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：	CS2203
学　　号：	U202215643
姓　　名：	王国豪
指导教师：	左　琼

分数	
教师签名	

2024 年 7 月 5 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	
6	

总分	

目　录

1 课程任务概述

“数据库系统原理实践”是配合“数据库系统原理”课程独立开设的实践课，注重理论与实践相结合。本课程以 MySQL 为例，系统性地设计了一系列的实训任务，内容涉及以下几个部分：

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，实践课程 url 见相关课堂教师发布链接及其邀请码。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。

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实训 1 到实训 16 的所有任务，下面将重点针对其中的各个实训中重点的任务阐述其完成过程中的具体工作。

2.1 数据库、表与完整性约束的定义(Create)

本章要求了解数据库、表与完整性约束的定义方法，了解数据库语言的基本语法。

本任务已完成全部关卡，由于较为简单，不展开分析。

2.2 表结构与完整性约束的修改（ALTER）

本章学习 alter table 语句的大部分功能和修改表结构的方法（如添加列、约束，删除列、约束，修改列）等基础知识，了解表的完整性约束。

已完成任务全部关卡，由于较为简单，不展开分析。

2.3 数据查询（Select）之一

本章采用某银行的一个金融场景应用的模拟数据库，提供了六个表，分别为：client（客户表）、bank_card（银行卡）、finances_product（理财资产表）、insurance（保险表）、fund（基金表）、property（资产表）。 本实训中大多采用 select 语句来对表进行查询满足要求的各项数据，设计了 子查询、数据库函数等知识点。

此部分实验已完成全部关卡，将针对第 2~3 关，第 6 关以及第 8~19 关详细展开分析。

2.3.2 邮箱为 null 的客户

本关卡已完成，因较为简单故略过分析，需要注意的是判断是否为 null 不能用 = ，要用 is null。

2.3.3 既买了保险又买了基金的客户

本关卡已完成。

查询既买了保险又买了基金的客户名称、邮箱和电话，结果按照 c_id 排序，仅利用 select、from、where exists、order by 即可实现，代码见图 2. 1。

```sql
SELECT c_name, c_mail, c_phone FROM client a
WHERE EXISTS (SELECT * FROM property WHERE pro_c_id = a.c_id AND pro_type = 2)
AND EXISTS (SELECT * FROM property WHERE pro_c_id = a.c_id AND pro_type = 3)
ORDER BY c_id;
```

图 2.1 任务 2.3.3 代码

2.3.6 商品收益的众数

在 SQL 中增加 HAVING 子句原因是，WHERE 关键字无法与聚合函数一起使用。HAVING 子句可以让我们筛选分组后的各组数据。本关需要用到 count 函数计算出现次数，用 having 语句实现的“众数”的要求，代码见图 2.2。

```sql
SELECT column1, aggregate_function(column2)
FROM table_name
GROUP BY column1
HAVING condition;
```

图 2.2 任务 2.3.6 代码

2.3.8 持有两张信用卡的用

查询持有两张（含）以上信用卡的用户的相关信息。本关同样使用嵌套查询，先在子查询中筛选出含有两张及其以上信用卡的用户的 id，再进行选取以及排序，代码见图 2.3。

```sql
SELECT c_name, c_id_card, c_phone FROM client
WHERE c_id IN (SELECT b_c_id FROM bank_card
WHERE b_type = '信用卡' GROUP BY b_c_id HAVING COUNT(*) >= 2)
ORDER BY c_id;
```

图 2.3 任务 2.3.8 代码

2.3.9 购买了货币型基金的客户信息

本关需要使用两次嵌套查询，先找出 f_type = ‘货币型’的 f_id，再从 property 表中找出 pro_type = 3（意为基金） 且 pro_pif_id 在第一个集合中的 pro_c_id，最后从 client 表中找到 c_id 在第二个 集合中的用户。代码见图 2.4。

```sql
select c_name,c_phone,c_mail from client
where c_id in ( select pro_c_id from property
    where pro_pif_id in( select f_id from fund where f_type = '货币型' ) and pro_type = 3)
order by c_id;
```

图 2.4 任务 2.3.9 代码

2.3.10 投资总收益前三名的客户

这一关涉及到子查询已经聚类函数 sum 以及用 limit 的输出前三个元组，代码见图 2.5。

```
select c.c_name,c_id_card,ch.sums as total_income
from client c,(select pro_c_id,sum(pro_income) as sums from property where pro_status =
'可用' group by pro_c_id ) as ch
where c_id = ch.pro_c_id
order by ch.sums desc limit 3;
```

图 2.5 任务 2.3.10 代码

2.3.11 黄姓客户持卡数量

这一关要学会用左外连接(left join),以及通配符%的使用，代码见图 2.6。

```
select c_id,c_name, count(b_c_id) number_of_cards
from client as c left join  bank_card as b
on c.c_id = b.b_c_id
where c_name like "黄%" group by c_id order by number_of_cards desc,c_id;
```

图 2.6 任务 2.3.11 代码

2.3.12 客户理财、保险与基金投资总额

本关要求计算用户的投资总金额，由于资产分为三类（理财、保险、 基金），故要用三个 select 语句去分别计算这三类资产的金额。之后，通过 union all 指令将不同的 select 结果连接（注意连接的 select 结果必须拥有相同数量的列，列也必须拥有相似的数据类型，同时，每条 select 语句中的列的顺序必须相同），得到了资产金额表后，将其与 client 表连接，用 sum 函数计算总金额即可， 但是计算时还要注意使用 ifnull 函数将空值转换成 0 值。

注意这里需要用到的是 union all，因为 union 不会去除重复的元组。由于篇幅问题，此关代码不展示。

2.3.13 客户总资产

使用外连接 left outer join 和 ifnull 函数实现在两表连接时将无记录项置为 0 的操作，并多次使用外连接和子查询统计多表项的数据和，最后在 select 语句中对多项数据进行加减操作运算以计算得出客户总资产。具体的代码不展示。

有一个细节值得注意，在 bank_card 表中信用卡余额即为透支金额，我这里使用了一个 if 的判别语句，见图 2.7，如果是储蓄卡就统计 b_balance,否则统计其相反数，其实也可以是用两次查询实现，分别判断是储蓄卡还是信用卡。

```
    if(
        b_type = "储蓄卡",
        b_balance,
        - b_balance
    )
```

图 2.7 任务 2.3.13 中 if 用法

2.3.14 第 N 高问题

本关只需使用一次嵌 套循环即可完成，使用 limit N, M 指令，可以从第 N 条记录开始，返回 M 条记 录，故在将 insurance 表中数据排序后很容易就能找到第 4 高的产品。代码见图 2.8。

```
SELECT DISTINCT i_id, i_amount
FROM insurance
WHERE i_amount = (
    SELECT DISTINCT i_amount FROM insurance ORDER BY i_amount DESC LIMIT 3, 1
);
```

图 2.8 任务 2.3.14 代码

2.3.15 基金收益两种方式排名

本关要求分别实现全局名次不连续的排名和连续的排名，即考察 rank() over 和 dense_rank() over 的区别，前者为名次不连续的排名，后者为连续的排名，代码见图 2.9。

```
-- (1) 基金总收益排名(名次不连续)
select pro_c_id, sum(pro_income) as total_revenue,
rank() over(order by sum(pro_income) desc) as `rank`
from property where pro_type = 3 group by pro_c_id
order by sum(pro_income) desc, pro_c_id;
-- (2) 基金总收益排名(名次连续)
select pro_c_id, sum(pro_income) as total_revenue,
dense_rank() over(order by sum(pro_income) desc) as `rank`
from property where pro_type = 3 group by pro_c_id
order by sum(pro_income) desc, pro_c_id;
```

图 2.9 任务 2.3.15 代码

2.3.16 持有完全相同基金组合的用户

查询所有持有相同基金组合的用户对。本关的思路为：先将两个 property 表连接（命名为 first 和 second），连接条件为两表中的 pro_type 均等于 3 且两个 表的 pro_pif_id 相等，这样可以将有相同基金的两个人连到一行里面，然后使用 select 语句查询 A、B 用户，按照 first 中的 c_id 和 second 中的 c_id 分组，使用 having 语句保证选出的行中，基金数与 A 和 B 持有的基金数均相等，

即可保证 A 和 B 持有的基金完全相同。

由于篇幅问题，代码在附录中给出。

2.3.17 购买基金的高峰期

首先进行 select 生成派生表，对每天的交易量进行计算(pro_quantity * f_amount)，并使用 sum 函数求和，标注为 total_amount，同时可根据 datediff 函数及相关计算求出该天是 2021 年第几个工作日（命名为 workday）。之后外部的一层查询将交易量未达到一百万的记录排除，使用 row_number() 函数对按照 workday 排序后的数据生成从 1 开始的行号，命名为 rownum。我们注意到，在连续的若干交易日期中，如果交易量均超过一百万，那么 workday 和 rownum 均是连续的。因此通过计算 workday 和 rownum 的差值并按此进行分组，使用 count 函数计算每组的行数，结果大于等于 3 的那些组，就是购买基金的高峰期。

由于篇幅限制，本关代码不再给出。

2.3.18 至少有一张信用卡余额超过 5000 元的客户信用卡总余额

通过嵌套查询找出有一张信用卡余额超过 5000 的客户编号，再进一步筛选，然后用 sum 函数求出符合的客户的总额度，代码见图 2. 10。图 2. 10 任务 2.3.18 代码

```
select b_c_id, sum(b_balance) as credit_card_amount from bank_card
where b_type = "信用卡" and b_c_id in
( select b_c_id from bank_card where b_type = "信用卡" and b_balance > 5000)
group by b_c_id order by b_c_id;
```

图 2. 10 任务 2.3.18 代码

2.3.19 以日历表格式显示每日基金购买总金额

以日历表格式列出 2022 年 2 余额每周每日基金购买总金额。本关需要用一次嵌套循环，先将 property 表与 fund 表连接，连接条件为 pro_pif_id = f_id，从中挑出 2022 年 2 月的数据，按照购买时间 pro_purchase_time 分组，使用 sum 计算每一交易日期的总交易金额（定义为 amount），并用 week 函数计算出该日期是 2 月的第几周，用 weekday 函数算出是星期几(定义为‘id’)。之后用外层的 select 算出所有周一至周五 9 的基金购买总金额，外层这里计算的关键代码见图 2. 11。

```
sum(if(`id` = 0, amount, null)) Monday,
sum(if(`id` = 1, amount, null)) Tuesday,
sum(if(`id` = 2, amount, null)) Wednesday,
sum(if(`id` = 3, amount, null)) Thursday,
sum(if(`id` = 4, amount, null)) Friday
```

图 2.11 任务 2.3.19 关键代码

2.4 数据查询(Select)-新增

使用 MySQL 语言实现数据查询。掌握 select 及其相关语句的使用。相较于 2.3 难度提升。

此部分实验已完成全部关卡，由于较为复杂，逐一进行分析。

2.4.1 查询销售总额前三的理财产品

查询 2010 年和 2011 年这两年每年销售总额前 3 名（如果有并列排名，则后 续排名号跳过之前的并列排名个数，如 1、1、3）的相关信息。本关卡需要使用 一次嵌套循环，内部的 select 查询出年份在 2010 和 2011 年中的数据（具体是哪 一年定义为 pyear），按 p_id 和 pyear 分组，用 sum 函数计算出销售总额（p_amount * pro_quantity），定义为 sumamount。外层的 select 使用 rank() over 函数对选出 的记录进行排名（以 pyear 分组，sumamount 为排序依据）。如果不分组，那么结果的排序按照顺序来，关键代码见图 2.12。

```
select pyear,
rank() over(partition by pyear order by sumamount desc) as rk,
p_id, sumamount
```

图 2.12 任务 2.4.1 关键代码

2.4.2 投资积极且偏好理财类产品的客户

查询所有此类客户的编号。本关 使用两个子查询先分别计算用户购买理财产品的数量（cnt1）和购买基金的数量 （cnt2），定义为两个表 t1 和 t2，将两个表按照 pro_c_id 连接，使用外层的 select 筛选出。

由于篇幅限制，本关代码不给出。

2.4.3 查询购买了所有畅销理财产品的客户

本关使用了嵌套了两层 where not exists 的查询，首先生成畅销理财产品的表与被用户购买了的理财产品的表。内层的 where not exists 找出那些没有被购买的畅销理财产品，（如果有）再利用外层的 where not exists 查找没有漏掉购买任何一个畅销理财产品的人（即购买了所有畅销理财 产品的人）。

由于篇幅限制，本关代码不给出。

2.4.4 查找相似的理财产品

首先从 property 表中找到全体持有 14 号理财产品的客户 以及每个客户的持有数量，定义为 t1 表；使用 dense_rank() over 函数将其中的数 据按持有数量排序，定义为 t2 表；从 t2 表中选出持有 14 号理财产品数量最多的 前三名（但可能不止三个人），定义为 t3 表；t3 表与 property 进行自然连接后， 从中选出前三名持有的理财产品（除 14 号以外），定义为 t4 表；t4 表与 property 再进行自然连接，数据按 pro_pif_id 分组，按 pro_pif_id 排序，选出 pro_pif_id 及该编号的理财产品的购买客户总人数 cc，定义为 t5 表；从 t5 表中查找 pro_pif_id，cc，以及用 dense_rank() over 函数生成的排名。

由于篇幅限制，本关代码不再给出。

2.4.5 查询任意两个客户的相同理财产品数

本关使用两个 property（命名为 p1 和 p2）进行自然连 接，按 p1.pro_c_id，p2.pro_c_id 进行分组，使用 count 函数计算两人持有的相同 理财产品的数量，使用 having count 满足“至少 2 种”的要求。

由于篇幅限制，本关代码不再给出。

2.4.6 查找相似的理财客户

本关卡已完成。

本关卡需要明确“相似的理财客户”的定义（比较复杂故不再赘述），先将 property 表定义为 t1，将从 property 中 select 出的 pro_type = 1 的用户 id 和业务 id 定义为 t2 表，t1 和 t2 表进行连接后，表的每一行就有了（A 客户，B 客户， 他们共同拥有的产品 id）这样的数据（但注意此时 A 客户和 B 客户可能是同一 个人）之后从连接的表中 select 出不同的 A 客户作为 pac，不同的 B 客户作为 pbc， 用 count 计算他们共同持有的理财产品数作为 common，将这些内容定义为 t3 表； 使用 rank() over 函数按照 pac 分组，按 common 降序，pbc 升序的方式进行排名， 作为 t4 表；从 t4 表中选出排名值小于 3 的相似客户即可。

由于篇幅限制，本关代码不再给出。

2.5 数据的插入、修改与删除

使用 MySQL 语言实现表数据的插入、修改与删除。需要掌握 Insert,Update,Delete 及其相关语句的使用。

此部分实验已完成全部关卡，着重分析第 6 关卡。

2.5.6 连接更新

这一关涉及到两个表，我直接 update 后跟两个表，通过 where 条件进行筛选后赋值，代码见

```
UPDATE property,client
SET pro_id_card=c_id_card
WHERE pro_c_id = c_id;
```

图 2.13 任务 2.5.6 代码

2.6 视图

视图（view）是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，数据库中只存放了视图的定义，而并没有存放视图中的数据，这些数据存放在原来的表中。使用视图查询时，数据库会从原来的表中取出对应的数据。本章要求学会用 create 创建视图，以及利用视图 查找表中数据。

此部分实验已完成全部关卡，由于较为简单，不展开分析。

2.7 存储过程与事务

存储过程是在大型数据库系统中，一组为了完成特定功能的 SQL 语句集，存储在数据库中，其经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果有）来调用存储过程。本章要求学会使用流程控制语句/游标/事务的存储过程。

2.7.1 使用流程控制语句的存储过程

首先使用 delimiter 重定义 mysql 分隔符为$$，使用 create procedure 创建存储过程，在函数体中，先用 declare 定义相关参数，使用流程控制语句以及迭代的思想设计插入斐波那契数列的功能，对前两项单独判断，第三项及以后的值为其前面两项之和。

由于篇幅限制，本关代码不再给出。

2.7.2 使用游标的存储过程

本关要先设置两个游标，分别用于指向护士和医生，且还要设置一个 NOT FOUND 的 HANDLER(处理例程)用来判断是否到达末尾，在我们的循环中进行排班工作，我们每次先从游标 fetch 一下，接着判断是否达到末尾，如果到达末尾的我们需要把游标关闭又开启，这样我们又能重新开始读取数据，安排完护士以后我们就可以进行医生的排班，我们要注意的是主任不安排在周末，我们要先判断当前是星期几，如果是星期一我们需要判断我们设置的存储变量是否为空，不为空就将存储变量中的主任放入排班，如果存储变量为空，则我们需要进行正

常的排班，且如果当前是周末的话，我们就要进行判断是否是主任，如果是主任我们就要把主任放在存储变量里面，然后取出下一个医生进行排班，依次往复操作。

由于篇幅限制，代码不给出。

2.7.3 使用事务的存储过程

这一关要读懂题目的意思，区分储蓄卡和信用卡的状态，理清楚，用三个 update 负责选择更新转账的需求，接着需要用 if 语句首先判断转出卡的余额是否剩余，若没有剩余，返回 0，接着检查接受者以及其卡号是否存在，若不存在，返回 0，剩下的情况就是符合的情况，进行提交。

由于篇幅限制，代码于附录中给出。

2.8 触发器

触发器（trigger）是由事件来触发某个操作。这些事件包括 insert 语句、update 语句和 delete 语句。当数据库系统执行这些事件时，就会激活触发器执行相应的操作。本章要求为指定的表编写触发器以实现特定的业务规则。

2.8.1 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

理解题意后这题变得简单，我们首先需要注意我们的触发器类型是行级触发器，每插入一行就触发一次，接着我们需要对插入的数据的 type 进行 1,2,3 类型分别处理，且我们插入数据的 pro_pif_id 必须是对应表的 id,然后如果不符合条件，就进行异常处理，最后如果我们插入的数据 type 不在 1,2,3 的范围内，我们也要发出 illegal 的异常处理。代码见图 2. 12。

```
delimiter $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW  -- 行级触发器
BEGIN
DECLARE msg VARCHAR(50);
IF new.pro_type = 1 AND NOT EXISTS (SELECT * FROM finances_product WHERE p_id = new.pro_pif_id) THEN
    SET msg = CONCAT("finances product #",new.pro_pif_id," not found!");
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
ELSEIF new.pro_type = 2 AND NOT EXISTS (SELECT * FROM insurance WHERE i_id = new.pro_pif_id) THEN
    SET msg = CONCAT("insurance #",new.pro_pif_id," not found!");
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
ELSEIF new.pro_type = 3 AND NOT EXISTS (SELECT * FROM fund WHERE f_id = new.pro_pif_id) THEN
    SET msg = CONCAT("fund #",new.pro_pif_id," not found!");
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
ELSEIF new.pro_type NOT IN (1,2,3) THEN
    SET msg = CONCAT("type ",new.pro_type," is illegal!");
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
END IF;
END$$
delimiter ;
```

图 2. 14 任务 2.8.1 代码

2.9 用户自定义函数

本章要求掌握在 SELECT 语句中应用自定义函数。。具体来说就是通过定义特定的模块化函数，满足用户特定的需求，在之后的多次调用中可以发挥很大的作用。此部分实验通过了所有关卡。

2.9.1 创建函数并在语句中使用它

本关卡已完成。

使用 CREATE FUNCTION function_name([para data_type[,...]]) returns data_type begin function_body; return expression; end 语句进行用户函数的自定义。在此部分函数定义的具体代码见图 2. 15。

```
create function get_deposit(client_id int)
returns numeric(10,2)
begin
    return ( SELECT SUM(b_balance) FROM bank_card WHERE b_type = '储蓄卡'
        GROUP BY b_c_id HAVING b_c_id = client_id);
end$$
```

图 2. 15 任务 2.9.1 的定义函数代码

进一步调用自定义函数的方法，与其他函数的调用方法基本相同，具体代码见图 2. 16。

```
select *
from (select c_id_card, c_name, get_deposit(c_id) total_deposit
    from client) tmp
where total_deposit >= 1000000
order by total_deposit desc;
```

图 2. 16 任务 2.9.1 的调用函数代码

2.10 安全性控制

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 部分实验中通过了所有的关卡。

2.10.1 用户和权限

这一关我们要掌握 create user 语句和 grant 和 revoke 语句，这些语句都有固定的语法，具体的实现代码见图 2. 17。

```
#(1) 创建用户tom和jerry，初始密码均为'123456';
create user tom identified by '123456';
create user jerry identified by '123456';
#(2) 授予用户tom查询客户的姓名，邮箱和电话的权限,且tom可转授权限;
grant select (c_mail, c_name, c_phone)
on table client
to tom
with grant option;
#(3) 授予用户jerry修改银行卡余额的权限;
grant update (b_balance)
on table bank_card
to jerry;
#(4) 收回用户Cindy查询银行卡信息的权限。
revoke select on table bank_card from Cindy;
```

图 2.17 任务 2.10.1 代码

2.10.2 用户、角色与权限

这一关创建角色，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。这一关目的在于让我们熟悉角色的用法，体会其便利之处，实现代码与上一关有异曲同工之处，由于篇幅限制，代码不给出。

2.11 并发控制与事务的隔离级别

对于并发操作，可能会产生数据不一致性，如丢失修改(lost update)，读脏数据(dirty read)，不可重复读(non-repeatable read)，幻读(phantom read)，选择合适的事务隔离级别，使得两个事务并发执行时不发生数据不一致的问题。

2.11.1 并发控制与事务的隔离级别

使用 set session transaction isolation level 语句设置事务的隔离等 级。事务隔离级别从低到高分以下四级，读未提交（READ UNCOMITTED）、读 已提交（READ COMMITTED）、可重复读（REPEATABLE READ）、可串行化（SERIALIZABLE）。然后通过 start transaction;开启事务。具体代码见图 2.18。

```
set session transaction isolation level read uncommitted;
start transaction;
insert into dept(name) values('运维部');
rollback;
```

图 2.18 任务 2.11.1 代码

2.11.2 读脏

读脏(dirty read)，或者又叫脏读，是指一个事务(t1)读取到另一个事务(t2)修改后的数据，后来事务 t2 又撤销了本次修改(即事务 t2 以 roll back 结束)，数据恢复原值。这样，事务 t1 读到的数据就与数据库里的实际数据不一致，这样的数据被称为“脏”数据，意即不正确的数据。只有一种隔离级别可能会产生读脏，即为读不提交。

其中主要通过 set @n = sleep(时间)进行时间上的控制，以事务 1 为例，其事务的具体代码见图 2. 19。

```
set @n = sleep(1);
select tickets from ticket where flight_no = 'CA8213';
commit;
```

图 2. 19 任务 2.11.2 事务的实现代码

2.11.3 不可重复读

不可重复读(unrepeatable read)，是指一个事务(t1)读取到某数据后，另一个事务(t2)修改了该，事务 t1 并未修改该数据，但当 t1 再次读取该数据时，发现两次读取的结果不一样。不可重复读产生的原因，是事务 t1 的两次读取之间，有另一个事务修改了 t1 读取的数据。有两种级别可以发生不可重复读的现象。具体来说，本实验需要通过更加精细化的时间控制，实现对于两个事务不可重复读现象的重现。具体代码省略。

2.11.4 幻读

幻读是指一个事务(t1)读取到某数据后，另一个事务(t2)作了 insert 或 delete 操作，事务 t1 再次读取该数据时，魔幻般地发现数据变多了或者变少了(记录数量不一致)；而不可重复读限指事务 t2 作了 update 操作，致使 t1 的两次读操作读到的结果(数据的值)不一致。幻读与不可重复读之间的区别就是幻读是发现数据莫名其妙地增加或减少，而不可重复读现象是数据在两次读取时，发现读取的内容不一致的现象。幻读产生的原因，是事务 t1 的两次读取之间，有另一个事务 insert 或 delete 了 t1 读取的数据集。具体代码省略。

2.11.5 主动加锁保证可重复读

MySQL 提供了主动加锁的机制，使得在较低的隔离级别下，通过加锁，以实现更高级别的一致性。SELECT 语句支持 for share 和 for update 短语，分别表示对表加共享(Share)锁和写(write)锁，共享锁也叫读锁，写锁又叫排它锁。具体添加共享锁的代码见图 2. 20。

```
select tickets from ticket where flight_no = 'MU2455' for share;
```

图 2. 20 任务 2.11.5 添加共享锁代码

2.11.6 可串行化

多个事务并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同。两个事务 t1,t2 并发执行，如果结果与 t1→t2 串行执行的结果相同，或者与 t2→t1 串行执行的结果相同，都是正确的(可串行化的)。

选择除 serializable(可串行化)以外的任何隔离级别，保证两个事务并发执行的结果是可串行化的。需要通过时间控制，交错执行程序，实现可串行化调度。

具体代码省略。

2.12 备份+日志：介质故障与数据库恢复

MySQL 利用备份、日志文件实现恢复，在这一关中我们需要了解并掌握之本的恢复机制和备份与恢复工具。

此部分实验已完成全部关卡。由于篇幅限制，此部分实验不展开分析。

2.13 数据库设计与实现

本章将从实际出发，从按需求建表到设计 E-R 图并进一步转换为关系模型，再到使用建模工具对数据库进行建模，了解一个数据库从概念模型到具体实现的步骤。此外，在工程认证中，制约因素分析与设计和工程师责任及其分析也是必不可少的内容。

此部分完成全部关卡，逐一分析全部关卡。

2.13.1 从概念模型到 MySQL 实现

本关卡已完成。按要求建表即可，要注意不要遗漏约束，本关略过分析。

2.13.2 从需求分析到逻辑模型

本关已完成。

根据题目要求画出 E-R 图如下图 2.21，具体图片存放位置见 https://cdn.jsdelivr.net/gh/niuniu0101/storageofPicture/images/ersolution.jpg。

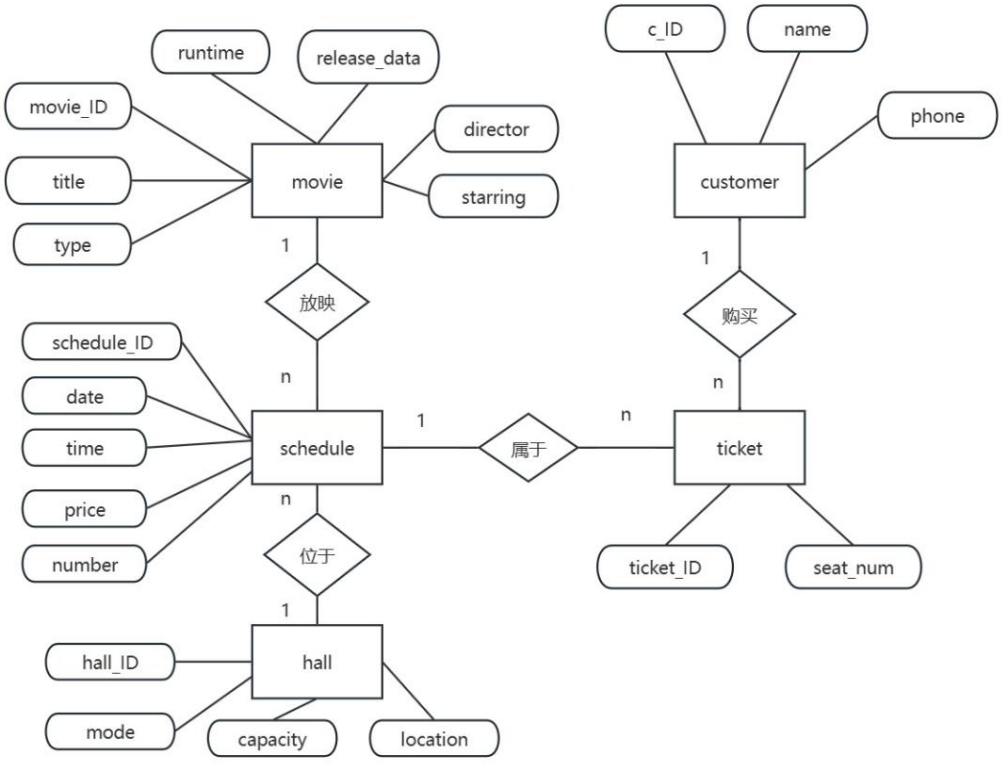

图 2.21 任务 2.13.2E-R 图

以此建立五个关系模式：

movie(movie_ID,title,type,runtime,release_data,director,starring),primary key:(movie_ID),foreign key:(schedule_ID)

customer(c_ID,name,phone), primary key:(c_ID)

hall(hall_ID,mode,capacity,location),primary key:(hall_ID)

schedule(schedule_ID,date,time,price,number), primary key:(schedule_ID),foreign key:(hall_ID,movie_ID)

ticket(ticket_ID,seat_num), primary key:(ticket_ID),foreign key:(c_ID,schedule_ID)

2.13.3 建模工具的使用

本关卡已完成。受篇幅限制，本关略过分析。

2.13.4 制约因素分析与设计

在实际工程中设计数据库时，除了满足客户需求外，还必须综合考虑社会、健康、安全、法律、文化及环境等制约因素。

社会层面：数据库设计应考虑用户友好性、数据透明性和隐私保护，确保公平性和无歧视性。设计要基于社会责任，满足不同背景用户的需求。

健康层面：确保健康相关数据的准确性和实时性，减少用户心理负担，提供简洁的操作界面和清晰的健康数据反馈，以帮助用户有效管理健康。

安全层面：设计方案必须保障数据安全与信息安全，采用数据加密、访问控制和多因素身份验证等措施，防止数据泄露和非法访问。

法律层面：设计必须合法合规，遵守相关数据保护法规，如 GDPR 和 HIPAA，保障用户权益，不利用数据库进行非法活动。

文化层面：设计应尊重不同文化背景，提供多语言支持和本地化服务，遵守工程职业道德和规范，维护用户权益。

环境层面：设计应优化数据库性能，减少能耗，采用绿色计算技术和云计算支持，考虑可持续发展原则，降低对环境的影响。

2.13.5 工程师责任及其分析

在产品实现过程中，社会、安全、法律及文化等因素对解决方案起到了制约和调整作用。工程师应承担一定的生态伦理责任，准确有效地说明新建工程或新技术可能带来的后果，避免对社会和生态环境的危害；同时，应承担职业伦理责任，秉持追求真理、客观求实、诚实公平的精神；并承担保护公众安全、健康和福祉的责任，不仅要遵守设计规则和标准，还应考虑产品或技术在市场上的效用。

工程师在设计、实现和维护数据库过程中，需综合考虑前述各种制约因素，设计出既能满足用户需求，又能在其他方面达到最优的数据库。同时，工程师必

须遵守法律，谨慎行事，不可投机取巧，不走歪门邪道。尤其在信息技术行业，由于数据库缺陷造成的损失可能是巨大的。

此外，中国的工程师还需肩负技术发展和提升国家综合实力的责任，不断进行创新设计，推动国家整体进步。通过综合考虑社会、安全、法律及文化等因素，工程师可以提供安全、可靠、高效的数据库解决方案，满足用户需求并促进社会和技术的可持续发展。

2.14 数据库应用开发（JAVA 篇）

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 提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序。

此部分实验完成所有的关卡，重点分析第 3 关到第 7 个关卡，由于篇幅限制，省略代码。

2.14.3 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。

具体的关键代码见图 2. 22。

```
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;
```

图 2. 22 任务 2.14.3 关键代码

2.14.4 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。

2.14.5 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。

2.14.6 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。

2.14.7 把稀疏表格转为键值对存储

将一个稀疏的表中有保存数据的列值，以键值对(列名，列值)的形式转存到另一个表中，这样可以直接丢失没有值列。具体来说，首先需要查询稀疏表中的所有元组，进行学生学号和各科成绩的提取，然后将其填入键值对表中。 然后，将已连接的数据库的 Connection 对象和键值对传入方法中。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。

3 课程总结

通过这次数据库系统原理的实践，我全面完成了从实训 1 到实训 16 的所有任务，深刻体验了数据库的基础操作，包括创建、删除、修改、查询等核心功能。在此基础上，我进一步探索了数据库的安全性控制、并发控制、自定义函数、数据库恢复等关键技术，这些技术对于数据库系统的运行和维护具有重要意义。通过这次实践，我对数据库管理系统的整体框架和特点有了更加深入的了解，同时也加深了对数据库及其上层应用程序的认识。

在具体实训任务中，我掌握了数据定义语言 DDL 的使用，学会了如何定义数据库、表以及完整性约束条件；我掌握了如何利用 SQL 语句修改表结构和约束条件；在数据查询方面，我熟练掌握了基本查询方法以及查询过程中所用到的函数、技巧；同时，我也完成了数据的插入、删除、修改、连接更新等相关任务。

此外，我还学会了创建视图与基于视图的查询，更加理解了模式和外模式之间的映像关系，以及应用程序是如何基于视图这一层次进行查询的。在存储过程与事务方面，我掌握了三种存储过程的具体结构和实现方法，并完成了相关实训任务。同时，我也掌握了触发器和用户自定义函数的设计方法。

在数据库安全性方面，我学会了创建用户、角色以及权限授予，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。在事务并发控制与隔离级别方面，我加深了对读脏、不可重复读、幻读、可串行化等概念的理解，并掌握了数据库事务的加锁操作。

最后，我还完成了使用备份和日志文件实现数据恢复的相关任务，对数据库恢复技术有了更加深入的理解。同时，我也掌握了数据库设计的全过程，包括概念模型设计、关系模式设计、建模实现设计等。通过这次实践，我还学会了如何利用 JDBC 体系实现数据库应用设计，这对于我未来的专业成长具有重要意义。

“纸上得来终觉浅，绝知此事要躬行”，总的来说，这次实践让我获得了巨大的成长。我不仅真正了解了数据库在应用程序中的重要作用，还深入掌握了如何通过数据库的系列操作实现数据的存储、管理和应用。特别是在数据查询方面，我以前从未想过可以使用 SQL 语句完成如此复杂的操作。这次实践让我更加自信地面对未来的专业挑战，并为我未来的职业发展奠定了坚实的基础。

附录

2.3.16 持有完全相同基金组合的用户

```
select first.pro_c_id as c_id1, second.pro_c_id as c_id2
from property first inner join property second on(
first.pro_c_id<second.pro_c_id
and first.pro_type='3' and second.pro_type='3' and
first.pro_pif_id=second.pro_pif_id
) # 将有相同基金的两个人连到一个元组中
group by c_id1,c_id2
# having 语句保证了选出的两个人持有的基金数相同
having count(*)=(
select count(*) from property
where pro_c_id=first.pro_c_id and pro_type='3' # 保证选出的元组，持有基金数与第一个人持有的基金数相同)
and count(*)=(
select count(*) from property
where pro_c_id=second.pro_c_id and pro_type='3 '# 保证选出的元组，持有基金数与第二个人持有的基金数相同);
```

2.7.3 使用事务的存储过程

```
create procedure sp_transfer(
                    IN applicant_id int,
                    IN source_card_id char(30),
                    IN receiver_id int,
                    IN dest_card_id char(30),
                    IN amount numeric(10,2),
                    OUT return_code int)
BEGIN
SET autocommit = OFF;
START TRANSACTION; -- 开启事务
    UPDATE bank_card SET b_balance = b_balance-amount WHERE b_number = source_card_id AND b_c_id =
applicant_id AND b_type = "储蓄卡";
    UPDATE bank_card SET b_balance = b_balance+amount WHERE b_number = dest_card_id AND b_c_id = receiver_id
AND b_type = "储蓄卡";
    UPDATE bank_card SET b_balance = b_balance-amount WHERE b_number = dest_card_id AND b_c_id = receiver_id
AND b_type = "信用卡";
    IF NOT EXISTS(SELECT * FROM bank_card WHERE b_number = source_card_id AND b_c_id = applicant_id AND
b_type = "储蓄卡" AND b_balance > 0) THEN    -- 检查申请的
        SET return_code = 0;
        ROLLBACK;
    ELSEIF NOT EXISTS(SELECT * FROM bank_card WHERE b_number = dest_card_id AND b_c_id = receiver_id) THEN -
- 检查接收到的
        SET return_code = 0;
        ROLLBACK;
    ELSE
        SET return_code = 1;
        COMMIT;
    END IF;
SET autocommit = TRUE;
END$$
```