

**数据库系统原理实践报告**

|  |  |
| --- | --- |
| 专 业： | 计算机科学与技术 |
| 班 级： | 1904 |
| 学 号： | U201915050 |
| 姓 名： | 高博文 |
| 指导教师： | 袁平鹏 |

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

2020 年 07 月 01 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](#_Toc107790268)

[2 任务实施过程与分析 2](#_Toc107790269)

[2.1 数据库、表与完整性约束的定义(Create) 2](#_Toc107790270)

[2.2 表结构与完整性约束的修改(ALTER) 3](#_Toc107790271)

[2.3 数据查询(Select) 4](#_Toc107790272)

[2.4 数据的插入、修改与删除(Insert,Update,Delete) 11](#_Toc107790273)

[2.5 视图 12](#_Toc107790274)

[2.6 存储过程与事务 13](#_Toc107790275)

[2.7 触发器 18](#_Toc107790276)

[2.8 用户自定义函数 19](#_Toc107790277)

[2.9 安全性控制 19](#_Toc107790278)

[2.10 并发控制与事务的隔离级别 20](#_Toc107790279)

[2.11 备份+日志：介质故障与数据库恢复 22](#_Toc107790280)

[2.12 数据库设计与实现 22](#_Toc107790281)

[2.13数据库应用开发(JAVA篇) 22](#_Toc107790282)

[3 课程总结 28](#_Toc107790283)

[3.1 实验总结 28](#_Toc107790284)

[附录 29](#_Toc107790285)

# 1 课程任务概述

### 1.1 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，需要我们熟悉并掌握SQL语法，明确数据库的设计方法，掌握数据库安全性控制、恢复、开发等一系列相关操作的基本知识。

### 1.2 实训任务内容

1. 数据库、表与完整性约束的定义
2. 表结构与完整性约束的修改
3. 数据查询
4. 数据的插入、修改与删除
5. 视图操作
6. 存储过程与事务
7. 触发器
8. 用户自定义函数
9. 安全性控制
10. 并发控制与事务的隔离级别
11. 用备份日志进行数据库的恢复
12. 数据库设计与实现
13. 数据库应用开发（JAVA篇）

# 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)

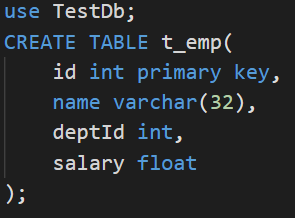
主要是用相关语句对数据库进行基础的创建、增加约束等操作。

### 2.1.1 创建数据库



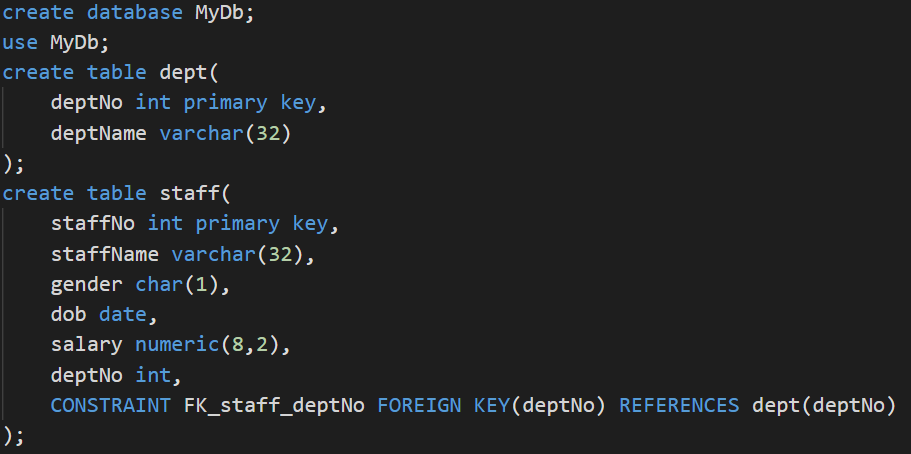
直接创建语句CREATE DATABASE。

### 2.1.2 创建表及表的主码约束



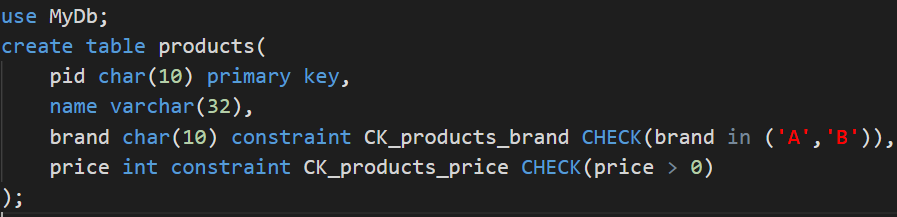
用建表语句一条条加入数据和对应类型。

### 2.1.3 创建外码约束



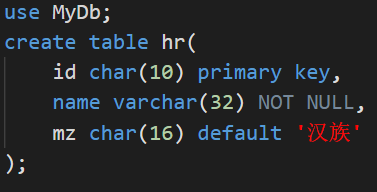
用CONSTRAINT语句设定外码。

### 2.1.4 CHECK约束



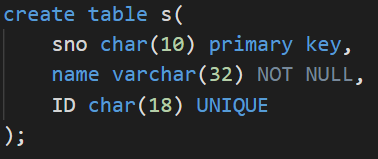
CHECK后面加条件。

### 2.1.5 DEFAULT约束



DEFAULT默认初始为汉族。

### 2.1.6 UNIQUE约束



设置一个不能重复的ID。

## 2.2 表结构与完整性约束的修改(ALTER)

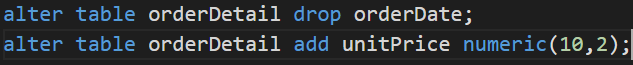
主要是用Alter语句对表进行一些定义的修改。

### 2.2.1 修改表名



直接alter用rename。

### 2.2.2添加与删除字段



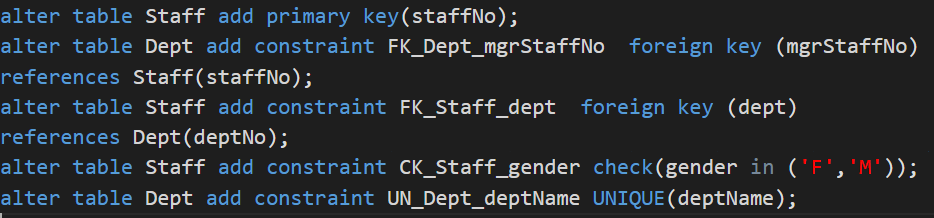
Drop删除字段，add添加字段。

### 2.2.3修改字段



Rename修改列名。

### 2.2.4添加、删除与修改约束



修改表，添加主码，外码、check、unique用限制方式添加。

## 2.3 数据查询(Select)

主要是数据的查询。每个查询任务都需要用一条SQL语句完成某个查询。但是要注意数据集是事先创建并已初始化。

### 2.3.1 金融应用场景介绍，查询客户主要信息



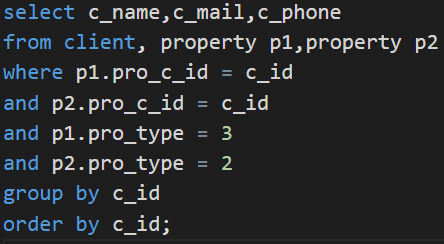
直接select对应信息。

### 2.3.2 邮箱为null的客户



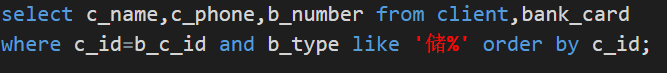
Where条件c\_mail为空。

### 2.3.3 既买了保险又买了基金的客户



同一个表不同命名查询，条件是既有2类型也有3类型（代表保险和基金）。

### 2.3.4 办理了储蓄卡的客户



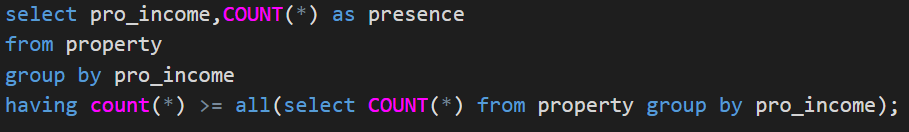
条件用like选择以储开头的类型，只有储蓄卡这一种。

### 2.3.5 每份金额在30000-50000之间的理财产品



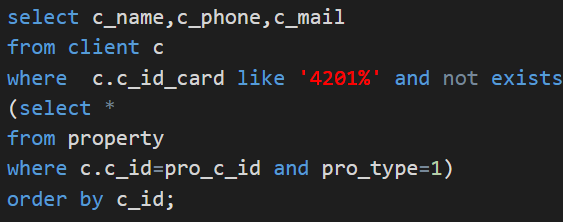
设置p\_amount在规定范围，先按金额升序，再按理财年限降序。

### 2.3.6 商品收益的众数



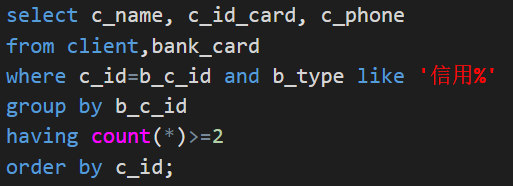
嵌套查询，内层查询是商品收益的计数，选择出来之后，外层查询用同样的计数，条件是计数大于等于内层查询出的所有数量，所以这样筛选出来就是众数。

### 2.3.7 未购买任何理财产品的武汉居民



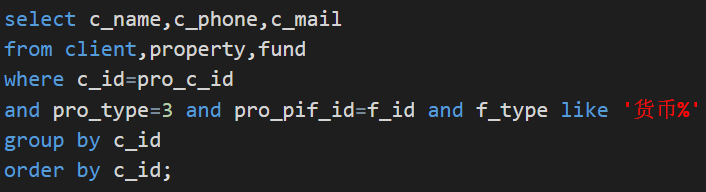
使用不存在语句，内层查询负责寻找购买了理财产品的居民，外层查询的条件就是NOT EXISTS不存在这样的居民。

### 2.3.8 持有两张信用卡的用户



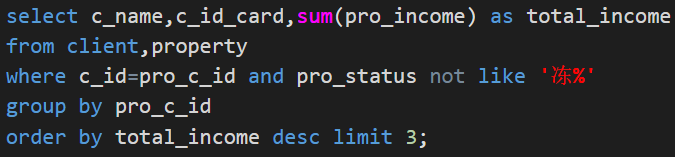
先将所有的用户的信用卡的数量计算出，注意用group by限制计算的是每个人的信用卡数，然后用having语句限制结果筛选出有两张以上信用卡的用户，最后结果用id升序排序。

### 2.3.9 购买了货币型基金的客户信息



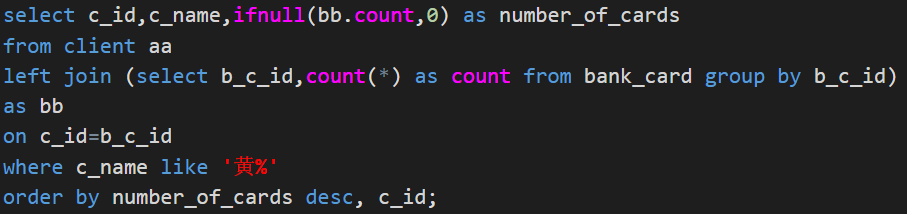
查询三个表，限制购买的理财产品为3类型（基金），再限制基金类型为货币型，用Like限制名称，最后排序。

### 2.3.10 投资总收益前三名的客户



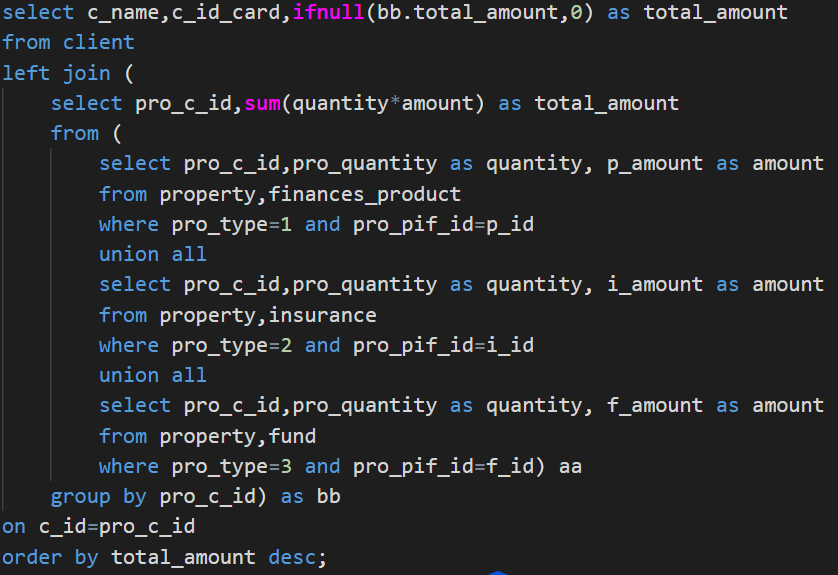
首先查询个人信息，用sum函数计算总收益，注意用group by限制sum函数的作用范围，计算每个人的总收益，最后在降序排序的同时，用Limit限制筛选前三个，也就是总收益的前三名。

### 2.3.11 黄姓客户持卡数量



用连接查询，首先查询id和持卡数量，限制条件是like筛选出黄姓的客户，然后将查询的结果做左外连接，这是根据id相同直接找出名字。注意如果计算持卡数量时，如果数量是0，连接后显示的是NULL，需要用Ifnull函数来判断，控制显示出数字0。

### 2.3.12 客户理财、保险与基金投资总额



因为投资有三种，理财、保险和基金，而每笔投资金额=商品数量\*该产品每份金额，因为在最内层查询首先用三次独立的查询语句查询三种投资的数量，用union将查询结果连接，然后根据id分组，保证查询到客户的投资情况，在外层用sum函数计算商品数量\*每份金额的结果，将这个结果用左外连接到客户表中，形成一个客户的姓名和投资总额都有的表，注意最后结果也是一样需要用ifnull函数判断一下是不是投资总额为0。

### 2.3.13 客户总资产

select c\_id,c\_name,ifnull(t3.total\_amount,0) as total\_property

from client

left join (

    select pro\_c\_id,sum(total\_amount1) as total\_amount

    from(

        select pro\_c\_id,sum(quantity\*amount) as total\_amount1

        from (

            select pro\_c\_id,pro\_quantity as quantity, p\_amount as amount

            from property,finances\_product

            where pro\_type=1 and pro\_pif\_id=p\_id

            union all

            select pro\_c\_id,pro\_quantity as quantity, i\_amount as amount

            from property,insurance

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

            union all

            select pro\_c\_id,pro\_quantity as quantity, f\_amount as amount

            from property,fund

            where pro\_type=3 and pro\_pif\_id=f\_id)t1

        group by pro\_c\_id

        union all

        select pro\_c\_id,sum(pro\_income) as total\_amount1

        from property

        group by pro\_c\_id

        union all

        select b\_c\_id as pro\_c\_id,sum(b\_balance) as total\_amount1

        from bank\_card

        where b\_type like '储%'

        group by b\_c\_id

        union all

        select b\_c\_id as pro\_c\_id,sum(0-b\_balance) as total\_amount1

        from bank\_card

        where b\_type like '信%'

        group by b\_c\_id

        ) as t2

    group by pro\_c\_id

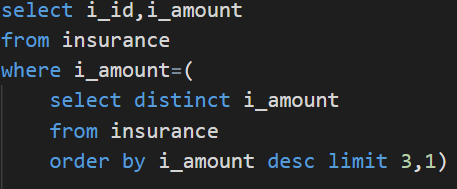
    ) as t3

on c\_id=pro\_c\_id

order by c\_id;

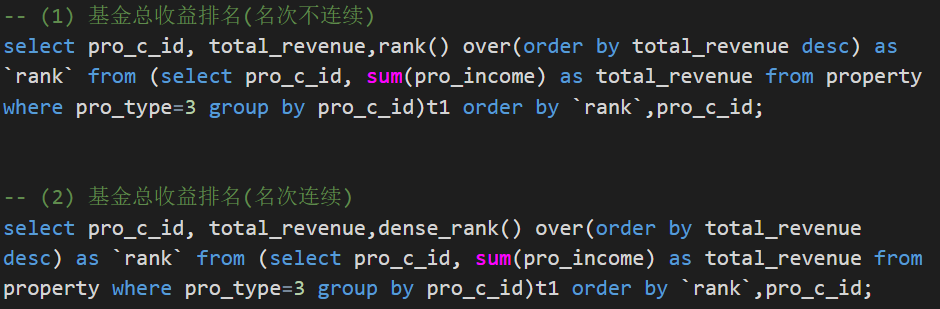
首先需要查询的信息里有客户的投资总金额，因此可以直接使用之前12关的查询语句，先将几个投资类型查询出来，作为一个表t1，然后查询出客户的收入，储蓄卡的余额，因为需要扣除信用卡的余额，因此需要用sum函数将信用卡余额用0减让它变成负的，这个结果作为t2。然后根据客户id，所有查询结果求和，并且用union合并之后，结果作为t3,建立一个左外连接，同样还是连接到client客户表中，这样输出一个客户的编号、名称和总资产。

### 2.3.14 第N高问题



筛选时注意，用distinct消除重复，因为排序同样高的也会占用名次，用一个降序排序，排出前三个，这样就能选出第四高的保险金额。这里的嵌套查询，外层查询负责查询出id和金额，金额等于内存查询中查出的第四高的金额。

### 2.3.15 基金收益两种方式排名



主要是用rank函数来排名，名次连续和不连续的区别主要是名次连续用dense\_rank，查询时筛选投资类型为3的基金，用id分组。

### 2.3.16 持有完全相同基金组合的客户

select t1.pro\_c\_id as c\_id1, t3.pro\_c\_id as c\_id2

        from(

            select t2.pro\_c\_id, group\_concat(distinct t2.pro\_pif\_id order by t2.pro\_pif\_id) as pro\_id

            from(

                select property.pro\_c\_id, property.pro\_pif\_id

                from property

                where property.pro\_type=3

            ) as t2

            group by property.pro\_c\_id

        ) as t1

        inner join(

            select t4.pro\_c\_id, group\_concat(distinct t4.pro\_pif\_id order by t4.pro\_pif\_id) as pro\_id

            from(

                select property.pro\_c\_id, property.pro\_pif\_id

                from property

                where property.pro\_type=3

            ) as t4

            group by property.pro\_c\_id

        ) as t3

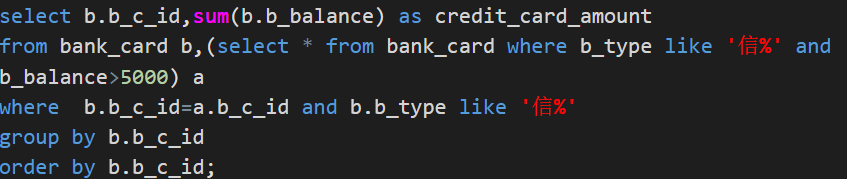
        on (t1.pro\_id=t3.pro\_id)and(t1.pro\_c\_id<t3.pro\_c\_id);

总的有三层嵌套查询，最内层负责查询投资类型为3基金，到外层从这些查询结果中查询同时分组作为c\_id1和c\_id2，这是两种类型的组合，通过一个内连接，连接到同样类型的查询，最后的筛选条件是t1的id和t3相等，同时t1的c\_id小于t3的c\_id。

### 2.3.17 购买基金的高峰期

本关跳过。

### 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



内层查询查找客户的信用卡类型的余额大于5000的所有客户的信息，这些信息作为表a，外层用sum函数计算总余额，用id分组和排序，作为最后的结果打印出来。

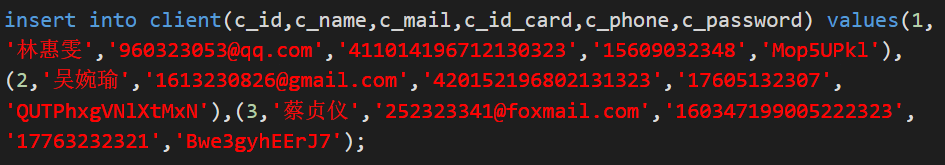
### 2.3.19 以日历表格式显示每日基金购买总金额

本关跳过。

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

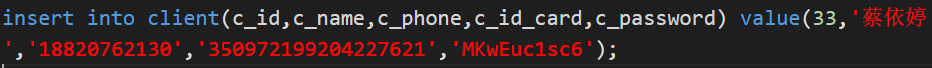
主要是使用insert、update等语句对数据库进行一些基本操作。

### 2.4.1 插入多条完整的客户信息



用insert语句插入，注意对应各表名称。

### 2.4.2 插入不完整的客户信息



用insert语句插入，没有的部分直接为空。

### 2.4.3 批量插入数据



将new\_client表的内容插入到client表。

### 2.4.4删除没有银行卡的客户信息



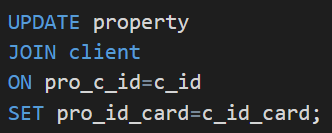
直接用delete语句删除。

### 2.4.5 冻结客户资产



用update语句设置状态为冻结。

### 2.4.6 连接更新

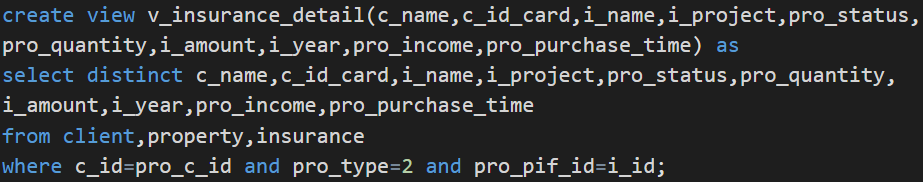


用update语句更新join连接。

## 2.5 视图

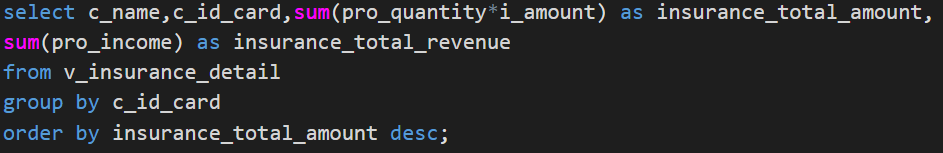
视图的一些简单操作，包括创建、查询等。

### 2.5.1 创建所有保险资产的详细记录试图



Create view创建视图，从client表中获取数据，注意数据用distinct设置不能重复。

### 2.5.2 基于视图的查询



视图的查询基本和表查询类似，用sum函数计算一下group中的保险投资总额和保险投资总收益。

## 2.6 存储过程与事务

掌握存储流程语句，能够实现一些过程模拟。

### 2.6.1 使用流程控制语句的存储过程

drop procedure if exists sp\_fibonacci;

delimiter $$

create procedure sp\_fibonacci(in m int)

begin

    declare n int default 2;

    declare a int default 0;

    declare b int default 1;

    declare c int default 0;

    truncate table fibonacci;

    insert into fibonacci(n,fibn) value(0,0);

    if m>1

        then

            insert into fibonacci(n,fibn) value(1,1);

    end if;

    while n<m do

            set c=a+b;

            insert into fibonacci(n,fibn) value(n,c);

            set a=b;

            set b=c;

            set n=n+1;

    end while;

end $$

delimiter ;

首先设置初始值，主要数列的构造是一个循环，每次都用前两个值相加，用insert语句插入，value中就是对应的值。

### 2.6.2 使用游标的存储过程

delimiter $$

create procedure sp\_night\_shift\_arrange(in start\_date date, in end\_date date)

begin

    declare senior int default(0);

    declare doctor char(8);

    declare nurse1,nurse2 char(8);

    declare doctor\_cursor cursor for select e\_type from employee where e\_type=1 or e\_type=2;

    declare sen\_cursor cursor for select e\_name from employee where e\_type=1;

    declare doc\_cursor cursor for select e\_name from employee where e\_type=2;

    declare nur\_cursor cursor for select e\_name from employee where e\_type=3;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    open doctor\_cursor;

    open sen\_cursor;

    open doc\_cursor;

    open nur\_cursor;

    while start\_date<=end\_date do

        set start\_date=date\_format(start\_date,'%Y-%m-%d');

        set d=weekday(start\_date);

        if (d<5 and senior>0) then

            fetch sen\_cursor into doctor;

            if done then

                set done=false;

                close sen\_cursor;

                open sen\_cursor;

                fetch sen\_cursor into doctor;

            end if;

            set senior=senior-1;

        else

            fetch doctor\_cursor into t;

            if done then

                set done=false;

                close doctor\_cursor;

                open doctor\_cursor;

                fetch doctor\_cursor into t;

            end if;

            if(d>4) then

                while t=1 do

                    fetch doctor\_cursor into t;

                    if done then

                        set done=false;

                        close doctor\_cursor;

                        open doctor\_cursor;

                        fetch doctor\_cursor into t;

                    end if;

                    set senior=senior+1;

                end while;

                fetch doc\_cursor into doctor;

                if done then

                    set done=false;

                    close doc\_cursor;

                    open doc\_cursor;

                    fetch doc\_cursor into doctor;

                end if;

            else

                if t=1 then

                    fetch sen\_cursor into doctor;

                    if done then

                        set done=false;

                        close sen\_cursor;

                        open sen\_cursor;

                        fetch sen\_cursor into doctor;

                    end if;

                else

                    fetch doc\_cursor into doctor;

                    if done then

                        set done=false;

                        close doc\_cursor;

                        open doc\_cursor;

                        fetch doc\_cursor into doctor;

                    end if;

                end if;

            end if;

        end if;

        fetch nur\_cursor into nurse1;

        if done then

            set done=false;

            close nur\_cursor;

            open nur\_cursor;

            fetch nur\_cursor into nurse1;

        end if;

        fetch nur\_cursor into nurse2;

        if done then

            set done=false;

            close nur\_cursor;

            open nur\_cursor;

            fetch nur\_cursor into nurse2;

        end if;

        insert into night\_shift\_schedule(n\_date,n\_doctor\_name,n\_nurse1\_name,n\_nurse2\_name) value(start\_date,doctor,nurse1,nurse2);

        set start\_date=date\_add(start\_date,interval 1 day);

    end while;

end$$

delimiter ;

首先创建两个游标：doc\_cursor和nur\_cursor，doc可以筛选查询出所有的医生，而nurse就是所有的护士，注意一天需要找出一个医生和两个护士。因为需要处理NOTFOUND异常，因此加一个状态变量，1号状态表示获取第一个护士，第二个护士就是状态2，因此根据这个状态值可以知道在handler重启操作之前的异常产生者。在周六或者周日时，如果获取到一个主任，因为只有一个主任，有主任标记变量判断主任队列是不是已经有主任了，空的就可以存储主任信息，周一如果已经有主任，需要取出主任，不用doc获取医生。

### 2.6.3 使用事务的存储流程

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 id1 int;

    declare id2 int;

    declare flag int;

    declare balance int;

    declare type1 char(20);

    declare type2 char(20);

    set flag =0;

    select b\_c\_id into id1 from bank\_card where (b\_number=source\_card\_id);

    if(id1=applicant\_id) then

        set flag=flag+1;

    end if;

    select b\_c\_id into id2 from bank\_card where (b\_number= dest\_card\_id);

    if(id2=receiver\_id) then

        set flag=flag+1;

    end if;

    select b\_type into type1 from bank\_card where (b\_number=source\_card\_id);

    if(type1='储蓄卡') then

        set flag=flag+1;

    end if;

    select b\_balance into balance from bank\_card where (b\_number=source\_card\_id);

    if(balance-amount>=0) then

        set flag=flag+1;

    end if;

    set return\_code =0;

    if flag=4 then

        set return\_code = 1;

    end if;

    select b\_type into type2 from bank\_card where (b\_number=dest\_card\_id);

    if return\_code =1 then

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

        if(type2='储蓄卡')then

            update bank\_card set b\_balance=b\_balance + amount where b\_number=dest\_card\_id;

        else

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

        end if;

    end if;

END$$

转账实际上就是对应的账户的金额的加减过程，用查询语句查询出卡的持有人为转款人和收款人时，可以用修改语句update将转出的卡的余额减去amout，将收款的卡的余额加上amount，并且注意当储蓄卡向信用卡转账时，此时是还款状态，仍然要减去amount。设置一个flag，如果转账金额超过了储蓄卡余额，或者是信用卡向储蓄卡转账，或者是转款人和收款人不是分别卡的持有人，那么flag会保持初始值为0，这时候说明转账不成功了。如果条件都满足，那flag会设置为4，这时候设置返回值为1，表示正常转账。

## 2.7 触发器

简单地利用create语句等创建触发器。

### 2.7.1 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

CREATE TRIGGER before\_property\_inserted BEFORE INSERT ON property

FOR EACH ROW

BEGIN

    declare msg varchar(128);

    if 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;

    elseif new.pro\_type=1 and not exists(select\* from finances\_product where new.pro\_pif\_id=p\_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 new.pro\_pif\_id=i\_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 new.pro\_pif\_id=f\_id) then

        set msg = concat('fund #',new.pro\_pif\_id,' not found!');

        SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = msg;

    end if;

END$$

创建触发器，然后设置相应地触发语句，例如如果投资的类型不对，会弹出窗口显示类型不合法，或者表中没有对应的信息，会显示not found之类。

## 2.8 用户自定义函数

创建一个函数并且使用。

### 2.8.1 创建函数并在语句中使用它

create function get\_deposit(client\_id int)

returns numeric(10,2)

begin

    declare total numeric(10,2);

    select sum(b\_balance) into total from bank\_card where b\_c\_id=client\_id and b\_type like '储%' group by b\_c\_id;

    return total;

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

from client,bank\_card

where c\_id=b\_c\_id and get\_deposit(c\_id)>=1000000

order by total\_deposit desc;

函数的内容就是利用sum计算总金额，函数名get\_deposit，查询时调用。

## 2.9 安全性控制

主要是考察权限相关的语句。

### 2.9.1 用户和权限

create user tom,jerry identified by '123456';

grant select(c\_name,c\_mail,c\_phone) on client to tom with grant option;

grant update(b\_balance) on bank\_card to jerry;

revoke select on bank\_card from Cindy;

授予权限就用grant语句，收回权限就用revoke语句。

### 2.9.2 用户、角色和权限

create role client\_manager,fund\_manager;

grant select,insert,update on client to client\_manager;

grant select(b\_number,b\_type,b\_c\_id) on bank\_card to client\_manager;

grant select,insert,update on fund to fund\_manager;

grant client\_manager to tom,jerry;

grant fund\_manager to Cindy;

增加了一个角色，实际就是先给角色授权，角色给对象授权，用相应语句。

## 2.10 并发控制与事务的隔离级别

多个事务并发访问数据，如果不加以控制，极易导致数据的不一致性问题。主要考察加锁操作等，使事务避免出现读脏、不可重复读等问题。

### 2.10.1 并发控制与事务的隔离级别

set session transaction isolation level read uncommitted;

start transaction;

insert into dept(name) values('运维部');

rollback;

设置事务隔离级别为read uncommitted，开启后插入一个名字，直接回滚。

### 2.10.2 读脏

set session transaction isolation level read uncommitted;

start transaction;

set @n = sleep(2);

select tickets from ticket where flight\_no = 'CA8213';

commit;

设置事务隔离级别，之后设置事务的等待时间，修改完后读取脏数据。

### 2.10.3 不可重复读

insert into result

select now(),1 t, tickets from ticket where flight\_no = 'CZ5525';

set @n = sleep(2);

update ticket set tickets = tickets - 1 where flight\_no = 'CZ5525';

insert into result

select now(),1 t, tickets from ticket where flight\_no = 'CZ5525';

set @n = sleep(4);

commit;

set @n = sleep(1);

insert into result

select now(), 1 t, tickets from ticket where flight\_no = 'CZ5525';

等待代码保证事务2的读取在事务1修改之前发生，并且设置一个更长的等待时间，事务2的读取在事务1修改之后，提交之前，最后设置等待时间，让事务1在事务2提交后读取。

### 2.10.4 幻读

use testdb1;

select @@transaction\_isolation;

start transaction;

select \* from ticket where tickets > 300;

set @n = sleep(2);

update ticket set aircraft = 'A330-300' where flight\_no = 'MU5111';

select \* from ticket where tickets > 300;

commit;

首先查询超过300张的票的航班信息，然后修改航班的机型，最后再次查询剩余票超过300张的航班的信息。

### 2.10.5 主动加锁保证可重复读

set session transaction isolation level read uncommitted;

start transaction;

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

set @n = sleep(5);

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

commit;

set @n = sleep(1);

select \* from ticket;

查询航班的余票，并设置等待时间，再次查询，并且提交，第三次查询在事务2提交之后。

### 2.10.6 可串行化

use testdb1;

set session transaction isolation level read uncommitted;

start transaction;

set @n = sleep(2);

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

set @n = sleep(2);

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

commit;

设置等待时间后，查询航班信息，同样的时间再次查询。

## 2.11 备份+日志：介质故障与数据库恢复

利用备份和日志实现恢复，做基本训练。

### 2.11.1备份与恢复

mysqldump -h 127.0.0.1 -u root --databases residents >residents\_bak.sql

mysql -h 127.0.0.1 -u root < residents\_bak.sql

对数据库备份至文件中，再利用这个文件还原。

### 2.11.2备份+日志：介质故障的发生与数据库的恢复

mysqldump -h 127.0.0.1 -u root --databases train > train\_bak.sql

mysqladmin -u root flush-logs

对数据库train作逻辑备份并新开日志文件。

mysql -h 127.0.0.1 -u root < train\_bak.sql

mysqlbinlog --no-defaults log/binlog.000018 | mysql -h 127.0.0.1 -u root

利用逻辑备份和日志恢复数据库。

## 2.12 数据库设计与实现

本关卡跳过。

## 2.13数据库应用开发(JAVA篇)

利用高级语言开发数据库应用系统，通过几个简单的练习掌握JAVA开发数据库应用的基本知识。

### 2.13.1 JDBC体系结构和简单的查询

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

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

String USER = "root";

String PASS = "123123";

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.println("姓名"+"\t"+"邮箱"+"\t"+"\t"+"\t"+"\t"+"电话");

while (resultSet.next()) {

System.out.println(resultSet.getString("c\_name")+"\t"+resultSet.getString("c\_mail")+"\t"+"\t"+resultSet.getString("c\_phone"));

}

首先确定一个连接地址，设置好用户名和密码，用连接和会话函数创建并生成会话，结果为一个查询，找到数据库中的对应属性的值。

### 2.13.2用户登录

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

            statement = connection.createStatement();

            resultSet = statement.executeQuery(sql);

            if (!resultSet.next()) {

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

            }

            else{

                if(!loginPass.equals(resultSet.getString("c\_password"))){

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

                }

                else{

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

                }

            }

如果返回结果失败，说明用户名或者密码错误，会输出。如果登录的密码不正确，也会进行判断，输出用户名或密码错误。

### 2.13.3 添加新客户

    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)

    {

        PreparedStatement pps = null;

        try{

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

            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);

            n = pps.executeUpdate();

        }catch (SQLException throwables) {

            throwables.printStackTrace();

        } finally {

            try {

                if (pps != null) {

                    pps.close();

                }

            } catch (SQLException throwables) {

                throwables.printStackTrace();

            }

        }

        return n;

    }

用setstring函数将客户信息打包赋值，实例化到对象中，因为该对象已经关联了一条insert语句，最后用execute执行修改即可。

### 2.13.4 银行卡销户

        PreparedStatement pps = null;

        try{

            String sql = "delete from bank\_card where b\_c\_id="+b\_c\_id+" and b\_number="+b\_number+";";

            pps = connection.prepareStatement(sql);

            n = pps.executeUpdate();

        }catch (SQLException throwables) {

            throwables.printStackTrace();

        } finally {

            try {

                if (pps != null) {

                    pps.close();

                }

            } catch (SQLException throwables) {

                throwables.printStackTrace();

            }

        }

        return n;

sql语句就是删除，参数有id和number，最后调用方法执行删除操作。

### 2.13.5 客户修改密码

        PreparedStatement pps = null;

        ResultSet rs = null;

        int n = -1;

        try{

            String sql1 = "select \* from client where c\_mail='"+mail+"';";

            pps = connection.prepareStatement(sql1);

            rs = pps.executeQuery();

            if(!rs.next()){

                n = 2;

            }

            else{

                String sql2 = "select \* from client where c\_mail='"+mail+"' and c\_password='"+password+"';";

                pps = connection.prepareStatement(sql2);

                rs = pps.executeQuery();

                if(!rs.next()){

                    n = 3;

                }

                else{

                    String sql3 = "update client set c\_password='"+newPass+"' where c\_mail='"+mail+"' and c\_password='"+password+"';";

                    pps = connection.prepareStatement(sql3);

                    n = pps.executeUpdate();

                    n = 1;}}

        }catch (SQLException throwables) {

            throwables.printStackTrace();

        } finally {

            try {

                if (pps != null) {

                    pps.close();

                }

            } catch (SQLException throwables) {

                throwables.printStackTrace();

            }

        }

        return n;

可以参照第三关的方法，先根据邮箱查找用户的信息，然后客户需要提供用户名和密码，再次查找时，需要密码一致，然后执行更新操作，将新的密码保存输入，并且输出两次新密码（判断相等），最后执行修改函数提交。

### 2.13.6 事务与转账操作

        PreparedStatement pps = null;

        ResultSet rs = null;

        double sourceAmount = 0;

        double destAmount = 0;

        int n = 0;

        try{

            String sql1 = "select \* from bank\_card where b\_number='"+sourceCard+"';";

            connection.setAutoCommit(false);

            pps = connection.prepareStatement(sql1);

            rs = pps.executeQuery();

            if(!rs.next()){

                n=-1;

            }

            else{

                String type = rs.getString("b\_type");

                if(type.equals("信用卡")){

                    n=-1;

                }

                else{

                    sourceAmount = rs.getDouble("b\_balance");

                    if(sourceAmount<amount){

                        n=-1;

                    }

                    else{

                        String sql2 = "select \* from bank\_card where b\_number='"+destCard+"';";

                        pps = connection.prepareStatement(sql2);

                        rs = pps.executeQuery();

                        if(!rs.next()){

                            n=-1;

                        }

                        else{

                            type = rs.getString("b\_type");

                            destAmount = rs.getDouble("b\_balance");

                            sourceAmount = sourceAmount-amount;

                            if(type.equals("信用卡")){

                                destAmount = destAmount-amount;

                            }

                            else{

                                destAmount = destAmount+amount;

                            }

                            String sql3 = "update bank\_card set b\_balance="+sourceAmount+" where b\_number='"+sourceCard+"';";

                            pps = connection.prepareStatement(sql3);

                            pps.executeUpdate();

                            String sql4 = "update bank\_card set b\_balance="+destAmount+" where b\_number='"+destCard+"';";

                            pps = connection.prepareStatement(sql4);

                            pps.executeUpdate();

                            connection.commit();

                            n=1;

                        }

                    }

                }

            }

首先查询转出卡卡号符合的卡的信息，连接成功后查看类型，在不是信用卡的情况下，查看余额，在余额比转账金额多的情况下，查找转入卡的信息，如果转入卡是信用卡，那么因为自动还款，需要减去amount,如果是储蓄卡，那就可以加上转账金额，最后执行修改语句，用execute方法和commit提交修改，注意修改要两个卡都修改，用两个sql语句。

# 3 课程总结

## 3.1 实验总结

本次实验总共完成了统计分数为一百分的实验内容，其中， 第十二关整个关卡跳过，前面的表查询实验有两道未完成，java应用开发有一篇未完成，总体上在规定时间内完成了大部分实验内容。

## 3.2 主要工作

本次实验的主要工作分为两种类型：一种是直接用相应的语句进行相关操作，属于需要掌握操作方法；另一种是设计相关的语句来实现特定的目标，属于设计类型。

本次实验主要训练了数据库的基本操作的写法，包括数据库本身的增删查改，以及相应延伸的视图的一些操作。掌握了一些数据库使用工具与技巧，比如触发器的使用、用户自定义函数的设计、安全性控制的权限授予。在事务操作上，掌握了一些数据库使用注意的问题，并且给出对应的解决方案，也可以通过操作控制事务的隔离以及数据库的恢复。

在设计上，数据查询训练了应对复杂查询的sql语句能力，java数据库应用开发训练了应用高级语言对数据库进行操作的实战能力，基本的增删查改操作以及一些有变化的操作也可以完成。

## 3.3 心得体会

本次数据库实验大部分都是用sql语句完成的，在此之前，我在项目开发中都应用过java或是python完成一些数据库的增删查改操作等，由于层级太高，往往对基础部分理解不够深刻。这门实验课配套数据库课程，原汁原味讲解了数据库的深入构造，sql语句的查询方式也是最直接的应用方式，这提高了我对数据库的理解，也熟练了数据库的相关操作，在一些事务处理上也有了相关的知识储备，不至于只懂技术不懂维护和优化。这门实验课也十分注重实践水平，比较贴近实际应用，让我受益良多。

# 附录