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：	CS2201
学　　号：	U202215365
姓　　名：	叶俊江
指导教师：	瞿彬彬

分数	
教师签名	

2024 年 7 月 3 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	
6	

总分	

目　录

1 课程任务概述

“数据库系统原理实践”是配合“数据库系统原理”课程独立开设的实践课，注重理论与实践相结合。本课程以 MySQL 为例，系统性地设计了一系列的实训任务，基础内容涉及以下几个部分，并可结合实际对 DBMS 原理的掌握情况向内核设计延伸：

1) 数据库、表、索引、视图、约束、存储过程、函数、触发器、游标等数据对象的管理与编程；
2) 数据查询，数据插入、删除与修改等数据处理相关任务；
3) 数据库的安全性控制，完整性控制，恢复机制，并发控制机制等系统内核的实验；
4) 数据库的设计与实现；
5) 数据库应用系统的开发(JAVA 篇)。

课程依托头歌实践教学平台，实践课程 url 见相关课堂教师发布链接及其邀请码。实验环境为 Linux 操作系统下的 MySQL 8.0.28（主要为 8.028 版本，部分关卡使用 8.022 版本，使用中基本无差别）。在数据库应用开发环节，使用 JAVA 1.8。

2 任务实施过程与分析

本次实践课程在头歌平台进行，实践任务均在平台上提交代码，所有完成的任务、关卡均通过了自动测评。本次实践最终完成了课程平台中所有实训任务，下面将重点针对其中的各个实训中重点的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)

本章要求了解数据库、表与完整性约束的定义方法，了解数据库语言的基本语法。本任务关卡均已完成，由于较为简单故不再赘述。

2.2 表结构与完整性约束的修改(ALTER)

本章学习 alter table 语句的大部分功能和修改表结构的方法（如添加列、约束，删除列、约束，修改列）等基础知识，了解表的完整性约束。本任务关卡均已完成，由于较为简单故不再赘述。

2.3 基于金融应用的数据查询(Select)

本实训采用的是某银行的一个金融场景应用的模拟数据库，测试库中有已有相应测试数据，提供了六个表使用，分别为：client（客户表）、bank_card（银行卡）、finances_product（理财资产表）、insurance（保险表）、fund（基金表）、property（资产表）。本实训中大多采用 select 语句来按关卡要求对表进行数据查询，下面将针对其中重点关卡进行分析。

2.3.1 查询客户主要信息

本关较为简单，答案即一个带 order by 排序的简单 select 语句，不再展开分析。

2.3.2 邮箱为 null 的客户

本关主要是注意要使用 is null 语句进行判空，由于关卡较为简单故不展开分析。

2.3.3 既买了保险又买了基金的客户

本关涉及嵌套 Select 语句，查询既买了保险又买了基金的客户的客户名称、邮箱和电话，结果按照 c_id 排序，仅使用 select、from、where、order by 等关键字即可实现。代码如下：

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

2.3.4 办理了储蓄卡的客户信息

本关较为简单，仅使用 select、from、where、order by 等关键字即可实现，不再展开分析。

2.3.5 每份金额在 30000～50000 之间的理财产品

本关较为简单，注意 between and 语句使用即可，不展开分析。

2.3.6 商品收益的众数

本关需要注意的是 where 无法与聚合函数一起使用，故而在此我们需要使用 having 子句来进行数据的筛选。并且本关还需要用到 count()函数来进行计算，用 having 子句来实现关卡中“众数”的要求。代码如下：

```
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

本关需要注意的是 not exists 或者 not in 的使用即可，由于实现较为简单故不再展开赘述。

2.3.8 持有两张信用卡的用户

本关也使用嵌套查询，先在子查询中筛选出含有两张及其以上信用卡的用户的 id，再进行选取以及排序。代码如下：

```
select c_name,c_id_card,c_phone from client
    where (c_id,"信用卡")in
    (select b_c_id,b_type from bank_card group by b_c_id,b_type having count(*)>=2);
```

2.3.9 购买了货币型基金的客户信息

本关需要使用两次嵌套查询，首先先找出 f_type=’货币型’的 f_id，再从 property 表中找出 pro_type=3（即为基金）且 pro_pif_id 在第一个集合中的 pro_c_id，最后从 client 表中找到 c_id 在第二个集合中的用户。代码如下：

```
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用户

本关主要是要注意聚合函数 sum()以及 limit 以及 inner join 的用法。代码如下：

```
select c_name,c_id_card,sum(pro_income) as total_income from client
    inner join property on pro_c_id=c_id and pro_status = "可用"
    --inner join:在表中至少有一个匹配项时返回行
    group by c_id
    order by total_income desc
    limit 3;
```

2.3.11 黄姓客户持卡数量

本关关键是 left join（左外连接）以及 like + 通配符%的用法。代码如下：

```
select c_id,c_name,count(b_c_id) as number_of_cards from client
        left join bank_card on c_id=b_c_id where c_name like "黄%" group by c_id
    --ta left join tb:返回 ta（左表）中匹配的行，右表不匹配的结果为 null
    order by number_of_cards desc,c_id;
```

2.3.12 客户理财、保险与基金投资总额

本关要求计算用户的投资总金额，由于资产分为三类（理财、保险、 基金），故要用三个 select 语句去分别计算这三类资产的金额。之后，通过 union all 关键字将不同的 select 结果连接（注意连接的 select 结果必须拥有相同数量的列，列也必须拥有相似的数据类型，同时，每条 select 语句中的列的顺序必须相同），得到了资产金额表后，将其与 client 表连接，用 sum 函数计算总金额即可， 但是计算时还要注意使用 ifnull 函数将空值转换成 0 值。

并且注意这里需要用到的是 union all，因为 union 不会去除重复的元组。此处由于篇幅问题，代码不予展示。

2.3.13 客户总资产

使用外连接 left join 和 ifnull 函数实现在两表连接时将无记录项置 0 的操作，并多次使用外连接和子查询统计多表项的数据和，最后在 select 语句中对多项数据进行加减操作运算以计算得出客户总资产。此处由于篇幅问题，具体的代码实现不予展示。

有一个细节值得注意，在 bank_card 表中信用卡余额即为透支金额，在进行运算时记得符号问题，如果是储蓄卡就统计 b_balance，否则统计其相反数，其实也可以是用两次查询实现，分别判断是储蓄卡还是信用卡。

2.3.14 第 N 高问题

本关同样需要使用嵌套查询，注意 limit N,M 表达式的用法，可以从第 N 条记录开始返回 M 条记录，故而在 insurance 表中数据排序后便可以找到第 N 高的产品。代码如下：

```
select i_id, i_amount
from insurance
```

```
3.where i_amount =
4.(
5.    select distinct i_amount
6.    from insurance
7.    order by i_amount desc
8.    limit 3, 1
9.)
```

2.3.15 基金收益两种方式排名

本关要求以两种方式分别实现全局名次不连续的排名和连续的排名，即考察 rank() over 以及 dense_rank() over 两种表达的区别，前者为名次不连续的排名，后者为连续的排名。代码如下：

```
1.-- (1) 基金总收益排名(名次不连续)
2.select pro_c_id, sum(pro_income) as total_revenue,
3.    rank() over(order by sum(pro_income) desc) as `rank`
4.    from property
5.    where pro_type = 3
6.group by pro_c_id
7.order by sum(pro_income) desc, pro_c_id;
8.
9.-- (2) 基金总收益排名(名次连续)
10. select pro_c_id, sum(pro_income) as total_revenue,
11.     dense_rank() over(order by sum(pro_income) desc) as `rank`
12.     from property
13.     where pro_type = 3
14. group by pro_c_id
15. order by sum(pro_income) desc, pro_c_id;
```

2.3.16 持有完全相同基金组合的用户

本关实现的大致思路为：首先创建一个名为 pro 的 CTE（公共表表达式），再从 property 表中选择 pro_type 为 3 的记录，按 pro_c_id 分组，并将每组的 pro_pif_id 按顺序连接成一个逗号分隔的字符串，接着生成一个包含 c_id 和对应的 f_id 字符串的临时表，从 CTEpro 中选择满足条件的记录对，然后自连接 CTEpro，自定义两个别名 t1 和 t2，选择 t1.c_id 小于 t2.c_id 且 t1.f_id 等于 t2.f_id 的记录对，最后返回即可。代码如下：

```
1.with pro(c_id, f_id) as (
2.    select pro_c_id c_id, group_concat(distinct pro_pif_id order by pr
  o_pif_id) f_id
3.    from property
4.    where pro_type = 3
5.    group by pro_c_id
6.)
```

```
select t1.c_id c_id1, t2.c_id c_id2
from pro t1, pro 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

本关实现的大致思路为：首先进行 select 生成派生表，对每天的交易量进行计算(pro_quantity * f_amount)，并使用 sum 函数求和，标注为 total_amount，同时可根据 datediff 函数及相关计算求出该天是 2021 年第几个工作日（命名为 workday）。之后外部的一层查询将交易量未达到一百万的记录排除，使用 row_number() 函数对按照 workday 排序后的数据生成从 1 开始的行号，命名为 rownum。我们注意到，在连续的若干交易日期中，如果交易量均超过一百万，那么 workday 和 rownum 均是连续的。因此通过计算 workday 和 rownum 的差值并按此进行分组，使用 count 函数计算每组的行数，结果大于等于 3 的那些组，就是购买基金的高峰期。

此处由于篇幅限制，本关代码不予展示。

2.3.18 至少有一张信用卡余额超过 5000 元的客户信用卡总余额

本关主要是通过嵌套查询以及 sum()函数求得总额度再进行筛选。代码如下：

```
select b_c_id, sum(b_balance) as credit_card_amount
from bank_card
where b_type = "信用卡" and b_c_id in
(
    select b_c_id
    from bank_card
    where b_type = "信用卡" and b_balance > 5000
)
group by b_c_id
order by b_c_id;
```

2.3.19 以日历表格式显示每日基金购买总金额

本关要求以日历表格式列出 2022 年 2 月余额每周每日基金购买总金额。本关需要用一次嵌套循环，先将 property 表与 fund 表连接，连接条件为 pro_pif_id = f_id，从中挑出 2022 年 2 月的数据，按照购买时间 pro_purchase_time 分组，使用 sum 计算每一交易日期的总交易金额（定义为 amount），并用 week 函数计算出该日期是 2 月的第几周，用 weekday 函数算出是星期几(定义为‘id’)。之后用外层的 select 算出所有周一至周五的基金购买总金额。代码如下：

```
select  `week` week_of_trading, --别名
    sum(if(`id` = 0, amount, null)) Monday, --如 id=0 即筛选周一的金额汇总，不是周一的情况返回 null
    sum(if(`id` = 1, amount, null)) Tuesday,
```

```
    sum(if(`id` = 2, amount, null)) Wednesday,
    sum(if(`id` = 3, amount, null)) Thursday,
    sum(if(`id` = 4, amount, null)) Friday
    from (
        select week(pro_purchase_time) - 5 `week`, weekday(pro_purchase_time) `id`, sum(pro_quantity * f_amount) amount      --减 5 是将年的周数换成二月的周数；计算交易日期是周几；计算当天购买金额
        from property
          join fund
          on pro_pif_id = f_id
        --连接 property 和 fund，匹配业务约束和基金编号
          where pro_purchase_time like "2022-02-%" and pro_type = 3
        --筛选 2022-02 且已购买基金的记录
          group by pro_purchase_time  --按购买时间分组
    ) as t  --临时表 t
group by `week`;
```

2.4 数据查询(Select)-新增

使用 MySQL 语言实现数据查询。掌握 select 及其相关语句的使用。相较于 2.3 难度提升。此部分实验已完成全部关卡，由于较为复杂，以下将逐一进行分析。

2.4.1 查询销售总额前三的理财产品

本关大致思路为：首先筛选出 property 表中 pro_type 为 1 且购买年份在 2010 和 2011 年的记录，然后与 finances_product 表进行连接，计算每年每个产品的总金额（通过购买数量乘以金额的和）。接着，对这些结果按年份进行分区，并根据总金额进行降序排名，最后选择每年总金额排名前 3 的产品记录。部分关键代码如下：

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

2.4.2 投资积极且偏好理财类产品的客户

这段 SQL 查询首先创建了两个 CTE（公共表表达式）：其中 table_cnt_1 用于计算每个 pro_c_id 对应的 pro_type 为 1 的不同 pro_pif_id 的数量，table_cnt_3 用于计算每个 pro_c_id 对应的 pro_type 为 3 的不同 pro_pif_id 的数量。然后，将这两个 CTE 与 property 表进行连接，筛选出 pro_type 为 1 且 cnt_1 大于 cnt_3 的记录，并按 pro_c_id 升序排列。此处由于篇幅问题，故代码不予展示。

2.4.3 查询购买了所有畅销理财产品的客户

本查询嵌套了两层 where not exists，首先生成畅销理财产品的表和被用户购

买的理财产品的表。内层的 where not exists 找出那些没有被购买的畅销理财产品，然后利用外层的 where not exists 查找没有漏掉购买任何一个畅销理财产品的用户（即购买了所有畅销理财产品的用户）。此处由于篇幅问题，代码不予展示。

2.4.4 查找相似的理财产品

本关大致的思路为：首先从 property 表中找到所有持有 14 号理财产品的客户及每个客户的持有数量，定义为 t1 表；使用 dense_rank() over 函数将其中的数据按持有数量排序，定义为 t2 表；从 t2 表中选出持有 14 号理财产品数量最多的前三名（但可能不止三个人），定义为 t3 表；t3 表与 property 进行自然连接后，从中选出前三名持有的理财产品（除 14 号以外），定义为 t4 表；t4 表与 property 再进行自然连接，数据按 pro_pif_id 分组，按 pro_pif_id 排序，选出 pro_pif_id 及该编号的理财产品的购买客户总人数 cc，定义为 t5 表；从 t5 表中查找 pro_pif_id，cc，以及用 dense_rank() over 函数生成的排名。此处由于篇幅问题，代码不予展示。

2.4.5 查询任意两个客户的相同理财产品数

本查询使用两个 property 表（命名为 p1 和 p2）进行自然连接，按 p1.pro_c_id 和 p2.pro_c_id 进行分组，使用 count 函数计算两人持有的相同理财产品的数量，使用 having count 满足“至少 2 种”的要求。代码如下：

```
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#去除本身
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

本关卡需要明确“相似的理财客户”的定义（比较复杂故不再赘述），先将 property 表定义为 t1，将从 property 中 select 出的 pro_type = 1 的用户 id 和业务 id 定义为 t2 表，t1 和 t2 表进行连接后，表的每一行就有了（A 客户，B 客户，他们共同拥有的产品 id）这样的数据（但注意此时 A 客户和 B 客户可能是同一个人）。之后从连接的表中 select 出不同的 A 客户作为 pac，不同的 B 客户作为 pbc，用 count 计算他们共同持有的理财产品数作为 common，将这些内容定义为 t3 表；使用 rank() over 函数按照 pac 分组，按 common 降序，pbc 升序的方式进行排名，作为 t4 表；从 t4 表中选出排名值小于 3 的相似客户即可。此处由于篇幅问题，代码不予展示。

2.5 数据的插入、修改与删除(Insert,Update,Delete)

本实训任务要求使用 MySQL 语言实现对表进行修改，即表数据的插入、修改与删除。需要掌握 Insert,Update,Delete 及其相关语句的使用。此实训关卡均已完成，以下将着重分析其中的重点关卡。

2.5.1 插入多条完整的客户信息

本关卡已完成，由于较为简单故此处不再赘述。

2.5.2 插入不完整的客户信息

本关卡已完成，由于较为简单故此处不再赘述。

2.5.3 批量插入数据

本关卡已完成，由于较为简单故此处不再赘述。

2.5.4 删除没有银行卡的客户信息

本关卡已完成，由于较为简单故此处不再赘述。

2.5.5 冻结客户资产

本关卡已完成，由于较为简单故此处不再赘述。

2.5.6 连接更新

本关需要对两个表操作，对两个表进行 update，然后通过 where 条件来进行赋值更新。代码如下：

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

2.6 视图

视图（view）是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，数据库中只存放了视图的定义，而并没有存放视图中的数据，这些数据存放在原来的表中。使用视图查询时，数据库会从原来的表中取出对应的数据。本实训要求使用 create view 创建视图，并且会利用视图进行查询。本实训关卡均已完成，由于较为简单故不展开分析。

2.7 存储过程与事务

存储过程是在大型数据库系统中，一组为了完成特定功能的 SQL 语句集，存储在数据库中。存储过程经过第一次编译后再次调用时不需要再次编译，用户通过指定存储过程的名字并给出参数（如果有）来调用存储过程。本章要求学会使用流程控制语句、游标和事务的存储过程。在本实训任务中要求学会使用流程

来控制语句/游标/事务的存储过程。

2.7.1 使用流程控制语句的存储过程

本关大致思路为：要创建一个插入斐波那契数列的存储过程，首先需要重定义 MySQL 分隔符为 $$，然后使用 create procedure 创建存储过程。在存储过程内，先用 declare 定义相关参数，并使用流程控制语句和迭代的思想来插入斐波那契数列。由于篇幅问题，在此处不予代码展示。

2.7.2 使用游标的存储过程

本关大致思路为：首先需要设置两个游标，分别用于指向护士和医生，并设置一个 NOT FOUND 的处理例程来判断是否到达末尾。在循环中进行排班工作时，每次先从游标中获取数据，接着判断是否到达末尾，如果到达末尾则需要关闭并重新开启游标以便重新读取数据。排班时确保主任医生不安排在周末，并根据当前星期几决定排班。如果是周末，判断是否是主任医生；如果是主任医生，将其存储在变量中并取出下一个医生进行排班；否则，进行正常排班。如果当前是星期一且存储变量不为空，则将主任医生放入排班。最后关闭游标。由于篇幅问题，在此处不予代码展示。

2.7.3 使用事务的存储过程

首先定义输入参数，包括申请者 ID、源卡号、接收者 ID、目标卡号、转账金额和返回代码。然后通过查询获取源卡和目标卡的信息，包括客户 ID、余额和卡类型。接着检查多种不合法情况（如申请者和卡号不匹配、信用卡转储蓄卡、储蓄卡余额不足等），如果存在不合法情况则设置返回代码为 0 并结束过程。接下来根据卡类型调整转账金额，对源卡余额进行扣款操作，对目标卡余额进行增加操作，最后设置返回代码为 1 表示转账成功。此处由于篇幅问题不在此处展示，可至附录中查看该部分代码。

2.8 触发器

触发器 (trigger) 是由事件来触发某个操作。这些事件包括 insert 语句、update 语句和 delete 语句。当数据库系统执行这些事件时，就会激活触发器执行相应的操作。本章要求为指定的表编写触发器以实现特定的业务规则。

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

对于本关，我们首先要注意我们的触发器类型是行级触发器，每插入一行就触发一次，接着我们需要对插入的数据的 type 进行 1,2,3 类型分别处理，且我们插入数据的 pro_pif_id 必须是对应表的 id，然后如果不符合条件，就进行异常处理，最后如果我们插入的数据 type 不在 1、2、3 的范围内，我们也要发

出 illegal 的异常处理。代码如下：

```
delimiter $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW
BEGIN
    IF (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;

    IF (NEW.pro_type = 1) AND (SELECT COUNT(*) FROM finances_product
 WHERE p_id = NEW.pro_pif_id) = 0 THEN
        SET @msg = CONCAT('finances product #', NEW.pro_pif_id, ' no
t found!');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
    END IF;

    IF (NEW.pro_type = 2) AND (SELECT COUNT(*) FROM insurance WHERE
i_id = NEW.pro_pif_id) = 0 THEN
        SET @msg = CONCAT('insurance #', NEW.pro_pif_id, ' not found!
');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
    END IF;

    IF (NEW.pro_type = 3) AND (SELECT COUNT(*) FROM fund WHERE f_id
= NEW.pro_pif_id) = 0 THEN
        SET @msg = CONCAT('fund #', NEW.pro_pif_id, ' not found!');

        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = @msg;
    END IF;
END$$
delimiter ;
```

2.9 用户自定义函数

本章要求掌握在 SELECT 语句中应用自定义函数。具体来说就是通过定义特定的模块化函数，满足用户特定的需求，在之后的多次调用中可以发挥很大的作用。本次实训任务已完成所有关卡。

2.9.1 创建函数并在语句中使用它

本关主要是对 create function 表达式的运用，创建函数的代码部分如下：

```
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 ;
```

调用自定义函数的代码如下：

```
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

考察 MySQL 的安全控制机制、create user 语句的使用和 rant 和 revoke 语句的使用等。具体来说就是通过自主存取控制方法，通过授予创建用户、 特定用户特定的权限和收回特定的权限等，保证数据库安全，防止数据泄露。此实训任务已完成全部关卡。

2.10.1 用户和权限

本关要求掌握 create user、grant 以及 revoke 等语句。代码如下：

```
--(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_mail, c_name, 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

这一关创建角色，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。这一关目的在于让我们熟悉角色的用法，体会其便利之处，实现代码与上一关相似。此处由于篇幅问题，代码不予展示。

2.11 并发控制与事务的隔离级别

对于并发操作，可能会产生数据不一致性，如丢失修改(lost update)，读脏数据(dirty read)，不可重复读(non-repeatable read)，幻读(phantom read)，选择合适的事务隔离级别，使得两个事务并发执行时不发生数据不一致的问题。

2.11.1 并发控制与事务的隔离级别

本关要求掌握 set session transaction isolation level 设置事务隔离等级等表达式语句。代码如下：

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

2.11.2 读脏

读脏(dirty read)，或者又叫脏读，是指一个事务(t1)读取到另一个事务(t2)修改后的数据，后来事务 t2 又撤销了本次修改(即事务 t2 以 roll back 结束)，数据恢复原值。这样，事务 t1 读到的数据就与数据库里的实际数据不一致，这样的数据被称为“脏”数据，意即不正确的数据。只有一种隔离级别可能会产生读脏，即为读不提交。

本关主要是使用 set @n = sleep(时间)来进行时间上的控制。由于篇幅原因，在此处不再赘述。

2.11.3 不可重复读

不可重复读(unrepeatable read)，是指一个事务(t1)读取到某数据后，另一个事务(t2)修改了该数据，事务 t1 并未修改该数据，但当 t1 再次读取该数据时，发现两次读取的结果不一样。不可重复读产生的原因，是事务 t1 的两次读取之间，有另一个事务修改了 t1 读取的数据。有两种级别可以发生不可重复读的现象。具体来说，本实验需要通过更加精细化的时间控制，实现对于两个事务不可重复读

现象的重现。此处由于篇幅问题，代码不予展示。

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。幻读与不可重复读之间的区别就是幻读是发现数据莫名其妙地增加或减少，而不可重复读现象是数据在两次读取时，发现读取的内容不一致的现象。幻读产生的原因，是事务 t1 的两次读取之间，有另一个事务 insert 或 delete 了 t1 读取的数据集。此处由于篇幅问题，代码不予展示。

2.11.5 主动加锁保证可重复读

MySQL 提供了主动加锁的机制，使得在较低的隔离级别下，通过加锁，以实现更高级别的一致性。SELECT 语句支持 for share 和 for update 短语，分别表示对表加共享(Share)锁和写(write)锁，共享锁也叫读锁，写锁又叫排它锁。此处添加共享锁的代码如下：

```
1.select tickets from ticket where flight_no = 'MU2455' for share;
```

2.11.6 可串行化

多个事务并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同。两个事务 t1,t2 并发执行，如果结果与 t1→t2 串行执行的结果相同，或者与 t2→t1 串行执行的结果相同，都是正确的(可串行化的)。

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。需要通过时间控制，交错执行程序，实现可串行化调度。此处由于篇幅问题，代码不予展示。

2.12 备份+日志：介质故障与数据库恢复

MySQL 利用备份、日志文件实现恢复，在这一关中我们需要了解并掌握之本的恢复机制和备份与恢复工具。本实训任务已完成全部关卡，在此由于篇幅问题，代码不予展示分析。

2.13 数据库设计与实现

本章将从实际出发，从按需求建表到设计 E-R 图并进一步转换为关系模型，再到使用建模工具对数据库进行建模，了解一个数据库从概念模型到具体实现的步骤。此外，在工程认证中，制约因素分析与设计和工程师责任及其分析也是必不可少的内容。本实训任务已完成全部关卡，下面将逐个分析关卡。

2.13.1 从概念模型到 MySQL 实现

本关按照要求创建表即可，注意约束条件，由于篇幅问题此处不予代码展示。

2.13.2 从需求分析到逻辑模型

根据关卡要求已画出 E-R 图如下图 2-13-1 所示。图片存放 URL 地址为：https://github.com/YanShuq1/2024SQL_Experiment/blob/main/E-R 图/ersolution.jpg?raw=true。

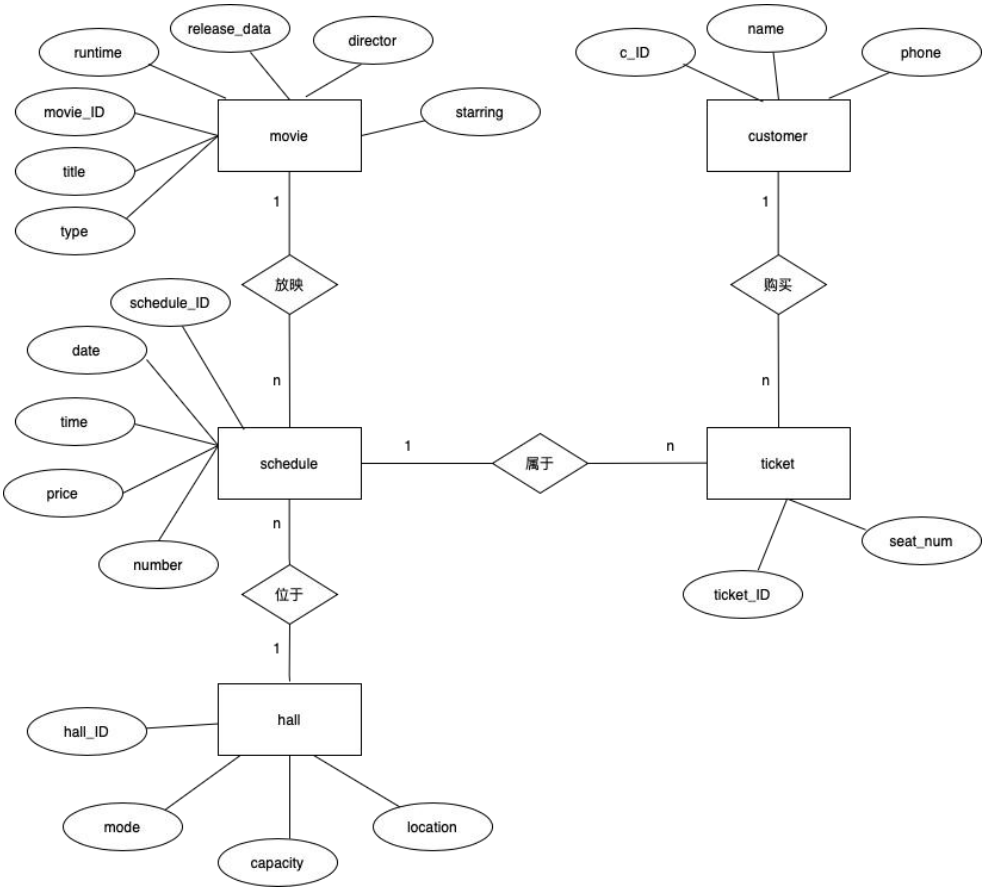

图 2-13-1 E-R 图

据此建立五个关系模式：

movie(movie_ID,title,type,runtime,release_data,director,starring),primary key:(movie_ID),foreign key:(schedule_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), primary key:(schedule_ID),foreign key:(hall_ID,movie_ID)

ticket(ticket_ID,seat_num), primary key:(ticket_ID),foreign key:(c_ID,schedule_ID)

2.13.3 建模工具的使用

本关由于篇幅问题，代码不予展示。

2.13.4 制约因素分析与设计

在实际工程中设计数据库时，除了满足客户需求外，还必须综合考虑社会、健康、安全、法律、文化及环境等制约因素。

社会层面：数据库设计应考虑用户友好性、数据透明性和隐私保护，确保公平性和无歧视性。设计要基于社会责任，满足不同背景用户的需求。

健康层面：确保健康相关数据的准确性和实时性，减少用户心理负担，提供简洁的操作界面和清晰的健康数据反馈，以帮助用户有效管理健康。

安全层面：设计方案必须保障数据安全与信息安全，采用数据加密、访问控制和多因素身份验证等措施，防止数据泄露和非法访问。

法律层面：设计必须合法合规，遵守相关数据保护法规，如 GDPR 和 HIPAA，保障用户权益，不利用数据库进行非法活动。

文化层面：设计应尊重不同文化背景，提供多语言支持和本地化服务，遵守工程职业道德和规范，维护用户权益。

环境层面：设计应优化数据库性能，减少能耗，采用绿色计算技术和云计算支持，考虑可持续发展原则，降低对环境的影响。

2.13.5 工程师责任及其分析

在产品实现过程中，社会、安全、法律及文化等因素对解决方案起到了制约和调整作用。工程师应承担一定的生态伦理责任，准确有效地说明新建工程或新技术可能带来的后果，避免对社会和生态环境的危害；同时，应承担职业伦理责任，秉持追求真理、客观求实、诚实公平的精神；并承担保护公众安全、健康和福祉的责任，不仅要遵守设计规则和标准，还应考虑产品或技术在市场上的效用。

工程师在设计、实现和维护数据库过程中，需综合考虑前述各种制约因素，设计出既能满足用户需求，又能在其他方面达到最优的数据库。同时，工程师必须遵守法律，谨慎行事，不可投机取巧，不走歪门邪道。尤其在信息技术行业，由于数据库缺陷造成的损失可能是巨大的。

此外，中国的工程师还需肩负技术发展和提升国家综合实力的责任，不断进行创新设计，推动国家整体进步。通过综合考虑社会、安全、法律及文化等因素，工程师可以提供安全、可靠、高效的数据库解决方案，满足用户需求并促进社会和技术的可持续发展。

2.14 数据库应用开发(JAVA 篇)

JDBC（Java DataBase Connectivity，java 数据库连接）是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。JDBC 提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序。本实训关卡均已完成，以下进行分析。

2.14.1 JDBC 体系结构和简单的查询

本关由于篇幅问题，代码不予展示。

2.14.2 用户登录

本关由于篇幅问题，代码不予展示。

2.14.3 添加新客户

编程完成向 client(客户表)插入记录的方法。已知用户的 id、姓名、邮箱、身份证号、电话和登陆密码，将上述信息与已连接的数据库的 Connection 对象传入方法中。可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的插入操作。关键部分代码如下：

```
1.public static int insertClient(Connection connection,int c_id, String c_name, String c_mail,String c_id_card, String c_phone,  String c_password) throws SQLException{
2.        String sql = "INSERT INTO client(c_id,c_name,c_mail,c_id_card,c_phone,c_password) "+"VALUES(?,?,?,?,?,?)";
3.      PreparedStatement statement = connection.prepareStatement(sql);
4.        statement.setInt(1,c_id);
5.        statement.setString(2,c_name);
6.        statement.setString(3,c_mail);
7.        statement.setString(4,c_id_card);
8.        statement.setString(5,c_phone);
9.        statement.setString(6,c_password);
10.         int result = statement.executeUpdate();
11.         statement.close();
12.         return result;
```

2.14.4 银行卡销户

编写一个能删除指定客户编号的指定银行卡号的方法。具体方法与插入类似。已知用户的 id 和银行卡号，将上述信息与已连接的数据库的 Connection 对象传入方法中。可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现

对于元组的删除操作。

2.14.5 客户修改密码

编写修改客户登录密码的的方法。具体方法也与插入类似。首先通过一系列方法获取需要修改的用户，同时获得其旧密码，进一步对于旧密码进行更新，将用户的邮箱、旧密码和新密码与已连接的数据库的 Connection 对象传入方法中。可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的更新操作。

2.14.6 事务与转账操作

使用 JDBC 的事务处理编写一个银行卡转账方法。需要传入已连接数据库的 Connection 对象、转出账号、转入账号和转账金额。首先设置隔离级别为 REPEATABLE READ，具体实现为 connection. SetTransactionIsolation(4)。然后，进行转出账号扣账，转入账号还账，为储蓄卡时入账。然后进行操作合法性的检验，如果操作不合法，例如账号不存在，扣账金额不足等，则进行事务回滚，具体实现为 connection.rollback()，且置返回值为 false。其他情况下则进行事务提交，具体实现为 connection.commit()，且置返回值为 true。

2.14.7 把稀疏表格转为键值对存储

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。具体来说，首先需要查询稀疏表中的所有元组，进行学生学号和各科成绩的提取，然后将其填入键值对表中。 然后，将已连接的数据库的 Connection 对象和键值对传入方法中。可以使用 PreparedStatement 接口进行 SQL 语句的传递。将上述用户的相关信息插入到 SQL 语句中，使用该接口的 executeUpdate()方法，实现对于元组的插入操作。

3 课程总结

在本次数据库系统原理实践课程中，我通过在头歌平台上的一系列实训任务，全面深入地学习了 MySQL 数据库的使用与操作。这些实训任务涵盖了数据库管理的各个方面，包括数据定义、数据操作、复杂查询、视图使用、存储过程编写、安全性控制以及数据库设计与实现等实训任务。在整个课程中，我逐一完成了所有实训任务，并成功掌握了相关知识和技能。

对于各个实训任务，具体主要工作如下：完成了数据库、表、完整性约束条件的定义相关的所有实训任务，掌握了如何使用数据定义语言 DDL 以及如何定义约束条件；完成了表结构和完整性约束的修改相关的所有实训任务，掌握了如何利用 SQL 语句实现对表结构和约束条件的修改；完成了数据查询相关的所有实训任务，对于基本的数据查询以及数据查询过程中所用到的函数、方法、技巧有了很好的掌握；完成了数据的插入、删除、修改、连接更新相关的所有实训任务；完成了创建视图与基于视图的查询，进一步理解了模式和外模式之间的映像，掌握了应用程序是如何基于视图这一层次进行查询；完成了存储过程与事务相关的部分实训任务，掌握了三种存储过程的具体结构和实现方法；完成了触发器相关的实训任务，掌握了根据要求完成触发器的设计；完成了用户自定义函数相关的实训任务，掌握了根据要求完成自定义函数的设计；完成了创建用户、角色以及权限授予相关的所有实训任务，掌握了自主存取方式对于用户权限的设置和对于数据库的保护；完成了事务并发控制与隔离级别相关的所有实训任务，加深了对读脏、不可重复读、幻读、可串行化的理解，掌握了对于数据库事务的加锁操作；完成了使用备份和日志文件实现数据恢复相关的所有实训任务，对于数据库恢复相关技术有更加深入的理解；完成了数据库设计相关的所有实训任务，掌握了概念模型设计、关系模式设计、建模实现设计；完成了 JAVA 数据库应用开发的所有实训任务，掌握了 JDBC 体系实现数据库应用设计。

通过本次数据库系统原理实践课程，我在理论知识和实际操作能力上都得到了显著提升。日后我将加强对数据库性能优化和安全性控制的研究，提升数据库管理水平。积极参与更多实际项目，积累经验，提升实际操作能力。

总之，通过本次课程，我不仅掌握了数据库系统的基本操作方法和高级功能，还为未来的学习和工作奠定了坚实的基础。在今后的学习中，我将继续努力，进一步提升自己的专业水平和实践能力。

附录

2.7.3 使用事务的存储过程相关代码

```
use finance1;
-- 在金融应用场景数据库中，编程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。
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)
pro:
BEGIN
    declare src_id, dst_id int;
    declare src_type, dst_type char(20);
    declare src_balance, dst_amount numeric(10, 2) default amount;

    select b_c_id, b_balance, b_type
    into src_id, src_balance, src_type
    from bank_card
    where b_number = source_card_id;
    select b_c_id, b_type
    into dst_id, dst_type
    from bank_card
        where b_number = dest_card_id;#筛选
    if src_id != applicant_id or dst_id != receiver_id  or (src_type = "信用卡" and dst_type = "储蓄卡") or (src_type = "储蓄卡" and src_balance < amount) then#不合法情况
        set return_code = 0;
        leave pro;
    end if;
    if src_type = "信用卡" then
        set amount = -amount;
    end if;
    if dst_type = "信用卡" then
        set dst_amount = -dst_amount;
    end if;
    update bank_card set b_balance = b_balance - amount
        where b_number = source_card_id;--更新原始余额
```

```
38.     update bank_card set b_balance = b_balance + dst_amount
39.         where b_number = dest_card_id;--更新目的余额
40.     set return_code = 1;
41. END$$
42. delimiter ;
```