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：	计科 2106 班
学　　号：	U202115512
姓　　名：	洪炜豪
指导教师：	丁晓峰

分数	
教师签名	

2023 年 6 月 24 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	
6	

总分	

1 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程，注重理论与实践相结合。本课程以 MySQL 为主要编程语言，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、删除与修改等数据修改等相关任务；

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)。

本课程依托头歌实践教学平台，实验环境在 Linux 操作系统下，编程语言主要为 MySQL 8.0.28。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

2.1 数据库、表与完整性约束的定义(Create)

本节主要任务为创建数据库，并且在创建的数据库内创建表，以及对表的规范性和完整性进行一定的约束，如表中主码、外码、 CHECK、 DEFAUT 和 UNIQUE 等约束的建立。

2.1.1　创建数据库

本关任务：创建用于 2022 年北京冬奥会信息系统的数据库:beijing2022。

实现方法：使用如下代码即可完成任务：

```
create database beijing2022;
```

2.1.2　创建表及表的主码约束

本关任务：创建指定数据库 TestDb，并在 TestDb 中创建表 t_emp。

实现方法：首先创建数据库，再创建表并指定一些约束。注意要使用 use 语句使用该表。

```
create database TestDb;
use TestDb;
create table t_emp(
    id int primary key,
    deptId int,
    name varchar(32),
    salary float
);
```

2.1.3　创建外码约束（foreign key）

本关任务：创建 MyDb 数据库，为表定义主键，并给表 staff 创建外键，这个外键约束的名称为 FK_staff_deptNo。

实现方法：使用 primary key 进行主码约束，然后使用 foreign key(column) references (column)实现外码约束。

```
create database MyDb;
use MyDb;
create table dept( deptNo INT primary key,deptName VARCHAR(32));
create table staff
(
    staffNo INT primary key,
    staffName VARCHAR(32),
```

```
    gender CHAR(1),
    dob date,
    salary numeric(8,2),
    deptNo INT,
    CONSTRAINT FK_staff_deptNo FOREIGN KEY(deptNo) REFERENCES dept(deptNo)
);
```

2.1.4 CHECK 约束

本关任务：创建表并对表加上特定约束。

实现方法：根据任务要求，创建表并添加 check 约束，代码如下。

```
create database MyDb;
use MyDb;
create table products(
    pid char(10) primary key, name varchar(32),
    brand char(10) constraint CK_products_brand CHECK(brand in ('A','B')),
    price int constraint CK_products_price CHECK(price > 0)
);
```

2.1.5 DEFAULT 约束

该关卡任务已完成，实施情况本报告略过。

2.1.6 UNIQUE 约束

该关卡任务已完成，实施情况本报告略过。

2.2 表结构与完整性约束的修改（ALTER）

本节主要围绕数据库的修改进行，主要使用 alter 语句对表的数据和约束条件进行增删改。

2.2.1 修改表名

该关卡任务已完成，实施情况本报告略过。

2.2.2 添加与删除字段

本关任务：向表中增加或删除部分数据。

实现方法：依照 ALTER TABLE 表名[修改事项[,修改事项]...]格式。

```
use MyDb;
alter table orderDetail drop column orderDate;
alter table orderDetail add column unitPrice decimal(10,2);
```

2.2.3 修改字段

该关卡任务已完成，实施情况本报告略过。

2.2.4 添加、删除与修改约束

该关卡任务已完成，实施情况本报告略过。

2.3 数据查询(Select)之一

本节的主要任务是对若干个表的数据进行查询操作，主要考察了 select 语句的各种用法，在具体的情境之中设计了单表和多表的直接、嵌套等查询。

2.3.1 金融应用场景介绍,查询客户主要信息

本关任务：查询所有客户的名称、手机号和邮箱信息，查询结果按照客户编号排序。

实现方法：最简单、基础的 select 语句查询。

```
select c_name,c_phone,c_mail from client order by c_id;
```

2.3.2 邮箱为 null 的客户

本关任务:查询客户表(client)中没有填写邮箱信息(邮箱字段值为 null)的客户的编号、名称、身份证号、手机号。

实现方法：基础的 where 语句使用。

```
select c_id,c_name,c_id_card,c_phone from client where c_mail is null;
```

2.3.3 既买了保险又买了基金的客户

本关任务：查询既买了保险又买了基金的客户的名称和邮箱。

实现方法：使用 exists 语句，将是否购买该种类的资产转化为两个判断。

```
select  c_name,c_mail,c_phone
    from client
    where c_id in
    (select s1.pro_c_id
    from property s1,property s2
    where  s1.pro_c_id=s2.pro_c_id and s1.pro_type='2'and s2.pro_type='3')
    order by c_id;
```

2.3.4 办理了储蓄卡的客户信息

本关任务：查询办理了储蓄卡的客户名称、手机号和银行卡号，查询结果结果依客户编号排序。

实现方法：基础多表连接操作。

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

2.3.5 每份金额在 30000～50000 之间的理财产品

该关卡任务已完成，实施情况本报告略过。

2.3.6 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。

实现方法：根据 pro_income 进行分组，使用 count 函数统计数量，再选择数量大等于最大数量的商品收益。

```
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)
```

2.3.7 未购买任何理财产品的武汉居民

本关任务：查询未购买任何理财产品的武汉居民的信息。

实现方法：使用 like 语句通过身份证号“4201%”筛选出武汉居民，然后再使用 not exists 语句统计未购买居民。

```
select c_name,c_phone,c_mail
    from client
    where c_id_card like '4201%'  and
    not exists(select * from property where pro_c_id=client.c_id and pro_type ='1')
    order by c_id;
```

2.3.8 持有两张信用卡的用户

该关卡任务已完成，实施情况本报告略过。

2.3.9 购买了货币型基金的客户信息

本关任务：查询购买了货币型基金的客户信息。

实现方法：使用 in 语句配合 where 语句的条件进行嵌套筛选。

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

2.3.10 投资总收益前三名的客户

本关任务：查询投资总收益前三名的客户。

实现方法：前三名可以通过排序后 limit 语句限制输出条数。统计收益通过 c_id 进行分组后 sum 聚集函数统计。注意只能统计可用资产的收益。

```
select  c_name,c_id_card,sum(pro_income) as total_income
    from client inner join property
    on pro_status='可用' and c_id=pro_c_id  group by c_id
    order by total_income desc
    limit 3;
```

2.3.11 黄姓客户持卡数量

该关卡任务已完成，实施情况本报告略过。

2.3.12 客户理财、保险与基金投资总额

本关任务：查询客户理财、保险、基金投资金额的总和，并排序。

实现方法：先将 client 表与 property 表进行左外连接，然后将 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左外连接，最后根据 client.c_id 进行分组统计。需要使用左外连接：因为可能该人并未购买该类型的产品。total_amount 的求得是使用 case 函数，当投资类型匹配时将其金额计入总额。这里使用 COALESCE 避免出现数值为 null 的情况。

```
SELECT
  client.c_name, client.c_id_card, COALESCE(SUM(case
  when pro_type='1'then COALESCE(pro_quantity*p_amount,0)
  when pro_type='2'then COALESCE(pro_quantity*i_amount,0)
  when pro_type='3'then COALESCE(pro_quantity*f_amount,0)
END),0)   AS total_amount
FROM
  client
  LEFT JOIN property ON client.c_id = property.pro_c_id
  LEFT JOIN finances_product ON property.pro_pif_id = finances_product.p_id and
pro_type='1'
  LEFT JOIN insurance ON property.pro_pif_id = insurance.i_id and pro_type='2'
  LEFT JOIN fund ON property.pro_pif_id = fund.f_id and pro_type='3'
GROUP BY
  client.c_id
ORDER BY
  total_amount DESC ;
```

2.3.13 客户总资产

本题方法与上一关类似，在此略去不写。

2.3.14 第 N 高问题

本关任务：查询每份保险金额第 4 高保险产品的编号和保险金额。

实现方法：排序后使用 limit 子句中的参数（从第几个开始，取多少个）。可以查询出去重后第四高的金额，再直接根据此金额进行筛选。

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

2.3.15 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。

实现方法：MySQL 中内置的两个函数：rank()进行并列名次相同且跳过的标号，dense_rank()进行并列名次相同且标号连续的标号。下仅以 rank()为例。

```
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 total_revenue desc, pro_c_id;
```

2.3.16 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。

实现方法：先按用户分组，再利用 group_concat() 函数将用户所购买的基金编号去重排序后拼接成一个字符串 f_id 。由于需要重复用到上述表，所以利用 with 子句创建一个公用临时表，如果表中两个客户的 f_id 相同则说明这两个客户的基金组合完全相同。

```
with pro(c_id, f_id) as (
    select pro_c_id as c_id, group_concat(distinct pro_pif_id order by pro_pif_id)
as f_id
    from property where pro_type = 3 group by pro_c_id)
select first.c_id as c_id1, second.c_id as c_id2 from pro first, pro second
where first.c_id < second.c_id and first.f_id = second.f_id;
```

2.3.17 购买基金的高峰期

本关未实现。

2.3.18 至少有一张信用卡余额超过 5000 元的客户信用卡总金额

该关卡任务已完成，实施情况本报告略过。

2.3.19 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。

实现方法：对资产表 property 和基金表 fund 做自然连接，并按交易时间分组，计算出购买总金额，再利用 week()函数和 weekday()函数获取当天是当年的第几周和当天是周几。最后按照周次分组，对 weekday()的结果使用 if 语句判断将金额放在哪一列输出即可。

```
select week week_of_trading, sum(if(dayId = 0, amount, null)) Monday, sum(if(dayId
= 1, amount, null)) Tuesday, sum(if(dayId = 2, amount, null)) Wednesday, sum(if(
dayId = 3, amount, null)) Thursday, sum(if(dayId = 4, amount, null)) Friday from (
    select week(pro_purchase_time) - 5 as week, weekday(pro_purchase_time) as
dayId, sum(pro_quantity * f_amount) as amount from property, fund
    where pro_pif_id = f_id and pro_purchase_time like "2022-02-%" and pro_type = 3
    group by pro_purchase_time
) temp group by week;
```

2.4 数据查询(Select)之二

本节内容在 Select 之一的背景和基础之上难度有所提高，仍然关注于数据的查询工作。

2.4.1 查询销售总额前三的理财产品

本关任务：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，例如 1、1、3）的统计年份（pyear）、销售总额排名值(rk)、理财产品编号(p_id)、销售总额(sumamount)。

实现方法：本次选择的目标是查询 2010 年和 2011 年这两年每年销售总额前 3 名的统计年份（pyear）、销售总额排名值(rk)、理财产品编号(p_id)、销售总额(sumamount)。因此，需要先筛选出符合条件的数据，即购买时间在 2010 年或 2011 年的所有理财产品，并且其类型为 1（表示理财产品），然后按照年份和销售总额进行分组，求出每个年份每种理财产品的销售总额。接着，在之前求得的结果上，再按照年份和销售总额排序，求出每个年份销售总额排名前三的理财产品的信息，包括排名值、理财产品编号和销售总额等。最终，通过 where 子句将排名值小于等于 3 的结果选出来，得到所需的查询结果。

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

```
            p_id,
            sumamount
    from
    (
        select
            year(pro_purchase_time) as pyear,
            p_id,
            sum(p_amount * pro_quantity) as sumamount
        from property, finances_product
        where pro_pif_id = p_id and pro_type = 1 and year(pro_purchase_time) in
(2010,2011)
        group by p_id, pyear
    ) as temp1
) as temp2
where rk <= 3
```

2.4.2 投资积极且偏好理财类产品的客户

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(pro_c_id)。

实现方法：首先使用两个子查询分别计算客户持有的理财产品种类数和基金产品种类数，并将结果按 pro_c_id 进行分组，得到两张表 table1 和 table2。然后使用 join 操作将这两张表连接在一起，以便在同一行中比较一个客户持有的理财产品数量和基金产品数量。最后，使用 where 子句过滤出符合条件的记录，即理财产品种类数大于基金产品种类数的客户编号(pro_c_id)。

```
select table1.pro_c_id from
(
  (
    select pro_c_id, count(distinct(pro_pif_id)) as cnt1 from property,
finances_product
    where pro_pif_id = p_id and pro_type = 1
    group by pro_c_id
  ) as table1
  join
  (
    select pro_c_id, count(distinct(pro_pif_id)) as cnt2 from property, fund
    where pro_pif_id = f_id and pro_type = 3
    group by pro_c_id
  ) as table2
```

```
  on table1.pro_c_id = table2.pro_c_id
)
where table1.cnt1 > table2.cnt2;
```

2.4.3 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)。

实现方法：

1 首先，从 property 表中选择所有类型为 1（即理财产品）的记录，然后与一个子查询的结果进行自然连接，这个子查询选出持有人数超过 2 的理财产品，返回这些产品的 pro_pif_id。

2 在自然连接的结果中，按照客户编号 pro_c_id 进行分组统计，得到每个客户购买了多少个畅销理财产品和有多少个畅销理财产品可供购买（即上一步中选出的畅销理财产品）。

3 接着，在第二步的结果中，从所有符合条件的客户记录中选出购买了所有畅销理财产品的客户记录，返回这些客户的编号 pro_c_id。

代码较长，故省略。

2.4.4 查找相似的理财产品

本关未实现。

2.4.5 查询任意两个客户的相同理财产品数

本关任务：查询任意两个客户之间持有的相同理财产品种数，并且结果仅保留相同理财产品数至少 2 种的用户对。

实现方法：使用两次 property 表的自连接，将同一个表看作是两个不同的表，同时通过 pro_pif_id 进行匹配，找到持有相同理财产品的用户对。其中，p1 和 p2 分别代表了两个不同的客户。

在连接过程中，使用 where 子句过滤只有 pro_type 为 1 的记录，保证了筛选出来的都是理财产品。为了避免重复计数，使用 p1.pro_c_id <> p2.pro_c_id 条件来过滤掉自身与自身的组合。

最后，使用 group by 将结果按照 p1 和 p2 进行分类，统计出持有相同理财产品种数，并且使用 having 条件过滤掉持有相同理财产品种数小于 2 种的客户对。最后使用 order by 按照 p1 进行排序输出。

```
select p1.pro_c_id, p2.pro_c_id, count(*) as total_count
from property as p1 inner join property as p2
on p1.pro_pif_id = p2.pro_pif_id
where p1.pro_type = 1 and p2.pro_type = 1
    and p1.pro_c_id <> p2.pro_c_id
group by p1.pro_c_id, p2.pro_c_id
having count(*) >= 2
order by p1.pro_c_id;
```

2.4.6 查找相似的理财客户

本关任务：请用一条 SQL 语句完成以下查询任务： 在某些推荐方法中，需要查找某位客户在理财行为上相似的其他客户，不妨设其定义为：对于 A 客户，其购买的理财产品集合为{P}，另所有买过{P}中至少一款产品的其他客户集合为{B}，则{B}中每位用户购买的{P}中产品的数量为他与 A 客户的相似度值。将{B}中客户按照相似度值降序排列，得到 A 客户的相同相似度值则按照客户编号升序排列，这样取前两位客户即为 A 客户的相似理财客户列表。 查询每位客户(列名：pac)的相似度排名值小于 3 的相似客户(列名：pbc)列表，以及该每位客户和他的每位相似客户的共同持有的理财产品数(列名：common)、相似度排名值(列名：crank)。 注意结果输出要求：要求结果先按照左边客户编号(pac)升序排列，同一个客户的相似客户则按照客户相似度排名值（crank）顺序排列。

实现方法：首先从 property 表中筛选出所有类型为 1（即理财产品）的记录，并将每个客户所持有的理财产品作为一个集合{P}。然后根据这个集合，对于每个客户 A，查询出与他至少持有过{P}中一款产品的其他客户，构成 B 集合。接着，统计出 B 集合中每个客户与 A 客户持有相同理财产品的数量，这个数量就是他们之间的相似度值。最后，对于每个客户 A 及他的相似客户 B，按照相似度值和客户编号进行排序，并计算出相似度排名值，再筛选出排名小于 3 的相似客户。

```
select * from
(select *,rank() over(partition by pac order by common desc,pbc asc) as crank
    from (
        select distinct t1.pro_c_id as pac,
               t2.pro_c_id as pbc,
```

```
            count(t2.pro_c_id) as common
        from property as t1
        join (
            select distinct pro_c_id, pro_pif_id
            from property
            where pro_type = 1
        )as t2
        on t2.pro_pif_id = t1.pro_pif_id
        where pro_type = 1 and t2.pro_c_id <> t1.pro_c_id
        group by t1.pro_c_id, t2.pro_c_id
    ) as t3
) as t4
where crank <3
```

2.5 数据的插入、修改与删除(Insert,Update,Delete)

本节的 6 个关卡围绕数据修改的三种语句 Insert, Update, Delete 语句在不同场景下的应用展开。

2.5.1 插入多条完整的客户信息

该关卡任务已完成，实施情况本报告略过。

2.5.2 插入不完整的客户信息

本关任务：向客户表 client 插入一条数据不全的记录。

实现方法：insert 语句中指明列名和对应信息即可，剩余信息填写 NULL。

```
insert into client(c_id, c_name, c_phone, c_id_card, c_password)
values('33', "蔡依婷", "18820762130", "350972199204227621", "MKwEuc1sc6");
```

2.5.3 批量插入数据

本关任务：已知表 new_client 保存了一批新客户信息，该表与 client 表结构完全相同。将 new_client 表的全部客户信息插入到客户表(client)。

实现方法：insert 语句可以插入一整张表。

```
insert into client select * from new_client;
```

2.5.4 删除没有银行卡的客户信息

本关任务：删除在本行没有银行卡的客户信息。

直接使用 delete 语句和 not exists 子查询即可。

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

2.5.5 冻结客户资产

本关任务：请用一条 update 语句将客户手机号码为 13686431238 的投资资产（理财、保险与基金）的状态置为“冻结”。

实现方法：使用 where 子句选择该客户的信息进行修改。

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

2.5.6 连接更新

本关任务：根据 client 表中提供的身份证号(c_id_card)，填写 property 表中对应的身份证号信息(pro_id_card)。

实现方法：可以使用 set 语句的查询更新实现。为得到身份证号信息可以使用连接查询。

```
update property, client set property.pro_id_card = client.c_id_card
where property.pro_c_id = client.c_id;
```

2.6 视图

本节主要围绕视图的创建与使用展开。

2.6.1 创建所有保险资产的详细记录视图

本关任务：创建所有保险资产的详细记录视图。

实现方法：根据任务要求编写 select 语句，并在前面加上 create view ... as。

```
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;
```

2.6.2 基于视图的查询

本关任务：基于视图 v_insurance_detail 查询每位客户保险资产的总额和保险总收益。

实现方法：将视图当做普通的表一样 select 查询即可。

```
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;
```

2.7 存储过程与事务

本节分别涉及使用流程控制语句的存储过程、使用游标的存储过程和使用事务的存储过程。

2.7.1 使用流程控制语句的存储过程

本关任务：创建一个存储过程，向表 fibonacci 插入斐波拉契数列的前 n 项。

实现方法：使用一条 with 子句编写递归过程，通过此递归过程产生连续的三项 fibonacci 数，通过此语句计算 fibonacci 数列的前 n 项。注意 fibonacci 的边界条件第一项为 1。

```
delimiter $$
create procedure sp_fibonacci(in m int)
begin
    set m = m - 1;
    with recursive cte(id, cur, pre) as (
        select 0, 0, 0
        union all
        select id + 1, if (id < 2, 1, cur + pre), cur from cte where id < m
    )
    select id n, cur fibn from cte;
end $$
delimiter ;
```

2.7.2 使用游标的存储过程

本关未实现。

2.7.3 使用事务的存储过程

本关未实现。

2.8 触发器

本节主要内容为触发器的应用。

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

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 $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property FOR EACH ROW
BEGIN
    declare message varchar(50);
    if (new.pro_type = 1) then
        if (new.pro_pif_id not in (select p_id from finances_product)) then
            set message=concat("finances product #",new.pro_pif_id," not found!");
        end if;
    elseif(new.pro_type = 2) then
        if (new.pro_pif_id not in (select i_id from insurance)) then
            set message = concat("insurance #", new.pro_pif_id, " not found!");
        end if;
    elseif(new.pro_type = 3) then
        if (new.pro_pif_id not in (select f_id from fund)) then
            set message = concat("fund #", new.pro_pif_id, " not found!");
        end if;
    else
        set message = concat("type ", new.pro_type, " is illegal!");
    end if;
    if (message is not null) then
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = message;
    end if;
END$$
delimiter ;
```

2.9 用户自定义函数

本节的主要应用为用户自定义函数的创建和使用。

2.9.1 创建函数并在语句中使用它

本关任务：编写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。

实现方法：使用 create function 语句创建统计给定客户编号，统计存储总额的函数（函数内部使用 select 语句实现），然后再进行调用即可。

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

2.10 安全性控制

2.10.1 用户与角色

本关任务：完成创建用户和授权操作。

实现方法：根据任务要求合理使用 create, grant, revoke 语句编写即可。

```
#(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_name,c_mail,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;
```

2.10.2 用户、角色与权限

本关任务： 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。

实现方法：根据 create 和 grant 语句按要求实现即可。

```
# (1) 创建角色 client_manager 和 fund_manager；
create role client_manager, fund_manager;
# (2) 授予 client_manager 对 client 表拥有 select,insert,update 的权限；
grant select, insert, update on client to client_manager;
# (3) 授予 client_manager 对 bank_card 表拥有查询除银行卡余额外的 select 权限；
grant select(b_c_id, b_number, b_type)
on bank_card to client_manager;
# (4) 授予 fund_manager 对 fund 表的 select,insert,update 权限；
grant select, insert, update on fund to fund_manager;
# (5) 将 client_manager 的权限授予用户 tom 和 jerry；
grant client_manager to tom, jerry;
# (6) 将 fund_manager 权限授予用户 Cindy.
grant fund_manager to Cindy;
```

2.11 并发控制与事务的隔离级别

本节的 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。

2.11.1 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）

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。

```
# 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;
```

2.11.2 读脏

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“读脏”现象。

实现方法：让第一个事务等待第二个事务修改而未 commit 的时候第一个事务进行修改，最后第二个事务回滚，此时第一个事务读脏。

事务 1：

```
-- 时刻 2 - 事务 1 读航班余票,发生在事务 2 修改之后
## 添加等待代码，确保读脏
set @n = sleep(1);
```

```
select tickets from ticket where flight_no = 'CA8213';
```

事务 2：

```
-- 时刻 1 - 事务 2 修改航班余票
update ticket set tickets = tickets - 1 where flight_no = 'CA8213';
-- 时刻 3 - 事务 2 取消本次修改
set @n = sleep(2);
rollback;
```

2.11.3 不可重复读

该关卡任务已完成，实施情况本报告略过。

2.11.4 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。事务 2 的执行代码如下：

```
-- 事务 2（采用默认的事务隔离级别- repeatable read）:
use testdb1;
start transaction;
set @n = sleep(1);
insert into ticket values('MU5111','A330-200',311);
commit;
```

实现方法：事务 2 在等待了 1 秒后进行插入，因而事务 1 如果在一开始读，然后在两秒后再读，此时会因为事务 2 的 commit 而发生幻读。

```
## 第 1 次查询余票超过 300 张的航班信息
select * from ticket where tickets > 300;
set @n = sleep(1);
-- 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;
```

2.11.5 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 update;
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;
```

2.11.6 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。

```
use testdb1;
start transaction;
set @n = sleep(2);
select tickets from ticket where flight_no = 'MU2455';
select tickets from ticket where flight_no = 'MU2455';
commit;

-- 事务 2:
use testdb1;
start transaction;
update ticket set tickets = tickets - 1 where flight_no = 'MU2455';
commit;
```

2.12 备份+日志：介质故障与数据库恢复

2.12.1 备份与恢复

本关任务：设有居民人口登记数据库 residents,请为该数据库做一次静态的(你一个人独享服务器)海量逻辑备份，备份文件命名为 residents_bak.sql。 然后再用

该逻辑备份文件恢复数据库。

实现方法：首先使用 mysqldump 命令将 residents 备份到 residents_bak.sql：

```
mysqldump -h127.0.0.1 -uroot --databases residents > residents_bak.sql
```

再使用该备份恢复回去：

```
mysql -h127.0.0.1 -uroot < residents_bak.sql
```

2.12.2 备份+日志：介质故障的发生与数据库的恢复

本关任务：摸拟介质故障的发生与数据库的恢复。

实现方法：由于需要对数据库 train 作逻辑备份并新开日志文件，所以需要在上一关的基础上 mysqldump 指令中加入--flush-logs 参数来新开日志文件：

```
mysqldump -h127.0.0.1 -uroot --flush-logs --databases train > train_bak.sql;
```

系统故障前的日志文件保存在 log/binlog.000018 中，因而除了 checkpoint 时刻产生的 train_bak.sql 文件中的数据，还需要根据日志文件对部分事务进行重做和撤销操作。由于 mysql 的默认字符集为 utf-8，与日志文件不兼容，因而需要使用--no-defaults 参数来取消 MySQL 默认参数。

```
mysql -h127.0.0.1 -uroot < train_bak.sql;
mysqlbinlog --no-defaults log/binlog.000018 | mysql -uroot;
```

2.13 数据库设计与实现

2.13.1 从概念模型到 MySQL 实现

该关卡任务已完成，实施情况本报告略过。

2.13.2 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：直接根据要求进行模拟即可，如图 2-1。

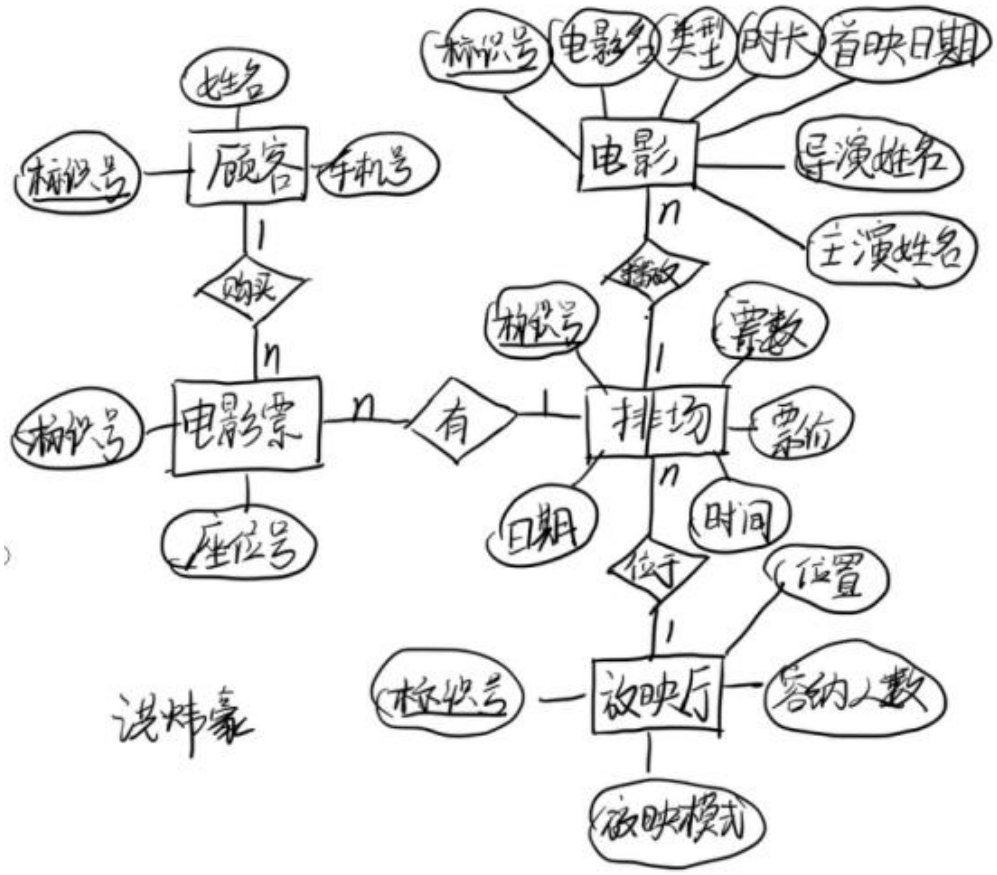

图 2-1 影院管理系统 E-R 图

并有如下的关系模式：

```
movie(movie_ID, title, type, runtime, release_date, director, starring),
primary key:(movie_ID);
customer(c_ID, name, type, phone), primary key:(c_ID);
hall(hall_ID, mode, capacity, location), primary key:(hall_ID);
schedule(schedule_ID, date, time, price, number, movie_ID, hall_ID),
primary key:(schedule_ID), foreign key(movie_ID, hall_ID);
ticket(ticket_ID, seat_num, schedule_ID), primary key(ticket_ID),
foreign key(schedule_ID);
```

2.13.3 建模工具的使用

本关未实现。

2.13.4 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。

2.13.5 工程师责任及其分析

在任务的解决过程中，我们必须考虑各种因素之间的相互影响，并通过合理的设计和实施来确保产品的安全性、健康性和合法性。例如，在涉及到用户隐私的场景中，我们需要采用加密技术和数据保护措施来保护用户的信息；在健康领域的解决方案设计中，我们需要遵守医疗法规和标准，并尽可能地避免虚假宣传和误导性信息。

在解决方案设计中，我们还需要考虑到当地文化差异和社会需要，以确保产品不会引起不必要的争议或对用户造成困扰。在设计中尊重多元文化是非常重要的，同时也需要注意避免使用含有歧视性的语言和符号等。

除此之外，工程师还应该承担一定的社会责任，这包括：

1. 尊重用户的权利和隐私，保护用户的个人信息
2. 遵守相关法律法规，防止违法行为的发生
3. 提供高质量的服务和产品，避免虚假宣传和欺诈行为
4. 关注环境保护和可持续发展，尽可能减少对环境的负面影响
5. 倡导科学合理、诚信守法的态度，避免不正当竞争和商业行为

3 课程总结

1. 数据对象管理与编程

在学习数据对象管理与编程方面，我掌握了 MySQL 中各种数据对象的基本概念、创建方法以及常用命令。我了解了如何使用 SQL 语言对表、索引、视图、约束、存储过程、函数、触发器、游标等数据对象进行管理和编程。通过这些实验，我能够更加熟练地操作 MySQL，对数据库系统的整体架构有了更加深入的了解。

2. 数据修改相关任务

在数据修改相关任务方面，我学习了如何使用 SQL 语言对数据库中的数据进行查询、插入、删除和修改等操作。我还掌握了如何使用事务和锁定机制来保证数据库的并发性和完整性。这些实验让我更好地理解了数据库系统中的数据操作流程和各种优化技术。

3. 数据库系统内核实验

在数据库系统内核实验方面，我重点学习了数据库的安全性控制、完整性控制、恢复机制和并发控制机制等系统内核实验。我学会了如何使用备份和恢复技术来保护数据的安全性，如何使用事务和锁机制来保证数据库的并发性和完整性。这些实验让我了解到了数据库系统内核的工作原理，以及如何在数据库系统中实现数据的高可用性和高可靠性。

4. 数据库的设计与实现

在数据库的设计与实现方面，我学习了如何根据应用需求进行数据库的设计和实现。我需要考虑到数据的结构、关系、业务流程等因素，并根据需求选择适当的数据库模型和规范。通过这些实验，我们更好地了解了数据的组织和存储方式，以及如何为不同的应用场景设计和部署数据库系统。

总之，“数据库系统原理实践”这门课程的实验任务涉及到许多领域，从数据库对象管理到安全性控制和并发控制等系统内核实验，再到数据库的设计和实现等方面。通过这些实验，我对 MySQL 数据库系统有了更深入的理解，能够更好地应用和开发数据库系统。