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：	本硕博 2201 班
学　　号：	U202215703
姓　　名：	张家万
指导教师：	袁平鹏

分数	
教师签名	

2024 年 6 月 18 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，基础内容涉及以下几个部分，并可结合实际对 DBMS 原理的掌握情况向内核设计延伸：

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课程平台中的第 1~14 实训任务，下面将重点针对其中的数据查询、存储过程与事务、触发器、用户自定义函数、并发控制与事务的隔离级别、数据库设计与实现、数据库应用开发任务阐述其完成过程中的具体工作。

2.1 基于金融应用的数据查询(Select)

本节由浅入深围绕 select 语句在不同的应用场景下展开。本节已完成所有关卡。

2.1.1 既买了保险又买了基金的客户

可以查询买了保险的客户和买了基金的客户，二者用 and 取交集，排序用 order by。可用 exists 实现，如图 2.1 所示，也可用 in 实现，如图 2.2 所示。一种常见的错误是使用 pro_type=2 and pro_type=3，由于一条记录中 pro_type 只能取一个值，这样查询结果为空。注意 in 后的子查询与外层查询无关，每个子查询执行一次，而 exists 后的子查询与外层的查询有关，需要执行多次。

```
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;
```

图 2.1 exists 实现

```
select c_name, c_mail, c_phone
from client
where c_id in (select pro_c_id from property where pro_type=2)
    and c_id in (select pro_c_id from property where pro_type=3)
order by c_id;
```

图 2.2 in 实现

2.1.2 商品收益的众数

本关任务：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。

在 property 表中基于 pro_income 分组，利用 count 函数统计数量，如果某个分组数量大于等于所有分组，则为众数。此处用到嵌套查询和 group by 子句。代码如图 2.3 所示。注意 all()不能换成 max()，因为 max 函数中不能出现查询子句。

```sql
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);
```

图 2.3 商品收益的众数

2.1.3 投资总收益前三名的客户

利用 join 对用户表 client 和资产表 property 做自然连接，然后按 c_id 分组统计总收益（资产状态为可用）并降序排序，利用 limit 3 输出前 3 民用户即可。代码如图 2.4 所示。注意 limit N 返回前 N 条记录，offset M 跳过 M 条记录，limit N,M 表示从第 N 条记录开始返回 M 条记录。join 默认为 inner join 是等值连接，区别于 left join（获取左表所有记录，右表无匹配记录时值为 NULL）和 right join（与 left join 相反）。

```sql
select c_name,c_id_card, sum(pro_income) total_income
from client join  property on c_id=pro_c_id
where pro_status='可用'
group by c_id
order by total_income desc
limit 3;
```

图 2.4 投资总收益前三名的客户

2.1.4 客户理财、保险与基金投资总额

可以查询查询客户的每笔理财投资金额，保险投资金额，基金投资金额（注意投资金额的计算方式，用产品价格乘购买数量）和储蓄卡总余额，构成 5 张表用 union all 将查询结果进行合并，这里可以先求和，但为了代码的简洁性，我选择合并后一起求和。然后将 client 表与合并后的表做左连接，再按 c_id 分组统计总资产，最后排序即可。代码如图 2.5 所示。注意 ifnull 和 if 语句的使用。对于 bank_card 若为储蓄卡则直接取 b_balance，否则表示信用卡透支金额，应该取 -b_balance。此外集合操作会自动去除重复元素，若要保留重复元素则必须用 all 关键字指明，那么 union all 不会移除重复的行，而 union 会移除重复的行，由于同一客户不同类型的金额数可能相等，故应该使用 union all。

```sql
select c_id,c_name,ifnull(sum(amount),0) as total_property
from client
  left join(
    (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
    -- 这里可以用b_c_id 而无需改为pro_c_id,因为是并在前面的表中
    -- 注意这里amount的计算
    from bank_card
    group by b_c_id)
  )pro on c_id=pro.pro_c_id
group by c_id
order by c_id;
```

图 2.5 客户总资产

2.1.5 持有完全相同基金组合的客户

可以先查询每个客户拥有的基金组合的到一张表，这个表会被使用两次，来找到持有完全相同的基金组合的客户，代码如图 2.6 所示。使用公共表表达式（CTE) 来简化查询，为了区分两个表，需要使用别名 t1, t2。group_concat 函数将分组内的值拼接成一个字符串，并使用指定分割符来分割这些值（默认为逗号），进行排序来表示集合（与顺序无关），代码中使用该函数得到基金组合，这样可以很方便的判断客户是否持有相同基金组合，只需要比较该字符串是否相等即可。考虑到题目要表示的客户对(A,B)需要满足编号小的在前，故还需要添加条件 t1.c_id<t2.c_id。

```sql
with property_cte 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 t1.c_id c_id1, t2.c_id c_id2
from property_cte t1,property_cte t2
where t1.c_id < t2.c_id and t1.f_id = t2.f_id;
```

图 2.6 持有完全相同基金组合的客户

2.1.6 购买基金的高峰期

考察嵌套查询，排序，分组，变量的使用和对时间的处理，综合性较强，关卡目标和代码如图 2.7 所示。

最内层查询 t1 得到 2022 年 2 月各交易日的总购买金额，以及该日对应第几个工作日。datediff 计算两个日期间的天数差值，这里可表示为 2022 年的第几天，week 获得日期位于所处年份的第几周，每周跳过周六周日这两个非交易日。@用来声明一个变量，则列 row 表示日期对应一年中的第几个工作日。

内层查询 t2 为 t1 中每个交易日分配一个顺序编号，并筛选出总购买金额大于等于 100 万的。

中层查询 t3 中使用窗口函数 count(*) over(partition by t2.workday - t2.rownum) 来计算每个分组中连续交易日的数量（式中差值一样的可以看作连续交易日，故以该差值来分组）。

最外层查询 t4 从 t3 中筛选出至少连续 3 个交易日总金额超过 100 万（含）即高峰期对应的日期和总购买金额。

```sql
-- 17 查询2022年2月购买基金的高峰期。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(含)，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,但不能保证每个交易日都有投资者购买基金。2022年春节假期之后的第1个交易日为2月7日,周六和周日是非交易日，其余均为交易日。请列出高峰时段的日期和当日基金的总购买金额，按日期顺序排序。总购买金额命名为total_amount。
--     请用一条SQL语句实现该查询：
select t3.t as pro_purchase_time, t3.amount as total_amount
from
    (select *,count(*) over(partition by t2.workday - t2.rownum) cnt
    -- 计算的这个差值一样时可以看作是连续的交易日
    from
        (select *,row_number() over(order by workday) rownum
        from
            -- 2022.02交易日总购买金额，workday表示第几个工作日
            (select pro_purchase_time t, sum(pro_quantity * f_amount) amount,
                @row :=datediff(pro_purchase_time, "2021-12-31")
                -- @声明一个变量
                -2*week(pro_purchase_time) workday
            from property, fund,(select @row) a
            where pro_purchase_time like "2022-02-%"
                and pro_type=3 and pro_pif_id=f_id
            group by pro_purchase_time) t1
        where amount>=100000) t2
    ) t3
where t3.cnt>=3;
```

图 2.7 购买基金的高峰期

2.2 存储过程与事务

本节的 3 个关卡分别涉及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。本节已完成所有关卡，下面分析后两个关卡。

2.2.1 使用游标的存储过程

任务目标 ：利用使用游标的存储过程得到符合条件的排版结果表。

代码如图 2.8 所示，对于存储过程，需要用 delimiter 在创建前后重新定义分割符号。创建时先声明各种变量，声明游标后还需至少定义一个 not found 的 handler(处理例程)来进行抛出 not found 异常时的处理。本任务需要灵活的打开关闭游标和使用游标进行遍历。可以分开考虑医生和护士的排班，各自用一个游标来获取。对于护士每次取两个即可，对于医生，若在周末遍历到主任，需要记录该主任并调班至下次周一，由排在主任后面的医生依次替补。代码在插入排版结果后使用 date_add 实现日期的递增。

```
delimiter $$
create procedure sp_night_shift_arrange(in start_date date, in end_date date)
begin
declare done, waitdir int default false; /*done标记游标遍历完毕，waitdir标记是否调班*/
declare nowdate date;
declare waitdr, doctor,nurse1,nurse2 char(30);/*waitder表示调班的主任*/
declare type int;

declare drlist cursor for select e_name,e_type from employee where e_type < 3;
declare nrlist cursor for select e_name from employee where e_type = 3;
declare continue handler for not found set done = true;

open drlist;open nrlist;

set nowdate = start_date;
while nowdate <= end_date do
    /*安排医生*/
    if weekday(nowdate) < 5 and waitdir then/*主任夜班调至周一*/
        set doctor = waitdr, waitdir = false;
    else
        fetch drlist into doctor, type;
        if done then
        -- 每次遍历完成一遍后done为true 需要重启表格重新开始遍历
            close drlist;open drlist;
            fetch drlist into doctor, type;
            set done = false;
        end if;
        if weekday(nowdate) >= 5 and type = 1 then/*主任遇到周末*/
            set waitdir = true, waitdr = doctor;
            fetch drlist into doctor, type;/*后面的医生递补*/
            if done then
                close drlist;open drlist;
                fetch drlist into doctor, type;
                set done = false;
            end if;
        end if;
    end if;
    /*安排第一名护士*/
    fetch nrlist  into nurse1;
    if done then
        close nrlist;open nrlist;
        fetch nrlist into nurse1;
        set done = false;
    end if;
    /*安排第二名护士*/
    fetch nrlist into nurse2;
    if done then
        close nrlist;open nrlist;
        fetch nrlist into nurse2;
        set done = false;
    end if;
    /*插入排班结果*/
    insert into night_shift_schedule values (nowdate, doctor, nurse1, nurse2);
    set nowdate = date_add(nowdate, interval 1 day);/*日期递增一天*/
end while;
end$$
delimiter ;      zhangjiawan2022, 2周前 • Update
```

图 2.8 使用游标的存储过程

2.2.2 使用事务的存储过程

本关任务：实现一个转账操作的存储过程。

代码如图 2.9 所示，可以先假设输入都是合法的，完成转账操作对转出账户和转入账户数据的更新，然后再做检查，若不合法则将返回值置为 0 并回滚事务，若合法返回值为置为 1 并提交事务。当然也可以先做合法性检查，如果不合法则返回值置 0，并直接利用 leave 退出事务，如果合法则更新数据并让返回值置 1。

```
-- 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
set autocommit = off; /*开启自动事务模式*/
start transaction; /*开启事务*/
    update bank_card set b_balance = b_balance-amount where
    b_number = source_card_id and b_c_id = applicant_id and b_type
    = "储蓄卡";
    update bank_card set b_balance = b_balance+amount where
    b_number = dest_card_id and b_c_id = receiver_id and b_type =
    "储蓄卡";
    update bank_card set b_balance = b_balance-amount where
    b_number = dest_card_id and b_c_id = receiver_id and b_type =
    "信用卡";

    /*检查转出转入账户的有效性，若无效则事务回滚，有效则提交 */
    if not exists(select * from bank_card where b_number =
    source_card_id and b_c_id=applicant_id and b_type = "储蓄卡"
    and b_balance > 0) then
        set return_code = 0;
        rollback;
    elseif not exists (select * from bank_card where b_number =
    dest_card_id and b_c_id = receiver_id) then
        set return_code = 0;
        rollback;
    else
        set return_code = 1;
        commit;
    end if;
set autocommit = true; /*关闭自动事务模式*/
END$$
delimiter ;
/*  end  of  your code  */        zhangjiawan2022, 4周前 • Update
```

图 2.9 使用事务的存储过程

2.3 触发器

本节的一个关卡涉及触发器的使用，本节已完成所有关卡。

2.3.1 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

本关任务：为资产表 property 编写一个触发器，以实现任务所要求的完整性业务规划。

根据任务要求应该声明 before insert on property 类型的触发器，然后根据任务要求判断并设置相应的报错信息即可，若报错信息为空则说明没有错误。代码如图 2.10 所示。

注意触发器中可以使用两类特殊表，delete 触发器可以访问 old 表，其内容为被 delete 掉的数据；insert 触发器可以访问 new 表，其内容为 insert 的新数据；update 触发器可以访问 old 表和 new 表，前者保存修改前数据，后者保存修改后内容。本关的 insert 触发器使用到了 new 表。触发器可以方便的实现 primary key，foreigh key，check 等约束无法实现的复杂业务规则。

```
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 $$
/*before插入前触发做检查*/
create trigger before_property_inserted before insert on property
for each row
begin
declare msg varchar(50); /*注意要先声明变量*/
/*做合法和存在性检查，用signal sqlstate抛出异常并设置错误信息*/
if new.pro_type = 1 and not exists (select * from finances_product where p_id = new.
pro_pif_id) then
    set msg = concat("finances product #", new.pro_pif_id, " not found!");
    signal sqlstate '45000' set message_text = msg;/*45000指用户定义的待处理异常*/
elseif new.pro_type = 2 and not exists (select * from insurance where i_id = new.pro_pif_id)
then
    set msg = concat("insurance #", new.pro_pif_id, " not found!");
    signal sqlstate '45000' set message_text = msg;
elseif new.pro_type = 3 and not exists (SELECT * FROM fund WHERE f_id = new.pro_pif_id) then
    set msg = concat("fund #", new.pro_pif_id, " not found!");
    signal sqlstate '45000' set message_text = msg;
elseif new.pro_type not in (1,2,3) then
    set msg = concat("type ", new.pro_type, " is illegal!");
    signal sqlstate '45000' set message_text = msg;
end if;
end$$
delimiter ;
```

图 2.10 触发器的使用

2.4 用户自定义函数

本节的 1 个关卡涉及用户自定义函数的定义和使用。

2.4.1 创建函数并在语句中使用它

本关任务：编写一个依据客户编号计算其在本金融机构的存贮总额的函数，并在 select 语句中使用这个函数。

定义和使用的代码如图 2.11 所示。实现逻辑较为简单，只需要留意相关语法即可，注意返回类型的声明用的是 returns 而不是 return。

利用函数，可以方便的完成复杂的计算，增加代码的复用性和可读性。其一旦定义就可像内部函数一样使用，比如用在 select 列表、表达式以及 where 子句的条件中。

此外函数分为标量函数和表函数，本关使用前者返回一个值，而后者可以返回表，二者的语法也有所区别。

```
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 $$
create function get_deposit(client_id int)
returns numeric(10,2)
begin
  declare total_dep int;
    select sum(b_balance) into total_dep
    from bank_card
    where b_c_id = client_id and b_type = "储蓄卡";
    return total_dep;
end$$
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 c_id_card,c_name,get_deposit(c_id) as total_deposit
from client
where get_deposit(c_id) >= 1000000
order by total_deposit desc;
```

图 2.11 函数的定义与使用

2.5 并发控制与事务的隔离级别

本关的 6 个关卡涉及数据库中并发控制与隔离级别的相关内容，包括隔离级别的设置，事务的开启、提交、回滚等，此外通过增加延时实现了读脏、不可重复读、幻读场景，还考察了主动加锁以及可串行化的知识。本节所有关卡均已完成，下面以不可重复读的模拟为例进行分析。

2.5.1 不可重复读

首先介绍 MySQL 的事务的隔离级别，其从低到高分为一下四级：

1) 读未提交（READ UNCOMITTED）
2) 读已提交（READ COMMITTED）
3) 可重复度（REPEATABLE READ）
4) 可串行化（SERIALIZABLE）

低隔离级别可以支持更高的并发处理，同时占用的系统资源更少，但可能产生数据不一致的情形也更多一些。MySQL 事务隔离级别及其可能产生的问题如表 2.1 所示。该表说明，最低的隔离级别不能避免读脏、不可重复度和幻读，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，但代价是并发度最低。

表 2.1 MySQL 事务隔离级别说明

隔离级别	读脏	不可重复读	幻读
READ UNCOMITTED	√	√	√
READ COMMITTED	×	√	√
REPEATABLE READ	×	×	√
SERIALIZABLE	×	×	×

本关为模拟不可重复读现象，可以将事务隔离级别设置为 read committed，可已根据时刻的提示添加 set @n = sleep( )语句来进行延时，使得执行到该语句时处于对应的时刻。事务 1 代码如图 2.12 所示，事务 2 代码如图 2.13 所示。其中不可重复读产生的原因，是事务 t2 的两次读取之间，另一个事务 t1 修改了 t2 读取的数据，导致 t2 两次读取的结果不一致。

```
-- 事务1:
-- 请设置适当的事务隔离级别
set session transaction isolation level read committed;
-- 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(1);
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(3);
insert into result
select now(), 1 t, tickets from ticket where flight_no = 'CZ5525';
```

图 2.12 不可重复读（事务 1）

```
-- 事务2
-- 请设置适当的事务隔离级别以构造不可重复读
set session transaction isolation level read committed;
start transaction;
-- 时刻2 - 事务2在事务1读取余票之后也读取余票
-- 添加代码，确保事务2的第1次读发生在事务1读之后，修改之前
set @n = sleep(1);
insert into result
select now(),2 t, tickets from ticket where flight_no = 'CZ5525';
-- 时刻4 - 事务2在事务1修改余票但未提交前再次读取余票，事务2的两次读取结果应该不同
-- 添加代码，确保事务2的读取时机
set @n = sleep(2);
insert into result
select now(), 2 t, tickets from ticket where flight_no = 'CZ5525';
-- 事务2立即修改余票
update ticket set tickets = tickets - 1 where flight_no = 'CZ5525';
-- 时刻5 - 事务2 读取余票（自己修改但未交的结果）:
set @n = sleep(1);
insert into result
select now(), 2 t, tickets from ticket where flight_no = 'CZ5525';
commit;
```

图 2.13 不可重复读（事务 2）

2.6 数据库设计与实现

本节的 3 个关卡涉及数据库设计与实现相关内容，包括从概念模型到 MySQL 实现、E-R 图的构建，建模工具的使用等。本节已完成全部关卡。

2.6.1 从概念模型到 MySQL 实现

本关根据机票订票系统概念模型 ER 图，逐个进行各个实体表的创建，并按照要求给定主码，自动增加（auto_increment），非空，默认值等约束。再根据 ER 图中的关系为表添加外码约束，即对于 n:1 的实体，需要在 n 边的实体表创建时添加对应 1 边的外码约束，本关并未涉及 n:m 的实体关系。对于涉及出发机场和到达机场的外码需做例外处理以防同名，前者命名为`from`，后者命名为`to`，注意符号不是单引号。更具体的代码实现从略。

2.6.2 从需求分析到逻辑模型

本关任务：根据应用场景需求描述，完成 ER 图，并转换成关系模式。

根据任务描述转换得到的 ER 图如图 2.14 所示。需要明确各实体集的属性和主码约束，根据实体间关系来确定外码约束。

该 ER 图对应的关系模式描述如图 2.15 所示。

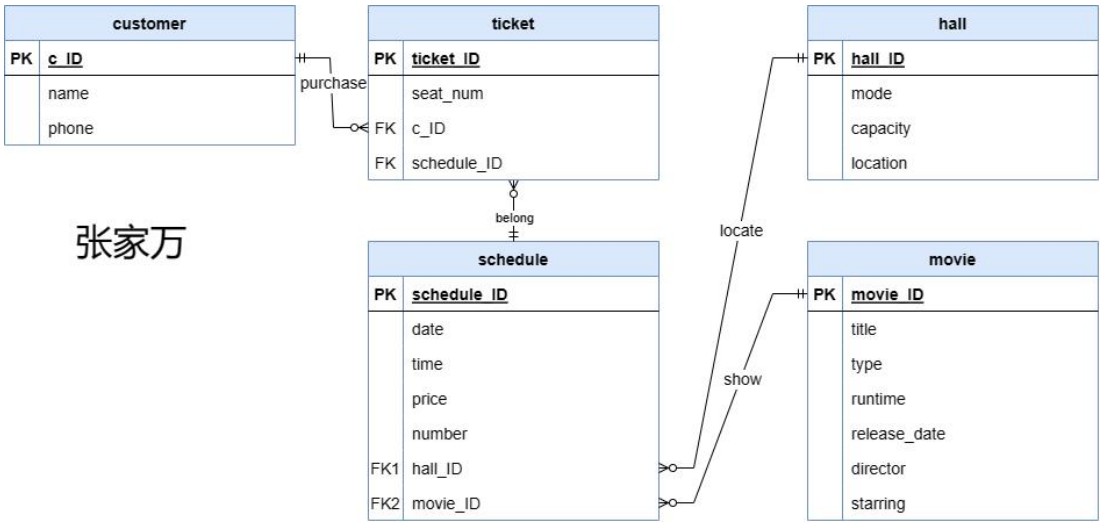

图 2.14 影院管理系统 ER 图

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

图 2.15 关系模式描述

2.6.3 建模工具的使用

用 MySQL Workbench 打开已经建好的模型文件 rbac.mwb，依次点击菜单 database、Forward engineering，对弹窗进行进一步操作，即可自动导出 SQL 脚本，由此实现从 ER 图向 SQL 脚本的转变，创建含对应关系模式的数据库。

2.6.4 制约因素分析与设计

从实际问题的建模到数据库的概念模型和逻辑模型的构建过程中，需要考虑若干制约因素。以第一关的机票订票系统为例，考虑到了以下因素：

1) 考虑到系统权限的不同，用户 user 分为了两类，普通股用户可以订票，管理用户有权限维护和管理整个系统的运营，实现中将两类用户合并，用 admin_tag 标记区分。
2) 考虑到旅客实际情况，用户登录系统不一定是替自己买票，所以用户 user 和旅客 passenger 信息是分开存储，这样用户也可以为多人购买多张机票。
3) 对于航班，一般依航班常规调度表 flightschedule 为基准安排，但调度表不是一成不变，也不是每个既定的航班都实际起飞，也不总是按既定的时间起飞，故实飞航班是单独安排并记录得到 flight 表。

2.6.5 工程师责任及其分析

工程师的责任应该包含以下几点：

1) 社会方面，工程师应该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。
2) 安全方面，工程师应该尽可能考虑系统中存在的安全漏洞，安全性是所有系统用户关心的重要命题。
3) 科学发展方面，工程师应该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。

2.7 数据库应用开发(JAVA 篇)

本节的 7 个关卡从 JDBC 体系结构出发，灵活考察了使用 JAVA 进行数据库应用开发的基本知识。本节已完成全部关卡。

2.7.1 JDBC 体系结构和简单的查询

本关任务：查询 client 表中邮箱非空的客户信息，列出客户姓名，邮箱和电话。

完成本关需要了解 JDBC 的体系结构，核心组件及使用步骤。在本关的 Client 类中，先定义 JDBC 驱动程序名，数据库链接，用户名及登录密码的字符串，以方便在函数中直接使用。具体步骤如下：

使用 Java 的 Class.forName()方法，将驱动程序的类文件动态加载到内存中，并将其自动注册。

加载驱动程序后，使用 DriverManager.getConnection(String url, String user, String password)方法创建数据库连接对象。通过使用用户名和密码来建立连接，来提高数据库的安全性。

在使用 Statement 对象执行 SQL 语句前，需要使用 Connection 对象的 createStatement()方法创建 Statement 的一个实例。

创建 Statement 对象后，可以使用它来执行一个 SQL 语句，其中有三种执行方法：

1) boolean execute(String SQL)：如果可以检索到 ResultSet 对象，则返回一个布尔值 true; 否则返回 false。使用此方法执行 SQL DDL 语句或需要使用真正的动态 SQL 时。
2) int executeUpdate(String SQL)：返回受 SQL 语句执行影响的行数。使用此方法执行预期会影响多个行的 SQL 语句，例如 INSERT，UPDATE 或 DELETE 语句。
3) ResultSet executeQuery(String SQL)：返回一个 ResultSet 对象。当您希望获得结果集时，请使用此方法，就像使用 SELECT 语句一样。

本关选用第 3 个执行方法，使用 resultSet.next()遍历 ResultSet 输出查询到的客户信息即可。代码如图 2.16 所示。

除了 Statement，还有 PreparedStatement 对其接口进行扩展，可以动态的提供参数，且当语句被多次执行时效率比 Statement 更高。

```java
import java.sql.*;
You, 1秒钟前 | 2 authors (zhangjiawan2022 and one other)
public class Client {
  static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
  static final String DB_URL = "jdbc:mysql://127.0.0.1:3306/
  finance?useUnicode=true&characterEncoding=UTF8&useSSL=false&
  serverTimezone=UTC";
  static final String USER = "root";
  static final String PASS = "123123";

  public static void main(String[] args) {
    Connection connection = null;
    Statement statement = null;
    ResultSet resultSet = null;

    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"));
      }
    } catch (ClassNotFoundException e) {
      System.out.println("Sorry,can`t find the JDBC Driver!");
      e.printStackTrace();
    } catch (SQLException throwables) {
      throwables.printStackTrace();
    } finally {
      try {
        if (resultSet != null)
          resultSet.close();
        if (statement != null)
          statement.close();
        if (connection != null)
          connection.close();
      } catch (SQLException throwables) {
        throwables.printStackTrace();
      }
    }
  }
}
```

图 2.16 使用 JDBC 进行客户信息查询

2.7.2 用户登录

本关任务：编写客户登录程序，提示用户输入邮箱和密码，并判断真确性，给出适当的提示。

本关需要对输入进行查询，查询到结果时给出成功登录的提示，未查询到结果时提示错误信息。本关有多种实现方式，部分方式代码如图 2.17 所示。可以将字符串直接拼接到 sql 语句中并使用 Statement 进行查询，也可作为参数使用 PreparedStatement 进行查询。此外若使用 executeQuery()，可以使用 resultSet.next()

判断是否有查询结果，还可直接使用 execute()方法的返回值（bool 类型）判断是否有查询结果。

```
26          // 法一，把变量直接拼接到sql语句中
27          statement = connection.createStatement();
28          String sql = "select * from client where c_mail = '"+loginName+"' and
    c_password = '" + loginPass + "';";
29          resultSet = statement.executeQuery(sql);
30
31          // 法二，利用PreparedStatement
32          // String sql = "select * from client where c_mail = ? and c_password = ?";
33          // PreparedStatement ps = connection.prepareStatement(sql);
34          // ps.setString(1, loginName);
35          // ps.setString(2, loginPass);
36          // resultSet = ps.executeQuery();
37          if (resultSet.next())
38            System.out.println("登录成功。");
39          else
40            System.out.println("用户名或密码错误！");
```

图 2.17 用户登录

2.7.3 客户修改密码

本关任务：编写修改客户登录密码的方法。

前面讨论了对 JDBC 对数据库的查询，本关不仅需要查询，还需要进行对数据库的修改。为实现 passwd 方法，需要先进行查询判断用户是否存在，若存在，还需要判断密码是否正确，最后利用 update 语句修改密码，代码如图 2.18 所示。

```
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(parameterIndex:1, mail);
    ResultSet res = ps.executeQuery();
    if (res.next()) {
      if (password.equals(res.getString(columnLabel:"c_password"))) {
        sql = "Update client set c_password = ? where c_mail = ?";
        ps = connection.prepareStatement(sql);
        ps.setString(parameterIndex:1, newPass);
        ps.setString(parameterIndex:2, mail);
        ps.executeUpdate();
        return 1; // 密码修改成功
      } else
        return 3; // 密码不正确
    } else
      return 2; // 用户不存在
  } catch (SQLException e) {
    e.printStackTrace();
  }
  return -1;// 程序异常
}
```

图 2.18 客户修改密码

3 课程总结

本课程实验围绕 MySQL 展开了对于数据库各功能的实践，在本次实验中，我完成了数据库、表与完整性约束定义，表结构与完整性约束的修改，数据查询、数据的插入、修改与删除，视图，存储过程与事务，触发器，用户自定义函数，安全性控制，并发控制与事务的隔离级别，介质故障与数据库恢复，数据库的设计与实现等 14 个实训实验，总计 68 个关卡。

本课程实验是对数据库的基础知识与实际运用的一次较好的实践，在本次实验中，我巩固了课堂中学习到的 mysql 的基本语法如增删改查，掌握了对于约束、存储过程、函数、触发器、游标等数据对象的管理与编程，同时通过查阅资料，学习了一些更为复杂的用法，来实现更为复杂的查询，以及实现 mysql 对于安全性控制、并发控制以及数据库恢复的支持，感受到 sql 的处理灵活与功能强大。此外在数据库设计与实现中，我掌握如何将实际问题一步步分析建模，从概念模型到逻辑模型最后到代码实现，也了解了数据库工程师的职责。在数据库应用开发中，我看到了 sql 的嵌入式的特点，也看到了开发过程之中代码安全的重要性。

本次实验内容完整充实，能由易到难、一步步引导学生了解和学习数据库的特点与功能，让我获益匪浅。对于最后两个实训存储管理和索引管理，由于综合性很强，工作量较大，实践起来有困难，我仅作了了解，希望之后能给出更加详细的指导，来帮助学生完成实验。最后，衷心感谢课程组老师对实验内容的精心打造以及实验过程中的悉心指导！

附录