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：	计算机 2207
学　　号：	U202215576
姓　　名：	王嘉成
指导教师：	赵小松

分数	
教师签名	

2024 年 7 月 5 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，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。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 全部实训任务、3~4 部分任务，5~11 全部实训任务，13~14 全部实训任务，下面将重点针对其中的 3，4，7，13，14 任务阐述其完成过程中的具体工作。

2.1 MySQL-数据库、表与完整性约束的定义(Create)

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，测试库中有已有相应测试数据，请依据关卡任务需求完成相应查询动作。本章完成了前 14 关。

2.1.1 金融应用场景介绍,查询客户主要信息

该关卡难度较小，具体情况本报告略过。

2.1.2 邮箱为 null 的客户

该关卡难度较小，具体情况本报告略过。

2.1.3 既买了保险又买了基金的客户

该关卡难度较小，具体情况本报告略过。

2.1.4 办理了储蓄卡的客户信息

该关卡难度较小，具体情况本报告略过。

2.1.5 每份金额在 30000～50000 之间的理财产品

该关卡难度较小，具体情况本报告略过。

2.1.6 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。

实现过程：首先，创建一个新表命名为 c，对资产表 property 中的 pro_income 字段进行分组统计，将相同 pro_income 值的记录进行分组，并计算每个分组中 pro_income 值的出现次数，使用 count(pro_income)函数，并将这个出现次数命名为 presence。接下来，在主查询中，选择出现次数最大的记录，先使用子查询找到 presence 字段的最大值，然后在 c 中筛选出 presence 等于该最大值的记录。

```sql
with c as
(
    select pro_income,count(pro_income) as presence
    from property
    group by pro_income
)
select pro_income, presence
from c
where presence =
(
    select max(presence)
    from c
)
;
```

图 1 商品收益的众数具体代码实现

2.1.7 未购买任何理财产品的武汉居民

任务要求：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。

实现过程：从 client 表中选择客户的名称、电话号和邮箱，然后，使用 WHERE 子句过滤出身份证隶属武汉市的客户，武汉市的身份证号码以 4201 开头，注意具体实现语句为 c_id_card LIKE '4201%'，接着，使用 NOT EXISTS 子查询来排除那些购买过理财产品的客户。在子查询中，从 property 表中选择所有记录，并且确保 c_id 等于 pro_c_id，同时理财产品的类型 pro_type 等于 1。如果子查询返回任何记录，则表示该客户购买过理财产品，因此 NOT EXISTS 将排除该客户，最后进行结果排序。

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

图 2 未购买任何理财产品的武汉居民具体代码实现

2.1.8 持有两张信用卡的用户

该关卡难度较小，具体情况本报告略过。

2.1.9 购买了货币型基金的客户信息

该关卡难度较小，具体情况本报告略过。

2.1.10 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。

实现过程：创建一个新表命名为 c，计算每个客户的总收益，将客户表 client 和资产表 property 进行连接，确保每条资产记录对应一个客户记录并过滤出资产状态为“可用”的资产记录，然后，对这些可用资产的收益进行求和，并将求和结果命名为 total_income，结果按照客户 ID 进行分组，以确保每个客户只有一条总收益记录。在主查询中，选择 c 中排名前三的客户记录，将结果按总收益降序排序，并限制输出前三条记录。

```
with c as
(
    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
)
select c_name,c_id_card,total_income
from c
order by total_income desc
limit 3
;
```

图 3 投资总收益前三名的客户具体代码实现

2.1.11 黄姓客户持卡数量

该关卡难度较小，具体情况本报告略过。

2.1.12 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，并按投资总金额降序排序。

实现过程：首先，从客户表 client 中选择客户的名称、身份证号，以及计

算每个客户的投资总金额，接下来，通过左连接将客户表和一个子查询的结果集连接，目的是从资产表和相应的产品表中计算每个客户的各类资产投资金额，并将这些金额合并到一个结果集中。

在子查询中，使用 UNION ALL 将三部分结果合并：对于理财产品，从资产表和理财产品表中选择客户 ID 和每笔投资金额（pro_quantity * p_amount），其中 pro_pif_id 等于 p_id，并且资产类型等于 1；对于保险产品，从资产表和保险表中选择客户 ID 和每笔投资金额，其中 pro_pif_id 等于 i_id，并且资产类型等于 2；对于基金产品，从资产表和基金表 fund 中选择客户 ID 和每笔投资金额，其中 pro_pif_id 等于基金产品 ID（f_id），并且资产类型等于 3。

通过 LEFT JOIN 将客户表 client 和子查询的结果集连接，使用 pro.pro_c_id = c_id 作为连接条件，使用 GROUP BY c_id 对结果进行分组，以确保每个客户只有一条记录，最终，对结果按总金额降序排序，确保总金额最大的客户排在最前。

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

图 4 客户理财、保险与基金投资总额具体代码实现

2.1.13 客户总资产

任务要求：列出所有客户的编号、名称和总资产。总资产定义为储蓄卡余额、投资总额、投资总收益的和，减去信用卡透支的金额，客户总资产包括被冻结的

资产。

实现过程：从客户表 client 中选择客户编号、客户名称以及总资产，使用左连接将客户表与一个子查询结果集连接，子查询中合并了六部分结果，通过 UNION ALL 操作符将它们组合在一起，每部分结果表示客户某类资产或负债的总额。

从资产表 property 和理财产品表 finances_product 中选择客户 ID 和每笔投资金额，根据资产类型（pro_type = 1）进行过滤，并按客户 ID 分组，求出理财产品的总金额，从资产表 property 和保险表 insurance 中选择客户 ID 和每笔投资金额，根据资产类型（pro_type = 2）进行过滤，并按客户 ID 分组，求出保险产品的总金额，从资产表 property 和基金表 fund 中选择客户 ID 和每笔投资金额，根据资产类型（pro_type = 3）进行过滤，并按客户 ID 分组，求出基金产品的总金额，从资产表 property 中选择客户 ID 和投资收益，按客户 ID 分组，求出投资总收益。

从银行卡表 bank_card 中选择客户 ID 和储蓄卡余额，根据卡类型（b_type = "储蓄卡"）进行过滤，并按客户 ID 分组，求出储蓄卡余额。从银行卡表 bank_card 中选择客户 ID 和信用卡透支金额，根据卡类型（b_type = "信用卡"）进行过滤，并按客户 ID 分组，求出信用卡透支金额。

子查询的结果集命名为 proper_total，并与客户表进行左连接，使用 c_id = pro_c_id 作为连接条件，最后，使用 GROUP BY 对结果进行分组，以确保每个客户只有一条记录，计算每个客户的总资产，将其命名为 total_property。

```
SELECT c_id,c_name,ifNULL(SUM(proper_total.total),0) AS total_property
FROM client LEFT OUTER JOIN
    (
    SELECT pro_c_id,SUM(finances_product.p_amount*property.pro_quantity)
AS total
    FROM finances_product,property
    WHERE p_id = pro_pif_id AND pro_type = 1
    GROUP BY pro_c_id
    UNION ALL
    SELECT pro_c_id,SUM(insurance.i_amount*property.pro_quantity) AS total
    FROM insurance,property
    WHERE i_id = pro_pif_id AND pro_type = 2
    GROUP BY pro_c_id
    UNION ALL
    SELECT pro_c_id,SUM(fund.f_amount*property.pro_quantity) AS total
    FROM fund,property
    WHERE f_id = pro_pif_id AND pro_type = 3
    GROUP BY pro_c_id
    UNION ALL
    SELECT pro_c_id,SUM(pro_income) AS total
    FROM property
    GROUP BY pro_c_id
```

```
    UNION ALL
    SELECT b_c_id AS pro_c_id,SUM(b_balance) AS total
    FROM bank_card
    WHERE b_type = "储蓄卡"
    GROUP BY b_c_id
    UNION ALL
    SELECT b_c_id AS pro_c_id,SUM(0-b_balance) AS total
    FROM bank_card
    WHERE b_type = "信用卡"
    GROUP BY b_c_id) AS proper_total ON c_id=pro_c_id
GROUP BY c_id;
```

图 5 客户总资产具体代码实现

2.1.14 第 N 高问题

该关卡难度较小，具体情况本报告略过。

2.1.15 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及其排名，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。

实现过程：第一条 SQL 语句实现基金总收益排名（名次不连续）。首先，内部子查询计算每个客户的基金投资总收益，将结果按总收益降序排列，而外部查询利用用户定义变量（@rankcount、@rank、@current_revenue）进行排名，确保总收益相同时名次相同，即并列名次，最终结果中包括客户编号、总收益和排名。

第二条 SQL 语句实现基金总收益排名（名次连续）。首先，内部子查询同样计算每个客户的基金投资总收益，并按总收益降序排列，然后，外部查询同样利用用户定义变量（@rank、@current_revenue）进行排名，确保总收益相同时名次相同，但与第一条 SQL 语句不同的是，名次是连续的。最终结果中包括客户编号、总收益和排名。

```sql
SELECT pro_c_id,total_revenue,trank AS 'rank'
FROM(
    SELECT pro_c_id,total_revenue,
        @rankcount := @rankcount + 1,
        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;
```

```sql
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;
```

图 6 基金收益两种方式排名两种代码具体实现

2.1.16 持有完全相同基金组合的客户

该关卡难度较小，具体情况本报告略过。

2.1.17 购买基金的高峰期

任务要求：查询 2022 年 2 月的基金购买高峰期。高峰期定义为至少连续三个交易日所有投资者购买基金的总金额超过 100 万。假设 2022 年春节假期之后的第一个交易日为 2 月 7 日，周六和周日是非交易日。查询结果按日期顺序列出高峰时段的日期和当日基金的总购买金额，总购买金额命名为 total_amount。

实现过程：内部子查询计算每个交易日的基金购买总金额，并筛选出总金额超过 100 万的日期，日期条件为 2022 年 2 月 7 日至 2 月 28 日，并排除了周六和周日，使用用户定义变量@dayOffset 和@curday 来计算每个日期相对于前一个日

期的间隔天数，判断当前日期与前一个日期是否连续，并相应地更新@dayOffset。通过 gapDay 字段标识日期间隔的组，在外层查询中使用嵌套子查询，计算每个 gapDay 组内的日期数量，筛选出至少有三个连续交易日的组，最终结果按日期顺序列出高峰时段的日期和当日基金的总购买金额。

```
SELECT pro_purchase_time,total_amount
FROM
    (SELECT pro_purchase_time, total_amount, IF(DATEDIFF
(pro_purchase_time, @curday) = IF(DATEDIFF(pro_purchase_time, "2022-2-7")
%7 = 0, 3, 1), @dayOffset, @dayOffset := @dayOffset+1) AS gapDay, @curday
:= pro_purchase_time
    FROM (SELECT @dayOffset := 0, @curday := 0 ) AS d,
        (SELECT pro_purchase_time, SUM(pro_quantity*f_amount) AS
total_amount
        FROM fund, property
        WHERE fund.f_id = property.pro_pif_id AND
                property.pro_type = 3 AND
                pro_purchase_time >= "2022-2-7" AND
                pro_purchase_time <= "2022-2-28" AND
                DATEDIFF(pro_purchase_time,"2022-2-5")%7 >1
        GROUP BY pro_purchase_time
        HAVING SUM(pro_quantity*f_amount) >= 1000000
        ORDER BY pro_purchase_time ASC) AS highs
    ) AS peaks
```

```
WHERE gapDay IN
    (SELECT gapDay
    FROM
        (SELECT pro_purchase_time, total_amount, IF(DATEDIFF
(pro_purchase_time, @curdays) = IF(DATEDIFF(pro_purchase_time, "2022-2-7")
%7 = 0, 3, 1), @dayOffsets, @dayOffsets := @dayOffsets+1) AS gapDay,
@curdays := pro_purchase_time
        FROM (SELECT @dayOffsets := 0, @curdays := 0 ) AS d,
            (SELECT pro_purchase_time, SUM(pro_quantity*f_amount) AS
total_amount
            FROM fund, property
            WHERE fund.f_id = property.pro_pif_id AND
                    property.pro_type = 3 AND
                    pro_purchase_time >= "2022-2-7" AND
                    pro_purchase_time <= "2022-2-28" AND
                    DATEDIFF(pro_purchase_time,"2022-2-5")%7 >1
            GROUP BY pro_purchase_time
            HAVING SUM(pro_quantity*f_amount) >= 1000000
            ORDER BY pro_purchase_time ASC) AS high
        ) AS peak
    GROUP BY gapDay
    HAVING COUNT(*)>=3);
```

图 7 购买基金的高峰期具体实现

2.2 MySQL-数据查询(Select)-新增

本小节子任务仍然以第 2.3 子任务的数据库内容为背景，但内容与统计、相似性推荐相关。本章完成了前 14 关。

2.2.1 查询销售总额前三的理财产品

任务要求：查询销售总额前三的理财产品

实现过程：通过 UNION ALL 合并两个子查询结果，每个子查询分别计算 2010 年和 2011 年的销售总额前三的理财产品，子查询中选择理财产品 ID 和计算其销售总额，从理财产品表 finances_product 和资产表 property 中获取数据，使用 WHERE 子句筛选条件，确保资产类型为理财产品且购买时间年份为 2010 或 2011，按理财产品 ID 分组并按销售总额降序排序，限制结果为销售总额前三的理财产品，然后，外层查询使用窗口函数 rank()计算每年内的排名，按年份分区，并按销售总额降序排序计算排名，最终结果包括年份、排名、理财产品 ID 和销售总额，并按年份、排名和理财产品 ID 升序排序。

```
SELECT pyear, rank() over(partition by pyear order by sumamount desc) AS
rk, p_id, sumamount
FROM ((SELECT "2010" AS pyear, p_id, SUM(p_amount*pro_quantity) AS
sumamount
        FROM finances_product, property
        WHERE pro_pif_id = p_id AND pro_type = 1 AND year
(pro_purchase_time) = "2010"
        GROUP BY p_id
        ORDER BY sumamount DESC
        LIMIT 0,3)
    UNION ALL
        (SELECT "2011" AS pyear, p_id, SUM(p_amount*pro_quantity) AS
sumamount
        FROM finances_product, property
        WHERE pro_pif_id = p_id AND pro_type = 1 AND year
(pro_purchase_time) = "2011"
        GROUP BY p_id
        ORDER BY sumamount DESC
        LIMIT 0,3)
    ) AS p1
ORDER BY pyear ASC, rk ASC, p_id ASC;
```

图 8 查询销售总额前三的理财产品具体代码实现

2.2.2 投资积极且偏好理财类产品的客户

该关卡难度较小，具体情况本报告略过。

2.2.3 查询购买了所有畅销理财产品的客户

任务要求：查询购买了所有畅销理财产品的客户，畅销理财产品定义为购买次数超过 2 次的理财产品。

实现过程：首先，从资产表 property 中选择出所有的畅销理财产品，选择出所有类型为理财产品，并且购买次数超过 2 次的产品 ID，然后，从资产表 property 中选择所有客户 ID 并通过 DISTINCT 确保客户 ID 的唯一性，使用 NOT EXISTS 来确保客户购买了所有的畅销理财产品，检查 fin_pos 子查询结果中的每个理财产品 ID 是否都在客户购买的理财产品列表中，如果有任何一个理财产品 ID 不在客户购买列表中，则该客户不符合条件。

```
select DISTINCT 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(*)>2
      ) fin_pos
      where fin_pos.pro_pif_id NOT IN
      (
         select pro_pif_id
         from property p2
         where p1.pro_c_id = p2.pro_c_id AND
         p2.pro_type = 1
      )
);
```

图 9 查询购买了所有畅销理财产品的客户具体代码实现

2.2.4 查询任意两个客户的相同理财产品数

任务要求：查询任意两个客户的相同理财产品数并筛选出持有至少两个相同理财产品的客户对。

实现过程：从资产表 property 中选择两个别名为 A 和 B 的实例，使用 WHERE 子句筛选条件：两个实例的资产类型必须是理财产品，客户 ID 必须不同，理财产品 ID 必须相同。在结果集中，选择 A.pro_c_id 和 B.pro_c_id 作为客户 ID，

并计算相同理财产品的数量，使用 count(distinct A.pro_pif_id)，按客户 ID 进行分组，并筛选出持有至少两个相同理财产品的客户对，最终结果按客户 ID 升序排序。

```
select
    A.pro_c_id as pro_c_id,
    B.pro_c_id as pro_c_id,
    count(distinct A.pro_pif_id) as total_count
from property A, property B
where
    A.pro_type = 1
    and B.pro_type = 1
    and A.pro_c_id != B.pro_c_id
    and A.pro_pif_id = B.pro_pif_id
group by
    A.pro_c_id,
    B.pro_c_id
having
    count(distinct A.pro_pif_id) >= 2
order by A.pro_c_id;
```

图 10 查询任意两个客户的相同理财产品数具体代码实现

2.3 存储过程与事务

使用 MySQL 语言实现存储过程的创建和使用，本章全部关卡通过。

2.3.1 使用流程控制语句的存储过程

任务要求：创建存储过程`sp_fibonacci(in m int)`，向表 fibonacci 插入斐波拉契数列的前 m 项，及其对应的斐波拉契数。

实现过程：先定义了一个名为 sp_fibonacci 的存储过程，接受一个整数参数 m，表示要生成的斐波那契数列的前 m 项，其次，存储过程开始时向名为 fibonacci 的表插入了一个初始值(0, 0)，通过使用变量和循环来生成斐波那契数列，在循环中，通过迭代计算当前值和前一个值的和，并将当前迭代数和当前值插入到 fibonacci 表中，循环通过递增计数器 i 来控制，直到插入了 m 项为止，最后，存储过程定义结束，并恢复了默认的 SQL 语句分隔符。

```
declare a,b,c,i int;
insert into fibonacci values(0,0);
set a=1,b=0,i=1;
while i<m Do
    set c=a+b;
    insert into fibonacci values(i,a);
    set b=a;
    set a=c;
    set i=i+1;
end while;
```

图 11 使用流程控制语句的存储过程具体代码实现

2.3.2 使用游标的存储过程

任务要求：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:

实现过程：使用了两个游标（cur1 和 cur2），分别从员工表中选择医生和护士的信息。主循环从 start_date 开始，逐日处理直至 end_date：

在每一天的处理过程中，首先从 cur1 游标中获取两个护士的姓名，然后根据当前日期的星期几判断是否需要等待特定的医生值班。如果是周日且有医生待命，则将其值班信息赋给变量 doctor，并清空待命标志。否则，从 cur2 游标中获取下一个医生的姓名和类型，并根据当前日期的工作日情况安排医生值班。如果当前日期是周五、周六或周日，并且医生的类型为 1，则将当前医生指定为待命医生，并再次从 cur2 中获取下一个医生的信息。

最后，将每天的日期、医生的姓名以及两个护士的姓名插入到预定的值班表 night_shift_schedule 中。循环结束后，关闭了游标，确保了资源的释放和存储过程的完成。通过这种方式，存储过程有效地管理了医护人员的值班安排，确保了在连续期间内的每天都有合适的人员进行大夜班值班。

```
delimiter $$
create procedure sp_night_shift_arrange(in start_date date, in end_date
date)
begin
DECLARE done, waitdir INT DEFAULT FALSE;
DECLARE nowdate DATE;
DECLARE waitdr,dr, nr1, nr2 CHAR(30);
DECLARE drtype INT;
DECLARE drlist CURSOR FOR SELECT e_name,e_type FROM employee WHERE e_type
< 3;
DECLARE nrlist CURSOR FOR SELECT e_name FROM employee WHERE e_type = 3;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN drlist;
OPEN nrlist;
SET nowdate = start_date;
WHILE nowdate <= end_date DO
    IF weekday(nowdate) < 5 AND waitdir THEN
        SET dr = waitdr, waitdir = FALSE;
    ELSE
        FETCH drlist INTO dr, drtype;
        IF done THEN
            CLOSE drlist;
            OPEN drlist;
```

```
            FETCH drlist INTO dr, drtype;
            SET done = FALSE;
        END IF;
        IF weekday(nowdate) >= 5 AND drtype = 1 THEN
            SET waitdir = TRUE, waitdr = dr;
            FETCH drlist INTO dr, drtype;
            IF done THEN
                CLOSE drlist;
                OPEN drlist;
                FETCH drlist INTO dr, drtype;
                SET done = FALSE;
            END IF;
        END IF;
    END IF;

    FETCH nrlist INTO nr1;
    IF done THEN
        CLOSE nrlist;
        OPEN nrlist;
        FETCH nrlist INTO nr1;
        SET done = FALSE;
    END IF;
```

```
    FETCH nrlist INTO nr2;
    IF done THEN
        CLOSE nrlist;
        OPEN nrlist;
        FETCH nrlist INTO nr2;
        SET done = FALSE;
    END IF;
    INSERT INTO night_shift_schedule VALUES (nowdate, dr, nr1, nr2);
    SET nowdate = DATE_ADD(nowdate,interval 1 day);
END WHILE;
end$$

delimiter ;
```

图 12 使用游标的存储过程具体代码实现

2.3.3 使用事务的存储过程

任务要求：实现一个转账操作的存储过程 sp_transfer_balance，实现从一个帐户向另一个帐户转账。

实现过程：首先关闭了自动提交模式，确保操作在整个事务内进行。然后，通过两个 UPDATE 语句分别更新源账户和目标账户的余额，根据账户类型进行相应的增减操作。接着，使用条件判断来验证转账的前提条件：确保源账户在扣除转出金额后余额大于零，并且目标账户存在。如果条件满足，将 return_code 设为 1，表示转账成功，并提交事务。如果条件不满足，则将 return_code 设为 0，并回滚事务，保证数据的一致性和完整性。整个过程利用了事务机制，确保了转账操作的原子性和安全性。

```
delimiter $$
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 ;
```

图 13 使用事务的存储过程具体实现过程

2.4 数据库设计与实现

本小节帮助学生全面掌握从需求分析到构建概念模型，再到逻辑模型的整个数据库建模流程，并最终能够独立创建一个符合特定业务需求的 MySQL 数据库。学生将学习如何将概念模型转化为实际的 MySQL 数据库实现，掌握使用建模工具辅助设计的技能，并从需求分析开始，逐步发展到构建逻辑模型的能力。

本章全部关卡通过。

2.4.1 从概念模型到 MySQL 实现

根据任务要求，按需对各个主体创建表，并按需指定各个主体的主码、外码

和其他码的缺省值、不可空性、不可重性等。在基于第一关数据库表创建的基础上，要重视索引和约束等元素的物理实现，这涉及到使用 SQL 语句将概念模型具体化到 DBMS 中。本关代码过长，在这仅展示部分表创建过程。

图 14 user 要求

```sql
DROP TABLE IF EXISTS user;
CREATE TABLE user(
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    firstname VARCHAR(50) NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    dob DATE NOT NULL,
    sex CHAR(1) NOT NULL,
    email VARCHAR(50),
    phone VARCHAR(30),
    username VARCHAR(20) NOT NULL UNIQUE,
    password CHAR(32) NOT NULL,
    admin_tag TINYINT NOT NULL DEFAULT 0
);
```

图 15 user 具体代码实现

2.4.2 从需求分析到逻辑模型

根据应用场景业务需求描述，完成 ER 图，并转换成关系模式。

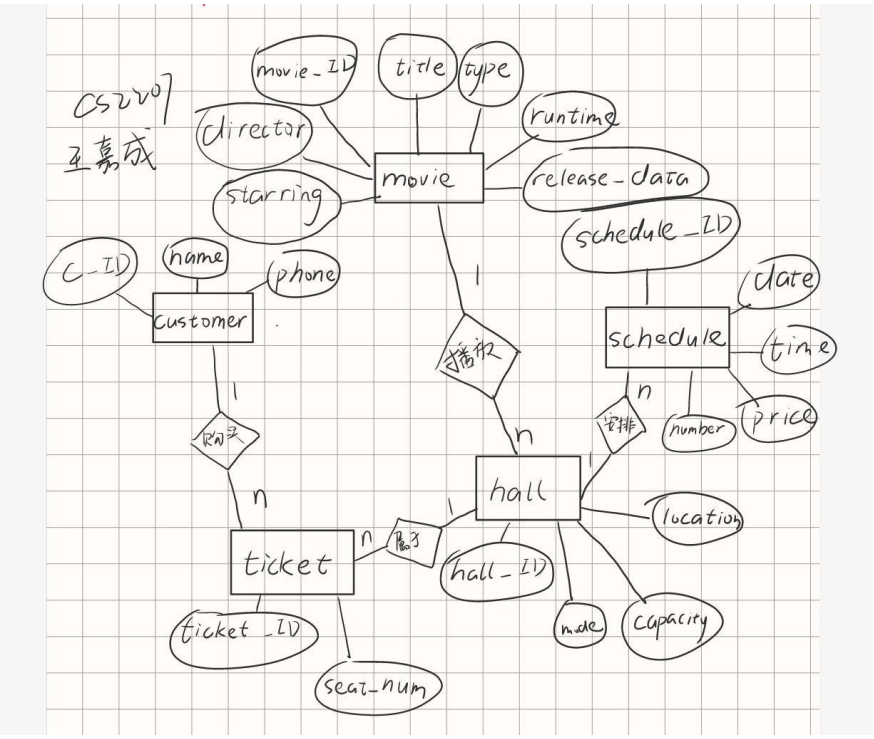

图 16 具体 ER 图实现

```
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)
```

图 17 具体关系模式实现

2.4.3 建模工具的使用

略。

2.4.4 制约因素分析与设计

在数据库设计与实现任务的解决方案设计中，考虑社会、健康、安全、法律、文化及环境等制约因素至关重要，这些因素可以影响数据库系统的设计、实施和使用的各个方面。

法律与法规合规性：数据库设计必须符合所在地区的法律和法规。例如，个人数据保护法律要求严格保护用户隐私，因此在设计中需要考虑数据的安全存储和访问控制。

社会和文化因素：数据库设计应考虑所处社会和文化背景对数据使用和处理的影响，不同文化对隐私、数据共享和访问的看法可能不同，设计时需谨慎考虑这些因素。

健康与安全：特别是涉及健康信息或安全敏感数据的数据库，必须严格遵守健康信息保护法律或安全标准，数据的存储和传输必须具备高度的安全性和保护措施。

综上所述，数据库设计与实现不仅仅是技术层面的考量，还需要综合考虑社会、健康、安全、法律、文化及环境等多方面的制约因素，以确保系统的可持续性、合规性和社会价值。

2.4.5 工程师责任及其分析

任务的解决过程和解决方案与社会、健康、安全、法律以及文化等因素之间的相互影响非常密切，工程师在这些影响因素下承担重要的社会责任。

社会影响：工程师需要在设计过程中考虑如何最大化社会利益，同时避免可能带来的负面影响。

健康与安全：特别是在健康信息系统的设计中，工程师有责任确保数据的保密性和完整性，以及系统的安全性，以避免任何形式的数据泄露或滥用。这不仅是法律义务，也是对公众信任的基本承诺。

法律合规性：工程师需要了解和遵守这些法律要求，并在系统设计中内建适当的合规措施。

在面对这些责任时，工程师应该具备跨学科的能力和综合的思维方式，不仅仅关注技术的实施，还要深入理解和评估技术对社会和个体的影响。他们需要在设计和实施过程中与各种利益相关者进行沟通和合作，包括政府监管机构、社会团体、行业协会和最终用户，以确保解决方案的可持续性和社会价值。

2.5 数据库应用开发(JAVA 篇)

利用 java 和数据库知识实现数据库应用开发，本章全部关卡通过。

2.5.1 JDBC 体系结构和简单的查询

该关卡已完成，具体情况本报告略过。

2.5.2 用户登录

任务要求：实现用户登录验证的功能。

实现过程：首先通过导入必要的 Java 类库，如 java.sql.* 和 java.util.Scanner，准备好与用户交互和数据库操作所需的工具，然后，程序提示用户输入用户名和密码，并使用 JDBC 连接到 MySQL 数据库，准备一个预编译的 SQL 查询语句，用于检查输入的用户名和密码是否与数据库中的记录匹配。通过执行 SQL 查询并检查结果集，程序确定是否成功找到匹配的用户名和密码，如果找到匹配，程序输出 "登录成功"；否则输出 "用户名或密码错误"。最后，程序关闭数据库连接和所有相关资源，以确保安全地释放使用的数据库资源。

```java
import java.sql.*;
import java.util.Scanner;

public class Login {
    public static void main(String[] args) {
        Connection connection = null;
        //申明下文中的resultSet, statement
        PreparedStatement statement = null;
        ResultSet resultSet = null;

        Scanner input = new Scanner(System.in);

        System.out.print("请输入用户名：");
        String loginName = input.nextLine();
        System.out.print("请输入密码：");
        String loginPass = input.nextLine();

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");

            String userName = "root";
            String passWord = "123123";
            String url = "jdbc:mysql://127.0.0.1:3306/finance?
useUnicode=true&characterEncoding=UTF8&useSSL=false&serverTimezone=UTC";
            connection = DriverManager.getConnection(url, userName,
passWord);
            // 补充实现代码:
            statement = connection.prepareStatement("SELECT * FROM client
WHERE c_mail = ? AND c_password = ?");
            statement.setString(1, loginName);
            statement.setString(2, loginPass);
            resultSet = statement.executeQuery();
            if(resultSet.next()){
                System.out.println("登录成功。");
            }
            else {
                System.out.println("用户名或密码错误！");
            }

        } catch (ClassNotFoundException e) {
            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();
            }
        }
    }
}
```

图 18 用户登录代码具体实现

2.5.3 添加新客户

该关卡已完成，具体情况本报告略过。

2.5.4 银行卡销户

该关卡已完成，具体情况本报告略过。

2.5.5 客户修改密码

该关卡已完成，具体情况本报告略过。

2.5.6 事务与转账操作

该关卡已完成，具体情况本报告略过。

2.5.7 把稀疏表格转为键值对存储

任务要求：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，这样可以直接丢失没有值列。

实现过程：从名为 entrance_exam 的数据库表中读取学生的考试成绩，并将非零成绩插入到名为 sc 的另一个表中，首先使用 DriverManager.getConnection() 建立与 MySQL 数据库的连接，然后执行查询以获取所有学生的成绩数据。对于每个学生，程序检查每个科目的成绩，如果成绩不为零，则将其插入到 sc 表中，通过设置参数和执行 update 操作，完成数据的插入。最后，程序关闭数据库连接并处理可能的异常情况。

```
import java.sql.*;

public class Transform {
    static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://127.0.0.1:3306/sparsedb?
allowPublicKeyRetrieval=true&useUnicode=true&characterEncoding=UTF8&
useSSL=false&serverTimezone=UTC";
    static final String USER = "root";
    static final String PASS = "123123";

    /**
     * 向sc表中插入数据
     *
     */
    public static int insertSC(Connection connection, int sno,
                               String col_name, int col_value){
        int result = 0;
        try {
            Statement statement = connection.createStatement();
            String sqlStatement = "INSERT INTO sc VALUES("+sno+",\""
+col_name+"\","+col_value+");";
            result = statement.executeUpdate(sqlStatement);
        } catch(SQLException e) {
            e.printStackTrace();
```

```
        } finally {
            return result;
        }
    }

    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();
        }

    }
}
```

图 19 把稀疏表格转为键值对存储具体代码实现

3 课程总结

在本次数据库课程的实践中，我全面深入地体验了从理论到实践的数据库技术应用，通过课程安排的多方面任务，我不仅加深了对数据库概念的理解，还提升了我的技术实践能力。

我系统学习了数据库的基础知识，包括数据库的创建、管理以及表、索引、视图等基础数据对象的建立。通过深入学习 SQL 语言，我掌握了数据的增删查改操作，这些是数据库管理的核心技能，我特别注重了数据类型的选择和索引的设计，以优化查询性能和数据访问速度。通过构建高效的查询逻辑框架，我学会了利用 SQL 的聚集函数和子查询解决复杂问题，深刻体会到了编程中逻辑设计的重要性，以及 SQL 语法在简化查询过程中的作用。

在确保数据正确性和一致性方面，我通过设置各种数据约束，如主键、外键和检查约束，有效地维护了数据的完整性。学习了视图的使用，不仅简化了复杂的查询，还增强了数据的安全性，通过视图控制用户对数据的访问权限，确保数据的安全性和隐私保护。在安全性控制方面，我深入了解了用户权限管理，学会了如何创建用户、分配权限和撤销权限，从而有效管理多用户环境中的数据库资源访问。

在数据库设计与实现过程中，我完成了从用户需求到概念模型的设计，进一步到逻辑模型的转化，最终实现了 MySQL 数据库的建立。这个过程锻炼了我的逻辑思维和抽象能力，深化了我对数据库设计全过程的理解。在数据库应用开发方面，我将理论知识与实际编程相结合，使用 JDBC API 实现了与 Java 应用程序的交互，包括数据库连接、查询和更新操作。这一过程不仅加深了我对数据库在软件开发中作用的理解，还显著提升了我的编程能力。

总的来说，本次数据库课程的实践使我不仅掌握了扎实的数据库基础知识和实际操作能力，还培养了解决问题的能力和逻辑思维。尽管在学习过程中遇到了某些挑战，例如 MySQL 特定函数的学习，但这些挑战激励我进一步深入学习和探索数据库技术的更高级特性。

衷心感谢老师们精心设计的实验内容和细致的指导，期待在未来的学习中继续拓展数据库理论知识和实践技能。