# 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：2107 班

学　　号：U202115538

姓　　名：陈侠锟

指导教师：江胜

| 分数 | |
|---|---|
| 教师签名 | |

2023 年 6 月 14 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

本次实践课程在头歌平台进行，实践任务均在平台上提交代码，所有完成的任务、关卡均通过了自动测评。本次实践最终完成了任务书中的 2.1~2.12、2.14 子任务，部分完成了 2.13、2.15 子任务，下面将重点完成的子任务阐述其完成过程中的具体工作。

## 2.1 数据库、表与完整性约束的定义(Create)

本章要求了解数据库、表与完整性约束的定义方法，了解数据库语言的基本语法。

### 2.1.1 创建数据库

本关卡已完成，因较为简单故略过分析。

### 2.1.2 创建表及表的主码约束

本关卡已完成，因较为简单故略过分析。

### 2.1.3 创建外码约束（foreign key）

本关卡已完成，因较为简单故略过分析。

### 2.1.2 CHECK 约束

本关卡已完成，因较为简单故略过分析。

### 2.1.2 DEFAULT 约束

本关卡已完成，因较为简单故略过分析。

### 2.1.2 UNIQUE 约束

本关卡已完成，因较为简单故略过分析。

## 2.2 表结构与完整性约束的修改（ALTER）

本章学习 alter table 语句的大部分功能和修改表结构的方法（如添加列、约束，删除列、约束，修改列）等基础知识，了解表的完整性约束。

### 2.2.1 修改表名

本关卡已完成，因较为简单故略过分析。

### 2.2.2 添加与删除字段

本关卡已完成，因较为简单故略过分析。

### 2.2.3 修改字段

本关卡已完成，因较为简单故略过分析。

### 2.2.4 添加、删除与修改约束

本关卡已完成，因较为简单故略过分析。

## 2.3 数据查询（Select）之一

本章采用某银行的一个金融场景应用的模拟数据库，提供了六个表，分别为：client（客户表）、bank_card（银行卡）、finances_product（理财资产表）、insurance（保险表）、fund（基金表）、property（资产表）。

本实训中大多采用 select 语句来对表进行查询满足要求的各项数据，设计了子查询、数据库函数等知识点。

### 2.3.1 查询客户主要信息

本关卡已完成。

要求从 client 表中查询客户名称、手机号和邮箱信息，查询结果按照客户编号排序，仅利用 select、from、order by 即可实现，代码如下。

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

### 2.3.2 邮箱为 null 的客户

本关卡已完成。

要求从 client 表中查询邮箱信息为 null 的客户编号、名称、身份证号、手机号，仅利用 select、from、where 即可实现，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

本关卡已完成。

查询既买了保险又买了基金的客户名称、邮箱和电话，结果按照 c_id 排序，仅利用 select、from、where exists、order by 即可实现，代码如下。

```
SELECT c_name, c_mail, c_phone FROM client
WHERE EXISTS
```

```
(SELECT * FROM property WHERE pro_c_id = c_id AND pro_type = 2)
AND EXISTS
(SELECT * FROM property WHERE pro_c_id = c_id AND pro_type = 3)
ORDER BY c_id ASC;
```

### 2.3.4 办理了储蓄卡的客户信息

本关卡已完成。

查询办理了储蓄卡的客户名称、手机号、银行卡号，查询结果按照 c_id 排序，查找 b_type = ‘储蓄卡’的用户即可，代码如下。

```
select c_name, c_phone, b_number from client, bank_card
where client.c_id = bank_card.b_c_id and b_type = '储蓄卡' order by c_id;
```

### 2.3.5 每份金额在 30000~50000 之间的理财产品

本关卡已完成。

查询理财产品中每份金额在 30000~50000 之间的理财产品的相关信息，并按照金额升序排序，金额相同的按照理财年限降序排序，使用 where 满足金额范围的要求，order by...asc/desc 实现升序/降序排序，代码如下。

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

### 2.3.6 商品收益的众数

本关卡已完成。

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。本关需要用到 count 函数计算出现次数，用 having 语句实现的“众数”的要求，代码如下。

```
select pro_income, count(pro_income) 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

本关卡已完成。

查询身份证隶属武汉市且未购买任何理财产品的客户的相关信息。本关需要用到嵌套查询，先查所有购买了理财产品的客户，然后查隶属武汉市但不在第一

个集合中的客户，代码如下。

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

### 2.3.8 持有两张信用卡的用户

本关卡已完成。

查询持有两张（含）以上信用卡的用户的相关信息。本关同样使用嵌套查询，先查持有两张（含）以上信用卡的用户 id 和卡 type（需要用到 having count()和 group 指令），然后从 client 表里找属于第一个集合的用户，代码如下。

```
select c_name,c_id_card,c_phone from client where (c_id,"信用卡") in
(select b_c_id,b_type from bank_card group by b_c_id,b_type
having count(*) > 1) order by c_id;
```

### 2.3.9 购买了货币型基金的客户信息

本关卡已完成。

查询购买了货币型基金的用户的相关信息。本关需要使用两次嵌套查询，先找出 f_type = ‘货币型’的 f_id，再从 property 表中找出 pro_type = 3（意为基金）且 pro_pif_id 在第一个集合中的 pro_c_id，最后从 client 表中找到 c_id 在第二个集合中的用户。代码如下。

```
select c_name, c_phone, c_mail from client where c_id in
(select pro_c_id from property where pro_type = 3 and pro_pif_id in
(select f_id from fund where f_type = '货币型')) order by c_id;
```

### 2.3.10 投资总收益前三名的客户

本关卡已完成。

查询当前总可用资产收益（被冻结的资产除外）前三名的客户的相关信息，按收益降序输出。本关需要用到 sum 函数激素按总资产收益，用 limit 输出前三名，代码如下。

```
select c_name, c_id_card, sum(pro_income) as total_income
from client, property
where property.pro_c_id = client.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

本关卡已完成。

给出黄姓用户的相关信息，按银行卡数量降序，持卡数量相同则按客户编号排序。本关需要连接 client 和 bank_card 表，使用 count 函数计算办卡数量，使用 like 查找所有黄姓用户。代码如下。

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

### 2.3.12 客户理财、保险与基金投资总额

本关卡已完成。

综合客户表、资产表、理财产品表、保险表和基金表，列出客户相关信息并按要求排序。本关要求计算用户的投资总金额，由于资产分为三类（理财、保险、基金），故要用三个 select 语句去分别计算这三类资产的金额。之后，通过 union 指令将不同的 select 结果连接（注意连接的 select 结果必须拥有相同数量的列，列也必须拥有相似的数据类型，同时，每条 select 语句中的列的顺序必须相同）。

得到了资产金额表后，将其与 client 表连接，用 sum 函数计算总金额即可，但是计算时还要注意使用 ifnull 函数将空值转换成 0 值。

由于篇幅限制，本关代码不再完整给出。使用 ifnull 函数的语句如下，该句也是最外层的 select 语句。

```
select c_name, c_id_card, ifnull(sum(amount), 0) as total_amount
```

### 2.3.13 客户总资产

本关卡已完成。

综合所给的六个表列出客户信息。本关要求计算每个用户的总资产，总资产为储蓄卡余额、投资总额、投资总收益的和，再扣除信用卡透支金额（及信用卡余额），客户总资产包括冻结的资产。

首先计算投资总额，分别用 select 语句选出理财产品、保险和基金的总额，用 union all 互相连接，再用一个外层 select 语句求和得到投资总额。然后，在 property 表中用 sum(pro_income)计算投资总收益；bank_card 表中取得储蓄卡余

额和信用卡透支余额，将这四类资产余额表用 union all 连接构成一个总表 b(pro_c_id, property)，再用 sum(property)得到每个用户的总资产。

由于篇幅限制，本关代码将于附录中给出。

### 2.3.14 第 N 高问题

本关卡已完成。

查询每份保险金额第 4 高保险产品的编号和保险金额。本关只需使用一次嵌套循环即可完成，使用 limit N, M 指令，可以从第 N 条记录开始，返回 M 条记录，故在将 insurance 表中数据排序后很容易就能找到第 4 高的产品。代码如下。

```
select i_id, i_amount from insurance where i_amount =
(select distinct i_amount from insurance order by i_amount desc limit 3, 1);
```

### 2.3.15 基金收益两种方式排名

本关卡已完成。

查询资产表中客户编号，客户基金投资总收益及其排名（从高到低），总收益相同时名次亦相同。本关要求分别实现全局名次不连续的排名和连续的排名，即考察 rank() over 和 dense_rank() over 的区别，前者为名次不连续的排名，后者为连续的排名，两句关键代码如下。

```
select pro_c_id, total_revenue, RANK() OVER(order by total_revenue desc) as `rank`
select pro_c_id, total_revenue, DENSE_RANK() OVER(order by total_revenue desc)
as `rank`
```

### 2.3.16 持有完全相同基金组合的用户

本关卡已完成。

查询所有持有相同基金组合的用户对。本关的思路为：先将两个 property 表连接（命名为 first 和 second），连接条件为两表中的 pro_type 均等于 3 且两个表的 pro_pif_id 相等，这样可以将有相同基金的两个人连到一行里面，然后使用 select 语句查询 A、B 用户，按照 first 中的 c_id 和 second 中的 c_id 分组，使用 having 语句保证选出的行中，基金数与 A 和 B 持有的基金数均相等，即可保证 A 和 B 持有的基金完全相同。

由于篇幅限制，本关代码在附录中给出。

### 2.3.17 购买基金的高峰期

本关卡已完成。

查询 2022 年 2 余额购买基金的高峰期。至少连续三个交易日，所有投资者购买基金的总金额超过 100 万，则称这段连续交易日为投资者购买基金的高峰期。只有交易日才能购买基金，周六和周日是非交易日，2022 年春节假期之后的第一个交易日为 2 月 7 日。

首先进行 select 生成派生表，对每天的交易量进行计算(pro_quantity * f_amount)，并使用 sum 函数求和，标注为 total_amount，同时可根据 datediff 函数及相关计算求出该天是 2021 年第几个工作日（命名为 workday）。

之后外部的一层查询将交易量未达到一百万的记录排除，使用 row_number() 函数对按照 workday 排序后的数据生成从 1 开始的行号，命名为 rownum。我们注意到，在连续的若干交易日期中，如果交易量均超过一百万，那么 workday 和 rownum 均是连续的。因此通过计算 workday 和 rownum 的差值并按此进行分组，使用 count 函数计算每组的行数，结果大于等于 3 的那些组，就是购买基金的高峰期。

由于篇幅限制，本关代码不再给出。

### 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

本关卡已完成。

查询至少有一章信用卡余额超过 5000 元的客户编号，以及该客户持有的喜用卡总余额，总余额命名为 credit_card_amount。本关直接在 bank_card 表中将数据按照 b_c_id 分组，用 having max()找到最大信用卡余额，大于 5000 的将被 select 出。由于篇幅限制，本关代码不再给出。

### 2.3.19 以日历表格式显示每日基金购买总金额

本关卡已完成。

以日历表格式列出 2022 年 2 余额每周每日基金购买总金额。本关需要用一次嵌套循环，先将 property 表与 fund 表连接，连接条件为 pro_pif_id = f_id，从中挑出 2022 年 2 月的数据（使用 extract 函数实现），按照购买时间 pro_purchase_time 分组，使用 sum 计算每一交易日期的总交易金额（定义为 amount），并用 week 函数计算出该日期是 2 月的第几周（定义为 wk），用 weekday 函数算出是星期几（定义为 dayId）。之后用外层的 select 算出所有周一至周五

的基金购买总金额，此步骤关键代码如下。

```
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
```

由于篇幅限制，本关完整代码不再给出。

## 2.4 数据查询（Select）之二

本章要求与 2.3 完全相同，是在此基础上的难度提升。

### 2.4.1 查询销售总额前三的理财产品

本关卡已完成。

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，如 1、1、3）的相关信息。本关卡需要使用一次嵌套循环，内部的 select 查询出年份在 2010 和 2011 年中的数据（具体是哪一年定义为 pyear），按 p_id 和 pyear 分组，用 sum 函数计算出销售总额（p_amount * pro_quantity），定义为 sumamount。外层的 select 使用 rank() over 函数对选出的记录进行排名（以 pyear 分组，sumamount 为排序依据）。关键代码如下。

```
select pyear, rank() over(partition by pyear order by sumamount desc) as rk, p_id,
sumamount
```

由于篇幅限制，本关完整代码不再给出。

### 2.4.2 投资积极且偏好理财类产品的客户

本关卡已完成。

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。本关使用两个子查询先分别计算用户购买理财产品的数量（cnt1）和购买基金的数量（cnt2），定义为两个表 t1 和 t2，将两个表按照 pro_c_id 连接，使用外层的 select 选出 t1,cnt1 > t2.cnt2 且 t1.cnt1 >= 3 的数据即可。

由于篇幅限制，本关代码不再给出。

### 2.4.3 查询购买了所有畅销理财产品的客户

本关卡已完成。

若定义持有人数超过 2 的理财产品称为畅销理财产品。查询购买了所有畅销理财产品的客户编号（注意去重）。本关使用了嵌套了两层 where not exists 的查询，首先生成畅销理财产品的表与被用户购买了的理财产品的表。内层的 where not exists 找出那些没有被购买的畅销理财产品，（如果有）再利用外层的 where not exists 查找没有漏掉购买任何一个畅销理财产品的人（即购买了所有畅销理财产品的人）。

此关用语言解释比较绕，故代码在附录中给出。

### 2.4.4 查找相似的理财产品

本关卡已完成。

本关需要明确“相似的理财产品”的定义（比较复杂故不再赘述），使用多重嵌套查询进行查找。首先从 property 表中找到全体持有 14 号理财产品的客户以及每个客户的持有数量，定义为 t1 表；使用 dense_rank() over 函数将其中的数据按持有数量排序，定义为 t2 表；从 t2 表中选出持有 14 号理财产品数量最多的前三名（但可能不止三个人），定义为 t3 表；t3 表与 property 进行自然连接后，从中选出前三名持有的理财产品（除 14 号以外），定义为 t4 表；t4 表与 property 再进行自然连接，数据按 pro_pif_id 分组，按 pro_pif_id 排序，选出 pro_pif_id 及该编号的理财产品的购买客户总人数 cc，定义为 t5 表；从 t5 表中查找 pro_pif_id，cc，以及用 dense_rank() over 函数生成的排名。

由于篇幅限制，本关代码不再给出。

### 2.4.5 查询任意两个客户的相同理财产品数

本关卡已完成。

查询任意两个客户之间持有的相同理财产品种数，且结果仅保留相同理财产品数至少 2 种的用户对。本关使用两个 property（命名为 p1 和 p2）进行自然连接，按 p1.pro_c_id，p2.pro_c_id 进行分组，使用 count 函数计算两人持有的相同理财产品的数量，使用 having count 满足“至少 2 种”的要求。

由于篇幅限制，本关代码不再给出。

### 2.4.6 查找相似的理财客户

本关卡已完成。

本关卡需要明确“相似的理财客户”的定义（比较复杂故不再赘述），先将 property 表定义为 t1，将从 property 中 select 出的 pro_type = 1 的用户 id 和业务 id 定义为 t2 表，t1 和 t2 表进行连接后，表的每一行就有了（A 客户，B 客户，他们共同拥有的产品 id）这样的数据（但注意此时 A 客户和 B 客户可能是同一个人）之后从连接的表中 select 出不同的 A 客户作为 pac，不同的 B 客户作为 pbc，用 count 计算他们共同持有的理财产品数作为 common，将这些内容定义为 t3 表；使用 rank() over 函数按照 pac 分组，按 common 降序，pbc 升序的方式进行排名，作为 t4 表；从 t4 表中选出排名值小于 3 的相似客户即可。

由于篇幅限制，本关代码不再给出。

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

本章主要对表进行修改，包括插入、修改、删除等操作，需要直接对数据库中的表进行修改得到一个新的表。以上操作需要用到 insert/update/delete 指令，本章即针对这三条指令的用法进行实验。

### 2.5.1 插入多条完整的客户信息

本关卡已完成，因较为简单故略过分析。

### 2.5.2 插入不完整的客户信息

本关卡已完成，因较为简单故略过分析。

### 2.5.3 批量插入数据

本关卡已完成，因较为简单故略过分析。

### 2.5.4 删除没有银行卡的客户信息

本关卡已完成，因较为简单故略过分析。

### 2.5.5 冻结客户资产

本关卡已完成，因较为简单故略过分析。

### 2.5.6 连接更新

本关卡已完成，因较为简单故略过分析。

## 2.6 视图

视图（view）是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，数据库中只存放了视图的定义，而并没有存放视图中的数据，这些数据存放在原来的表中。使用视图查询时，数据库会从原来的表中取出对应的数据。本章要求学会用 create 创建视图，以及利用视图查找表中数据。

### 2.6.1 创建所有保险资产的详细记录视图

本关卡已完成，因较为简单故略过分析。

### 2.6.2 基于视图的查询

本关卡已完成，因较为简单故略过分析。

## 2.7 存储过程与事务

存储过程是在大型数据库系统中，一组为了完成特定功能的 SQL 语句集，存储在数据库中，其经过第一次编译后再次调用不需要再次编译，用户通过指定存储过程的名字并给出参数（如果有）来调用存储过程。本章要求学会使用流程控制语句/游标/事务的存储过程。

### 2.7.1 使用流程控制语句的存储过程

本关卡已完成。

首先使用 delimiter 重定义 mysql 分隔符，使用 create procedure 创建存储过程，在函数体中，先用 declare 定义相关参数，使用流程控制语句（有 begin...end / if-else / while 等）设计插入斐波那契数列的功能，对前两项单独判断，第三项及以后的值为其前面两项之和。

由于篇幅限制，本关代码不再给出。

### 2.7.2 使用游标的存储过程

本关卡已完成。

本关需要用 declare 定义游标，游标相当于一个存储于内存的带有指针的表，每次可以存取指针指向的一行数据，并将指针向前推进一行。本关要求针对医院的夜班安排一个值班表，每个夜班安排一名医生及两名护士，限制要求是主任（属于医生）不能值周末的夜班，如果按照正常的工号顺序应该由某主任值周末夜班，

此夜班延迟到周一上，设计存储过程时需注意这一点。

首先我创立了两个游标 cur1 和 cur2，前者用于找一个护士，后者用于找一个医生（含主任），创建 head 代表需要调班的主任（如果没有则为 null），nur1，nur2，doc 分别为值班的护士 1，护士 2 和医生。然后开始遍历所有需要排班的日期，对于每一个日期，先找到两个护士，然后使用 weekday 函数获取当天是星期几，对医生进行如下判断：

若当天是周一且有之前调班的主任（即 head is not null），那么将设置 doc = head， head = null；

否则，先利用游标 cur2 找到一个医生，然后：如果当天为周末且找到的这个医生为主任，那么设置 head = doc，再重新利用 cur2 找一个医生（由于一个科室只有一个主任，所以找到的第二位医生一定不是主任）；否则，就让找到的这个医生值班。

每找完一组护士和医生（即当天的排班完成），就用 insert into...values...语句将排班结果插入到排班表中，然后利用 date_add 函数将日期加 1。

本关中还要注意，由于游标只能前进不能后退，因此每次使用游标都要进行判断游标是否已经到达结尾，若到达结尾则需要 close 后重新 open 游标。

由于篇幅限制，本关代码于附录中给出。

### 2.7.3 使用事务的存储过程

本关卡已完成。

在金融应用场景数据库中，编程实现一个转账操作的存储过程 sp_transfer_balance，实现从一个账户向另一个账户转账。转账操作涉及对表 bank_card 的操作，需要满足以下要求：①仅当转款人是转出卡的持有人时，才可转出；②仅当收款人是收款卡的持有人时，才可转入；③储蓄卡之间可以相互转账；④允许储蓄卡向信用卡转账，成为信用卡还款（允许替他人还款），还款可以超过信用卡余额，此时，信用卡余额为负数；⑤信用卡不能向储蓄卡转账；⑥转账金额不能超过储蓄卡余额。

本关设计思路是三层 if 语句嵌套，第一层 if 保证转款人是转出卡的持有人，收款人是收款卡的持有人，且转出卡是储蓄卡，否则设置 return_code = 0；第二层 if 保证转账金额未超过储蓄卡余额，否则设置 return_code = 0；第三层使用

if-else 判断收款卡是储蓄卡还是信用卡，以便进行不同的操作，之后设置 return_code = 1，代表转账成功.

由于篇幅限制，本关代码不再给出。

## 2.8 触发器

触发器（trigger）是由事件来触发某个操作。这些事件包括 insert 语句、update 语句和 delete 语句。当数据库系统执行这些事件时，就会激活触发器执行相应的操作。本章要求为指定的表编写触发器以实现特定的业务规则。

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

本关卡已完成。

使用条件语句来判断使用的是哪一种业务类型，如果 pro_type = 1，使用理财业务，则 pro_pif_id 只能引用 finances_product 表的 p_id；如果 pro_type = 2，办理保险业务，则 pro_pif_id 只能引用 insurance 表的 i_id；如果 pro_type = 3，使用基金业务，则 pro_pif_id 只能引用 fund 表的 f_id；同时，pro_type 不接受 123 以外的值，这时应该捕获异常并进行提示输入非法。触发器业务结束后返回新的元组即可。

由于篇幅限制，本关代码不再给出。

## 2.9 用户自定义函数

在 mysql 中，用户可以自定义一些函数方便对数据进行相关操作。函数与存储过程有类似之处，都是持久性存储模块，定义方法也类似，不同之处是函数必须指定返回的类型。

### 2.9.1 创建函数并在语句中使用它

本关卡已完成。

本关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，即关使用 create function 语句创建名称为 get_deposit 的函数，该函数可以根据客户编号计算该客户所有储蓄卡余额的总和。函数参数为客户编号，返回值为该客户储蓄卡余额总和，之后通过 select 语句调用该函数可查询存款总额在 100 万以上的客户的身份证号，姓名和存储总额。

由于篇幅限制，本关代码不再给出。

## 2.10 安全性控制

本章的主要内容是数据库安全性控制。数据库的安全性是指保护数据库以防止不合法使用所造成的数据泄露、更改或破坏。大型数据库管理系统都支持自主存取控制，SQL 标准也对自主存取控制提供支持，这主要通过 SQL 的 GRANT 语句和 REVOKE 语句来实现。

### 2.10.1 用户和权限

本关卡已完成，因较为简单故略过分析。

### 2.10.2 用户、角色与权限

本关卡已完成，因较为简单故略过分析。

## 2.11 并发控制与事务的隔离级别

数据库是共享资源，允许多个用户同时访问同一数据库。但并发操作不加控制，便会产生数据的不一致性。并发操作可能带来的数据不一致性包括丢失修改、读脏数据、不可重复读、幻读。为解决上述不一致性问题，DBMS 设计了专门的并发控制子系统，采用封锁机制进行并发控制，以保证事务的隔离性和一致性。但事务的隔离程度越高，并发度就会越低，很多时候，需要在一致性和并发度间进行取舍，从而产生了事务的隔离级别的概念。

### 2.11.1 并发控制与事务的隔离级别

本关卡已完成，因较为简单故略过分析。

### 2.11.2 读脏

本关卡已完成。

两个事务的隔离级别应设置为 read uncommitted，使用 sleep()函数使得 t1 在读到 t2 修改的数据后，t2 撤销其修改，从而构造读脏现象。

由于篇幅限制，本关代码不再给出。

### 2.11.3 不可重复读

本关卡已完成。

t1 事务隔离级别设置为 read uncommitted，t2 事务隔离级别设置为 read committed，使用 sleep()函数使得 t2 读取数据之后，t1 修改了其读取的数据，从而构造不可重复读现象。

由于篇幅限制，本关代码不再给出。

### 2.11.4 幻读

本关卡已完成。

使用 sleep()函数使得 t1 事务在查询一次数据之后，t2 事务对数据进行插入操作，从而导致 t1 事务出现幻读现象。

由于篇幅限制，本关代码不再给出。

### 2.11.5 主动加锁保证可重复读

本关卡已完成，因较为简单故略过分析。

### 2.11.6 可串行化

本关卡已完成，因较为简单故略过分析。

## 2.12 备份+日志：介质故障与数据库恢复

在数据库中，数据有可能发生损坏，因此需要设计应对方法也恢复数据。头歌平台的 mysql 提供了 mysqldump 等工具，利用备份、日志文件实现恢复。

### 2.12.1 备份与恢复

本关卡已完成。

使用 mysqldump 进行逻辑备份，并用逻辑备份文件恢复数据库，代码如下。

```
mysqldump -h127.0.0.1 -uroot --databases residents > residents_bak.sql
mysql -h127.0.0.1 -uroot < residents_bak.sql
```

### 2.12.2 备份+日志：介质故障的发生与数据库的恢复

本关卡已完成。

使用 mysqldump 进行逻辑备份，并用逻辑备份文件恢复数据库，使用 mysqlbinlog 使用日志，代码如下。

```
mysqldump -h127.0.0.1 -uroot --flush-logs --databases train > train_bak.sql
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

本章将从实际出发，从按需求建表到设计 E-R 图并进一步转换为关系模型，

再到使用建模工具对数据库进行建模，了解一个数据库从概念模型到具体实现的步骤。此外，在工程认证中，制约因素分析与设计和工程师责任及其分析也是必不可少的内容。

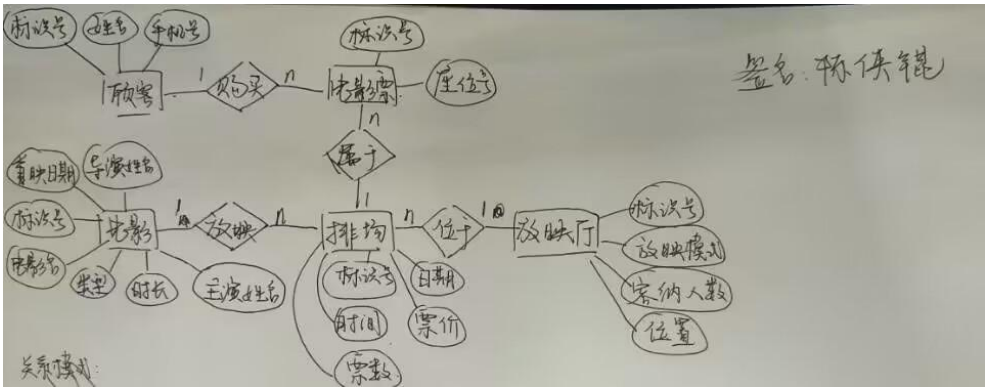
### 2.13.1 从概念模型到 MySQL 实现

本关卡已完成。按要求建表即可，本关略过分析。

### 2.13.2 从需求分析到逻辑模型

本关卡已完成。

根据题目所给信息，可以画出 E-R 图如下图所示。




**E-R 图**

共设立以下五个关系模式：

customer(c_ID, name, phone), primary key:(c_ID)

ticket(ticket_ID, seat_num, c_ID, schedule_ID), primary key:(ticket_ID), foreign key(c_ID, schedule_ID)

movie(movie_ID, title, type, runtime, release_date, director, starring, schedule_ID), primary key(movie_ID), foreign key(schedule_ID)

schedule(schedule_ID, date, time, price, number, hall_ID), primary key(schedule_ID), foreign key(hall_ID)

hall(hall_ID, mode, capacity, location), primary key(hall_ID)

### 2.13.3 建模工具的使用

本关卡已完成。受篇幅限制，本关略过分析。

### 2.13.4 制约因素分析与设计

由在实际工程中设计数据库时，除客户的需求外，我们必须综合考虑社会、

健康、安全、法律、文化及环境等制约因素。

在社会层面，要求我们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的影响，并理解应承担的社会责任。

在安全层面，我们的设计方案应该能保证数据安全与信息安全，给用户营造安全的网络环境。在当前时代，任何一个网民的几乎所有的信息都可以从网络中查询到，如何保护用户的信息安全是每一个工程师都应该思考的问题。

在法律层面，我们的设计方案首先要合法合规，同时应在法律的框架中保障用户的权益，不可以利用数据库来实现一些非法的事情。

在文化层面，我们应具有人文社会科学素养、社会责任感，能够在工程实践中理解并遵守工程职业道德和规范，履行责任。

在环境层面，我们的设计方案应能够理解和评价针对复杂工程问题的工程实践对环境、社会可持续发展的影响。

### 2.13.5 工程师责任及其分析

在一个产品的实现过程中，社会、安全、法律以及文化等各种因素对任务的解决方案起到了制约和调整作用。一个工程师应承担一定的生态伦理责任，要准确和有效地说明新建工程或新建技术可能带来地后果，从而避免对社会和生态环境的危害；应承担一定的职业伦理责任，要有追求真理、客观、求实、诚实、公平、公正的精神；应承担保护公众安全、健康和福祉的责任，不仅要遵守相应的设计规则和标准，而且应考虑到产品或技术作用域市场后的效用。

工程师在设计、实现、维护数据库的过程中，需要考虑上一节所阐述的各种制约因素，设计出既能满足用户需要，又能尽量在其他方面达到最优的数据库。同时工程师必须遵守法律，谨慎行事，不可投机取巧，走歪门邪道。特别是在信息技术行业，由于数据库的缺陷而产生的损失可能是不可估量的。同时，我们国家的工程设计师还要秉承从技术上发展中国以及提升国家综合实力的思想不断的进行创新设计。

## 2.14 数据库应用开发(JAVA 篇)

JDBC（Java DataBase Connectivity，java 数据库连接）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。JDBC 提供了一种基准，据此可以构建更高级的工具和接口，

使数据库开发人员能够编写数据库应用程序。

在本章中，主要使用 java 语言完成数据库的开发。

### 2.14.1 JDBC 体系结构和简单的查询

本关卡已完成。

使用 Class.forName()注册驱动程序，用 DriverManager.getConnection()建立连接，参数需要传入 url，user 和 password。之后使用 Connection 的 createStatement()方法创建一个实例，使用 statement 对象的 executeQuery 返回一个 ResultSet 对象，用 Syetem.out.println 输出文字内容，随后开始遍历 ResultSet 对象（即使用 next()方法取得游标当前行的值），只要还没遍历完，就不断输出遍历到的内容。

由于篇幅限制，本关代码不再给出。

### 2.14.2 用户登录

本关卡已完成。受篇幅限制，本关略过分析。

### 2.14.3 添加新客户

本关卡已完成。受篇幅限制，本关略过分析。

### 2.14.4 银行卡销户

本关卡已完成。受篇幅限制，本关略过分析。

### 2.14.5 客户修改密码

本关卡已完成。受篇幅限制，本关略过分析。

### 2.14.6 事务与转账操作

本关卡已完成。受篇幅限制，本关略过分析。

### 2.14.7 把稀疏表格转为键值对存储

本关卡已完成。受篇幅限制，本关略过分析。

## 2.15 数据库的索引 B+树实现

索引是数据库中的重要组成成分，能够免于遍历数据的耗时，快速定位记录。在本章中我们主要运用 C++语言来实现数据库的 B+树索引，了解其具体构造形式，加深对 B+树的理解。

### 2.15.1 BPlusTreePage 的设计

本关卡已完成，该关卡需要实现以下函数：

**1. bool IsLeafPage() const**

功能为判断页类型是否为叶子节点，return page_type_ == IndexPageType :: LEAF_PAGE 即可。

**2. bool IsRootPage() const**

功能为判断页类型是否为根节点，return parent_page_id_ == INVALID_PAGE_ID 即可。

**3. void SetPageType(IndexPageType page_type)**

功能为设置索引页类型，令 page_type_ = page_type 即可。

**4. int GetSize() const**

功能为得到当前结点中存放的元素（键值对）个数，return size_即可。

**5. void SetSize(int size)**

功能为设置当前结点中存放的元素（键值对）个数，size_ = size 即可。

**6. void IncreaseSize(int amount)**

功能为增加结点元素大小，size_ += amount 即可。

**7. int GetMaxSize() const**

功能为获取最大元素（键值对）个数，return max_size_即可。

**8. void SetMaxSize(int size)**

功能为设置最大元素（键值对）个数，max_size_ = size 即可。

**9. int GetMinSize() cosnt**

功能为获取当前结点允许的最少元素个数，需要先判断是否为根结点，如果是，再判断是否为叶子结点，如果是，返回 1，否则（即根结点是内部结点）返回 2；如果不是根结点，如果是叶结点，return (max_size_ - 1 + 1) / 2，如果不是，return (max_size_ -1 -1 +1) / 2 + 1。

**10.page_id_t GetParentPageId() const**

功能为获得父结点，return parent_page_id_即可。

**11.void SetParentPageId(page_id_t parent_page_id)**

功能为设置父结点，parent_page_id_ = parent_page_id 即可。

**12.page_id_t GetPageId() const**

功能为获得 self page id，令 return parent_page_id_即可。

**13.page_id_t GetPageId() const**

功能为设置 self page id，令 page_id_ = page_id 即可。

**14.void SetLSN(lsn_t lsn)**

功能为设置 lsn，令 lsn_ = lsn 即可。

### 2.15.2 BPlusTreeInternalPage 的设计

本关卡已完成，受篇幅限制，本关略过分析。

### 2.15.3 BPlusLeafPage 的设计

本关卡已完成。受篇幅限制，本关略过分析。

### 2.15.4 B+树索引：Insert

本关卡已完成，受篇幅限制，本关略过分析。

### 2.15.5 B+树索引：Remove

本关卡已完成，受篇幅限制，本关略过分析。

# 3 课程总结

在本次课程实践中，我主要了解了数据库的设计及管理方法、sql 语言中各功能的运用、数据库应用系统的开发以及数据库的内核实验。

本课程实验主要在头歌平台进行，设计有 73 关卡，我共完成 73 关，获头歌平台分数 163 分（实验检查时为 130+分，检查过后将剩余关卡完成），圆满完成试验任务。

在第 1、2 章内容中，我对建表的方法及过程有了更加深刻的记忆，加深了对表的结构与完整性约束的理解和掌握。

在第 3、4、5 章内容中，我对 select 语句进行了较为详细的实践学习，从一开始的两三行语句就可以过关，到后面大量的嵌套查询、连接查询、还有各种聚集函数及 sql 内置函数的使用，整个过程的难度层层递进，充满了挑战性。在数据查询二中的题目难度更是进一步提升，首先需要好好理解题意，然后将目标一步步拆解，再思考哪些部分要先做，哪些部分要后做……在这个复杂又艰巨的过程中，我对 mysql 选择功能的理解大大加深，在查询资料的过程中，还学会了一些非常好用的函数，比如获取日期的函数 extract、week、weekday，排名函数 rank() over、dense_rank() over 等，有助于我日后的学习。我还进一步掌握了修改表中数据的方法，在使用 insert、update、delete 功能时也可能会遇到比较复杂的情况，比如 update 功能需要将某个属性的值设置为某个聚集函数的结果，我在查找资料的过程中也学习到了许多编写方法。

在第 6、7、8 章内容中，我学会了视图的创建及使用视图进行查找数据，这可以使得查找效率大大增加；学会了使用流程控制语句、游标、事务表示的存储过程，对存储过程有了更加深刻的了解；理解了触发器的作用，当基础的约束无法实现某个复杂的业务规则时，可以考虑用触发器来实现。

在第 9、10 章内容中，我学习了用户自定义函数的方法与其调用方式；掌握了安全性控制的内容，学会如何使用 grant、revoke 等授予或收回权限。

在第 11、12 章内容中，我对并行事务的调度与数据库恢复有了更深刻的了解，也进一步体会到这两方面内容在实际应用中的重要性。

第 13、14、15 章的内容，对我的能力进行了更进一步的挑战，我对设计数据库的具体步骤更加清楚，也了解了一些 java 编程与 B+树索引的知识，不过这三章中有些内容还是太难，最后我不得不寻求多方帮助得以过关。

总的说来，整个实验难度适中，但有一些超过我能力的题目。实验课程与理论课程结合的十分紧密，又具有自己的特点，有许多内容是课本上没有强调但在工程中很可能使用到的。我从代码的世界中认识到了和课本中不一样的数据库，也对诸如安全性问题、数据存储问题等实际性问题有了更进一步的思考。

# 附录

### 2.3.13 客户总资产

```
select c_id, c_name, ifnull(sum(property) ,0) as total_property
from client left outer join(
    select pro_c_id, ifnull(sum(amount),0) as total_amount
    from (
        select pro_c_id, pro_quantity*p_amount as amount_p
        from property, finances_product
        where pro_type = 1 and pro_pif_id=p_id
        union all
        select pro_c_id, pro_quantity*i_amount as amount_i
        from property, insurance
        where pro_type = 2 and pro_pif_id=i_id
        union all
        select pro_c_id, pro_quantity*f_amount as amount_f
        from property, fund
        where pro_type = 3 and pro_pif_id=f_id
    ) as a(pro_c_id, amount)
    group by pro_c_id #  投资总额
    union all
    select pro_c_id, ifnull(sum(pro_income),0) total_income
    from property
    group by pro_c_id #  投资总收益
    union all
    select b_c_id as pro_c_id, b_balance
    from bank_card
    where b_type='储蓄卡' #  储蓄卡余额
    union all
    select b_c_id as pro_c_id, 0-b_balance
```

```
    from bank_card
    where b_type='信用卡' # 信用卡透支余额
) as b(pro_c_id, property)
on(c_id=pro_c_id)
group by c_id
order by c_id;
```

**2.3.16 持有完全相同基金组合的用户**

```
select first.pro_c_id as c_id1, second.pro_c_id as c_id2
from property first inner join property second on(
    first.pro_c_id<second.pro_c_id
    and first.pro_type='3'
    and second.pro_type='3'
    and first.pro_pif_id=second.pro_pif_id
)  # 将有相同基金的两个人连到一个元组中
group by c_id1,c_id2
# having 语句保证了选出的两个人持有的基金数相同
having count(*)=(
    select count(*)
    from property
    where pro_c_id=first.pro_c_id and pro_type='3' # 保证选出的元组，持有基金数与第一个人持有的基金数相同
) and count(*)=(
    select count(*)
    from property
    where pro_c_id=second.pro_c_id and pro_type='3 '# 保证选出的元组，持有基金数与第二个人持有的基金数相同
);
```

**2.4.3 查询购买了所有畅销理财产品的客户**

```
select pro_c_id
```

```
from (
    select distinct pro_c_id # 所有购买了理财产品的客户
    from property
    where pro_type=1
) as table1
where not exists(
    select pro_pif_id
    from(
        select distinct pro_pif_id # 找出所有的畅销理财产品
        from property
        where pro_type=1
        group by pro_pif_id
        having count(*)>2
    ) as table2
    where not exists(
        select pro_pif_id # 被购买了的理财产品
        from property
        where pro_type=1
        and property.pro_c_id = table1.pro_c_id
        and property.pro_pif_id = table2.pro_pif_id
    )
)
```

### 2.7.2 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 done, tp, wk int default false;
    declare doc, nur1, nur2, head char(30);
    declare cur1 cursor for select e_name from employee where e_type = 3;
```

```
declare cur2 cursor for select e_type, e_name from employee where e_type < 3;
declare continue handler for not found set done = true;
open cur1;
open cur2;
while start_date <= end_date do
    fetch cur1 into nur1;
    if done then
        close cur1;
        open cur1;
        set done = false;
        fetch cur1 into nur1;
    end if;
    fetch cur1 into nur2;
    if done then
        close cur1;
        open cur1;
        set done = false;
        fetch cur1 into nur2;
    end if;
    set wk = weekday(start_date);
    if wk = 0 and head is not null then
        set doc = head;
        set head = null;
    else
        fetch cur2 into tp, doc;
        if done then
            close cur2;
            open cur2;
            set done = false;
```

```
                fetch cur2 into tp, doc;
            end if;
            if wk > 4 and tp = 1 then
                set head = doc;
                fetch cur2 into tp, doc;
                if done then
                    close cur2;
                    open cur2;
                    set done = false;
                    fetch cur2 into tp, doc;
                end if;
            end if;
        end if;
        insert into night_shift_schedule values (start_date, doc, nur1, nur2);
        set start_date = date_add(start_date, interval 1 day);
    end while;
end$$

delimiter ;
```