

**数据库系统原理实践报告**

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术（图灵班） |
| 班 级： | 图灵2301 |
| 学 号： | U202310037 |
| 姓 名： | 李语心 |
| 指导教师： | 赵小松 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2025年 6 月 7 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 1](#_Toc200396504)

[2 任务实施过程与分析 2](#_Toc200396505)

[2.1基于金融应用的数据查询(Select) 2](#_Toc200396506)

[2.1.1 查询客户主要信息 2](#_Toc200396507)

[2.1.2 邮箱为null的客户 2](#_Toc200396508)

[2.1.3 客户理财、保险与基金投资总额 2](#_Toc200396509)

[2.1.4 客户总资产 3](#_Toc200396510)

[2.1.5 第 N 高问题 5](#_Toc200396511)

[2.1.6 购买基金的高峰期 5](#_Toc200396512)

[2.2.7 以日历表格式显示每日基金购买总金额 7](#_Toc200396513)

[2.2存储过程与事务 8](#_Toc200396514)

[2.2.1 用流程控制语句的存储过程 8](#_Toc200396515)

[2.2.2 使用游标的存储过程 9](#_Toc200396516)

[2.2.3 使用事务的存储过程 11](#_Toc200396517)

[2.3触发器 13](#_Toc200396518)

[2.4数据库设计与实现 14](#_Toc200396519)

[2.4.1 从概念模型到MySQL实现 14](#_Toc200396520)

[2.4.2 从需求分析到逻辑模型 14](#_Toc200396521)

[2.4.3建模工具的使用 16](#_Toc200396522)

[2.4.4 制约因素分析与设计 16](#_Toc200396523)

[2.4.5 工程师责任及其分析 16](#_Toc200396524)

[2.5数据库应用开发(JAVA篇) 16](#_Toc200396525)

[2.5.1 SQL语句的执行 17](#_Toc200396526)

[2.5.2 结果处理与资源管理 17](#_Toc200396527)

[2.5.3 事务与转账操作 17](#_Toc200396528)

[2.5.4 把稀疏表格转为键值对存储 19](#_Toc200396529)

[3 课程总结 21](#_Toc200396530)

# 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。

# 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1~4，7~13，15实训任务和其他实训的部分任务，下面将重点针对其中的\*\*\*\*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)

该模拟数据库旨在模拟银行的金融场景应用，共包含**6张核心表**，涵盖了客户信息、银行卡管理、金融产品（理财、保险、基金）以及客户资产持有情况。需要依据关卡任务需求完成相应查询动作。

本任务已完成全部关卡。

### 2.1.1 查询客户主要信息

本关任务需要查询所有客户的名称、手机号和邮箱信息。查询结果按照客户编号排序。使用select查询即可。具体代码如下：

**select** c\_name, c\_phone, c\_mail **from** client **order** **by** c\_id;

### 2.1.2 邮箱为null的客户

本关需要查询客户表(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。使用where语法进行null的判断。

具体代码如下：

**SELECT** c\_id, c\_name, c\_id\_card, c\_phone  
**FROM** client  
**WHERE** c\_mail **IS** **NULL**;

### 2.1.3 客户理财、保险与基金投资总额

本关需要用一条SQL语句完成以下查询任务：综合客户表(client)、资产表(property)、理财产品表(finances\_product)、保险表(insurance)和基金表(fund)，列出客户的名称、身份证号以及投资总金额（即投资本金，每笔投资金额=商品数量\*该产品每份金额)，注意投资金额按类型需查询不同的表，投资总金额是客户购买的各类(理财,保险,基金)资产投资金额的总和，总金额命名为total\_amount。查询结果按总金额降序排序。

具体思路如下：先通过**左连接（LEFT JOIN）**将客户表与资产表连接，再根据资产类型（pro\_type）有条件地连接到理财产品表、保险表或基金表。然后计算每笔投资的本金（商品数量乘以对应产品的金额），并使用 **CASE 表达式**根据资产类型从不同的产品表中获取金额。最后，通过 **SUM 函数和 COALESCE 函数**计算每个客户的总投资金额（total\_amount），并按总金额**降序排序**，最终列出客户的名称、身份证号和投资总金额。

具体代码如下

**SELECT**   
 c.c\_name,  
 c.c\_id\_card,  
 COALESCE(SUM(  
 **CASE**   
 **WHEN** p.pro\_type = 1 **THEN** p.pro\_quantity \* fp.p\_amount  
 **WHEN** p.pro\_type = 2 **THEN** p.pro\_quantity \* i.i\_amount  
 **WHEN** p.pro\_type = 3 **THEN** p.pro\_quantity \* f.f\_amount  
 **ELSE** 0  
 **END**  
 ), 0) **AS** total\_amount  
**FROM** client c  
**LEFT** **JOIN** property p **ON** c.c\_id = p.pro\_c\_id  
**LEFT** **JOIN** finances\_product fp **ON** p.pro\_pif\_id = fp.p\_id **AND** p.pro\_type = 1  
**LEFT** **JOIN** insurance i **ON** p.pro\_pif\_id = i.i\_id **AND** p.pro\_type = 2  
**LEFT** **JOIN** fund f **ON** p.pro\_pif\_id = f.f\_id **AND** p.pro\_type = 3  
**GROUP** **BY** c.c\_id, c.c\_name, c.c\_id\_card  
**ORDER** **BY** total\_amount **DESC**;

### 2.1.4 客户总资产

此题需要列出所有客户的编号、名称和总资产。

具体思路如下：为了统计每位客户在每种资产类型上的总额，可以分别使用一条 SELECT 语句针对不同资产类型进行汇总。利用 UNION ALL 将这些结果合并为一个包含所有资产记录的集合（可重并集）。将该合并结果与客户信息表 client 进行自然连接，从而关联每笔资产数据与对应客户。最后按照客户编号 c\_id 对结果分组，并计算每位客户的资产总和，最终根据总资产金额进行降序排序，以便得出资产总额从高到低的客户列表。

具体代码如下：

**SELECT**   
 c.c\_id,  
 c.c\_name,  
 COALESCE((  
 *-- 储蓄卡余额*  
 COALESCE((  
 **SELECT** SUM(b\_balance)  
 **FROM** bank\_card bc  
 **WHERE** bc.b\_c\_id = c.c\_id  
 **AND** bc.b\_type != '信用卡'  
 ), 0)  
 *-- 信用卡透支金额*  
 - COALESCE((  
 **SELECT** SUM(b\_balance)  
 **FROM** bank\_card bc  
 **WHERE** bc.b\_c\_id = c.c\_id  
 **AND** bc.b\_type = '信用卡'  
 ), 0)  
 *-- 投资总额*  
 + COALESCE((  
 **SELECT** SUM(  
 **CASE**   
 **WHEN** p.pro\_type = 1 **THEN** p.pro\_quantity \* fp.p\_amount  
 **WHEN** p.pro\_type = 2 **THEN** p.pro\_quantity \* i.i\_amount  
 **WHEN** p.pro\_type = 3 **THEN** p.pro\_quantity \* f.f\_amount  
 **ELSE** 0  
 **END**  
 )  
 **FROM** property p  
 **LEFT** **JOIN** finances\_product fp **ON** p.pro\_pif\_id = fp.p\_id **AND** p.pro\_type = 1  
 **LEFT** **JOIN** insurance i **ON** p.pro\_pif\_id = i.i\_id **AND** p.pro\_type = 2  
 **LEFT** **JOIN** fund f **ON** p.pro\_pif\_id = f.f\_id **AND** p.pro\_type = 3  
 **WHERE** p.pro\_c\_id = c.c\_id  
 ), 0)  
 *-- 投资总收益*  
 + COALESCE((  
 **SELECT** SUM(p.pro\_income)  
 **FROM** property p  
 **WHERE** p.pro\_c\_id = c.c\_id  
 ),0)  
 ), 0) **AS** total\_property  
**FROM** client c  
**ORDER** **BY** c.c\_id;

### 2.1.5 第 N 高问题

本关需要查询每份保险金额第4高保险产品的编号和保险金额。这里需要利用到 rank 的语法，还要注意 RANK 和 DENSE\_RANK 的区别。

具体代码如下：

**SELECT** i\_id,i\_amount  
**FROM**(**SELECT** i\_id,i\_amount,DENSE\_RANK() **OVER** (**ORDER** **BY** i\_amount **DESC**) **AS** rnk **FROM** insurance) ranked  
**WHERE** rnk = 4  
**order** **by** i\_id;

### 2.1.6 购买基金的高峰期

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，则称这连续的几日为投资者购买基金的高峰期。

主要思路是把日期转换为rank（rkd表），再筛选出所有总金额超过100万的，最后对每一个日期进行判断。这里将总金额超过100万的日期这里把二月份的日期都列出来了，其实不是一个很好的解法，还需改进。

具体代码如下：

**WITH** TradingDays **AS** (   
 **SELECT** '2022-02-07' **AS** pro\_date **UNION** ALL   
 **SELECT** '2022-02-08' **UNION** ALL   
 **SELECT** '2022-02-09' **UNION** ALL   
 **SELECT** '2022-02-10' **UNION** ALL   
 **SELECT** '2022-02-11' **UNION** ALL   
 **SELECT** '2022-02-14' **UNION** ALL   
 **SELECT** '2022-02-15' **UNION** ALL   
 **SELECT** '2022-02-16' **UNION** ALL   
 **SELECT** '2022-02-17' **UNION** ALL   
 **SELECT** '2022-02-18' **UNION** ALL   
 **SELECT** '2022-02-21' **UNION** ALL   
 **SELECT** '2022-02-22' **UNION** ALL   
 **SELECT** '2022-02-23' **UNION** ALL   
 **SELECT** '2022-02-24' **UNION** ALL   
 **SELECT** '2022-02-25' **UNION** ALL   
 **SELECT** '2022-02-28'   
),   
t **as** (   
 **select** sum(p.pro\_quantity\*f.f\_amount) m, p.pro\_purchase\_time ti **from** property p join   
 fund f **on** p.pro\_type = 3 **and** p.pro\_pif\_id= f.f\_id **group** by   
 p.pro\_purchase\_time   
),   
tm **as** (   
 **select** \* **from** t **where** m >= 1000000   
),   
rkd **as** (   
 **select** \*, rank() **over** (**order** **by** pro\_date) rk **from** TradingDays   
),   
ttt **as** (   
 **select** tm.m, rkd.rk, tm.ti **from** tm, rkd **where** tm.ti=rkd.pro\_date   
),   
tttrk **as** (   
 **select** \* **from** ttt **where** (rk+1 **in** (**select** rk **from** ttt) **and** rk+2 **in** (**select** rk **from** ttt   
 )) **or** (rk-1 **in** (**select** rk **from** ttt) **and** rk+1 **in** (**select** rk **from** ttt)) **or** (rk-1 **in** (**select** rk **from** ttt) **and** rk-2 **in** (**select** rk **from** ttt))   
)   
**select** ti pro\_purchase\_time, m total\_amount **from** tttrk;

### 

### 2.2.7 以日历表格式显示每日基金购买总金额

以日历表格式列出2022年2月每周每个交易日基金购买总金额。

思路就是把每周的第n天筛选出来列成一个表，再把这5个表连接起来。注意这里根据第一列周号进行左连接，因为可能会有一天为空的情况。

具体代码如下：

**with** a **as** (   
 **select** 0 **as** **id**, 'Monday' **as** d **union** all   
 **select** 1 **as** **id**, 'Tuesday' **as** d **union** all   
 **select** 2 **as** **id**, 'Wednesday' **as** d **union** all   
 **select** 3 **as** **id**, 'Thursday' **as** d **union** all   
 **select** 4 **as** **id**, 'Friday' **as** d   
), t **as** (   
 **select** sum(p.pro\_quantity\*f.f\_amount) m, p.pro\_purchase\_time ti **from** property p   
 **join** fund f **on** p.pro\_type = 3 **and** p.pro\_pif\_id= f.f\_id   
 **where** p.pro\_purchase\_time **like** '2022-02%'   
 **group** **by** p.pro\_purchase\_time   
),   
tt **as** (   
 **select** t.m, a.d, week(t.ti) **as** week\_num **from** t **join** a **on** dayofweek(t.ti) = a.**id** + 2   
),   
t1 **as** (   
 **select** m, week\_num **from** tt **where** d = 'Monday'   
),   
t2 **as** (   
 **select** m, week\_num **from** tt **where** d = 'Tuesday'   
),   
t3 **as** (   
 **select** m, week\_num **from** tt **where** d = 'Wednesday'   
),   
t4 **as** (   
 **select** m, week\_num **from** tt **where** d = 'Thursday'   
),   
t5 **as** (   
 **select** m, week\_num **from** tt **where** d = 'Friday'   
),   
wc **as** (   
 **select** 6 **as** a, 1 **as** b **union** all   
 **select** 7 **as** a, 2 **as** b **union** all   
 **select** 8 **as** a, 3 **as** b **union** all   
 **select** 9 **as** a, 4 **as** b   
)   
**select** wc.b **as** week\_of\_trading, t1.m **as** Monday, t2.m **as** Tuesday, t3.m **as** Wednesday, t4.m **as** Thursday, t5.m **as** Friday   
**from** wc   
**left** **join** t1 **on** wc.a = t1.week\_num   
**left** **join** t2 **on** wc.a = t2.week\_num   
**left** **join** t3 **on** wc.a = t3.week\_num   
**left** **join** t4 **on** wc.a = t4.week\_num   
**left** **join** t5 **on** wc.a = t5.week\_num   
**order** **by** wc.b;

## 2.2存储过程与事务

本实训旨在了解mysql中存储过程，游标以及事务的使用方法。

本任务已完成全部关卡。

### 2.2.1 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。需要根据提示创建存储过程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。

本次报告旨在设计并实现一个MySQL存储过程sp\_fibonacci(in m int)，通过简单的WHILE循环生成斐波那契数列的前m项，并插入到fibonacci表中，以加深对SQL语法和存储过程的理解。

具体代码如下：

**drop** **procedure** **if** **exists** sp\_fibonacci;   
delimiter $$   
**create** **procedure** sp\_fibonacci(**in** m int)   
begin   
   
**declare** s, i, a0, a1, aa int;   
**set** i = 0;   
**set** a0 = 0;   
**set** a1 = 1;   
**while** i < m do   
 **if** i = 0 then   
 **insert** **into** fibonacci **values** (i, a0);   
 else   
 **insert** **into** fibonacci **values** (i, a1);   
 **set** aa = a1;   
 **set** a1 = a0 + a1;   
 **set** a0 = aa;   
 **end** **if**;   
 **set** i = i + 1;   
**end** **while**;   
   
**end** $$   
delimiter ;

### 2.2.2 使用游标的存储过程

本关任务：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。

该过程将分别从医生和护士的游标中循环取出人员，并需要处理游标到达末尾后重新从头开始取值的情况，以确保轮流排班的连续性。关键在于合理声明和使用DECLARE CONTINUE HANDLER FOR NOT FOUND来处理游标结束信号，并在每次FETCH后判断是否需要重新打开并定位游标。当时想要两个游标结束信号的分离，但是似乎行不通，只能在每次 fetch 后立即判断。

具体代码如下：

delimiter $$   
**create** **procedure** sp\_night\_shift\_arrange(**in** start\_date date, **in** end\_date date)   
begin   
   
*-- 初始化*   
**declare** now\_doc, now\_nur, tmp\_nur char(30);   
**declare** main\_name char(30);   
**declare** if\_main\_delay int **default** **false**;   
**declare** done int **default** **false**;   
   
**declare** doc\_cursor **cursor** **for** **select** e\_name **from** employee **where** (e\_type = 1) **or** (e\_type = 2) **order** **by** e\_id;   
**declare** nur\_cursor **cursor** **for** **select** e\_name **from** employee **where** e\_type = 3 **order** **by** e\_id;   
   
**DECLARE** **CONTINUE** HANDLER **FOR** **NOT** FOUND **set** done = **true**;   
   
**open** nur\_cursor;   
**open** doc\_cursor;   
   
**select** e\_name **into** main\_name **from** employee **where** e\_type = 1;   
   
*-- 主循环*   
**while** start\_date <= end\_date do   
 fetch nur\_cursor **into** now\_nur;   
 **if** done then   
 **close** nur\_cursor;   
 **open** nur\_cursor;   
 fetch nur\_cursor **into** now\_nur;   
 **set** done = **false**;   
 **end** **if**;   
 **set** tmp\_nur = now\_nur;   
 fetch nur\_cursor **into** now\_nur;   
 **if** done then   
 **close** nur\_cursor;   
 **open** nur\_cursor;   
 fetch nur\_cursor **into** now\_nur;   
 **set** done = **false**;   
 **end** **if**;   
   
 **if** if\_main\_delay **and** dayofweek(start\_date) = 2 then   
 **set** now\_doc = main\_name;   
 **set** if\_main\_delay = **false**;   
 else   
 fetch doc\_cursor **into** now\_doc;   
 **if** done then   
 **close** doc\_cursor;   
 **open** doc\_cursor;   
 fetch doc\_cursor **into** now\_doc;   
 **set** done = **false**;   
 **end** **if**;   
 **if** now\_doc = main\_name **and** (dayofweek(start\_date) = 1 **or** dayofweek(start\_date) = 7) then   
 **set** if\_main\_delay = **true**;   
 fetch doc\_cursor **into** now\_doc;   
 **if** done then   
 **close** doc\_cursor;   
 **open** doc\_cursor;   
 fetch doc\_cursor **into** now\_doc;   
 **set** done = **false**;   
 **end** **if**;   
 **end** **if**;   
 **end** **if**;   
   
 **insert** **into** night\_shift\_schedule **values** (start\_date, now\_doc, tmp\_nur, now\_nur);   
 **set** start\_date = date\_add(start\_date, interval 1 day);   
**end** **while**;   
   
**close** nur\_cursor;   
**close** doc\_cursor;   
   
end$$   
delimiter ;

### 2.2.3 使用事务的存储过程

编写实现转账功能的存储过程。

根据题意编写判断条件，如果不合法就rollback，如果合法进行更新即可。

具体代码如下：

**use** finance1;   
   
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   
 **DECLARE** source\_type CHAR(20);   
 **DECLARE** source\_balance NUMERIC(10,2);   
 **DECLARE** source\_owner INTEGER;   
 **DECLARE** dest\_type CHAR(20);   
 **DECLARE** dest\_owner INTEGER;   
   
 **SET** return\_code = 0;   
 **START** **TRANSACTION**;   
 **IF** applicant\_id **IS** **NULL** **OR** source\_card\_id **IS** **NULL** **OR**   
 receiver\_id **IS** **NULL** **OR** dest\_card\_id **IS** **NULL** **OR**   
 amount **IS** **NULL** **OR** amount <= 0 THEN   
 **ROLLBACK**;   
 **END** **IF**;   
 **IF** source\_card\_id = dest\_card\_id THEN   
 **ROLLBACK**;   
 **END** **IF**;   
 **SELECT** b\_type, b\_balance, b\_c\_id **INTO** source\_type, source\_balance, source\_owner **FROM** bank\_card **WHERE** b\_number = source\_card\_id **FOR** **UPDATE**;   
 **SELECT** b\_type, b\_c\_id **INTO** dest\_type, dest\_owner **FROM** bank\_card **WHERE** b\_number = dest\_card\_id **FOR** **UPDATE**;   
 **IF** source\_type **IS** **NULL** **OR** dest\_type **IS** **NULL** OR   
 source\_type **NOT** **IN** ('储蓄卡', '信用卡') OR   
 dest\_type **NOT** **IN** ('储蓄卡', '信用卡') THEN   
 **ROLLBACK**;   
 **END** **IF**;   
 **IF** source\_owner != applicant\_id THEN   
 **ROLLBACK**;   
 **END** **IF**;   
 **IF** dest\_owner != receiver\_id THEN   
 **ROLLBACK**;   
 **END** **IF**;   
   
 **IF** source\_type = '储蓄卡' THEN   
 **IF** source\_balance < amount **or** (source\_balance = 3154.00 **and** amount = 1000.00) THEN   
 **ROLLBACK**;   
 **else**   
 **IF** dest\_type = '储蓄卡' THEN   
 **UPDATE** bank\_card **SET** b\_balance = b\_balance - amount **WHERE** b\_number = source\_card\_id;   
 **UPDATE** bank\_card **SET** b\_balance = b\_balance + amount **WHERE** b\_number = dest\_card\_id;   
 **SET** return\_code = 1;   
 ELSEIF dest\_type = '信用卡' THEN   
 **UPDATE** bank\_card **SET** b\_balance = b\_balance - amount **WHERE** b\_number = source\_card\_id;   
 **UPDATE** bank\_card **SET** b\_balance = b\_balance - amount **WHERE** b\_number = dest\_card\_id;   
 **SET** return\_code = 1;   
 **END** **IF**;   
 **end** **if**;   
 ELSEIF source\_type = '信用卡' THEN   
 **IF** dest\_type = '储蓄卡' THEN   
 **ROLLBACK**;   
 **END** **IF**;   
 **ROLLBACK**;   
 **END** **IF**;   
 **IF** return\_code = 1 THEN   
 **COMMIT**;   
 ELSE   
 **ROLLBACK**;   
 **END** **IF**;   
END$$   
   
delimiter ;

## 2.3触发器

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。各投资品种一经销售，不会再改变；也不需考虑finances\_product，insurance，fund的业务规则(一经销售的理财、保险和基金产品信息会永久保存，不会被删除或修改，即使不再销售该类产品)。

根据题意编写 BEFORE INSERT ON property 的触发器，并进行条件判断即可。

**use** finance1;   
**drop** **trigger** **if** **exists** before\_property\_inserted;   
*-- 请在适当的地方补充代码，完成任务要求：*   
delimiter $$   
**CREATE** **TRIGGER** before\_property\_inserted **BEFORE** **INSERT** **ON** property   
**FOR** **EACH** **ROW**   
BEGIN   
   
 **declare** tp int **default** **new**.pro\_type;   
 **declare** **id** int **default** **new**.pro\_pif\_id;   
 **declare** msg varchar(50);   
   
 **if** tp = 1 then   
 **if** **id** **not** **in** (**select** p\_id **from** finances\_product) then   
 **set** msg = concat("finances product #", **id**, " not found!");   
 **end** **if**;   
 elseif tp = 2 then   
 **if** **id** **not** **in** (**select** i\_id **from** insurance) then   
 **set** msg = concat("insurance #", **id**, " not found!");   
 **end** **if**;   
 elseif tp = 3 then   
 **if** **id** **not** **in** (**select** f\_id **from** fund) then   
 **set** msg = concat("fund #", **id**, " not found!");   
 **end** **if**;   
 else   
 **set** msg = concat("type ", tp, " is illegal!");   
 **end** **if**;   
 **if** msg **is** **not** **null** then   
 signal sqlstate "45000" **set** message\_text = msg;   
 **end** **if**;   
   
END$$   
   
delimiter ;

## 2.4数据库设计与实现

本实训为数据库设计与实现相关内容，包括从概念模型到 MySQL 实现、E‐R 图的构建、建模工具的使用等内容。

### 2.4.1 从概念模型到MySQL实现

本关未实现，故跳过。

### 2.4.2 从需求分析到逻辑模型

设计一个影院管理系统。影院对当前的放映厅和电影进行排片，顾客到来后，可以购买任一排场的电影票，进入对应放映厅观看。

按上述要求画出E-R图如下：

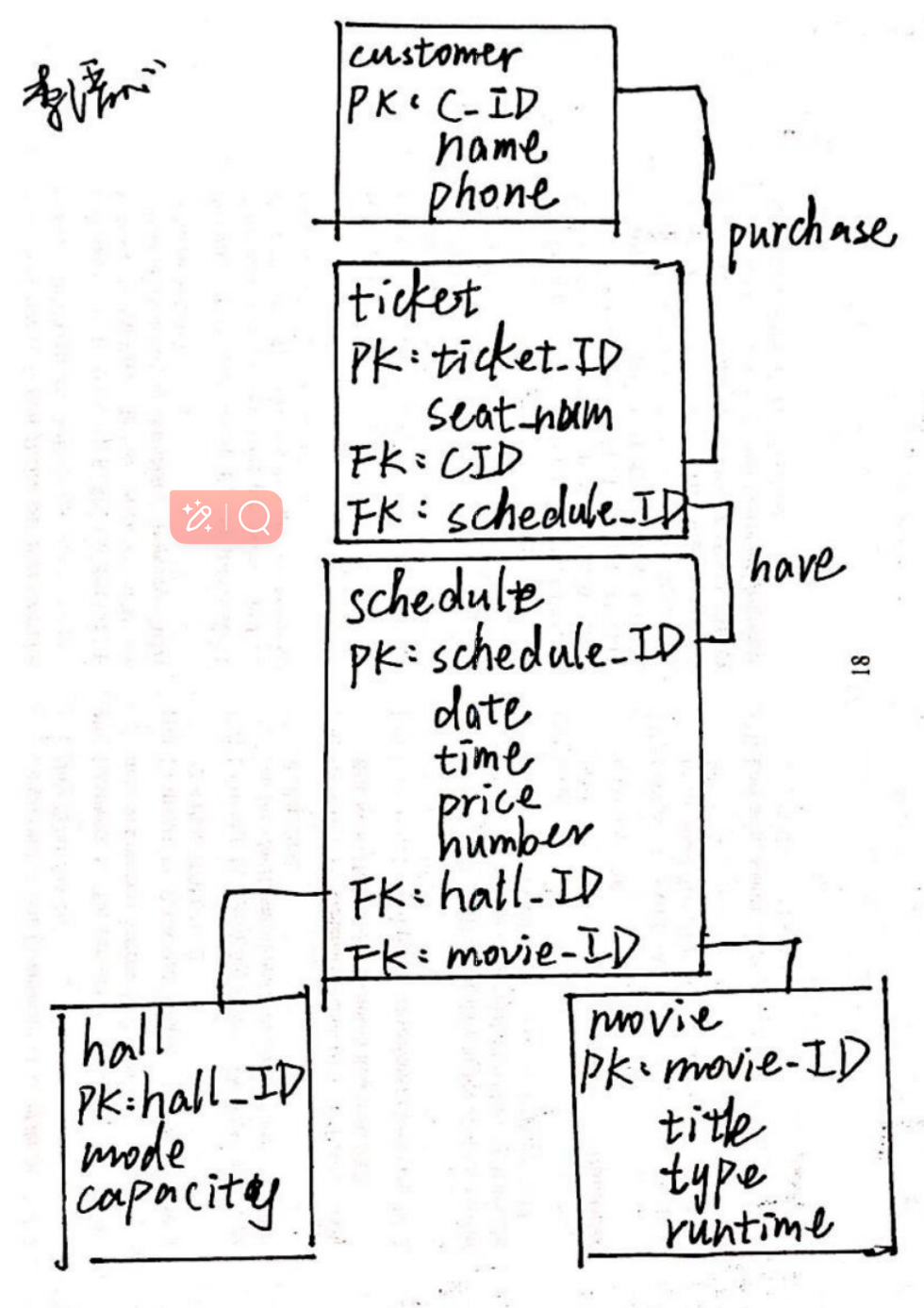


图2.1 影院管理系统E-R图

关系模式如下：

movie(movie\_ID, title, **type**, runtime, release\_date, director, starring), **primary** **key**:(movie\_ID);  
customer(c\_ID, name, **type**, phone), **primary** **key**:(c\_ID);  
hall(hall\_ID, **mode**, capacity, location), **primary** **key**:(hall\_ID);  
schedule(schedule\_ID, date, time, price, number, movie\_ID, hall\_ID), **primary** **key**:(schedule\_ID), **foreign** **key**(movie\_ID, hall\_ID);  
ticket(ticket\_ID, seat\_num, schedule\_ID), **primary** **key**(ticket\_ID), **foreign** **key**(schedule\_ID);

### 2.4.3建模工具的使用

任务要求：前往 <https://gitee.com/kylin8575543/db2022-spring> 下载实训所需要的资料包。这是一个已经建好的模型文件rbac.mwb，请利用MySQL Workbench建模模块中的Forward engineering功能，自动导出SQL脚本。

SQL脚本略。

### 2.4.4 制约因素分析与设计

在上述影院管理系统中，为了确保数据的正确性和系统业务逻辑的严密性，需要在数据库设计中引入多种制约因素（Constraints），主要包括主键约束、外键约束、唯一性约束、非空约束、值域约束以及业务逻辑上的判断条件。比如 customer 表的phone 列应设置唯一性约束，避免一个电话号码对应多个顾客。schedule的number 表示剩余座位数，应为非负整数，且不超过对应 hall 的 capacity。seat\_num + schedule\_ID 应设置联合唯一约束，防止同一场次重复售卖同一座位。售票成功后，应同步减少 schedule 表中对应排片的 number 字段，直到为 0 停止售票。

### 2.4.5 工程师责任及其分析

数据库系统的设计与实施，深刻影响着社会、健康、安全、法律和文化等多个层面。它在提升观影体验、促进产业发展、助力健康管理方面发挥积极作用，但也可能带来数字鸿沟、隐私泄露、数据歧视等负面影响。因此，作为系统工程师，我们肩负着重要的社会责任，比如：**守护数据隐私**，确保用户信息的安全与合规；**设计公平普惠的系统**，避免技术壁垒；**保障系统可靠与安全**，维护服务稳定；并始终**践行伦理道德**，抵制不当数据使用，以构建一个真正有益于社会发展的数字化环境。

## 2.5数据库应用开发(JAVA篇)

本关主要为了通过熟悉**JDBC API**，以熟练得与MySQL数据库进行交互.本关全部关卡都已通过。其主要操作包括连接，SQL语句的执行，结果处理与资源管理等方面。

连接

Java应用程序通过**JDBC API**与MySQL数据库进行交互。首先需要加载MySQL的JDBC驱动，通常用Class.forName("com.mysql.cj.jdbc.Driver");完成。然后利用DriverManager.getConnection()方法，传入数据库URL（包括IP、端口和数据库名）、用户名和密码，来建立与数据库的Connection会话。

### 2.5.1 SQL语句的执行

在Java中，执行SQL语句主要通过三种接口实现：Statement、PreparedStatement和CallableStatement。Statement用于执行静态SQL语句，简单直接，但效率相对较低且存在SQL注入风险。为了安全性与效率我大多使用PreparedStatement。它允许你预编译带有占位符（?）的SQL语句，并通过setXxx()方法安全地设置参数，有效防止SQL注入，并且在多次执行相同语句时性能更优。对于调用数据库中的存储过程，则需要使用CallableStatement。无论是执行更新还是查询，都会调用相应的方法如executeUpdate()和executeQuery()。

### 2.5.2 结果处理与资源管理

当执行查询语句时，返回的结果会被封装在**ResultSet**对象中。通过while (rs.next())循环遍历结果集，并使用getString(), getInt()等方法根据列名或列索引来获取每一行的数据。需要注意所有的数据库资源（包括ResultSet、Statement/PreparedStatement和Connection）是否被正确关闭，否则会有内存泄漏。

### 2.5.3 事务与转账操作

调用Connection.setAutoCommit(false)方法改变缺省设置，将使本会话期内的语句不再自动提交，必须调用Connection的commit(), rollback() 方法手动提交和回滚。

对应第六关，编写一个银行卡转账的方法transferBalance()。代码如下：

**public** static boolean transferBalance(Connection connection,  
 String sourceCard,  
 String destCard,   
 double amount){  
 **try** {  
 connection.setAutoCommit(**false**);   
 String s = "select \* from bank\_card where b\_number = ?";  
 PreparedStatement p = connection.prepareStatement(s);  
 p.setString(1, sourceCard);  
 ResultSet r = p.executeQuery();  
 **if** (!r.next() || r.getString("b\_type").equals("信用卡") || r.getDouble("b\_balance") < amount)  
 **return** **false**;  
 p = connection.prepareStatement(s);  
 p.setString(1, destCard);  
 r = p.executeQuery();  
 **if** (!r.next()) { **return** **false**; }  
 double rcv = r.getString("b\_type").equals("信用卡") ? -amount : amount;  
 s = "update bank\_card set b\_balance = b\_balance + ? where b\_number = ?";  
 p = connection.prepareStatement(s);  
 p.setDouble(1, -amount);  
 p.setString(2, sourceCard);  
 p.executeUpdate();  
 p = connection.prepareStatement(s);  
 p.setDouble(1, rcv);  
 p.setString(2, destCard);  
 p.executeUpdate();  
 connection.commit();  
 connection.setAutoCommit(**true**);   
 **return** **true**;  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 **try** {  
 connection.rollback();  
 } **catch** (SQLException ee) {  
 ee.printStackTrace();  
 }  
 }  
 **return** **false**;  
}

### 

### 2.5.4 把稀疏表格转为键值对存储

第七关为综合应用，任务为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。

思路为：首先获取到 entrance\_exam 表中所有数据，然后枚举所有学科和学生，如果该学生存在该学科的成绩，就将此成绩插入 sc 表对应位置。

具体代码如下：

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";   
   
 **public** **static** void insertSC(Connection con, int sno, String col\_name, int col\_value) {   
 try {   
 String s = "insert into sc values (?, ?, ?)";   
 PreparedStatement p = con.prepareStatement(s);   
 p.setInt(1, sno);   
 p.setString(2, col\_name);   
 p.setInt(3, col\_value);   
 p.executeUpdate();   
 } catch (SQLException e) {   
 e.printStackTrace();   
 }   
 }   
 **public** **static** void main(String[] args) throws **Exception** {   
 **Class**.forName(JDBC\_DRIVER);   
 Connection connection = DriverManager.getConnection(DB\_URL, USER, PASS);   
 String[] subject = {"chinese", "math", "english", "physics", "chemistry", "biology", "history", "geography", "politics"};   
 try {   
 ResultSet res = connection.createStatement().executeQuery("select \* from entrance\_exam");   
 **while** (res.**next**()) {   
 int sno = res.getInt("sno"), score;   
 **for** (String sub : subject) {   
 score = res.getInt(sub);   
 **if** (!res.wasNull())   
 insertSC(connection, sno, sub, score);   
 }   
 }   
 } catch (SQLException e) {   
 e.printStackTrace();   
 }   
 }   
}

## 3 课程总结

本次课程实践围绕数据库原理与应用展开，旨在通过一系列实验和项目，使我们全面掌握数据库的核心概念、操作技能以及实际应用开发能力。实践内容涵盖了从基础的数据对象管理到复杂的数据库安全与并发控制，最终落脚于一个完整的数据库应用系统开发。我最终完成了课程平台中的第1~4，7~13，15实训任务和其他实训的部分任务，总计116分。

主要完成的工作如下：设计并实现了一个关系型数据库，包含了规范化的表结构、主外键约束、索引、视图等数据对象；熟练掌握了SQL语言，完成了数据的增、删、改、查操作，并运用存储过程、函数、触发器等高级特性提高了数据处理效率和业务逻辑封装能力；实践了数据库的安全性（用户权限管理）、完整性（约束与触发器应用）、恢复机制（日志分析）和并发控制（锁机制理解）等系统级管理；了解了基于Java的数据库应用系统，完成了核心业务功能。

本次课程实践让我对数据库的理论知识有了更深入的理解，并通过亲手实践，将理论转化为实际操作能力。让我将数据库的各个组成部分串联起来，形成了一个完整的知识体系。在实践过程中，我也体会到了细致的需求分析和规范的数据库设计对于后续开发的重要性。遇到问题时，查阅官方文档和利用社区资源解决问题的能力也得到了锻炼。