华中科技大学

数据库系统原理实践报告

专　　业：	计算机科学与技术
班　　级：	ACM1901班
学　　号：	U201915035
姓　　名：	邹雅
指导教师：	杨茂林

分数	
教师签名	

2022年 6月30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

“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 为:https://www.educoder.net/paths/x7rzkcvp。实验环境为 Linux 操作系统下的 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

2.1 数据库、表、完整性约束的创建与修改

通过本节实验，可以学会如何启动 MySQL，并创建数据库。掌握在数据库中创建表以及各类约束。了解用 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)。

2.1.1 数据库、表与完整性约束的定义(Create)

1. 创建数据库

使用 MySQL 语句启动数据库服务，接着直接创建数据库 beijing2022，如下图所示，可以看到数据库创建成功。

```
root@evassh-10321793:~# mysql -h127.0.0.1 -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.28 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database beijing2022;
Query OK, 1 row affected (0.01 sec)
```

2. 创建表及表的主码约束

创建表时，增加主码列约束需要在列后加上 PRIMARY KEY 约束。

3. 创建外码约束(foreign key)

创建外码约束格式为：CONSTRAINT 约束名 FOREIGN KEY (列名) REFERENCES 参考的表名(参考表的主码)

因此创建外码约束代码如下所示：

```
drop DATABASE if exists MyDb;
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)
);
```

4. CHECK约束

创建 CHECK 约束的格式为：CONSTRAINT 约束名 CHECK(条件表达式)。

```
CREATE TABLE products(
    pid char(10) PRIMARY KEY,
    name varchar(32),
    brand char(10),
    price INT,
    constraint CK_products_brand CHECK(brand in ('A','B')),
    constraint CK_products_price CHECK(price > 0)
);
```

5. DEFAULT约束

添加 DEFAULT 约束的方式是在创建列时，列的数据类型后加上 default(默认值)。

```
CREATE TABLE hr(
    id char(10) PRIMARY KEY,
    name varchar(32) NOT NULL,

    mz char(16) default("汉族")

);
```

6. UNIQUE约束

添加 UNIQUE 约束的方式是在创建列时，列的数据类型后加上 UNIQUE。

2.1.2 表结构与完整性约束的修改(ALTER)

1. 修改表名

使用 alter 语句进行修改表明，语句如下：

```
alter table your_table rename my_table;
```

2. 添加与删除字段

```
#删除表 orderDetail 中的列 orderDate

alter table orderDetail drop orderDate;

#添加列 unitPrice

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

3. 修改字段

```
#将 QQ 号的数据类型改为 char(12);

alter table addressBook modify QQ CHAR(12);

#将列名 weixin 改为 wechat

alter table addressBook rename column weixin to wechat;
```

4. 添加、删除与修改约束

```
#为表Staff 添加主码
alter table Staff add primary key(staffNo);
#Dept.mgrStaffNo 是外码，对应的主码是Staff.staffNo:
alter table Dept add constraint FK_Dept_mgrStaffNo FOREIGN KEY(mgrSt
affNo) references Staff(staffNo);
#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'));
#为表Dept 添加unique 约束：deptName 不允许重复。
alter table Dept add constraint UN_Dept_deptName UNIQUE(deptName);
```

2.2 数据查询(Select)

本实训采用的是某银行的一个金融场景应用的模拟数据库，数据库中有client(客户表)、bank_card(银行卡)、finances_product(理财产品表)、insurance(保险表)、fund(基金表)、property(资产表)六个表。

这个实践项目主要是数据的查询。每个查询任务都需要用一条 SQL 语句完成某个查询（允许嵌套,允许使用衍生表，但只能是一条 SQL 语句，而不是两条或多条语句，除非关卡有多个任务要求）。

2.2.1 查询客户主要信息

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。

注意到其中的排序子句需要用 order by 进行，默认为升序，加上 desc 关键词后为降序排序。

2.2.2 邮箱为null的客户

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。

在 where 语句中用 is null 进行邮箱字段是否为 null 的判断。

```
1. SELECT c_id,c_name,c_id_card,c_phone
2. FROM client
3. where c_mail is null;
```

2.2.3 既买了保险又买了基金的客户

查询既买了保险又买了基金的客户的名称、邮箱和电话,结果依客户编号排序。

客户是否买了保险和基金需要通过 property 表查询，所以为了查询到既买了保险又买了基金的客户需要两个不相关嵌套子查询。代码如下：

```
1. SELECT c_name, c_mail,c_phone
2. FROM client
3. WHERE c_id in (
4.     SELECT distinct c_id
```

```
    FROM client c1,property p1
    WHERE c1.c_id = p1.pro_c_id and p1.pro_type = 2
) and c_id in (
    SELECT distinct c_id
    FROM client c2,property p2
    WHERE c2.c_id = p2.pro_c_id and p2.pro_type = 3
)
order by c_id
```

2.2.4 办理了储蓄卡的客户信息

查询办理了储蓄卡的客户名称、手机号和银行卡号。注意一个客户在本行可能不止一张储蓄卡，应全列出。查询结果结果依客户编号排序。

需要将 client 表和 bank_card 表进行等值连接查询，并进行判断 bank_card 表中卡的类型为储蓄卡。

```
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;
```

2.2.5 每份金额在30000～50000之间的理财产品

查询理财产品中每份金额在 30000～50000 之间的理财产品的编号,每份金额，理财年限，并按金额升序排序，金额相同的按照理财年限降序排序。

使用 between…and 语句判断理财产品的金额在 30000～50000 之间。

```
select p_id,p_amount,p_year
from finances_product
where p_amount between 30000 and 50000
order by p_amount,p_year desc;
```

2.2.6 商品收益的众数

众数是一组数据中出现次数最多的数值，有时众数在一组数中会有好几个。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，出现的次数命名为 presence。

使用 group by 语句用 pro_income 将 property 表进行分组，用 count(pro_income)函数来计算某个 pro_income 出现的次数。因为要选择出出现次数最多的商品收益，所以需要嵌套子查询和>= all 语句。代码如下：

```
select pro_income,count(pro_income) presence
from property
group by pro_income
having count(pro_income) >= all (
    select count(pro_income)
    from property
    group by pro_income
);
```

2.2.7 未购买任何理财产品的武汉居民

已知身份证前 6 位表示居民地区，其中 4201 开头表示湖北省武汉市。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。依客户编号排序。

首先 4201 开头需要用 like 语句进行判断，其次通过 not exists 相关子查询来进行没有买过理财产品的判断，最后用 order by 语句来进行排序。代码如下：

```
1. select c_name,c_phone,c_mail
2. from client
3. where c_id_card like '4201%' and not exists (
4.     select * from property
5.     where pro_c_id = client.c_id and pro_type = 1
6. )
7. order by c_id;
```

2.2.8 持有两张信用卡的用户

查询持有两张(含）以上信用卡的用户的名称、身份证号、手机号。查询结果依客户编号排序。

用 client 作为外表进行查询，和 bank_card 进行相关嵌套子查询，用 count() 函数来计算出用户拥有的信用卡数量。代码如下：

```
1. select c_name,c_id_card,c_phone
2. from client
3. where (
4.     select count(b_c_id) from bank_card
5.     where b_c_id = c_id and b_type = '信用卡'
6. ) >= 2
7. order by c_id;
```

2.2.9 购买了货币型基金的客户信息

查询购买了货币型(f_type='货币型')基金的用户的名称、电话号、邮箱。依客户编号排序。

首先需要查出货币型的基金编号 f_id，接着查询 property 表中有这些基金编号的客户编号 pro_c_id，最后在 client 表中找出这些客户即可；按照上述流程需要三层嵌套子查询。

```
5. select c_name,c_phone,c_mail
6. from client
7. where c_id in (
8.     select pro_c_id from property
9.     where c_id = pro_c_id and pro_type = 3 and pro_pif_id in (
10.        select f_id from fund
11.        where f_type='货币型'
12.    )
```

```
13.)
14.order by c_id;
```

2.2.10 投资总收益前三名的客户

查询当前总的可用资产收益(被冻结的资产除外)前三名的客户的名称、身份证号及其总收益，按收益降序输出，总收益命名为 total_income。不需要考虑并列排名情形。

在本关中使用了派生表来计算客户总的可用资产收益，并在用 order by 子句降序(desc)排序后，用 limit 3 语句来选择出前三名。按照上述思路可以写出代码如下：

```
1. select c_name,c_id_card,c_income total_income
2. from client,(select pro_c_id,sum(pro_income) c_income
3.    from property
4.    where pro_status not in ('冻结')
5.    group by pro_c_id
6. ) as P_Client
7. where c_id = pro_c_id
8. order by c_income desc
9. limit 3;
```

2.2.11 黄姓客户持卡数量

给出黄姓用户的编号、名称、办理的银行卡的数量(没有办卡的卡数量计为0)。银行卡数量命名为 number_of_cards。 按办理银行卡数量降序输出,持卡数量相同的,依客户编号排序。

由于需要输出没有办卡的客户信息，所以需要用到左外连接 left outer join。

```
1. select c_id,c_name,count(b_c_id) number_of_cards
2. from client left outer join bank_card on b_c_id = c_id
3. where c_name like '黄%'
4. group by c_id
5. order by count(b_c_id) desc,c_id;
```

2.2.12 客户理财、保险与基金投资总额

综合客户表(client)、资产表(property)、理财产品表(finances_product)、保险表(insurance)和基金表(fund)，列出客户的名称、身份证号以及投资总金额（即投资本金，每笔投资金额=商品数量*该产品每份金额)，注意投资金额按类型需查询不同的表，投资总金额是客户购买的各类(理财,保险,基金)资产投资金额的总和，总金额命名为 total_amount。查询结果按总金额降序排序。

分析上述任务题意，其中最关键的就是求出投资总金额。分析可知，对于理财、保险、基金我们需要分别求出其投资金额，而计算投资金额需要用 property 表中的 pro_quantity 分别 finances_product、insurance、fund 表中的金额相乘来得到。需要注意的是，如果选择到的数据为 null 的话，应该需要判断为空

并赋予 0 值，否则可能会出现错误，我们用 ifnull 语句进行。按照思路写出代码如下：

```
select c_name,c_id_card,
(ifnull(
    (select sum(pro_quantity*p_amount)
    from property,finances_product
    where c_id = pro_c_id and pro_type = 1 and p_id = pro_pif_id),0
) +
ifnull(
    (select sum(pro_quantity*i_amount)
    from property,insurance
    where c_id = pro_c_id and pro_type = 2 and i_id = pro_pif_id),0
) +
ifnull(
    (select sum(pro_quantity*f_amount)
    from property,fund
    where c_id = pro_c_id and pro_type = 3 and f_id = pro_pif_id),0
)) as total_amount
from client
order by total_amount desc;
```

2.2.13 客户总资产

综合客户表(client)、资产表(property)、理财产品表(finances_product)、保险表(insurance)、基金表(fund)，列出所有客户的编号、名称和总资产，总资产命名为 total_property。总资产为储蓄卡总余额，投资总额，投资总收益的和，再扣除信用卡透支的总金额(信用卡余额即为透支金额)。客户总资产包括被冻结的资产。

按照给的任务思想，核心在于求出客户的总资产。客户总资产需要用 sum()函数分别求出储蓄卡中的总余额、客户的商品收益总和，再加上同 2.2.12 中所求的客户理财、保险、基金投资总额，再减去信用卡中投资的总金额。整体思路和上一小节一致。写出代码如下。

```
select c_id,c_name,ifnull(
    (select sum(b_balance)
    from bank_card
    where c_id = b_c_id and b_type = '储蓄卡'),0
) +
ifnull(
    (select sum(pro_income)
    from property
    where c_id = pro_c_id),0
) +
ifnull(
    (select sum(pro_quantity*p_amount)
    from property,finances_product
```

```
  where c_id = pro_c_id and pro_type = 1 and p_id = pro_pif_id),0
) +
ifnull(
   (select sum(pro_quantity*i_amount)
   from property,insurance
  where c_id = pro_c_id and pro_type = 2 and i_id = pro_pif_id),0
) +
ifnull(
   (select sum(pro_quantity*f_amount)
   from property,fund
  where c_id = pro_c_id and pro_type = 3 and f_id = pro_pif_id),0
) -
ifnull(
   (select sum(b_balance)
   from bank_card
   where c_id = b_c_id and b_type = '信用卡'),0
)
as total_property
from client;
```

2.2.14 第N高问题

查询每份保险金额第 4 高保险产品的编号和保险金额。在数字序列 8000,8000,7000,7000,6000 中，两个 8000 均为第 1 高，两个 7000 均为第 2 高,6000 为第 3 高。

需要通过相关子查询计算出大于等于当前保险金额是否为是 4 个。

```
select i_id,i_amount
from insurance as SI
where 4 = (
select count(distinct i_amount) from insurance
where i_amount >= SI.i_amount
);
```

2.2.15 基金收益两种方式排名

查询资产表中客户编号，客户基金投资总收益,基金投资总收益的排名(从高到低排名)。总收益相同时名次亦相同(即并列名次)。总收益命名为 total_revenue,名次命名为 rank。第一条 SQL 语句实现全局名次不连续的排名，第二条 SQL 语句实现全局名次连续的排名。不管哪种方式排名，收益相同时,客户编号小的排在前。

实现排名需要使用派生表来先计算出客户基金投资总收益，接着再计算出基金总收益的排名需要用子查询嵌套子查询来完成。计算排名的思路是，查询出比当前外表所列出的总收益 total_revenue 大的总收益数量。

当实现的是全局名次不连续的情况，应是计算大于当前 total_revenue 的数量加一，举例来说，没有比当前总收益更大的情况下当前总收益应该排名第一；当实现的是全局名次连续的情况，应是计算大于等于当前 total_revenue 的数量。

a) 基金总收益排名（全局名次不连续）

```
select distinct pro_c_id, total_revenue,(
    select count(income_sum) + 1
    from (
        select distinct pro_c_id,sum(pro_income) income_sum
        from property p3
        where pro_type = 3
        group by pro_c_id
    ) as income_m
    where income_sum > total_revenue
) as `rank`
from (select pro_c_id,sum(pro_income) total_revenue
        from property
        where pro_type = 3
        group by pro_c_id
) as income_m
order by total_revenue desc,pro_c_id;
```

b) 基金总收益排名（全局名次连续）

```
select distinct pro_c_id, total_revenue,(
    select count(distinct income_sum)
    from (
        select distinct pro_c_id,sum(pro_income) income_sum
        from property p3
        where pro_type = 3
        group by pro_c_id
    ) as income_m
    where income_sum >= total_revenue
) as `rank`
from (select pro_c_id,sum(pro_income) total_revenue
        from property
        where pro_type = 3
        group by pro_c_id
) as income_m
order by total_revenue desc,pro_c_id;
```

2.2.16 持有完全相同基金组合的客户

查询持有相同基金组合的客户对，如编号为 A 的客户持有的基金，编号为 B 的客户也持有，反过来，编号为 B 的客户持有的基金，编号为 A 的客户也持有，则(A,B)即为持有相同基金组合的二元组，请列出这样的客户对。为避免过多的重复，如果(1,2)为满足条件的元组，则不必显示(2,1)，即只显示 编号小者在前的那一对，这一组客户编号分别命名为 c_id1,c_id2。

分析上述题意，可以用 not exists 相关子查询来完成。外表查询需要有两个 property 表 p1,p2 进行连接，为了避免重复增加 p1.pro_c_id < p2.pro_c_id 的条件。此外用 not exists 语句判断出不存在 p1 中客户持有的基金 p2 没有，也不存在 p2 中客户持有的基金 p1 没有。按照此思路即可写出代码。

```
select distinct p1.pro_c_id c_id1,p2.pro_c_id c_id2
from property p1,property p2
where p1.pro_type = 3 and p1.pro_c_id < p2.pro_c_id
and not exists(
select * from property p3
where p3.pro_c_id = p1.pro_c_id and p3.pro_type = 3 and not exists(
select * from property p4
where p4.pro_c_id = p2.pro_c_id and p4.pro_type = 3
and p3.pro_pif_id = p4.pro_pif_id
)
)
and not exists(
select * from property p3
where p3.pro_c_id = p2.pro_c_id and p3.pro_type = 3 and not exists(
select * from property p4
where p4.pro_c_id = p1.pro_c_id and p4.pro_type = 3
and p3.pro_pif_id = p4.pro_pif_id
)
)
order by p1.pro_c_id;
```

2.2.17 购买基金的高峰期

该关卡任务已完成，实施情况本报告略过。

2.2.18 至少有一张信用卡余额超过5000元的客户信用卡总余额

查询至少有一张信用卡余额超过 5000 元的客户编号，以及该客户持有的信用卡总余额，总余额命名为 credit_card_amount。查询结果依客户编号排序。

在 bank_card 表中用 b_c_id 进行分组，需要选择出至少有一张信用卡信用卡余额超过 5000 元的用户，需要用 sum 函数统计：如果余额超过 5000 元则计数加一，否则不计算，如果该值大于 0 则可以选择。为了在选择出的列表中展示总金额，需要用一个子查询。

```
select b_c_id,(
    select sum(b_balance)
    from bank_card b2
    where b1.b_c_id = b2.b_c_id and b_type = '信用卡'
)credit_card_amount
from bank_card b1
where b_type = '信用卡'
group by b_c_id
having sum(if(b_balance > 5000,1,0)) >= 1
order by b_c_id;
```

2.2.19 以日历表格式显示每日基金购买总金额

以日历表格式列出 2022 年 2 月每周每个交易日基金购买总金额。

利用 weekday()函数来判断当天的日期，函数返回 0 为 Monday，1 为 Tuesday，以此类推。每个交易日的基金购买总金额都需要通过子查询语句来完成，子查询的 where 子句需要判断多个条件：1. 年份为 2022 年 2. 所在交易周次为表第一列所展示的周次 3. 日期和当前的列名一致 4. 类型为基金 5. property 表和 fund 表的等值连接。在外查询中，需要筛选 2022 年 2 月，用 year()函数判断年份，用 month()函数判断月份。列出代码如下所示：

```
1. select distinct week(pro_purchase_time) - 5 as week_of_trading,(
2. select sum(pro_quantity*f_amount)
3. from property p2,fund
4. where week_of_trading+5 = week(p2.pro_purchase_time)
5. and weekday(p2.pro_purchase_time) = 0 and p2.pro_pif_id = f_id
6. and p2.pro_type = 3 and year(p2.pro_purchase_time) = 2022
7. ) as Monday,(
8. select sum(pro_quantity*f_amount)
9. from property p2,fund
10.where week_of_trading+5 = week(p2.pro_purchase_time)
11.and weekday(p2.pro_purchase_time) = 1 and p2.pro_pif_id = f_id
12.and p2.pro_type = 3 and year(p2.pro_purchase_time) = 2022
13.) as Tuesday,(
14.select sum(pro_quantity*f_amount)
15.from property p2,fund
16.where week_of_trading+5 = week(p2.pro_purchase_time)
17.and weekday(p2.pro_purchase_time) = 2 and p2.pro_pif_id = f_id
18.and p2.pro_type = 3 and year(p2.pro_purchase_time) = 2022
19.) as Wednesday,(
20.select sum(pro_quantity*f_amount)
21.from property p2,fund
22.where week_of_trading+5 = week(p2.pro_purchase_time)
23.and weekday(p2.pro_purchase_time) = 3 and p2.pro_pif_id = f_id
24.and p2.pro_type = 3 and year(p2.pro_purchase_time) = 2022
25.) as Thursday,(
26.select sum(pro_quantity*f_amount)
27.from property p2,fund
28.where week_of_trading+5 = week(p2.pro_purchase_time)
29.and weekday(p2.pro_purchase_time) = 4 and p2.pro_pif_id = f_id
30.and p2.pro_type = 3 and year(p2.pro_purchase_time) = 2022
31.) as Friday
32.from property p1
33.where year(pro_purchase_time) = 2022 and month(pro_purchase_time)
    = 2;
```

2.3 数据的插入、修改与删除(Insert,Update,Delete)

2.3.1 插入多条完整的客户信息

插入数据 SQL 语句的格式为：insert 表名 values(value1,value2,……)

2.3.2 插入不完整的客户信息

插入不完整数据的 SQL 语句格式为：insert into 表名(属性名 1, 属性名 2,……) values(value1,value2,……);

其中 values 后的值与属性名应该一一对应。

2.3.3 批量插入数据

表 new_client 保存了一批新客户信息，该表与 client 表结构完全相同。此时任务为需要用一条 SQL 语句将 new_client 表的全部客户信息插入到客户表(client)。很显然，我们可以在 insert 语句中加入 select 查询语句来完成。

```
insert client select * from new_client;
```

2.3.4 删除没有银行卡的客户信息

用一条 SQL 语句删除 client 表中没有银行卡的客户信息。使用 delete 语句来实现，注意的是在 MySQL 中 from 关键词不能省略。

```
delete from client
where c_id not in (
    select b_c_id from bank_card
);
```

2.3.5 冻结客户资产

用一条 update 语句将手机号码为“13686431238”这位客户的投资资产的状态置为“冻结”。在 update 时，将条件写在 where 分句中。

```
update property

set pro_status = '冻结'

where pro_c_id = (
    select c_id from client where c_phone = '13686431238'
);
```

2.3.6 连接更新

用一条 update 语句，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)。我们可以直接使用内连接 property 表和 client 表来完成。

```
update property inner join client on property.pro_c_id = client.c_id
set property.pro_id_card = client.c_id_card;
```

2.4 视图

2.4.1 创建所有保险资产的详细记录视图

创建包含所有保险资产记录的详细信息的视图 v_insurance_detail。

创建视图的语句格式为 create view 视图名 as 查询语句。通过查询语句选择出我们需要在视图中展示的属性列，根据数据查询的知识不难写出。

```
create view v_insurance_detail as
select c_name,c_id_card,i_name,i_project,pro_status,pro_quantity,i_a
mount,i_year,pro_income,pro_purchase_time
from client,insurance,property
where c_id = pro_c_id and pro_pif_id = i_id and pro_type = 2;
```

2.4.2 基于视图的查询

对视图的查询和对基本表的查询使用的语句是一致的。本关任务需要完成列出每位客户的姓名，身份证号，保险投资总额和保险投资总收益，结果依保险投资总额降序排列。

其中需要特别关注的是保险投资总额和保险投资总收益，需要使用单独的 select 语句进行计算，而且还需要注意的是，如果 select 出的总收益为 null 的话，需要处理成 0。在这里用 ifnull 语句来进行判断完成。

```
1. select distinct c_name,c_id_card,ifnull(
2. (select sum(pro_quantity*i_amount) from v_insurance_detail v2
3. where v1.c_name = v2.c_name),0
4. ) as insurance_total_amount,ifnull(
5. (select sum(pro_income) from v_insurance_detail v3
6. where v1.c_name = v3.c_name),0
7. ) as insurance_total_revenue
8. from v_insurance_detail v1
9. order by insurance_total_amount desc;
```

2.5 存储过程与事务

2.5.1 使用流程控制语句的存储过程

该关卡任务已完成，实施情况本报告略过。

2.5.2 使用游标的存储过程

该任务关卡跳过。

2.5.3 使用事务的存储过程

该任务关卡跳过。

2.6 触发器

2.6.1 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

该关卡任务已完成，实施情况本报告略过。

2.7 用户自定义函数

2.7.1 创建函数并在语句中使用它

该关卡任务已完成，实施情况本报告略过。

2.8 安全性控制

2.8.1 用户和权限

使用 create 语句创建用户 tom 和 jerry，初始密码均为'123456'：

```
create user 'tom' identified by '123456';
create user 'jerry' identified by '123456';
```

使用 grant 语句进行授权：

```
grant select(c_name,c_mail,c_phone) on client
to 'tom'
with grant option;
```

使用 revoke 语句进行授权：

```
revoke select on bank_card from 'Cindy';
```

2.8.2 用户、角色与权限

使用 create 语句创建角色。

```
create role client_manager;
create role fund_manager;
```

使用 grant 语句授予 client_manager 对 client 表拥有 select, insert, update 的权限：

```
grant select,insert,update on client to client_manager;
```

使用 grant 语句将将 client_manager 的权限授予用户 tom 和 jerry：

```
grant client_manager to 'tom','jerry';
```

2.9 并发控制与事务的隔离级别

本实训先介绍并发控制和事务的隔离级别，接着要求通过设置恰当的的隔离级别，编写两个事务(测试平台会让两个事务并发执行)，构造读脏、不可重复读、幻读等场景，最后要求在 read uncommitted 隔离级别下，手工加锁，保证不可重复读。

2.9.1 并发控制与事务的隔离级别

可以利用如下代码设置事务的隔离级别为 read uncommitted：

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

2.9.2 读脏

该关卡任务已完成，实施情况本报告略过。

2.9.3 不可重复读

该关卡任务已完成，实施情况本报告略过。

2.9.4 幻读

该关卡任务已完成，实施情况本报告略过。

2.9.5 主动加锁保证可重复读

该关卡任务已完成，实施情况本报告略过。

2.9.6 可串行化

事务t1两次查询航班MU2455的余票，事务t2修改航班MU2455的余票(减1)。请对两个代码文件进行修改，使得两个事务并发执行的结果与t2→t1串行执行的结果相同。

除两个事务的 select 和 update 语句不可修改外，可以修改、添加代码。但不得将事务的隔离级别设置为 serializable，可以保持默认隔离级别，或设置成其它隔离级别。

采用让事务休眠不同时间的方式来错开事务的执行。

事务一：

```
use testdb1;
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;
```

事务二：

```
use testdb1;
start transaction;
set @n = sleep(1);
update ticket set tickets = tickets - 1 where flight_no = 'MU2455';
commit;
```

2.10 备份+日志：介质故障与数据库恢复

MySQL 利用备份、日志文件实现恢复，提供给用户逻辑备份工具 mysqldump、物理备份工具 mysqlbackup、日志工具 mysqlbinlog、还原工具 mysql、管理工具 mysqladmin 来进行备份与恢复操作。在本章节需要学会使用这些工具。

2.10.1备份与恢复

在 test1_1.sh 文件中，对数据库 residents 作海量备份，备份至文件 residents_bak.sql 中，代码如下：

```
mysqldump -h127.0.0.1 -uroot --databases residents --result-file=res
idents_bak.sql
```

在 test1_2.sh 文件中，利用备份文件 residents_bak.sql 还原数据库：

```
mysql -h127.0.0.1 -uroot  < residents_bak.sql
```

2.10.2备份+日志：介质故障的发生与数据库的恢复

在 test1_1.sh 文件中，对数据库 train 作逻辑备份并新开日志文件的命令：

```
mysqldump -h127.0.0.1 -uroot --databases train --result-file=train_b
ak.sql
mysqladmin flush-logs
```

在test1_2.sh文件中，利用逻辑备份和日志恢复数据库：

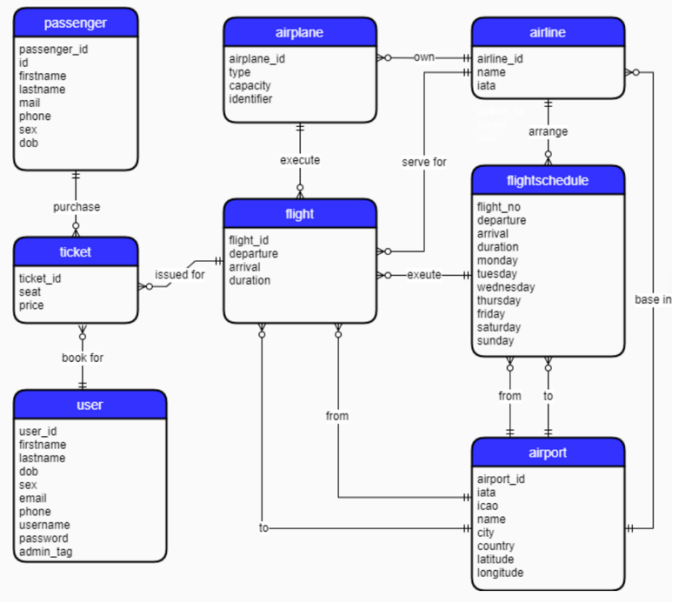
```
mysql -h127.0.0.1 -uroot  < train_bak.sql
mysqlbinlog --no-defaults log/binlog.000018 | mysql -u root
```

2.11数据库设计与实现

数据库有概念模型、逻辑模型和物理模型。本章节需要完成从概念模型到MySQL 实现、从需求分析到逻辑模型的转换以及建模工具的使用。完成本章节需要深入掌握 ER 图，同时根据具体设计要求完成 MySQL 的代码实现。

2.11.1 从概念模型到 MySQL 实现

根据 ER 图中所标注出来的实体——用户、旅客、机场、航空公司、民航飞机、航班常规调度表、航班表、机票分别 create table，省略完整的具体代码，列出部分具有代表性的代码，在这里指出各个实体之间的联系以及需要注意的地方。



航空公司有机场，需要增加一个 airport_id 属性；民航飞机属于某一家航空公司，需要增加 airline_id 属性。由于声明表的顺序，所以将部分外码约束的声明放在表外进行，具体代码给出 airline、airport、airplane 部分进行示例：

```
drop database if exists flight_booking;
create database flight_booking;
use flight_booking;

-- #航空公司

create table airline(
    airline_id int PRIMARY KEY  AUTO_INCREMENT,
    `name` varchar(30) not null,
```

```
    iata char(2) not null unique,
    airport_id int not null
);

-- #民航飞机

create table airplane(
    airplane_id int PRIMARY KEY  AUTO_INCREMENT,
    `type` varchar(50) not null,
    capacity smallint not null,
    identifier varchar(50) not null,
    airline_id int not null
);

-- #机场

create table airport(
    airport_id int PRIMARY KEY AUTO_INCREMENT,
    iata char(3) not null unique,
    icao char(4) not null unique,
    `name` varchar(50) not null,
    city varchar(50),
    country varchar(50),
    latitude decimal(11,8),
    longitude decimal(11,8)
);
alter table airline
add CONSTRAINT FK_airline_airport
FOREIGN KEY (airport_id) REFERENCES airport(airport_id);

alter table airplane
add CONSTRAINT FK_airplane_airline
FOREIGN KEY (airline_id) REFERENCES airline(airline_id);
```

此外还需要创建索引 index：

```
create index name_idx on airport(`name`);
```

航班表的外码属性比较多。航班表中需要增加一个 flight_no 属性作为航班常规调度表的外码，增加 airline_id、airplane_id 来标注航班表的航空公司和民航飞机，增加 from、to 来标记起飞和降落的机场号，具体实现部分如下：

```
-- #航班表

create table flight(
    flight_id int PRIMARY KEY AUTO_INCREMENT,
    departure datetime not null,
    arrival datetime not null,
    duration smallint not null,
    flight_no char(8) not null,
```

```
    airline_id int not null,
    airplane_id int not null,
    `from` int not null,
    `to` int not null,
     constraint FK_flight_airline FOREIGN KEY (airline_id) REFERENCE
S airline(airline_id),
     constraint FK_flight_airplane FOREIGN KEY (airplane_id) REFEREN
CES airplane(airplane_id),
     CONSTRAINT FK_flight_from FOREIGN KEY (`from`) REFERENCES airpo
rt(airport_id),
     CONSTRAINT FK_flight_to FOREIGN KEY (`to`) REFERENCES airport(a
irport_id)
);
```

航班常规调度表同理需要增加 airline_id、from、to 属性；机票需要增加 flight_id 标注航班表的 id 号、passenger_id 标注旅客的 id、user_id 标注用户的 id。用户、旅客、航班常规调度表、机票具体实现同上所给出的示例相同进行建表即可。

2.11.2 从需求分析到逻辑模型

该任务关卡跳过。

2.11.3建模工具的使用

该任务关卡跳过。

2.11.4 制约因素分析与设计

从工程的角度进行分析，我们在完成数据库任务的解决方案设计中需要考虑了社会、健康、安全、法律、文化及环境等制约因素。

比如在 2.11.1 从概念模型到 MySQL 实现具体例子来看，我们设计一个机票订票系统，除了考虑到单个实体的属性以外，还需要结合社会应用中各个实体之间的联系来设计。此外，我们必须要考虑到订票系统对航空公司、用户等各方面的安全性保障，其中对密码安全性等的考虑需要遵循相关法律，防止在实际应用中出现事故。除了这些必要性要求之外，我们还需要考虑如何让用户在使用该订票系统的时候觉得舒适，不会有不合理、反直觉的设计。

2.11.5 工程师责任及其分析

数据俨然已经成为所有企业的重要资产，互联网企业更是将数据作为自己的生命线。随着数据库技术的不断发展，企业为了管理、利用越来越多的信息，都建立了自己的数据库。而这些企业数据库，都需要有专门的人员进行维护，这就是数据库工程师的工作。

对于数据库开发工程师而言，其职责有：

1. 负责公司业务数据库系统的模型设计，表结构设计
2. 负责数据处理中的语句实现，存储过程逻辑实现
3. 负责指导开发人员对语句的性能优化和指导

而对于数据库管理员（DBA）则是：

1. 负责公司业务数据库系统的部署实现

2. 负责数据库系统的高可用性，备份恢复，性能调忧，监控等实现

除了上述的工程师职责要求之外，工程师还需要承担起社会责任，在心中建立起社会主义核心价值观，遵守法律。

2.12数据库应用开发(JAVA篇)

JDBC 是一种用于执行 SQL 语句的 Java API，可以为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成，使数据库开发人员能够编写数据库应用程序。

在这个章节，我们需要正确掌握 JDBC 的体系结构、核心组件和使用步骤，完成对 MySQL 中的数据查询、插入、删除、修改、事务处理等任务。

2.12.1 JDBC体系结构和简单的查询

在本小节中需要完成查询金融应用场景数据库 finance 的 client 表中邮箱不为空的客户信息，列出客户姓名，邮箱和电话。

在 try{}catch 块中补充我们需要的代码，这是因为在执行 JDBC 语句时有可能会抛出 ClassNotFoundException、SQLException 异常。

我们写出对应的 MySQL 查询语句如下：

```
select c_name,c_mail,c_phone from client where c_mail is not null;
```

使用JDBC首先需要注册 JDBC 驱动程序，使用如下语句：

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

加载驱动程序之后，进行数据库 URL 配置，创建数据库连接对象：

```
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);
```

JDBC 中的 Statement 和 PreparedStatement 接口定义了能够发送 SQL 命令并从数据库接收数据的方法和属性。需要创建 Statement 的一个实例，并执行。

```
statement = connection.createStatement();
resultSet = statement.executeQuery(SQL_str);
System.out.print("姓名\t");
System.out.print("邮箱\t\t\t\t");
System.out.print("电话\n");//打印出标题
while (resultSet.next()) {//对resultSet进行遍历打印出查询到的数据
    System.out.print(resultSet.getString("c_name"));
    System.out.print("\t");
    System.out.print(resultSet.getString("c_mail"));
    System.out.print("\t\t");
    System.out.print(resultSet.getString("c_phone"));
    System.out.print("\n");
}
```

2.12.2 用户登录

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。

加载驱动程序、数据库 URL 配置、创建数据库连接对象等操作都和上一小节一致，查询数据并执行的代码如下所示：

```
1. String sql = "select * from client where c_mail = ? and c_passwor
   d = ?;";
2. statement = connection.prepareStatement(sql);
3. statement.setString(1,loginName);
4. statement.setString(2,loginPass);
5. resultSet = statement.executeQuery();
```

如果查询到了结果，那么说明数据库中有对应的用户，登陆成功；反之登陆失败。对应给出结果的代码如下所示：

```
1. if(resultSet.next()){
2. System.out.println("登录成功。");
3. }else{
4. System.out.println("用户名或密码错误！");
5. }
```

2.12.3 添加新客户

编程完成向 client 插入记录的方法 insertClient()，其每个参数都在注释中进行了详细的说明。我们利用 PreparedStatement 类创建其对象，来完成条件不确定的查询，可以把变量加入到 SQL 语句中。在 insertClient()方法的 try 块中添加如下语句：

```
1. PreparedStatement statement = null;
2. String sql = "insert client values(?,?,?,?,?,?);";
3. statement = connection.prepareStatement(sql);
4. statement.setInt(1,c_id);statement.setString(2,c_name);
5. statement.setString(3,c_mail); statement.setString(4,c_id_card);
6. statement.setString(5,c_phone);statement.setString(6,c_password);
7. int n = statement.executeUpdate();
8. return n;
```

2.12.4 银行卡销户

编写一个能删除指定客户编号的指定银行卡号的方法。我们编程的思路同上，只需要使用 SQL 语句的 delete 语句即可。对应语句如下所示：

```
1. String sql = "delete from bank_card where b_c_id = ? and b_number
    = ?";
2. statement = connection.prepareStatement(sql);
3. statement.setInt(1,b_c_id);
4. statement.setString(2,b_number);
5. int n = statement.executeUpdate();
```

```
6. return n;
```

2.12.5 客户修改密码

实现修改密码的方法 passwd()。客户修改密码通常需要确认客户身份，即客户需提供用户名(以邮箱为用户名)和密码，同时还需要输两次新密码，只有当所有条件(合法的客户，两次密码输入一致)时才修改密码。

passwd()方法返回一个整数：1 表示密码修改成功，2 表示用户不存在，3 表示密码不正确，-1 表示程序异常(如没能连接到数据库等）。

由于程序有很多种可能性，所以我们需要分类选择来讨论情况。

首先是判断用户是否存在：

```
1. String sql_select = "select * from client where c_mail = "+"'"+mail+"'"+";";
2. se_statement = connection.createStatement();
3. resultSet = se_statement.executeQuery(sql_select);
4. if(!resultSet.next()){
5. return 2;
6. }
```

其次是判断用户的密码是否正确：

```
1. String sql_select_pass = "select * from client where c_mail = "+"'"+mail+"'"+" and c_password = "+"'"+password+"'"+";";
2. resultSet = se_statement.executeQuery(sql_select_pass);
3. if(!resultSet.next()){
4. return 3;
5. }
```

接着是修改密码并判断密码是否修改成功。我们知道 executeUpdate()函数返回大于 0 的整数时表示执行成功。这部分代码如下：

```
1. String sql = "update client set c_password = ? where c_mail = ?;";
2. statement = connection.prepareStatement(sql);
3. statement.setString(1,newPass);
4. statement.setString(2,mail);
5. int n = statement.executeUpdate();
6. if(n>0)  return 1;   else  return -1;
```

2.12.6 事务与转账操作

编写一个银行卡转账的方法 transferBalance()。transferBalance()在被调用前，柜台已经确认过转出帐号持有者身份，所以转帐方法只接受转出卡号，转入卡号和转账金额三个参数。由调用者保证转账金额为正数。transferBalance()返回 boolean 值，true 表示转帐成功，false 表示转账失败，并不需要细分或解释失败的原因。

下列任一情形都不可转账(转账失败的原因)：转出或转入帐号不存在、转出账号是信用卡、转出帐号余额不足。

首先我们先判断转出或转入帐号是否不存在，这里列出了查询转出账号是否存在的代码，查询转入账号和后续更新操作合并见下一部分代码。其次判断转出

账号是否是信用卡，此外还需要判断转出账号是否余额不足，这部分代码如下所示：

```
se_statement = connection.createStatement();
String sql_select_source = "select * from bank_card where b_numbe
r = "+"'"+sourceCard+"'"+";";
resultSet = se_statement.executeQuery(sql_select_source);

//转出账号不存在

if(!resultSet.next()){
    return false;
}

//转出账号是信用卡


if(resultSet.getString("b_type").equals("信用卡")){

    return false;
}

//转出账号余额不足

if(resultSet.getInt("b_balance")<amount){
    return false;
}
```

最后执行转入和转出操作，其中需要注意的是，如果转入的是信用卡，那么其操作应该是“-”，而不是增加：

```
String sql_select_desc = "select * from bank_card where b_number
= "+"'"+destCard+"'"+";";
resultSet = se_statement.executeQuery(sql_select_desc);

//转入账号不存在

if(!resultSet.next()){
    return false;
}

if(resultSet.getString("b_type").equals("信用卡")){

    flag = true;
}

//执行转出操作

String sql_out = "update bank_card set b_balance = b_balance - ?
where b_number = ?;";
statement = connection.prepareStatement(sql_out);
statement.setDouble(1,amount);
statement.setString(2,sourceCard);
statement.executeUpdate();
```

```
//执行转入操作
if(flag){
String sql_in = "update bank_card set b_balance = b_balance - ? where b_number = ?;";
    statement = connection.prepareStatement(sql_in);
    statement.setDouble(1,amount);
    statement.setString(2,destCard);
    statement.executeUpdate();
}else{
String sql_in = "update bank_card set b_balance = b_balance + ? where b_number = ?;";
    statement = connection.prepareStatement(sql_in);
    statement.setDouble(1,amount);
    statement.setString(2,destCard);
    statement.executeUpdate();
}
return true;
```

2.12.7把稀疏表格转为键值对存储

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。

编写 insertSC(Connection connection,int sno,String col_name,String col_value)函数用于向 SC 表中插入数据。在 main 函数中，用下列查询语句选择出数据：

```
select * from entrance_exam;
```

接着通过 resultSet 进行遍历，如果某个列值不为空，则转换形式插入 SC 表中，这里举两个例子进行描述，实际代码中应该对 chinese、math、English、physics、chemistry、biology、history、geography、politics 都进行处理。代码如下所示：

```
while (resultSet.next()) {
int sno = resultSet.getInt("sno");
if(resultSet.getString("chinese")!=null){
insertSC(connection,sno,"chinese",resultSet.getString("chinese"));
}
if(resultSet.getString("math")!=null){
insertSC(connection,sno,"math",resultSet.getString("math"));
}
if(resultSet.getString("english")!=null){
insertSC(connection,sno,"english",resultSet.getString("english"));
}
if(resultSet.getString("physics")!=null){
insertSC(connection,sno,"physics",resultSet.getString("physics"));
}
}
……
}
```

3 课程总结

数据库系统原理实践是在理论课程数据库系统原理的基础上通过MySQL数据库进行实践的实验，我们需要用针对于 MySQL 的SQL语句完成发布在头歌实践平台上的任务。任务涉及对数据库、表等数据对象地管理与编程、数据查询、数据的增删查改、系统内核实验等五大部分内容。本人在此次实验中共完成58关。

总结本次实验，我主要完成的内容有：

1. 实现了对数据库、表、视图、完整性约束条件等内容的创建与修改。这一部分实验让我了解到了MySQL数据库的启动和创建，同时也让我掌握了如何创建表以及在表中实现数据库逻辑模型中所要求的实体完整性、参照完整性和用户自定义完整性。
2. 通过对数据查询的实践与应用，我在更深层次上理解了查询语句，对嵌套子查询、相关子查询、派生表等等知识点都有了更加清晰的认识和掌握。此外我还学习到了很多数据库条件函数、数据库日期函数的使用。除此之外，我还应用了数据的插入、删除和修改，在外模式的理论基础上实践了视图的创建和查询使用。
3. 实现了安全性控制、事务并发与事务隔离级别、介质故障与数据库恢复等任务。这部分任务让我从系统的角度了解了数据库的运行机制，更深层次地掌握了数据库事务并发的情况下如何实现事务隔离和可串行化调度，实践了数据库发生故障如何进行恢复。
4. 完成了数据库设计与实现和数据库应用系统的开发。通过完成从ER图到MySQL实现的实验，我经历了设计一个完整数据库系统的过程，体会到其中设计实体联系的多重考虑。而在数据库应用系统开发的实验中，通过使用 Java 来实现数据库的应用，有助于我们和实际工程接轨，不会只停留在SQL语言中而对工程一无所知。

总体来说，这一门课程给了我实践数据库系统原理的机会，加深了我对理论知识的理解。我认为该实验和课本上学到的知识结合得非常紧密，但又让我们跳出了理论 SQL 语句的框架，和 MySQL 数据库实际接轨。这是一门设计完备，且让实验课时发挥其充分意义的实验。

如果要给出一些建议，那么我觉得在数据查询这个章节的设计，涉及到了很多复杂的查询要放在一条 SQL 语句里去完成，这在实际应用中是效率低、可扩展性差、易读性不高的。我觉得可以给学生更多的空间，结合课本的知识去思考如何在复杂的场景下写出高效、易读的语句。