

**数据库系统原理实践报告**

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： |  |
| 学 号： |  |
| 姓 名： |  |
| 指导教师： |  |

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

2022 年 12 月 7 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](#_Toc28206)

[2 任务实施过程与分析 2](#_Toc2287)

[2.1 数据库、表与完整性约束的定义(Create) 2](#_Toc31668)

[2.1.1 创建数据库 2](#_Toc31995)

[2.1.2 创建创建表及表的主码约束 2](#_Toc17493)

[2.1.3 创建外码约束(foreign key) 2](#_Toc17639)

[2.1.4 CHECK约束 3](#_Toc22210)

[2.1.5 DEFAULT约束 3](#_Toc24617)

[2.1.6 UNIQUE约束 3](#_Toc23036)

[2.2 表结构与完整性约束的修改 (ALTER) 3](#_Toc29082)

[2.2.1 修改表名 3](#_Toc1513)

[2.2.2 添加与删除字段 4](#_Toc14388)

[2.2.3 修改字段 4](#_Toc4510)

[2.2.4 添加，删除与修改约束 4](#_Toc15020)

[2.3 数据查询（Select）之一 5](#_Toc19725)

[2.3.1 金融应用场景介绍,查询客户主要信息 5](#_Toc9467)

[2.3.2 邮箱为null的客户 5](#_Toc2285)

[2.3.3 既买了保险又买了基金的客户 5](#_Toc3860)

[2.3.4办理了储蓄卡的客户信息 5](#_Toc18200)

[2.3.5每份金额在 30000∼50000 之间的理财产品 5](#_Toc24147)

[2.3.6 商品收益的众数 5](#_Toc16057)

[2.3.7 未购买任何理财产品的武汉居民 6](#_Toc16242)

[2.3.8 持有两张信用卡的用户 6](#_Toc23290)

[2.3.9 购买了货币型基金的客户信息 6](#_Toc25853)

[2.3.10 投资总收益前三名的客户 6](#_Toc20499)

[2.3.11 黄姓客户持卡数量 7](#_Toc22178)

[2.3.12客户理财、保险与基金投资总额 7](#_Toc494)

[2.3.13 客户总资产 7](#_Toc16341)

[2.3.14 第N高问题 8](#_Toc4274)

[2.3.15 基金收益两种方式排名 8](#_Toc24721)

[2.3.16 持有完全相同基金组合的客户 8](#_Toc17150)

[2.3.17 购买基金的高峰期 8](#_Toc6475)

[2.3.18 至少有一张信用卡余额超过5000元的客户信用卡总余额 8](#_Toc662)

[2.3.19 以日历表格式显示每日基金购买总金额 8](#_Toc4943)

[2.4 数据查询（Select）之二 8](#_Toc1993)

[2.4.1 查询销售额前三的理财产品 8](#_Toc20759)

[2.4.2 投资积极且偏好理财类产品的客户 8](#_Toc3686)

[2.4.3 查询购买了所有畅销理财产品的客户 9](#_Toc11908)

[2.4.4 查找相似的理财产品 9](#_Toc22057)

[2.4.5 查询任意两个客户的相同理财产品数 9](#_Toc12847)

[2.4.6 查找相似的理财客户 9](#_Toc9323)

[2.5 数据的插入、修改与删除（Insert, Update, Delete） 9](#_Toc32063)

[2.5.1 插入多条完整的客户信息 9](#_Toc23775)

[2.5.2 插入不完整的客户信息 9](#_Toc25298)

[2.5.3 批量插入数据 9](#_Toc23145)

[2.5.4 删除没有银行卡的客户信息 10](#_Toc17787)

[2.5.5 冻结客户资产 10](#_Toc16303)

[2.5.6 连接更新 10](#_Toc19462)

[2.6 视图 10](#_Toc28311)

[2.6.1 创建所有保险资产的详细记录视图 10](#_Toc18132)

[2.6.2 基于视图的查询 10](#_Toc31297)

[2.7 存储过程与事务 11](#_Toc30650)

[2.7.1 使用流程控制语句的存储过程 11](#_Toc4993)

[2.7.2 使用游标的存储过程 12](#_Toc12620)

[2.7.3 使用事务的存储过程 12](#_Toc15225)

[2.8 触发器 13](#_Toc3449)

[2.8.1 为投资表property实现业务约束规则-根据投资类别分别引用不同表的主码 13](#_Toc13389)

[2.9 用户自定义函数 14](#_Toc24065)

[2.9.1 创建函数并在语句中使用它 14](#_Toc19845)

[2.10 安全性控制 14](#_Toc785)

[2.10.1 用户和权限 14](#_Toc7752)

[2.10.2 用户、角色与权限 15](#_Toc5635)

[2.11 并发控制与事务的隔离级别 15](#_Toc7134)

[2.11.1 并发控制与事务的隔离级别 15](#_Toc10131)

[2.11.2 读脏 16](#_Toc16215)

[2.11.3 不可重复读 17](#_Toc20311)

[2.11.4 幻读 17](#_Toc16801)

[2.11.5 主动加锁保证可重复读 17](#_Toc19283)

[2.11.6 可串行化 18](#_Toc30273)

[2.12 备份+日志：介质故障与数据库恢复 18](#_Toc8282)

[2.12.1 备份与恢复 18](#_Toc22675)

[2.12.2 备份+日志：介质故障的发生与数据库的恢复 18](#_Toc9931)

[2.13 数据库设计与实现 19](#_Toc18472)

[2.13.1 从概念模型到MySQL实现 19](#_Toc11769)

[2.13.2 从需求分析到逻辑模型 19](#_Toc5374)

[2.13.3 建模工具的使用 19](#_Toc29685)

[2.13.4 制约因素分析与设计 19](#_Toc17069)

[2.13.5 工程师责任及其分析 19](#_Toc23707)

[2.14 数据库应用开发（JAVA篇） 20](#_Toc25020)

[2.14.1 JDBC体系结构和简单的查询 20](#_Toc27150)

[2.14.2 用户登录 22](#_Toc6316)

[2.14.3 添加新客户 23](#_Toc18949)

[2.14.4 银行卡销户 24](#_Toc657)

[2.14.5 客户修改密码 24](#_Toc9882)

[2.14.6 事务与转账操作 25](#_Toc30503)

[2.14.7 把稀疏表格转换为键值对存储 26](#_Toc7802)

[3 课程总结 28](#_Toc12478)

# 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，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、删除与修改等数据处理相关任务；
3. 数据库的安全性控制，完整性控制，恢复机制，并发控制机制等系统内核的实验；
4. 数据库的设计与实现；
5. 数据库应用系统的开发(JAVA 篇)。

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

# 2 任务实施过程与分析

## 2.1 数据库、表与完整性约束的定义(Create)

本节的六个关卡围绕数据库和表的创建展开，分别完成了数据库的创建、表的创建以及表中主码、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。

在开始操作前，需要连接到数据库，在命令行输入:

mysql ‐h host ‐u user ‐ppassword dbname

然后使用如下命令创建数据库：

CREATE database beijing2022;

### 2.1.2 创建创建表及表的主码约束

本关任务：在指定的数据库中创建一个表，并为表指定主码。

创建数据库 TestDb ，在 TestDb 下使用 create table 命令创建表 t\_emp ，并对主码 id附加 primary key 约束。

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：创建外码约束（参照完整性约束）。

按关卡要求创建 MyDb 数据库，并在其中创建 dept 表和 staff 表，并给表 staff 创建外键，

代码如下：

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)

    );

代码中创建了表 staff 的 deptNo 到表 dept 的 deptNo 的外键约束，约束名为FK\_staff\_deptNo 。

### 2.1.4 CHECK约束

本关任务：为表创建 CHECK 约束

根据任务要求，创建表并添加 check 约束，代码如下：

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

该关卡任务已完成，实施情况本报告略过。

### 2.1.6 UNIQUE约束

该关卡任务已完成，实施情况本报告略过。

## 2.2 表结构与完整性约束的修改 (ALTER)

本节的四个关卡围绕数据库中表的基本修改操作展开，设计用 alter 语句对表的定义进行修改，如更换/修改表名、列名、列的类型、列约束、表约束；添加或删除列、约束等。

### 2.2.1 修改表名

本关任务：修改表的名称，将表名 your\_table 更改为 my\_table 。

ALTER TABLE 的简化的语法为：

1. ALTER TABLE 表名
2. [修改事项 [, 修改事项] ...]

根据任务要求选择 “RENAME [TO|AS] 新表名” 修改事项，代码如下：

ALTER table your\_table rename my\_table;

### 2.2.2 添加与删除字段

本关任务：删除表 orderDetail 中的列 orderDate ，添加列 unitPrice 。

根据任务要求选择“DROP [COLUMN] 列名”和“ADD [COLUMN] 列名 数据类型 [列约束]”

修改事项，代码如下：

#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

本关任务：修改指定字段的类型和名称。

根据任务要求选择 “MODIFY [COLUMN] 列名 数据类型 [列约束]” 和 “RENAME COLUMN列名 TO 新列名” 修改事项，代码如下：

ALTER TABLE addressBook MODIFY QQ char(12);

ALTER TABLE addressBook RENAME COLUMN  weixin TO wechat;

### 2.2.4 添加，删除与修改约束

本关任务：添加、删除与修改约束。

对每个要求修选取合适的修改事项即可，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,请添加这个外码，名字为FK\_Dept\_mgrStaffNo:

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. 请添加这个外码，名字为FK\_Staff\_dept:

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；约束名为CK\_Staff\_gender:

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 数据查询（Select）之一

本节十九个关卡对同一个database进行由易到难各种不同场景的查询应用。

### 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，查询结果按照客户编号排序。

基础的 select 查询，代码如下：

SELECT c\_name, c\_phone, c\_mail FROM client ORDER BY c\_id;

### [2.3.2](https://www.educoder.net/tasks/btxm4ukr3whf) 邮箱为null的客户

本关任务：查询client表中没有填写邮箱的客户的编号，名称，身份证号，手机号

使用 where 字句即可完成该条件查询，代码如下：

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。结果按c\_id排序。

使用多条件判断子查询即可，代码如下：

SELECT c\_name, c\_mail, c\_phone

FROM client

INNER JOIN property a

INNER JOIN property b

ON a.pro\_type = 2 and b.pro\_type = 3 and a.pro\_c\_id = client.c\_id and a.pro\_c\_id = b.pro\_c\_id

GROUP BY client.c\_id

ORDER BY c\_id;

### [2.3.4办理了储蓄卡的客户信息](https://www.educoder.net/tasks/8xsmial2796f)

该关卡任务已完成，实施情况本报告略过。

### 2.3.5每份金额在 30000∼50000 之间的理财产品

该关卡任务已完成，实施情况本报告略过。

### 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。

在 property 表中基于 pro\_income 分组，利用 count 函数统计数量，如果某个分组数量大于等于所有分组或者数量最大的分组，则为众数。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);

### 2.3.7 未购买任何理财产品的武汉居民

本关任务：查询身份证隶属武汉市没有买过任何理财产品的客户的名称、电话号、邮箱。

湖北省武汉市居民的身份证号开头为 “4201”，利用 “like "4201%"” 即可筛选出武汉居民，再利用 exists 查询是否购买理财产品即可。代码如下：

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

该关卡任务已完成，实施情况本报告略过。

### 2.3.9 购买了货币型基金的客户信息

该关卡任务已完成，实施情况本报告略过。

### 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。

利用 “inner join” 对用户表 client 和资产表 property 做自然连接，然后按 c\_id 分组统计收益和并降序排序，利用 “limit 3” 输出前 3 名用户即可。代码如下：

SELECT c\_name, c\_id\_card, SUM(pro\_income) AS total\_income

FROM client

INNER JOIN property

WHERE pro\_c\_id = c\_id

AND pro\_status = '可用'

GROUP BY c\_id

ORDER BY total\_income desc

LIMIT 3;

### 2.3.11 黄姓客户持卡数量

该关卡任务已完成，实施情况本报告略过。

### 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。

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)，选取每种商品的类型和该商品每份的价格进行求和，按照总金额降序进行排序。代码如下：

select c1.c\_name,c1.c\_id\_card,sum(ifnull(all\_amount.am\_sum,0)) as total\_amount

from client c1

left join

(SELECT c.c\_id, sum(pro.pro\_quantity\*f.f\_amount) as am\_sum

from fund f join property pro join client c

where pro.pro\_pif\_id = f\_id and pro\_c\_id = c\_id and pro\_type = 3 and c.c\_id = pro.pro\_c\_id

group by c.c\_id

    UNION all

        SELECT c.c\_id, sum(pro.pro\_quantity\*i.i\_amount) as am\_sum

        from insurance i join property pro join client c

        where pro.pro\_pif\_id = i\_id and pro\_c\_id = c\_id and pro\_type = 2 and c.c\_id = pro.pro\_c\_id

        group by c.c\_id

    UNION all

        SELECT c.c\_id, sum(pro.pro\_quantity\*p.p\_amount) as am\_sum

        from finances\_product p join property pro join client c

        where pro.pro\_pif\_id = p\_id and pro\_type = 1 and pro\_c\_id = c\_id and c.c\_id = pro.pro\_c\_id

        group by c.c\_id)

        as all\_amount

on c1.c\_id = all\_amount.c\_id

group by c1.c\_id,c1.c\_name,c1.c\_id\_card

order by total\_amount desc;

### 2.3.13 客户总资产

跳过。

### 2.3.14 第N高问题

跳过。

### 2.3.15 基金收益两种方式排名

跳过。

### 2.3.16 持有完全相同基金组合的客户

跳过。

### 2.3.17 购买基金的高峰期

跳过。

### 2.3.18 至少有一张信用卡余额超过5000元的客户信用卡总余额

跳过。

### 2.3.19 以日历表格式显示每日基金购买总金额

跳过。

## 2.4 数据查询（Select）之二

### 2.4.1 查询销售额前三的理财产品

本关任务：查询销售总额前三的理财产品

使用rank over (partition by ... order by)对销售总额进行rank排序，然后挑选2010和2011年两年每年销售总额的前三名的统计年份，销售总额排名值，理财产品编号和销售总额。代码如下：

SELECT \* FROM(

SELECT YEAR(pro\_purchase\_time) as pyear, rank() over(partition by YEAR(pro\_purchase\_time) order by p\_amount\*pro\_quantity desc) as rk, p\_id, p\_amount\*pro\_quantity as sumamount

FROM finances\_product

INNER JOIN property

WHERE pro\_type = 1 and pro\_pif\_id = p\_id and (pro\_purchase\_time like '2010%' or pro\_purchase\_time like '2011%')

ORDER BY pyear, rk, p\_id) P

WHERE P.rk <= 3;

### 2.4.2 投资积极且偏好理财类产品的客户

跳过。

### 2.4.3 查询购买了所有畅销理财产品的客户

跳过。

### 2.4.4 查找相似的理财产品

跳过。

### 2.4.5 查询任意两个客户的相同理财产品数

跳过。

### 2.4.6 查找相似的理财客户

跳过。

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

本节的六个关卡主要聚焦于对Insert, Update, Delete语句在不同场景下的基本使用方法。

### 2.5.1 插入多条完整的客户信息

本关任务：向客户表client插入数据

利用Insert into语句完成，代码如下：

INSERT INTO client (c\_id, c\_name, c\_mail, c\_id\_card, c\_phone, c\_password)

VALUES (1, '林惠雯', '960323053@qq.com', '411014196712130323', '15609032348', 'Mop5UPkl'),

(2, '吴婉瑜',  '1613230826@gmail.com', '420152196802131323',   '17605132307',  'QUTPhxgVNlXtMxN'),

(3, '蔡贞仪',  '252323341@foxmail.com',    '160347199005222323',   '17763232321',  'Bwe3gyhEErJ7');

### 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。

插入不完整的数据记录需要在 values 前指定插入的列，代码如下：

INSERT INTO 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：向客户表 client 批量插入数据。

将 insert 语句中的 value 字段更换为 select 查询语句即可，代码如下：

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 exists 子查询即可，代码如下：

DELETE FROM client c WHERE NOT EXISTS(SELECT \* FROM bank\_card b WHERE c.c\_id = b.b\_c\_id);

### 2.5.5 冻结客户资产

本关任务：冻结客户的投资资产。

使用一条 update 语句结合 exists 子查询即可实现。代码如下：

UPDATE property p SET p.pro\_status = '冻结' WHERE EXISTS(SELECT \* FROM client WHERE c\_id = p.pro\_c\_id and c\_phone = '13686431238');

### 2.5.6 连接更新

本关任务：根据客户表的内容修改资产表的内容。

根据任务要求使用 update 语句更新即可，代码如下：

UPDATE property p SET p.pro\_id\_card = (SELECT c\_id\_card FROM client WHERE c\_id = p.pro\_c\_id);

## 2.6 视图

本节的两个关卡围绕视图的创建和使用进行简单的应用了解。

### 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。

根据任务要求编写 select 语句，并在前面加上 “create view ... as” 即可，代码如下：

create view

v\_insurance\_detail

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,property,insurance

where client.c\_id=property.pro\_c\_id and pro\_type =2 and insurance.i\_id = property.pro\_pif\_id;

### 2.6.2 基于视图的查询

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

将视图当做普通的表一样 select 查询即可，代码如下：

select c\_name,c\_id\_card,SUM(pro\_quantity\*i\_amount) 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

本节的 3 个关卡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。

利用 “with recursive ... as” 定义一个递归查询子句，定义初值和递归过程即可，注意特判第一项为 1 ，代码如下：

drop procedure if exists sp\_fibonacci;

delimiter $$

create procedure sp\_fibonacci(in m int)

begin

######## 请补充代码完成存储过程体 ########

declare n1 bigint default 0;

declare n2 bigint default 1;

declare n3 bigint default 1;

declare i int default 2;

if not exists (select \* from fibonacci where n = 0) then

insert into fibonacci values(0,n1);

else

update fibonacci set fibn = n1 where n = 0;

end if;

if m >= 2 then

if not exists (select \* from fibonacci where n = 1) then

insert into fibonacci values(1,n2);

else

update fibonacci set fibn = n2 where n = 1;

end if;

end if;

while i < m

do

set n3 = n1 + n2;

if not exists (select \* from fibonacci where n = i) then

insert into fibonacci values(i,n3);

else

update fibonacci set fibn = n3 where n = i;

end if;

set n1 = n2;

set n2 = n3;

set i = i + 1;

end while;

end $$

delimiter ;

### 2.7.2 使用游标的存储过程

该关卡任务已完成，实施情况本报告略过。

### 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。

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。代码如下：

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

start transaction;

set return\_code = 0;

if 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 >= amount)

and exists(select\* from bank\_card where b\_number = dest\_card\_id and b\_c\_id = receiver\_id)

then

update bank\_card set b\_balance = if(b\_type = '信用卡', b\_balance - amount, b\_balance + amount)

where b\_number = dest\_card\_id;

update bank\_card set b\_balance = b\_balance - amount where b\_number = source\_card\_id;

set return\_code = 1;

end if;

commit;

END$$

delimiter ;

## 2.8 触发器

本节涉及触发器的简单使用。

### 2.8.1 为投资表property实现业务约束规则-根据投资类别分别引用不同表的主码

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。

根据任务要求声明 “BEFORE INSERT ON property” 类型的触发器，然后根据任务要求判断并设置相应的报错信息即可，如果报错信息为空则说明没有错误。代码如下：

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 ;

## 2.9 用户自定义函数

本节主要涉及如何在语句中使用自定义函数

### 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。

根据任务要求，函数定义部分的代码如下：

delimiter $$

create function get\_deposit(client\_id int)

returns numeric(10,2)

begin

return (select sum(b\_balance) from bank\_card where b\_c\_id = client\_id and b\_type = '储蓄卡'

group by b\_c\_id);

end$$

delimiter ;

在select语句中需要的地方像普通函数一样直接调用自定义函数即可，代码如下：

select \* from(

select c\_id\_card, c\_name, get\_deposit(c\_id) as total\_deposit

from client) a

where total\_deposit >= 1000000

order by total\_deposit desc;

## 2.10 安全性控制

本节的 2 个关卡涉及数据库中的用户、角色和权限等内容。

### 2.10.1 用户和权限

本关任务：在金融应用场景数据库环境中，创建用户，并给用户授予指定的权限。

根据任务要求合理使用 create, grant, revoke 语句编写指令即可，代码如下：

#(1) 创建用户tom和jerry，初始密码均为'123456'；

create user jerry identified by '123456';

create user tom 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 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 bank\_card to jerry;

#(4) 收回用户Cindy查询银行卡信息的权限。

revoke select on 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。

根据任务要求使用 create, grant 语句编写语句即可，代码如下：

# (1) 创建角色client\_manager和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 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 bank\_card to client\_manager;

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

grant select, insert, update on 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相关内容，包括隔离级别的设置，事务的开启、提交和回滚等，还通过添加等待代码实现了读脏、不可重复读、幻读等出错场景。

### 2.11.1 并发控制与事务的隔离级别

本关任务：设置事务的隔离级别。

MySQL 的事务隔离级别从低到高分以下四级：

1. 读未提交（READ UNCOMITTED ）

2. 读已提交（READ COMMITTED ）

3. 可重复读（REPEATABLE READ ）

4. 可串行化（SERIALIZABLE ）

低隔离级别可以支持更高的并发处理，同时占用的系统资源更少，但可能产生数据不一致的情形也更多一些。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。

根据任务要求把事务隔离级别设置为read uncommitted，并以rollback语句作为事务结束的回滚。代码如下：

# 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;

/\* 结束 \*/

### 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。要发生 “读脏” 需要将隔离级别设置为 read uncommitted 。同时，确保事务1读航班余票发生在在事务 2 修改之后，事务 2 撤销发生在事务 1 读取之后，据此设置事务休眠时间，即可发生 “读脏”。

事务一代码如下：

-- 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;

事务二代码如下：

-- 事务2

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

## 请添加代码，使事务1在事务2撤销前读脏;

set @n = sleep(5);

rollback;

### 2.11.3 不可重复读

该关卡任务已完成，实施情况本报告略过。

### 2.11.4 幻读

该关卡任务已完成，实施情况本报告略过。

### 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。事务2尝试在事务1两次读之间出票一张，但由于事务1代码中第 6 行加了锁，事务2的更新代码无法在事务1的锁释放之前执行，因此实现了可重复读，即事务1的两次读取结果是一致的。

事务一代码如下：

-- 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' for update;

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;

事务二代码如下：

-- 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。

直接让事务1休眠到事务2执行完后再执行即可。

事务1代码如下：

-- 事务1:

use testdb1;

start transaction;

set @n = sleep(2);

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

set @n = sleep(1);

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

commit;

事务2代码如下：

-- 事务2:

use testdb1;

start transaction;

set @n = sleep(1);

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：备份数据库，然后再恢复它。

使用 mysqldump 指令将服务器上的数据库 residents 备份至文件 residents\_bak.sql 中：

mysqldump -h127.0.0.1 -uroot  --databases residents > 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：模拟介质故障的发生，以及如何利用备份和备份之后的日志恢复数据库。

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在mysql‐dump 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;

系统在故障发生的时间点会将备份后新开的日志文件保存为 log/binlog.000018 。为了保证两次发生的业务数据都不丢失，除了根据备份文件 train\_bak.sql 恢复数据库外，还需要利用 mysqlbinlog 指令从 log/binlog.000018 文件中恢复数据库：

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

mysqlbinlog --no-defaults log/binlog.000018 | mysql -uroot;

## 2.13 数据库设计与实现

本节的 3 个关卡涉及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。

### 2.13.1 从概念模型到MySQL实现

该关卡任务已完成，实施情况本报告略过。

### 2.13.2 从需求分析到逻辑模型

该关卡任务已完成，实施情况本报告略过。

### 2.13.3 建模工具的使用

该关卡任务已完成，实施情况本报告略过。

### 2.13.4 制约因素分析与设计

在从实际问题的建模到数据库的概念模型和逻辑模型的构建过程中，需要考虑若干制约因素。以机票订票系统为例，系统需要考虑到旅客的实际情况，旅客可以多次乘坐飞机，一张机票肯定是某个用户为某个特定的旅客购买的特定航班的机票，所以机票信息不仅跟乘坐人有关，同时还需要记录购买人信息 (虽然两者有时是同一人) 。此外，对于系统的权限也存在若干要求，例如实体 “用户” 就刻印根据权限分成两类，用户分两类：普通用户可以订票，管理用户有权限维护和管理整个系统的运营。

### 2.13.5 工程师责任及其分析

社会方面，工程师应该能够基于工程相关背景知识进行合理分析，评价专业工程实践和复杂工程问题解决方案对社会、健康、安全、法律以及文化的影响，并理解应承担的责任。安全方面，工程师应该尽可能考虑系统中存在的安全漏洞，安全性是所有系统用户关心的重要命题；科学发展方面，工程师应该能够基于科学原理并采用科学方法对复杂工程问题进行研究，包括设计实验、分析与解释数据、并通过信息综合得出合理有效的结论。

## 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 表中邮箱非空的客户信息，列出客户姓名，邮箱和电话。

使用 Java 的 Class.forName() 方法，将驱动程序的类文件动态加载到内存中，并将其自动注册。加载驱动程序后，使用 DriverManager.getConnection(String url, String user,String password) 方法建立连接。在使用 Statement 对象执行 SQL 语句之前，需要使用 Connection 对象的 createState‐ment() 方法创建 Statement 的一个实例。创建 Statement 对象后，可以使用它来执行一个 SQL 语句，其中有三个执行方法：

1. boolean execute(String SQL) ：如果可以检索到 ResultSet 象，则返回一个true否则返回 false 。使用此方法执行 SQL DDL 语句或需要使用真正的动态 SQL 时。

2. int executeUpdate(String SQL) ：返回受 SQL 语句执行影响的行数。使用此方法执行预期会影响多个行的 SQL 语句，例如 INSERT, UPDATE 或 DELETE 语句。

3. ResultSet executeQuery(String SQL) ：返回一个 ResultSet 对象。当希望获得结果集时，使用此方法，就像使用 SELECT 语句一样。

本关需要选用第 3 个执行方法，使用 resultSet.next() 遍历 ResultSet 输出相应的信息即可，代码如下：

import java.sql.\*;

public class Client {

    public static void main(String[] args) {

        String URL = "jdbc:mysql://localhost:3306/finance?useUnicode=true&characterEncoding=UTF8&useSSL=false&serverTimezone=UTC";

        String USER = "root";

        String PASS = "123123";

        Connection connection = null;

        Statement statement = null;

        ResultSet resultSet = null;

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

            connection = DriverManager.getConnection(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.print("姓名\t邮箱\t\t\t\t电话\n");

            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。

用 PreparedStatement 类执行 SQL 语句，把 SQL 语句中变化的部分当成参数，以防御 SQL注入攻击。代码如下：

import java.sql.\*;

import java.util.Scanner;

public class Login {

    public static void main(String[] args) {

        Connection connection = null;

        //申明下文中的resultSet, statement

        ResultSet resultSet = null;

        Statement statement = null;

        Scanner input = new Scanner(System.in);

        System.out.print("请输入用户名：");

        String loginName = input.nextLine();

        System.out.print("请输入密码：");

        String loginPass = input.nextLine();

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

            String userName = "root";

            String passWord = "123123";

            String url = "jdbc:mysql://127.0.0.1:3306/finance?useUnicode=true&characterEncoding=UTF8&useSSL=false&serverTimezone=UTC";

            connection = DriverManager.getConnection(url, userName, passWord);

            // 补充实现代码:

            statement = connection.createStatement();

            String sql = "select \* from client where c\_mail = '"+loginName+"' and c\_password = '"+loginPass+"';";

            resultSet = statement.executeQuery(sql);

            if(resultSet.next())

            {

                System.out.println("登录成功。");

            }

            else

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

         } catch (ClassNotFoundException e) {

            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();

            }

        }

    }

}

### 2.14.3 添加新客户

本关任务：编程完成向客户表 client 插入记录的方法。

编写 SQL 语句，用 PreparedStatement 类执行即可，代码如下：

public static int insertClient(Connection connection,

                                   int c\_id, String c\_name, String c\_mail,

                                   String c\_id\_card, String c\_phone,

                                   String c\_password){

        String sql = "insert into client values(?,?,?,?,?,?)";

        try{

            PreparedStatement pps = connection.prepareStatement(sql);

            pps.setInt(1,c\_id);

        pps.setString(2,c\_name);

        pps.setString(3,c\_mail);

        pps.setString(4,c\_id\_card);

        pps.setString(5,c\_phone);

        pps.setString(6,c\_password);

        return pps.executeUpdate();

        }catch(SQLException e){

            e.printStackTrace();

        }

        return 0;

    }

### 2.14.4 银行卡销户

该关卡任务已完成，实施情况本报告略过。

### 2.14.5 客户修改密码

本关任务：编写修改客户登录密码的的方法。

先判断用户是否存在，再判断密码是否正确，最后利用 update 语句修改密码。代码如下：

 public static int passwd(Connection connection,

                             String mail,

                             String password,

                             String newPass){

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

        try{

            PreparedStatement pps = connection.prepareStatement(sql);

            pps.setString(1,mail);

            ResultSet resultSet = pps.executeQuery();

            if(resultSet.next())

            {

                if(password.equals(resultSet.getString("c\_password")))

                {

                    sql = "update client set c\_password = ? where c\_mail = ? and c\_password = ?";

                    pps = connection.prepareStatement(sql);

                    pps.setString(1,newPass);

                    pps.setString(2,mail);

                    pps.setString(3,password);

                    pps.executeUpdate();

                    return 1;

                }

                else

                    return 3;

            }

            else

                return 2;

        }catch(SQLException e){

            e.printStackTrace();

        }

        return -1;

    }

### 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。

根据题意编写判断条件，如果不合法则直接return false ，然后判断收款方卡类型是否为信用卡决定是否要把收款金额设置为负数。代码如下：

public class Transfer {

    static final String JDBC\_DRIVER = "com.mysql.cj.jdbc.Driver";

    static final String DB\_URL = "jdbc:mysql://127.0.0.1:3306/finance?allowPublicKeyRetrieval=true&useUnicode=true&characterEncoding=UTF8&useSSL=false&serverTimezone=UTC";

    static final String USER = "root";

    static final String PASS = "123123";

    /\*\*

     \* 转账操作

     \*

     \* @param connection 数据库连接对象

     \* @param sourceCard 转出账号

     \* @param destCard 转入账号

     \* @param amount  转账金额

     \* @return boolean

     \*   true  - 转账成功

     \*   false - 转账失败

     \*/

    public static boolean transferBalance(Connection connection,

                             String sourceCard,

                             String destCard,

                             double amount){

        try{

            String sql = "select \* from bank\_card where b\_number = ?";

            PreparedStatement pps = connection.prepareStatement(sql);

            pps.setString(1,sourceCard);

            ResultSet resultSet = pps.executeQuery();

            if(!resultSet.next() || resultSet.getString("b\_type").equals("信用卡") || resultSet.getDouble("b\_balance") < amount)

                return false;

            else

            {

                pps = connection.prepareStatement(sql);

                pps.setString(1,destCard);

                resultSet = pps.executeQuery();

                if(!resultSet.next())

                    return false;

                else{

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

                    sql = "update bank\_card set b\_balance = b\_balance + ? where b\_number = ?";

                    pps = connection.prepareStatement(sql);

                    pps.setDouble(1,-amount);

                    pps.setString(2,sourceCard);

                    pps.executeUpdate();

                    pps.setDouble(1, rcv\_balance);

                    pps.setString(2, destCard);

                    pps.executeUpdate();

                    return true;

                }

            }

        }catch(SQLException e){

            e.printStackTrace();

        }

        return false;

    }

### 2.14.7 把稀疏表格转换为键值对存储

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。

选出 entrance\_exam 表中所有数据，对每个学生，并枚举所有学科，如果该学生存在该学科的成绩，就将其插入 sc 表中。代码如下：

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";

    /\*\*

     \* 向sc表中插入数据

     \*

     \*/

    public static void insertSC(Connection connection, int sno, String col\_name, int col\_value){

        try{

            String sql = "insert into sc value(?,?,?)";

        PreparedStatement pps = connection.prepareStatement(sql);

        pps.setInt(1,sno);

        pps.setString(2,col\_name);

        pps.setInt(3,col\_value);

        pps.executeUpdate();

        }catch(SQLException e){

            e.printStackTrace();

        }

    }

    public static void main(String[] args) throws Exception{

        try {

            String[] subject = {"chinese", "math", "english", "physics", "chemistry", "biology", "history", "geography", "politics"};

            Class.forName(JDBC\_DRIVER);

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

            ResultSet res = con.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(con, sno, sub, score);

                }

            }

        } catch (SQLException e) {

            e.printStackTrace();

        }

    }

}

# 3 课程总结

本次实验从数据库和表的定义出发，分别完成了表的完整性约束的创建和修改、数据查询、数据的插入、修改与删除、视图的创建与使用、存储过程与事务、触发器、用户自定义函数、安全性控制、并发控制与事务的隔离级别、数据库的备份与日志、数据库的设计与实现和数据库应用开发，数据库的索引B+树实现等15个实训实验，总计73个关卡。本人完成了除B+树索引实现和部分数据查询关卡以外的全部关卡，在任务书规定范围内拿到满分。

本实验最初部分的关卡主要是数据库的定义和一些简单的增删改查部分的操作，属于MySQL基础应用部分，数据查询中因其在实际应用中的重要性，增添了许多关卡，不仅有基础查询，还有需要复杂语句和数据库连接的难度较大的关卡。这部分实验主要帮助我们熟悉了数据库最常用的部分的操作，以及能熟练使用MySQL进行一些复杂的查询，感受MySQL的强大功能。随后关于视图，存储事务，触发器，用户自定义函数，安全性控制，并发，备份等关卡都较少，但是能够利用实际操作帮我们了解这些概念的简单应用。在JDBC数据库开发中则从数据库的操作深入到了数据库的实现，在编写的过程中深切体会到了开发过程中对安全性的重视，要考虑到各种可能出现的异常情况，保证数据库不会因为SQL注入攻击等崩溃或者导致其他灾难性后果。

本次实验内容充实完善，引导性强，完成实验的过程虽然不易，但是完成后的收获很大，相比于课上较于晦涩难懂抽象的理论知识，MySQL语言的代码操作极大提升了对数据库的操作的了解，Java开发数据库应用则增进了对数据库底层实现的了解，B+树由于时间和难度问题暂时没有机会完成，但是后续希望有时间能够通过这个实验对数据库的索引有着更深入的了解。

最后，衷心感谢课程组老师们精心设计的实验和实验过程中的悉心指导！