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：	CS2206
学　　号：	U202215523
姓　　名：	丁丁丁
指导教师：	瞿彬彬

分数	
教师签名	

2024 年 7 月 1 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

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 1~2、部分 3、4~14 实训任务，下面将重点针对其中的 1~14 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

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。我完成了全部 6 关。

2.1.1 创建数据库

任务简单，代码略过。

2.1.2 创建表及表的主码约束

任务简单，代码略过。

2.1.3 创建外码约束

本关任务：创建 MyDb 数据库，为表定义主键，并给表 staff 创建外键，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，用 foreign key 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

任务简单，代码略过。

2.1.5 DEFAULT 约束

任务简单，代码略过。

2.1.6 UNIQUE 约束

任务简单，代码略过。

2.2 表结构与完整性约束的修改

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。我完成了全部的 4 关。

2.2.1 修改表名

任务简单，代码略过。

2.2.2 添加与删除字段

任务简单，代码略过。

2.2.3 修改字段

任务简单，代码略过。

2.2.4 添加、删除与修改约束

本关任务：对表的约束进行修改操作。

实现方法：依照 constraint 的语句格式进行表的修改操作。

```
use MyDb;
alter table Staff add primary key (staffNo);
alter table Dept add constraint FK_Dept_mgrStaffNo foreign
key(mgrStaffNo) references Staff(staffNo);
alter table Staff add constraint FK_Staff_dept foreign key(dept)
references Dept(deptNo);
alter table Staff add constraint CK_Staff_gender check(gender in
('F','M'));
alter table Dept add constraint UN_Dept_deptName unique(deptName);
```

2.3 基于金融应用的数据查询

本节采用的是某银行的一个金融场景应用的模拟数据库，包含 client(客户表)，bank_card(银行卡)，finances_product(理财产品表)，insurance(保险表)，fund(基金表)，property(资产表)。主要任务是对若干个表的数据进行查询操作，考察 select 语句的各种用法，我完成了其中的 1~15 关。

2.3.1 金融应用场景介绍,查询客户主要信息

任务要求：用一条 SQL 语句完成以下查询任务： 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：简单的查询语句结合 order by 语句。

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

2.3.2 邮箱为 null 的客户

任务要求：查询客户表(client)中没有填写邮箱的客户的编号、名称、身份证号、手机号。

实现方法：简单的 null 值判断。

```
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

任务要求：查询既买了保险又买了基金的客户的名称、邮箱和电话,结果依客户编号排序。

实现思路：首先通过资产表筛选出购买了保险和基金的客户编号。然后通过交集运算找出同时满足这两个条件的客户编号。最后从客户表中提取这些客户的名称、邮箱和电话，并按 c_id 排序。

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

2.3.4 办理了储蓄卡的客户信息

任务要求：查询办理了储蓄卡的客户名称、手机号、银行卡号。

实现思路：简单的连接后查询，在此略过。

2.3.5 每份金额在 30000～50000 之间的理财产品

任务要求：查询理财产品中每份金额在 30000～50000 之间的理财产品的编号,每份金额，理财年限，并按金额升序排序，金额相同的按照理财年限降序排序。

实现思路：可以用 between and 或者两次比较实现，注意升序降序排列。代码略过。

2.3.6 第 6 关：商品收益的众数

任务要求：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，出现的次数命名为 presence。

实现思路：通过对 pro_income 进行分组统计，计算每个收益值出现的次数。然后，使用 HAVING 子句筛选出出现次数最多的收益值及其对应的出现次数，从而得到商品收益的众数及其出现的次数。

```
select pro_income,count(*) 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

任务要求：已知身份证前 6 位表示居民地区，其中 4201 开头表示湖北省武汉市。查询身份证隶属武汉市没有买过任何理财产品的客户的名称、电话号、邮箱。依客户编号排序。

实现思路：通过客户表筛选出身份证前四位为“4201”的客户，这表示这些客户的身份证隶属武汉市。然后通过资产表筛选出没有购买理财产品的客户编号（c_id），并排除这些编号。

```
select c_name,c_phone,c_mail from client
where(select left(client.c_id_card, 4)=4201)
and c_id not in(select pro_c_id from property where pro_type=1);
```

2.3.8 持有两张信用卡的用户

任务要求：查询持有两张(含）以上信用卡的用户的名称、身份证号、手机号。查询结果依客户编号排序。

与第六关雷同，在此略过。

2.3.9 购买了货币型基金的客户信息

任务要求：查询购买了货币型(f_type='货币型')基金的用户的名称、电话号、邮箱。依客户编号排序。

实现思路：通过资产表筛选出商品类型为基金的记录，并提取相应的客户编号。再通过基金表筛选出基金类型为货币型的 f_id。将这些基金编号与资产表中 pro_pif_id 进行匹配，找到购买了这些基金的客户编号。最后从客户表中提取这些客户的名称、电话号和邮箱，并按 c_id 排序。

```
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

任务要求：查询当前总的可用资产收益(被冻结的资产除外)前三名的客户的名称、身份证号及其总收益，按收益降序输出，总收益命名为 total_income。

实现思路：通过聚集函数 SUM，在 client,property 的连接表中查询出 pro_status='可用'的元组，按照 c_id 分组。再按收益降序输出，注意使用 limit 取前三个记录。

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

2.3.11 黄姓客户持卡数量

与前几题雷同，在此略过。

2.3.12 客户理财、保险与基金投资总额

任务要求：综合客户表、资产表、理财产品表、保险表和基金表，列出客户的名称、身份证号以及投资总金额（每笔投资金额=商品数量*该产品每份金额），注意投资金额按类型需查询不同的表，投资总金额是客户购买的各类(理财,保险,基金)资产投资金额的总和，总金额命名为 total_amount。结果按总金额降序排序。

实现思路：

1.连接各表：从客户表出发，通过客户编号连接资产表，获取客户的资产信息。根据资产类型（pro_type），分别连接理财产品表、保险表（insurance）和基金表，获取相应产品的每份金额。

2.计算投资金额：使用 CASE 语句，根据资产类型（pro_type）来选择不同的表，计算每笔投资的金额（pro_quantity * 每份金额）。

3. 汇总投资金额：使用 SUM 函数对每个客户的所有投资金额进行汇总，得

到总投资金额（total_amount）。

4. 分组和排序：按 c_name 和 c_id_card 进行分组，以确保每个客户的投资金额被正确汇总。按 total_amount 降序排序。

关键步骤：

使用 LEFT JOIN 连接客户表和资产表，以确保所有客户都被包含，即使他们没有任何资产。

根据资产类型（pro_type）分别连接理财产品表、保险表和基金表，并通过 CASE 语句来计算每笔投资金额。

使用 SUM 函数汇总每个客户的投资金额，并使用 GROUP BY 进行分组。

最后通过 ORDER BY total_amount DESC 按总金额降序排序。

```
SELECT c.c_name, c.c_id_card,
       SUM(
         CASE
           WHEN pro.pro_type = 1 THEN pro.pro_quantity *
fp.p_amount
           WHEN pro.pro_type = 2 THEN pro.pro_quantity * i.i_amount
           WHEN pro.pro_type = 3 THEN pro.pro_quantity * f.f_amount
           ELSE 0
         END
       ) as total_amount
FROM client c
LEFT JOIN property pro ON c.c_id = pro.pro_c_id
LEFT JOIN finances_product fp ON pro.pro_pif_id = fp.p_id AND
pro.pro_type = 1
LEFT JOIN insurance i ON pro.pro_pif_id = i.i_id AND pro.pro_type
= 2
LEFT JOIN fund f ON pro.pro_pif_id = f.f_id AND pro.pro_type = 3
GROUP BY c.c_name, c.c_id_card
ORDER BY total_amount DESC;
```

2.3.13 客户总资产

任务要求：综合客户表、资产表、理财产品表、保险表(insurance)、基金表，列出所有客户的编号、名称和总资产，总资产命名为 total_property。总资产为储蓄卡总余额，投资总额，投资总收益的和，再扣除信用卡透支的总金额(信用卡余额即为透支金额)。客户总资产包括被冻结的资产。

实现思路：

1. 连接各表：客户表，通过客户编号连接银行卡表和资产表（property），

获取客户的银行卡余额和投资信息。根据资产类型，分别连接理财产品表、保险表和基金表，获取相应产品的每份金额。

2. 计算储蓄卡余额：从银行卡表中筛选出储蓄卡（b_type='储蓄卡'）的余额，并对每个客户的储蓄卡余额进行汇总。

3. 计算信用卡透支金额：从 bank_card 中筛选出信用卡（b_type='信用卡'）的余额，并对每个客户的信用卡透支金额进行汇总。

4. 计算投资总额和总收益：从资产表中，由资产类型分别连接理财产品表、保险表和基金表，计算每笔投资的金额（pro_quantity * 每份金额）和总收益。

5. 计算总资产：总资产 = 储蓄卡余额 + 投资总额 + 投资总收益 - 信用卡透支金额。

```
SELECT a.c_id, a.c_name,
   (coalesce(b.b_balance, 0) + coalesce(c.p_amount,0) -
coalesce(d.b_overdraft,0)) as total_property
FROM client a
LEFT JOIN (SELECT
        b_c_id, sum(CASE WHEN b_type='储蓄卡' THEN b_balance
                         ELSE 0 END) as b_balance
          FROM bank_card
          GROUP BY b_c_id) b
ON a.c_id=b.b_c_id
LEFT JOIN
    (SELECT pro_c_id, sum(CASE
       WHEN pro_type = 1 THEN fp.p_amount * pro_quantity
       WHEN pro_type = 2 THEN i.i_amount * pro_quantity
       WHEN pro_type = 3 THEN f.f_amount * pro_quantity
       ELSE 0
       END + pro_income) as p_amount
       FROM property pro LEFT JOIN finances_product fp
       ON pro.pro_pif_id = fp.p_id AND pro.pro_type = 1
       LEFT JOIN insurance i
       ON pro.pro_pif_id = i.i_id AND pro.pro_type = 2
       LEFT JOIN fund f ON pro.pro_pif_id = f.f_id AND pro.pro_type
   = 3
       GROUP BY pro_c_id) c
ON a.c_id = c.pro_c_id
LEFT JOIN (SELECT b_c_id, sum(CASE
                        WHEN b_type='信用卡' THEN b_balance
                        ELSE 0  END) as b_overdraft
            FROM bank_card
```

```
            GROUP BY b_c_id) d
ON a.c_id = d.b_c_id;
```

2.3.14 第 N 高问题

任务要求：查询每份保险金额第 4 高保险产品的编号和保险金额。在数字序列 8000,8000,7000,7000,6000 中，两个 8000 均为第 1 高，两个 7000 均为第 2 高,6000 为第 3 高。

实现思路：注意采用 disinct 关键字，limit3,1 跳过前三个记录取第四个。其他为简单的查询语句。

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

2.3.15 基金收益两种方式排名

任务要求：查询资产表中客户编号，客户基金投资总收益,基金投资总收益的排名(从高到低排名)。总收益相同时名次亦相同(即并列名次)。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。

实现思路：通过一个子查询初始化变量。例如，(SELECT @rank := 0, @current_revenue := 0, @rankcount := 0) AS v 初始化了三个变量 @rank、@current_revenue 和 @rankcount。可以在 SELECT 语句中使用这些变量，并在每一行处理时更新它们。例如：@rankcount := @rankcount + 1

计算基金投资总收益：

从资产表中筛选出基金（pro_type=3）的记录，并按客户编号进行分组，计算每个客户的基金投资总收益（total_revenue）。

实现名次不连续的排名：

使用变量 @rankcount、@rank 和 @current_revenue，通过 IF 函数判断当前总收益是否与上一行相同，相同则名次不变，不同则更新名次。总收益相同时名次并列且名次不连续。

实现名次连续的排名：

使用变量 @rank 和 @current_revenue，通过 IF 函数判断当前总收益是否与上一行相同，相同则名次不变，不同则名次递增。总收益相同时名次并列但名次连续。

(1)名次不连续

```
SELECT pro_c_id, total_revenue, trank AS 'rank'
FROM (
    SELECT pro_c_id, total_revenue,
           @rankcount := @rankcount + 1 AS rankcount,
           IF(@current_revenue = total_revenue, @rank, @rank :=
@rankcount) AS trank,
           @current_revenue := total_revenue
    FROM (
        SELECT pro_c_id, SUM(pro_income) AS total_revenue
        FROM property
        WHERE pro_type = 3
        GROUP BY pro_c_id
        ORDER BY total_revenue DESC, pro_c_id
    ) AS i,
    (SELECT @rank := 0, @current_revenue := 0, @rankcount := 0) AS
v
) AS t;
```

(2)名次连续

```
SELECT pro_c_id, total_revenue, trank AS 'rank'
FROM (
    SELECT pro_c_id, total_revenue,
           IF(@current_revenue = total_revenue, @rank, @rank :=
@rank + 1) AS trank,
           @current_revenue := total_revenue
    FROM (
        SELECT pro_c_id, SUM(pro_income) AS total_revenue
        FROM property
        WHERE pro_type = 3
        GROUP BY pro_c_id
        ORDER BY total_revenue DESC, pro_c_id
    ) AS i,
    (SELECT @rank := 0, @current_revenue := 0) AS v
) AS t;
```

2.4 数据查询(Select)之二

本节实验依旧参考 2.3 的关系模型，拓展了更多书本上未曾学习到的查询方法。我完成了其中 5 关。

2.4.1 查询销售总额前三的理财产品

任务要求：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)。

(1)按照年份升序排列，同一年份按照销售总额的排名值升序排列，如遇到并列排名则按照理财产品编号升序排列;

(2)属性显示：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)；

(3)结果显示顺序：先按照统计年份升序排，同一年份按照销售总额排名值（rk）升序排，同一排名值的按照理财产品编号（p_id）升序排。

实现思路：对理财产品按年份和产品 ID 进行分组，计算每个理财产品每年的销售总额。使用 RANK() 函数按年份分组对销售总额进行排名。最后筛选。

WITH 子句 (WITH RankedProducts AS (...)): 用于定义一个临时结果集，该结果集包括每个理财产品每年的销售总额和对应的排名。

RANK() 函数：在 OVER 子句中使用，按年份分组 (PARTITION BY YEAR(pro_purchase_time)) 并按销售总额降序排列 (ORDER BY SUM(p_amount * pro_quantity) DESC)，为每个理财产品分配一个销售总额排名。

```
WITH RankedProducts AS (
    SELECT
        YEAR(pro_purchase_time) AS pyear,
        pro_pif_id AS p_id,
        SUM(p_amount * pro_quantity) AS sumamount,
        RANK() OVER(PARTITION BY YEAR(pro_purchase_time) ORDER BY
SUM(p_amount * pro_quantity) DESC) AS rk
    FROM property
    JOIN finances_product ON property.pro_pif_id =
finances_product.p_id
    WHERE pro_type = 1 AND YEAR(pro_purchase_time) IN ('2010',
'2011')
    GROUP BY YEAR(pro_purchase_time), pro_pif_id
```

```
)
SELECT pyear,rk,p_id,sumamount
FROM RankedProducts
WHERE rk <= 3
ORDER BY pyear ASC, rk ASC, p_id ASC;
```

2.4.2 投资积极且偏好理财类产品的客户

任务要求：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)。按照客户编号的升序排列，且去除重复结果。

实现思路：

内部子查询 (SELECT DISTINCT pro_c_id, pro_type, pro_pif_id FROM property WHERE pro_type IN (1, 3)): 首先筛选出所有购买了理财产品或基金产品的记录，并使用 DISTINCT 确保每种产品只计数一次。

条件聚合 (SUM(CASE WHEN pro_type = 1 THEN 1 ELSE 0 END) AS finances_num 和 SUM(CASE WHEN pro_type = 3 THEN 1 ELSE 0 END) AS fund_num): 对每个客户，计算其购买的理财产品种类数和基金产品种类数。

筛选条件 (HAVING finances_num >= 3 AND finances_num > fund_num): 选择满足条件的客户，即购买了 3 种及以上理财产品且理财产品种类数大于基金产品种类数的客户。

结果去重和排序 (ORDER BY pro_c_id ASC): 对结果进行排序，由于我们在最外层查询中直接选择了客户编号，所以结果自然是去重的。

```
SELECT pro_c_id
FROM (
    SELECT pro_c_id,
           SUM(CASE WHEN pro_type = 1 THEN 1 ELSE 0 END) AS
finances_num,
           SUM(CASE WHEN pro_type = 3 THEN 1 ELSE 0 END) AS fund_num
    FROM (
        SELECT DISTINCT pro_c_id, pro_type, pro_pif_id
        FROM property
        WHERE pro_type IN (1, 3)
    ) AS distinct_products
    GROUP BY pro_c_id
```

```
    HAVING finances_num >= 3 AND finances_num > fund_num
) AS investment_preference
ORDER BY pro_c_id ASC;
```

2.4.3 查询购买了所有畅销理财产品的客户

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)。结果输出要求：按照客户编号的升序排列，且去除重复结果。

实现思路：

识别畅销理财产品：从资产表中筛选出理财产品（pro_type = 1），按理财产品编号（pro_pif_id）分组，并计算每种理财产品的购买次数，筛选出购买次数超过 2 次的理财产品作为畅销理财产品。

查询购买了所有畅销理财产品的客户：对每个客户，检查是否存在他们未购买的畅销理财产品。使用 NOT EXISTS 结合子查询来实现这一点。如果不存在这样的畅销理财产品（即客户购买了所有畅销理财产品），则选择这些客户。

结果去重：使用 DISTINCT 确保结果中客户编号（pro_c_id）的唯一性。

```
SELECT DISTINCT p1.pro_c_id
FROM property p1
WHERE NOT EXISTS (
    SELECT *
    FROM (
        SELECT pro_pif_id
        FROM property
        WHERE pro_type = 1
        GROUP BY pro_pif_id
        HAVING COUNT(DISTINCT pro_c_id) > 2
    ) AS bestselling
    WHERE NOT EXISTS (
        SELECT *
        FROM property p2
        WHERE p2.pro_c_id = p1.pro_c_id
        AND p2.pro_type = 1
        AND p2.pro_pif_id = bestselling.pro_pif_id));
```

2.4.4 查询任意两个客户的相同理财产品数

任务要求：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。结果输出要求：第一列和第二列输出客户编号(pro_c_id,pro_c_id)，第三列输出他们持有的相同理财产品数(total_count)，

按照第一列的客户编号的升序排列。

实现思路：

自连接资产表：首先，对资产表（property）进行自连接操作，以比较不同客户（p1.pro_c_id <> p2.pro_c_id）持有的理财产品（pro_type = 1）。连接的条件是两个客户持有相同的理财产品编号（p1.pro_pif_id = p2.pro_pif_id）。

筛选理财产品：在自连接过程中，只考虑理财产品（pro_type = 1），确保比较的是客户持有的理财产品。

计算相同理财产品种数：通过分组（GROUP BY pid_1, pid_2）统计每一对客户之间共同持有的理财产品数（COUNT(*)）。使用 HAVING 子句筛选出共同持有的理财产品数至少为 2 种的客户对（HAVING COUNT(*) >= 2）。

```
SELECT pid_1 AS pro_c_id, pid_2 AS pro_c_id, COUNT(*) AS
total_count
FROM
    (SELECT p1.pro_c_id AS pid_1, p2.pro_c_id AS pid_2,
p1.pro_pif_id
    FROM property p1, property p2
    WHERE p1.pro_type = 1 AND
        p2.pro_type = 1 AND
        p1.pro_c_id <> p2.pro_c_id AND
        p1.pro_pif_id = p2.pro_pif_id) same_fin
GROUP BY pid_1,pid_2
HAVING COUNT(*) >= 2
ORDER BY pid_1;
```

2.4.5 查找相似的理财客户

任务要求：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)。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。

实现思路：内层子查询（common_c）：通过自连接 property 表，比较 p1.pro_c_id 和 p2.pro_c_id 之间持有的理财产品（pro_type = 1）。通过筛选 p2.pro_pif_id 在 p1.pro_c_id 持有的理财产品 ID 集合中，我们可以找出每对客户之间共有的理财产品。使用 DISTINCT 确保每对客户和理财产品 ID 的组合是唯一的，防止重复计数。

次内层子查询（common_b）：对上一步的结果按客户对（pac, pbc）分组，并计算每对客户之间共有的理财产品数（common）。这一步骤得到了每对客户之间共有理财产品的总数。

外层子查询（common_a）：使用 rank()窗口函数，按 pac 分组，并根据共有理财产品数（common）和另一客户编号（pbc）进行排序，为每个客户对应的其他客户分配一个排名（crank）。这个排名按共有理财产品数降序排列，当共有理财产品数相同，按 pbc 升序排列。

最外层查询：从上一步的结果中筛选出每个客户（pac）对应的排名前两（crank <= 2）的记录。这样，每个客户最多与两个其他客户的信息被保留，这两个客户是与该客户共有理财产品数最多的。

代码从略，参考代码包。

2.5 数据的插入、修改与删除

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。我完成了全部的 6 关。

2.5.1 插入多条完整的客户信息

任务要求：向客户表插入以下 3 条数据。

采用最基础的 insert into…values 语句，代码从略。

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。与 2.5.1 类似，从略。

2.5.3 批量插入数据

任务要求：向客户表 client 批量插入数据。

实现思路：采用 insert into 后接基本 select 语句。

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

2.5.4 删除没有银行卡的客户信息

任务要求：删除在本行没有银行卡的客户信息。

采用最基本的 delete 语句，代码从略。

2.5.5 冻结客户资产

任务要求：用一条 update 语句将手机号码为“13686431238”这位客户的投

资资产(理财、保险与基金)的状态置为“冻结”。

实现思路：基本的 update 语句。

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

2.5.6 连接更新

任务要求：用一条 update 语句，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)。

采用最基础的 update 语句，代码略过。

2.6 视图

本节围绕视图的基本操作展开，我完成了全部 2 关。

2.6.1 创建所有保险资产的详细记录视图

任务要求：创建包含所有保险资产记录的详细信息的视图 v_insurance_detail，包括购买客户的名称、客户的身份证号、保险名称、保障项目、商品状态、商品数量、保险金额、保险年限、商品收益和购买时间。

实现思路：基本的视图创建语句，代码从略。

2.6.2 基于视图的查询

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。

实现思路：基本的试视图查询。把视图看作表进行 select。代码从略。

2.7 存储过程与事务

本节包含流程控制语句的入门，游标的使用，事务的存储过程。我完成了全部的 3 关。

2.7.1 使用流程控制语句的存储过程

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

实现思路：定义局部变量，包括循环计数器 i，斐波那契数列的两个连续项 a 和 b。初始时，a 设置为 0，b 设置为 1，这对应斐波那契数列的前两项。向 fibonacci 表插入初始值，即数列的第 0 项，值为 0。这是斐波那契数列的起始值。

使用 WHILE 循环，循环条件为 i < m，确保插入的斐波那契数项不超过用户

指定的 m 项。在每次循环中，计算下一个数（next = a + b），将当前项（b）插入到 fibonacci 表中，然后更新 a 和 b 为下一对连续的斐波那契数，最后更新循环计数器 i。当 i 达到用户指定的 m 时，循环结束。

```
drop procedure if exists sp_fibonacci;
delimiter $$
create procedure sp_fibonacci(in m int)
begin
DECLARE i INT DEFAULT 1;
    DECLARE a INT DEFAULT 0;
    DECLARE b INT DEFAULT 1;
    DECLARE next INT;
insert into fibonacci values(0,0);
while i<m do
    set next=a+b;
    insert into fibonacci value(i,b);
    set a=b;
    set b=next;
    set i=i+1;
end while;
end $$
delimiter ;
```

2.7.2 使用游标的存储过程

任务要求：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)，现在需要编写一存储过程，自动安排某个连续期间的大夜班(即每天 00:00-8:00 时间段)的值班表，排班规则为：

1.每个夜班安排 1 名医生，2 名护士；

2.值班顺序依工号顺序循环轮流安排(即排至最后 1 名后再从第 1 名接着排)；

3.科室主任参与轮值夜班，但不安排周末(星期六和星期天)的夜班，当周末轮至科主任时，主任的夜班调至周一，由排在主任后面的医生依次递补值周末的夜班。存储过程的两个输入参数：start_date, end_date，分别指排班的起始时间和结束时间。排班结果直接写入表 night_shift_schedule。

实现思路：

1. 初始化变量和游标：定义所需的局部变量，包括用于标记循环结束的 done，用于标记科室主任周末值班调整的 iswait 和 waitdr，以及用于迭代日期的 currentdate。定义两个游标，drc 用于遍历医生（包括科室主任）的信息，nrc 用

于遍历护士的信息。

2. 处理科室主任的周末值班调整：在循环内部，根据当前日期判断是否为周末，如果是周末且当前医生为科室主任（drtype = 1），则将科室主任的值班调整到下一个非周末日，同时设置 iswait 标记以便后续处理。

3. 循环遍历日期：从 start_date 开始，每次循环对应一天，直到 end_date。对每天，根据上述规则选择一名医生和两名护士进行排班。

4. 选择医生和护士：使用游标 drc 和 nrc 分别选择医生和护士。如果遍历完所有医生或护士，则重新打开游标继续选择，确保值班顺序循环轮流。

5. 插入排班结果：将每天的排班结果（包括日期、医生和两名护士的工号）插入到 night_shift_schedule 表中。在循环结束后，关闭所有打开的游标。

代码过于冗长，不在此展示，参见代码包。

2.7.3 使用事务的存储过程

本关任务：编写实现转账功能的存储过程。

实现方法：根据题意编写判断条件，如果不合法则将返回值置为 0，并直接利用 leave 退出事务。如果合法，则先判断付款方和收款方的卡类型，如果是信用卡则要把转账金额设置为负数，最后使用 update 更新即可。

代码过于冗长，不在此展示，参见代码包。

2.8 触发器

2.8.1 为投资表实现业务约束规则-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。

```
use finance1;
drop trigger if exists before_property_inserted;
delimiter $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW
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 ;
```

2.9 用户自定义函数

2.9.1 创建函数并在语句中使用它

任务要求：

(1) 用 create function 语句创建符合以下要求的函数：

依据客户编号计算其所有储蓄卡余额的总和。函数名为：get_deposit

(2) 利用创建的函数，仅用一条 SQL 语句查询存款总额在 100 万(含)以上的客户身份证号，姓名和存款总额(total_deposit)，结果依存储总额从高到低排序。

实现思路：函数 get_deposit 接收一个客户编号（client_id）作为参数。使用 SELECT SUM(b_balance)计算指定客户编号的所有储蓄卡余额总和。b_c_id = client_id 确保只计算特定客户的储蓄卡，且 b_type = "储蓄卡"确保只考虑储蓄卡的余额。将查询到的总和赋值给局部变量 total_dep，并通过 RETURN 语句返回这个总和。

利用创建的函数 get_deposit，在 SELECT 语句中对 client 表中的每个客户调用此函数，计算其存款总额。筛选出存款总额 100 万以上的客户。通过 ORDER BY total_deposit DESC 对结果按存款总额从高到低进行排序。最后从 client 表中选择客户的身份证号、姓名和使用存款总额。

```
delimiter $$
create function get_deposit(client_id int)
returns numeric(10,2)
begin
    DECLARE total_dep INT;
    SELECT SUM(b_balance) INTO total_dep
    FROM bank_card
    WHERE b_c_id = client_id AND b_type = "储蓄卡";
    RETURN total_dep;
end$$

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 用户和权限

任务要求：创建用户，并给用户授予指定的权限。

实现思路：这关主要熟悉基本的 grant，revoke 语句。代码略过。

2.10.1 用户、角色与权限

任务要求：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。

实现思路：创建角色，把角色授予对应用户。代码略过。

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。我完成了所有的 6 关。

并发操作可能带来的数据不一致性包括：丢失修改(lost update)；读脏数据(dirty read)；不可重复读(non-repeatable read)；幻读(phantom read)。为解决上述不一致性问题，DBMS 设计了专门的并发控制子系统，采用封锁机制进行并发控制，以保证事务的隔离性和一致性。

不同的 DBMS，其事务的隔离级别划分是不同的。MySQL 的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）

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。

2.11.1 并发控制与事务的隔离级别

任务要求：在代码文件里根据提示补充适当的代码，将事务的隔离级别设置为 read uncommitted; 并以 rollback 语句结束事务。

```
use testdb1;
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

任务要求：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。

实现思路：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。

```
use testdb1;
## 请设置适当的事务隔离级别
set session transaction isolation level ;
start transaction;
-- 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(2);
select tickets from ticket where flight_no = 'CA8213';
commit;
```

2.11.3 不可重复读

任务要求：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。包含两个文件，构造方法与上一题类似，代码略过。

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。

任务要求：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。

实现方法：让事务 2 等待 1 秒。代码略过，参见代码包。

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。由于有多个文件，代码略过，参见代码包。

2.11.6 可串行化

任务要求：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。代码见代码包。

2.12 备份+日志：介质故障与数据库恢复

2.12.1 备份与恢复

设有居民人口登记数据库 residents,为该数据库做一次静态的海量逻辑备份，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。备份和恢复的命令分别写在 test1_1.sh 和 test1_2.sh 文件中。

实现方法：首先使用 mysqldump 命令将 residents 备份到 residents_bak.sql 中，再使用该备份恢复回去。

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

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

2.12.2 介质故障的发生与数据库的恢复

任务：摸拟介质故障的发生与数据库的恢复。依时间顺序发生下列事件(其

中 2 和 5 的部分由我来完成,其余部分由评测程序完成):

时间点 1：数据库 train 有业务数据产生； 时间点 2：对数据库 train 作一次海量备份，同时新开日志； 时间点 3：数据库 train 又有业务数据产生； 时间点 4：故障发生； 时间点 5：恢复数据库，保证两次发生的业务数据都不丢失。

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

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

2.13 数据任务库设计与实现

本节给定一个应用背景，要求从概念模型完成对各个实体的关系模型的构建，绘制 ER 图，完成建表。

2.13.1 从概念模型到 MySQL 实现

题目提供了一个机票订票系统，代码已经评测通过，在此略过，参见代码包。

2.13.2 从需求分析到逻辑模型

本关任务： 根据应用场景业务需求描述，完成 ER 图，并转换成关系模式。

ER 图文件存放的 URL：ER 图 ddd

以下给出关系模式：

```
movie(movie_ID, title, type, runtime, release_date, director, starring), primary key(movie_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, movie_ID, hall_ID), primary key(schedule_ID), foreign key(movie_ID, hall_ID)
ticket(ticket_ID, seat_num, c_ID, schedule_ID), primary key(ticket_ID), foreign key(c_ID, schedule_ID)
```

2.13.3 建模工具的使用

测试通过，在此略过

2.13.4 制约因素分析与设计

机票订票系统要符合实际，在现实生活中受到很多因素制约，因此我们在构建数据库的各个表时，也要加上很多限制条件，使得这个系统符合现实。例如，用户可以多次订票，旅客可以多次乘坐飞机。一张机票肯定是某个用户为某个特

定的旅客购买的特定航班的机票。即机票信息不仅跟乘坐人有关，同时记录购买人信息(虽然两者有时是同一人)。

对于不同用户，在该系统中的权限也不同。普通用户和管理员的权限涉及到数据，信息的安全性问题。

2.13.5 工程师责任及其分析

在设计和实现机票订票系统的过程中，工程师面临的挑战远远超出了技术实现本身。系统的设计和功能实现必须综合考虑其对社会、健康、安全、法律以及文化等多个方面的影响。这不仅要求工程师具备高超的技术能力，还要求他们具有深刻的社会责任感和道德观念。

系统的设计必须确保所有用户，包括有特殊需求的用户，都能公平、便捷地访问和使用。这包括但不限于提供清晰易懂的用户界面，确保残疾人士也能无障碍使用。此外，系统应当保障用户信息的安全，采用先进的数据加密和安全存储技术，防止数据泄露和未授权访问。系统的稳定性和可靠性也至关重要，以避免因系统故障而给用户带来不便和损失。

在法律责任方面，工程师必须确保系统的设计和实现严格遵守相关法律法规，包括数据保护法、消费者权益保护法等。对用户隐私的保护是系统设计中的重要部分，必须确保所有用户数据的收集、使用和分享都基于用户的明确同意。

2.14 数据库应用开发(JAVA 篇)

本节要求了解 JDBC 体系结构和基础的使用方法，我完成了全部的 7 关。

2.14.1 JDBC 体系结构和简单的查询

本关任务：查询 client 表邮箱非空的客户信息，客户姓名，邮箱和电话。

实现方法：使用 ResultSet executeQuery(String SQL)方法获取一个 ResultSet 对象，其中 String SQL 语句可以使用 MySQL 语句进行查询。使用 resultSet.next() 遍历 ResultSet 输出相应的信息即可。代码省略。

2.14.2 用户登录

任务简单，代码略过。

2.14.3 添加新客户

任务简单，代码略过，参见代码包。

2.14.4 银行卡销户

本关任务：实现银行卡销户的方法。

实现方法：在 sql 语句中使用？，然后再使用 setInt、setString 等方法依次填 补？对应的字段，然后进行查询。代码略过，参见代码包。

2.14.5 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：根据题意编写判断条件：首先判断是否存在该用户，再判断两次输入密码是否相同，如果符合修改密码的流程则使用 update 语句更新。代码略过，参见代码包。

2.14.6 事务与转账操作

任务要求：编写一个银行卡转账的方法 transferBalance()。

transferBalance()在被调用前，柜台已经确认过转出帐号持有者身份，所以转帐方法只接受转出卡号，转入卡号和转账金额三个参数。由调用者保证转账金额为正数。transferBalance()方法的每个参数都在注释中进行了详细的说明。transferBalance()返回 boolean 值，true 表示转帐成功，false 表示转账失败，并不需要细分或解释失败的原因。

具体实现见代码包。

2.14.7 把稀疏表格转为键值对存储

本关任务：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。

实现方法：对于每项表中数据，枚举每个学生和学科，如果值非空则转存到新表去，使用 insertSC 函数实现每条数据的插入。代码省略，参见代码包。

3 课程总结

本次实验课进行了 MySQL 各种语法的具体实操。包括建立数据库，建表，完整性约束，表的修改，数据查询等基础操作，还有视图，事务，触发器，自定义函数，安全性控制，并发控制一系列提高操作，最后还完成了数据库备份恢复的了解，一个现实数据库设计，数据库的 Java 应用开发等数据库周边应用。

实训 1 和 2 是对最基本的建立数据库，建表的练习，过程十分简单。实训 3 和 4 是本次实验中最庞大的部分，基于一个金融应用的数据查询。从最简单的查询方法，到各种嵌套查询，连接查询。越往后越是有难度，我结合书本完成了实训 3 中的大部分题目。4 中向我们介绍了书中没有学到了一个查询技巧，例如 rank()的使用，这些新知识结合上偏难的查询要求，让整个实训 4 成为最难的挑战，我与同学激烈讨论，花费了大量时间完成了全部关卡。实训 5 到 10 是各种提高操作，虽然对应关卡不多，但都是典型例子，在实践过程中我收获到的知识应用经验很多，这也是只看书无法得到的。

本实验一个有特色的实训在于并发控制的读脏、幻读、不可重复读现象的设计，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，以此能更好的帮助学生掌握隔离级别的正确应用。

通过本次实验的训练，我对 MySQL 语句的使用更加熟练，并深刻体会到了 MySQL 语句的灵活性和便捷性。我学会了如何使用 MySQL 语句来实现复杂的查询需求，例如使用嵌套查询和窗口函数中的 rank()函数来进行数据分组和排序。这为我在实际工作中处理大量数据和复杂查询提供了很大的帮助。我相信通过不断的练习和实践，我将能够更加熟练地运用 MySQL 语句，提高自己的数据处理能力和工作效率。