华中科技大学

数据库系统原理实践报告

专　　业：	计算机
班　　级：	CS2201
学　　号：	U202215357
姓　　名：	王文涛
指导教师：	瞿彬彬

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 1~6、8~11、13~14 实训任务，下面将重点针对其中的第 3~4、8、11、13~14 实训任务阐述其完成过程中的具体工作。

2.1 基于金融应用的数据查询(Select)

本章采用的是某银行的一个金融场景应用的模拟数据库，根据提供的数据库中表，表结构以及所有字段的说明，实现不同的查询目标。

2.1.1 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。

此关需要用到嵌套查询。范围为表 client，条件为存在该用户购买了保险和购买了基金的记录。

代码如下：

```
select c_name, c_mail, c_phone
from client
where exists(
  select 1 from property where c_id = pro_c_id and pro_type = 2) # 购买了保险
and
exists(
  select 1 from property where c_id = pro_c_id and pro_type = 3) # 购买了基金
order by c_id
```

2.1.2 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。

这关需要查找众数，需要用到子查询，分组统计与 COUNT()函数 ALL 关键词。先使用 COUNT()函数并分组，再根据分组在 having 语句中使用子查询来得到次数。

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
);
```

2.1.3 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。

本关需要用到 like 语句进行模糊匹配和 not exists 引导的相关子查询。

代码如下：

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

2.1.4 购买了货币型基金的客户信息

本关任务：购买了货币型基金的客户信息。

本关需要用到 in 引导的非相关子查询。

代码如下：

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

2.1.5 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。

本关需要使用派生表。

首先先从 property 表中将商品属性为可用的元组的收益相加并按照用户 id 分组，得到有总收益和用户 id 的派生表 c_income。接着将派生表和 client 表自然连接，按照总收益降序排序，并使用 limit 取前三位。

代码如下：

```
select c_name, c_id_card, total_income
from client,
(select sum(pro_income) as total_income, pro_c_id from property where pro_st
atus = "可用"
  group by pro_c_id) c_income
where c_id = pro_c_id
order by total_income desc
limit 0, 3;
```

2.1.6 黄姓客户持卡数量

本关任务：查询黄姓用户的编号、名称、办理的银行卡的数量。

本关需要用到外连接，因为需要显示所有黄姓用户的信息。将 client 表左外连接 bank_card 表。

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
order by number_of_cards desc, c_id;
```

2.1.7 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。

本关需要使用外连接和表的合并。

首先分别将理财产品表(finances_product)、保险表(insurance)和基金表(fund)和资产表(property)连接，分别使用 sum 函数求出该部分的资金总和并命名为相同的名字 total。为了避免金额信息丢失，使用 union all 进行连接，得到派生表 proper_total，接着使用左外连接连接 client 表和 proper_total，查询后根据 total_amount 降序排序。

代码如下：

```
S
elect c_name, c_id_card, ifnull(sum(proper_total.total),0) as total_amount
from client left join(
    select pro_c_id, sum(p_amount*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(i_amount*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(f_amount*pro_quantity) as total
    from fund, property
    where f_id=pro_pif_id and pro_type=3
    group by pro_c_id) proper_total
    on c_id=pro_c_id
group by c_id
order by total_amount desc;
```

2.1.8 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

本关需要使用 rank() 函数来进行排名不连续的排序和 dense_rank()函数来进行排名连续的函数。

代码如下：

```
  --(1) 基金总收益排名(名次不连续)
select pro_c_id, total_revenue, rank() over(order by total_revenue desc) as "rank"
from(
  select pro_c_id, sum(pro_income) as total_revenue from property where pro_type = 3 group by pro_c_id
) a
order by total_revenue desc, pro_c_id;
--(2) 基金总收益排名(名次连续)
select pro_c_id, total_revenue, dense_rank() over(order by total_revenue desc) as "rank"
from(
  select pro_c_id, sum(pro_income) as total_revenue from property where pro_type = 3 group by pro_c_id
) a
order by total_revenue desc, pro_c_id;
```

2.1.9 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。

本关需要用到 not exists 和 not in 引导的多层嵌套查询。

使用 not exists 和 not in 双重否定来表示∀关系。条件首先有 A.pro_c_id < B.pro_c_id 避免客户对重复，接着要求编号为 A 的客户持有的基金， 编号为 B 的客户也持有，接着要求编号为 B 的客户持有的基金， 编号为 A 的客户也持有；最后要求编号为 A 的客户持有任意基金。最后对结果去重即可得到答案。

```
select distinct A.pro_c_id as c_id1, B.pro_c_id as c_id2
from property A, property B
where A.pro_c_id < B.pro_c_id
and not exists(
  select 1 from property C where A.pro_c_id = C.pro_c_id and C.pro_type = 3 and C.pro_pif_id not in (
    select pro_pif_id from property D where D.pro_c_id = B.pro_c_id and D.pro_type = 3
  )
)
and not exists(
  select 1 from property C where B.pro_c_id = C.pro_c_id and C.pro_type = 3 and C.pro_pif_id not in (
    select pro_pif_id from property D where D.pro_c_id = A.pro_c_id and D.pro_type = 3
  )
)
and exists(
  select 1 from property C where C.pro_c_id = A.pro_c_id and C.pro_type = 3

)
```

2.2 数据查询(Select)-新增

基于上一个实训任务的场景实现不同的查询任务。

2.2.1 查询销售总额前三的理财产品

本关任务：查询销售总额前三的理财产品。

首先在 property 表中选出对应年份的资产表，然后分资产类型与对应的资产进行自然连接，统计出销售总额。最后根据输出要求采用 rank 函数对销售额进行编号，同时使用 limit 子句限制输出条数。由于需要对 2010 年和 2011 年分开

统计，因而需要将两个年份单独查询后取并集。

代码如下：

```
select pyear,
rank() over(partition by pyear order by sumamount desc) 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(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)
) a
order by pyear asc, rk asc, p_id asc;
```

2.2.2 查询购买了所有畅销理财产品的客户

本关任务：查询购买了所有畅销理财产品的客户。

在 property 表中通过 count 函数统计购买人数（distinct pro_c_id），以 p_id 进行分组，得到畅销产品编号。购买了所有畅销产品等效于不存在一个畅销产品该人没买，因而使用双重否定，两层 not exists 子句即可筛选出购买了全部畅销产品的人。

代码如下：

```
select distinct pro_c_id
   from property p1
   where not exists(
     select 1 from(
       select pro_pif_id from property where pro_type = 1 group by pro_pif_id having count( * ) > 2
     ) hot where hot.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
     )
   )
   order by pro_c_id asc;
```

2.2.3 查找相似的理财产品

本关任务：对于某款理财产品 A，可找到持有 A 数量最多的“3”个（包括所有持有相同数量的客户，因此如有 3 个并列第一、1 个第二、一个第三，则排列结果是 1,1,1,2,3）客户，然后对于这“3”个客户持有的所有理财产品（不包含产品 A 自身），每款产品被全体客户持有总人数被认为是和产品 A 的相似度，若有相似度相同的理财产品，则为了便于后续处理的确定性，则这些相似度相同的理财产品间按照产品编号的升序排列。按照和产品 A 的相似度，最多的“3”款（同上理，前 3 名允许并列的情况，例如排列结果是 1,2,2,2,3）理财产品，就是产品 A 的相似的理财产品。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）。结果按照相似度值降序排列，相同相似度的理财产品之间则按照产品编号的升序排列。

根据本关相似的定义，首先通过 dense_rank()按照持有 14 号理财产品数量将客户排序，将产品名作为派生表 fin_rk，接着查找与 14 号产品相似的其他理财产品。然后计算这些理财产品的相似度，按照相似度排序。

代码如下：

```
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
    ) fin_rk
  )
)
group by pro_pif_id
order by cc desc, pro_pif_id asc;
```

2.2.4 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。

查找任意 id1、id2 两个客户，统计二人有相同理财产品的元组，根据一对 id1, id2 进行分组，最后只输出相同理财产品数大于等于 2 的结果

代码如下：

```
select id1 pro_c_id, id2 pro_c_id, count( * ) as total_count
from(
  select p1.pro_c_id id1, p2.pro_c_id id2, p1.pro_pif_id from property p1, p
roperty p2 where p1.pro_c_id < > p2.pro_c_id and p1.pro_type = 1 and p2.pro_
type = 1 and p1.pro_pif_id = p2.pro_pif_id
) a
group by id1, id2
having count( * ) >= 2
order by id1
```

2.3 触发器

本任务要求为表编写一个触发器，以实现完整性业务规则

2.3.1 为投资表 property 实现业务约束规则

本关任务：为表 property(资产表)编写一个触发器，以实现以下完整性业务规则：

如果 pro_type = 1，则 pro_pif_id 只能引用 finances_product 表的 p_id；

如果 pro_type = 2，则 pro_pif_id 只能引用 insurance 表的 i_id；

如果 pro_type = 3，则 pro_pif_id 只能引用 fund 表的 f_id；

pro_type 不接受(1,2,3)以外的值。

为表 property 声明 BEFORE INSERT 类型的触发器。先声明字符数组 msg。首先判断 pro_type 的值是否合法，接着根据 pro_type 的值判断不同的表，如果不存在该类型的投资产品则报相应错误。如果报错信息为空则说明没有错误。

代码如下：

```
delimiter $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW
BEGIN
declare msg varchar(128);
if new.pro_type = 1 and not exists (select * from finances_product where p_i
d = new.pro_pif_id) then
    set msg = concat("finances product #",new.pro_pif_id," not found!");
```

```
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!");
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!");
elseif new.pro_type not in (1,2,3) then
    set msg = concat("type ",new.pro_type," is illegal!");
end if;
if msg is not null then
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
end if;
END$$
delimiter ;
```

2.4 并发控制与事务的隔离级别

本任务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。

2.4.1 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。

第一个事务等待，第二个事务修改而未 commit 的时对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。

代码如下：

```
-- 事务 1:
use testdb1;
## 请设置适当的事务隔离级别
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;
```

2.4.2 幻读

本关任务：在 repeatable read 事务隔离级别，构造两个事务并发执行时，发生“幻读”现象。

事务 2 在等待了 1 秒后进行插入，让事务 1 在一开始读，然后在两秒后再读，此时会因为事务 2 的 commit 而发生幻读。

代码如下：

```
-- 事务 1（采用默认的事务隔离级别- repeatable read）：
use testdb1;
select @@transaction_isolation;
start transaction;
## 第 1 次查询余票超过 300 张的航班信息
select * from ticket where tickets > 300;
set @n = sleep(2);
-- 修改航班 MU5111 的执飞机型为 A330-300:
update ticket set aircraft = 'A330-300' where flight_no = 'MU5111';
-- 第 2 次查询余票超过 300 张的航班信息
select * from ticket where tickets > 300;
commit;
```

2.4.3 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

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。

代码如下：

```
-- 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;
```

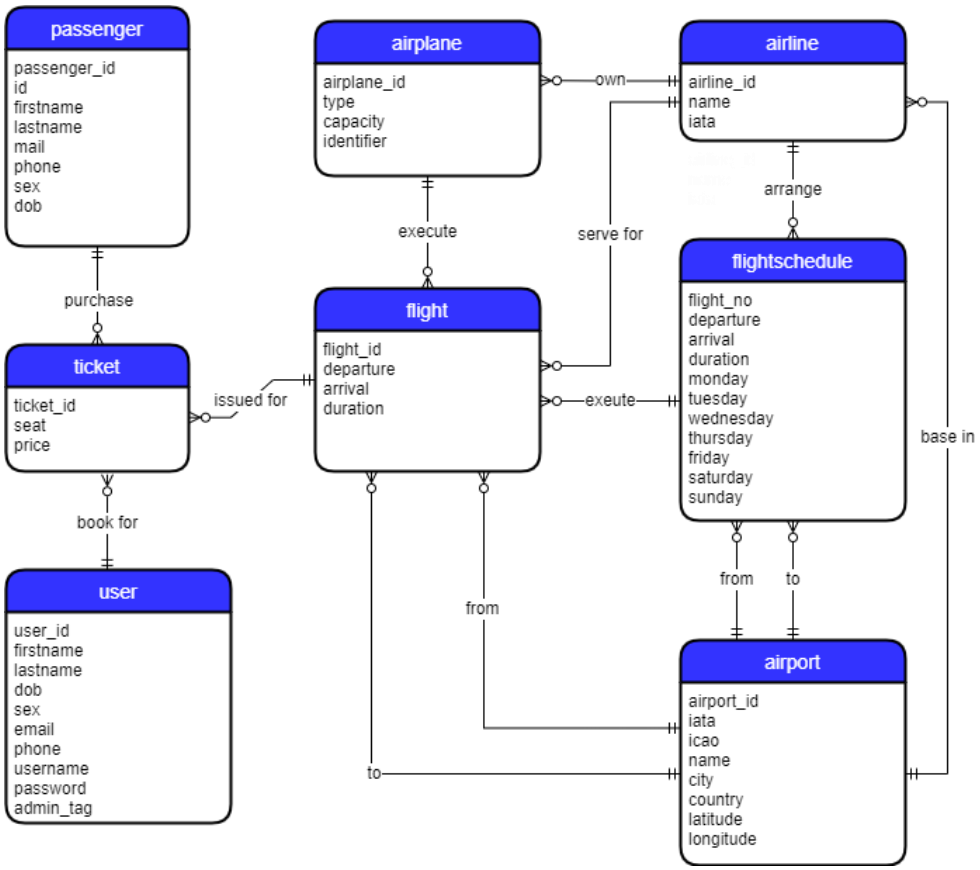
```
-- 第 3 次查询所有航班的余票，发生在事务 2 提交后
set @n = sleep(1);
select * from ticket;
-- 事务 2:
use testdb1;
set session transaction isolation level read uncommitted;
start transaction;
set @n = sleep(1);
# 在事务 1 的第 1，2 次查询之间，试图出票 1 张(航班 MU2455):
update ticket set tickets = tickets-1 where flight_no = "MU2455";
commit;
```

2.5 数据库设计与实现

本任务需要根据应用场景业务需求描述，通过完成 ER 图，转换成关系模式，最终生成 sql 代码。

2.5.1 从概念模型到 MySQL 实现

本关任务：是根据已建好的逻辑模型，完成 MySQL 的实现。逻辑 ER 图如图 2.*.1.1 所示。应用背景如附录 2 所示。



根据提供的信息，依次创建用户(user)、旅客(passenger)、机场(airport)、航空

公司(airline)、民航飞机(airplane)、航班常规调度表(flightschedule)、航班表(flight)、机票(ticket)八张表，并写下属性和约束。为了方便测试和改动，在创建每个表之前加上 drop table if exists table_name。

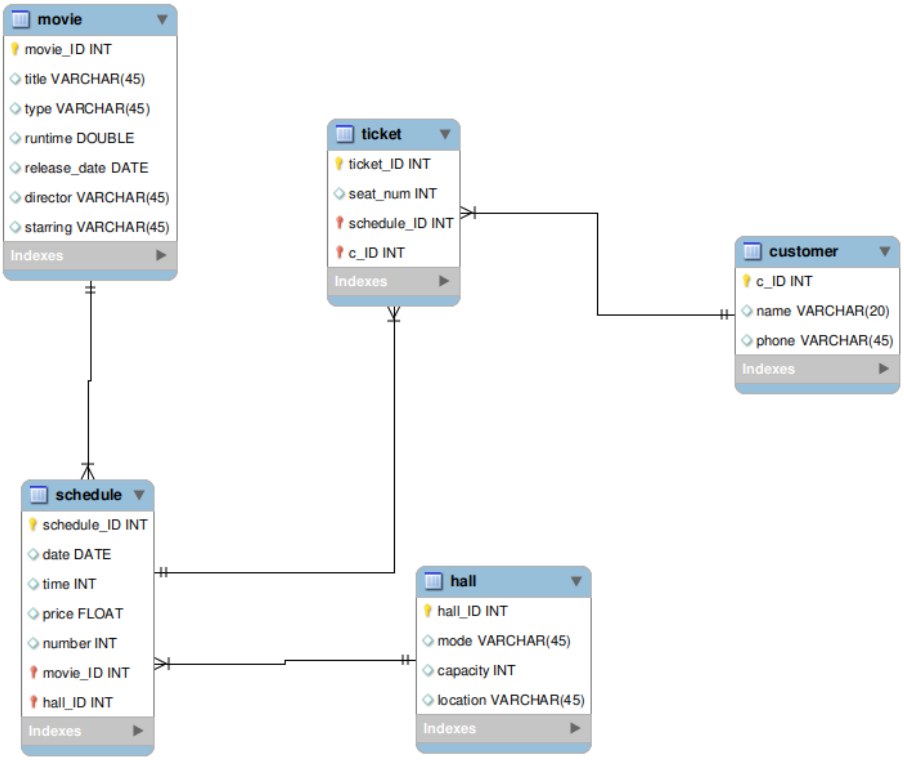
由于代码过长，这里不展示。

2.5.2 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。

我选择使用 mysql workbench 来创建 ER 图，并将图片上传到 hithub 仓库中以便提交。

根据描述创建实体，确定属性和约束，得到 ER 图如下。



关系模式如下：

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)

2.5.3 建模工具的使用

本关任务：将一个建好的模型文件，利用 MySQL Workbench 的 forward engineering 功能，自动转换成 SQL 脚本。

本关只需使用 MySQL Workbench 的 forward engineering 功能即可，因结果太长，在这里不予展示。

2.5.4 制约因素分析与设计

在现实生活中，可能会出现各种制约因素。

例如在影院管理系统中，为了避免重复售票或座位冲突，需确保数据的准确性和实时性，可以通过设置较高的事务隔离级别来防止丢失修改，读脏等问题。又比如机票订票系统，机票可以由其他人代替购买，因而在购买机票信息不仅需要记录乘机人的信息，还需要记录购票人的信息。还需要允许用户在一定条件下退票或改签，确保用户的权益。同时我们需要保护用户个人信息和购票记录的隐私，需要区分普通用户和管理员权限。普通用户只能查看影片信息、购票和退票改签；管理员则能管理影片信息、放映计划、座位信息和用户数据等。

2.5.5 工程师责任及其分析

从社会因素看，任务的解决过程需要考虑到社会的需求和期望。例如，在影院管理系统或机票订票系统的设计中，应确保系统的易用性、可靠性和可访问性，以满足广大用户群体的需求。同时，系统的运营和维护也需要考虑对社会的贡献，如提供就业机会、促进经济发展等。

从健康因素看，在设计和实施系统时，工程师需要评估系统可能带来的健康风险。例如，在影院管理系统中，应确保影院的通风、照明和座椅设计符合健康标准，以减少观众的健康问题。

从安全因素看，数据安全性：在任务解决过程中，工程师需要确保系统数据的安全性。这包括用户个人信息的保护、交易数据的加密和传输安全等。避免数据泄露或非法访问对用户造成损失，甚至引发社会安全问题。

从法律因素看，解决方案必须遵守相关法律法规的要求。这包括数据保护法、隐私法、版权法等与系统设计、运营和数据处理相关的法律法规。

从文化因素看，工程师应尊重并适应不同文化背景的用户需求。可能需要支持多种语言、习俗和支付方式等。工程师需要确保系统在不同文化环境中的适用性和可接受性。还可以通过系统设计和功能开发来传承和弘扬优秀文化。例如在影院管理系统中可以加入对本土电影的推广和支持。

基于上述分析，工程师应承担以下社会责任，保障用户权益，促进社会进步，遵守法律法规，关注健康安全：在设计和实现系统时考虑健康和安全因素，确保用户和系统本身的安全性和健康性。尊重文化差异。这样才能设计出既符合技术要求又符合社会责任的高质量解决方案。

2.6 数据库应用开发(JAVA 篇)

本任务需要掌握 JDBC 的体系结构，JDBC 的核心组件和 JDBC 的使用步骤。

2.6.1 JDBC 体系结构和简单的查询

本关任务：查询金融应用场景数据库 finance 的 client 表(客户表)中邮箱不为空的客户信息，列出客户姓名，邮箱和电话.

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。

代码如下：

```
1.  Class.forName("com.mysql.cj.jdbc.Driver");
2.        connection = DriverManager.getConnection("jdbc:mysql://127.0.0.1:3306/
    finance?useUnicode=true&characterEncoding=UTF8&useSSL=false&serverTimezone=U
    TC", "root", "123123");
3.        statement = connection.createStatement();
4.        resultSet = statement.executeQuery("SELECT c_name, c_mail, c_phone FRO
    M client WHERE c_mail IS NOT NULL");
5.        System.out.println("姓名\t 邮箱\t\t\t\t 电话");
6.        while (resultSet.next()) {
7.          System.out.println(resultSet.getString("c_name") + "\t" + resultSet.
    getString("c_mail") + "\t\t" + resultSet.getString("c_phone"));
8.        }
```

2.6.2 银行卡销户

本关任务：编写一个能删除指定客户编号的指定银行卡号的方法。

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。

代码如下：

```
1.  public static int removeBankCard(Connection connection, int b_c_id, String b
    _number) {
2.    int result = 0;
3.    try {
4.      String sql = "DELETE FROM bank_card WHERE b_c_id = " + b_c_id + " AND b_
    number = '" + b_number + "';";
5.      Statement statement = connection.createStatement();
6.      result = statement.executeUpdate(sql);
7.    } catch (SQLException e) {
8.      e.printStackTrace();
9.    } finally {
10.     return result;
11.   }
12. }
```

2.6.3 客户修改密码

本关任务：编写修改客户登录密码的的方法。

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。

```
1.  public static int passwd(Connection connection, String mail, String password
    , String newPass{
2.      int result = 0;
3.      try{
4.          String sqlSelect = "select c_password from client where c_mail = '"+
    mail+"';";
5.          String sqlUpdate = "update client set c_password = '"+newPass+"' whe
    re c_mail = '"+mail+"';";
6.          Statement statement = connection.createStatement();
7.          ResultSet resultSet = statement.executeQuery(sqlSelect);
8.  resultSet.next()) {
9.              String oldPassword = resultSet.getString("c_password");
```

```
            if(oldPassword.equals(password)) {
                int state = statement.executeUpdate(sqlUpdate);
                if(state == 1)  result = 1;
                else result = -1;
            }
            else  result = 3;
        }
        else  result = 2;
        }   catch(SQLException e){
        result = -1;
        e.printStackTrace();
    } finally {
        return result;
    }
}
```

2.6.4 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。

代码如下：

```
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);
```

```
16.     preparedStatement.executeUpdate();
17.     preparedStatement = connection.prepareStatement(sql);
18.     double rcv_amount = resultSet.getString("b_type").
19.     equals("信用卡") ? -amount : amount;
20.     preparedStatement.setDouble(1, rcv_amount);
21.     preparedStatement.setString(2, destCard);
22.     preparedStatement.executeUpdate();
23.     return true;
24.   } catch (SQLException e) {
25.     e.printStackTrace();
26.   }
27.   return false;
28. }
29.  }
```

2.6.5 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，这样可以直接丢失没有值列。

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。

代码如下：

```
1.  public static int insertSC(Connection connection, int sno, String col_name,
    int col_value) {
2.    int result = 0;
3.    try {
4.      Statement statement = connection.createStatement();
5.      String sql = "insert into sc values(" + sno + ", '" + col_name + "', " +
     col_value + ");";
6.      result = statement.executeUpdate(sql);
7.    } catch (SQLException e) {
8.      e.printStackTrace();
9.    } finally {
10.     return result;
11.   }
12. }
13. public static void main(String[] args) throws Exception {
14.         Class.forName(JDBC_DRIVER);
15.         Connection connection = DriverManager.getConnection(DB_URL, USER, PA
    SS);
16.         ResultSet resultSet = null;
```

```
        String[] Subject = {"chinese", "math", "english", "physics", "chemistry",
"biology", "history", "geography", "politics"};
        try {
            resultSet = connection.createStatement().executeQuery(
"select * from entrance_exam");
            while (resultSet.next()) {
                int sno = resultSet.getInt("sno"), score;
                for (String subject : Subject) {
                    score = resultSet.getInt(subject);
                    if (!resultSet.wasNull())
                        insertSC(connection, sno, subject, score);
                }
            }
        } catch (SQLException e) {
          e.printStackTrace();
          }
    }
```

3 课程总结

在本次精心设计的实验课程中，我们深入学习了 MySQL 语言，通过一系列实践任务，不仅熟练了数据库的建立、修改、查询等基础操作，还接触到了安全性保障、并发控制以及储存管理和索引管理等较为复杂的理论，更是学习了 Java 环境下的数据库应用开发实践，全面而系统地提升了我的数据库技能与问题解决能力。

在实验过程中，数据查询的任务最多，难度也比较大，是耗费时间最多的一个部分。虽然比较艰难，但通过与同学积极交流学习，逐渐解决了难题，这一过大大加深了我对 SQL 中 select 用法的理解，也锻炼了我的逻辑思维能力。在完成一些任务时，遇见很多在课堂内没有见过的知识，比如说查找中的编号，事务存储过程等，我在网上搜集资料，深入研究，掌握了这些知识，这一经历无疑极大地增强了我的自学能力和解决问题的韧性。

这个过程中，我还学会 mysql-workbench 的使用，学会了使用 mysql-workbench 来绘制 ER 图，以及 ER 图和 sql 代码的转换。

此外，通过 Java 环境下的 MySQL 应用开发实践，我掌握了如何将 MySQL 与 Java 集成，可以用更加简单的方式实现较复杂的功能。

总而言之，本次实验课程以其丰富的内容、精心的设计、以及老师们无私的指导，为我们提供了一次难得的学习与成长机会。它不仅让我熟练掌握了 MySQL 的各类操作，更在潜移默化中培养了我的问题解决能力、团队协作精神和持续学习的热情。在此，我衷心感谢课程组老师们的辛勤付出。