华中科技大学

数据库系统原理实践报告

专　　业：	计算机专业
班　　级：	CS1906
学　　号：	U201915116
姓　　名：	廖翔
指导教师：	胡侃

分数	
教师签名	

2022　　年 7　月 1　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

2 任务实施过程与分析 2

2.1 数据库、表、完整性约束的创建与修改 2
2.2 数据查询(SELECT) 4
2.3 数据的插入、修改与删除(INSERT,UPDATE,DELETE) 13
2.4 视图 14
2.5 存储过程与事务 15
2.6 触发器 17
2.7 用户自定义函数 18
2.8 安全性控制 19
2.9 并发控制与事务的隔离级别 20
2.10 备份+日志：介质故障与数据库恢复 23
2.11 数据库设计与实现 24
2.12 数据库应用开发 27

3 课程总结 30

附录 31

1 课程任务概述

总体任务：

本实践课程作为“数据库系统原理”课程配套开设的实践课，主要通过在 Educoder 上编写数据库相关代码通过测评达到实验目的，并通过此来提高我们对于实际操作数据库的熟练度。

任务分解情况说明：

整个实践内容可以分为如下几个部分：

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)。

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

这一小节需要我们学会使用 MySQL 语句创建数据库，创建表以及表的主码约束，创建外码约束，CHECK，DEFAULT 和 UNIQUE 约束。同时还要学会使用修改表名，添加、删除和修改字段，添加、删除和修改约束。

2.1.1 创建数据库

```
create database beijing2022;
```

2.1.2 创建表及表的主码约束

```
drop database TestDb;
create database TestDb;
use TestDb;
create table if not exists t_emp(
        id int primary key,
        name varchar(32),
        deptId int,
        salary float
);
```

2.1.3 创建外码约束(foreign key)

```
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)
)
```

2.1.4 CHECK 约束

```
use MyDb;
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)
)
```

CHECK 约束由关键字 CHECK 引出，后跟一对括弧，括弧里为条件表达式，只有当条件表达式的值为 true 时，数据（插入的新数据，或修改后的数据）才会被接受，否则将被拒绝。

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。

2.1.7 修改表名

```
USE TestDb1;
#请在以下空白处添加恰当的语句，将表名 your_table 更改为 my_table:
alter table your_table
    rename my_table;
```

2.1.8 添加与删除字段

```
use MyDb;
#请在以下空白处添加适当的 SQL 代码，实现编程要求
#语句 1：删除表 orderDetail 中的列 orderDate
alter table orderDetail drop orderDate;
#语句 2：添加列 unitPrice
alter table orderDetail
add unitPrice numeric(10, 2);
```

2.1.9 修改字段

```
use MyDb;
#请在以下空白处添加适当的 SQL 语句，实现编程要求
alter table addressBook
modify QQ char(12),
    rename column weixin to wechat;
```

2.1.10 添加、删除与修改约束

```
use MyDb;
#请在以下空白处填写适当的诘句，实现编程要求。
#(1) 为表 Staff 添加主码
alter table Staff
add primary key (staffNo);
#(2) Dept.mgrStaffNo 是外码，对应的主码是 Staff.staffNo,请添加这个外码，名字为
FK_Dept_mgrStaffNo:
```

```
alter table Dept
add constraint FK_Dept_mgrStaffNo foreign key (mgrStaffNo) references Staff(staffNo);
#(3) Staff.dept 是外码，对应的主码是 Dept.deptNo. 请添加这个外码，名字为 FK_Staff_dept:
alter table Staff
add constraint FK_Staff_dept foreign key (dept) references Dept(deptNo);
#(4) 为表 Staff 添加 check 约束，规则为：gender 的值只能为 F 或 M；约束名为 CK_Staff_gender:
alter table Staff
add constraint CK_Staff_gender check (gender in ('F', 'M'));
#(5) 为表 Dept 添加 unique 约束：deptName 不允许重复。约束名为 UN_Dept_deptName：
alter table Dept
add constraint UN_Dept_deptName unique(deptName);
```

2.2 数据查询(Select)

本小节主要是数据的查询。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）。数据集是一个事先创建并已初始化的数据库：某银行金融服务场景模拟数据库。只需写出能实现查询要求的那条 SQL 语句即可。

2.2.1 查询客户主要信息

```
select c_name, c_phone, c_mail from client order by c_id;
```

2.2.2 邮箱为 null 的客户

```
select c_id,
       c_name,
       c_id_card,
       c_phone
from client
where c_mail is NULL;
```

2.2.3 既买了保险又买了基金的客户

```
select c_name,
        c_mail,
         c_phone
from(
            select distinct t1.c_id,
                    t1.c_name,
                    t1.c_mail,
                    t1.c_phone
            from (
                        select c_name,
```

```
                c_mail,
                c_phone,
                c_id
            from client
                join property
            where c_id = property.pro_c_id
                and property.pro_type = 2
        ) t1
        join (
            select c_name,
                c_mail,
                c_phone,
                c_id
            from client
                join property
            where c_id = property.pro_c_id
                and property.pro_type = 3
        ) t2
    where t1.c_id = t2.c_id
) c
order by c_id;
```

2.2.4 办理了储蓄卡的客户信息

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

2.2.5 每份金额在 30000～50000 之间的理财产品

```
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;
```

2.2.6 商品收益的众数

```
select pro_income,
        count(pro_income) as 'presence'
from property
where pro_income is not null
group by pro_income
having count(*) >= all(
                select count(*)
                from property
                group by pro_income
        );
```

这一关需要使用 count(*)函数来进行众数的筛选。

2.2.7 未购买任何理财产品的武汉居民

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

2.2.8 持有两张信用卡的用户

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

2.2.9 购买了货币型基金的客户信息

```
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;
```

2.2.10 投资总收益前三名的客户

```
select c_name,
       c_id_card,
       sum(pro_income) as total_income
from (
              select c_name,
                     c_id_card,
                     p.pro_income
              from client c,
                     property p
              where c.c_id = p.pro_c_id
                     and p.pro_status = "可用"
       ) t1
group by c_name,
       c_id_card
order by total_income desc
limit 3;
```

注意这里使用 limit 来限制结果为 3 项。

2.2.11 黄姓客户持卡数量

```
select c_id,
       c_name,
       count(b_c_id) as number_of_cards
from client
       left outer join bank_card on (c_id = b_c_id)
```

```
group by c_id
having c_name like '黄%'
```

这里使用 like 来进行模糊查询。

2.2.12 客户理财、保险与基金投资总额

```
select c_name,
       c_id_card,
       sum(
          ifnull(p_amount, ifnull(i_amount, ifnull(f_amount, 0))) * ifnull(pro_quantity, 0)
       ) as total_amount
from (
    select c.c_id,
           c.c_name,
           c.c_id_card,
           p.p_amount,
           i.i_amount,
           f.f_amount,
           pro.pro_quantity
    from client as c
           left join property as pro on c.c_id = pro.pro_c_id
           left join finances_product as p on pro.pro_pif_id = p.p_id
           and pro_type = 1
           left join insurance as i on pro.pro_pif_id = i.i_id
           and pro_type = 2
           left join fund as f on pro_pif_id = f.f_id
           and pro_type = 3
) as union_table
group by c_id
order by 3 desc;
```

2.2.13 客户总资产

```
select c_id,
       c_name,
       coalesce(sum(money), 0) as total_property
from client
       left join (
              select b_c_id as id,
                     b_balance as money
              from bank_card
              where b_type = '储蓄卡'
              union all
              select b_c_id as id,
       (-1 * b_balance) as money
```

```
            from bank_card
            where b_type = '信用卡'
            union all
            select pro_c_id as id,
        (p_amount * pro_quantity) as money
            from finances_product,
                property
            where p_id = pro_pif_id
                and pro_type = 1
            union all
            select pro_c_id as id,
        (i_amount * pro_quantity) as money
            from insurance,
                property
            where i_id = pro_pif_id
                and pro_type = 2
            union all
            select pro_c_id as id,
        (f_amount * pro_quantity) as money
            from fund,
                property
            where f_id = pro_pif_id
                and pro_type = 3
            union all
            select pro_c_id as id,
                pro_income as money
            from property
        ) as bank on c_id = id
    group by c_id
    order by c_id;
```

首先要明确总资产的求法，即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。所以需要使用(-1*b_balance)这样的语句，同样其余的资产就使用(amout*quantity)来计算。

2.2.14 第 N 高问题

```
    select i_id,
        i_amount
    from insurance
    where i_amount in (
            select min(i_amount) as mi
```

```
        from(
                select distinct i_amount
                from insurance
                order by i_amount desc
                limit 4
        ) as new
    );
```

2.2.15 基金收益两种方式排名

```
    select pro_c_id,
        total_revenue,
        r as 'rank'
    from (
            select pro_c_id,
                total_revenue,
                rank() over (
                    order by total_revenue desc
                ) as r
            from (
                    select pro_c_id,
                        sum(pro_income) as total_revenue
                    from property
                    where pro_status = '可用'
                        and pro_type = 3
                    group by pro_c_id
                    order by pro_c_id
                ) as a
        ) as b
    order by r,
        pro_c_id;
-- (2) 基金总收益排名(名次连续)
    select pro_c_id,
        total_revenue,
        r as 'rank'
    from (
            select pro_c_id,
                total_revenue,
                dense_rank() over (
                    order by total_revenue desc
                ) as r
            from (
                select pro_c_id,
                    sum(pro_income) as total_revenue
```

```
                from property
            where pro_status = '可用'
                and pro_type = 3
            group by pro_c_id
            order by pro_c_id
        ) as a
    ) as b
order by r,
    pro_c_id
```

2.2.16 持有完全相同基金组合的客户

该任务关卡跳过

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

```
select b_c_id,
    sum(b_balance) as credit_card_amount
from bank_card
where b_c_id in (
        select b_c_id as id
        from bank_card
        where b_type = '信用卡'
            and b_balance > 5000
        group by b_c_id
    )
    and b_type = '信用卡'
group by b_c_id
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

```
select week_of_trading,
    if(sum(Monday) = 0, NULL, sum(Monday)) as Monday,
    if(sum(Tuesday) = 0, NULL, sum(Tuesday)) as Tuesday,
    if(sum(Wednesday) = 0, NULL, sum(Wednesday)) as Wednesday,
    if(sum(Thursday) = 0, NULL, sum(Thursday)) as Thursday,
    if(sum(Friday) = 0, NULL, sum(Friday)) as Friday
from (
        select week_of_trading,
            (
                case
```

```
                    dayofweek(pro_purchase_time)
                    when 2 then sum(pro_quantity * f_amount)
                    else 0
                end
        ) as Monday,
        (
                case
                    dayofweek(pro_purchase_time)
                    when 3 then sum(pro_quantity * f_amount)
                    else 0
                end
        ) as Tuesday,
        (
                case
                    dayofweek(pro_purchase_time)
                    when 4 then sum(pro_quantity * f_amount)
                    else 0
                end
        ) as Wednesday,
        (
                case
                    dayofweek(pro_purchase_time)
                    when 5 then sum(pro_quantity * f_amount)
                    else 0
                end
        ) as Thursday,
        (
                case
                    dayofweek(pro_purchase_time)
                    when 6 then sum(pro_quantity * f_amount)
                    else 0
                end
        ) as Friday
from fund
        right join (
                select pro_pif_id,
                    (week(pro_purchase_time) -5) as week_of_trading,
                    pro_purchase_time,
                    pro_quantity
                from property
                where pro_type = 3
                    and pro_purchase_time >= date('2022-02-07')
        ) as a on fund.f_id = a.pro_pif_id
group by pro_purchase_time
```

```
        ) as b
group by week_of_trading;
```

为了通过本关需要学会使用 if 和 case 函数，同时需要会灵活应用题中所给的几个重要函数，包括 week(), dayofweek(), weekday(), if()等，使用这些函数与 if-case 相结合即可解题。

2.3 数据的插入、修改与删除(Insert,Update,Delete)

本小节需要我们学会使用 MySQL 编写语句完成数据的插入，修改和删除。

2.3.1 插入多条完整的客户信息

该关卡任务已完成，实施情况本报告略过

2.3.2 插入不完整的客户信息

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

2.3.3 批量插入数据

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

2.3.4 删除没有银行卡的客户信息

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

2.3.5 冻结客户资产

```
update property
set pro_status = "冻结"
where pro_c_id in (
            select c_id
            from client
```

```
            where c_phone = "13686431238"
        );
```

2.3.6 连接更新

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

2.4 视图

本小节需要我们学会使用 MySQL 实现视图的创建和使用。

2.4.1 创建所有保险资产的详细记录视图

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

创建视图其实很简单，学会使用查询语句即可

2.4.2 基于视图的查询

```
select c_name,
    c_id_card,
    sum(i_amount * pro_quantity) 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;
```

2.5 存储过程与事务

存储过程（Stored Procedure）是一种可编程的数据对象，是由一组为了完成特定功能的 SQL 语句和控制结构组成的复杂程序。存储过程经编译后存储在数据库中，外部程序通过指定存储过程的名字并给定参数（如果存储过程定义了参数）来调用它。由于存储过程、自定义函数以及触发器，都是可编程对象，都会用到程序的控制结构和各类语句，本小节需要我们学会使用包括：变量定义语句，BEGIN...END 语句，IF 语句，CASE 语句，REPEAT 语句，WHILE 语句，游标(CURSOR)的定义和使用，事务的定义等。

2.5.1 使用流程控制语句的存储过程

```
use fib;

-- 创建存储过程`sp_fibonacci(in m int)`，向表 fibonacci 插入斐波拉契数列的前 m 项，及其对应的斐波拉契数。fibonacci 表初始值为一张空表。请保证你的存储过程可以多次运行而不出错。

drop procedure if exists sp_fibonacci;
delimiter $$
create procedure sp_fibonacci(in m int)
begin
######## 请补充代码完成存储过程体 ########
declare an1 int;
declare an2 int;
declare an int;
declare i int;
set an1 = 1;
set an2 = 0;
set i = 0;
set an=an2+an1;
while i <m do
    if i=0 then
        insert into fibonacci(n, fibn) values(0,0);
    elseif i=1 then
        insert into fibonacci(n, fibn) values(1,1);
    else
        insert into fibonacci(n, fibn) values(i, an);
        set an2=an1;
        set an1=an;
        set an=an2+an1;
    end if;
```

```
        set i=i+1;
    end while;
    end $$
    delimiter ;
```

2.5.2 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

2.5.3 使用事务的存储过程

```
    use finance1;
-- 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，实现从一个帐户向另一个帐户转账。
-- 请补充代码完成该过程：
    delimiter $$
    create procedure sp_transfer(
                        IN applicant_id int,
                        IN source_card_id char(30),
                        IN receiver_id int,
                        IN dest_card_id char(30),
                        IN amount numeric(10,2),
                        OUT return_code int)
    BEGIN
    declare owner1 , owner2 int default 0;
    declare b1, b2 NUMERIC(10,2);
    declare t1, t2 char(30);
    select b_c_id, b_balance, b_type into owner1, b1, t1 from bank_card where b_number =
    source_card_id ;
    select b_c_id, b_balance, b_type into owner2, b2, t2 from bank_card where b_number =
    dest_card_id;
    start transaction;
    if (owner1 <> applicant_id OR owner2 <> receiver_id) then
        set return_code = 0;
    elseif (t1 = '信用卡' and t2 = '储蓄卡')then
        set return_code = 0;
    elseif (t1 = '储蓄卡' and amount > b1) then
        set return_code = 0;
    elseif (t2 = '信用卡') then
        update bank_card set b_balance = b_balance-amount where b_number = source_card_id;
        update bank_card set b_balance = b_balance-amount where b_number = dest_card_id;
        set return_code = 1;
    else
        update bank_card set b_balance = b_balance-amount where b_number = source_card_id;
        update bank_card set b_balance = b_balance+amount where b_number = dest_card_id;
        set return_code = 1;
```

```
    end if;
    commit;

    END$$

    delimiter ;
```

为了实现转账功能的存储过程，需要使用 if-else 语句来进行判断并执行不同的操作。

2.6 触发器

触发器(trigger)是对约束(constraints)的补充，它用程序来完成 constraint 实现不了的业务规则。我们知道，当执行 insert,delete 或 update 语句时，DBMS 将检查该语句是否会导致数据违返约束。同样的，如果在表上定义了 insert,delete 或 update 事件驱动的触发器，那么当执行 insert,delete 或 update 语句时，还会激活相应的触发器，以检查数据的完整性。

约束和触发器共同构成 DBMS 的完整性子系统。

本实训的任务是用 create trigger 语句，创建符合任务要求的触发器。

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

```
use finance1;
drop trigger if exists before_property_inserted;
-- 请在适当的地方补充代码，完成任务要求：
delimiter $$
CREATE TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW
BEGIN
declare msg char(60);
declare id int;
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 then
        select p_id into id from finances_product where p_id = new.pro_pif_id;
        if id is null then
        set msg = concat('finances product #', new.pro_pif_id, ' not found!    ');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
```

```
        end if;
    elseif new.pro_type = 2 then
        select i_id into id from insurance where i_id = new.pro_pif_id;
        if id is null then
        set msg = concat('insurance #', new.pro_pif_id, ' not found!    ');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
        end if;
    elseif new.pro_type = 3 then
        select f_id into id from fund where f_id = new.pro_pif_id;
        if id is null then
        set msg = concat('fund #', new.pro_pif_id, ' not found!    ');
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
        end if;
    end if;
    END$$
    delimiter ;
```

因为投资类别不同，所以需要针对不同的投资类别实现不同的业务规则，例如针对 pro_type 为 1，2，3 等，同时还要限制 pro_type 不可以取其他值。

2.7 用户自定义函数

本小节需要我们创建一个完成指定任务的函数，并在语句中使用它。

2.7.1 创建函数并在语句中使用它

```
    use finance1;
    set global log_bin_trust_function_creators=1;
    drop function IF EXISTS get_deposit;
/*
   用 create function 语句创建符合以下要求的函数：
   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。函数的参数名可以自己命名:*/

    delimiter $$
    create function get_deposit(client_id int)
    returns numeric(10,2)
    begin
        declare sumAmount int;
       select sum(b_balance) into sumAmount from bank_card where b_c_id = client_id and b_type =
    '储蓄卡' group by b_c_id   ;
        return sumAmount;
    end$$
    delimiter ;

    /*   应用该函数查询存款总额在 100 万以上的客户身份证号，姓名和存储总额(total_deposit)，
```

```
结果依存款总额从高到代排序  */
select distinct c_id_card, c_name, get_deposit(c_id) as total_deposit
from bank_card, client
where get_deposit(c_id) >= 1000000
order by get_deposit(c_id) desc;
```

本小节我们自己创建了一个 get_deposit 函数来获取数据库中的特定内容，同时配合其完成其他查询任务。

2.8 安全性控制

安全性一般要从全方位、多层次考虑具体的安全措施，比如不仅要考虑 MySQL 的安全性，还要考虑服务器主机的安全性(如防 eavesdropping, altering, playback, 以及 DoS 攻击等）。本小节需要完成 MySQL 安全性中的一部分：存取控制。

2.8.1 用户和权限

```
#(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 `finance`.`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 `finance`.`bank_card` TO `jerry` @`%`;
#(4) 收回用户 Cindy 查询银行卡信息的权限。
revoke
select on bank_card
from Cindy;
```

2.8.2 用户、角色与权限

```
# 请填写语句，完成以下功能：
# (1) 创建角色 client_manager 和 fund_manager；
create role 'client_manager',
'fund_manager';
# (2) 授予 client_manager 对 client 表拥有 select,insert,update 的权限；
GRANT SELECT,
      INSERT,
      UPDATE ON `finance`.`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 `finance`.`bank_card` TO `client_manager`
@`%`;
# (4) 授予 fund_manager 对 fund 表的 select,insert,update 权限；
```

```
GRANT SELECT,
    INSERT,
    UPDATE ON `finance`.`fund` TO `fund_manager` @`%`;
# (5) 将 client_manager 的权限授予用户 tom 和 jerry；
GRANT `client_manager` @`%` TO `jerry` @`%`,
`tom` @`%`;
# (6) 将 fund_manager 权限授予用户 Cindy.
GRANT `fund_manager` @`%` TO `Cindy` @`%`;
```

2.9 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，包括：丢失修改(lost update)，读脏(dirty read)，不可重复读(unrepeatable read)以及幻读(phantom read)。本小节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

```
-- 请不要在本代码文件中添加空行！！！
use testdb1;
# 设置事务的隔离级别为 read uncommitted
set session transaction isolation level read uncommitted;
-- 开启事务
start transaction;
insert into dept(name)
values('运维部');
# 回滚事务：
rollback;
/* 结束 */
```

根据题目要求将事务的隔离级别设置为 read uncommitted 即可。

2.9.2 读脏

```
-- 事务 1:
use testdb1;
## 请设置适当的事务隔离级别
set session transaction isolation level READ UNCOMMITTED;
start transaction;
-- 时刻 2 - 事务 1 读航班余票,发生在事务 2 修改之后
## 添加等待代码，确保读脏
```

```
set @n = sleep(1);
select tickets from ticket where flight_no = 'CA8213';
commit;
```

```
    -- 事务 2
```

```
use testdb1;
## 请设置适当的事务隔离级别
set session transaction isolation level READ UNCOMMITTED;
start transaction;
-- 时刻 1 - 事务 2 修改航班余票
update ticket set tickets = tickets - 1 where flight_no = 'CA8213';

-- 时刻 3 - 事务 2 取消本次修改
## 请添加代码，使事务 1 在事务 2 撤销前读脏;
set @n = sleep(2);

rollback;
```

根据题目要求添加等待代码，同时通过 rollback 取消本次修改。

2.9.3 不可重复读

```
-- 事务 1:
## 请设置适当的事务隔离级别
set session transaction isolation level read uncommitted;
--开启事务
start transaction;
-- 时刻 1 - 事务 1 读航班余票:
insert into result
select now(),1 t, tickets from ticket where flight_no = 'CZ5525';
## 添加等待代码，确保事务 2 的第一次读取在事务 1 修改前发生
set @n = sleep(2);
-- 时刻 3 - 事务 1 修改余票，并立即读取:
update ticket set tickets = tickets - 1 where flight_no = 'CZ5525';
insert into result
select now(),1 t, tickets from ticket where flight_no = 'CZ5525';
## 添加代码，使事务 2 的第 2 次读取在事务 1 修改之后，提交之前发生
set @n = sleep(4);
commit;
-- 时刻 6 - 事务 1 在 t2 也提交后读取余票
## 添加代码，确保事务 1 在事务 2 提交后读取
set @n = sleep(1);
insert into result
select now(), 1 t, tickets from ticket where flight_no = 'CZ5525';
```

2.9.4 幻读

```
-- 事务 1（采用默认的事务隔离级别- repeatable read）:
use testdb1;
select @@transaction_isolation;
start transaction;
## 第 1 次查询余票超过 300 张的航班信息
select * from ticket where tickets > 300;
set @n = sleep(2);
-- 修改航班 MU5111 的执飞机型为 A330-300：
update ticket set aircraft = 'A330-300' where flight_no = 'MU5111';
-- 第 2 次查询余票超过 300 张的航班信息
select * from ticket where tickets > 300;
commit;
```

解答本题时首先需要明确幻读的概念，在某些文献中，幻读被归为不可重复读(unrepeatable read)中的一类，而另一些则把它与不可重复读区分开来：幻读是指一个事务(t1)读取到某数据后，另一个事务(t2)作了 insert 或 delete 操作，事务 t1 再次读取该数据时，魔幻般地发现数据变多了或者变少了(记录数量不一致)；而不可重复读限指事务 t2 作了 update 操作，致使 t1 的两次读操作读到的结果(数据的值)不一致。

除了最高级别 serializable(可串行化)以外的任何隔离级别，都有可能发生幻读。明确这一特点后，就可以很清晰地编写该关代码了。

2.9.5 主动加锁保证可重复读

```
-- 事务 1:
use testdb1;
set session transaction isolation level read uncommitted;
start transaction;
# 第 1 次查询航班'MU2455'的余票
select tickets from ticket where flight_no = 'MU2455' for share;
set @n = sleep(5);
# 第 2 次查询航班'MU2455'的余票
select tickets from ticket where flight_no = 'MU2455';
commit;
-- 第 3 次查询所有航班的余票，发生在事务 2 提交后
set @n = sleep(1);
select * from ticket;
```

```
-- 事务 2:
use testdb1;
```

```
set session transaction isolation level read uncommitted;
start transaction;
set @n = sleep(1);
# 在事务 1 的第 1，2 次查询之间，试图出票 1 张(航班 MU2455)：
update ticket set tickets = tickets - 1 where flight_no = 'MU2455';
commit;
```

MySQL 的 select 语句支持 for share 和 for update 短语，分别表示对表加共享(Share)锁和写(write)锁，共享锁也叫读锁，写锁又叫排它锁。所以在这关我们使用 for share 语句来加读锁。

2.9.6 可串行化

```
-- 事务 1:
use testdb1;
start transaction;
set @n = sleep(2);
select tickets from ticket where flight_no = 'MU2455';
select tickets from ticket where flight_no = 'MU2455';
commit;
-- 事务 2:
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

和大多数 DBMS 一样，MySQL 在事务机制的保障下，利用备份、日志文件实现恢复。在生产环境中，通常需要部署多台服务器，可以利用复制、镜像、master-slave 等机制保障数据的安全性、可用性，同时提供高并发服务。这一节需要我们学会如何备份。

2.10.1 备份与恢复

```
# 对数据库 residents 作海量备份,备份至文件 residents_bak.sql:
mysqldump -h127.0.0.1 -uroot --databases residents > residents_bak.sql
# 利用备份文件 residents_bak.sql 还原数据库:
mysql -h127.0.0.1 -uroot < residents_bak.sql
```

mysql 是 MySQL 最重要的客户端管理工具，通常用户都是通过 mysql 登录到 MySQL 服务器，进行各种操作。此外，还可以直接通过它执行 SQL 脚本，还原或创建新库。mysql -h127.0.0.1 -uroot -p12313 < mydb.sql。这样会直接执行 mydb.sql 的脚本。通过 mysqldump 备份出来的脚 本文件，可以用该方法直接用来恢复原数据库。

2.10.2 备份+日志：介质故障的发生与数据库的恢复

该关卡任务已完成，实施情况本报告略过

2.11 数据库设计与实现

数据库设计的主要过程可以概括为：在精准需求分析的基础上，为数据库建模，从概念模型到逻辑模型，再到物理模型，即在某个具体的 DBMS 上实现，最后是应用系统的实施和运维。其中最重要的工作是数据库建模。本小节需要我们完成概念模型、逻辑模型和物理模型的设计。

2.11.1 从概念模型到 MySQL 实现

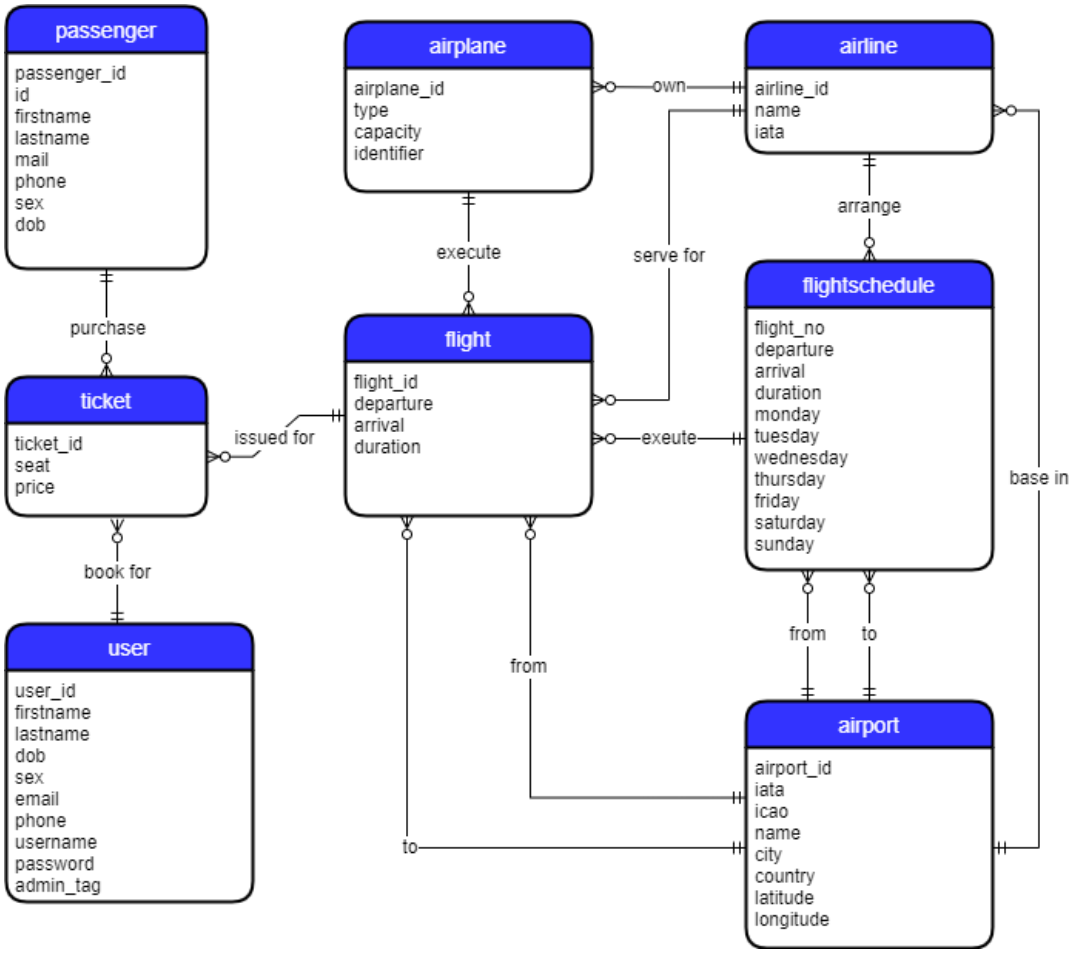

图 2.1

根据题目已给出的逻辑 ER 图可以非常简单地写出相关代码。具体代码这里略过。

2.11.2 从需求分析到逻辑模型

根据题目描述和相关的关系描述可以给出如下的关系模式：

```
customer(c_ID,name,phone),主码:(c_ID)
movie(movie_ID,title,type,runtime,release_date,director,starring),主码:(movie_ID)
schedule(schedule_ID,date,time,price,number,movie_ID,hall_ID),主码:(schedule_ID);外码(movie_ID,hall_ID)
hall(hall_ID,mode,capacity,location),主码:(hall_ID)
ticket(ticket_ID,seat_num,schedule_ID),主码:(ticket_ID);外码(schedule_ID)
```

并据此可以画出如下的 ER 图：

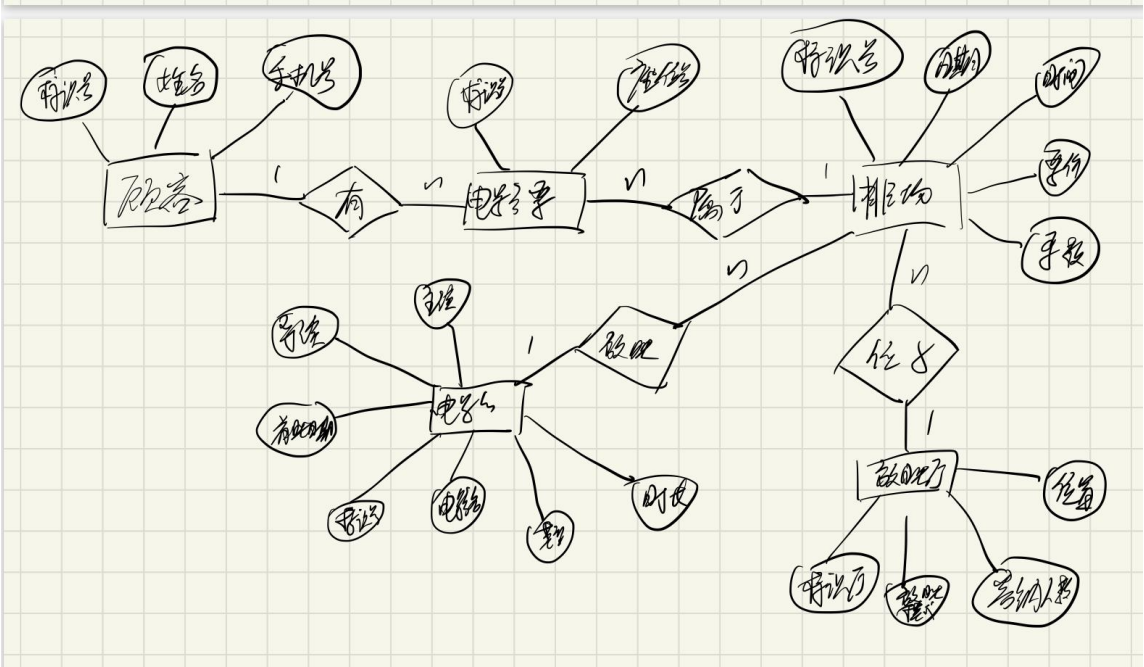

图 2.2

2.11.3 建模工具的使用

该关卡任务已完成，实施情况本报告略过

2.11.4 制约因素分析与设计

数据库应用于生产管理方面，规模化的批量工作代替传统的人工操作模式，大大提升了工作效益和工作质量，方便了人们的生活，也不断地推进着社会的生产力的发展；因此说数据库技术在我们现代社会中起着不可忽视的作用，数据技术也正推动着时代的进步。但我们同样需要意识到法律，文化，安全等这些稍有差错就有可能酿成大错的因素，这就需要我们在设计过程中充分考虑各种可能性，尽可能交付一个足够完美的作品。

2.11.5 工程师责任及其分析

作为一名工程师，需要能够基于工程相关背景知识进行合理分析，评价专业工程实践和复杂工程问题解决方案对社会、健康、安全、法律以及文化的影响，并理解应承担的责任。同时还要能够能够基于科学原理并采用科学方法对复杂工程问题进行研究，包括设计实验、分析与解释数据、并通过信息综合得出合理有

效的结论。并且还要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。

2.12 数据库应用开发

数据库应用开发，系指利用高级语言开发数据库应用系统，这些应用系统可是 C/S 架构的，也可以是 B/S 架构的(即 web 应用)，开发的语言可是 C++,JAVA，PHP，JSP，Python 等。本小节通过几个简单的练习来掌握 JAVA 开发数据库应用的基本知识。

2.12.1 JDBC 体系结构和简单的查询

```
Class.forName("com.mysql.cj.jdbc.Driver");
String userName = "root";
String passWord = "123123";
String url =
"jdbc:mysql://127.0.0.1:3306/finance?useUnicode=true&characterEncoding=UTF8&useSSL=false&serverTimezone=UTC";
connection = DriverManager.getConnection(url, userName, passWord);

statement = connection.createStatement();
resultSet = statement.executeQuery("select * 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"));
```

通过 username, password, url 访问到数据库后执行对应的语句即可。

2.12.2 从需求分析到逻辑模型用户登录

该关卡任务已完成，实施情况本报告略过

2.12.3 添加新客户

```
String sql = "insert into client values(?,?,?,?,?,?)";
PreparedStatement pps = connection.prepareStatement(sql);
pps.setString(1, String.valueOf(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);
int n = pps.executeUpdate();
```

```
return n;
```

通过事先设置占位符，填空后再执行来实现添加客户的效果。

2.12.4 银行卡销户

该关卡任务已完成，实施情况本报告略过

2.12.5 客户修改密码

```
try{
        Statement statement = connection.createStatement();
        String sql1 = "select * from client where c_mail= '"+mail+"'";
        ResultSet resultSet = statement.executeQuery(sql1);
        if(resultSet.next()){
                String originalPassword = resultSet.getString("c_password");
                if(!password.equals(originalPassword)){
                        return 3;
                }
        }
        String sql = "update client set c_password = '"+newPass+"' where c_mail = '"+mail+"';";
        int n    = statement.executeUpdate(sql);
        if(n==1){
                return 1;
        }else{
                return 2;
        }
}catch(Exception e){
        return -1;
}
```

为了通过本关，最主要的就是要理解题意，对不同的处理结果返回对应的返回值。

2.12.6 事务与转账操作

该关卡任务已完成，实施情况本报告略过

2.12.7 把稀疏表格转为键值对存储

```
if(mathScore!=null){
                insertSql = "insert into sc values(?,?,?)";
                PreparedStatement pps = connection.prepareStatement(insertSql);
                pps.setString(1, sno);
                pps.setString(2,"math");
                pps.setString(3,mathScore);
                int n = pps.executeUpdate();
            }
```

```
            if(englishScore!=null){
                    insertSql = "insert into sc values(?,?,?)";
                    PreparedStatement pps = connection.prepareStatement(insertSql);
                    pps.setString(1, sno);
                    pps.setString(2,"english");
                    pps.setString(3,englishScore);
                    int n = pps.executeUpdate();
                }
            if(physicsScore!=null){
                    insertSql = "insert into sc values(?,?,?)";
                    PreparedStatement pps = connection.prepareStatement(insertSql);
                    pps.setString(1, sno);
                    pps.setString(2,"physics");
                    pps.setString(3,physicsScore);
                    int n = pps.executeUpdate();
                }
```

如上述代码所示，为了转换成键值对，只需要每一条执行后得出的语句的每一个成绩进行判断，如果不为空，添加到结果中即可。

3 课程总结

本学期的数据库实验主要在 Educoder 平台上实现，通过编写代码通过测评的方式来通过实验，整个实验共分为 12 个大关卡，通过每个关卡中的每个小实验可以获得分数。具体到我个人的这次情况，除了数据查询的第 16 关没完成外，其余关卡全部通过，最后成绩为 114 分(如果没有记错的话，因为现在学生端不能看数据统计了)。

首先在数据库、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这一部分，我学会了使用 MySQL 语言实现了数据库一些基本而又非常有效的工作，也认识到数据库在当今这样一个大数据时代所发挥的重要性和极远的发展前景。而数据查询，数据插入、删除与修改等数据处理相关任务又让我们学习如何去修改已经存在的数据。之后的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又将安全这一在数据库领域内极其重要的话题展现在我们面前，也让我们看到了如何实现最基本的数据库防护，但现实世界的数据库所面临的挑战肯定要危险的多。之后的数据库设计与实现也让我们学会根据实际需求去自己设计一个尽可能满足要求的数据库，也为以后可能面临的需求做了一个良好的铺垫。至于最后的使用 JAVA 语言来进行数据库应用系统，也让我们了解如何在高级语言中操作数据库，这种方式可能比编写 MySQL 语句或在命令行运行有更多的应用场景。

总的来说，整个数据库实验的顺序大体跟数据库理论课程的顺序相契合，并且难度也有易有难，如果碰到实在比较棘手的也可以选择暂时跳过，后面再来完成即可。通过测评的方式，方便学生自己找寻代码的问题，也减轻了老师的压力。通过数据库实验，对于数据库的许多操作也已经了然于胸，我也期待后面将所学的知识运用到真正的工程场景之中。

附录

附录中附上在正文中没有贴的代码。

2.2.17 购买基金的高峰期

```
 select distinct s1.pro_purchase_time, s1.total_amount
from (
    select pro_purchase_time, total_amount
    from(
    select pro_purchase_time, sum(pro_quantity*f_amount) as total_amount
    from(select pro_pif_id, pro_purchase_time, pro_quantity
            from property
            where pro_type=3
    ) as s1 join fund on pro_pif_id=f_id
    where pro_purchase_time between '2022-2-7' and '2022-2-28'
    group by pro_purchase_time
    order by pro_purchase_time
)as temp
where total_amount>1000000
)as s1,(
    select pro_purchase_time, total_amount
    from(
        select pro_purchase_time, sum(pro_quantity*f_amount) as total_amount
        from(
        select pro_pif_id, pro_purchase_time, pro_quantity
        from property
        where pro_type = 3)
    as s1 join fund on pro_pif_id = f_id
    where pro_purchase_time between '2022-2-7' and '2022-2-28'
    group by pro_purchase_time
    order by pro_purchase_time
) as temp
where total_amount>100000
) as s2,(
    select pro_purchase_time, total_amount
    from(
        select pro_purchase_time, sum(pro_quantity*f_amount) as total_amount
        from(select pro_pif_id, pro_purchase_time, pro_quantity
            from property
            where pro_type=3
            ) as s1 join fund on pro_pif_id = f_id
        where pro_purchase_time between '2022-2-7' and '2022-2-28'
        group by pro_purchase_time
```

```
            order by pro_purchase_time
        ) as temp
        where total_amount > 100000
) as s3

where ((dayofmonth(s1.pro_purchase_time)+1 = dayofmonth(s2.pro_purchase_time) or
(weekday(s1.pro_purchase_time)=4 and dayofmonth(s1.pro_purchase_time)+3 =
dayofmonth(s2.pro_purchase_time))) and
(dayofmonth(s2.pro_purchase_time)+1 = dayofmonth(s3.pro_purchase_time) or
(weekday(s2.pro_purchase_time)=4 and dayofmonth(s2.pro_purchase_time)+3 =
dayofmonth(s3.pro_purchase_time)))) or
((dayofmonth(s2.pro_purchase_time)+1 = dayofmonth(s1.pro_purchase_time) or
(weekday(s2.pro_purchase_time)=4 and dayofmonth(s2.pro_purchase_time)+3 =
dayofmonth(s1.pro_purchase_time))) and
(dayofmonth(s1.pro_purchase_time)+1 = dayofmonth(s3.pro_purchase_time) or
(weekday(s1.pro_purchase_time)=4 and dayofmonth(s1.pro_purchase_time)+3 =
dayofmonth(s3.pro_purchase_time)))) or
((dayofmonth(s2.pro_purchase_time)+1 = dayofmonth(s3.pro_purchase_time) or
(weekday(s2.pro_purchase_time)=4 and dayofmonth(s2.pro_purchase_time)+3 =
dayofmonth(s3.pro_purchase_time))) and
(dayofmonth(s3.pro_purchase_time)+1 = dayofmonth(s1.pro_purchase_time) or
(weekday(s3.pro_purchase_time)=4 and dayofmonth(s3.pro_purchase_time)+3 =
dayofmonth(s1.pro_purchase_time)))) order by pro_purchase_time;
```

2.3.1 插入多条完整的客户信息

```
insert into client
(c_id, c_name, c_mail, c_id_card, c_phone,c_password)
values
(1, "林惠雯", "960323053@qq.com", "411014196712130323", "15609032348", "Mop5UPkl"),
(2, "吴婉瑜", "1613230826@gmail.com", "420152196802131323","17605132307",
"QUTPhxgVNlXtMxN"),
(3, "蔡贞仪", "252323341@foxmail.com", "160347199005222323","17763232321", "Bwe3gyhEErJ7");
```