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：	本硕博 2301 班
学　　号：	U202315763
姓　　名：	王家乐
指导教师：	杨茂林

分数	
教师签名	

2025 年 6 月 8 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	

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所有实训任务，下面将重点针对其中的部分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。

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

该关卡已完成，此报告略过。

2.1.6 UNIQUE 约束

该关卡已完成，此报告略过。

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

该关卡已完成，此报告略过。

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。

```
-- 语句1：删除表orderDetail 中的列orderDate
alter table orderDetail drop orderDate;
-- 语句2：添加列unitPrice
alter table orderDetail add unitPrice numeric(10, 2);
```

2.2.3 修改字段

该关卡已完成，此报告略过。

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，代码如下：

```
-- (1) 为表Staff 添加主码
alter table Staff add primary key (staffNo);
-- (2) Dept.mgrStaffNo 是外码，对应的主码是 Staff.staffNo:
alter table Dept add constraint FK_Dept_mgrStaffNo foreign key(mgrStaffNo)
references Staff(staffNo);
-- (3) Staff.dept 是外码，对应的主码是 Dept.deptNo:
alter table Staff add constraint FK_Staff_dept foreign key(dept) references
Dept(deptNo);
-- (4) 为表Staff 添加 check 约束，规则为：gender 的值只能为 F 或 M;
alter table Staff add constraint CK_Staff_gender check (gender in ('F', 'M'));
-- (5) 为表Dept 添加 unique 约束：deptName 不允许重复。约束名为 UN_Dept_deptName:
alter table Dept add constraint UN_Dept_deptName unique(deptName);
```

2.3 基于金融应用的数据查询(Select)

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。

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 exists(select* from property where client.c_id=pro_c_id and pro_type='3')
and exists(select* from property where client.c_id=pro_c_id and pro_type='2')
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。

实现方法：基础多表连接操作。

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

2.3.5 每份金额在 30000～50000 之间的理财产品

该关卡已完成，此报告略过。

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，再对 count 函数结果使用 max 函数统计众数。

```
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

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 = c_id and pro_type = 1 )
order by c_id;
```

2.3.8 持有两张信用卡的用户

该关卡已完成，此报告略过。

2.3.9 购买了货币型基金的客户信息

该关卡已完成，此报告略过。

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, property
where c_id = pro_c_id and pro_status = '可用'
group by pro_c_id
order by sum(pro_income) desc
limit 3;
```

2.3.11 黄姓客户持卡数量

该关卡已完成，此报告略过。

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：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。最后的分组需要使用左连接：因为可能该人并未购买该类型的产品。

```
select c_name, c_id_card, ifnull(sum(pro_amount), 0) as total_amount from client
left join (
    select pro_c_id, pro_quantity * p_amount as pro_amount
    from property, finances_product
    where pro_pif_id = p_id and pro_type = 1
    union all
    select pro_c_id, pro_quantity * i_amount as pro_amount
    from property, insurance
    where pro_pif_id = i_id and pro_type = 2
    union all
    select pro_c_id, pro_quantity * f_amount as pro_amount
    from property, fund
    where pro_pif_id = f_id and pro_type = 3
) as pro
on pro.pro_c_id = c_id
group by c_id
order by total_amount desc;
```

2.3.13 客户总资产

该关卡已完成，此报告略过。

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
    from insurance
    order by i_amount desc
    limit 1 offset 3 );
```

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 t1.c_id as c_id1, t2.c_id as c_id2
from pro as t1, pro as t2 where t1.c_id < t2.c_id and t1.f_id = t2.f_id;
```

2.3.17 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：考虑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，如果是购买高峰期，则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-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。

```
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
            ) as t1
        where amount > 1000000
    ) as t2
    ) as t3
where t3.cnt >= 3;
```

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

该关卡已完成，此报告略过。

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

2.4 数据查询(Select)--新增

本节内容在上一节 SELECT 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：首先从 property 表和 finances_product 表中筛选出 2010 年和 2011 年（通过 year(pro_purchase_time)提取年份）的理财产品销售记录（pro_type=1），按产品编号(p_id)和年份分组计算每款产品每年的销售总额（pro_quantity*p_amount 的求和）；然后使用窗口函数 rank()对每年的产品按销售总额降序排名，处理并列情况时采用标准排名方式（如 1、1、3）；最后筛选

出每年排名前三的产品记录，并按年份、排名和产品编号排序输出结果。

```
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(pro_quantity * p_amount) 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 t1
) as t2
where rk<=3
order by pyear, rk, p_id;
```

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：从 property 表和 finances_product 表中筛选出投资理财产品的客户，统计每位客户持有的不同理财产品数量（按 pro_pif_id 去重计数），记为 t1 临时表；同时从 property 表和 fund 表中筛选出投资基金产品的客户，统计每位客户持有的不同基金产品数量（按 pro_pif_id 去重计数），记为 t2 临时表。然后将这两个临时表按客户编号(pro_c_id)进行关联，筛选出同时出现在两个临时表中、且理财产品持有种类数(cnt1)大于基金产品持有种类数(cnt2)的客户记录，最后按客户编号升序排列输出结果。

```
select t1.pro_c_id
from (
( select pro_c_id, count(distinct(pro_pif_id)) as cnt1
  from property, finances_product
```

```
  where pro_type = 1  and pro_pif_id = p_id group by pro_c_id ) as t1,
( select pro_c_id, count(distinct(pro_pif_id)) as cnt2
  from property, fund
   where pro_type = 3 and pro_pif_id = f_id group by pro_c_id ) as t2 )
where t1.pro_c_id = t2.pro_c_id
and t1.cnt1 > t2.cnt2
order by t1.pro_c_id;
```

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：首先定义畅销理财产品为持有人数超过 2 的理财产品（通过分组统计 pro_pif_id 并筛选 count(*)>2 的记录确定），然后使用 NOT EXISTS 子查询确保客户不存在任何未购买的畅销理财产品，即对于每个客户检查是否存在某个畅销理财产品不在该客户的购买记录中，若不存在这样的产品则说明该客户购买了所有畅销理财产品，最后返回满足条件的客户编号并去除重复值。

```
select distinct pro_c_id from property as p1
where not exists(
    select *
    from (
        select pro_pif_id from property
        where pro_type = 1
        group by pro_pif_id
        having count(*) > 2
    ) as t1
    where t1.pro_pif_id not in (
        select pro_pif_id from property as p2
        where p1.pro_c_id = p2.pro_c_id and p2.pro_type = 1 ));
```

2.4.4 查找相似的理财产品

本关任务：查找产品 14 的相似理财产品编号（不包含 14 自身）（pro_pif_id）、该编号的理财产品的客购买客户总人数（cc）以及该理财产品对于 14 号理财产品的相似度排名值（prank）。

实现方法：筛选出 14 号理财产品购买量排名前三的客户（通过 dense_rank() 窗口函数按 pro_quantity 升序排名确定），然后找出这些客户购买过的其他理财产品（排除 14 号产品本身），统计这些相似理财产品的购买客户总人数并按人数降序排列，最后使用 dense_rank()为相似度排名（避免排名号跳跃），最终输出相似理财产品编号、购买客户总人数和相似度排名值，并按客户人数降序和产品编号升序排列结果。

```
select pro_pif_id, count(*) as cc, dense_rank() over(order by count(*) desc) as prank
from property
where
    pro_type = 1
    and pro_pif_id in (
        select distinct pro_pif_id
        from property
        where
            pro_type = 1 and pro_pif_id <> 14 and
            pro_c_id in (
                select pro_c_id
                from (
                    select pro_c_id,
                           dense_rank() over(order by pro_quantity) as rk
                    from property
                    where
                    pro_type = 1
                    and pro_pif_id = 14
                    ) as fin_rk
                where fin_rk.rk <= 3))
group by pro_pif_id
order by cc desc, pro_pif_id;
```

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：通过自连接 property 表（分别别名为 a 和 b）匹配不同客户但持有相同理财产品的记录，筛选条件限定为理财产品类型，然后按客户对分组统计共同持有的理财产品种类数，保留共同持有 2 种及以上产品的客户对，最后按第一个客户编号升序排列输出结果，展示客户对编号及共同持有的产品种数。

```
select a.pro_c_id, b.pro_c_id,count(*) as total_count
from property a, property b
where a.pro_c_id <> b.pro_c_id
    and a.pro_type = 1
    and b.pro_type = 1
    and a.pro_pif_id = b.pro_pif_id
group by a.pro_c_id, b.pro_c_id
having count(*) >= 2
order by a.pro_c_id;
```

2.4.6 查找相似的理财客户

该关卡已完成，此报告略过。

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

该关卡已完成，此报告略过。

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，剩余信息为 NULL。

```
insert nto 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 in 子查询即可。

```
delete from client
where c_id not in (
    select b_c_id from bank_card );
```

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 in (
    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 pro_id_card=c_id_card where pro_c_id = 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 (c_name, c_id_card, i_name, i_project, pro_status,
pro_quantity, i_amount, i_year, pro_income, pro_purchase_time)
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, insurance, property
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(i_amount*pro_quantity) as insurance_total_amount,
sum(pro_income) as insurance_total_revenue
from v_insurance_detail
group by c_name,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 $$
create procedure sp_fibonacci(in m int)
begin
declare x0, x1, i, xt int;
insert into fibonacci values (0, 0);
set x0 = 0, x1 = 1, i = 1;
while i<m DO
set xt = x0 + x1;
insert into fibonacci values (i, x1);
set x0 = x1;
set x1 = xt;
set i = i + 1;
end while;
end $$
delimiter ;
```

2.7.2 使用游标的存储过程

该关卡已完成，此报告略过。

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。

```
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;
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
        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
        set return_code = 0;
        rollback;
    else
        set return_code = 1;
        commit;
    end if;
    set autocommit = true;
    end$$
    delimiter ;
```

2.8 触发器

本节主要内容为触发器的应用。

2.8.1 为 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 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

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。

```
set global log_bin_trust_function_creators=1;
drop function IF EXISTS get_deposit;
-- 用create function 语句创建符合以下要求的函数:
-- 依据客户编号计算该客户所有储蓄卡的存款总额。
-- 函数名为:get_Records。函数的参数名可以自己命名:
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 ;
-- 应用该函数查询存款总额在100万以上的客户身份证号，姓名和存储总额
-- 结果依存款总额从高到低排序
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

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；
drop role if exists client_manager, 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 table 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 table bank_card
to client_manager;
-- (4) 授予 fund_manager 对 fund 表的 select,insert,update 权限；
grant select, insert, update
on table 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）

各事务隔离级别说明如表 2-1：

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	×	×	×

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:
use testdb1;
set session transaction isolation level read uncommitted;
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(1);
select tickets from ticket where flight_no = 'CA8213';
commit;
-- 事务2
use testdb1;
set session transaction isolation level read uncommitted;
start transaction;
-- 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
-- 添加代码，使事务1在事务2撤销前读脏;
set @n = sleep(2);
rollback;
```

2.11.3 不可重复读

该关卡已完成，此报告略过。

2.11.4 幻读

该关卡已完成，此报告略过。

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";
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。

```
-- 事务1:
use testdb1;
start transaction;
set @n = sleep(1);
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。

```
-- 对数据库residents 作海量备份,备份至文件residents_bak.sql:
mysqldump -h127.0.0.1 -uroot --databases residents > residents_bak.sql
-- 利用备份文件residents_bak.sql 还原数据库:
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 -h127.0.0.1 -u root
```

2.13 数据库设计与实现

2.13.1 从概念模型到 MySQL 实现

该关卡已完成，此报告略过。

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。

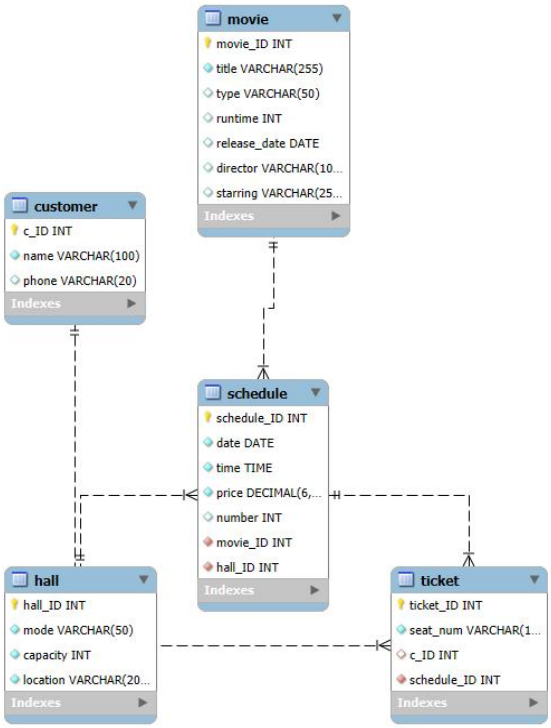

图 2-1 影院管理系统 E-R 图

并有如下关系模式：

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

2.13.3 建模工具的使用

利用 MySQL Workbench 建模模块中的 Forward engineering 功能，自动导出 SQL 脚本。该关卡已完成，此报告略过。

2.14 数据库应用开发（JAVA 篇）

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，列出客户姓名，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。

```
public static void main(String[] args) {
        Connection connection = null, Statement statement = null;
        ResultSet resultSet = null;
        try {
            Class.forName(JDBC_DRIVER);
            connection = DriverManager.getConnection(DB_URL, USER, PASS);
            statement = connection.createStatement();
            resultSet = statement.executeQuery("select c_name, c_mail, c_phone
                            from client where c_mail is not null");
            System.out.println("姓名\t 邮箱\t\t\t\t 电话");
            while (resultSet.next()) {
                System.out.print(resultSet.getString("c_name") + "\t");
                System.out.print(resultSet.getString("c_mail") + "\t\t");
                System.out.println(resultSet.getString("c_phone"));
            }
        } catch (ClassNotFoundException e) {
            System.out.println("Sorry,can`t find the JDBC Driver!");
            e.printStackTrace();
        } catch (SQLException throwables) { throwables.printStackTrace();}
        finally { try {
                if (resultSet != null) { resultSet.close(); }
                if (statement != null) { statement.close(); }
                if (connection != null) { connection.close(); }
            } catch (SQLException throwables) { throwables.printStackTrace(); }
        }
    }
```

2.14.2 用户登录

本关任务：编写客户登录程序，提示用户输入邮箱和密码，并判断正确性，给出适当的提示信息。

实现方法：方法同上关，当密码不匹配时（查询集合为空）直接返回用户名或密码错误，以防止入侵。

```
statement = connection.createStatement();
String sql = "select * from client where c_mail = ? and c_password = ?";
PreparedStatement ps = connection.prepareStatement(sql);
ps.setString(1, loginName), ps.setString(2, loginPass);
resultSet = ps.executeQuery();
if (resultSet.next()) System.out.println(" 登录成功。 ");
else System.out.println(" 用户名或密码错误！ ");
```

2.14.3 添加新客户

该关卡已完成，此报告略过。

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。

```
public static int removeBankCard(Connection connection,
                                 int b_c_id, String b_number){
    int result = 0;
    try{
        String sqlStatement = "DELETE FROM bank_card WHERE b_c_id = "+b_c_id+"
                                              AND b_number = \""+b_number+"\";";
        Statement statement = connection.createStatement();
        result = statement.executeUpdate(sqlStatement);
    } catch(SQLException e){
         .printStackTrace();
    } finally {
        return result;
    }
 }
```

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。

```
public static int passwd(Connection connection,
                         String mail,
                         String password,
                         String newPass){
    int code = -1;
    try {
```

```
            String sqlStatementSelect = "SELECT c_password FROM client WHERE c_mail
                                                    = \""+mail+"\";";
            String sqlStatementUpdate = "UPDATE client SET c_password =
                            \""+newPass+"\" WHERE c_mail = \""+mail+"\";";
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery(sqlStatementSelect);
            if(resultSet.next()) {
                String oldPassword = resultSet.getString("c_password");
                if(oldPassword.equals(password)) {
                    int result = statement.executeUpdate(sqlStatementUpdate);
                    if(result == 1) { code = 1; }
                    else { code = -1; }
                }
                else { code = 3; }
            }
            else { code = 2; }
        } catch(SQLException e){
            code = -1;
            e.printStackTrace();
        } finally { return code; }
    }
```

2.14.6 事务与转账操作

本关任务：编写一个银行卡转账的方法。

实现方法：根据题意编写判断条件：如果不合法则直接返回错误，然后判断收款方卡类型是否为信用卡决定是否要把变化的金额设置为负数。

```
    public static boolean transferBalance(Connection connection, String sourceCard,
    String destCard,  double amount){
           try {
              String sql = "select * from bank_card where b_number = ?";
              PreparedStatement preparedStatement=connection.prepareStatement(sql);
               preparedStatement.setString(1, sourceCard);
              ResultSet resultSet = preparedStatement.executeQuery();
              if (!resultSet.next() || resultSet.getString("b_type").equals("信用卡
") || resultSet.getDouble("b_balance") < amount) return false;
              preparedStatement = connection.prepareStatement(sql);
              preparedStatement.setString(1, destCard);
              resultSet = preparedStatement.executeQuery();
              if (!resultSet.next()) return false;
              sql="update bank_card set b_balance= b_balance+? where b_number=?";
              preparedStatement = connection.prepareStatement(sql);
              preparedStatement.setDouble(1, -amount);
```

```
                preparedStatement.setString(2, sourceCard);
                preparedStatement.executeUpdate();
                preparedStatement = connection.prepareStatement(sql);
                double rcv_amount = resultSet.getString("b_type").equals("信用卡
") ? -amount : amount;
                preparedStatement.setDouble(1, rcv_amount);
                preparedStatement.setString(2, destCard);
                preparedStatement.executeUpdate();
                return true;
            } catch (SQLException e) { e.printStackTrace();}
            return false;
        }
```

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。

```
    public static void main(String[] args) {
        try {
            Class.forName(JDBC_DRIVER);
            Connection connection = DriverManager.getConnection(DB_URL, USER, PASS);
            Statement statement = connection.createStatement();
            ResultSet resultSetAll = statement.executeQuery("SELECT * FROM
entrance_exam;");
            while(resultSetAll.next()) {
                int sno = resultSetAll.getInt("sno");
                int chinese = resultSetAll.getInt("chinese");
                if(chinese != 0){
                    int ret = insertSC(connection, sno, "chinese", chinese);
                }
                int math = resultSetAll.getInt("math");
                if(math != 0){
                    int ret = insertSC(connection, sno, "math", math);
                }
                int english = resultSetAll.getInt("english");
                if(english != 0){
                    int ret = insertSC(connection, sno, "english", english);
                }
                int physics = resultSetAll.getInt("physics");
                if(physics != 0){
                    int ret = insertSC(connection, sno, "physics", physics);
```

```
            }
            int chemistry = resultSetAll.getInt("chemistry");
            if(chemistry != 0){
                int ret = insertSC(connection, sno, "chemistry", chemistry);
            }
            int biology = resultSetAll.getInt("biology");
            if(biology != 0){
                int ret = insertSC(connection, sno, "biology", biology);
            }
            int history = resultSetAll.getInt("history");
            if(history != 0){
                int ret = insertSC(connection, sno, "history", history);
            }
            int geography = resultSetAll.getInt("geography");
            if(geography != 0){
                int ret = insertSC(connection, sno, "geography", geography);
            }
            int politics = resultSetAll.getInt("politics");
            if(politics != 0){
                int ret = insertSC(connection, sno, "politics", politics);
            }
        }
    } catch(Exception e) {
        e.printStackTrace();
    }
}
```

3 课程总结

本次实验课程主要通过 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，并增加了数据库设计、、数据库的 Java 应用开发等数据库周边应用，共 78 关卡。

在实验过程当中，我在数据查询 select 中遇到了一些困难，对于较难的关卡也是通过和其他同学进行交流后得到了大致的解题思路。在实现事务存储过程中，查阅了大量的资料。在数据库设计关卡中，任务要求的表和数据项非常繁杂，且制约关系非常复杂，因而在本关实现中也是遇到了不少的麻烦

本次实验的训练，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，并且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。

我深刻体会到理论知识与工程实践的深度融合：事务隔离级别在并发场景中的具体表现、索引对复杂查询的性能优化作用等抽象概念，在代码调试中得到直观验证。金融查询任务中“持有相同基金组合客户识别”等综合题型，极大提升了解决实际问题的能力。本课程为构建高可靠、高性能数据库系统奠定了扎实基础，也为后续分布式系统学习提供了重要支撑。