



华中科技大学

数据库系统原理实践报告

专 业：	数据科学与大数据技术
班 级：	BD2202
学 号：	U202215641
姓 名：	曲睿
指导教师：	赵小松

分数	
教师签名	

2024 年 7 月 3 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
2 任务实施过程与分析	2
2.1 实训 3 数据查询(SELECT)	2
2.1.1 第 6 关：商品收益的众数	2
2.1.2 第 11 关：黄姓客户持卡数量	3
2.1.3 第 13 关：客户总资产	4
2.1.4 第 15 关：基金收益两种方式排名	5
2.1.5 第 17 关：购买基金的高峰期	6
2.2 实训 4 数据查询(SELECT)新增	8
2.3 实训 7 存储过程和事务	9
2.4 实训 8 触发器	11
2.5 实训 11 并发控制与事务的隔离级别	12
2.5.1 第 1 关：并发控制与事务的隔离级别	12
2.5.2 第 3 关：不可重复读	13
2.5.3 第 5 关：主动加锁保证可重复读	15
2.6 实训 13 数据库设计与实现	15
2.6.1 第 1 关：从概念模型到 MySQL 实现	16
2.6.2 第 2 关：从需求分析到逻辑模型	16
2.6.2 制约因素分析与设计	17
2.6.3 工程师责任及其分析	17
2.7 实训 14 数据库应用开发(JAVA 篇)	17
3 课程总结	19

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，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第 1~14 实训任务，下面将重点针对其中的 10 个具有代表性任务阐述其完成过程中的具体工作。

但请注意：本人实训 1~14 中所有关卡均已完成。本节后续只挑选实训 3、7、8、11、13、14 中的部分任务进行讲解，对于被选中的实训，我会简要介绍“该实训已全部完成”；对于未选中的实训，并不单独列出，但已全部完成，请阅读本报告的老师与同学知晓。

2.1 实训 3 数据查询(Select)

本实训采用的是某银行的一个金融场景应用的模拟数据库，测试库中已有相应测试数据，其主要任务是根据关卡任务需求对若干个表的数据完成相应查询动作，主要考察了 select 语句的诸多用法以及多种函数的功能，在具体的情境中设计了基于单表或多表的直接、嵌套、连接等查询。

本实训已完成 1~19，即全部关卡。

2.1.1 第 6 关：商品收益的众数

本关任务：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。

实现思路：商品收益的众数即出现次数最多的商品收益值，一种思路是先查出众数值，再去资产表中查商品收益与其相等的行。需注意众数可能有多个，因此比较好的做法是在子查询中只取降序排序后的第一条记录的出现次数，在父查询中直接比较出现次数即可，无需生成众数集合。

另一种更加简单的思路是使用 all。若用另一种方式看待众数，即众数的出现次数大于其他的数，则会发现我们已不需要众数和其出现次数的具体值，只需判断该数的出现次数是否大于等于所有数的出现次数。

实现方法：采用第二种思路编写代码。父查询在 property 表中基于 pro_income 分组，使用 count 集函数统计行数，并使用 having 子句对组进行筛选，子查询同样返回各组及其行数，若父查询中该组行数大于等于子查询中所有组的行数，则符合要求，留下该组。“所有组”采用 all 查询实现。代码如下：

```

select pro_income, count(pro_income) as 'presence'
  from property
 group by pro_income
 having count(*) >= all(
    select count(*)
      from property
     group by pro_income
  );

```

注意此处的 all 不能用 max 替换，因为 max 作为集函数，其中不能出现查询子句。本关测试如图 2.1 所示，可以观察到代码可以应对有多个众数的情况，关卡已完成。

测试输入: finance2			
—— 预期输出 ——		—— 实际输出 ——	
pro_income	presence	pro_income	presence
24000	6	24000	6
8000	6	8000	6

图 2.1 实训 3 第 6 关测试

2.1.2 第 11 关：黄姓客户持卡数量

本关任务：给出黄姓用户的编号、名称、办理的银行卡的数量（没有办卡的卡数量计为 0）。按办理银行卡数量降序输出，持卡数量相同的，依客户编号排序。

实现方法：由于需要输出没有办卡的用户，需将 client 表与 bank_card 表进行左外连接，以确保没有办卡的用户仍留在表中。在 where 子句中使用 like 进行字符串匹配，'黄%'表示第一个中文字符为黄，后可接 0 或多个字符。基于 c_id 进行分组，使用 count 集函数对 b_c_id 进行计数，因为 bank_card 中 b_c_id 出现的次数即为银行卡数，若某用户没有办卡，则 b_c_id 中不会出现该用户编号，计数值为 0。代码如下：

```

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

```

本关测试如图 2.2 所示，可以观察到。


测试输入: finance1					
—— 预期输出 ——			—— 实际输出 ——		
c_id	c_name	number_of_cards	c_id	c_name	number_of_cards
400	黄美娟	4	400	黄美娟	4
700	黄建中	4	700	黄建中	4
1500	黄正萍	2	1500	黄正萍	2
2000	黄婉婷	2	2000	黄婉婷	2
2100	黄雨莹	0	2100	黄雨莹	0 

图 2.2 实训 3 第 11 关测试

2.1.3 第 13 关：客户总资产

本关任务：列出所有客户的编号、名称和总资产，总资产为储蓄卡总余额、投资总额、投资总收益的和，再扣除信用卡余额，包括被冻结的资产，命名为 total_property。

实现思路：可以将查询看作三个子查询结果的汇总，即投资总额、储蓄卡、信用卡三部分。一种思路是先计算储蓄卡与信用卡，因为同在一张表中，将计算结果与资产表以及各类产品表相连接。但此处若使用连接，会导致银行卡属性被多次计算，从而产生错误答案，其示意图如表 2-1 所示，显然，用户 A 银行卡项为 300，却因为有多行投资项而多次计算了银行卡项。

表 2-1 实训 3 第 13 关错误思路示意图

姓名	ID	投资	银行卡
A	1	200	300
A	1	300	300
A	1	400	300
B	2	300	1000

另一种思路是使用 union。三个子查询分别将投资总额、储蓄卡、信用卡的组成部分命名为同一个列名，例如 partial_property，使用 union all 进行行合并，则每个用户都只会有三行，将其相加即可，此时不会出现连接查询的冗余问题。

实现方法：将 property 表中投资项首先根据不同的投资类型与对应的三种资产信息表进行自然连接，得到每份理财产品的总投资额（单价乘以购买份额）后再将三种资产表进行集合信息的合并，最后根据 pro_c_id 进行分组统计。最后的连接需要使用左连接，因为存在某人并未购买单个类型的产品。而如果存在某人

未购买任何产品，需要使用 if 语句进行特殊判定，否则会失去其信息。代码如下：

```
select c.c_id, c.c_name, sum(partial_property) as total_property
  from client as c left join (
    select c_id, c_name, sum(ifnull(ifnull(p_amount * pro_quantity, 0) +
ifnull(i_amount * pro_quantity, 0) + ifnull(f_amount * pro_quantity, 0) +
ifnull(pro_income, 0), 0)) as partial_property
      from client
        left join property on c_id = pro_c_id
        left join finances_product on pro_pif_id = p_id and pro_type = 1
        left join insurance on pro_pif_id = i_id and pro_type = 2
        left join fund on pro_pif_id = f_id and pro_type = 3
      group by c_id
    union all
    select c_id, c_name, sum(ifnull(b_balance, 0)) as partial_property
      from client left join bank_card on c_id = b_c_id and b_type = '储蓄卡'
      group by c_id
    union all
    select c_id, c_name, sum(ifnull(-b_balance, 0)) as partial_property
      from client left join bank_card on c_id = b_c_id and b_type = '信用卡'
      group by c_id
  ) as p on c.c_id = p.c_id
group by c.c_id;
```

2.1.4 第 15 关：基金收益两种方式排名

本关任务：查询资产表中客户编号，客户基金总收益，基金投资总收益的排名（从高到低）。总收益相同时名次亦相同。总收益命名为 total_revenue，名次命名为 rank。第一条 SQL 语句实现全局名词不连续的排名，第二条 SQL 语句实现全局名次连续的排名。不管用哪种方式排名，收益相同时，客户编号小的排在前。

实现思路：两次查询都要求同数同名次，分别要求总排名不连续、总排名连续，例如对 300、200、200、150、150、100 排名时，前者结果为 1、2、2、4、4、6，后者结果为 1、2、2、3、3、4。显然，手动实现这样的排名方式较困难，需要记录当前名次、是否与上条记录相等、同名次记录的数目。

查询资料可知，MySQL 内置的排名函数可解决上述问题，rank 函数进行同数同名次、总排名不连续的排名，dense_rank 函数进行同数同名次、总排名连续的排名。二者都可作为查询属性，传入 order by 子句，即可返回排名。

实现方法：将 property 表与 fund 表以 pro_pif_id 与 f_id 相等作为连接条件

进行左外连接,以同时使用用户编号与其投资的基金收益 `pro_income`。使用 `where` 子句筛选保留其中资产类型 `pro_type` 为 3 的行,即资产类型为基金,并按用户编号进行,便可使用 `sum` 集函数计算基金投资总收益,作为查询属性以及 `order by` 子句使用的属性。两次查询的代码如下:

```
-- (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) `rank`
  from property left join fund on pro_pif_id = f_id
 where pro_type = 3
 group by 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) `rank`
  from property left join fund on pro_pif_id = f_id
 where pro_type = 3
 group by pro_c_id;
```

注意此处将排名重命名为 `rank`,需在前后加“`”符号,因为 `rank` 是 MySQL 中的关键字,不能直接用作变量名。本关测试如图 2.3 与 2.4 所示,测试 1 为名次不连续的 SQL 运行结果,测试 2 为名次连续的 SQL 运行结果,可以观察到代码对同名次的全局排名分别有不连续、连续的效果,关卡已完成。

200	35200	11
400	35200	11
1000	29200	13
800	24200	14
1200	24200	14
600	17600	16

图 2.3 实训 3 第 15 关测试 1

200	35200	11
400	35200	11
1000	29200	12
800	24200	13
1200	24200	13
600	17600	14

图 2.4 实训 3 第 15 关测试 2

2.1.5 第 17 关: 购买基金的高峰期

本关任务: 查询 2022 年 2 月购买基金的高峰期, 如果连续三个交易日, 投资者购买基金的总金额超过 100 万, 则称这连续几日为投资者购买基金的高峰期。

实现方法: 判断高峰期的一种思路是将前后日期进行比较, 但显然需要考虑如何保存当前日期, 以及基于何种策略更换日期, 实现起来略有难度。另一种思路是首先将购买金额超过 100 万的天数选出来对交易天数(从 2022 年 1 月 1 日开始计算, 记为 `daycnt`)进行标号, 记为 `rownum`。对于购买高峰期, 该高

峰期内的交易天数和标号必然同时连续，则交易日天数减标号的值必然相同。因而根据这一差值进行分组，如果某一组的值个数大于等于三则可认为是高峰期。

注意到 2022 年 2 月只有第一周因为春节假期而休市，因而统计自 2022 年 1 月 1 日的交易日可以直接通过 MySQL 内置的 datediff 函数统计天数，快速计算出 daycnt 的值。代码如下：

```
select
    t3.t as pro_purchase_time,
    t3.amount as total_amount
from
    (
        select
            *,
            count(*) over (
                partition by t2.workday - t2.rownum
            ) cnt
        from
            (
                select
                    *,
                    row_number() over (
                        order by
                            workday
                    ) rownum
                from
                    (
                        select
                            pro_purchase_time t,
                            sum(pro_quantity * f_amount) amount,
                            @row := datediff(
                                pro_purchase_time,
                                "2021-12-31"
                            ) - 2 * week(pro_purchase_time) workday
                        from
                            property,
                            fund,
                            (
                                select
                                    @row
                                ) a
                    )
                where
```

```

        pro_purchase_time like "2022-02-%"
        and pro_type = 3
        and pro_pif_id = f_id
    group by
        pro_purchase_time
    ) t1
    where
        amount > 1000000
    ) t2
) t3
where
    t3.cnt >= 3;

```

2.2 实训 4 数据查询(Select)新增

本实训同样基于实训 3 的数据，主要任务同样是进行完成查询工作，但在其基础上难度有所提高。

本实训已完成 1~6，即全部关卡。

只选择“第 2 关：投资积极且偏好理财类产品的客户”进行讲解。

本关任务：购买了 3 种（同一编号的理财产品记为一种）以上理财产品的客户被认为投资积极的客户，若该客户持有基金产品种类数（同一基金编号记为相同的基金产品种类）小于其持有的理财产品种类数，则认为该客户为投资积极且偏好理财产品的客户。查询所有此类客户的编号。

实现方法：使用 property 表对理财和基金进行分别的分组去重统计，用 count 集函数统计购买的资产种类，得到两张新表，包含 pro_c_id 和购买该类产品种类数量，对得到的两个新表比较是否基金购买种类小于理财购买种类。最后再利用 where 子句选择购买三种的理财产品的客户。代码如下：

```

select distinct finance_cnt.pro_c_id as pro_c_id from (
    select pro_c_id, if(count(distinct pro_pif_id), count(distinct pro_pif_id), null)
    as finance_product
    from property where property.pro_type = 1 group by property.pro_c_id
) as finance_cnt, ( //理财子表
    select pro_c_id, if(count(distinct pro_pif_id), count(distinct pro_pif_id), null)
    as fund
    from property where property.pro_type = 3 group by property.pro_c_id
) as fund_cnt //基金子表
where finance_cnt.pro_c_id = fund_cnt.pro_c_id and finance_cnt.finance_product >
fund_cnt.fund and finance_cnt.pro_c_id in (
    select pro_c_id from property group by pro_c_id

```

```

        having count(distinct pro_pif_id) >= 3
    ) order by pro_c_id;

```

2.3 实训 7 存储过程和事务

本实训的主要任务是编写存储过程，相当于把需要重复执行的动作函数化，便于调用，三个关卡分别涉及流程控制语句、游标、事务的使用。

本实训已完成 1~3，即全部关卡。

只选择“第 2 关：使用游标的存储过程”进行讲解。

本关任务：医院的某科室有科室主任 1 名（亦为医生），医生若干（至少 2 名，不含主任），护士若干（至少 4 人）。编写一存储过程，自动安排某个连续期间的大夜班的值班表。

实现方法：对于护士，由于其不存在周末换班操作，因此设置一个用于遍历护士的游标，对护士表进行扫描遍历，当遍历到表尾时（done 标记）再重置该游标即可。

对于医生，需要判断当天是周一还是周末，并设置 head 变量，存储因周末而暂时未排班的主任。如果是周末，并且排到主任的班，将 head 标记为主任，然后游标继续遍历；如果当前是周一，并且 head 不为空，表示主任跳班到周一了，需要将主任安排到周一。其余情况与护士相同。需注意，这种思路并不完备，能够使用的原因是 doctor 表中一定不会出现相邻的主任，否则一个 head 变量不足以存储未排班的主任。代码如下：

```

delimiter $$
create procedure sp_night_shift_arrange(in start_date date, in end_date date)
begin
    -- 定义变量
    declare done int default false;
    -- type 记录 doctor 类型，day_of_week 记录当前日期是星期几
    declare type, day_of_week int;
    -- skipped_officer 记录因周末而跳过排班的 officer doctor
    -- 此处利用了数据特点，officer doctor 行后至少跟着两个 doctor，至少可以覆盖整个周末
    -- 保证不会出现在周末同时跳过了两个 officer doctor，因此只需存一个，在周一用掉即可
    declare doctor, nurse1, nurse2, skipped_officer char(30);
    -- 定义游标
    declare nurse_cursor cursor for select e_name from employee where e_type = 3;
    declare doctor_cursor cursor for select e_name, e_type from employee where
e_type < 3;
    -- 定义特情处理程序

```

```

declare continue handler for not found set done = true;

open nurse_cursor;
open doctor_cursor;

while start_date <= end_date do
    -- 护士 1
    fetch nurse_cursor into nurse1;
    if done then
        close nurse_cursor;
        open nurse_cursor;
        set done = false;
        fetch nurse_cursor into nurse1;
    end if;
    -- 护士 2
    fetch nurse_cursor into nurse2;
    if done then
        close nurse_cursor;
        open nurse_cursor;
        set done = false;
        fetch nurse_cursor into nurse2;
    end if;
    -- 医生
    -- weekday 返回 0-6, 对应周一至周日
    set day_of_week = weekday(start_date);
    if day_of_week = 0 and skipped_officer is not null then
        set doctor = skipped_officer;
        set skipped_officer = null;
    else
        fetch doctor_cursor into doctor, type;
        if done then
            close doctor_cursor;
            open doctor_cursor;
            set done = false;
            fetch doctor_cursor into doctor, type;
        end if;
        if day_of_week >= 5 and type = 1 then
            set skipped_officer = doctor;
            fetch doctor_cursor into doctor, type;
            if done then
                close doctor_cursor;
                open doctor_cursor;
                set done = false;
                fetch doctor_cursor into doctor, type;
            end if;
        end if;
    end if;
end while;

```

```

        end if;
    end if;
end if;
insert into night_shift_schedule values (start_date, doctor, nurse1,
nurse2);
-- 以天为单位更新日期
set start_date = date_add(start_date, interval 1 day);
end while;
end$$
delimiter ;

```

2.4 实训 8 触发器

本实训的主要任务是编写触发器，用于实现复杂的完整性约束，已完成其中的一个关卡。

对“第 1 关：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”进行讲解。

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。

实现方法：声明 BEFORE INSERT ON property 类型的触发器，首先判断是否存在该类型的投资产品，然后判断该 p_id 是否为对应类型的投资产品，如果报错信息为空则说明没有错误。

```

delimiter $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW
BEGIN
    declare x int;
    declare msg varchar(50);
    set x = new.pro_type;
    if x = 1 then
        if new.pro_pif_id not in (select p_id from finances_product) then
            set msg = concat("finances product #", new.pro_pif_id, " not found!");
        end if;
    elseif x = 2 then
        if new.pro_pif_id not in (select i_id from insurance) then
            set msg = concat("insurance #", new.pro_pif_id, " not found!");
        end if;
    elseif x = 3 then
        if new.pro_pif_id not in (select f_id from fund) then
            set msg = concat("fund #", new.pro_pif_id, " not found!");
        end if;
    end if;
end if;
end$$

```

```

        end if;
    else
        set msg = concat("type ", x, " is illegal!");
    end if;

    if msg is not null then
        signal sqlstate '45000' SET MESSAGE_TEXT = msg;
    end if;
END$$

delimiter ;

```

2.5 实训 11 并发控制与事务的隔离级别

本实训的主要任务是模拟并发控制中遇到的三种问题，并通过加锁解决。其中 6 个关卡涉及数据库中并发控制与事务的隔离级别的内容，包括隔离级别的设置，事务的开启、提交和回滚等，还通过在合适的位置和时机添加等待代码以实现在隔离级别不够的各种场景下产生读脏、不可重复读、幻读等出错情况。

本实训已完成 1~6，即全部关卡。

2.5.1 第 1 关：并发控制与事务的隔离级别

本关任务：设置事务隔离级别。

实现方法：MySQL 事务隔离级别从高到低为：

- 1) READ UNCOMMITTED（读未提交）
- 2) READ COMMITTED（读已提交）
- 3) REPEATABLE READ（可重复读）
- 4) SERIALIZABLE（可串行化）

各事务隔离级别说明如表 2-2 所示。

表 2-2 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	×	×	×

最低的隔离级别不能避免读脏、不可重复读和幻读，而最高的隔离级别可保

证多个并发事务的任何调度，都不会产生数据的不一致性，但其代价是并发度最低。根据任务要求将事务的隔离级别设置为 `read uncommitted`，并以 `rollback` 语句结束事务。代码如下：

```
use testdb1;
# 设置事务的隔离级别为 read uncommitted
set session transaction isolation level read uncommitted;
-- 开启事务
start transaction;
insert into dept(name) values('运维部');
# 回滚事务:
rollback;
```

2.5.2 第 3 关：不可重复读

本关任务：选择合适的事务隔离级别，构造两个事务并发执行时，发生“不可重复读”现象。

实现方法：根据第 1 关的分析可知，若要发生“不可重复读”，事务隔离级别需设定为 `read committed`。不可重复读指同一事务重复读同一数据，但获得结果不同。原因主要是一事务读取后，另一事务对之进行了修改，其示意如图 2.所示，TA 两次读取 B 结果不同。

表 2-3 不可重复读示意

时间	TA	DB 中值	TB
T1	R(A)=50 R(B)=100 C:=A+B	50(A) 100(B)	
T2		50(A) 200(B)	R(B)=100 W(B)=B*2 COMMIT
T3	R(A)=50 R(B)=200 D:=A+B	50(A) 200(B)	

为模拟不可重复读，需要确保事务 2 的第一次读取在事务 1 修改前发生，事务 2 的第二次读取在事务 1 修改后、提交前发生，在关键处添加 `sleep` 语句即可。代码如下：


```

-- 事务 1:
set session transaction isolation level read committed;

-- 开启事务
start transaction;

-- 时刻 1 - 事务 1 读航班余票:
insert into result
select now(),1 t, tickets from ticket where flight_no = 'CZ5525';

## 添加等待代码, 确保事务 2 的第一次读取在事务 1 修改前发生
set @n = sleep(2);

-- 时刻 3 - 事务 1 修改余票, 并立即读取:
update ticket set tickets = tickets - 1 where flight_no = 'CZ5525';
insert into result
select now(),1 t, tickets from ticket where flight_no = 'CZ5525';

## 添加代码, 使事务 2 的第 2 次读取在事务 1 修改之后, 提交之前发生
set @n = sleep(6);

commit;

-- 时刻 6 - 事务 1 在 t2 也提交后读取余票
## 添加代码, 确保事务 1 在事务 2 提交后读取
set @n = sleep(4);

insert into result
select now(), 1 t, tickets from ticket where flight_no = 'CZ5525';
-- 事务 2
## 请设置适当的事务隔离级别以构造不可重复读
set session transaction isolation level read committed;
start transaction;

-- 时刻 2 - 事务 2 在事务 1 读取余票之后也读取余票
## 添加代码, 确保事务 2 的第 1 次读发生在事务 1 读之后, 修改之前
set @n = sleep(1);

insert into result
select now(),2 t, tickets from ticket where flight_no = 'CZ5525';

-- 时刻 4 - 事务 2 在事务 1 修改余票但未提交前再次读取余票, 事务 2 的两次读取结果应该不同
## 添加代码, 确保事务 2 的读取时机
set @n = sleep(8);
insert into result
select now(), 2 t, tickets from ticket where flight_no = 'CZ5525';

```

```
-- 事务 2 立即修改余票
update ticket set tickets = tickets - 1 where flight_no = 'CZ5525';

-- 时刻 5 - 事务 2 读取余票（自己修改但未交的结果）：
set @n = sleep(1);
insert into result
select now(), 2 t, tickets from ticket where flight_no = 'CZ5525';

commit;
```

2.5.3 第 5 关：主动加锁保证可重复读

本关任务：在事务隔离级别较低的 read uncommitted 情形下，通过主动加锁，保证事务的一致性。

实现方法：由于事务 2 尝试在事务 1 的两次操作之间进行修改，因而当事务 1 加上 X 锁后，事务 2 无法在事务 1 进行过程中打断进行修改，因而保证了事务 1 的可重复读。

```
-- 事务 1:
use testdb1;
set session transaction isolation level read uncommitted;
start transaction;
# 第 1 次查询航班'MU2455'的余票
select tickets from ticket where flight_no = 'MU2455' for update;
set @n = sleep(5);
# 第 2 次查询航班'MU2455'的余票
select tickets from ticket where flight_no = 'MU2455' for share;
commit;
-- 第 3 次查询所有航班的余票，发生在事务 2 提交后
set @n = sleep(1);
select * from ticket;

-- 事务 2:
use testdb1;
set session transaction isolation level read uncommitted;
start transaction;
set @n = sleep(1);
# 在事务 1 的第 1, 2 次查询之间，试图出票 1 张(航班 MU2455):
update ticket set tickets = tickets - 1 where flight_no = 'MU2455';
commit;
```

2.6 实训 13 数据库设计与实现

本实训的主要任务是根据所给概念模型自主设计、实现数据库，涉及根据概

念模型编写 SQL 语句、建立逻辑模型、使用建模工具编写 SQL 语句。

本实训已完成 1~3，即全部关卡。

2.6.1 第 1 关：从概念模型到 MySQL 实现

本关任务：根据一个已建好的逻辑模型，完成 MySQL 的实现。

实现方法：由于本题任务量较大，需建立 7 个表，并为其添加完整性约束，故此处只对编写代码时遇到的重难点进行讲解。

- 1) 用户表的表名 `user` 以及属性 `password` 都是 MySQL 中的关键字，不能直接作为变量名，须在前后添加 “`” 符号；
- 2) 对于主码、不可为空、不可有重、自动增加、缺省值这类简单约束，直接在列定义中说明，而不单独定义表约束；
- 3) 对于外码，定义表约束 `constraint` 以指定参照列，外码不仅包括由题目文字直接说明的，还包括结合实际情况从题目中分析得到的；
- 4) 需对某些属性建立普通索引，此时无法在建表语句中直接建立索引，而应另外编写 `create index` 进行普通索引的构建；
- 5) 在头歌平台多次对同一数据库进行评测时，系统不会自动清空数据，若新建的表已存在，则不会覆盖，因此若需重新建立表，应使用 `drop table if exists` 语句，将此前评测时所建好的表删除，再重新建立。

代码篇幅过长，此处省略。

2.6.2 第 2 关：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。

实现方法：由于题目已给出各表结构与实体间关系的文字描述，根据要求进行模拟即可，E-R 图如图 2.5 所示。

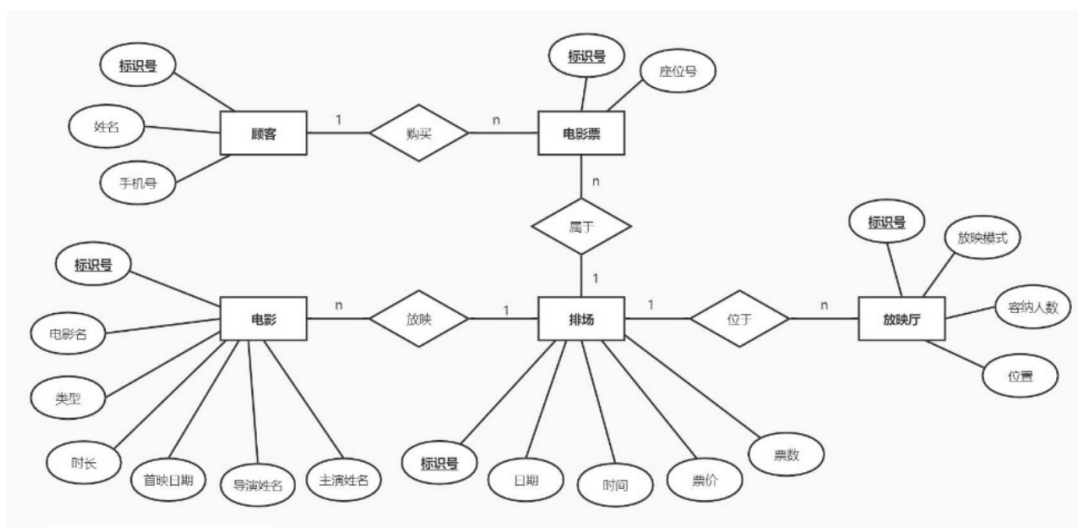


图 2.5 实训 13 第 2 关 E-R 图

2.6.2 制约因素分析与设计

在实际问题中，通常会因为实际情况而产生许多现实制约因素。如本节的机票订票系统，一张机票仅可以让一人乘坐，而由一人代为其购买，因而在购买机票信息不仅需要记录乘机人的信息，还需要记录购票人的信息。对于用户，通常来说根据其购买的机票次数和消费金额，通常会给予不同程度的优惠和购买特权，而后台的管理者相较于上两类用户，权限又有所不同，管理员拥有查看、修改机票信息表以及管理整个数据库系统的权限。因而对于这三类角色对应的同一实体，需要对其加以区分。

2.6.3 工程师责任及其分析

社会方面，工程师应该能够基于工程相关背景知识进行合理分析，评价专业工程实践和复杂工程问题解决方案对社会、健康、安全、法律以及文化等各方面的影响，并理解应承担的责任；安全方面，工程师应该尽可能考虑系统中存在的安全漏洞，防范各类黑客入侵等隐患，以及加强数据库物理存储的安全，因为安全性是所有系统用户关心的重要命题；科学发展方面，工程师应该能够基于科学原理并采用合适的科学方法对复杂工程问题进行研究，包括设计实验、分析与解释数据、并通过信息综合得出合理有效的结论。

2.7 实训 14 数据库应用开发(JAVA 篇)

本实训的主要任务是利用 Java 中的数据库系统实现库 JDBC 进行数据库实现与管理。

本实训已完成 1~7，即全部关卡。

只选择“第 6 关：事务与转账过程”进行讲解。

本关任务：编写一个银行卡转账的方法。

实现方法：难点在于非法输入的判断与事务的手动提交、回滚。非法输入包括转出或转入账号不存在、转出账号是信用卡、转出账号余额不足，实际上，只需一次查询结果即可判断三种非法输入，即查询 `bank_card` 表中转出账号所对应的类型与余额，若 `resultSet` 中无结果，则说明转出账号不存在，否则可直接判断其类型是否为信用卡，余额是否大于转账金额。

在事务执行中，若发生异常，其余事务会停止执行，此时需要将已完成的事务回滚以使数据保持正确，而在 Java 中，事务回滚在 `catch` 子句捕捉到异常时进行。代码篇幅过长，此处省略。

3 课程总结

本次实验主要通过 MySQL 语言实现了数据库的建立、修改、查询、安全性、并发性等实际应用中数据库操作的训练，并额外涉及了数据库设计、数据库底层 B+树的设计与实现、数据库的 Java 应用开发等数据库周边应用，我完成了其中 14 个实训，收获良多。

诚然，实验的数据查询、存储过程、并发控制与 Java 应用开发部分很有难度，我也在编写代码时遇到了很多困难。尽管理论课已涉及 select 语句，但在实际情景中编写 select 语句进行数据查询仍然很困难，从 group by 与 having 子句的含义，到功能多样的函数，我都曾有过疑问，也在同学老师的指导以及网络论坛的讨论下得以解决；由于课内并未涉及存储过程，在实现该实训时，我查阅了许多资料，也花费了许多时间；并发控制与 Java 应用开发是本实验最有特色的两个实训，前者让学生通过亲自设置不同事务的等待和执行时机来实现不同隔离级别下的错误并行处理情况，后者将 JDBC 引入实验，让学生了解数据库如何投入生产应用，使用 Java 进行更高效的数据库管理，而由于本学期我同时选修了 Java 语言程序设计，完成本实训时更得心应手。

本次实验较为系统性的训练，让我熟练掌握了 MySQL 语句的使用，并充分感受到了 MySQL 语句的灵活性和使用便捷性，掌握了如何正确使用 MySQL 语句实现复杂的查询要求。在最后的 Java 实验中也通过实例说明常见的 MySQL 注入和相对应的防御措施。

本次实验的内容涵盖了数据库的绝大多数实际应用，从物理底层到最上层的视图均有涉及，内容详实，衷心地感谢课程组老师们对实验内容的精心打造，也感谢老师对实验过程中的悉心指导。