# 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： | CS2005 |
| 学　　号： | U202090063 |
| 姓　　名： | 董玲晶 |
| 指导教师： | 袁平鹏 |

| | |
|---|---|
| 分数 | |
| 教师签名 | |

2023　　年　1 月　3 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 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）

本小节共有六个关卡，主要围绕数据库和表的创建展开，主要内容有：数据库的创建、表的创建和表与表之间的主码、外码、CHECK、DEFAULT、UNIQUE约束。

### 2.1.1 创建数据库

- 任务要求：创建用于 2022 年北京冬奥会信息系统的数据库（beijing2022）。

```
create database beijing2022;
```

代码块 2.1.1-1 实验 1_1 答案

### 2.1.2 创建表及表的主码约束

- 任务要求：创建数据库 TestDb，在 TestDb 下创建表 t_emp，并为表建主码。
- 思路解析

先使用 create 语句创建数据库，使用 use 语句使用数据库；再用 create 语句按要求创建表，同时使用 primary key 来指定主码为 id。

```
create database TestDb;
use TestDb;
create table t_emp(
        id INT primary key,
        name VARCHAR(32),
        deptId INT,
        salary FLOAT
);
```

代码块 2.1.2-1 实验 1_2 答案

### 2.1.3 创建外码约束

- 任务要求

建要求的两个表 dept 和 staff，并为表定义主键，给表 staff 创建外键。

- 思路解析

使用两个 create 语句分别创建 dept 表和 staff 表，主码分别为 deptNo 和 staffNo，使用 primary key 来设置。按照要求，如代码块中高亮行所示，创建表 staff 的 deptNo 到表 dept 的 deptNo 的外键约束，约束名命名为 FK_staff_deptNo。

```
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)
);
```

代码块 2.1.3-1 实验 1.3 代码

### 2.1.4 CHECK 约束

- 任务要求

在数据库 MyDb 中创建表 products，并分别实现对品牌和价格的约束，两个 CHECK 约束的名称分别为 CK_products_brand 和 CK_products_price，主码约束不要显示命名。

- 思路解析

使用 check 关键词后跟括弧引出条件表达式。后续 DEFAULT 和 UNIQUE 约束与此相似，不再赘述。

```
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)
);
```

代码块 2.1.4-1 实验 1.4 代码

### 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）

本小节共有四个关卡，主要围绕数据库中表的基本修改操作展开，主要内容为修改表名、添加与删除字段、修改字段和添加删除修改约束。

### 2.2.1 修改表名

- 任务要求：将表名 your_table 更改为 my_table。
- 思路解析

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

代码 2.2.1-1 实验 2_1 代码

### 2.2.2 添加与删除字段

- 任务要求：具体要求见下代码注释。
- 思路解析

```
#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);
```

代码 2.2.2-1 实验 2_2 代码

### 2.2.3 修改字段

- 任务要求

对表 addressBook 作以下修改：1.将 QQ 号的数据类型改为 char(12)；2.将列名 weixin 改为 wechat。

- 思路解析

MODIFY [COLUMN] 列名 数据类型 [列约束] 和 RENAME COLUMN 列名 TO 新列名

```
alter table addressBook modify QQ char(12);
alter table addressBook rename column weixin to wechat;
```

代码 2.2.3-1 实验 2_3 代码

### 2.2.3 添加、删除与修改约束

- 任务要求

添加、删除与修改约束，具体要求见注释。

- 思路解析

```
#(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);
```

代码 2.2.4-1 实验 2_4 代码

## 2.3 基于金融应用的数据查询（Select）之一

本小节共有 19 个关卡，主要内容为金融应用系统中 select 语句在不同场景下的使用，包括但不限于资产查询与客户筛选、客户信息查询、金融产品查询、用户排序和资产计算等。

### 2.3.1 查询客户主要信息

- 任务要求

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。

- 思路解析

```
SELECT c_name, c_phone, c_mail
FROM client
ORDER BY c_id;
```

代码 2.3.1-1 实验 3_1 代码

### 2.3.2 查询客户主要信息

- 任务要求

查询客户表(client)中没有填写邮箱的客户的编号、名称、身份证号、手机号。

- 思路解析

使用 null 值是否相等来判断。

```
SELECT c_id, c_name, c_id_card, c_phone
FROM client
WHERE c_mail is null;
```

代码 2.3.2-1 实验 3_2 代码

### 2.3.3 既买了保险又买了基金的客户

- 任务要求

查询既买了保险又买了基金的客户的名称和邮箱。

- 思路解析

使用嵌套查询，先筛选出买了保险的客户，再从这些客户里寻找买了基金的客户，最后列出这些客户的 c_name, c_mail, c_phone。一共是三层 select 语句。

```
SELECT c_name, c_mail, c_phone
FROM client
WHERE c_id in
      (SELECT pro_c_id
       FROM property
       WHERE pro_c_id in
           (SELECT pro_c_id
            FROM property
            WHERE pro_type = 3) and pro_type = 2)
ORDER BY c_id;
```

代码 2.3.3-1 实验 3_3 代码

### 2.3.4 办理了储蓄卡的客户信息

该关卡任务已完成，实施情况本报告略过。

### 2.3.5 每份金额再 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

- 任务要求

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。

- 思路解析

先使用嵌套查询，在内层循环内使用 count 统计出所有商品收益的数目；再在外层循环使用 count 计数，若该数目大于或者等于所有商品收益的数目，则就为题目所求的商品收益的众数。

```
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);
```

代码 2.3.6-1 实验 3_6 代码

### 2.3.7 未购买任何理财产品的武汉居民

- 任务要求

查询未购买任何理财产品的武汉居民的信息。

- 思路解析

先在内层循环，筛选出买过理财产品的客户；再在外层使用 not exists 求反选出没有购买过的客户，同时使用 like 关键词选出身份证号开头为 4201（武汉客户身份证号格式）的客户；最后按 c_id 排序。

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

代码 2.3.7-1 实验 3_7 代码

### 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

- 任务要求

查询投资总收益前三名的客户。

- 思路解析

先使用 inner join 关键词，通过 pro_c_id 和 c_id 将表 client 和 property 内连接，然后按照 c_id 分组统计再将总收益（这里使用 sum 关键词求和）降序（desc）排序，最后使用 limit3 输出前三名用户。

```
select c_name, c_id_card, sum(pro_income)as total_income
from client inner join property on pro_c_id=c_id and pro_status="可用"
group by c_id
order by total_income desc
limit 3;
```

代码 2.3.10-1 实验 3_10 代码

### 2.3.11 黄姓客户持卡数量

- 任务要求

查询黄姓用户的编号、名称、办理的银行卡的数量。

- 思路解析

使用 left join，通过 c_id 和 b_c_id 以 client 为基础将表 client 和 bank_card 外连接，然后使用 like 关键词筛选出黄姓开头的客户，按 c_id 分组统计（使用 count 关键词求数目并重命名为 number_of_cards），最后按照银行卡的数目降序输出（desc）。

```
select c_id, c_name, count(b_c_id) as number_of_cards
from client left join bank_card on c_id=b_c_id
where c_name like "黄%"
group by c_id
order by number_of_cards desc, c_id;
```

代码 2.3.11-1 实验 3_11 代码

### 2.3.12 客户理财、保险与基金投资总额

该关卡任务已完成，实施情况本报告略过。

### 2.3.13 客户总资产

- 任务要求：查询客户在本行的总资产。

- 思路解析

先单独各自使用 select 语句计算出每个客户的每种资产总和，利用 pro_pif_id

和 p_id 进行连接，利用 pro_type 属性对种类进行筛选；再使用 union all 对以上的结果取并集。

理财产品、保险表、基金表的计算方法都是一样的，使用数目×单额的方法计算资产收入；同时再加上投资总收益；最后加上储蓄卡余额，减去信用卡透支的数目。

```
select pro_c_id, pro_quantity*p_amount as amount
from property, finances_product
where pro_pif_id=p_id and pro_type=1
union all

select pro_c_id, pro_quantity*i_amount as amount
from property, insurance
where pro_pif_id=i_id and pro_type=2
union all

select pro_c_id, pro_quantity*f_amount as amount
from property, fund
where pro_pif_id=f_id and pro_type=3
union all

select pro_c_id, sum(pro_income) as amount
from property
group by pro_c_id
union all

select b_c_id, sum(if(b_type='储蓄卡', b_balance, -b_balance)) as amount
from bank_card
group by b_c_id
```

代码 2.3.13-1 实验 3_13 分组再取并集代码

内层筛选好之后，将其与 client 表根据 c_id 和资产表的 pro_c_id 属性做自然连接，按 c_id 分组，最后按总资产量降序排序。

```
select c_id, c_name, ifnull(sum(amount), 0) as total_property
from client
left join (
    ① 代码 2.3.13-1
) pro on c_id=pro.pro_c_id
```

```
group by c_id
order by c_id
```

代码 2.3.13-2 实验 3_13 总体代码框架

### 2.3.14 第 N 高问题

该关卡任务已完成，实施情况本报告略过。

### 2.3.15 基金收益两种方式排名

- 任务要求

对客户基金投资收益实现两种方式的排名次：排名不连续和排名连续。

- 思路解析

两种方法其他都一样，只不过调用的函数不一样。若要进行名次不连续的排名则调用 rank()，而进行名次连续的排名就要调用 dense_rank()。代码如下所示，高亮处为不同之处。

```
select pro_c_id, sum(pro_income) as total_revenue,
    rank() over(order by sum(pro_income) desc) as 'rank'
from property
where pro_type = 3
group by pro_c_id
order by total_revenue desc, pro_c_id;
```

代码 2.3.15-1 实验 3_15 名次不连续排序代码（rank）

```
select pro_c_id, sum(pro_income) as total_revenue,
    dense_rank() over(order by sum(pro_income) desc) as 'rank'
from property
where pro_type = 3
group by pro_c_id
order by total_revenue desc, pro_c_id;
```

代码 2.3.15-2 实验 3_15 名次连续排序代码（dense_rank）

### 2.3.16 持有完全相同基金组合的客户

- 任务要求：查询持有完全相同基金组合的客户。
- 思路解析

由于客户与客户之间购买的基金有重复，而且在查询的时候都会使用到基金表的内容，所以要先使用 with as 创建一个公用临时表，同时在 as 里的 select 语句中使用 group_concat()函数将客户购买的基金按照基金 f_id 去重（在资产表里是

pro_pif_id）并筛选出来。

而在外层只要通过比较两个两个客户的 f_id 是否相同来判断他们购买的基金组合是否相同即可。

```
with pro(c_id, f_id) as (
    select pro_c_id c_id, group_concat(distinct pro_pif_id order by pro_pif_id) f_id
    from property
    where pro_type = 3
    group by pro_c_id
)
select p1.c_id c_id1, p2.c_id c_id2
from pro p1, pro p2
where p1.c_id<p2.c_id and p1.f_id=p2.f_id;
```

代码 2.3.16-1 实验 3_16 代码

### 2.3.17 购买基金的高峰期

- 任务要求

查询 2022 年 2 月购买基金的高峰期，如果连续三个交易日，投资者购买基金的总金额超过 100 万，则称这连续的几日为投资者购买基金的高峰期。

- 思路解析

最核心的思想是使用 datediff(pro_purchase_time, "2021-12-31")来查询待求交易日位于 2022 年所有天数里的次序，再通过 week()函数来求得此天处于当年的哪一周。在不是节假日的前提下，二者相减即可求得该天到底是第几个交易日。

以此为基础排掉 2022 年 2 月中的非基金交易后，筛选出当天交易总金额超过 100 万的后，调用 row_number()函数获取其行号。而若交易日次序减去行号相同，则说明它们就处于同一个连续段之中；若处于同一个连续段，则再调用 count()函数获取该连续段的长度，判断该连续段的长度，若大于等于 3，则根据题目要求定义，是 2022 年 2 月购买基金的高峰期。

```
select day3.time as pro_purchase_time, day3.amount as total_amount
from (
    select *, count(*) over(partition by day2.workday-day2.rownum) cnt
    from (
        select *, row_number() over (order by workday) rownum
        from (
            select pro_purchase_time time,
```

```
                    sum(pro_quantity*f_amount) as amount,
                @row := datediff(pro_purchase_time,
                    "2021-12-31")-2*week(pro_purchase_time) workday
                from property, fund, (select @row) aa
                where pro_purchase_time
                    like "2022-02-%"
                    and pro_type=3
                    and pro_pif_id=f_id
                group by pro_purchase_time
            ) day1
            where amount > 1000000
        ) day2
) day3
where day3.cnt >= 3;
```

代码 2.3.17-1 实验 3_17 代码

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

该关卡任务已完成，实施情况本报告略过。

### 2.3.19 以日历表格式显示每日基金购买总金额

- 任务要求

以日历表格式显示 2022 年 2 月每周每日基金购买总金额。

- 思路解析

总体框架是使用两层嵌套进行查询。

在内层使用 join 通过 pro_pif_id 和 f_id 将 property 表和 fund 表自然连接，按照购买日期使用 sum()函数按照数目×单价的公式求出当日基金购买的总金额。再使用 weeek()函数和 weekday()函数求得当日所处的周在当年的次序和当年在本周所处的位置（命名为 dayId）。于外层循环中按照周次序进行分组，将内层中的 dayId 归类（0 为 Monday，1 为 Tuesday…4 为 Friday），再加入该列进行求和计算，如代码 2.3.19-1 所示。

```
    sum(if(dayId = 0, amount, null)) Monday,
    sum(if(dayId = 1, amount, null)) Tuesday,
    sum(if(dayId = 2, amount, null)) Wednesday,
    sum(if(dayId = 3, amount, null)) Thursday,
    sum(if(dayId = 4, amount, null)) Friday
```

代码 2.3.19-1 实验 3_19 归类和求和计算

进行合并并筛选出对应的时间和投资类型（pro_type=3 基金）组装后，代码总体如 2.3.19-2 所示。

```
select wk week_of_trading,
      ① 代码 2.3.19-1
from (
      select week(pro_purchase_time)-5 wk,
         weekday(pro_purchase_time) dayId,
         sum(pro_quantity*f_amount) amount
      from property
            join fund on pro_pif_id = f_id
      where pro_purchase_time like "2022-02-%" and pro_type = 3
      group by pro_purchase_time
) tt
group by wk;
```

代码 2.3.19-2 实验 3_19 代码

## 2.4 基于金融应用的数据查询（Select）之二

本小节共有 6 个关卡，主要内容是对上一节 2.3 的补充练习：select 查询语句在不同场景下的使用，本节主要针对的是查询产品及用户的相似度，探索用户偏好和产品的受欢迎程度。

### 2.4.1 查询销售总额前三的理财产品

- 任务要求

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

- 思路解析

分三层嵌套查询。最内层使用 sum()函数求出 2010 年和 2011 年的销售总额，年份使用 year()函数进行求取，按年份分组计算得到销售总额。然后在中层查询中使用 rank()函数对内层求得的销售总额进行排序，并将排名值命名为 rk 以便最外层使用。最外层则根据 rk 筛选出 rk<4 即排名前三的理财产品。

注意本例使用的是 rank()函数而不是 dense_rank()函数，这是因为根据题目要求应进行名次不连续的排序。

```
select pyear, rk, p_id, sumamount
from (
```

```
    select pyear,
        rank() over(partition by pyear order by sumamount desc) as rk,
        p_id, sumamount
    from (
        select year(pro_purchase_time) as pyear, p_id,
            sum(pro_quantity * p_amount) as sumamount
        from property join finances_product on p_id = pro_pif_id
        where pro_type = 1 and (year(pro_purchase_time) = 2010
            or year(pro_purchase_time) = 2011)
        group by pyear, p_id
    ) as t
) as tt
where rk < 4
order by pyear, rk, p_id;
```

代码 2.4.1-1 实验 4_1 代码

### 2.4.2 投资积极且偏好理财类产品的客户

- 任务要求

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

- 思路解析

先创建公用表 a_id 和 b_id：

① a_id 中使用 count()对每个客户投资理财产品的种数进行计数，数目记作 cnta，筛选出投资理财产品种类数目大于 3 的客户；

② b_id 中也使用 count()函数每个客户购买的基金种树进行计数，数目基座 cntb。

```
with a_id as (
    select pro_c_id, count(*) as cnta
    from property
    where pro_type = 1
    group by pro_c_id having cnta >= 3
), b_id as (
    select pro_c_id, count(*) as cntb
```

```
    from property
    where pro_type = 3
    group by pro_c_id)
```

代码 2.4.2-1 实验 4_2 公用表创建

创建完毕后，根据 pro_c_id 相等时比较两者的 cnta 和 cntb，若前者大于后者，即同一个客户（pro_c_id）购买的理财产品数大于购买的基金购买数目，由此筛选出符合条件的客户。

```
select a_id.pro_c_id
from a_id
join b_id on a_id.pro_c_id = b_id.pro_c_id and cnta > cntb;
```

代码 2.4.2-2 实验 4_2 查询核心代码

### 2.4.3 查询购买了所有畅销理财产品的客户

该关卡任务已完成，实施情况本报告略过。

### 2.4.4 查找相似的理财产品

该任务关卡跳过。

### 2.4.5 查询任意两个客户的相同理财产品数

该关卡任务已完成，实施情况本报告略过。

### 2.4.6 查找相似的理财客户

- 任务要求

查询每位客户(列名：pac)的相似度排名值小于 3 的相似客户(列名：pbc)列表，以及该每位客户和他的每位相似客户的共同持有的理财产品数(列名：common)、相似度排名值(列名：crank)。

- 思路解析

首先创建公用表 fin_p，即所有购买了理财产品的客户。然后将分组排名比较，分别记作 a 和 b，比较两者购买理财产品的编号和两用户的 c_id；若相等即不为同一个用户购买的同一种产品，若二人购买的理财产品相同，则对此进行自然连接，对二者购买的相同理财产品数进行 count()计数。

同时再调用 rank()函数对以上得到的 count()结果，记排名次序为 crank()；最后通过 where 关键词限制输出相似度排名小于 3 的相似客户。

具体代码如下所示。

```
with fin_p as (
```

```
    select *
    from property
    where pro_type = 1)
select pac, pbc, cnt as common, crank
from(
    select a.pro_c_id as pac, b.pro_c_id as pbc, count(*) as cnt,
        rank() over(partition by a.pro_c_id order by count(*) desc,
            b.pro_c_id) as crank
    from fin_p as a join fin_p as b
        on a.pro_pif_id = b.pro_pif_id and a.pro_c_id != b.pro_c_id
    group by pac, pbc order by a.pro_c_id) as t
where crank < 3 order by pac, crank;
```

代码 2.4.6-1 实验 4_6 代码

## 2.5 数据的插入、修改于删除（Insert, Update, Delete）

本小节共有 6 个关卡，主要内容是使用 insert、update、delete 语句实现对表数据的插入、修改于删除等操作。

### 2.5.1 插入多条完整的客户信息

- 任务要求：向客户表 client 插入数据。
- 思路解析

使用“insert into + 表名 + values”的格式插入数据，一条语句插入三条数据。

```
insert into client 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");
```

代码 2.5.1-1 实验 5_1 代码

### 2.5.2 插入不完整的客户信息

- 任务要求：向客户表 client 插入一条数据不全的记录。
- 思路解析

将 values 字段前加入对应待插入的列名即可。

```
insert into
```

```
    client (c_id, c_name, c_phone, c_id_card, c_password)
        values (33, "蔡依婷", "18820762130",
            "350972199204227621", "MKwEuc1sc6");
```

代码 2.5.2-1 实验 5_2 代码

### 2.5.3 批量插入数据

- 任务要求

将 new_client 表的全部客户信息插入到客户表(client)。

- 思路解析

将 insert 语句中的 values 字段替换为 select 开头的查询语句。

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

代码 2.5.3-1 实验 5_3 代码

### 2.5.4 删除没有银行卡的客户信息

- 任务要求

请用一条 SQL 语句删除 client 表中没有银行卡的客户信息。

- 思路解析

用 not exists 查询没有办银行卡的客户，再使用 delete 关键字删除。

```
delete from client
where not exists (
    select *
    from bank_card
    where client.c_id=bank_card.b_c_id);
```

代码 2.5.4-1 实验 5_4 代码

### 2.5.5 冻结客户资产

- 任务要求

请用一条 update 语句将手机号码为“13686431238”这位客户的投资资产(理财、保险与基金)的状态置为“冻结”。

- 思路解析

使用 select 语句按电话号码筛选出该用户后使用 update 语句即可，并使用 set 设置 pro_status 即可。

```
update property
set pro_status = '冻结'
where exists (
```

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

代码 2.5.5-1 实验 5_5 代码

### 2.5.6 连接更新

- 任务要求

请用一条 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)。

- 思路解析

```
update property, client
set property.pro_id_card=client.c_id_card
where property.pro_c_id=client.c_id;
```

代码 2.5.6-1 实验 5_6 代码

## 2.6 视图

本小节共有 6 个关卡，主要内容是视图的创建与使用。

### 2.6.1 创建所有保险资产的详细记录视图

- 任务要求：创建所有保险资产的详细记录视图。
- 思路解析

使用“create view xxx as”来创建视图，剩下的只需要通过 select 语句来查询和筛选即可。

```
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 c_id = pro_c_id and pro_pif_id=i_id and pro_type=2
```

代码 2.6.1-1 实验 6_1 代码

### 2.6.2 基于视图的查询

- 任务要求

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。

- 思路解析

视图的基本操作和表的基本操作是一样的，同样使用 select 语句来查询，所有格式均相同，只不过在 from 字段指定 v_insurance_detail 即可。

```
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_id_card order by insurance_total_amount desc;
```

代码 2.6.2-1 实验 6_2 代码

## 2.7 存储过程与事务

本小节共有 3 个关卡，主要内容是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

- 任务要求

创建一个存储过程，向表 fibonacci 插入斐波拉契数列的前 n 项。

- 思路解析

使用“with recursive … as”语句定义一个递归查询，第一项为特值 1，如下所示。当 id 小于 2 时为 1，否则则为这一次的值 cur 加上上一次的值 pre。

```
with recursive cte (id, cur, pre) as (
            select 0, 0, 0
            union all
            select id+1, if(id<2, 1, cur+pre), cur from cte where id<m)
```

代码 2.7.1-1 实验 7_1 递归查询代码

使用 create 创建事务 sp_fibonacci，所有代码如下。

```
delimiter $$
create procedure sp_fibonacci(in m int)
begin
      set m=m-1;
      ① 代码 2.7.1-1
      select id n, cur fibn from cte;
end $$
delimiter;
```

代码 2.7.1-2 实验 7_1 代码

### 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

该关卡任务已完成，实施情况本报告略过。

## 2.8 触发器

本小节共有 1 个关卡，主要内容是触发器的使用。

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

该任务关卡跳过。

## 2.9 用户自定义函数

本小节共有 1 个关卡，主要内容是在 select 语句中用户自定义函数的应用。

### 2.9.1 创建函数并在语句中使用它

- 任务要求

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。

- 思路解析

用 create function 语句创建函数定义，依据客户编号计算该客户所有储蓄卡的存款总额，函数名为 get_Records。中间使用 select 语句和 sum()函数来计算 b_balance 的总和，代码如下。

```
delimiter $$
create function get_deposit(client_id int)
returns numeric(10,2)
begin
    return (
        select sum(b_balance) from bank_card
        where b_type = "储蓄卡"
        group by b_c_id having b_c_id = client_id);
end$$
delimiter;
```

代码 2.9.1-1 实验 9_1 函数定义

再使用 select 嵌套查询存款总额再 100 万以上客户的身份证号、姓名和存款总额，在内层查询中调用上面定义的 get_deposits()函数求得该客户的存款总额，代

码如下。

```
select *
from (
      select c_id_card, c_name, get_deposit(c_id) total_deposit
      from client)
where total_deposit >= 1000000
order by total_deposit desc;
```

代码 2.9.1-2 实验 9_2 函数调用代码

## 2.10 安全性控制

本小节共有 2 个关卡，主要内容是用户和权限设置、角色和权限。

### 2.10.1 用户和权限

- 任务要求

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。

- 思路解析

```
#(1) 创建用户 tom 和 jerry，初始密码均为'123456';
create user tom identified by '123456';
create user jerry identified by '123456';
#(2) 授予用户 tom 查询客户的姓名，邮箱和电话的权限,且 tom 可转授权限;
grant select (c_mail, c_name, 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;
```

代码 2.10.1-1 实验 10_1 代码

### 2.10.2 用户、角色与权限

- 任务要求

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。

- 思路解析

```
# (1) 创建角色 client_manager 和 fund_manager;
create user client_manager;
create user 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;
```

代码 2.10.2-1 实验 10_2 代码

## 2.11 并发控制与事务的隔离级别

本小节共有 6 个关卡，主要内容是及数据库中并发控制与事务的隔离级别，包括隔离级别的设置，事务的开启、提交和回滚等，读脏、不可重复读、幻读场景处理等。

### 2.11.1 并发控制与事务的隔离级别

- 任务要求

设置事务的隔离级别

- 思路解析

将事务的隔离级别设置为 read uncommitted，用 rollback 回滚结束事务。

```
# 设置事务的隔离级别为 read uncommitted
set session transaction isolation level read uncommitted;
# 回滚事务：
rollback;
```

代码 2.11.1-1 实验 11_1 代码

### 2.11.2 读脏

- 任务要求

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。

- 思路解析

产生读脏的原因，是事务 t1 读取数据时，修改该数据的事务 t2 还没有结束(commit 或 roll back，统称 uncommitted),且 t1 读取的时间点又恰在 t2 修改该数据之后。最低的隔离级别不能避免读脏，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。

因此由上可得要发生“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 1 代码如下。

```
use testdb1;
set session transaction isolation level read uncommitted;
start transaction;
set @n = sleep(1);
select tickets from ticket where flight_no = 'CA8213';
commit;
```

代码 2.11.2-1 实验 11_2 事务 1 代码

事务 2 代码如下。

```
use testdb1;
set session transaction isolation level read uncommitted;
start transaction;
update ticket set tickets = tickets - 1 where flight_no = 'CA8213';
set @n = sleep(2);
rollback;
```

代码 2.11.2-2 实验 11_2 事务 2 代码

### 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 主动加锁保证可重读

该关卡任务已完成，实施情况本报告略过。

### 2.11.6 可串行化

- 任务要求

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。

- 思路解析

在事务 1 中添加代码休眠代码，让其一直休眠直到事务 2 执行完毕后再宠幸继续执行即可，事务 1 代码如下。

```
use testdb1;
start transaction;
set @n = sleep(5);
select tickets from ticket where flight_no = 'MU2455';
select tickets from ticket where flight_no = 'MU2455';
commit;
```

代码 2.11.6-1 实验 11_6 事务 1 代码

事务 2 代码如下所示。

```
use testdb1;
start transaction;
update ticket set tickets = tickets - 1 where flight_no = 'MU2455';
commit;
```

代码 2.11.6-2 实验 11_6 事务 2 代码

## 2.12 备份+日志：介质故障与数据库恢复

本小节共有 2 个关卡，主要内容是 MySQL 数据库的备份与恢复、介质故障发生时数据的恢复。

### 2.12.1 备份与恢复

- 任务要求：备份数据库，然后再恢复它。
- 思路解析

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
```

代码 2.12.1-1 实验 12_1 备份数据库代码

使用 mysql 指令根据备份文件 residents_bak.sql 还原数据库。

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

代码 2.12.1-2 实验 12_1 恢复数据库代码

### 2.12.2 介质故障的发生与数据库的恢复

- 任务要求

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。

- 思路解析

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;
```

代码 2.12.2-1 实验 12_2 备份时打开新日志文件

为了恢复数据库，保证两次发生的业务数据都不丢失，需利用 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;
```

代码 2.12.2-2 实验 12_2 恢复数据库代码

## 2.13 数据库设计与实现

本节一共 3 个关卡，主要内容为

### 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

该任务关卡跳过。

### 2.13.3 建模工具的使用

该任务关卡跳过。

### 2.13.4 制约因素分析与设计

在实际场景中，限制因素占据了很重要的一环。若考虑不全可能造成很大的失误。以本章节中的机票订票系统为例，一个系统除了使用者，还要有管理员，因此就应该为使用者设置权限，进行权限的隔离；否则普通使用者就可以对其他客户的航班进行修改，将会造成十分严重的后果。

### 2.13.5 工程师责任及其分析

作为一名工程师，在完成产品之前，要考虑多种因素，包括但不限于该产品对社会、健康、安全、法律以及文化的影响，确认该产品的可行性。而在完成的过程中，除了要做好产品之外，还应该站在使用者、用户的角度来设计和完成产品，一切从用户出发、为用户考虑，尽力完善作品。

## 2.14 数据库应用开发(JAVA 篇)

本关共有 7 个关卡，主要内容为 JDBC 体系结构和简单的查询、用户登录、添加新客户、银行卡销户、客户修改密码、事务与转账操作、把稀疏表格转为键值对存储等内容。

### 2.14.1 JDBC 体系结构和简单的查询

- 任务要求

正确使用 JDBC，查询金融应用场景数据库 finance 的 client 表(客户表)中邮箱不为空的客户信息，列出客户姓名，邮箱和电话.

- 思路解析

先使用 Class.forName() 加载驱动程序的类文件到内存中，再使用 DriverManager.getConnection(DB_URL, USER, PASS)进行连接的建立。

使用 connection 对象的 createStatement()方法创建一个 statement 实例并使用该实例的 executeQuery()方法来执行 select 语句，内容是 select c_name, c_mail, c_phone from client where c_mail is not null，即查询 client 表中非空的客户信息（题目所求）。

再使用 ResultSet executeQuery(String SQL)方法即 resultSet.next()输出相应的信息，核心代码如下。

```
try {
        Class.forName(JDBC_DRIVER);
        connection = DriverManager.getConnection(DB_URL, USER, PASS);
        statement = connection.createStatement();
        resultSet = statement.executeQuery("select c_name, c_mail,
              c_phone from client where c_mail is not null");
        System.out.println("姓名\t 邮箱\t\t\t\t 电话");
        while (resultSet.next()) {
              System.out.print(resultSet.getString("c_name") + "\t");
              System.out.print(resultSet.getString("c_mail") + "\t\t");
              System.out.println(resultSet.getString("c_phone"));
}
```

代码 2.14.1-1 实验 14_1 核心代码

### 2.14.2 用户登录

- 任务要求

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。

- 思路解析

先定义一个字符串，内容为 select 语句的内容。再使用 connection 类的

preparedStatement()方法来执行 SQL 语句，把 SQL 语句中变化的部分（账号和密码）当成参数，防御 SQL 注入攻击。

最后同上一关中一样，使用 ResultSet executeQuery(String SQL)方法即 resultSet.next()输出相应的信息，核心代码如下。

```
statement = connection.createStatement();
        String sql = "select * from client where c_mail = ? and c_password = ?";
        PreparedStatement ps = connection.prepareStatement(sql);
        ps.setString(1, loginName);
        ps.setString(2, loginPass);
        resultSet = ps.executeQuery();
        if (resultSet.next())
            System.out.println("登录成功。");
        else
            System.out.println("用户名或密码错误！");
```

代码 2.14.2-1 实验 14_2 核心代码

### 2.14.3 添加新用户

该关卡任务已完成，实施情况本报告略过。

### 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

- 任务要求

编写修改客户登录密码的的方法。

- 思路解析

具体调用的类方法同上前几关，包括 connection.prepareStatement()，connection.executeQuery()等。只不过新增加了判断语句，需要先判断用户是否存在，若存在则继续判断原密码是否正确。若正确，则可进行修改密码的操作，核心代码如下。

```
    public static int passwd(Connection connection, String mail,String password,
String newPass) {
        String sql = "select * from client where c_mail = ?";
        try {
            PreparedStatement ps = connection.prepareStatement(sql);
            ps.setString(1, mail);
            ResultSet res = ps.executeQuery();
```

```
                if (res.next()) {
                    if (password.equals(res.getString("c_password"))) {
                        sql = "update client set c_password = ? where c_mail = ?
                            and c_password = ?";
                        ps = connection.prepareStatement(sql);
                        ps.setString(1, newPass);
                        ps.setString(2, mail);
                        ps.setString(3, password);
                        ps.executeUpdate();
                        return 1;
                    } else    return 3;
                } else    return 2;
            } catch (SQLException e) {
                e.printStackTrace();    }
            return -1;
        }
```

代码 2.14.5-1 实验 14_5 核心代码

### 2.14.6 事务与转账操作

该关卡任务已完成，实施情况本报告略过。

### 2.14.7 把稀疏表格转为键值对存储

该任务关卡跳过。

## 2.15 数据库的索引 B+树实现

本节共有 5 个关卡，主要内容是 BPlusTreePage 的设计、BPlusTreeInternalPage 的设计、BPlusTreeLeafPage 的设计、B+树索引：Insert、B+树索引：Remove 等。

### 2.15.1 BPlusTreePage 的设计

该关卡任务已完成，实施情况本报告略过。

### 2.15.2 BPlusTreeInternalPage 的设计

该任务关卡跳过。

### 2.15.3 BPlusTreeLeafPage 的设计

该任务关卡跳过。

### 2.15.4 B+树索引：Insert

该任务关卡跳过。

### 2.15.5 B+树索引：Remove

该任务关卡跳过。

# 3 课程总结

本次课程我通过 15 个小节的各个关卡，巩固了许多数据库相关的知识，如数据库表的创建、修改、删除，视图的创建与使用，并发控制与事务的隔离级别、数据库的设计与实现和数据库应用开发等。

收获最大的应该是第三、四两节，这两节主要内容是在让我们使用 select 语句进行一些查询操作。在课上学习了一些 select 查询语句的相关知识，但只听课和看书本例子很难完全或者说较好地掌握这些内容，同时知识也只是停留在书本的层面。但通过本次实验，我进一步掌握了 select 语句与各种相关知识的运用，如自然连接、各种函数的使用，在真实的场景下将知识运用于实践，巩固了曾经所学并提高了相关水平，在不管是简单还是复杂场景下都能熟练运用 select 查询语句解决问题。

同时在第十一节并发控制与事物的隔离级别中，通过设置事务的不同隔离级别，达到了不同的效果。而在倒二节数据库应用的开发中，我感受到了数据库应用场景的多样性与鲁棒性。这次实验比较遗憾的是由于时间原因有些关卡没有完成，但仍然心怀感激地结束了本次实验报告的撰写。