华中科技大学

数据库系统原理实践报告

专　　业：	计算机科学与技术
班　　级：	CS2304
学　　号：	U202315594
姓　　名：	贾柠泽
指导教师：	左琼教师

分数	
教师签名	

2025 年　6 月　20 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	

总分	

目　录

1 课程任务概述

本实践课程旨在通过16个实训模块，系统培养学生在MySQL数据库管理、开发及优化方面的综合能力，涵盖数据库设计、SQL 编程、性能优化、安全管理、应用开发五大核心领域。学生需完成从基础操作（如建库、建表）到高级应用（如存储过程、事务控制、高并发处理）的全流程实战任务，最终达到独立设计、实现并维护企业级数据库系统的能力。

本课程以MySQL为例，系统性地设计了一系列的实训任务，基础内容涉及以下几个部分，并可结合实际对DBMS原理的掌握情况向内核设计延伸：

1) 数据库、表、索引、视图、约束、存储过程、函数、触发器、游标等数据对象的管理与编程；
2) 数据查询，数据插入、删除与修改等数据处理相关任务；
3) 数据库的安全性控制，完整性控制，恢复机制，并发控制机制等系统内核的实验；
4) 数据库的设计与实现；
5) 数据库应用系统的开发(JAVA 篇)。

课程依托头歌实践教学平台，实验环境为Linux操作系统下的MySQL 8.0.28（主要为8.028版本，部分关卡使用8.022版本，使用中基本无差别）。在数据库应用开发环节，使用JAVA 1.8。

本课程相关资料网站：

MYSQL手册：https://dev.mysql.com/doc/

JAVA手册：https://docs.oracle.com/javase/8/docs/api/index.html

课程开放资源：https://gitee.com/kylin8575543/db2022-spring

2 任务实施过程与分析

本次实践课程在头歌平台进行，实践任务均在平台上提交代码，所有完成的任务、关卡均通过了自动测评。本次实践最终完成了课程平台中的第 1~11、14、15 实训任务，下面将重点针对其中的****任务阐述其完成过程中的具体工作。

2.1 基于金融应用的数据查询(Select)

此部分主要考察对于数据库中表中信息的数据查询操作，主要内容为通过 SELECT 语句实现包括但不限于单表查询、嵌套查询、（多）条件查询、多表连接、对于查询结果的排序、分组、消重及统计、特殊函数、子查询、谓词\关键词、对表达式列命名、衍生表、分表合并等一系列操作。此部分实验中通过了 1~19 关除 17 关的关卡，仅展示部分实验的详细思路。

2.1.1 办理了储蓄卡的客户信息

查询办理了储蓄卡的客户名称、手机号、银行卡号。将 client 和 bank_card 通过 join 进行多表连接，条件是 b_c_id = c_id，然后筛选出办理储蓄卡的客户，具体代码如下：

```
    select c_name,c_phone,b_number from client join bank_card on b_c_id = c_id
    where b_type='储蓄卡'
    order by c_id;
```

2.1.2 商品收益的众数

查询资产表中所有资产记录里商品收益的众数和它出现的次数。对 property 表使用 group by 子句对进行分组统计，并且使用 having 子句选出其中元组个数最大的分组，在这里 having 子句有两种思路，第一种思路为选择组内元组个数大于等于所有分组元组个数的分组，使用关键字 ALL 和子查询，具体代码如下：

```
    HAVING COUNT(*) >= (SELECT MAX(num) FROM (SELECT COUNT(*) AS num FROM property GROUP BY pro_income) AS t);
```

另一种思路为选择组内元组数目大于等于最大的元组个数，使用 MAX 函数和子查询，具体代码如下：

```
    HAVING COUNT(*) >= (SELECT MAX(num) FROM (SELECT COUNT(*) AS num FROM property GROUP BY pro_income) AS t);
```

2.1.3 持有两张信用卡的用户

查询在本行持有两张及以上信用卡的客户信息。首先在子查询中对于 b_c_id 和 b_type 按照分组查询，选择元组数大于 1 的组别，通过(c_id, '信用卡')外部筛

选，通过 c_id 在 client 查找相关信息。具体代码如下：

```
SELECT c_name, c_id_card, c_phone
FROM client
WHERE (c_id, '信用卡') in (
    SELECT b_c_id, b_type
    FROM bank_card
    GROUP BY b_c_id, b_type
    HAVING COUNT(*) > 1
);
```

2.1.4 投资总收益前三名的客户

查询投资总收益前三名的客户。对于 client 和 property 进行等值连接，然后选取其中 pro_status 状态为“可用”的元组，之后按照 c_id 分组，使用 SUM 函数统计其中的 pro_income，然后通过 rank 函数进行对 total_income 降序排序，最后获取其中的前三个元组，即总收益前 3 名的客户。具体代码如下：

```
Select c_name, c_id_card, sum(pro_income) as total_income
from client, property
where pro_c_id=c_id and pro_status="可用"
group by c_id
having rank() over(order by total_income desc) > 3;
```

2.1.5 客户理财、保险与基金投资总额

查询客户理财、保险、基金投资金额的总和，并排序。此问题的查询通过派生表的统计、多表合并统计总金额。具体来说，首先通过分别对于理财产品表(finances_product)、保险表(insurance)和基金表(fund)与资产表(property)进行连接，通过 SUM(商品数量*该产品每份金额)公式进行计算不同用户每类商品的投资总金额，最终通过 union all（此处同样可以使用 union）将三个派生表进行合并，然后再与客户表(client)进行 left outer join 连接，获得可以进行投资金额查询的派生表。

然后进行投资金额的统计。按照用户 c_id 进行分组，统计每个用户的投资总金额。具体代码省略。

2.1.6 第 N 高问题

查询每份保险金额第 4 高保险产品的编号和保险金额。通过子查询对于保险表(insurance)按照金额进行排序，考虑到可能会有相同金额的保险产品，因此通过 dense_rank() over(order by 列名)进行排名的获取，得到派生表，最终通过设定 rank 为 4 的保险进行获取，按照 i_id 进行排序获取最终结果。

```
    SELECT i_id, i_amount
    FROM (
        SELECT i_id, i_amount, dense_rank() OVER (ORDER BY i_amount
DESC) AS rk
        FROM insurance
         ) t
    WHERE rk = 4
    ORDER BY i_id;
```

2.1.7 持有完全相同基金组合的客户

查询持有完全相同基金组合的客户。首先需要构造两张相同的客户基金组合派生表，然后进行元组的选取。

构造客户基金组合派生表时，主要通过使用聚集函数 GROUP_CONCAT 进行组内基金号的合并，命名为 f_id，生成客户基金的派生表，分别定义其为两张表 t1 和 t2。进行元组选取时，则通过进行 f_id 的比较，按照选取前用户的 id 小于后用户 id 的原则进行元组的选取，最终按照用户 id 进行升序排列，具体代码省略。

2.2 数据查询(Select)之二

此部分主要考察对于数据库中表中信息的数据查询操作，主要内容为通过 SELECT 语句实现包括但不限于嵌套查询、子查询、关键函数使用等一系列操作。此部分实验中通过了 1~4 关所有的关卡，仅展示部分实验的详细思路。

2.2.1 查询销售总额前三的理财产品

关联 property（资产表）和 finances_product（理财产品表），通过 pro_pif_id = p_id 匹配理财产品的购买记录。过滤 2010 和 2011 年的数据（YEAR(pro_purchase_time) IN (2010, 2011)）。按年份和产品分组（GROUP BY YEAR(...), p_id），计算每类理财产品年度销售总额（SUM(pro_quantity * p_amount)），仅统计类型 1（pro_type=1）的产品。

使用 RANK() OVER (PARTITION BY ... ORDER BY ... DESC)生成年度销售额排名，PARTITION BY YEAR(...)按年份分区，每年独立计算排名。ORDER BY sumamount DESC 销售额降序排序，相同销售额并列排名（如两个第 1 名，下一个为第 3 名）。

从子查询结果中筛选排名≤3 的记录（rk <= 3），返回年份、排名、产品 ID 和销售额。

2.2.2 查询购买了所有畅销理财产品的客户

内层子查询：识别热门理财产品生成所有“热门理财产品”列表（被 >2 个客户投资），按产品 ID 分组（GROUP BY pro_pif_id）用 HAVING 过滤投资客户数 >2 的产品

```
SELECT pro_pif_id
FROM property
WHERE pro_type = 1
GROUP BY pro_pif_id
HAVING COUNT(DISTINCT pro_c_id) > 2
```

中层子查询：验证客户是否遗漏热门产品，检查当前客户（p1.pro_c_id）是否投资了某个热门产品（popular.pro_pif_id）

```
SELECT 1
FROM property p2
WHERE p2.pro_c_id = p1.pro_c_id
  AND p2.pro_pif_id = popular.pro_pif_id
  AND p2.pro_type = 1
```

外层 NOT EXISTS 逻辑，确保客户没有遗漏任何一个热门产品。若存在至少一个热门产品客户未投资 → 不满足条件，被过滤掉。若客户投资了所有热门产品 → 条件成立，被保留。

```
NOT EXISTS (
  SELECT 1
  FROM (热门产品列表) popular
  WHERE NOT EXISTS (中层子查询)   -- 客户未投资该热门产品
)
```

2.3 数据的插入、修改与删除(Insert,Update,Delete)

此部分主要考察对于数据库中表中数据进行插入、删除与修改操作，主要内容为通过 INSERT、DELETE 和 UPDATE 实现上述操作。此部分实验中通过了 1~6 关所有的关卡。

2.3.1 插入多条完整的客户信息

使用 INSERT INTO <表名> VALUES (元组值)进行数据插入。具体代码如下：

```
INSERT INTO client
VALUES
(1,'林惠雯','960323053@qq.com',
'411014196712130323','15609032348','Mop5UPkl');
```

2.3.2 插入不完整的客户信息

使用 INSERT 插入不完整信息要声明列名，具体代码如下：

```
INSERT INTO client (c_id, c_name, c_phone, c_id_card, c_password)
VALUES(33,'蔡依婷','18820762130',
'350972199204227621','MKwEuc1sc6');
```

2.3.3 批量插入数据

对于结构完全相同的数据可以通过子查询方式进行插入。具体代码如下：

```
INSERT INTO client
select * from new_client;
```

2.3.4 删除没有银行卡的客户信息

使用 DELETE FROM <表名> <子查询>进行数据删除。具体代码如下：

```
DELETE FROM  client
where not exists (select * from bank_card where client.c_id = bank_card.b_c_id);
```

2.3.5 冻结客户资金

使用 UPDATE 语句，WHERE 子句使用嵌套子查询。具体代码如下：

```
update property
set pro_status = '冻结'
where pro_c_id in (
    select c_id
    from client
    where c_phone = '13686431238'
);
```

2.3.6 连接更新

使用 UPDATE 语句，通过 set 子句，将 pro_c_id 和 c_id 进行等值连接，通过 c_id_card 更新对应的 pro_id_card。具体代码如下：

```
update property
set pro_id_card = (
    select c_id_card from client where pro_c_id = c_id
);
```

2.4 视图

此部分主要考察对于数据库进行视图的创建、基于视图的查询操作等，主要

内容为通过 CREATE VIEW 进行视图创建、SELECT 基于视图进行查询。此部分实验中通过了 1~2 关所有的关卡。

2.4.1 创建所有保险资产的详细记录视图

使用 CREATE 语句实现视图的创建，具体实现语句为 CREATE VIEW <视图名> [<列名>[，<列名>，…]] AS <子查询> [WITH CHECK OPTION];。在本次实验中为对于 client 表、property 表和 insurance 表进行等值连接，将其中所需要的属性投影到新的外模式中。具体代码如下：

```
CREATE VIEW v_insurance_detail AS
SELECT
c_name,c_id_card,i_name,i_project,pro_status,pro_quantity,i_amount,i_year,
pro_income,pro_purchase_time
FROM client,property, insurance
WHERE
    c_id = pro_c_id
    AND pro_type = 2
    AND pro_pif_id = i_id;
```

2.4.2 基于视图的查询

基于视图的查询与基于基本表的查询没有太大的区别，具体代码如下：

```
SELECT
    c_name,
    c_id_card,
    SUM(pro_quantity * i_amount) AS insurance_total_amount,
    SUM(pro_income) AS insurance_total_revenue
FROM v_insurance_detail
GROUP BY c_id_card
ORDER BY insurance_total_amount desc;
```

2.5 触发器

此部分主要考察 MySQL 的流程控制编程、触发器的基本知识、触发器的创

建、触发触发器的时机、触发触发器的事件以及触发器内的特殊表等。具体来说就是通过创建触发器进行合法性检查，要求在数据的 INSERT、UPDATE 和 DELETE 的过程中，对于数据的完整性进行检查，依照检查结果进行反馈，包括正确执行或者返回出错信息。此部分实验中通过了第 1 关所有的关卡。

2.5.1 为投资表 property 实现业务约束规则-根据投资类别分别引用不同表的主码

在本部分，主要为插入数据构建触发器，对应的基本原则是 insert 触发器可以访问 new 表,其内容为 insert 的新数据，因此可以通过 new 表的数据进行信息是否出错的判断。

首先，需要创建触发器结构，在 INSERT 插入数据之前对于新数据进行合法性的判断。

其次则需要考虑错误的具体情况，在此部分为 4 中错误情况：

（1） pro_type 数据不合法时，即插入的资产类型不在已有的资产类列表里面时，显示: type x is illegal!

（2） pro_type = 1,但 pro_pif_id 不是 finances_product 表中的某个主码值，显示:finances product #x not found!

（3） pro_type = 2,但 pro_pif_id 不是 insurance 表中的某个主码值，显示:insurance #x not found!

（4） pro_type = 3,但 pro_pif_id 不是 fund 表中的某个主码值，显示:fund #x not found!

在过程中，根据不同的情况获取报错信息之后，通过 SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;进行报错信息的抛出。

2.6 安全性控制

此部分主要考察 MySQL 的安全控制机制、create user 语句的使用和 rant 和 revoke 语句的使用等。具体来说就是通过自主存取控制方法，通过授予创建用户、特定用户特定的权限和收回特定的权限等，保证数据库安全，防止数据泄露。此部分实验中通过了 1~2 关所有的关卡。

2.6.1 用户和权限

使用 CREATE USER <用户名> identified by <用户登录密码>创建用户，具体

代码如下：

```
create user tom identified by '123456';
```

使用 GRANT [SELECT|DELETE|UPDATE|INSERT] <列名> on <表名> to <用户> [with grant option]进行插入、删除、修改或者查询权限的授予。具体代码如下：

```
grant
select (c_name, c_mail, c_phone) on client to tom with grant option;
```

使用 REVOKE 权限[,权限]... on 数据库对象 from user|role[,user|role]...将权限从用户或者角色中收回。具体代码如下：

```
revoke
select on bank_card from Cindy;
```

2.6.2 用户、角色与权限

创建角色，授予角色一组权限，并将角色代表的权限授予指定的一组用户。使用 CREATE ROLE <角色名>，具体代码如下：

```
create role client_manager;
```

将权限授予角色的方式与授予角色的方式相似。具体代码如下：

```
grant
select, insert, update on client to client_manager;
```

使用 GRANT <角色> TO <用户>将角色权限授予用户。具体代码如下：

```
grant client_manager to tom, jerry;
```

2.7 数据库设计与实现

此部分主要考察数据库设计的阶段和每阶段的任务、概念模型、逻辑模型及其与概念模型的关系、在 DBMS 中的物理实现等。具体来说就是在整体数据库的设计流程中。首先在需求分析的基础上，设计概念模型（主要为 E-R 图方法），进一步转换成逻辑模型，最后进行物理模型的构建，完成整个数据库的设计。此部分实验中通过了 1~3 关所有的关卡。

2.7.1 从概念模型到 MySQL 实现

此实验主要通过对于建模好的概念模型，主要是用 ER 图描述数据及数据间

的关系，进行关系模型的转换，进而构建基本表。此实验针对的机票订票系统包括用户、旅客、机场、航空公司、民航飞机、航班常规调度表、航班表、机票八个实体以及各种关系。同时，为了保证实体间关系的正确表达，可以将其中的 ticket、flight、flightschedule 等视作关系，将其中的关联关系作为其外码进行管理。

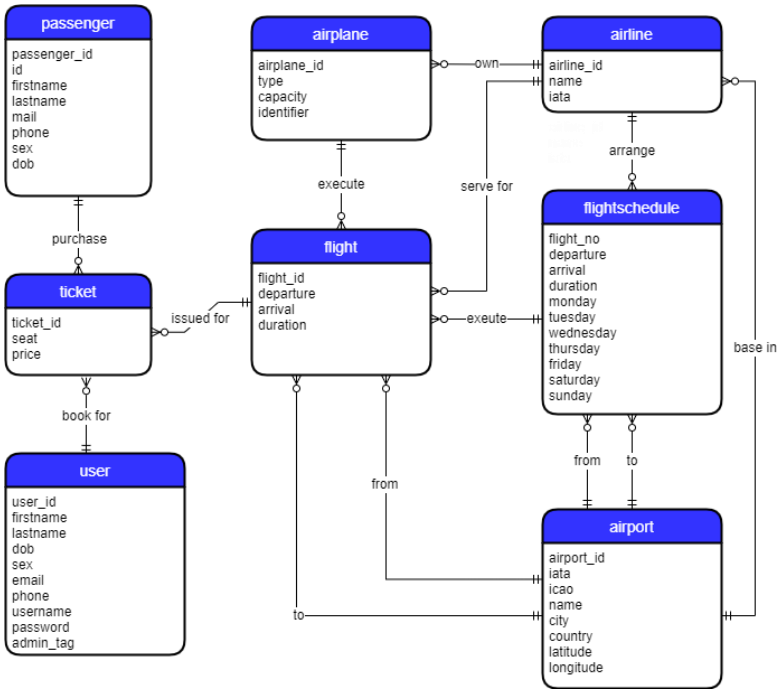

图 2.1 机票订票系统概念模型

除此之外，建表时也要同时设立完整性约束，包括建库，建表，创建主码，外码，索引，指定缺省，不能为空等约束，所有索引采用 BTREE。在列名与关键字同名的时候，需要将列名添加单引号。

2.7.2 从需求分析到逻辑模型

本实验需要设计一个影院管理系统。影院对当前的放映厅和电影进行排片，顾客到来后，可以购买任一排场的电影票，进入对应放映厅观看。系统主要包括电影(movie)、顾客(customer)、放映厅(hall)、排场(schedule)和电影票(ticket)五个实体，同时也提供了实体间的关系，根据其设计概念模型完成 E-R 图如图 2.2 所示，具体链接为 https://github.com/2980454492/mysql/blob/main/ersolution.jpg。

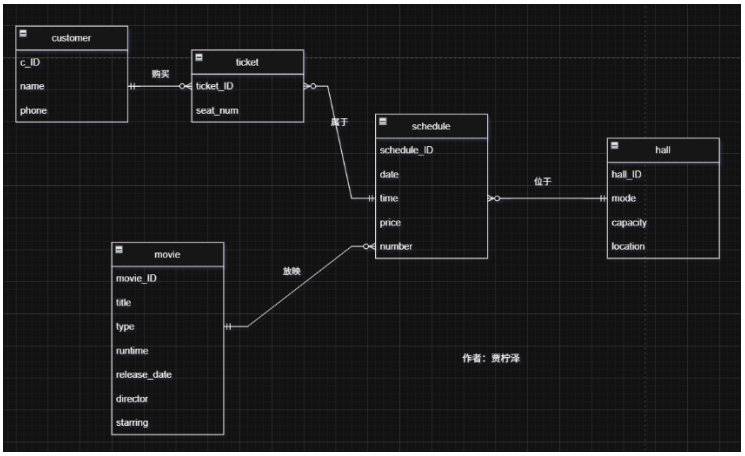

图 2.2 影院管理系统概念模型

根据 E-R 图转换成关系模式如下：

电 影 (movie)(movie_ID,title,type,runtime,release_date,director,starring); 主码:(movie_ID);

顾客(customer)(c_ID,name,phone);主码:(c_ID);

放映厅(hall)(hall_ID,mode,capacity,location);主码:(hall_ID);

排场(schedule)(schedule_ID, date, time, price, number, hall_ID, movie_ID);主码:(schedule_ID);外码:(hall_ID,movie_ID);

电 影 票 (ticket)(ticket_ID,seat_num,c_ID,schedule_ID); 主 码 :(ticket_ID); 外码:(c_ID,schedule_ID);

2.7.3 建模工具的使用

此部分需要单独安装 MySQL Workbench，利用 MySQL Workbench 的 forward engineering 功能，将已有的模型文件自动转换成 SQL 脚本。具体来说，将已建模的模型文件 rabc.mwb 导入到 MySQL Workbench 中，如图 2.3 所示。

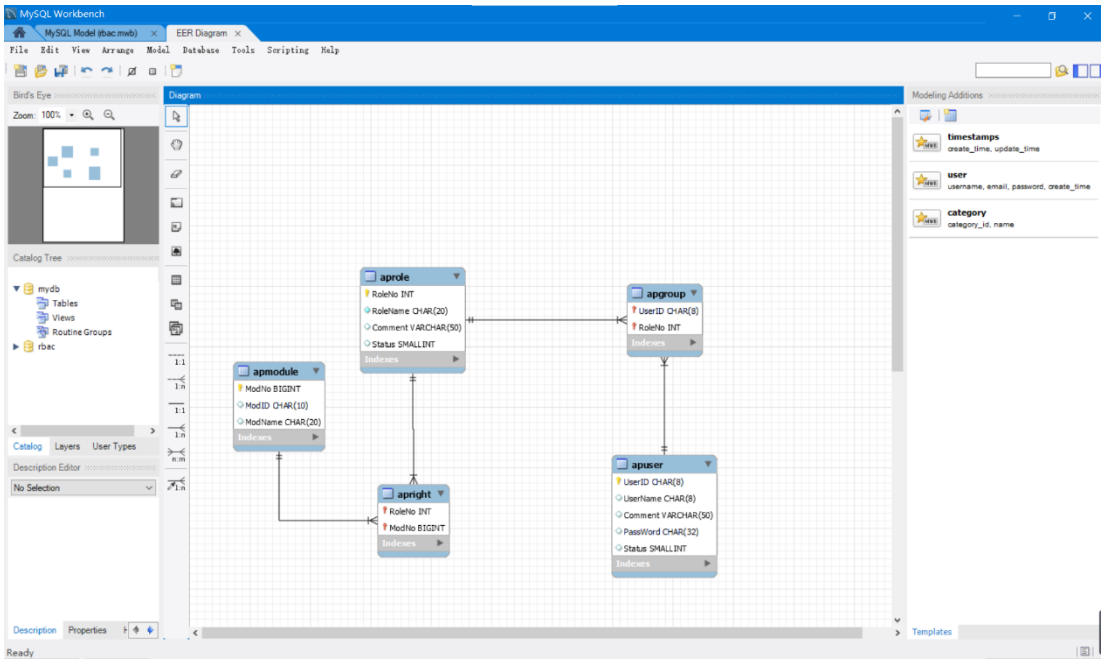

图 2.3 建模文件导入

然后使用菜单栏中的 Database->Forward Engineer 功能，使用 localhost 作为 connection，输入数据库密码则可以完成 SQL 脚本导出，如图 2.4 所示。

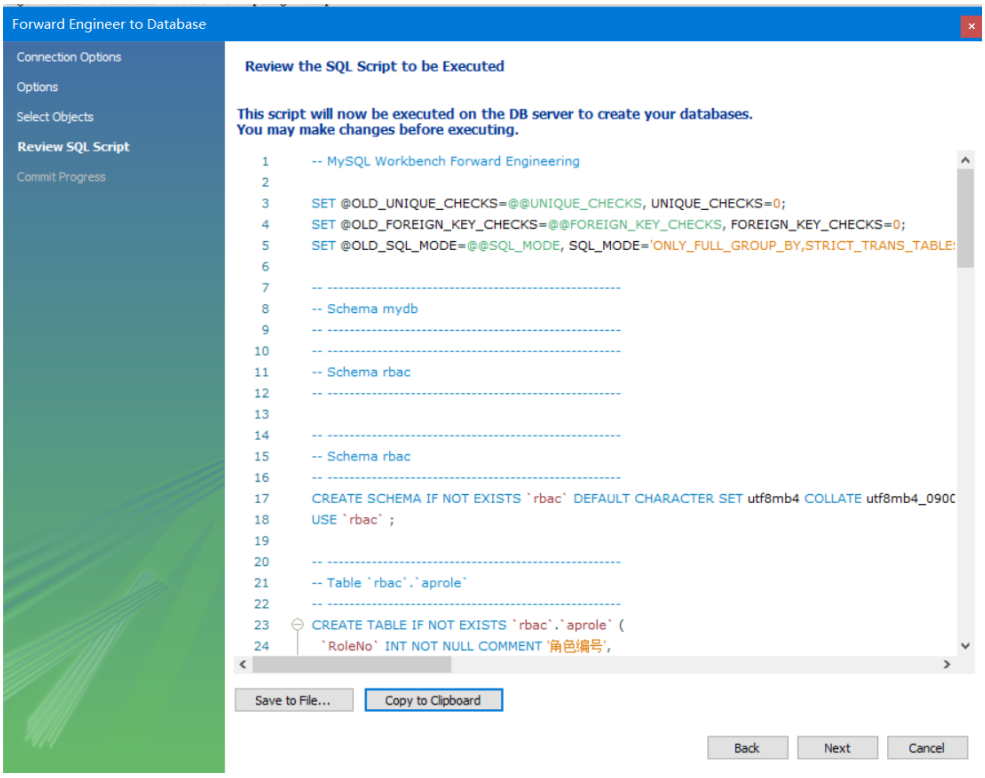

图 2.4 SQL 语句导出

2.7.4 制约因素分析与设计

在进行需求分析时需要充分考虑社会上的需求，在充分考虑的基础上并建立

概念模型，例如在上述图 2.1 的机票订票系统概念模型以及图 2.2 影院管理系统概念模型，都是在充分进行需求分析和考虑的基础上实现的。在购价数据库的逻辑模型和物理模型时，则需要考虑数据库的安全性问题，隔离性、一致性、合法性问题，尤其是在系统中，权限的设置尤为重要，例如在机票订票系统概念模型中，管理员的权限和用户的权限一定是被严格制定的，这样才能保证用户的隐私性。为了数据库安全性，也可以在存储数据库时进行加密，设计密文存储数据库，即密态云盘等。

2.7.5 工程师责任及其分析

工程师不仅需要完成技术上的需求，也需要承担社会责任。不仅需要完成设计数据库结构，设计相应系统体系，设计概念模型等计算机操作方面的工作，更需要考虑和分析设计的应用场景、法律效力、社会安全、隐私信息等的问题，在设计过程中，要使设计切合需求并要考虑实现成本，也要注意其法律效力，不能违背法律规定，需要在设计数据库时，同时考虑社会、健康、安全、法律以及文化等各因素之间的相互影响，让数据库更好服务于社会。

2.8 数据库应用开发(JAVA 篇)

此部分主要考察 JDBC 的体系结构、JDBC 的核心组件、使用步骤等。具体来说就是通过 JAVA 高级语言进行数据库应用系统的开发和使用，能够同时利用数据库的数据和高级语言面向对象进行编程的特性，在此部分需要通过 JDBC 实现操纵数据库的一系列操作，实现 Java 应用程序与各种不同数据库之间进行对话。此部分实验中通过了 1~6 关。

2.8.1 JDBC 体系结构和简单的查询

JDBC（Java DataBase Connectivity,java 数据库连接）是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。

JDBC 的核心组件包括：

（1） DriverManager： 此类管理数据库驱动程序列表。使用通信子协议将来自 java 应用程序的连接请求与适当的数据库驱动程序匹配。

（2） Driver:此接口处理与数据库服务器的通信，我们很少会直接与 Driver

对象进行交互。而是使用 DriverManager 对象来管理这种类型的对象。

（3） Connection：该界面具有用于联系数据库的所有方法。连接对象表示通信上下文，即，与数据库的所有通信仅通过连接对象。

（4） Statement：使用从此接口创建的对象将 SQL 语句提交到数据库。除了执行存储过程之外，一些派生接口还接受参数。

（5） ResultSet：在使用 Statement 对象执行 SQL 查询后，这些对象保存从数据库检索的数据。它作为一个迭代器，允许我们遍历其数据。

（6） SQLException：此类处理数据库应用程序中发生的任何错误。

构建 JDBC 应用程序涉及以下六个步骤：

（1） 导入包：需要包含包含数据库编程所需的 JDBC 类的包。大多数情况下，使用 import java.sql.*就足够了。

（2） 注册 JDBC 驱动程序：要求您初始化驱动程序，以便您可以打开与数据库的通信通道。

（3） 打开连接：需要使用 DriverManager.getConnection（）方法创建一个 Connection 对象，该对象表示与数据库的物理连接。

（4） 执行查询：需要使用类型为 Statement 的对象来构建和提交 SQL 语句到数据库。

（5） 从结果集中提取数据：需要使用相应的 ResultSet.getXXX（）方法从结果集中检索数据。

（6） 释放资源：需要明确地关闭所有数据库资源，而不依赖于 JVM 的垃圾收集。

2.8.2 用户登录

编写客户登录程序，提示用户输入邮箱和密码，并判断正确性，给出适当的提示信息。首先需要获取用户输入的用户名和密码。然后进行数据库的连接以及 JDBC 的创建，构建接口 prepareStatement 包含用户名和密码，将结果存入 resultSet 中，通过 next()方法进行查询，如果用户名和密码一致，则输出“登陆成功。”，否则输出“用户名或密码错误！”。

2.8.3 添加新客户

编程完成向 client(客户表)插入记录的方法。已知用户的 id、姓名、邮箱、身

份证号、电话和登陆密码，将上述信息与已连接的数据库的 Connection 对象传入方法中。可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的插入操作。

具体的关键代码如下：

```
String sql = "insert into client values(?, ?, ?, ?, ?, ?)";
        try {
            PreparedStatement pps = connection.prepareStatement(sql);
            pps.setInt(1, c_id);
            pps.setString(2, c_name);
            pps.setString(3, c_mail);
            pps.setString(4, c_id_card);
            pps.setString(5, c_phone);
            pps.setString(6, c_password);
            return pps.executeUpdate();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return 0;
```

2.8.4 银行卡销户

编写一个能删除指定客户编号的指定银行卡号的方法。具体方法与插入类似。已知用户的 id 和银行卡号，将上述信息与已连接的数据库的 Connection 对象传入方法中。可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的删除操作。

2.8.5 客户修改密码

编写修改客户登录密码的的方法。具体方法也与插入类似。首先通过一系列方法获取需要修改的用户，同时获得其旧密码，进一步对于旧密码进行更新，将用户的邮箱、旧密码和新密码与已连接的数据库的 Connection 对象传入方法中。

可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的更新操作。

2.8.6 事务与转账操作

使用 JDBC 的事务处理编写一个银行卡转账方法。需要传入已连接数据库的 Connection 对象、转出账号、转入账号和转账金额。首先设置隔离级别为 REPEATABLE READ，具体实现为 connection. SetTransactionIsolation(4)。然后，进行转出账号扣账，转入账号还账，为储蓄卡时入账。然后进行操作合法性的检验，如果操作不合法，例如账号不存在，扣账金额不足等，则进行事务回滚，具体实现为 connection.rollback()，且置返回值为 false。其他情况下则进行事务提交，具体实现为 connection.commit()，且置返回值为 true。

3 课程总结

本次数据库系统原理实践整体完成了 2.1~2.14 的所有实训任务（除了存储过程与事务中的最后一个关卡），在整个实验过程中依次完成了数据库的基础实践，包括数据库中相关创建、删除、修改、查询等操作，进一步探索数据库的安全性控制、并发控制、自定义函数、数据库恢复等数据库中意义重大的相关技术，深入了解了数据库管理系统的整体框架和特点，对于数据库及其上层应用程序有了更加深入的理解。

对于各个实训任务，具体主要工作如下：完成了数据库、表、完整性约束条件的定义相关的所有实训任务，掌握了如何使用数据定义语言 DDL 以及如何定义约束条件；完成了表结构和完整性约束的修改相关的所有实训任务，掌握了如何利用 SQL 语句实现对表结构和约束条件的修改；完成了数据查询相关的所有实训任务，对于基本的数据查询以及数据查询过程中所用到的函数、方法、技巧有了很好的掌握；完成了数据的插入、删除、修改、连接更新相关的所有实训任务；完成了创建视图与基于视图的查询，进一步理解了模式和外模式之间的映像，掌握了应用程序是如何基于视图这一层次进行查询；完成了存储过程与事务相关的部分实训任务，掌握了三种存储过程的具体结构和实现方法；完成了触发器相关的实训任务，掌握了根据要求完成触发器的设计；完成了用户自定义函数相关的实训任务，掌握了根据要求完成自定义函数的设计；完成了创建用户、角色以及权限授予相关的所有实训任务，掌握了自主存取方式对于用户权限的设置和对于数据库的保护；完成了事务并发控制与隔离级别相关的所有实训任务，加深了对读脏、不可重复读、幻读、可串行化的理解，掌握了对于数据库事务的加锁操作；完成了使用备份和日志文件实现数据恢复相关的所有实训任务，对于数据库恢复相关技术有更加深入的理解；完成了数据库设计相关的所有实训任务，掌握了概念模型设计、关系模式设计、建模实现设计；完成了 JAVA 数据库应用开发的所有实训任务，掌握了 JDBC 体系实现数据库应用设计。