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：	CS2011
学　　号：	U202011675
姓　　名：	徐锦慧
指导教师：	潘鹏

分数	
教师签名	

2022 年 12 月 17 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)

本节围绕创建数据库、表、完整性约束展开，主要包括 create 语句、alter 语句、constraint 完整性约束语句、check 语句的学习使用。

2.1.1 创建数据库

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

方法：使用 create database dbname 语句创建数据库。代码如下：

```
create database beijing2022;
```

2.1.2 创建表及表的主码约束

任务：创建数据库 TestDb，在 TestDb 下创建表 t_emp。

方法：首先使用 create database 语句创建数据库 TestDb，接着在 TestDb 下使用 create table 语句创建表 t_temp，并对主码添加 primary key 约束。代码如下：

```
create database TestDb;
use TestDb;
create table t_emp(
    id int primary key,
    name varchar(32),
    deptId int,
    salary float);
```

2.1.3 创建外码约束(Foreign Key)

任务：创建两个表，为表定义主键，并给表 staff 创建外键。

方法：首先创建 MyDb 数据库，在其中创建 dept 表和 staff 表，并用 constraint 语句为 staff 表创建外键。关键代码如下：

```
create table staff(
    staffNo int primary key,
    ......
    constraint FK_staff_deptNo foreign key(deptNo) references dept(deptNo));
```

2.1.4 Check 约束

任务：数据库 MyDb 中创建表 products，并分别实现对品牌和价格的约束。

方法：创建所给表并添加 check 约束语句，关键代码如下：

```
create table products(
    pid char(10) primary key,
    ......
    constraint CK_products_brand check(brand in ('A','B')),
    constraint CK_products_price check(price > 0));
```

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)

本节围绕表和完整性约束的修改展开，主要包括使用 alter 语句修改表名、添加/删除/修改字段、添加/删除/修改约束。

2.2.1 修改表名

任务：将表名 your_table 更改为 my_table。

方法：使用“alter table <旧表名> rename to <新表名>”语句。关键代码如下：

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

2.2.2 添加与删除字段

任务：在 orderDetail 表中删除 orderDate 字段，添加列 unitPrice。

方法：使用“alter table <表名> drop [column] <列名>”语句删除 orderDate 列，使用“alter table <表名> add [column] <列名> <数据类型> [列约束]”语句添加 unitPrice 字段。关键代码如下：

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

2.2.3 修改字段

任务：将 QQ 号的数据类型改为 char(12)，将列名 weixin 改为 wechat。

方法：使用“modify [column] <列名> <数据类型> [列约束]”修改 QQ 号数据类型，使用“rename column <旧列名> to <新列名>”语句修改 weixin 名。关键代码如下：

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

2.2.4 添加、删除与修改约束

任务：为数据库中的表添加对应的主码、外码、check、unique 约束。

方法：使用 primary key 添加主码约束，使用 foreign key 添加外码约束，使用 check、unique 添加对应的约束。关键代码如下：

```
    alter table Staff add primary key(staffNo);
    alter table Dept add constraint FK_Dept_mgrStaffNo  foreign key (mgrStaffNo)
references Staff(staffNo);
    alter table Staff add constraint FK_Staff_dept  foreign key (dept)
references Dept(deptNo);
    alter table Staff add constraint CK_Staff_gender check(gender in ('F','M'));
    alter table Dept add constraint UN_Dept_deptName unique(deptName);
```

2.3 数据查询(Select)之一

本节围绕 Mysql 的 select 查询语句展开，以银行的一个模拟金融应用为背景，内容由浅到深，循序渐进。

2.3.1 金融应用场景介绍,查询客户主要信息

任务：查询所有客户的名称、手机号和邮箱信息。查询结果按客户编号排序。

方法：使用 select 语句查询，用 order by 子句对查询结果排序。代码如下：

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

2.3.2 邮箱为 null 的客户

任务：查询客户表(client)中邮箱信息为 null 的客户的编号、名称、身份证号、手机号。

方法：在 where 子句中，使用 is null 判断属性是否为空。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

任务：查询既买了保险又买了基金的客户的名称、邮箱和电话，结果依 c_id

排序。

方法：使用带 in 谓词的子查询，先查询买了保险的客户，再查询买了基金的客户，通过 and 逻辑连接。代码如下：

```
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;
```

2.3.4 办理了储蓄卡的客户信息

任务：查询办理了储蓄卡的客户信息，查询结果结果依客户编号排序。

方法：将 client 表和 bank_card 表进行用户 id 的等值连接，并令银行卡类型为储蓄卡，最后对查询结果进行排序。代码如下：

```
select c_name,c_phone,b_number
from client,bank_card
where c_id=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

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

方法：使用 between…and 条件查找满足条件的产品，使用 desc 让查询结果降序排列。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

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。

方法：在 property 表中基于 pro_income 分组，利用 count()函数统计商品收益出现次数，如果某个分组数量大于等于所有分组，则为众数。代码如下：

```
select pro_income,count(pro_income) as presence
from property
group by pro_income
```

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

2.3.7 未购买任何理财产品的武汉居民

任务：查询身份证隶属武汉市且没有买过任何理财产品的客户信息。依客户编号排序。

方法：武汉居民的身份证号开头为“4201”，因此利用“like 4201%”即可筛选出武汉居民，再利用 not exists 查询是否购买过理财产品。代码如下：

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

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：查询当前总的可用资产收益前三名的客户信息，按收益降序输出，总收益命名为 total_income，不考虑并列排名情形。

方法：对 client 和 property 表在用户 id 上进行等值连接，投资状态为“可用”。按用户 id 进行分组，使用聚集函数 sum()统计收益和。使用 desc 将结果按收益和降序输出，limit 用于获取前 3 个元组。完整代码如下：

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

该关卡任务已完成，实施情况本报告略过。

2.3.12 客户理财、保险与基金投资总额

该关卡任务已完成，思路与下一关类似，实施情况本报告略过。

2.3.13 客户总资产

任务：查询所有客户的编号、名称和总资产，总资产命名为 total_property。

方法：首先将表 client、表 property、表 finance_product、表 insurance、表 fund 利用对应的产品 id 和产品类型连接起来，再分储蓄卡和用户卡将 bank 表与上述表连接起来，连接条件为客户 id 相等。接着利用 c_id, c_name 进行分组，通过 sum(ifnull(a,b))计算出结果，其中总资产=储蓄卡额+投资总额+投资总收益-信用卡透支金额。代码如下：

```
    select c_id,c_name,sum(ifnull(pro_quantity,0)*(ifnull(p_amount,0)+
ifnull(i_amount,0)+ifnull(f_amount,0))+ifnull(pro_income,0))+ifnull(card,0)
as total_property
    from client left outer join property on (pro_c_id=c_id)
    left outer join finances_product on (pro_pif_id=p_id ) and pro_type=1
    left outer join insurance on (pro_pif_id=i_id ) and pro_type=2
    left outer join fund on (pro_pif_id=f_id ) and pro_type=3
    left outer join
        (select b_c_id,sum(if(b_type='储蓄卡',b_balance,0)-if(b_type='信用卡
',b_balance,0)) as card from bank_card
        group by b_c_id) as bank on (b_c_id=c_id)
    group by c_id,c_name;
```

2.3.14 第 N 高问题

任务：查询每份保险金额第 4 高保险产品的编号和保险金额。

方法：在表 insurance 中利用 distinct 选出不同的保险金额并降序，再利用“limit 3, 1”获取第四高的保险金额，最后在外部查询中通过判断保险金额是否等于该金额。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

任务：查询资产表中客户编号、客户基金投资总收益、基金投资总收益的两种不同排名。

方法：首先在 from 子句中查询出客户基金投资总收益，接着用 rank()函数查询名次不连续的排名，用 dense_rank()函数查询名次连续的排名。具体代码省略。

2.3.16 持有完全相同基金组合的客户

任务：查找基金组合完全相同的不同客户对。

方法：首先在 from 子句中放置两个表 property p1，property p2，接着在 where 条件中令 p1.pro_c_id < p2.pro_c_id，以此保证两个客户不是同一人，再用 not exist 和嵌套查询确保客户 1 有的基金客户 2 都有，且客户 2 有的基金客户 1 也都有，即两人拥有完全相同的基金组合，最后在 select 子句中使用 distinct 避免查询结果重复。具体代码省略。

2.3.17 购买基金的高峰期

任务：查询 2022 年 2 月购买基金的高峰期。

方法：首先对表 property 和表 fund 进行等值连接，选取交易日期属于 2022 年 2 月交易日的元组组成派生表。然后查找 3 个连续的交易日中总金额大于 100 万的元组。最后使用 union 整合所有满足要求的元组，并用 distinct 排除重复元组。具体代码省略。

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：以日历表格式列出 2022 年 2 月每周每日基金购买总金额。

方法：首先在 where 子句中筛选出类型为基金且购买日期属于 2 月的产品，接着通过 week()和 weekday()函数获取购买基金日期属于一年的第几周、一周的周几，再用 group by 按照 week_of_trading 分组，对 weekday()的结果使用 if 语句判断金额放在哪一列计算。代码如下：

```
    select week(pro_purchase_time)-5 as week_of_trading,
sum(if((weekday(pro_purchase_time))=0,pro_quantity*f_amount,null)) as
Monday,sum(if((weekday(pro_purchase_time))=1,pro_quantity*f_amount,null)) as
Tuesday,sum(if((weekday(pro_purchase_time))=2,pro_quantity*f_amount,null))
as Wednesday,sum(if((weekday(pro_purchase_time))=3,pro_quantity*f_amount,
null)) as Thursday,sum(if((weekday(pro_purchase_time))=4,pro_quantity*
f_amount,null)) as Friday
    from property,fund
    where pro_type=3 and pro_pif_id=f_id and pro_purchase_time >= '2022-02-07'
and pro_purchase_time <= '2022-02-28'
    group by week_of_trading
    order by week_of_trading;
```

2.4 数据查询(Select)之二

本节围绕 Mysql 的 select 查询语句展开，是第 3 节的扩展延续，难度适中。

2.4.1 查询销售总额前三的理财产品

任务：查询 2010 年和 2011 年每年销售总额前 3 名的年份、销售总排名、产品编号、理财总额。

方法：首先在内查询中通过 like ‘2010%’查询出 2010 年的理财产品，用 year()函数计算购买日期的年份，用 sum()函数计算理财总额；接着在外查询中使用 rank()函数查询销售总排名，用 limit 3 取结果的前 3 名。对 2011 年的理财产品重复以上步骤，最终用 union 连接起来，得到查询结果。具体代码省略。

2.4.2 投资积极且偏好理财类产品的客户

任务：查询投资积极且偏好理财类产品的客户。

方法：首先将表 property、表 fund、表 finance_product 用对应的产品 id 和产品类型连接起来，再按照客户 id 进行分组，用 count(distinct p_id)>3 确保购买 3 种以上理财产品，用 count(distinct p_id)>count(distinct f_id)确保基金产品种类数小于理财产品种类数。代码如下：

```
select pro_c_id
from property left outer join fund on (pro_c_id=f_id) and pro_type=3
    left outer join finances_product on (pro_pif_id=p_id ) and pro_type=1
group by pro_c_id
having count(distinct p_id)>3 and count(distinct p_id)>count(distinct f_id)
order by pro_c_id asc;
```

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

任务：查找产品 14 的相似理财产品编号、购买客户总人数和相似度排名值。

方法：①找出持有产品 14 的数量最多的前 3 个用户：首先用 dense_rank()函数查找出所有用户购买产品 14 的数量排名，接着在外查询中令 rk<=3 找出前 3 个用户。②接着在外查询中用谓词 in 找出持有数量最多的前 3 个用户购买的所有理财产品。③最后在外查询中用 count()和 dense_rank()函数查找这些产品中被全体客户持有的总人数前 3 的产品，得到最终结果。具体代码省略。

2.4.5 查询任意两个客户的相同理财产品数

任务：查询任意两个客户之间持有的相同理财产品种数，并且结果仅保留相同理财产品数至少 2 种的用户对。

方法：首先在 from 中设置三张表为 property p1、property p2、property p3，p1.pro_c_id 和 p2.pro_c_id 用来代表 2 个被比较的客户 id，p3.pro_pif_id 表示两个客户都有的理财产品，通过谓词 in 来实现；接着用 count(distinct p3.pro_pif_id) 来计算相同理财产品数；最后用 having count(distinct p3.pro_pif_id)>1 筛选出相同理财产品数至少 2 中的结果。代码如下：

```
    select distinct p1.pro_c_id as pro_c_id, p2.pro_c_id as
pro_c_id,count(distinct p3.pro_pif_id) as total_count
    from property p1,property p2,property p3
    where p1.pro_c_id!=p2.pro_c_id and p3.pro_type=1 and p3.pro_pif_id in
        (select pro_pif_id from property where pro_type=1
        and pro_c_id=p2.pro_c_id) and p3.pro_pif_id in
        (select pro_pif_id from property where pro_type=1
        and pro_c_id=p1.pro_c_id)
    group by p1.pro_c_id,p2.pro_c_id
    having count(distinct p3.pro_pif_id)>1;
```

2.4.6 查找相似的理财客户

该关卡任务已完成，思路与上一关类似，实施情况本报告略过。

2.5 数据的插入、修改与删除(Insert,Update,Delete)

本节围绕数据的插入、删除与修改展开，主要涉及 insert、update、delete 语句的学习使用。

2.5.1 插入多条完整的客户信息

任务：向客户表中插入给定 3 条数据。

方法：使用 3 条 insert 语句完成，代码如下：

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

2.5.2 插入不完整的客户信息

任务：向客户表 client 插入一条数据不全的记录。

方法：通过在 values 前指定插入的列，来插入不完整的数据记录。代码如下：

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

2.5.3 批量插入数据

任务：将 new_client 表的全部客户信息插入到客户表(client)中。

方法：将 insert 语句中的 value 字段换为 select 查询语句。代码如下：

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

2.5.4 删除没有银行卡的客户信息

任务：删除 client 表中没有银行卡的客户

方法：首先通过 not in 谓词筛选出没有银行卡的客户，再使用 delete 语句删除数据。代码如下：

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

2.5.5 冻结客户资产

任务：将给定手机号码的客户的投资资产的状态设置为“冻结”。

方法：首先用 select 子句查询出给定手机号的客户，再用 update 语句修改数据。代码如下：

```
update property
set pro_status='冻结'
where pro_c_id=(select c_id from client where c_phone='13686431238');
```

2.5.6 连接更新

任务：根据 client 表中提供的身份证号，填写 property 表中对应的身份证号信息。

方法：首先用 select 子句从 client 表中查找出身份证号，再用 update 子句更新数据。代码如下：

```
    update property
    set pro_id_card=(select c_id_card from client where pro_c_id=c_id);
```

2.6 视图

本节围绕视图展开，要求学会使用 create 创建视图、使用 select 进行基于视图的查询。

2.6.1 创建所有保险资产的详细记录视图

任务：创建包含所有保险资产记录的详细信息的视图。

方法：首先用 select 语句查询所需数据，接着用“create view <视图名> as <子查询>”语句创建视图。代码如下：

```
    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,insurance,property
    where c_id=pro_c_id and pro_pif_id=i_id and pro_type=2;
```

2.6.2 基于视图的查询

任务：对视图 v_insurance_detail 进行分组统计查询。

方法：使用 select 语句，将视图当做表一样查询即可。代码如下：

```
    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;
```

2.7 存储过程与事务

本节围绕存储过程展开，分别要求学习实现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 基于视图的查询

任务：创建存储过程，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。

方法：首先声明变量 n, fibn, a, b，其中 n 用于递归计算斐波那契数，b 表示第 n 个斐波那契数的前 1 项斐波那契数，a 表示 b 的前一项斐波那契数；接着分别判断 m 是否是 0 或 1，并插入对应斐波那契数；最后在 while 循环中递归计算前 n 项斐波那契数，直到 m=n。关键代码如下：

```
declare n,fibn,a,b int;
set n = 2,a = 0,b = 1;
if m > 0 then
     insert into fibonacci values(0,0);
end if;
if m > 1 then
     insert into fibonacci values(1,1);
end if;
while m > n do
     set fibn = a + b;
     insert into fibonacci values(n,fibn);
     set a = b;
     set b = fibn;
     set n = n + 1;
end while;
```

2.7.2 使用游标的存储过程

任务：编写一个存储过程，自动安排某个连续期间的大夜班的值班表。

方法：首先定义医生游标、护士游标，分别打开游标并读取数据，当 now_date<=end_date 时进行 while 循环，分别考虑三种情况：科室主任轮到夜班、科室主任上周末轮到夜班、其他。具体流程如图 2.1 所示。

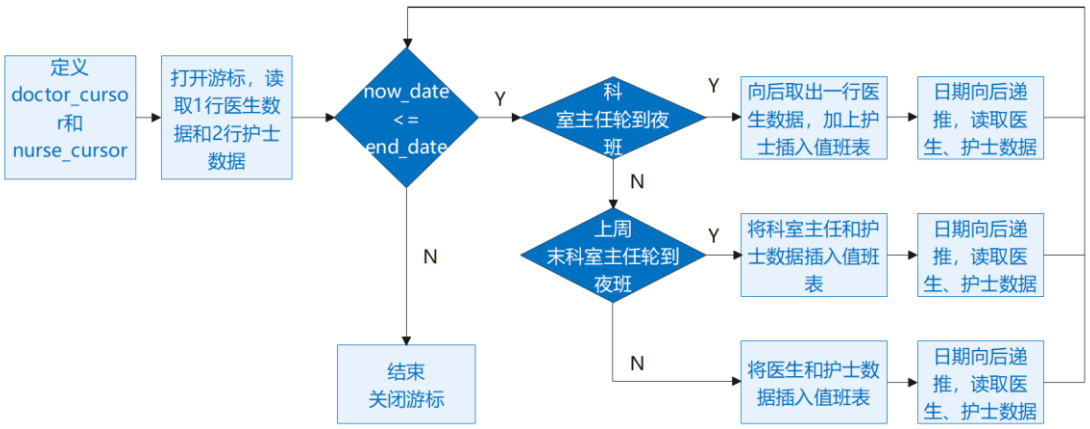

图 2.1 游标实现存储过程

2.7.3 使用事务的存储过程

任务：实现一个转账操作的存储过程，从一个帐户向另一个帐户转账。

方法：由于事务具有原子性，根据该特性可以指定一系列操作，判断这些操作是否满足要求，若满足要求则 commit 事务提交，否则 rollback 事务回滚。在进行自定义事务操作前，应将自动事务模式关闭，使用 start transaction 开启事务。

在此关卡中，首先进行相关更新，完成转账操作。随后进行 if 判断，当在 bank_card 表中没有转款人转出卡号码的储蓄卡，或收款人卡号在 bank_card 表中不存在时，返回参数置 0 并进行事务回滚。否则返回参数置 1 并进行事务提交。具体代码省略。

2.8 触发器

本节围绕触发器展开，要求学习在进行 insert、delete、update 操作时激活触发器，并进行数据完整性的检查。

2.8.1 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

任务：为表 property 编写一个触发器，以实现规定的完整性业务规则。

方法：首先创建触发器结构，并在使用 insert 插入数据之前判断数据的合法性，接着进行 4 种情况的判断：①若添加元组的类型不在 3 类投资内，设置对应报错提示；②若添加的元组类型为理财产品，且 finance_product 表中不含添加的产品 id，设置对应报错提示；③若添加元组的类型为保险，且 insurance 表中不含添加的保险 id，设置对应报错提示；④若添加的元组类型为基金，且 fund 表不含添加的基金 id，设置对应报错提示。关键代码如下：

```
    declare msg varchar(40);
    if new.pro_type <> 1 and new.pro_type <> 2 and new.pro_type <> 3 then
        set msg=concat('type ',new.pro_type,' is illegal!');
        signal sqlstate '45000' set message_text=msg;
    end if;
    if new.pro_type=1 and not exists(select* from property,finances_product
where new.pro_pif_id=finances_product.p_id) then
        set msg=concat('finances product #',new.pro_pif_id,' not found!');
        signal sqlstate '45000' set message_text=msg;
    end if;
```

```
    if new.pro_type=2 and not exists(select* from property,insurance where
new.pro_pif_id=insurance.i_id) then
        set msg=concat('insurance #',new.pro_pif_id,' not found!');
        signal sqlstate '45000' set message_text=msg;
    end if;
    if new.pro_type=3 and not exists(select* from property,fund where
new.pro_pif_id=fund.f_id) then
        set msg=concat('fund #',new.pro_pif_id,' not found!');
        signal sqlstate '45000' set message_text=msg;
    end if;
```

2.9 用户自定义函数

本节围绕用户自定义函数展开，要求学习掌握用户自定义函数的定义和调用。

2.9.1 创建函数并在语句中使用它

任务：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。

方法：首先用 create function 语句创建函数 get_deposit()，依据客户编号计算其所有储蓄卡余额的总和；接着利用创建的函数，在 select 语句中查询存款总额在 100 万以上的客户身份证号、姓名和存款总额。创建函数的代码如下：

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

使用函数在 select 语句中调用自定义函数查询的代码如下：

```
    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;
```

2.10 安全性控制

本节围绕数据库的自主存取方法展开，主要涉及用户和角色的创建、权限的授予与回收。

2.10.1 用户和权限

本关任务：创建用户，并给用户授予指定的权限。

方法：①首先使用 create user 语句创建用户 tom 和 jerry，初始密码均为'123456'；②使用 grant 语句授予 tom 查询客户姓名、邮箱、电话的权限，通过 with grant option 使 tom 可转授权限；③使用 grant 语句授予 jerry 修改银行卡余额的权限；④使用 revoke 语句收回 Cindy 查询银行卡信息的权限。代码如下：

```
create user 'tom' identified by '123456';
create user 'jerry' identified by '123456';
grant select(c_name,c_mail,c_phone) on client to tom with grant option;
grant update(b_balance) on bank_card to jerry;
revoke select on bank_card from Cindy;
```

2.10.2 用户、角色与权限

该关卡任务已完成，思路与上一关类似，实施情况本报告略过。

2.11 并发控制与事务的隔离级别

本节围绕数据库中的并发控制与事务的隔离级别展开，要求学习掌握隔离级别的设置、事务的开启/提交/回滚等、通过添加代码实现读脏/不可重复读/幻读等出错场景。

2.11.1 并发控制与事务的隔离级别

任务：设置事务的隔离级别。

方法：Mysql 对事务的隔离级别由高到低依次为 READ UNCOMITTED、READ COMMITTED、REPEATABLE READ、SERIALIZABLE，依次扩展解决读脏数据问题、不可重复读问题、幻读问题。按照题目要求，将事务的隔离级别设置为 read uncommitted，通过 start transaction 开启事务，通过 rollback 回滚。代码如下：

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

```
start transaction;
insert into dept(name) values('运维部');
rollback;
```

2.11.2 读脏

任务：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。

方法：读脏是指当某一事务回滚前的修改被其他事务错误读取。根据所给条件可知，为了重现读脏错误，需要将事务隔离级别设置为最低级别 READ COMMITTED。事务 1 读取余票前需要设置 sleep，等待事务 2 修改；事务 2 回滚需等待事务 1 读取余额，故在事务 2 的回滚前需设置 sleep。事务 1 代码如下：

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

事务 2 代码如下：

```
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
set @n=sleep(2);
rollback;
```

2.11.3 不可重复读

该关卡任务已完成，思路与上一关类似，实施情况本报告略过。

2.11.4 幻读

该关卡任务已完成，思路与上一关类似，实施情况本报告略过。

2.11.5 主动加锁保证可重复读

任务：在事务隔离级别较低的 read uncommitted 情形下，通过主动加锁，保证事务的一致性

方法：共享锁保证同一事务重复读取某数据时，其他事务将无法修改该数据，从而保证了可重复读。在事务 1 代码的第 6 行中使用 for update 语句添加锁，则

事务 2 尝试在事务 1 两次读之间出一张票时，新代码无法在事务 1 的锁释放之前执行，因此实现了可重复读，即事务 1 的两次读取结果一致。事务 1 代码如下：

```
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' for share;
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;
```

2.11.6 可串行化

任务：对两个代码文件进行修改，使得两个事务并发执行的结果与 t2→t1 串行执行的结果相同。

方法：多个事务并发执行是正确的，当且仅当其结果与按某一次序串行地执行这些事务时的结果相同。本关使用 sleep()函数让事务 1 休眠，至到事务 2 执行完后再开始执行即可。事务 1 代码如下：

```
start transaction;
set @n = sleep(4);
select tickets from ticket where flight_no = 'MU2455';
select tickets from ticket where flight_no = 'MU2455';
commit;
```

2.12 备份+日志：介质故障与数据库恢复

本节围绕数据库的备份与恢复展开，要求掌握基于备份的数据库恢复、带日志文件的数据库恢复等，从而解决数据库损坏、存储介质故障等问题。

2.12.1 备份与恢复

任务：为 residents 数据库作逻辑备份，再用逻辑备份文件恢复数据库。

方法：使用 mysqldump 工具将服务器上的数据库 residents 备份至 residents_bak.sql 中，代码如下：

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

使用 mysql 指令根据 residents_bak.sql 还原数据库，代码如下：

```
    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

任务：对数据库 train 作一次海量备份，同时新开日志；利用备份文件和日志文件恢复数据库。

方法：使用 mysqldump 工具进行逻辑备份，由于需要新开日志，需要在该语句中添加--flush-logs 参数。代码如下：

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

当发生介质故障时，系统会将备份后新开的日志文件保存为 log/binlog.000018。为了保证两次发生的业务数据都不丢失，首先通过还原工具 mysql 备份数据库；接着使用日志工具，通过故障发生器的日志文件继续恢复数据库。代码如下：

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

2.13 数据库设计与实现

本节围绕数据库的设计与实现展开，主要包括从概念模型到 mysql 实现、需求分析和逻辑模型的构建、建模工具的使用等。

2.13.1 从概念模型到 MySQL 实现

任务：根据给出的 E-R 图，在 mysql 中实现数据库、表、完整性约束的创建。航空公司订票系统的 E-R 图如图 2.2 所示。

方法：①分析所给 E-R 图的实体，主要有用户、旅客、机场、航空公司、民航飞机、航班常规调度表、航班表、机票，分别为其建表。②根据 E-R 图中的连接线和题目要求分析实体间的关系，将这些一对多的关系扩充到建好的表中，特别要注意外码约束。③检查完整性约束，例如主码约束、唯一、非空等要求。具体代码省略。

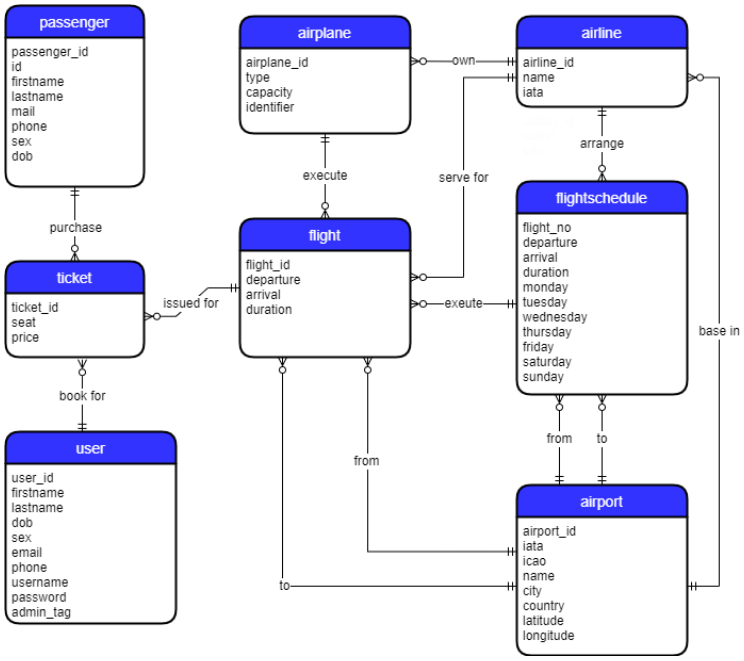

图 2.2 机票订票系统 E-R 图

2.13.2 从需求分析到逻辑模型

任务：根据应用场景业务需求描述，完成 ER 图，并转换成关系模式。

方法：通过需求分析，得到主要实体为 movie、customer、hall、schedule、ticket；movie 与 schedule 存在放映关系，hall 与 schedule 存在位于关系，ticket 与 schedule 存在属于关系，customer 与 ticket 存在购买关系。得出的 E-R 如图 2.3 所示：

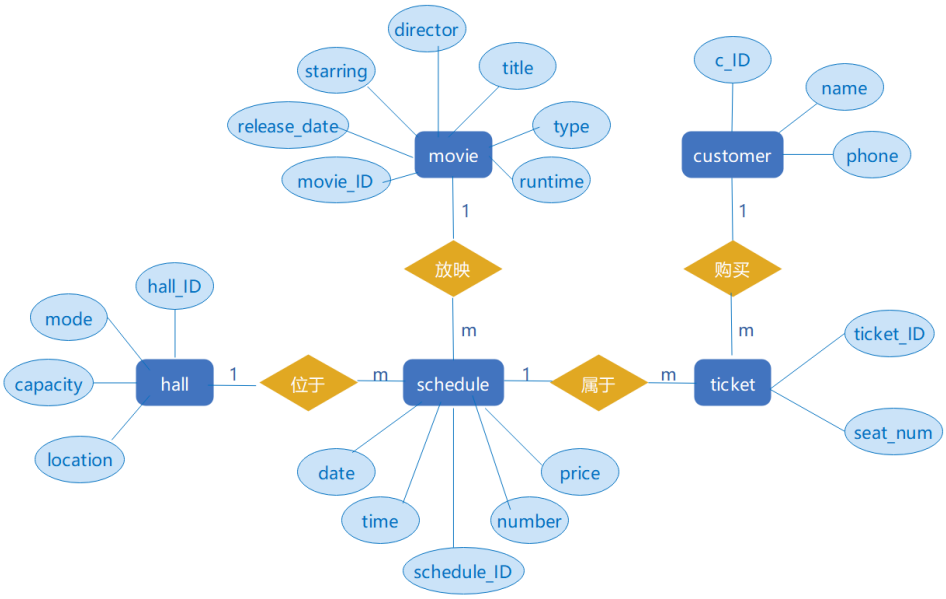

图 2.3 影院管理系统 E-R 图

构建的关系模式为：

- movie(movie_ID,title,type,runtime,release_date,director,starring),

主码:(movie_ID)

- customer(c_ID,name,phone),主码:(c_ID)
- hall(hall_ID,mode,capacity,location),主码:(hall_ID)
- schedule(schedule_ID,date,time,price,number,hall_ID,movie_ID),
  主码:(schedule_ID);外码:(hall_ID,movie_ID)
- ticket(ticket_ID,seat_num,c_ID,schedule_ID),
  主码:(ticket_ID);外码:(c_ID,schedule_ID)

2.13.3 建模工具的使用

该关卡任务已完成，实施情况本报告略过。

2.13.4 制约因素分析与设计

数据库设计中的制约因素主要有数据的性质、数据的冗余度、数据的一致性、数据的完整性、数据的可访问性、数据的安全性、数据的性能、数据的可维护性等。

在社会方面，设计数据库需要考虑用户群体在社会中的分布情况、数据库的使用方式和频率，以及社会的发展变化对数据库的影响。

在安全方面，设计数据库必须考虑如何保护数据的保密性，避免未经授权的用户访问或篡改数据；同时也要考虑如何保护数据的完整性，避免数据被以外删除或篡改。

在法律方面，法律法规可能会对数据保护、隐私保护、数据安全等方面提出要求，设计数据库时应考虑如何满足这些法律要求。

在环境方面，不同的使用的场景也会对数据库的设计产生影响。例如，如果数据库需要在网络环境中使用，则应考虑网络带宽、网络延迟等因素。如果数据库要在分布式环境中使用，则应考虑如何实现数据一致性、事务处理等功能。

例如在 2.11 节中，数据库在分布式环境中使用，要求允许用户同时访问同一数据库，因此在设计时我们需要考虑到数据的一致性约束，通过设置事务的隔离级别、主动加锁避免丢失修改、读脏、不可重复读、幻读等问题。在 2.13.1 节中，考虑到用户的安全性制约因素，我们设计数据库时将用户分为 2 类，普通用户可以订票，管理用户有权限维护管理整个系统的运营。

2.13.5 工程师责任及其分析

工程师在设计数据库时应承担的社会责任包括：

1) 保证数据安全：数据库工程师需要保证数据库系统的安全性，避免数据泄露或被未经授权的人员访问。
2) 保证数据准确性：数据库工程师需要保证数据库中存储的数据准确无误，这是公司决策制定和业务运行的基础。
3) 维护数据库性能：数据库工程师需要确保数据库的性能良好，以满足业务的需要。
4) 协助业务开发：数据库工程师可以为业务开发人员提供帮助，协助设计数据库结构和优化查询。
5) 提供技术支持：数据库工程师还需要为公司的数据库用户提供技术支持，帮助他们解决使用过程中遇到的问题。
6) 保护个人隐私：在处理数据时，数据库工程师需要尽量保护个人隐私，避免将个人信息泄露给第三方。

2.14 数据库应用开发(JAVA 篇)

本关围绕 JDBC 的体系结构展开，要求学习使用 Java 高级语言进行数据库应用系统的开发。

2.14.1 JDBC 体系结构和简单的查询

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。

方法：在所给 Java 程序中通过构建 JDBC 应用完成查询。具体步骤如下：

① 导入包：使用 import java.sql.* 导入数据库编程所需的 JDBC 类的包。
② 注册 JDBC 驱动程序：使用 Class.forName("com.mysql.cj.jdbc.Driver")初始化驱动程序，打开与数据库的通信通道。
③ 打开连接：使用 DriverManager.getConnection()方法创建一个 Connection 对象，该对象表示与数据库的物理连接。
④ 执行查询：使用类型为 Statement 的对象来构建和提交 SQL 语句到数据库。
⑤ 从结果集中提取数据：需要使用相应的 ResultSet.getXXX()方法从结果集中检索数据。

⑥ 释放资源：需要明确地关闭所有数据库资源，而不依赖于 JVM 的垃圾收集。

部分关键代码如下：

```
    // 注册驱动程序
    Class.forName("com.mysql.cj.jdbc.Driver");
    // 数据库 URL 配置，创建数据库连接对象
    String URL = "jdbc:mysql://127.0.0.1:3306/finance?useUnicode=
true&characterEncoding=UTF8&useSSL=false&serverTimezone=UTC";
    String USER = "root";
    String PASS = "123123";
    connection = DriverManager.getConnection(URL, USER, PASS);
    // 创建 statement 对象
    statement = connection.createStatement( );
    String SQL = "select* from client where c_mail is not null;";
    resultSet = statement.executeQuery(SQL);
    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"));
     }
```

2.14.2 用户登录

任务：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。

方法：首先获取用户输入的邮箱与密码，进行数据库连接以及 JDBC 对象的创建。接着查询表中邮箱为用户输入邮箱的元组，将结果存入 ResultSet 对象中。接着进行 next()方法查询，若无数据，输出“用户名或密码错误!”；否则继续比较密码。若与用户输入密码一致，则输出“登陆成功”，否则输出“用户名或密码错误!”。处理完毕后，进行 JDBC 对象的释放。具体代码省略。

2.14.3 添加新用户

任务：向 client(客户表)插入记录。

方法：成功连接数据库后，实例化 PreparedStatement 类的一个对象。通过该对象关联一条 sql insert 语句，然后调用对象的 executeUpdate()方法即可。部分代码如下：

```
    String sql = "insert into client values(?,?,?,?,?,?);";
    pstmt = connection.prepareStatement(sql);
```

```
    pstmt.setInt(1,c_id);
    pstmt.setString(2,c_name);
    pstmt.setString(3,c_mail);
    pstmt.setString(4,c_id_card);
    pstmt.setString(5,c_phone);
    pstmt.setString(6,c_password);
    n = pstmt.executeUpdate();
```

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：编写修改客户登录密码的的方法。

方法：首先使用 prepareStatement 接口的 executeQuery()方法查询邮箱为输入邮箱的元组。若查询结果为空，则返回 2 表示用户不存在，否则查看该元组密码是否为用户输入的旧密码。若不是，则返回 3 表示密码不正确，否则使用 executeUpdate()进行修改，并返回 1 表示密码修改正确。当程序出现异常时返回 -1。具体流程如图 2.4 所示：

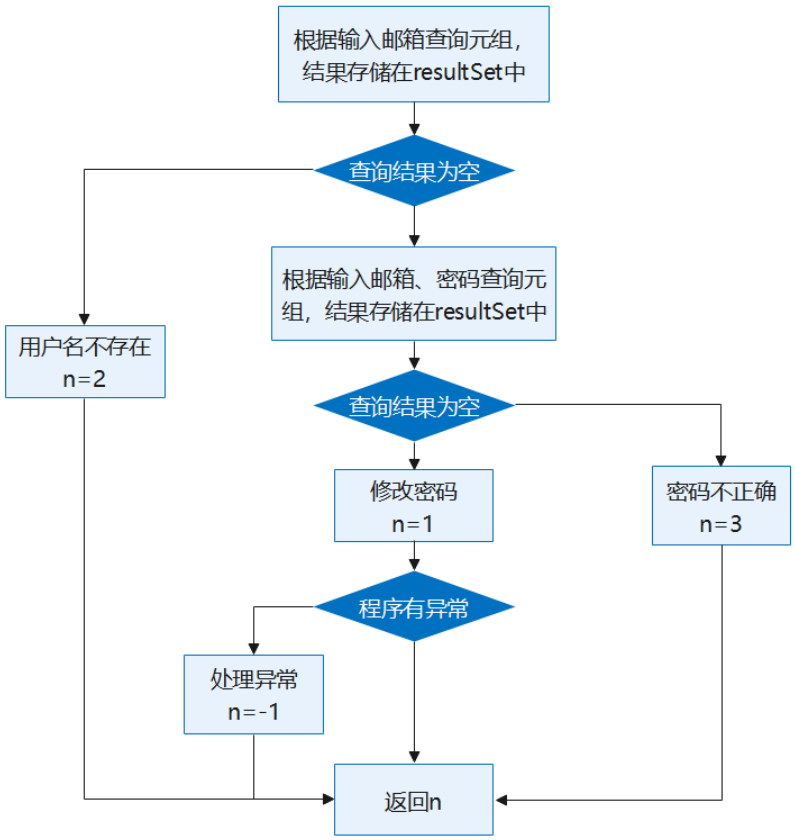

图 2.4 修改密码流程图

2.14.6 事务与转账操作

任务：编写一个银行卡转账的方法。

方法：首先创建转账操作方法，传入已连接数据库的 connection 对象、转出账号、转入账号和转账金额，并设置事务隔离级别为 REPEATABLE_READ。接着再进行合法性的判断，若转出储蓄卡账号不存在，则事务回滚并返回 false；若转出账号存在但转出余额不足，则事务回滚并返回 false；若转入账号不存在，则事务回滚并返回 false。若无以上情况，则表明转账操作合法，进行事务提交并返回 true。合法性判断部分代码如下：

```
resultSet = pstmt.executeQuery();
if(resultSet.next()==false){
    // 卡号不为储蓄卡或不存在
    n = false;
    connection.rollback();
}else{
    if(resultSet.getDouble("b_balance")<0){
    // 转出卡的余额需要大于转出金额
    n=false;
    connection.rollback();
    }else{
        sql="select* from bank_card where b_number=?;";
        pstmt = connection.prepareStatement(sql);
        pstmt.setString(1,destCard);
        resultSet = pstmt.executeQuery();
        if(resultSet.next()==false) {
            n = false;
            connection.rollback();
        }
        else connection.commit();
    }
}
```

2.14.7 把稀疏表格转为键值对存储

任务：将一个稀疏的表中有保存数据的列值，以键值对的形式转存到另一个表中。

方法：首先完成 insertSC()函数。传入已连接数据库的 connection 对象、学生学号、科目名称、成绩，使用 prepareStatement 接口的 executeUpdate()方法进行该元组的插入。

接着完成 main()函数。在稀疏表中查询每个元组，并提取学生学号和各科成绩。若某科成绩不存在，则视成绩为 0，将不为 0 的成绩填入键值对表中。完成

所有稀疏表的元组转化后，进行各 JDBC 对象的释放。具体代码省略。

2.15 数据库的索引 B+树实现

本节围绕索引 B+树展开。B+树的索引类型有聚簇索引和非聚簇索引两种，本模块重点是考察 B+树的逻辑与实现，使用非聚簇索引，仅用 RID 类存储记录在表数据页中的位置，实现 B+树的插入、删除等功能。

2.15.1 BPlusTreePage 的设计

一棵完整的 B+树，由内部节点与叶子节点构成，并能够在记录插入和删除时维持 B+树性质。正常地进行节点的分裂与合并，就会存在页的获取与释放，整个章节将设计并实现一个完整的索引 B+树，其结构设计如图 2.5 所示。

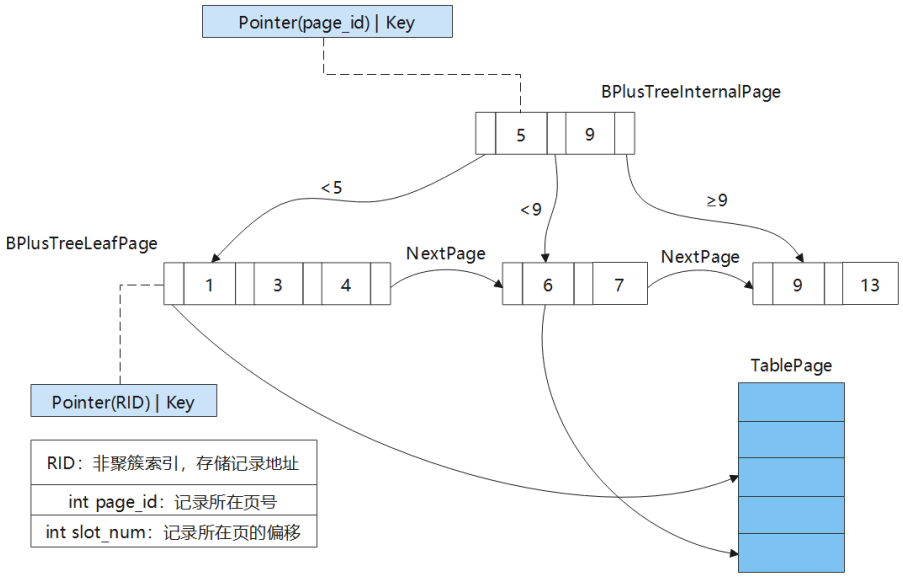

图 2.5 B+树设计结构图

该关卡中最重要的函数为 GetMinSize()，由于篇幅限制，我们在这里只分析该函数的实现。该函数的功能为获取当前节点允许的最少元素个数，分两类讨论：

①此时为根节点，则继续看该节点是内部节点还是叶子节点。如果是内部节点，则此时至少存在 2 个索引；如果是叶子节点，则此时至少存在 1 个索引。

②此时不是根节点，则正常处理。具体代码如下：

```
int BPlusTreePage::GetMinSize() const {
    // 根节点讨论是否为叶子节点
    if (IsRootPage()) {
        return IsLeafPage() ? 1 : 2 ;
```

```
    }
    // 非根结点正常处理
    return (max_size_ + 1 )/ 2;
}
```

2.15.2 BPlusTreeInternalPage 的设计

BPlusTreeInternalPage 内部页不存储任何实际数据，而是存储有序的 m 个键条目和 m+1 个指针（也称为 page_id）。 由于指针的数量不等于键的数量，因此将第一个键设置为无效，并且查找方法应始终从第二个键开始。 任何时候，每个内部页面至少有一半已满。 在删除期间，可以将两个半满页面合并为合法页面，或者可以将其重新分配以避免合并，而在插入期间，可以将一个完整页面分为两部分。

内部结点成员在 BPlusTreePage 的基础上额外增加一个数组，用于存储索引信息。一个 Max_size 为 n 的内部结点，有 n-1 个索引键和 n 个子结点索引（由于二者数量不相同，一般第一个索引键为无效键），即该数组有如下结构：

INVALID_KEY+PAGE_ID(0)	KEY(1)+PAGE_ID(1)	……	KEY(n)+PAGE_ID(n)

根据以上结构可以实现 b_plus_tree_internal_page.cpp 文件中的函数。具体代码省略。

2.15.3 BPlusTreeLeafPage 的设计

该关卡任务已完成，实施情况本报告略过。

2.15.4 B+树索引：Insert

本关卡中，需要完成 B+树索引的查找以及插入功能，并实现 B+树记录迭代器。

当有记录插入时，从根节点进入 B+树，定位叶子节点并插入该记录。插入后判断内部记录是否超过最大值，如果超过，对叶子节点进行分裂，内部记录对半分离，产生新叶子节点，插入到叶子节点链表中。同时，在父节点中插入新叶子节点索引，即进行递归插入，直至不再有分裂。

在插入功能完成后，需要继续对 B+树存储信息进行查找和迭代。查找功能即 B+树的记录的定位，从根节点进入，通过内部节点的索引信息不断向下遍历，直至对应的叶子节点，即可查找特定的记录信息。同时，结合查找功能，B+树需要生成自身记录的迭代器，迭代器由当前叶子节点指针和元素下表组成，通过

实现对++运算符的重载，完成迭代器在叶子节点链表中的遍历，并实现末尾判定函数以结束遍历。迭代器的主要函数代码如下：

```
// 迭代器移动至下一个记录(实现迭代器顺序遍历 B+树记录的功能)，返回当前迭代器
INDEX_TEMPLATE_ARGUMENTS
INDEXITERATOR_TYPE &INDEXITERATOR_TYPE::operator++() {
  if ((index_ + 1) >= leaf_->GetSize()) {
    // 利用 NextPageId 属性实现跨结点遍历
    page_id_t next = leaf_->GetNextPageId();
    if (next == INVALID_PAGE_ID) {
      // 叶子结点不再使用后，需要及时 unpin 释放
      bufferPoolManager_->UnpinPage(leaf_->GetPageId(), false);
      leaf_ = nullptr;
      index_ = 0;
      return *this;
    } else {
      bufferPoolManager_->UnpinPage(leaf_->GetPageId(), false);
      Page *page = bufferPoolManager_->FetchPage(next);
      leaf_ = reinterpret_cast<B_PLUS_TREE_LEAF_PAGE_TYPE *>(page);
      index_ = 0;
      return *this;
    }
  } else {
    index_++;
    return *this;
  }
}
```

2.15.5 B+树索引：Remove

本关需要了解记录删除时 B+树结构的变化过程和细节，完成 B+树索引的删除功能。

当删除特定记录时，首先从根节点进入 B+树，通过内部节点的索引信息不断向下遍历，直至对应的叶子节点，之后删除该记录，删除后判断内部记录是否已小于最小值，是则开始进行树的调整。

接着先判断是否可以与相邻兄弟节点合并，如果可以合并兄弟节点，则合并后删除其在父节点中的索引，即递归进行元素删除操作，直至不再有合并发生；否则可以进行元素的调动，移动兄弟节点中一个元素补充到叶子节点中，并维持 B+树的索引正确。通过以上原理，使用 BufferPoolManager 类对缓冲区进行操作，即可实现 b_plus_tree_delete.cpp 文件中的函数。具体代码省略。

3 课程总结

本次数据库系统原理实践完成了 15 个实训任务的所有子任务，共计 73 个关卡，总分 163 分。本次实践的主要工作内容如下：

1) 完成了数据库、表、视图、约束、存储过程、函数、触发器等数据对象的管理与编程，掌握了如何用 MySQL 的相关 DDL 语句去创建、使用他们；
2) 完成了数据查询、插入、删除与修改等数据处理相关任务，尤其是对 select 语句的运用更加熟练，并学会使用了更多的 MySQL 自带的函数；
3) 完成了数据库的安全性控制、恢复机制、并发机制等实验，掌握了如何进行自助存取控制提升系统安全性、日志文件的使用与数据库备份的方法等；
4) 完成了数据库的设计与实现，对数据库的需求分析、设计数据库的制约因素、工程师的社会责任等有了更深的认识；
5) 完成了 Java 语言的数据库应用系统的开发，掌握了 JDBC 体系架构的使用；
6) 完成了数据库索引 B+树的设计实现

理论课的学习使我掌握了 MySQL 的基础语法，例如如何创建表、修改表；而实验课让我学会了更多更复杂的 MySQL 语句，并能通过合理使用搜索引擎来学习更多巧妙的函数、简化 select 查找语句，例如 rank()函数、dense_rank()函数，同时也让我感受到了 MySQL 的灵活性。

我认为本课程的一大亮点是使用一个典型的金融场景案例贯穿大部分章节，且数据库应用设计部分与实际生活紧密结合。这样可以让我更系统地理解数据库在实际生活的使用，印象更加深刻。此外，老师们也十分认真负责，积极解答我们的疑问，在头歌平台上的每个关卡边都提供了很详细很有用的指导资料。

总的来说，此次实验课程内容丰富有趣，实验目标清晰明了，收获颇丰。实验建议部分，希望以后关于现场检查答辩的内容、流程要求可以更加明确；在实训 15 中，索引 B+树部分的指导资料可以更加详实、丰富。