

**数据库系统原理实践报告**

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计科2109 |
| 学 号： | U202111110 |
| 姓 名： | 罗劲豪 |
| 指导教师： | 瞿彬彬 |

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

2023 年 6 月 26 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 1](#_Toc342585681)

[2 任务实施过程与分析 2](#_Toc433882960)

[2.1数据查询(Select)之一 2](#_Toc1563927155)

[2.2数据查询(Select)之二 5](#_Toc1871338452)

[2.3 存储过程与事务 7](#_Toc1687352449)

[2.4 触发器 10](#_Toc1811051708)

[2.5 用户自定义函数 11](#_Toc2060327425)

[2.6 并发控制与事务的隔离级别 12](#_Toc1896707747)

[2.7 数据库设计与实现 14](#_Toc719847761)

[2.8 数据库应用开发(JAVA篇) 16](#_Toc1705935576)

[3 课程总结 21](#_Toc605054735)

# 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。

本实验可以使同学们尽情在实践环境中大展身手，以提升个人能力。

# 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任务书中的2.1~2.13所有关卡、2.14子任务前6关，下面将重点针对其中的2.3，2.4，2.7，2.8，2.9，2.11，2.13，2.14任务阐述其完成过程中的具体工作。

## 2.1数据查询(Select)之一

本节的主要任务是在基于某银行的一个金融场景应用的模拟数据库中，对我们需要实现的查询，用一条sql语句对若干个表数据进行查询操作。主要考察的内容在于select语句的各种用法，包括单表和多表的直接、嵌套、连接查询等。本任务已经完成所有的关卡。

### 2.1.1 基金收益两种方式排名

本关任务: 对客户基金投资收益实现两种方式的排名次，第一条SQL语句实现全局名次不连续的排名，第二条SQL语句实现全局名次连续的排名，不管哪种方式排名，收益相同时,客户编号小的排在前。

实现方法: 由于此时我们还没有学sort函数， 因此想到的实现方式是将两个总收益表连接，连接条件为左表的总收益小于右表的总收益， 然后对于每一个拥有左表对应收益的用户来说，它的排名为总收益小于它连接到的右表元组个数。因此将这样的左右连接的收益表作为排名表之后与资产表连接就可以得到每一个客户的排名。对于名次不连续，我们将右表就设为所有用户的总收益表而不用去重，而对于名词可以连续，我们将右表总收益数值去重distinct即可。以下代码仅展示要求名次不连续的排名情况，如果需要名次连续，在B表的select中加distinct关键字即可。

SELECT D.pro\_c\_id as pro\_c\_id,D.total\_revenue as total\_revenue,C.rk as 'rank'

FROM (

SELECT A.total\_revenue as total\_revenue ,COUNT(B.total\_revenue)+1 as rk

from (

SELECT DISTINCT sum( pro\_income ) as total\_revenue

FROM property

WHERE pro\_type = 3

GROUP BY pro\_c\_id

) as A

LEFT JOIN (

SELECT sum( pro\_income ) as total\_revenue

FROM property

WHERE pro\_type = 3

GROUP BY pro\_c\_id

) as B on A.total\_revenue < B.total\_revenue

GROUP BY A.total\_revenue/\*总收益左右表连接之后，按照左表为码，则右表的个数+1就是排名\*/

)

as C

JOIN (

SELECT sum( pro\_income ) as total\_revenue , pro\_c\_id

FROM property

WHERE pro\_type = 3

GROUP BY pro\_c\_id

) as D /\*与资产表连接之后就知道每一个对应基金投资总收益排名了 \*/

on C.total\_revenue = D.total\_revenue

ORDER BY C.rk ASC,pro\_c\_id;

### 2.1.2 持有完全相同基金组合的客户

本关任务:查询持有完全相同基金组合的客户。

本关实现:由于要找出所有的这样的二元组，并且二元组的前一个的id小于后一个的id,因此，我们只需要将两个client表连接起来，连接条件左表id小于右表id。之后我们对每一个这样的客户组合进行遍历。之后对于这样的客户组合有两种情况不应该出现，有一种情况必须要满足(分别对应于代码中的两个not exists和一个exists)这两种不能出现的情况就是，存在某一个基金左客户选了，但是右客户没有选，或存在某一个基金右客户选了左客户没选。这两个不应该出现的情况其实就是要求对于某一个客户选的基金另外一个客户也必须选。最后必须要满足的情况就是我们在考虑中时不考虑完全不买基金的客户的。因此我们还要求客户至少要买一个基金。

代码实现如下

SELECT C.c\_id as c\_id1, D.c\_id as c\_id2

from client as C JOIN client as D on C.c\_id < D.c\_id

WHERE NOT EXISTS( //左客户买的基金右客户也要买

SELECT \*

from property as A

where NOT EXISTS(

SELECT \*

from property as B

where B.pro\_c\_id = D.c\_id and B.pro\_type=3 and A.pro\_pif\_id = B.pro\_pif\_id

) and A.pro\_c\_id = C.c\_id and A.pro\_type=3

) and NOT EXISTS( //右客户买的基金左客户也要买

SELECT \*

from property as A

where NOT EXISTS(

SELECT \*

from property as B

where B.pro\_c\_id = C.c\_id and B.pro\_type=3 and A.pro\_pif\_id = B.pro\_pif\_id

) and A.pro\_c\_id = D.c\_id and A.pro\_type=3

) and EXISTS( //至少要买一个基金

SELECT 1

from property

where C.c\_id = pro\_c\_id and pro\_type = 3

);

2.1.3 以日历表格式显示每日基金购买总金额  
 本关任务:以日历表格式显示2022年2月每周每日基金购买总金额

本关实现:首先考虑将property表和fund表进行自然连接，之后从该表中挑选出基金的数据并且在2022年2月期间，并且投影出是那一周购买（用week函数),以及总金额sum(pro\_quantity\*f\_amount)。之后由于它要求按照日志表的格式，因此一个比较容易的想法是选择的时候同时选择它是周几的，然后将所有周一与周二。。。周五的顺次连接就可以。按照周次这个只要分组就可以实现。因此我们最后就是连接五个表(分别表示从周一到周五)之后挑选出我们需要的数据和格式就可以实现。

以下代码仅展示了选取周一表和周二表的内容的核心代码，完整代码后面左外连接其他每一周期的内容，并且更改where条件中weekday为对应的值。

SELECT wk-5 as week\_of\_trading,mon.tot as Monday,tue.tot as Tuesday,wed.tot as Wednesday,thu.tot as Thursday,fri.tot as Friday

FROM (

SELECT sum(pro\_quantity\*f\_amount) as tot,week( pro\_purchase\_time) as wk

FROM property JOIN fund on pro\_pif\_id = f\_id

WHERE pro\_type = 3 and weekday(pro\_purchase\_time) = 0 and pro\_purchase\_time BETWEEN '2022-02-07' and '2022-02-28'

GROUP BY pro\_purchase\_time

) as mon /\* 选取周一表,sum为需要的值，该表选取了需要的，之后每周都是一样的类似 \*/

LEFT JOIN

(

SELECT sum(pro\_quantity\*f\_amount) as tot,week( pro\_purchase\_time) wk

FROM property JOIN fund on pro\_pif\_id = f\_id

WHERE pro\_type = 3 and weekday(pro\_purchase\_time) = 1 and pro\_purchase\_time BETWEEN '2022-02-07' and '2022-02-28'

GROUP BY pro\_purchase\_time

) as tue USING(wk)

## 2.2数据查询(Select)之二

本节的主要任务与上一关相同，同样是是在基于某银行的一个金融场景应用的模拟数据库中，对我们需要实现的查询，用一条sql语句对若干个表数据进行查询操作。主要考察的内容在于select语句的其他用法，实现更加复杂更加具体地业务功能。本任务已经完成所有的关卡。

### 2.2.1 查询购买了所有畅销理财产品的客户

本关任务：若定义持有人数超过2的理财产品称为畅销理财产品。查询购买了所有畅销理财产品的客户编号(pro\_c\_id)

本关实现:按照pro\_pif\_id将property分组，挑选出其中pro\_c\_id超过2的，这样就找出了所有的理财产品。现在要求的是对于对于某一个用户，所有挑选出的理财产品他都要购买，也就是全称量词。但是，由于sql中没有全称量词和除法。因此我们可以考虑用双重否定，也就是现在的任务转化为了，不存在说这样的情况，存在某一个畅销商品，对于这个畅销商品存在它没买的情况。也就是(∀x)P 的问题，这样的量词可以进行转化(∀x)P=┐(∃x(┐P))。

SELECT DISTINCT A.pro\_c\_id as pro\_c\_id

from property as A

WHERE NOT EXISTS( //不存在以下的情况

SELECT \*

from(

SELECT pro\_pif\_id

from property

WHERE pro\_type = 1

GROUP BY pro\_pif\_id

HAVING COUNT(pro\_c\_id) > 2

) as B //存在某一个畅销理财产品

WHERE NOT EXISTS(

SELECT \*

from property as C

WHERE A.pro\_c\_id = C.pro\_c\_id and B.pro\_pif\_id = C.pro\_pif\_id and C.pro\_type = 1 //它没买该理财产品

)

)

ORDER BY A.pro\_c\_id;

### 2.2.2查找相似的理财产品

本关任务：查找相似的理财产品。

本关实现: 首先用一个表，将所有持有产品14的用户找出来，并将它们按照持有产品14的数量从大到小排序(代码中A表)，同时定义排名，这样就可以根据排名判断是否前三。之后扫描所有产品，判定它们是不是被这些前三用户持有。这样就挑选出所有被这些前三用户持有的产品。之后将该表(代码中为D表)与全产品表连接，然后按照全产品表id作为主码，连接到了多少D表中元组就表姐相似度，然后按照相似度排名就可以。

**以下为代码实现。**

SELECT D.pro\_pif\_id as pro\_pif\_id,count(\*) as cc,dense\_rank() over(order by count(E.pro\_c\_id) DESC) as prank /\* 与全产品连接，之后D表中连接到了多少的产品，就表示多少与产品14的相似度 \*/

FROM (

SELECT B.pro\_pif\_id as pro\_pif\_id

FROM property as B

WHERE EXISTS(

SELECT \*

FROM (

SELECT pro\_c\_id,dense\_rank() over(order by sum(pro\_quantity) DESC) as rk

FROM property

WHERE pro\_type = 1 and pro\_pif\_id = 14

GROUP BY pro\_c\_id

) as A /\* 持有14产品的用户id按照持有数排序 \*/

WHERE A.rk <= 3 and A.pro\_c\_id = B.pro\_c\_id

) and B.pro\_type = 1 and B.pro\_pif\_id != 14

) as D /\* 扫描所有产品，判定是否被前三用户持有，挑选出所有被这些前三用户持有的产品 \*/

JOIN (

SELECT \*

FROM property

WHERE pro\_type = 1 and pro\_pif\_id != 14

) as E

USING (pro\_pif\_id)

GROUP BY D.pro\_pif\_id

ORDER BY D.pro\_pif\_id;

## 2.3 存储过程与事务

本节子任务主要设计如何使用流程控制语句的存储过程，通过游标实现存储过程和使用存储过程。通过存储过程，可以简化一些常用的重复过程中每次用户和程序员增删查改的步骤。本子任务已经完成所有关卡。

### 2.3.1 使用游标的存储过程

本关任务：使用游标编程存储过程为医院的某科室排夜班值班表。

实现方法:我们针对护士开设一个游标，由于游标不存在周末换班的情况，因此我们只要对于护士的排班情况，就用这一个游标不断地循环遍历就可以实现对护士的轮班操作。对于护士的游标遍历到了尾部，重置该游标。

而对于医生，如果是周末还刚好排到主任的班，那么采取一个懒标记zhuren记录下来，然后继续往后跳转游标进行排班，等到周一了再判断周末是否有zhuren标记，如果有则给主任排班，否则游标继续跳转给医生排班。而对于该游标的操作与护士的游标操作类似，都是一直往后fetch，如果fetch到了尾部则重置游标。

以下是核心代码，去除了开头对于一些基本数据的定义和默认值的设置环节。

DECLARE cur1 cursor for select e\_name from employee WHERE e\_type = 3;//开设护士的游标

DECLARE cur2 cursor for select e\_type, e\_name from employee WHERE e\_type = 2 or e\_type =1;//开设主任和医生的游标

declare continue handler for not found set done = true;

open cur1;

open cur2;

WHILE now\_day <= end\_date DO

FETCH cur1 into nurse1;

if done then

close cur1;

open cur1;

set done = false;

FETCH cur1 into nurse1;

end if;

FETCH cur1 into nurse2;

if done then

close cur1;

open cur1;//当游标到达结尾，则重置游标。

set done = false;

FETCH cur1 into nurse2;

end if;

set wk = weekday(now\_day);

if wk = 0 and zhuren IS NOT NULL then //正好在周一，并且周末有主任的懒标记

set doctor = zhuren;

set zhuren = null;

else

FETCH cur2 into typ,doctor;

if done then

close cur2;

open cur2;

set done = false;

FETCH cur2 into typ,doctor;

end if;

if wk >= 5 and typ = 1 then //如果正好在周末，并且轮转到了主任，添加懒标记并往后轮转一位。

set zhuren = doctor;

FETCH cur2 into typ,doctor;

if done then

close cur2;

open cur2;

set done = false;

FETCH cur2 into typ,doctor;

end if;

END IF;

end if;

insert into night\_shift\_schedule values(now\_day,doctor,nurse1,nurse2);

set now\_day = date\_add(now\_day,INTERVAL 1 day);

end while;

end$$

delimiter ;

## 2.4 触发器

本节子任务主要在sql数据库中实现一个触发器，实现触发器任务，便于实现各种约束和任务自动控制实现。本子任务已完成所有关卡。

### 2.4.1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

本关任务：为资产表property编写一个触发器，以实现任务所要求的完整性业务规则。

本关实现:考虑到它需要实现对表property的多个完整性约束。我们需要做的就是声明一个在insert之前的类型触发器即BEFORE INSERT ON property,之后根据它的pro\_type和pro\_pif\_id判断它是否在其应该对应的理财产品类型中，如果不在则输出不同的报错信息，否则就不进行操作表示一切正常。

以下为代码实现

delimiter $$

CREATE TRIGGER before\_property\_inserted BEFORE INSERT ON property

FOR EACH ROW

BEGIN

declare tp int default new.pro\_type;

declare id int default new.pro\_pif\_id;

declare msg char(128) default null;

if tp = 1 then //根据插入资产表的类型的进行判断，从不同的资产表中查找

if id not in (select p\_id from finances\_product) then

set msg = concat("finances product #", id, " not found!");

end if;

elseif tp = 2 then

if id not in (select i\_id from insurance) then

set msg = concat("insurance #", id, " not found!");

end if;

elseif tp = 3 then

if id not in (select f\_id from fund) then

set msg = concat("fund #", id, " not found!");

end if;

else

set msg = concat("type ", tp, " is illegal!");

end if;

if msg is not null then//如果存在信息，说明不合法，那么发出信息警告

signal sqlstate "45000" set message\_text = msg;

end if;

END$$

delimiter ;

## 2.5 用户自定义函数

**本子任务主要涉及到如何自行定义一个用户自定义的函数，并且实现最终将其运用在我们的sql查询语句和实际应用中。这一部分对于我们实际数据库执行中，简化sql语句和简化查询过程，代码复用很有帮助。**

### 2.5.1 创建函数并在语句中使用它

**本关任务:一个依据客户编号计算其在本金融机构的存储总额的函数,并在SELECT语句使用这个函数。**

**本关实现: 本关中主要需要整我的就是create function，实现基本的标量函数，只要将该函数定义，就可以如内部函数一样的使用出现在各个表达式中，而对于该关来说，需要用函数创建根据客户编号查询在金额机构中的存储总额，这个实际上就是一个简单的select查询，然后返回就可以。之后我们在调用sql的时候直接调用该函数就可以。**

**以下是代码实现，仅展示创建的函数部分。**

delimiter $$

create function get\_deposit(client\_id int)

returns numeric(10,2)

begin

DECLARE deposit numeric(10,2) DEFAULT 0;

select sum(b\_balance) into deposit

from bank\_card

where b\_c\_id = client\_id and b\_type='储蓄卡';

return deposit;

end$$

delimiter ;

## 2.6 并发控制与事务的隔离级别

**本子任务主要涉及到的是数据库中并发控制以及事务隔离性相关的内容，前面四关通过设定在不同的隔离等级下存在不同并发事务不一致性来体验不同隔离等级的不同要求与并发但隔离等级不够可能带来的问题。最后两关考虑如何调度/加锁实现可串行化的正确并发调度，总体而言是提升并发环境下数据库事务执行的能力。子任务中所有关卡均已完成。**

### 2.6.1 幻读

本关任务:在repeatable read事务隔离级别，构造两个事务并发执行时，发生“幻读”现象。

本关实现: 在mysql中，存在四种隔离等级,如图表2.1。除了最高等价的串行化外或多或少都会存在并发问题，但根据不同的应用场景,需要不同程度的隔离，可以采取不同的并行化的隔离策略，如默认的隔离等级是read committed

**表2.1 MySQL事务隔离级别说明表**

|  |  |  |  |
| --- | --- | --- | --- |
| **隔离级别** | **读脏** | **不可重复读** | **幻读** |
| READ UNCOMMITTED | √ | √ | √ |
| READ COMMITTED | × | √ | √ |
| REPEATABLE READ | × | × | √ |
| SERIALIZABLE | × | × | × |

而本关中，需要的就是构造两个幻读。幻读，即出现一个事务在两次读取中，在第二次读取时相较于第一次读取多出来了一些记录或者少了一些记录，在一些数据库中被归类到不可重复读中。

本子任务中线程2已经给出，是在sleep(1)秒后插入一条数据然后提交。因此，我们需要做的就是先查询，然后sleep多秒后等待它插入数据完毕，再一次查询，此时就可以发现已经被多出来的那一条数据。也就是对于事务1来说先读，之后事务2插入记录，事务1再读，此时就发生了幻读现象。

实现代码如下:  
-- 事务1（采用默认的事务隔离级别- repeatable read）:

use testdb1;

select @@transaction\_isolation;

start transaction;

## 第1次查询余票超过300张的航班信息

select \* from ticket where tickets > 300;

set @a=sleep(10);

-- 修改航班MU5111的执飞机型为A330-300：

update ticket set aircraft = 'A330-300' where flight\_no = 'MU5111';

-- 第2次查询余票超过300张的航班信息

select \* from ticket where tickets > 300;

commit;

### 2.6.2 可串行化

本关任务：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。

实现方法: 本关卡中即使实现两个事务，通过自行规划事务的调度实现事务的可串行化。事实上，我们只要做到了事务冲突可穿行化也就实现了事务的串行化调度。而需要实现事务的冲突可串行化，也就是避免读写冲突和写写冲突。这样我们只要将这两个事务规划好各自的时间，就可以实现不用隔离级别实现两个事务的并发执行可串行化。

以下是本关的实现

-- 事务1:

use testdb1;

start transaction;

select tickets from ticket where flight\_no = 'MU2455';

set @n = sleep(2);

select tickets from ticket where flight\_no = 'MU2455';

commit;

-- 事务2:x

use testdb1;

start transaction;

set @n = sleep(1);

update ticket set tickets = tickets - 1 where flight\_no = 'MU2455';

commit;

## 2.7 数据库设计与实现

本小节的任务是熟悉和完成数据库的完整从需求分析到概念模型再到逻辑模型的整个建模过程，同时最终可以学会从零开始完整搭建一个业务需要的数据库。其中要求了解并掌握从概念模型到mysql的实现以及从需求开始最终到逻辑模型的方式以及建模软件的使用等。本子任务所有关卡均已完成。

### 2.7.1 从需求分析到逻辑模型

本关任务：按影院管理系统要求画出E-R图，并给出对应的关系模式。

本关实现:这道题要求我们根据需求分析在不同实体之间建立联系，设计出相应的E-R图，并最终给出相应的关系模式。在仔细分析需求之后，在实体之间有如下几个联系:(1)电影与排场间1:n (2)放映厅与排场1:n (3)电影票与排场1:n (4)电影票和顾客n:1。由此再加上根据需求分析已经得出来的实体条件，我们可以画出以下E-R图，如图2.1

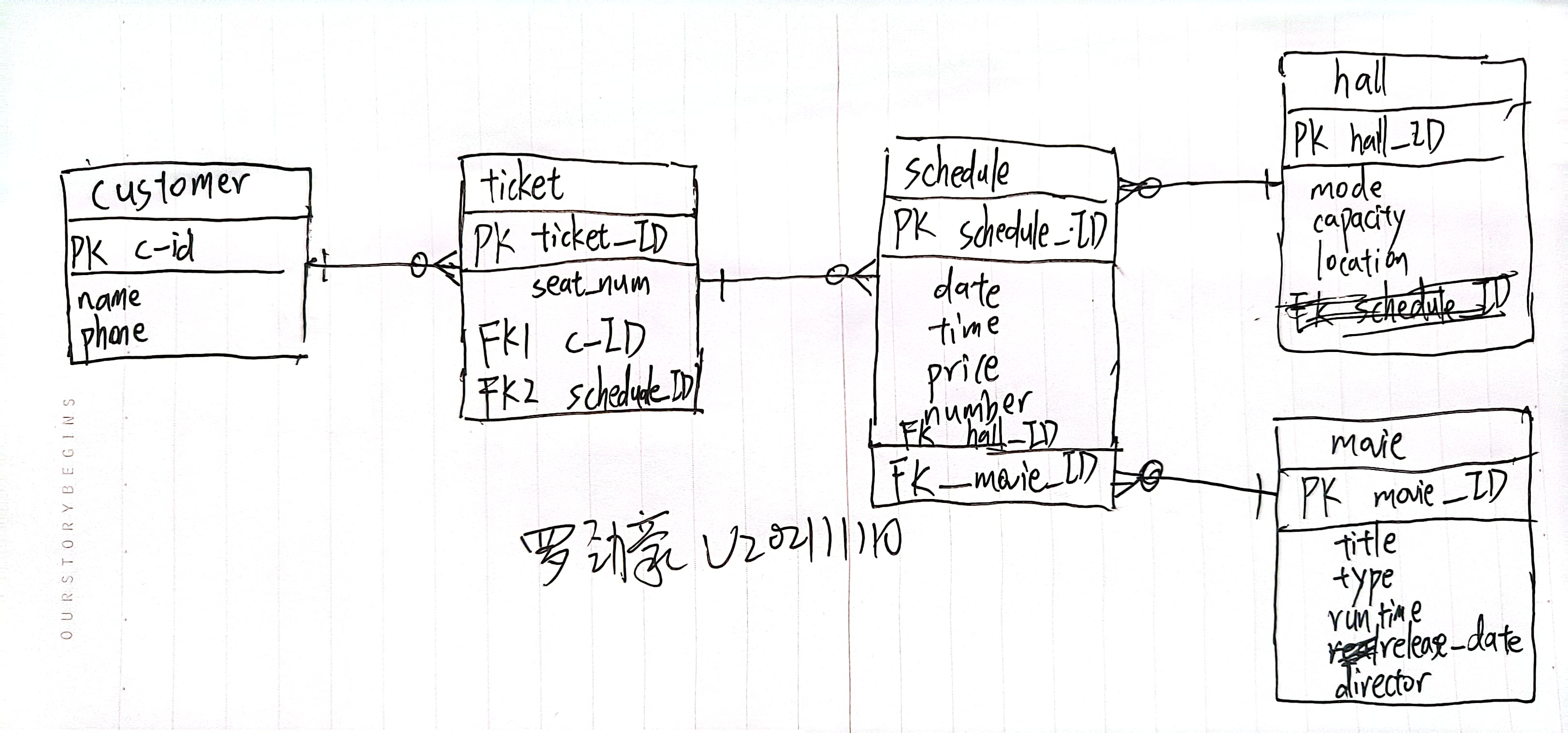


图2.1 影院管理系统E-R图

同时我们根据ER图也可以转化得到以下关系模式

customer(c\_ID, name, type, phone), primary key:(c\_ID);

ticket(ticket\_ID, seat\_num, schedule\_ID), primary key(ticket\_ID), foreign key(schedule\_ID);

schedule(schedule\_ID, date, time, price, number, movie\_ID, hall\_ID), primary key:(schedule\_ID), foreign key(movie\_ID, hall\_ID);

hall(hall\_ID, mode, capacity, location), primary key:(hall\_ID);

movie(movie\_ID, title, type, runtime, release\_date, director, starring), primary key:(movie\_ID);

### 2.7.3 制约因素分析与设计

关于数据库的设计与实现部分，如设计影院管理系统或者金融银行商务系统为例子，首先第一个需要考虑这些设计的部分是否与社会普遍隐私安全认知有关。一个数据库系统尽管可以方便于数据库用户查询设置利用这些数据以实现更好的商务情形，但是实际上，并不是所有的用户数据都应该收集存储，或者部分用户数据属于隐秘隐私，尽管可能数据库逻辑上不需要单独设置一个表，但从社会隐私安全性考虑，可能单独设计一个表并采用加权限设置视图查询会保密。

同时，设计数据库的逻辑、概念模型时也应该考虑到文化道德和法律的制约。应该遵从当地道德良序和法律，对于部分数据如对于老年人和未成年人的数据可以单独分组，加以区分，以避免这些应受保护的人群遭到业务歧视或者攻击，同时可以对它们的购买权加以限制限和同时增加可以查看的数据约束。

### 2.7.4 工程师责任及其分析

在社会层面上，工程师应该能够基于不同的具体相关背景，解决相应的工程问题为社会贡献自己的力量。同时工程师也应当承担工程实现过程及后续维护的责任，理解其中的责任并在之后承担其后果与影响。在如我们在前面根据具体需求，设计了相应的数据库逻辑模式并且将其部署在sql数据库上。从社会责任考虑，之后，如果有新的需求增加，或者有初始设计没有思虑妥当完善的地方，那么工程师应该负责将这些部分补充完善。

在安全方面，而对于数据库的维护，如果出现了安全漏洞，这意味着可能出现触碰社会安全、道德和法律的事情，此时，工程师应当负责因自己原因带来的这些漏洞，修复好并承担因代码或设计不完善带来的后果。

## 2.8 数据库应用开发(JAVA篇)

本子任务要求我们在掌握sql的基础上，实现将数据库应用到JAVA的开发上，学会应用JAVA的JDBC体系架构并且将其实操实现简易的应用场景。本子任务要求包括在JAVA下运用数据库的事务和增删查改的基本操作。本子任务完成关卡1-6，即前六关。

### 2.8.1 客户修改密码

本关任务：实现客户修改密码的方法。

实现方法: 只需要根据题意编写我们的函数就可以，首先判断我们用户输入的原密码与数据库中的对应密码是否相同。在这个过程中，只有我们需要查询的用户是合法的客户，并且password和数据库中的密码一致的时候，我们才会更改为newPass。而根据题意，编写的这个函数已经在主函数判断过两次输入密码一致的问题，故我们只需要查找数据库中用户的合法性和密码的正确性即可。

以下为代码，仅展示了passwd()函数

public static int passwd(Connection connection,String mail,String password,

String newPass){

PreparedStatement pps=null;

ResultSet rs=null;

int n=-1;

try{//以下为首先从数据库中找到用户对应的密码

String sql1 = "select c\_password from client where c\_mail = ?;";

pps = connection.prepareStatement(sql1);

pps.setString(1,mail);

rs=pps.executeQuery();

if(rs.next()) { //判断该用户是否数据库中存在

if(rs.getString("c\_password").equals(password)) {//判断是否正确，然后再更新

String sql2 = "update client set c\_password = ? where c\_mail = ?;";

pps = connection.prepareStatement(sql2);

pps.setString(1,newPass);

pps.setString(2,mail);

n = pps.executeUpdate();

} else {

n = 3;

}} else {

n = 2;

}

}catch (SQLException throwables) {

throwables.printStackTrace();

n = -1;

} finally {

try {

if (pps != null) {

pps.close();

}

if (rs != null) {

rs.close();

}

} catch (SQLException throwables) {

throwables.printStackTrace();

n = -1;

}

}

return n;

}

### 2.8.2 事务与转账操作

本关任务:编写一个银行卡转账的方法。

本关实现:本关属于综合性较高的一关，需要运用JDBC的基本知识(在该子任务中的前几关中学习)，同时，

本关卡中由于需要实现银行卡转账的业务，首先第一步是取消事务的自动提交，否则对于sql中每一个语句只要录入就默认提交。之后我们设置隔离等级--可串行化，这是考虑到银行转账业务的重要性和不容出错性。之后我们设置更新来源卡中的信息，同时确认来源卡信息是否有误或者是否不够更新，如果有误，则及时撤回。之后我们再确认目标卡中信息、类型是否有误，之后根据它是储蓄卡或者是信用卡进行不同的更新操作。在这个过程中，有误也撤回就好。我们发现，由于有事务的存在，这些事务的更新不再需要多重的条件判断，或者麻烦的撤回操作，转而只需要判断失误就可以直接整体回滚，这对与sql事务的需求是十分方便。

下文中代码中只给出转账函数中的核心代码，没有给出函数定义，变量定义和try失败以及finally中的内容，集中关注在事务的实现方面。

try{

while(true){

connection.setAutoCommit(false);

connection.setTransactionIsolation(4);

//首先更新来源卡中信息

String sql = "update bank\_card set b\_balance = b\_balance - ? where b\_number = ?;";

pps = connection.prepareStatement(sql);

pps.setDouble(1,amount);

pps.setString(2,sourceCard);

pps.executeUpdate();

//确认来源卡信息

sql = "select \* from bank\_card where b\_number = ? and b\_type = '储蓄卡';";

pps = connection.prepareStatement(sql);

pps.setString(1,sourceCard);

rs = pps.executeQuery();

if(rs.next()==false) { //判定是否合法

n = false;

connection.rollback();

break;

}

else { //以下都是判定卡源是否合法（出现b\_balance<0则说明不对)

if(rs.getDouble("b\_balance") < 0){

n = false;

connection.rollback();

break;

}

else{

sql = "select \* from bank\_card where b\_number = ?;";

pps = connection.prepareStatement(sql);

pps.setString(1,destCard);

rs = pps.executeQuery();

if(rs.next()==false) {

n = false;

connection.rollback();

break;

}

}

}

//查询目标卡类型 和 目标卡信息

sql = "select b\_type from bank\_card where b\_number = ?;";

pps = connection.prepareStatement(sql);

pps.setString(1,destCard);

rs = pps.executeQuery();

if(rs.next()==false) {

n = false;

connection.rollback();

break;

} else {//根据目标卡的不同类型指定不同的更新策略,即储蓄卡加储蓄，信用卡减欠额

if(rs.getString("b\_type").equals("储蓄卡")) {

sql = "update bank\_card set b\_balance = b\_balance + ? where b\_number = ?;";

pps = connection.prepareStatement(sql);

pps.setDouble(1,amount);

pps.setString(2,destCard);

pps.executeUpdate();

} else if(rs.getString("b\_type").equals("信用卡")) {

sql = "update bank\_card set b\_balance = b\_balance - ? where b\_number = ?;";

pps = connection.prepareStatement(sql);

pps.setDouble(1,amount);

pps.setString(2,destCard);

pps.executeUpdate();

}

}

connection.commit();

break;

}

# 3 课程总结

本次课程实践主要完成了sql数据库的搭建管理，各种约束的建立，对各种需求包括基础的增删查改需求的实现，和触发器、事务和并发、隔离、授权等业务和安全需要。同时完成了从用户需求到概念模型最后到逻辑模型到搭建实施的一整个完整设计实现过程和在JAVA环境下的搭配数据库的应用开发。总体而言，涉及了数据库的理论和实践的大体方方面面。而我完成的情况是除了JAVA的最后一关以及B+树的手写搭建以外所有子任务和所有关卡均全部完成。

具体而言的本实验实践的各个方面的话，1) 实现了创建和管理数据库、表、索引、视图、存储过程、函数、触发器、游标，这些基础的数据对象。这些数据对象都是我们数据需要管理和使用基本构成要素，因此对对它们的掌握，帮助我们正确管理和实现数据库的功能。2) 关于数据处理，主要是对于数据库的增删查改方面。这一方面是数据库的主要功能，即我们需要通过数据库主要实现的就是对数据的处理和查询。这一部分也算是花费时间最多关卡涉及内容最多的部分，值得下来继续好好研究 3）关于数据库的安全性、完整性、恢复机制、并发控制的控制方面，这里是为了保证数据库的安全和完整、正确所必须做的内容，这一部分的内容在搭建时可能体现不出太大用处，但是与后续数据库的管理方面息息相关，十分重要。4)数据库的设计与实现，这一部分内容让我们可以了解数据库的设计和实现两个层面观察数据库的搭建，从而高屋建瓴地看待数据库整个过程。5)对于JAVA篇的数据库应用系统开发，这让我们理解到数据库的sql语句是可以比较容易地部署架构在我们所需要的应用中的，从而理解到数据库在应用开发中的重要作用。

总体而言，参加本次课程实践花费了我不少心血和时间，最后得到的效果也是很好的，很大地帮助我提升了数据库的整体理论和实践双方面能力。其中我觉得我本人有必要改进的方面是可以在写sql语句的同时添加更多的注释，而不是在之后再来看的时候看不懂自己的代码和自己的内容。

最后，真诚地感谢课程组老师们对于该实践课程的精心打造，并且感谢老师对于实验过程中的悉心指导。