

**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 1](#_Toc171097418)

[2 任务实施过程与分析 2](#_Toc171097419)

[2.1 基于金融应用的数据查询(Select) 2](#_Toc171097420)

[2.1.1 既买了保险又买了基金的客户 2](#_Toc171097421)

[2.1.2 商品收益的众数 2](#_Toc171097422)

[2.1.3 未购买任何理财产品的武汉居民 3](#_Toc171097423)

[2.1.4 购买了货币型基金的客户信息 3](#_Toc171097424)

[2.1.5 投资总收益前三名的客户 3](#_Toc171097425)

[2.1.6 黄姓客户持卡数量 4](#_Toc171097426)

[2.1.7 客户理财、保险与基金投资总额 4](#_Toc171097427)

[2.1.8 基金收益两种方式排名 5](#_Toc171097428)

[2.1.9 持有完全相同基金组合的客户 6](#_Toc171097429)

[2.2 数据查询(Select)-新增 6](#_Toc171097430)

[2.2.1 查询销售总额前三的理财产品 6](#_Toc171097431)

[2.2.2 查询购买了所有畅销理财产品的客户 7](#_Toc171097432)

[2.2.3 查找相似的理财产品 8](#_Toc171097433)

[2.2.4查询任意两个客户的相同理财产品数 8](#_Toc171097434)

[2.3 触发器 9](#_Toc171097435)

[2.3.1为投资表property实现业务约束规则 9](#_Toc171097436)

[2.4并发控制与事务的隔离级别 10](#_Toc171097437)

[2.4.1读脏 10](#_Toc171097438)

[2.4.2幻读 11](#_Toc171097439)

[2.4.3主动加锁保证可重复读 11](#_Toc171097440)

[2.5 数据库设计与实现 12](#_Toc171097441)

[2.5.1 从概念模型到MySQL实现 12](#_Toc171097442)

[2.5.2 从需求分析到逻辑模型 13](#_Toc171097443)

[2.5.3 建模工具的使用 14](#_Toc171097444)

[2.5.4 制约因素分析与设计 14](#_Toc171097445)

[2.5.5 工程师责任及其分析 14](#_Toc171097446)

[2.6 数据库应用开发(JAVA篇) 15](#_Toc171097447)

[2.6.1 JDBC体系结构和简单的查询 15](#_Toc171097448)

[2.6.2 银行卡销户 16](#_Toc171097449)

[2.6.3 客户修改密码 16](#_Toc171097450)

[2.6.4 事务与转账操作 17](#_Toc171097451)

[2.6.5 把稀疏表格转为键值对存储 18](#_Toc171097452)

[3 课程总结 20](#_Toc171097453)

# 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：

1. **select** c\_name, c\_mail, c\_phone
2. **from** client
3. **where** exists(
4. **select** 1 **from** property **where** c\_id = pro\_c\_id and pro\_type = 2) # 购买了保险
5. and
6. exists(
7. **select** 1 **from** property **where** c\_id = pro\_c\_id and pro\_type = 3) # 购买了基金
8. **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：

1. **select** pro\_income, count( \* ) **as** presence
2. **from** property
3. **group** **by** pro\_income
4. **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：

1. **select** c\_name, c\_phone, c\_mail
2. **from** client
3. **where** c\_id\_card like "4201%"
4. and
5. not exists(
6. **select** 1 **from** property **where** pro\_c\_id = c\_id and pro\_type = 1
7. )
8. **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：

1. **select** c\_name, c\_phone, c\_mail
2. **from** client
3. **where** c\_id in (
4. **select** pro\_c\_id **from** property, fund **where** f\_type = "货币型"
5. and pro\_pif\_id = f\_id and pro\_type = 3
6. )
7. **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：

1. **select** c\_name, c\_id\_card, total\_income
2. **from** client,
3. (**select** sum(pro\_income) **as** total\_income, pro\_c\_id **from** property **where** pro\_status = "可用"
4. **group** **by** pro\_c\_id) c\_income
5. **where** c\_id = pro\_c\_id
6. **order** **by** total\_income **desc**
7. 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：

1. **select** c\_id, c\_name, count(b\_c\_id) **as** number\_of\_cards
2. **from** client left join bank\_card **on** c\_id = b\_c\_id
3. **where** c\_name like "黄%"
4. **group** **by** c\_id
5. **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：

1. s elect c\_name, c\_id\_card, ifnull(sum(proper\_total.total),0) **as** total\_amount
2. **from** client left join(
3. **select** pro\_c\_id, sum(p\_amount\*pro\_quantity) **as** total
4. **from** finances\_product, property
5. **where** p\_id=pro\_pif\_id and pro\_type=1
6. **group** **by** pro\_c\_id
7. **union** all
8. **select** pro\_c\_id, sum(i\_amount\*pro\_quantity) **as** total
9. **from** insurance, property
10. **where** i\_id=pro\_pif\_id and pro\_type=2
11. **group** **by** pro\_c\_id
12. **union** all
13. **select** pro\_c\_id, sum(f\_amount\*pro\_quantity) **as** total
14. **from** fund, property
15. **where** f\_id=pro\_pif\_id and pro\_type=3
16. **group** **by** pro\_c\_id) proper\_total
17. **on** c\_id=pro\_c\_id
18. **group** **by** c\_id
19. **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. --(1) 基金总收益排名(名次不连续)
2. **select** pro\_c\_id, total\_revenue, rank() over(**order** **by** total\_revenue **desc**) **as** "rank"
3. **from**(
4. **select** pro\_c\_id, sum(pro\_income) **as** total\_revenue **from** property **where** pro\_type = 3 **group** **by** pro\_c\_id
5. ) a
6. **order** **by** total\_revenue **desc**, pro\_c\_id;
7. --(2) 基金总收益排名(名次连续)
8. **select** pro\_c\_id, total\_revenue, dense\_rank() over(**order** **by** total\_revenue **desc**) **as** "rank"
9. **from**(
10. **select** pro\_c\_id, sum(pro\_income) **as** total\_revenue **from** property **where** pro\_type = 3 **group** **by** pro\_c\_id
11. ) a
12. **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。

1. **select** **distinct** A.pro\_c\_id **as** c\_id1, B.pro\_c\_id **as** c\_id2
2. **from** property A, property B
3. **where** A.pro\_c\_id < B.pro\_c\_id
4. and not exists(
5. **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 (
6. **select** pro\_pif\_id **from** property D **where** D.pro\_c\_id = B.pro\_c\_id and D.pro\_type = 3
7. )
8. )
9. and not exists(
10. **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 (
11. **select** pro\_pif\_id **from** property D **where** D.pro\_c\_id = A.pro\_c\_id and D.pro\_type = 3
12. )
13. )
14. and exists(
15. **select** 1 **from** property C **where** C.pro\_c\_id = A.pro\_c\_id and C.pro\_type = 3
16. )

## 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：

1. **select** pyear,
2. rank() over(partition **by** pyear **order** **by** sumamount **desc**) rk,
3. p\_id,
4. sumamount
5. **from**(
6. (**select** "2010"
7. **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"
8. **group** **by** p\_id **order** **by** sumamount **desc** limit 0, 3) **union**(**select** "2011"
9. **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"
10. **group** **by** p\_id **order** **by** sumamount **desc** limit 0, 3)
11. ) a
12. **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：

1. **select** **distinct** pro\_c\_id
2. **from** property p1
3. **where** not exists(
4. **select** 1 **from**(
5. **select** pro\_pif\_id **from** property **where** pro\_type = 1 **group** **by** pro\_pif\_id **having** count( \* ) > 2
6. ) hot **where** hot.pro\_pif\_id not in (
7. **select** pro\_pif\_id **from** property p2 **where** p1.pro\_c\_id = p2.pro\_c\_id and p2.pro\_type = 1
8. )
9. )
10. **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：

1. **select** pro\_pif\_id, count( \* ) **as** cc, dense\_rank() over(**order** **by** count( \* ) **desc**) **as** prank
2. **from** property
3. **where** pro\_type = 1 and pro\_pif\_id in (
4. **select** **distinct** pro\_pif\_id **from** property **where** pro\_type = 1 and pro\_pif\_id < > 14 and pro\_c\_id in (
5. **select** pro\_c\_id **from**(
6. **select** pro\_c\_id, dense\_rank() over(**order** **by** pro\_quantity) **as** rk **from** property **where** pro\_type = 1 and pro\_pif\_id = 14
7. ) fin\_rk
8. )
9. )
10. **group** **by** pro\_pif\_id
11. **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：

1. **select** id1 pro\_c\_id, id2 pro\_c\_id, count( \* ) **as** total\_count
2. **from**(
3. **select** p1.pro\_c\_id id1, p2.pro\_c\_id id2, p1.pro\_pif\_id **from** property p1, property 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
4. ) a
5. **group** **by** id1, id2
6. **having** count( \* ) >= 2
7. **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：

1. delimiter $$
2. **CREATE** **TRIGGER** before\_property\_inserted BEFORE **INSERT** **ON** property
3. **FOR** EACH ROW
4. **BEGIN**
5. **declare** msg **varchar**(128);
6. if new.pro\_type = 1 and not exists (**select** \* **from** finances\_product **where** p\_id = new.pro\_pif\_id) **then**
7. **set** msg = concat("finances product #",new.pro\_pif\_id," not found!");
8. elseif new.pro\_type = 2 and not exists (**select** \* **from** insurance **where** i\_id = new.pro\_pif\_id) **then**
9. **set** msg = concat("insurance #",new.pro\_pif\_id," not found!");
10. elseif new.pro\_type = 3 and not exists (**select** \* **from** fund **where** f\_id = new.pro\_pif\_id) **then**
11. **set** msg = concat("fund #",new.pro\_pif\_id," not found!");
12. elseif new.pro\_type not in (1,2,3) **then**
13. **set** msg = concat("type ",new.pro\_type," is illegal!");
14. **end** if;
15. if msg **is** not null **then**
16. SIGNAL SQLSTATE '45000' **SET** MESSAGE\_TEXT = msg;
17. **end** if;
18. **END**$$
19. 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：

1. -- 事务1:
2. use testdb1;
3. ## 请设置适当的事务隔离级别
4. **set** session **transaction** **isolation** **level** **read** **uncommitted**;
6. start **transaction**;
8. -- 时刻2 - 事务1读航班余票,发生在事务2修改之后
9. ## 添加等待代码，确保读脏
10. **set** @n = sleep(1);
11. **select** tickets **from** ticket **where** flight\_no = 'CA8213';
12. **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：

1. -- 事务1（采用默认的事务隔离级别- repeatable read）:
2. use testdb1;
3. **select** @@transaction\_isolation;
4. start **transaction**;
5. ## 第1次查询余票超过300张的航班信息
6. **select** \* **from** ticket **where** tickets > 300;
7. **set** @n = sleep(2);
8. -- 修改航班MU5111的执飞机型为A330-300：
9. **update** ticket **set** aircraft = 'A330-300' **where** flight\_no = 'MU5111';
10. -- 第2次查询余票超过300张的航班信息
11. **select** \* **from** ticket **where** tickets > 300;
12. **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：

1. -- 事务1:
2. use testdb1;
3. **set** session **transaction** **isolation** **level** **read** **uncommitted**;
4. start **transaction**;
5. # 第1次查询航班'MU2455'的余票
6. **select** tickets **from** ticket **where** flight\_no = "MU2455" **for** **update**;
7. **set** @n = sleep(5);
8. # 第2次查询航班'MU2455'的余票
9. **select** tickets **from** ticket **where** flight\_no = "MU2455";
10. **commit**;
11. -- 第3次查询所有航班的余票，发生在事务2提交后
12. **set** @n = sleep(1);
13. **select** \* **from** ticket;
14. -- 事务2:
15. use testdb1;
16. **set** session **transaction** **isolation** **level** **read** **uncommitted**;
17. start **transaction**;
18. **set** @n = sleep(1);
19. # 在事务1的第1，2次查询之间，试图出票1张(航班MU2455)：
20. **update** ticket **set** tickets = tickets-1 **where** flight\_no = "MU2455";
21. **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。

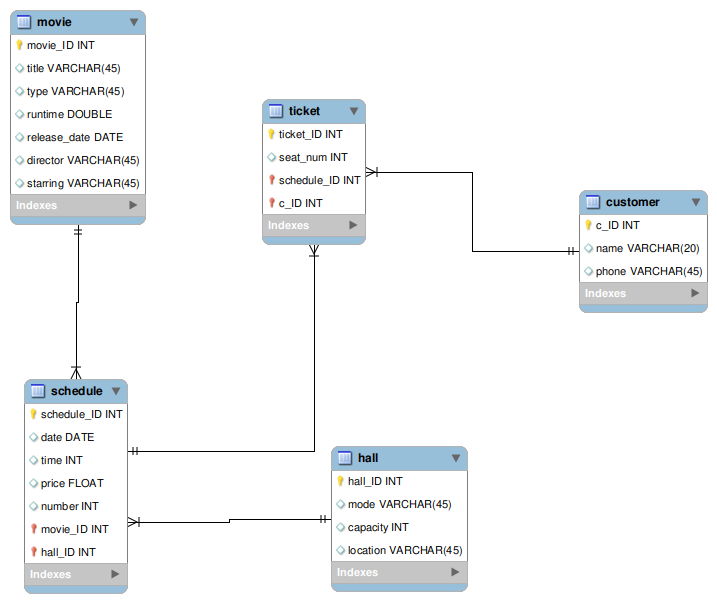
由于代码过长，这里不展示。

### 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=UTC", "root", "123123");
3. statement = connection.createStatement();
4. resultSet = statement.executeQuery("SELECT c\_name, c\_mail, c\_phone FROM 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+"' where c\_mail = '"+mail+"';";
6. Statement statement = connection.createStatement();
7. ResultSet resultSet = statement.executeQuery(sqlSelect);
8. resultSet.next()) {
9. String oldPassword = resultSet.getString("c\_password");
10. **if**(oldPassword.equals(password)) {
11. **int** state = statement.executeUpdate(sqlUpdate);
12. **if**(state == 1)  result = 1;
13. **else** result = -1;
14. }
15. **else**  result = 3;
16. }
17. **else**  result = 2;
18. }   **catch**(SQLException e){
19. result = -1;
20. e.printStackTrace();
21. } **finally** {
22. **return** result;
23. }
24. }

### 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：

1. **public** **static** **boolean** transferBalance(Connection connection, String sourceCard, String destCard, **double** amount){
2. **try** {
3. String sql = "select \* from bank\_card where b\_number = ?";
4. PreparedStatement preparedStatement = connection.prepareStatement(sql);
5. preparedStatement.setString(1, sourceCard);
6. ResultSet resultSet = preparedStatement.executeQuery();
7. **if** (!resultSet.next() || resultSet.getString("b\_type").equals("信用卡") || resultSet.getDouble("b\_balance") < amount) **return** **false**;
8. preparedStatement = connection.prepareStatement(sql);
9. preparedStatement.setString(1, destCard);
10. resultSet = preparedStatement.executeQuery();
11. **if** (!resultSet.next()) **return** **false**;
12. sql = "update bank\_card set b\_balance = b\_balance+? where b\_number=?";
13. preparedStatement = connection.prepareStatement(sql);
14. preparedStatement.setDouble(1, -amount);
15. 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, PASS);
16. ResultSet resultSet = **null**;
17. String[] Subject = {"chinese", "math", "english", "physics", "chemistry",
18. "biology", "history", "geography", "politics"};
19. **try** {
20. resultSet = connection.createStatement().executeQuery(
21. "select \* from entrance\_exam");
22. **while** (resultSet.next()) {
23. **int** sno = resultSet.getInt("sno"), score;
24. **for** (String subject : Subject) {
25. score = resultSet.getInt(subject);
26. **if** (!resultSet.wasNull())
27. insertSC(connection, sno, subject, score);
28. }
29. }
30. } **catch** (SQLException e) {
31. e.printStackTrace();
32. }
33. }

# 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。