

**数据库系统原理实践报告**

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | 本硕博2301班 |
| 学 号： | U202315763 |
| 姓 名： | 王家乐 |
| 指导教师： | 杨茂林 |

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

2025年 6 月 8 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](#_Toc21261)

[2 任务实施过程与分析 2](#_Toc11046)

[2.1 数据库、表与完整性约束的定义(Create) 2](#_Toc10483)

[2.2 表结构与完整性约束的修改（Alter） 3](#_Toc12831)

[2.3 基于金融应用的数据查询(Select) 4](#_Toc17556)

[2.4 数据查询(Select)--新增 9](#_Toc8758)

[2.5 数据的插入、修改与删除(Insert,Update,Delete) 13](#_Toc11650)

[2.6 视图 14](#_Toc13686)

[2.7 存储过程与事务 14](#_Toc7364)

[2.8 触发器 16](#_Toc13133)

[2.9 用户自定义函数 17](#_Toc27786)

[2.10 安全性控制 18](#_Toc25531)

[2.11 并发控制与事务的隔离级别 19](#_Toc2333)

[2.12 备份+日志：介质故障与数据库恢复 22](#_Toc23818)

[2.13 数据库设计与实现 23](#_Toc9788)

[2.14 数据库应用开发（JAVA篇） 24](#_Toc2092)

[3 课程总结 29](#_Toc17525)

# 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**所有**实训任务，下面将重点针对其中的部分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)****

本节主要任务为创建数据库，并且在创建的数据库内创建表，以及对表的规范性和完整性进行一定的约束，如表中主码、外码、 CHECK、 DEFAUT 和 UNIQUE 等约束的建立。

### 2.1.1 创建数据库

本关任务：创建用于2022年北京冬奥会信息系统的数据库:beijing2022。

实现方法：使用如下代码即可完成任务：

**create** **database** beijing2022;

### 2.1.2 创建表及表的主码约束

本关任务：创建指定数据库TestDb，并在TestDb中创建表t\_emp。

实现方法：首先创建数据库，再创建表并指定一些约束。

**create** **database** TestDb;

use TestDb;

**create** **table** t\_emp(

     id **int** **primary** **key**,

**name** **varchar**(32),

     deptId **int**,

     salary **float**);

### 2.1.3 创建外码约束（foreign key）

本关任务：创建MyDb数据库，为表定义主键，并给表staff创建外键，这个外键约束的名称为FK\_staff\_deptNo。

实现方法：使用primary key进行主码约束，然后使用foreign key(column) references (column)实现外码约束。

**create** **database** MyDb;

use MyDb;

**create** **table** dept(deptNo **int** **primary** **key**, deptName **varchar**(32));

**create** **table** staff(

staffNo **int** **primary** **key**, staffName **varchar**(32),

     gender **char**(1), dob **date**,

     salary **numeric**(8, 2), deptNo **int**,

**constraint** FK\_staff\_deptNo **foreign** **key**(deptNo) **references** dept(deptNo));

### 2.1.4 CHECK约束

本关任务：创建表并对表加上特定约束。

实现方法：根据任务要求，创建表并添加 check 约束，代码如下。

**create** **database** MyDb;

use MyDb;

**create** **table** products(

     pid **char**(10) **primary** **key**,

**name** **varchar**(32),

     brand **char**(10) **constraint** CK\_products\_brand **check**(brand in ('A', 'B')),

     price **int** **constraint** CK\_products\_price **check**(price > 0));

### 2.1.5 DEFAULT约束

该关卡已完成，此报告略过。

### 2.1.6 UNIQUE约束

该关卡已完成，此报告略过。

## 2.2 表结构与完整性约束的修改（Alter）

本节主要围绕数据库的修改进行，主要使用alter语句对表的数据和约束条件进行增删改。

### 2.2.1 修改表名

该关卡已完成，此报告略过。

### 2.2.2 添加与删除字段

本关任务：向表中增加或删除部分数据。

实现方法：依照 ALTER TABLE 表名[修改事项[,修改事项]...]格式。

*-- 语句1：删除表orderDetail中的列orderDate*

**alter** **table** orderDetail **drop** orderDate;

*-- 语句2：添加列unitPrice*

**alter** **table** orderDetail **add** unitPrice **numeric**(10, 2);

### 2.2.3 修改字段

该关卡已完成，此报告略过。

### 2.2.4 添加、删除与修改约束

本关任务：对表的约束进行修改操作。

实现方法：依照constraint的语句格式进行表的修改操作，代码如下：

*-- (1) 为表Staff添加主码*

**alter** **table** Staff **add** **primary** **key** (staffNo);

*-- (2) Dept.mgrStaffNo是外码，对应的主码是Staff.staffNo:*

**alter** **table** Dept **add** **constraint** FK\_Dept\_mgrStaffNo **foreign** **key**(mgrStaffNo) **references** Staff(staffNo);

*-- (3) Staff.dept是外码，对应的主码是Dept.deptNo:*

**alter** **table** Staff **add** **constraint** FK\_Staff\_dept **foreign** **key**(dept) **references** Dept(deptNo);

*-- (4) 为表Staff添加check约束，规则为：gender的值只能为F或M；*

**alter** **table** Staff **add** **constraint** CK\_Staff\_gender **check** (gender in ('F', 'M'));

*-- (5) 为表Dept添加unique约束：deptName不允许重复。约束名为UN\_Dept\_deptName：*

**alter** **table** Dept **add** **constraint** UN\_Dept\_deptName **unique**(deptName);

## 2.3 基于金融应用的数据查询(Select)

本节的主要任务是对若干个表的数据进行查询操作，主要考察了select语句的各种用法，在具体的情境之中设计了单表和多表的直接、嵌套等查询。

### 2.3.1 金融应用场景介绍,查询客户主要信息

本关任务：查询所有客户的名称、手机号和邮箱信息。

实现方法：最简单、基础的select语句查询。

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

### 2.3.2 邮箱为null的客户

本关任务：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。

实现方法：基础的where语句使用

**select** c\_id, c\_name, c\_id\_card, c\_phone **from** client **where** c\_mail **is** null;

### 2.3.3 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。

实现方法：使用exists语句，将是否购买该种类的资产转化为两个判断。

**select** c\_name, c\_mail, c\_phone **from** client

**where** exists(**select**\* **from** property **where** client.c\_id=pro\_c\_id and pro\_type='3')

and exists(**select**\* **from** property **where** client.c\_id=pro\_c\_id and pro\_type='2')

**order** **by** c\_id;

### 2.3.4 办理了储蓄卡的客户信息

本关任务：查询办理了储蓄卡的客户名称、手机号和银行卡号。

实现方法：基础多表连接操作。

**select** c\_name, c\_phone, b\_number **from** client, bank\_card

**where** client.c\_id = bank\_card.b\_c\_id and b\_type = '储蓄卡'

**order** **by** c\_id;

### 2.3.5 每份金额在30000～50000之间的理财产品

该关卡已完成，此报告略过。

### 2.3.6 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。

实现方法：根据pro\_income进行分组，使用count函数统计数量，再对count函数结果使用max函数统计众数。

**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

本关任务：查询未购买任何理财产品的武汉居民的信息。

实现方法：使用like语句通过身份证号“4201%”筛选出武汉居民，然后再使用not exists语句统计未购买居民。

**select** c\_name, c\_phone, c\_mail **from** client

**where** c\_id\_card like "4201%"

and not exists (

**select** \*

**from** property

**where** pro\_c\_id = c\_id and pro\_type = 1 )

**order** **by** c\_id;

### 2.3.8 持有两张信用卡的用户

该关卡已完成，此报告略过。

### 2.3.9 购买了货币型基金的客户信息

该关卡已完成，此报告略过。

### 2.3.10 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。

实现方法：前三名可以通过排序后limit语句限制输出条数。统计收益通过c\_id进行分组后sum聚集函数统计。注意只能统计可用资产的收益。

**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** pro\_c\_id

**order** **by** sum(pro\_income) **desc**

limit 3;

### 2.3.11 黄姓客户持卡数量

该关卡已完成，此报告略过。

### 2.3.12 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。

实现方法：将property表中投资项首先根据不同的投资类型与对应的三种资产信息表进行自然连接，得到每份理财产品的总投资额（单价乘以购买份额）后再将三种资产表进行集合信息的合并，最后根据pro\_c\_id进行分组统计。最后的分组需要使用左连接：因为可能该人并未购买该类型的产品。

**select** c\_name, c\_id\_card, ifnull(sum(pro\_amount), 0) **as** total\_amount **from** client

left join (

**select** pro\_c\_id, pro\_quantity \* p\_amount **as** pro\_amount

**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** pro\_amount

**from** property, insurance

**where** pro\_pif\_id = i\_id and pro\_type = 2

**union** all

**select** pro\_c\_id, pro\_quantity \* f\_amount **as** pro\_amount

**from** property, fund

**where** pro\_pif\_id = f\_id and pro\_type = 3

     ) **as** pro

**on** pro.pro\_c\_id = c\_id

**group** **by** c\_id

**order** **by** total\_amount **desc**;

### 2.3.13 客户总资产

该关卡已完成，此报告略过。

### 2.3.14 第N高问题

本关任务：查询每份保险金额第4高保险产品的编号和保险金额。

实现方法：排序后使用limit子句中的参数（从第几个开始，取多少个）。可以查询出去重后第四高的金额，再直接根据此金额进行筛选。

**select** i\_id, i\_amount  **from** insurance

**where** i\_amount = (

**select** **distinct** i\_amount

**from** insurance

**order** **by** i\_amount **desc**

     limit 1 offset 3 );

### 2.3.15 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。

实现方法：MySQL中内置的两个函数：rank()进行并列名次相同且跳过的标号，dense\_rank()进行并列名次相同且标号连续的标号。下仅以rank()为例。

**select** pro\_c\_id, sum(pro\_income) **as** total\_revenue, rank() over(**order** **by** sum(pro\_income) **desc**) **as** "rank" **from** property

**where** pro\_type = 3

**group** **by** pro\_c\_id

**order** **by** total\_revenue **desc**, pro\_c\_id;

### 2.3.16 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。

实现方法：先按用户分组，再利用 group\_concat() 函数将用户所购买的基金编号去重排序后拼接成一个字符串f\_id 。由于需要重复用到上述表，所以利用with子句创建一个公用临时表，如果表中两个客户的f\_id相同则说明这两个客户的基金组合完全相同。

**with** pro(c\_id, f\_id) **as** (

**select** pro\_c\_id **as** c\_id,

group\_concat(**distinct** pro\_pif\_id **order** **by** pro\_pif\_id) **as** f\_id

**from** property **where** pro\_type = 3 **group** **by** pro\_c\_id )

**select** t1.c\_id **as** c\_id1, t2.c\_id **as** c\_id2

**from** pro **as** t1, pro **as** 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

本关任务：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，则称这连续几日为投资者购买基金的高峰期。

实现方法：考虑首先将购买金额超过100万的天数选出来对交易天数（从2022年1月1日开始计算，记为daycnt）进行标号，记为rownum，如果是购买高峰期，则该高峰期内的交易天数和标号必然同时连续，则交易日天数-标号的值必然相同。因而根据这一差值进行分组，如果某一组的值个数大于等于三则可认为是高峰期。注意到2022年2月只有第一周因为春节假期而休市，因而统计自2022年1月1日的交易日可以直接通过MySQL内置的datediff函数统计天数，快速计算出daycnt的值。

**select** t3.t **as** pro\_purchase\_time, t3.amount **as** total\_amount

**from** (

**select** \*, count(\*) over(partition **by** t2.workday - t2.rownum) cnt

**from** (

**select** \*, row\_number() over(**order** **by** workday) rownum

**from** (

**select** pro\_purchase\_time t,

sum(pro\_quantity \* f\_amount) amount,

              @row := datediff(pro\_purchase\_time, "2021-12-31") - 2 \* week(pro\_purchase\_time) workday

**from** property, fund, (**select** @row) a

**where** pro\_purchase\_time like "2022-02-%"

             and pro\_type = 3

             and pro\_pif\_id = f\_id

**group** **by** pro\_purchase\_time

         ) **as** t1

**where** amount > 1000000

     ) **as** t2

) **as** t3

**where** t3.cnt >= 3;

### 2.3.18 至少有一张信用卡余额超过 5000 元的客户信用卡总金额

该关卡已完成，此报告略过。

### 2.3.19 以日历表格式显示每日基金购买总金额

本关任务：以日历表格式显示 2022 年 2 月每周每日基金购买总金额。

实现方法：对资产表property和基金表fund做自然连接，并按交易时间分组，计算出购买总金额，再利用week()函数和weekday()函数获取当天是当年的第几周和当天是周几。最后按照周次分组，对weekday()的结果使用if语句判断将金额放在哪一列输出即可。

**select**

     wk **as** week\_of\_trading,

     sum(if(dayId = 0, amount, null)) **as** Monday,

     sum(if(dayId = 1, amount, null)) **as** Tuesday,

     sum(if(dayId = 2, amount, null)) **as** Wednesday,

     sum(if(dayId = 3, amount, null)) **as** Thursday,

     sum(if(dayId = 4, amount, null)) **as** Friday

**from** (

**select**

         week(pro\_purchase\_time) - 5 **as** wk,

         weekday(pro\_purchase\_time) **as** dayId,

         sum(pro\_quantity \* f\_amount) **as** amount

**from** property, fund

**where**

         pro\_type = 3

         and pro\_pif\_id = f\_id

         and pro\_purchase\_time like "2022-02-%"

**group** **by** pro\_purchase\_time

) **as** t

**group** **by** wk;

## 2.4 数据查询(Select)--新增

本节内容在上一节SELECT的背景和基础之上难度有所提高，仍然关注于数据的查询工作。

### 2.4.1 查询销售总额前三的理财产品

本关任务：查询2010年和2011年这两年每年销售总额前3名（如果有并列排名，则后续排名号跳过之前的并列排名个数，例如1、1、3）的统计年份（pyear）、销售总额排名值(rk)、理财产品编号(p\_id)、销售总额(sumamount)。

实现方法：首先从property表和finances\_product表中筛选出2010年和2011年（通过year(pro\_purchase\_time)提取年份）的理财产品销售记录（pro\_type=1），按产品编号(p\_id)和年份分组计算每款产品每年的销售总额（pro\_quantity\*p\_amount的求和）；然后使用窗口函数rank()对每年的产品按销售总额降序排名，处理并列情况时采用标准排名方式（如1、1、3）；最后筛选出每年排名前三的产品记录，并按年份、排名和产品编号排序输出结果。

**select** \* **from**(

**select**

         pyear,

         rank() over(partition **by** pyear **order** **by** sumamount **desc**) **as** rk,

         p\_id,

         sumamount

**from** (

**select**

             year(pro\_purchase\_time) **as** pyear,

             p\_id,

             sum(pro\_quantity \* p\_amount) **as** sumamount

**from** property, finances\_product

**where** pro\_pif\_id = p\_id

         and pro\_type = 1

         and year(pro\_purchase\_time) in (2010, 2011)

**group** **by** p\_id, pyear

     ) **as** t1

) **as** t2

**where** rk<=3

**order** **by** pyear, rk, p\_id;

### 2.4.2 投资积极且偏好理财类产品的客户

本关任务：购买了3种（同一编号的理财产品记为一种）以上理财产品的客户被认为投资积极的客户，若该客户持有基金产品种类数（同一基金编号记为相同的基金产品种类）小于其持有的理财产品种类数，则认为该客户为投资积极且偏好理财产品的客户。查询所有此类客户的编号(pro\_c\_id)。

实现方法：从property表和finances\_product表中筛选出投资理财产品的客户，统计每位客户持有的不同理财产品数量（按pro\_pif\_id去重计数），记为t1临时表；同时从property表和fund表中筛选出投资基金产品的客户，统计每位客户持有的不同基金产品数量（按pro\_pif\_id去重计数），记为t2临时表。然后将这两个临时表按客户编号(pro\_c\_id)进行关联，筛选出同时出现在两个临时表中、且理财产品持有种类数(cnt1)大于基金产品持有种类数(cnt2)的客户记录，最后按客户编号升序排列输出结果。

**select** t1.pro\_c\_id

**from** (

( **select** pro\_c\_id, count(**distinct**(pro\_pif\_id)) **as** cnt1

**from** property, finances\_product

**where** pro\_type = 1 and pro\_pif\_id = p\_id **group** **by** pro\_c\_id ) **as** t1,

( **select** pro\_c\_id, count(**distinct**(pro\_pif\_id)) **as** cnt2

**from** property, fund

**where** pro\_type = 3 and pro\_pif\_id = f\_id **group** **by** pro\_c\_id ) **as** t2 )

**where** t1.pro\_c\_id = t2.pro\_c\_id

and t1.cnt1 > t2.cnt2

**order** **by** t1.pro\_c\_id;

### 2.4.3 查询购买了所有畅销理财产品的客户

本关任务：若定义持有人数超过2的理财产品称为畅销理财产品。查询购买了所有畅销理财产品的客户编号(pro\_c\_id)。

实现方法：首先定义畅销理财产品为持有人数超过2的理财产品（通过分组统计pro\_pif\_id并筛选count(\*)>2的记录确定），然后使用NOT EXISTS子查询确保客户不存在任何未购买的畅销理财产品，即对于每个客户检查是否存在某个畅销理财产品不在该客户的购买记录中，若不存在这样的产品则说明该客户购买了所有畅销理财产品，最后返回满足条件的客户编号并去除重复值。

**select** **distinct** pro\_c\_id **from** property **as** p1

**where** not exists(

**select** \*

**from** (

**select** pro\_pif\_id **from** property

**where** pro\_type = 1

**group** **by** pro\_pif\_id

**having** count(\*) > 2

    ) **as** t1

**where** t1.pro\_pif\_id not in (

**select** pro\_pif\_id **from** property **as** p2

**where** p1.pro\_c\_id = p2.pro\_c\_id and p2.pro\_type = 1 ));

### 2.4.4 查找相似的理财产品

本关任务：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客购买客户总人数（cc）以及该理财产品对于14 号理财产品的相似度排名值（prank）。

实现方法：筛选出14号理财产品购买量排名前三的客户（通过dense\_rank()窗口函数按pro\_quantity升序排名确定），然后找出这些客户购买过的其他理财产品（排除14号产品本身），统计这些相似理财产品的购买客户总人数并按人数降序排列，最后使用dense\_rank()为相似度排名（避免排名号跳跃），最终输出相似理财产品编号、购买客户总人数和相似度排名值，并按客户人数降序和产品编号升序排列结果。

**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

                 ) **as** fin\_rk

**where** fin\_rk.rk <= 3))

**group** **by** pro\_pif\_id

**order** **by** cc **desc**, pro\_pif\_id;

### 2.4.5 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。

实现方法：通过自连接property表（分别别名为a和b）匹配不同客户但持有相同理财产品的记录，筛选条件限定为理财产品类型，然后按客户对分组统计共同持有的理财产品种类数，保留共同持有2种及以上产品的客户对，最后按第一个客户编号升序排列输出结果，展示客户对编号及共同持有的产品种数。

**select** a.pro\_c\_id, b.pro\_c\_id,count(\*) **as** total\_count

**from** property a, property b

**where** a.pro\_c\_id <> b.pro\_c\_id

     and a.pro\_type = 1

     and b.pro\_type = 1

     and a.pro\_pif\_id = b.pro\_pif\_id

**group** **by** a.pro\_c\_id, b.pro\_c\_id

**having** count(\*) >= 2

**order** **by** a.pro\_c\_id;

### 2.4.6 查找相似的理财客户

该关卡已完成，此报告略过。

## 2.5 数据的插入、修改与删除(Insert,Update,Delete)

本节的 6 个关卡围绕数据修改的三种语句Insert, Update, Delete语句在不同场景下的应用展开。

### 2.5.1 插入多条完整的客户信息

该关卡已完成，此报告略过。

### 2.5.2 插入不完整的客户信息

本关任务：向客户表client插入一条数据不全的记录。

实现方法：insert语句中指明列名和对应信息即可，剩余信息为NULL。

**insert** nto client (c\_id, c\_name, c\_phone, c\_id\_card, c\_password)

**values**(33, "蔡依婷", "18820762130", "350972199204227621", "MKwEuc1sc6");

### 2.5.3 批量插入数据

本关任务：已知表new\_client保存了一批新客户信息，该表与client表结构完全相同。将new\_client表的全部客户信息插入到客户表(client)。

实现方法：insert语句可以插入一整张表。

**insert** **into** client **select** \* **from** new\_client;

### 2.5.4 删除没有银行卡的客户信息

本关任务：删除在本行没有银行卡的客户信息。  
直接使用 delete 语句和 not in 子查询即可。

**delete** **from** client

**where** c\_id not in (

**select** b\_c\_id **from** bank\_card );

### 2.5.5 冻结客户资产

本关任务：请用一条update语句将客户手机号码为13686431238的投资资产（理财、保险与基金）的状态置为“冻结”。

实现方法：使用where子句选择该客户的信息进行修改。

**update** property **set** pro\_status="冻结"

**where** pro\_c\_id in (

**select** c\_id **from** client **where** c\_phone="13686431238" );

### 2.5.6 连接更新

本关任务：根据client表中提供的身份证号(c\_id\_card)，填写property表中对应的身份证号信息(pro\_id\_card)。

实现方法：可以使用set语句的查询更新实现。为得到身份证号信息可以使用连接查询。

**update** property,client **set** pro\_id\_card=c\_id\_card **where** pro\_c\_id = c\_id;

## 2.6 视图

本节主要围绕视图的创建与使用展开。

### 2.6.1 创建所有保险资产的详细记录视图

本关任务：创建所有保险资产的详细记录视图。

实现方法：根据任务要求编写select语句，并在前面加上create view ... as。

**create** **view** v\_insurance\_detail (c\_name, c\_id\_card, i\_name, i\_project, pro\_status, pro\_quantity, i\_amount, i\_year, pro\_income, pro\_purchase\_time)

**as**

**select** c\_name, c\_id\_card, i\_name, i\_project, pro\_status, pro\_quantity, i\_amount, i\_year, pro\_income, pro\_purchase\_time

**from** client, insurance, property

**where** c\_id = pro\_c\_id and

     pro\_type = 2 and

     pro\_pif\_id = i\_id;

### 2.6.2 基于视图的查询

本关任务：基于视图v\_insurance\_detail查询保险资产的总额和保险总收益。

实现方法：将视图当做普通的表一样 select 查询即可。

**select** c\_name, c\_id\_card, sum(i\_amount\*pro\_quantity) **as** insurance\_total\_amount, sum(pro\_income) **as** insurance\_total\_revenue

**from** v\_insurance\_detail

**group** **by** c\_name,c\_id\_card

**order** **by** insurance\_total\_amount **desc**;

## 2.7 存储过程与事务

本节分别涉及使用流程控制语句的存储过程、使用游标的存储过程和使用事务的存储过程。

### 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。

实现方法：使用一条with子句编写递归过程，通过此递归过程产生连续的三项fibonacci数，通过此语句计算fibonacci数列的前n项。注意fibonacci的边界条件第一项为1。

delimiter $$

**create** **procedure** sp\_fibonacci(in m **int**)

**begin**

**declare** x0, x1, i, xt **int**;

**insert** **into** fibonacci **values** (0, 0);

**set** x0 = 0, x1 = 1, i = 1;

while i<m DO

**set** xt = x0 + x1;

**insert** **into** fibonacci **values** (i, x1);

**set** x0 = x1;

**set** x1 = xt;

**set** i = i + 1;

**end** while;

**end** $$

delimiter ;

### 2.7.2 使用游标的存储过程

该关卡已完成，此报告略过。

### 2.7.3 使用事务的存储过程

本关任务：编写实现转账功能的存储过程。

实现方法：根据题意编写判断条件，如果不合法则将返回值置为0，并直接利用leave退出事务。如果合法，则先判断付款方和收款方的卡类型，如果是信用卡则要把转账金额设置为负数，最后使用update更新即可。

**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 ;

## 2.8 触发器

本节主要内容为触发器的应用。

### 2.8.1 为 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

本关任务：为资产表property编写一个触发器，以实现任务所要求的完整性业务规则。

实现方法：声明BEFORE INSERT ON property类型的触发器，首先判断是否存在该类型的投资产品，然后判断该p\_id是否为对应类型的投资产品，如果报错信息为空则说明没有错误。

delimiter $$

**create** **trigger** before\_property\_inserted before **insert** **on** property

**for** each row

**begin**

**declare** msg **varchar**(50);

if new.pro\_type = 1 and not exists (

**select** \* **from** finances\_product

**where** p\_id = new.pro\_pif\_id) **then**

**set** msg = concat("finances product #",new.pro\_pif\_id," not found!");

    signal sqlstate '45000' **set** message\_text = msg;

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!");

    signal sqlstate '45000' **set** message\_text = msg;

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!");

    signal sqlstate '45000' **set** message\_text = msg;

elseif 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;

**end**$$

delimiter ;

## 2.9 用户自定义函数

本节的主要应用为用户自定义函数的创建和使用。

### 2.9.1 创建函数并在语句中使用它

本关任务：编写一个依据客户编号计算其在本金融机构的存储总额的函数,并在SELECT语句使用这个函数。

实现方法：使用create function语句创建统计给定客户编号，统计存储总额的函数（函数内部使用select语句实现），然后再进行调用即可。

**set** **global** log\_bin\_trust\_function\_creators=1;

**drop** **function** IF EXISTS get\_deposit;

*-- 用create function语句创建符合以下要求的函数：*

*-- 依据客户编号计算该客户所有储蓄卡的存款总额。*

*-- 函数名为：get\_Records。函数的参数名可以自己命名:*

delimiter $$

**create** **function** get\_deposit(client\_id **int**)

**returns** **numeric**(10,2)

**begin**

**declare** total\_dep **int**;

**select** SUM(b\_balance) **into** total\_dep

**from** bank\_card

**where** b\_c\_id = client\_id and b\_type = "储蓄卡";

**return** total\_dep;

**end**$$

delimiter ;

*-- 应用该函数查询存款总额在100万以上的客户身份证号，姓名和存储总额*

*-- 结果依存款总额从高到低排序*

**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

### 2.10.1 用户和权限

本关任务：完成创建用户和授权操作。

实现方法：根据任务要求合理使用create, grant, revoke语句编写即可。

*-- (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\_name, c\_mail, 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

本关任务： 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。

实现方法：根据create和grant语句按要求实现即可。

*-- (1) 创建角色client\_manager和fund\_manager；*

**drop** role if exists client\_manager, fund\_manager;

**create** role client\_manager, fund\_manager;

*-- (2) 授予client\_manager对client表拥有select,insert,update的权限；*

**grant** **select**, **insert**, **update**

**on** **table** client

**to** client\_manager;

*-- (3) 授予client\_manager对bank\_card表拥有查询除银行卡余额外的select权限；*

**grant** **select**(b\_c\_id, b\_number, b\_type)

**on** **table** bank\_card

**to** client\_manager;

*-- (4) 授予fund\_manager对fund表的select,insert,update权限；*

**grant** **select**, **insert**, **update**

**on** **table** fund

**to** fund\_manager;

*-- (5) 将client\_manager的权限授予用户tom和jerry；*

**grant** client\_manager

**to** tom, jerry;

*-- (6) 将fund\_manager权限授予用户Cindy.*

**grant** fund\_manager

**to** Cindy;

## 2.11 并发控制与事务的隔离级别

本节的6个关卡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。

### 2.11.1 并发控制与事务的隔离级别

本关任务：设置事务隔离级别。

实现方法：MySQL事务隔离级别从高到低为：

1. READ UNCOMMITTED（读未提交）
2. READ COMMITTED（读已提交）
3. REPEATABLE READ（可重复读）
4. SERIALIZABLE（可串行化）

各事务隔离级别说明如表2-1：

**表2-1 MySQL事务隔离级别说明表**

|  |  |  |  |
| --- | --- | --- | --- |
| **隔离级别** | **读脏** | **不可重复读** | **幻读** |
| READ UNCOMMITTED | √ | √ | √ |
| READ COMMITTED | × | √ | √ |
| REPEATABLE READ | × | × | √ |
| SERIALIZABLE | × | × | × |

最低的隔离级别不能避免读脏、不可重复读和幻读，而最高的隔离级别可保证多个并发事务的任何调度，都不会产生数据的不一致性，但其代价是并发度最低。根据任务要求将事务的隔离级别设置为read uncommitted，并以rollback语句结束事务。

*-- 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

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。

实现方法：让第一个事务等待第二个事务修改而未commit的时候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。

*-- 事务1:*

use testdb1;

**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**;

*-- 事务2*

use testdb1;

**set** session **transaction** **isolation** **level** **read** **uncommitted**;

start **transaction**;

*-- 时刻1 - 事务2修改航班余票*

**update** ticket **set** tickets = tickets - 1 **where** flight\_no = 'CA8213';

*-- 时刻3 - 事务2 取消本次修改*

*-- 添加代码，使事务1在事务2撤销前读脏;*

**set** @n = sleep(2);

**rollback**;

### 2.11.3 不可重复读

该关卡已完成，此报告略过。

### 2.11.4 幻读

该关卡已完成，此报告略过。

### 2.11.5 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。

实现方法：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。

*-- 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**;

### 2.11.6 可串行化

本关任务：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。

*-- 事务1:*

use testdb1;

start **transaction**;

**set** @n = sleep(1);

**select** tickets **from** ticket **where** flight\_no = 'MU2455';

**select** tickets **from** ticket **where** flight\_no = 'MU2455';

**commit**;

*-- 事务2:*

use testdb1;

start **transaction**;

**update** ticket **set** tickets = tickets - 1 **where** flight\_no = 'MU2455';

**commit**;

## 2.12 备份+日志：介质故障与数据库恢复

### 2.12.1 备份与恢复

本关任务：设有居民人口登记数据库residents,请为该数据库做一次静态的(你一个人独享服务器)海量逻辑备份，备份文件命名为residents\_bak.sql。 然后再用该逻辑备份文件恢复数据库。

*-- 对数据库residents作海量备份,备份至文件residents\_bak.sql:*

mysqldump -h127.0.0.1 -uroot --databases residents>residents\_bak.sql

*-- 利用备份文件residents\_bak.sql还原数据库:*

mysql -h127.0.0.1 -uroot < residents\_bak.sql

### 2.12.2 备份+日志：介质故障的发生与数据库的恢复

本关任务：摸拟介质故障的发生与数据库的恢复。

实现方法：由于需要对数据库train作逻辑备份并新开日志文件，所以需要在上一关的基础上mysqldump指令中加入‐‐flush‐logs参数来新开日志文件：

### mysqldump -h127.0.0.1 -uroot *--flush-logs --databases train > train\_bak.sql*

系统故障前的日志文件保存在log/binlog.000018中，因而除了checkpoint时刻产生的train\_bak.sql文件中的数据，还需要根据日志文件对部分事务进行重做和撤销操作。由于mysql的默认字符集为utf-8，与日志文件不兼容，因而需要使用--no-defaults参数来取消MySQL默认参数。

mysql -h127.0.0.1 -uroot < train\_bak.sql

mysqlbinlog *--no-defaults log/binlog.000018 | mysql -h127.0.0.1 -u root*

## 2.13 数据库设计与实现

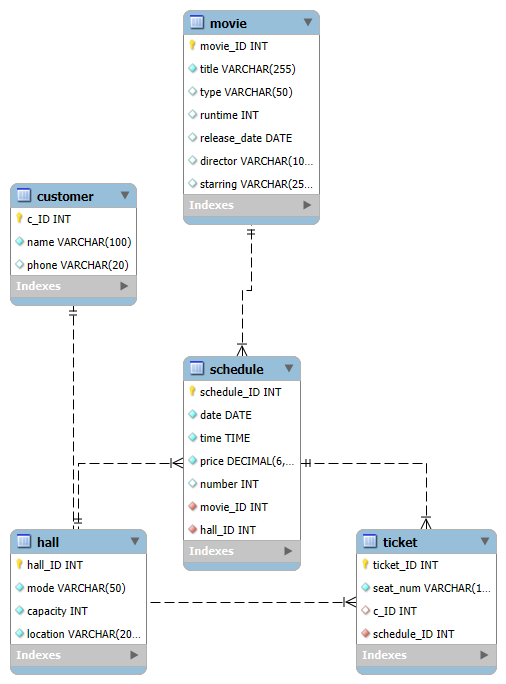
### 2.13.1 从概念模型到MySQL实现

该关卡已完成，此报告略过。

### 2.13.2 从需求分析到逻辑模型

本关任务：按影院管理系统要求画出E-R图，并给出对应的关系模式。

实现方法：直接根据要求进行模拟即可，如图2-1。



**图2-1** **影院管理系统E-R图**

并有如下关系模式：

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)

### 2.13.3 建模工具的使用

利用MySQL Workbench建模模块中的Forward engineering功能，自动导出SQL脚本。该关卡已完成，此报告略过。

## 2.14 数据库应用开发（JAVA篇）

### 2.14.1 JDBC体系结构和简单的查询

本关任务：查询client表邮箱非空的客户信息，列出客户姓名，邮箱和电话。

实现方法：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。

**public** **static** **void** main(String[] args) {

        Connection connection = **null**, Statement statement = **null**;

        ResultSet resultSet = **null**;

**try** {

            Class.forName(JDBC\_DRIVER);

            connection = DriverManager.getConnection(DB\_URL, USER, PASS);

            statement = connection.createStatement();

            resultSet = statement.executeQuery("select c\_name, c\_mail, c\_phone  from client where c\_mail is not null");

            System.out.println("姓名\t邮箱\t\t\t\t电话");

**while** (resultSet.next()) {

                System.out.print(resultSet.getString("c\_name") + "\t");

                System.out.print(resultSet.getString("c\_mail") + "\t\t");

                System.out.println(resultSet.getString("c\_phone"));

            }

        } **catch** (ClassNotFoundException e) {

            System.out.println("Sorry,can`t find the JDBC Driver!");

            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(); }

        }

    }

### 2.14.2 用户登录

本关任务：编写客户登录程序，提示用户输入邮箱和密码，并判断正确性，给出适当的提示信息。

实现方法：方法同上关，当密码不匹配时（查询集合为空）直接返回用户名或密码错误，以防止入侵。

statement = connection.createStatement();

String sql = "select \* from client where c\_mail = ? and c\_password = ?";

PreparedStatement ps = connection.prepareStatement(sql);

ps.setString(1, loginName), ps.setString(2, loginPass);

resultSet = ps.executeQuery();

**if** (resultSet.next()) System.out.println(" 登录成功。 ");

**else** System.out.println(" 用户名或密码错误！ ");

### 2.14.3 添加新客户

该关卡已完成，此报告略过。

### 2.14.4 银行卡销户

本关任务：实现银行卡销户的方法。

实现方法：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。

**public** **static** **int** removeBankCard(Connection connection,

**int** b\_c\_id, String b\_number){

**int** result = 0;

**try**{

        String sqlStatement = "DELETE FROM bank\_card WHERE b\_c\_id = "+b\_c\_id+" AND b\_number = \""+b\_number+"\";";

        Statement statement = connection.createStatement();

        result = statement.executeUpdate(sqlStatement);

    } **catch**(SQLException e){

         .printStackTrace();

    } **finally** {

**return** result;

    }

 }

### 2.14.5 客户修改密码

本关任务：实现客户修改密码的方法。

实现方法：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。

**public** **static** **int** passwd(Connection connection,

                             String mail,

                             String password,

                             String newPass){

**int** code = -1;

**try** {

        String sqlStatementSelect = "SELECT c\_password FROM client WHERE c\_mail = \""+mail+"\";";

        String sqlStatementUpdate = "UPDATE client SET c\_password = \""+newPass+"\" WHERE c\_mail = \""+mail+"\";";

        Statement statement = connection.createStatement();

        ResultSet resultSet = statement.executeQuery(sqlStatementSelect);

**if**(resultSet.next()) {

            String oldPassword = resultSet.getString("c\_password");

**if**(oldPassword.equals(password)) {

**int** result = statement.executeUpdate(sqlStatementUpdate);

**if**(result == 1) { code = 1; }

**else** { code = -1; }

            }

**else** { code = 3; }

        }

**else** { code = 2; }

    } **catch**(SQLException e){

        code = -1;

        e.printStackTrace();

    } **finally** { **return** code; }

}

### 2.14.6 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。

**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);

          preparedStatement.executeUpdate();

          preparedStatement = connection.prepareStatement(sql);

**double** rcv\_amount = resultSet.getString("b\_type").equals("信用卡") ? -amount : amount;

          preparedStatement.setDouble(1, rcv\_amount);

          preparedStatement.setString(2, destCard);

          preparedStatement.executeUpdate();

**return** **true**;

       } **catch** (SQLException e) { e.printStackTrace();}

**return** **false**;

   }

### 2.14.7 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。

实现方法：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。

**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();

        }

    }

# 3 课程总结

本次实验课程主要通过MySQL语言实现了数据库的建立、修改、查询、安全性、并发性等实际应用中数据库操作的训练，并增加了数据库设计、、数据库的Java应用开发等数据库周边应用，共78关卡。

在实验过程当中，我在数据查询select中遇到了一些困难，对于较难的关卡也是通过和其他同学进行交流后得到了大致的解题思路。在实现事务存储过程中，查阅了大量的资料。在数据库设计关卡中，任务要求的表和数据项非常繁杂，且制约关系非常复杂，因而在本关实现中也是遇到了不少的麻烦

本次实验的训练，让我熟练掌握了MySQL语句的使用，并充分感受到了MySQL语句的灵活性和使用便捷性，并且掌握了如何正确使用MySQL语句实现复杂的查询要求。在最后的Java实验中也通过实例说明常见的MySQL注入和相对应的防御措施。

我深刻体会到理论知识与工程实践的深度融合：事务隔离级别在并发场景中的具体表现、索引对复杂查询的性能优化作用等抽象概念，在代码调试中得到直观验证。金融查询任务中“持有相同基金组合客户识别”等综合题型，极大提升了解决实际问题的能力。本课程为构建高可靠、高性能数据库系统奠定了扎实基础，也为后续分布式系统学习提供了重要支撑。