<input type="checkbox"/>	吉祥航空HO1074	武汉	上海	天河国际机场T3	浦东国际机场T2	June 4, 2019, 11:05 a.m.	June 4, 2019, 12:35 p.m.	80	310.0	0	0.0
<input type="checkbox"/>	中国国航CA8215	武汉	北京	天河国际机场T3	首都国际机场T2	June 4, 2019, 7:50 p.m.	June 4, 2019, 11:35 p.m.	100	1620.0	0	0.0
<input type="checkbox"/>	中国国航CA8209	武汉	北京	天河国际机场T3	首都国际机场T2	June 4, 2019, 7:05 p.m.	June 4, 2019, 9:05 p.m.	100	1620.0	0	0.0
<input type="checkbox"/>	中国国航CA8201	武汉	北京	天河国际机场T3	首都国际机场T2	June 4, 2019, 8:10 a.m.	June 4, 2019, 10:10 a.m.	90	1620.0	0	0.0
<input type="checkbox"/>	东方航空MU2455	武汉	北京	天河国际机场T3	首都国际机场T2	June 4, 2019, 5:55 p.m.	June 4, 2019, 7:55 p.m.	80	1610.0	0	0.0
<input type="checkbox"/>	东方航空MU2451	武汉	北京	天河国际机场T3	首都国际机场T2	June 4, 2019, 8:20 a.m.	June 4, 2019, 10:20 a.m.	90	1610.0	0	0.0
<input type="checkbox"/>	厦门航空MF1291	武汉	北京	天河国际机场T3	首都国际机场T2	June 4, 2019, 7:50 a.m.	June 4, 2019, 9:55 a.m.	80	1610.0	0	0.0
<input type="checkbox"/>	南方航空CZ3117	武汉	北京	天河国际机场T3	首都国际机场T2	June 4, 2019, 7:50 a.m.	June 4, 2019, 9:55 a.m.	80	1610.0	0	0.0
<input type="checkbox"/>	东方航空MU2455	武汉	北京	天河国际机场T3	首都国际机场T2	June 3, 2019, 5:55 p.m.	June 3, 2019, 7:55 p.m.	80	1610.0	0	0.0

图 4-74 航班信息

也可以选中航班删除。

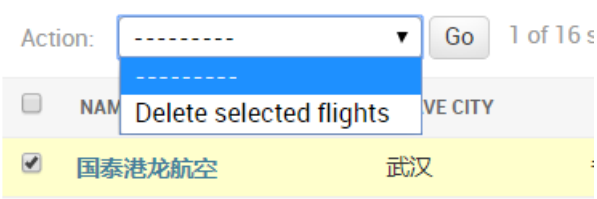


图 4-75 删除航班

当然，也可以添加航班。

可以按照自定义的数据添加航班所需信息。

Add flight

Name:	国泰港龙航空
Leave city:	武汉
Arrive city:	香港
Leave airport:	天河国际机场
Arrive airport:	香港国际机场T1
Leave time:	Date: 2019-06-17 Today Time: 20:50:00 Now
Arrive time:	Date: 2019-06-17 Today Time: 22:50:00 Now
Capacity:	90
Price:	866
Book sum:	0
Income:	0

[Save and add another](#) [Save and continue editing](#) [SAVE](#)

图 4-76 添加航班

添加完航班，再去搜索，发现航班已经被加上。

武汉	香港	2019/06/17	重新搜索
航班信息	起飞时间	到达时间	价格
国泰港龙航空	20:50 天河国际机场	22:50 香港国际机场T1	座位数 8/100 ¥866.0 订票

图 4-77 添加航班成功

4.7 系统设计与实现总结

在本次课程设计中，采用 B/S 模式实现了一个机票预订系统。

完成的主要工作如下：

- 完成了需求分析
- 根据需求分析完成了数据库的分析，该建立哪些表
- 设计了界面
- 完成了模块的编写，实现了后台功能
- 进行了完整性测试，安全性测试。用户密码被加密，不能通过数据库查看。

5 课程总结

本次课程实验分为两部分，一部分是 SQL 实验，另外一部分是课程设计。这两个部分各有侧重点，层层递进。牢牢掌握 SQL 语句会对后面的实验有一定的帮助。课程设计则是将应用与数据库结合起来，数据库只是一个工具，在必要时使用就行。

在 SQL 实验部分不仅仅是练习 SQL 查询语句，还做了许多工作。一开始就让我们练习了 sqlserver 的两种备份方式，并且让我们自己增加了用户，还配置了权限。同时根据实验所给出的表自己在 sqlserver 里面建立表，并自己准备了一些数据，便于后面的实验。

在实验一中，还完成了对基本表的增删改查的操作，数据的导入导出。更为重要的是，还编写了一个触发器。触发器的编写是通过网上介绍基本触发器的知识，自己试着完成的。触发器是数据库的很重要的一部分，但有没有触发器完全是数据库开发者决定的，有的时候需要，有的时候并不需要。但了解这方面的知识还是很重要的。触发器的存在有助于完整性控制规则。比如在实验中，不允许自己给自己点赞，这就需要用到一个触发器，当插入或更新点赞表时，检查一下，如果自己给自己点赞，那么就 rollback。接下来就是这个实验中比较难的一部分了，有 17 道查询的题目需要用 sql 语句完成。因为当初对 sql 语句不太熟练，所以这部分花了很长的时间。之后的实验就用到了数据库更加高级的功能，函数、存储过程与事务，函数其实就是一个封装好的东西，用 sql 语句实现。类似地，存储过程也是如此。在网上查找了大量资料，也是对它们有了更深刻的理解，而不是简简单单停留在如何去实现它们上。事务其实在上面的触发器中也用到了，就是 rollback，这里的事务有很多特性，在实验中给出的是原子性的检查。事务出现有效防止了可能会出现错误操作。

实验二就是一个课程设计，需要理解需求分析并且自己设计数据库，app。其实在这个实验中数据库是一个工具，怎样利用好它就需要实验一的基础了。本次课程设计的工作量不算小，准备利用自己熟悉的语言来完成这个工作。界面的设计参照了一些模板，尽量让界面美观，同时信息量要大，这样才能增强用户体验，否则做的就是一个没有用的工作。然后本次实验还是把中心放在了数据库这边，主要考虑的就是数据库的一致性，完整性，安全性的控制。数据库一致性要求数据要一致，不能在两边数据不一样，比如我这边退票了，那么我的取票中心就应该没有这张票了等等。还有安全性的考虑，数据库存储的是加密后的密码，而不是明文，这在一定程度上保护了用户密码的安全性。

经过这次实验，让我深刻地理解了数据库的各种机制，并且能在实践项目中运用好这个工具，这在今后的工作中是非常有意义的，丰富了自己的知识，还掌握了一个非常强大的工具。

本次数据库设计还有做的不是很好的地方，比如界面的易操作性，数据库的有些功能没有利用上等等。

如果能做的更好的话，一定是要在这些方面改进的，比如运用好数据库的函数功能等等，自己创建一些函数或者存储过程完成一些工作，实现一个接口能够让编写程序的人更加方便。如果可能，还想在性能上有一定的优化。

6 附录

源代码已经在电子档中，因此这里只会给出核心功能的代码（为了篇幅较短）。

```
1. from django.contrib.auth.models import Permission, User
2. from django.db import models
3.
4.
5. # Create your models here.
6. class Flight(models.Model):
7.     user = models.ManyToManyField(User, default=1)
8.     name = models.CharField(max_length=100) # 航班信息
9.     leave_city = models.CharField(max_length=100, null=True) # 离开城市
10.    arrive_city = models.CharField(max_length=100, null=True) # 到达城市
11.    leave_airport = models.CharField(max_length=100, null=True) # 离开机场
12.    arrive_airport = models.CharField(max_length=100, null=True) # 到达机
    场
13.    leave_time = models.DateTimeField(null=True) # 离开时间
14.    arrive_time = models.DateTimeField(null=True) # 到达时间
15.    capacity = models.IntegerField(default=0, null=True) # 座位数
16.    price = models.FloatField(default=0, null=True) # 价格
17.    book_sum = models.IntegerField(default=0, null=True) # 订票人数
18.    income = models.FloatField(default=0, null=True) # 收入
19.
20.    def __str__(self):
21.        return self.name
22.
23.
24. class FlightSeats(models.Model):
25.     flight_id = models.ForeignKey('Flight', on_delete=models.CASCADE)
26.     seats = models.IntegerField(default=0, null=True)
27.
28.
29. class PassengerBills(models.Model):
30.     flight_id = models.ForeignKey('Flight', on_delete=models.CASCADE)
31.     user = models.ManyToManyField(User, default=1)
32.     bills = models.FloatField(default=0, null=True)
33.
34.
35. class TicketNotice(models.Model):
36.     user = models.ManyToManyField(User, default=1)
37.     flight_id = models.ForeignKey('Flight', on_delete=models.CASCADE)
38.     notice_time = models.DateTimeField(null=True)
39.     name = models.CharField(max_length=100)
40.     leave_airport = models.CharField(max_length=100, null=True)
41.     arrive_airport = models.CharField(max_length=100, null=True)

1. def book_ticket(request, flight_id):
2.     if not request.user.is_authenticated():
3.         return render(request, 'booksystem/login.html')
4.     else:
5.         flight = Flight.objects.get(pk=flight_id)
6.         flightseats_tuple = FlightSeats.objects.get_or_create(flight_id_id=f
light_id)
7.         flightseats = flightseats_tuple[0]
8.         userbills_tuple = PassengerBills.objects.get_or_create(flight_id_id=
flight_id)
9.         userbills = userbills_tuple[0]
```

```

10.         ticket_notice_tuple = TicketNotice.objects.get_or_create(flight_id_id=flight_id)
11.         ticket_notice = ticket_notice_tuple[0]
12.         booked_flights = Flight.objects.filter(user=request.user)
13.
14.         if flight in booked_flights:
15.             return render(request, 'booksystem/book_conflict.html')
16.
17.         # 不能重复订购
18.         if request.method == 'POST':
19.             if flight.capacity > 0:
20.                 flight.book_sum += 1
21.                 flight.capacity -= 1
22.                 flight.income += flight.price
23.                 flight.user.add(request.user)
24.                 flight.save()
25.                 flightseats.seats = flight.book_sum
26.                 flightseats.save()
27.                 userbills.user.add(request.user)
28.                 userbills.bills += flight.price
29.                 userbills.save()
30.                 ticket_notice.notice_time = flight.leave_time + datetime.timedelta(-1)
31.                 ticket_notice.user.add(request.user)
32.                 ticket_notice.name = flight.name
33.                 ticket_notice.leave_airport = flight.leave_airport
34.                 ticket_notice.arrive_airport = flight.arrive_airport
35.                 ticket_notice.save()
36.             context = {
37.                 'flight': flight,
38.                 'username': request.user.username
39.             }
40.             return render(request, 'booksystem/book_flight.html', context)

```

```

1. def refund_ticket(request, flight_id):
2.     flight = Flight.objects.get(pk=flight_id)
3.     flightseats_tuple = FlightSeats.objects.get_or_create(flight_id_id=flight_id)
4.     flightseats = flightseats_tuple[0]
5.     userbills_tuple = PassengerBills.objects.get_or_create(flight_id_id=flight_id)
6.     userbills = userbills_tuple[0]
7.     ticket_notice_tuple = TicketNotice.objects.get_or_create(flight_id_id=flight_id)
8.     ticket_notice = ticket_notice_tuple[0]
9.     flight.book_sum -= 1
10.    flight.capacity += 1
11.    flight.income -= flight.price
12.    flight.user.remove(request.user)
13.    flight.save()
14.    flightseats.seats = flight.book_sum
15.    flightseats.save()
16.    userbills.bills -= flight.price
17.    ticket_notice.user.remove(request.user)
18.    TicketNotice.objects.filter(flight_id_id=flight_id, user=request.user).delete()
19.    ticket_notice.save()
20.    return HttpResponseRedirect('/booksystem/user_order')

```

```

1. def result(request):
2.     if request.method == 'POST':
3.         form = PassengerInfoForm(request.POST)
4.         if form.is_valid():
5.             pleave_city = form.cleaned_data.get('leave_city')
6.             parrive_city = form.cleaned_data.get('arrive_city')
7.             pleave_date = form.cleaned_data.get('leave_date')
8.             pleave_time = datetime.datetime.combine(pleave_date, datetime.time())
9.             # 搜索航班
10.            all_flights = Flight.objects.filter(leave_city=pleave_city, arrive_city=parrive_city)
11.            avliable_f = []
12.            for flight in all_flights:
13.                flight.leave_time = flight.leave_time.replace(tzinfo=None)
14.                if flight.leave_time.date() == pleave_date:
15.                    avliable_f.append(flight)
16.            avliable_f_leave_time = sorted(avliable_f, key=attrgetter('leave_time'))
17.            avliable_f_arrive_time = sorted(avliable_f, key=attrgetter('arrive_time'), reverse=True)
18.            avliable_f_price = sorted(avliable_f, key=attrgetter('price'))
19.
20.            time_format = '%H:%M'
21.
22.            for flight in avliable_f_price:
23.                flight.leave_time = flight.leave_time.strftime(time_format)
24.                flight.arrive_time = flight.arrive_time.strftime(time_format)
25.
26.            dfail = 'none'
27.            dhead = 'block'
28.            if len(avliable_f_price) == 0:
29.                dfail = 'block'
30.                dhead = 'none'
31.            context = {
32.                # 搜索框数据
33.                'leave_city': pleave_city,
34.                'arrive_city': parrive_city,
35.                'leave_date': str(pleave_date),
36.                # 搜索结果
37.                'avliable_f_leave_time': avliable_f_leave_time,
38.                'avliable_f_arrive_time': avliable_f_arrive_time,
39.                'avliable_f_price': avliable_f_price,
40.                # 标记
41.                'dhead': dhead,
42.                'dfail': dfail
43.            }
44.            if request.user.is_authenticated():
45.                context['username'] = request.user.username
46.            return render(request, 'booksystem/result.html', context)
47.        else:
48.            return render(request, 'booksystem/result.html')
49.    else:
50.        context = {
51.            'dhead': 'none',
52.            'dfail': 'none'
53.        }
54.    return render(request, 'booksystem/result.html', context)

```

```

1. def finance_report(request):

```

```

2.     flights_a = Flight.objects.all()
3.     flights_a = sorted(flights_a, key=attrgetter('leave_time'))
4.
5.     per_month_incomes = []
6.     per_year_incomes = []
7.     month_set = set()
8.     year_set = set()
9.     for flight in flights_a:
10.        if flight.income > 0:
11.            this_month = flight.leave_time.strftime('%m') # datetime 获取
月
12.            per_month_incomes.append((this_month, flight.income)) # tuple (
month, income)
13.            month_set.add(this_month)
14.            this_year = flight.leave_time.strftime('%Y') # datetime 获取年
15.            per_year_incomes.append((this_year, flight.income)) # tuple (ye
ar, income)
16.            year_set.add(this_year)
17.
18.     month_incomes = []
19.     for month in month_set:
20.         income = sum(per_month_income[1] for per_month_income in per_month_i
ncomes if per_month_income[0] == month)
21.         month_income = MyIncomeList(month, income)
22.         month_incomes.append(month_income)
23.     month_incomes = sorted(month_incomes, key=attrgetter('timeStore'))
24.
25.     #季度
26.     period_incomes = []
27.     income1 = sum(month_income.income for month_income in month_incomes if m
onth_income.timeStore == '01' or month_income.timeStore == '02' or month_inc
ome.timeStore == '03')
28.     income2 = sum(month_income.income for month_income in month_incomes if m
onth_income.timeStore == '04' or month_income.timeStore == '05' or month_inc
ome.timeStore == '06')
29.     income3 = sum(month_income.income for month_income in month_incomes if m
onth_income.timeStore == '07' or month_income.timeStore == '08' or month_inc
ome.timeStore == '09')
30.     income4 = sum(month_income.income for month_income in month_incomes if m
onth_income.timeStore == '10' or month_income.timeStore == '11' or month_inc
ome.timeStore == '12')
31.     period_income1 = MyIncomeList('01', income1)
32.     period_incomes.append(period_income1)
33.     period_income2 = MyIncomeList('02', income2)
34.     period_incomes.append(period_income2)
35.     period_income3 = MyIncomeList('03', income3)
36.     period_incomes.append(period_income3)
37.     period_income4 = MyIncomeList('04', income4)
38.     period_incomes.append(period_income4)
39.     period_incomes = sorted(period_incomes, key=attrgetter('timeStore'))
40.
41.     # 年
42.     year_incomes = []
43.     for year in year_set:
44.         income = sum(per_year_income[1] for per_year_income in per_year_inco
mes if per_year_income[0] == year)
45.         year_income = MyIncomeList(year, income)
46.         year_incomes.append(year_income)
47.     year_incomes = sorted(year_incomes, key=attrgetter('timeStore'))
48.
49.     passengers = User.objects.exclude(pk=1)
50.     order_set = set()
51.     for passenger in passengers:
52.         flights = Flight.objects.filter(user=passenger)
53.         for flight in flights:

```

```
54.         route = flight.leave_city + ' → ' + flight.arrive_city
55.         order = Order(passenger.username, flight.name, route, flight.leave_time, flight.price)
56.         order_set.add(order)
57.
58.     context = {
59.         'month_incomes': month_incomes,
60.         'year_incomes': year_incomes,
61.         'period_incomes': period_incomes,
62.         'order_set': order_set
63.     }
64.     return context
```