

**数据库系统原理实践报告**

|  |  |
| --- | --- |
| 专 业： | 数据科学与大数据技术 |
| 班 级： | 大数据2101班 |
| 学 号： | U202115652 |
| 姓 名： | 李嘉鹏 |
| 指导教师： | 赵小松 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2023 年 6 月 3 日

**教师评分页**

|  |  |
| --- | --- |
| 子目标 | 子目标评分 |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
|  |  |

|  |  |
| --- | --- |
| 总分 |  |

目 录

[1 课程任务概述 1](#_Toc136724515)

[2 任务实施过程与分析 2](#_Toc136724516)

[2.1 数据库、表与完整性约束的定义(Create) 2](#_Toc136724517)

[2.2 表结构和完整性约束的修改(Alter) 3](#_Toc136724518)

[2.3 数据查询(Select) 4](#_Toc136724519)

[2.4 数据的插入、修改与删除(Insert,Update,Delete) 11](#_Toc136724520)

[2.5 视图 11](#_Toc136724521)

[2.6 存储过程与事务 11](#_Toc136724522)

[2.7 触发器 15](#_Toc136724523)

[2.8 并发控制与事务的隔离级别 16](#_Toc136724524)

[2.9 备份+日志：介质故障与数据库恢复 18](#_Toc136724525)

[2.10 数据库设计与实现 19](#_Toc136724526)

[2.11 数据库的索引B+树实现 23](#_Toc136724527)

[3 课程总结 25](#_Toc136724528)

# 1 课程任务概述

本学期的《数据库系统原理实践》课程是《数据库系统原理》的配套实验课。实验课注重理论与实践相结合，其内容主要是对课内理论学习内容的具体化和拓展，要求我用OpenGauss、Java和C++语言等实现数据库的一系列操作。具体来说，实验内容包括以下几个部分：

（1）数据库、表、索引、视图、完整性约束、存储过程与事务、函数、触发器、游标等数据对象的管理与编程，以及用户自定义函数的使用；

（2）数据查询（select）、数据插入（insert）、数据删除（delete）与数据修改（update）等数据处理相关任务；

（3）数据库的安全性控制、完整性控制，数据的备份与恢复机制、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）数据库应用系统开发（Java篇）；

（6）数据库的索引B+树的实现。

课程主要依托于头歌实践教学平台，代码可以线上实时评测。实践课程链接的url为<https://www.educoder.net/classrooms/fpm4gqxz/shixun_homework>。实验环境为Linux操作系统下的OpenGauss2.1。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的全部子任务以及2.13、2.14的部分子任务，总共在头歌平台上通过了66关，总分数为133分。下面将重点针对其中的24个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)

本小节主要是围绕数据库的创建以及表的各类约束（包括表的主码约束、外码约束、CHECK约束、DEFAULT约束、UNIQUE约束）展开。

本任务已完成全部6个关卡。

**2.1.1 创建数据库（第1关）**

本关卡任务已完成，此处略过实验过程分析。（下文中仅列出要详细解释的关卡，不再单独说明简单的关卡）

**2.1.2 CHECK约束（第4关）**

**本关任务**：创建表products，并分别实现对品牌和价格的约束，主码约束不要显示命名。

**实验过程**：首先观察题目的要求，pid为主码，brand只能为’A’、’B’中的某一个，price必须大于0。据此即可分别创建一个主码约束和两个CHECK约束（分别命名为CK\_products\_brand和CK\_products\_price）。CHECK约束的语法是：CHECK约束 ::= [CONSTRAINT [约束名]] CHECK (条件表达式)]。

需要注意的是，在代码开头处要先drop前一关的数据库，并创建一个新的数据库，从而实现对数据库的重置，否则可能会受到前面关卡的影响。代码如下。

drop database MyDb;

create database MyDb;

use MyDb;

create table products(

    pid char(10) primary key, name varchar(32),

    brand char(10) constraint CK\_products\_brand check(brand in ('A', 'B')),

    price int constraint CK\_products\_price check(price > 0));

**2.1.3 DEFAULT约束（第5关）**

**本关任务**：创建表hr，并分别为id创建主码约束、为name创建NOT NULL约束、为mz创建DEFAULT约束（缺省值为“汉族”）。

**实验过程**：按照题意写出各类约束即可。DEFAULT约束的语法是：col\_name data\_type [DEFAULT {literal | (expr)} ]。代码如下。

create table hr(

    id char(10) primary key, name varchar(32) not null,

    mz char(16) default '汉族');

## 2.2 表结构和完整性约束的修改(Alter)

本小节主要是围绕数据库表结构与完整性约束的修改操作展开。任务包括对数据库中表名/列名的修改、添加与删除字段、修改字段、添加列/表约束、删除与修改列/表约束等。

本任务已完成全部4个关卡。

**2.2.1 添加、删除与修改约束（第4关）**

**本关任务**：数据库中已经有dept和staff两个表，现在需要补充代码完成以下要求：(1) 为表Staff添加主码；(2) Dept.mgrStaffNo是外码，对应的主码是Staff.staffNo，添加这个外码，名字为FK\_Dept\_mgrStaffNo；(3) Staff.dept是外码，对应的主码是Dept.deptNo，添加这个外码，名字为FK\_Staff\_dept；(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；(5) 为表Dept添加unique约束，规则为deptName不允许重复。约束名为UN\_Dept\_deptName。

**实验过程**：根据题目要求逐步写出各个约束的代码即可。

首先添加主码，语法是ALTER TABLE 表名 ADD [CONSTRAINT [约束名]] PRIMARY KEY(列1,列2,...)；然后添加外码并给外码约束命名，语法是ALTER TABLE 表名 ADD [CONSTRAINT [约束名]]；最后分别添加CHECK约束和UNIQUE约束，语法是ALTER TABLE 表名 ADD [CONSTRAINT [约束名]] check(条件表达式)以及alter table 表名 ADD [CONSTRAINT [约束名]] UNIQUE(列1,...)。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)

本小节主要覆盖了各类需求下的数据库查询任务。任务包括对数据库中数据的简单查询（例如直接用select语句配合简单的条件选择语句实现的查询）、对查询结果排序、多层嵌套查询与多条件查询、多表连接、外连接、对结果进行升序或降序排序、LIKE、NOT EXISTS的用法、子查询、集函数与分组统计、top n查询、数据消重、rank函数的使用等等。

本任务已完成全部25个关卡。

**2.3.1 商品收益的众数（第6关）**

**本关任务**：查询资产表中所有资产记录里商品收益的众数（可能不止一个）和它出现的次数，出现次数命名为presence。

**实验过程**：使用集函数count(\*)分别统计每个商品收益出现的次数，利用all求出全部次数的最大值，并找出出现次数大于等于这个最大值的商品收益，即可得到商品收益的众数。这里使用了嵌套查询，并利用group by子句实现了分组统计。代码如下。

select pro\_income, count(\*) as presence from property

group by pro\_income

having count(\*) >= all(

    select count(\*) from property

    group by pro\_income);

**2.3.2 投资总收益前三名的客户（第10关）**

**本关任务**：查询当前总的可用资产收益（被冻结的资产除外）前三名的客户的名称、身份证号及其总收益，按收益降序输出，总收益命名为total\_income。

**实验过程**：首先根据用户id对客户表client和财产表property进行自然连接，然后按照用户名称和身份证号进行分组统计，利用sum函数计算每个用户总的可用资产收益，注意此处pro\_status数据项的值需要为“可用”。最后利用limit函数输出前三名的客户即可。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\_name, c\_id\_card

order by total\_income desc

limit 3;

**2.3.3 客户理财、保险与基金投资总额（第12关）**

**本关任务**：综合客户表(client)、资产表(property)、理财产品表(finances\_product)、保险表(insurance)和基金表(fund)，列出客户的名称、身份证号以及投资总金额（每笔投资金额=商品数量\*该产品每份金额）。注意投资总金额是客户购买的各类资产投资金额的总和，总金额命名为total\_amount。查询结果按总金额降序排序。

**实验过程**：这是一道比较复杂的查询题，主要是由于客户的投资总金额覆盖了理财产品、保险和基金三类，因此需要对这三个表进行连接，并按照用户名称和身份证号进行分组统计，取出每个表相应的商品数量与该产品单份投资金额，二者的乘积之和即为用户在该类别中的投资金额之和。最后，只需将三部分的和相加即可得到最终结果。实验中使用union all对三个表取可重复的并集，再使用left outer join与用户表做自然连接处理。这里需要特别小心，对于没有购买任何产品的客户，其sum(pro\_amount)的值可能为空值NULL，因此需要使用coalesce函数对空值进行处理，强制将其赋为0。代码如下。

select c\_name, c\_id\_card,

    coalesce(sum(pro\_amount), 0) as total\_amount

from client

left outer join

(select pro\_c\_id, pro\_quantity \* p\_amount as pro\_amount

    from property, finances\_product

    where pro\_type=1 and p\_id=pro\_pif\_id

union all

    select pro\_c\_id, pro\_quantity \* i\_amount as pro\_amount

    from property, insurance

    where pro\_type=2 and i\_id=pro\_pif\_id

union all

    select pro\_c\_id, pro\_quantity \*f\_amount as pro\_amount

    from property, fund

    where pro\_type=3 and f\_id=pro\_pif\_id) on c\_id=pro\_c\_id

group by c\_name,c\_id\_card

order by total\_amount desc;

若没有使用coalesce函数，则结果会显示为空，且不会被正常降序排序，无法通过评测，如图2.1所示。

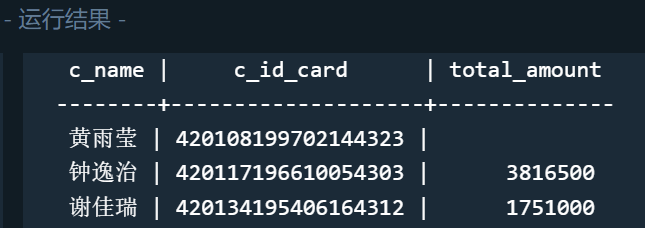


图2.3.1：不对空值NULL赋值为0的结果

**2.3.4 第N高问题（第14关）**

**本关任务**：查询单份保险金额的第4高保险产品的编号和保险金额。排序时，相同的金额被视为同一排名。

**实验过程**：这道题可以先直接获取第4高的保险金额，然后再找出保险金额等于这个金额的所有保险产品。由于排名可以重叠，因此在选取金额时用distinct确保选取到正确的值，最后使用limit 3,1获得第4高的金额即可。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);

**2.3.5 基金收益两种方式排名（第15关）**

**本关任务**：查询资产表中客户编号、客户基金投资总收益和其基金投资总收益的排名(从高到低)。总收益相同时名次并列。总收益命名为total\_revenue, 名次命名为rank。第一条SQL语句实现全局名次不连续的排名，第二条SQL语句实现全局名次连续的排名。不管哪种方式排名，收益相同时客户编号小的都在前。

**实验过程**：以降序的基金投资总收益为关键字，直接使用rank()和dense\_rank()函数即可，其中前者名次不连续，后者名次连续。

rank()函数的语法是：rank() over(partition by ...order by)，其中partition by用于给结果集分组，如果没有指定，那么视为把整个结果集作为一个分组。dense\_rank()函数的语法与此类似。

实现名次不连续的代码如下：

select pro\_c\_id, coalesce(sum(pro\_income), 0) as total\_revenue,

    rank() over(order by total\_revenue desc) as "rank"

from property where pro\_type=3

group by pro\_c\_id;

实现名次连续的代码如下：

select pro\_c\_id, coalesce(sum(pro\_income),0) as total\_revenue,

    dense\_rank() over(order by total\_revenue desc) as "rank"

from property where pro\_type=3

group by pro\_c\_id;

**2.3.6 持有完全相同基金组合的客户（第16关）**

**本关任务**：查询持有相同基金组合的客户对。如编号为A的客户持有的基金，编号为B的客户也持有；反过来，编号为B的客户持有的基金，编号为A的客户也持有，则(A,B)即为持有相同基金组合的二元组。最终只需要显示编号小者在前的那一对。

**实验过程**：这一题使用了两次NOT EXISTS条件判断语句。“两个客户持有的基金完全一致”这句话可以被转化成更清晰的意思，即不存在这样的基金是客户A持有而客户B不持有，也不存在这样的基金是客户B持有而客户A不持有，利用两层NOT EXISTS即可实现查询。需要注意两个客户必须至少购买一项基金，即两个没有购买任何基金的客户不计入统计。代码如下：

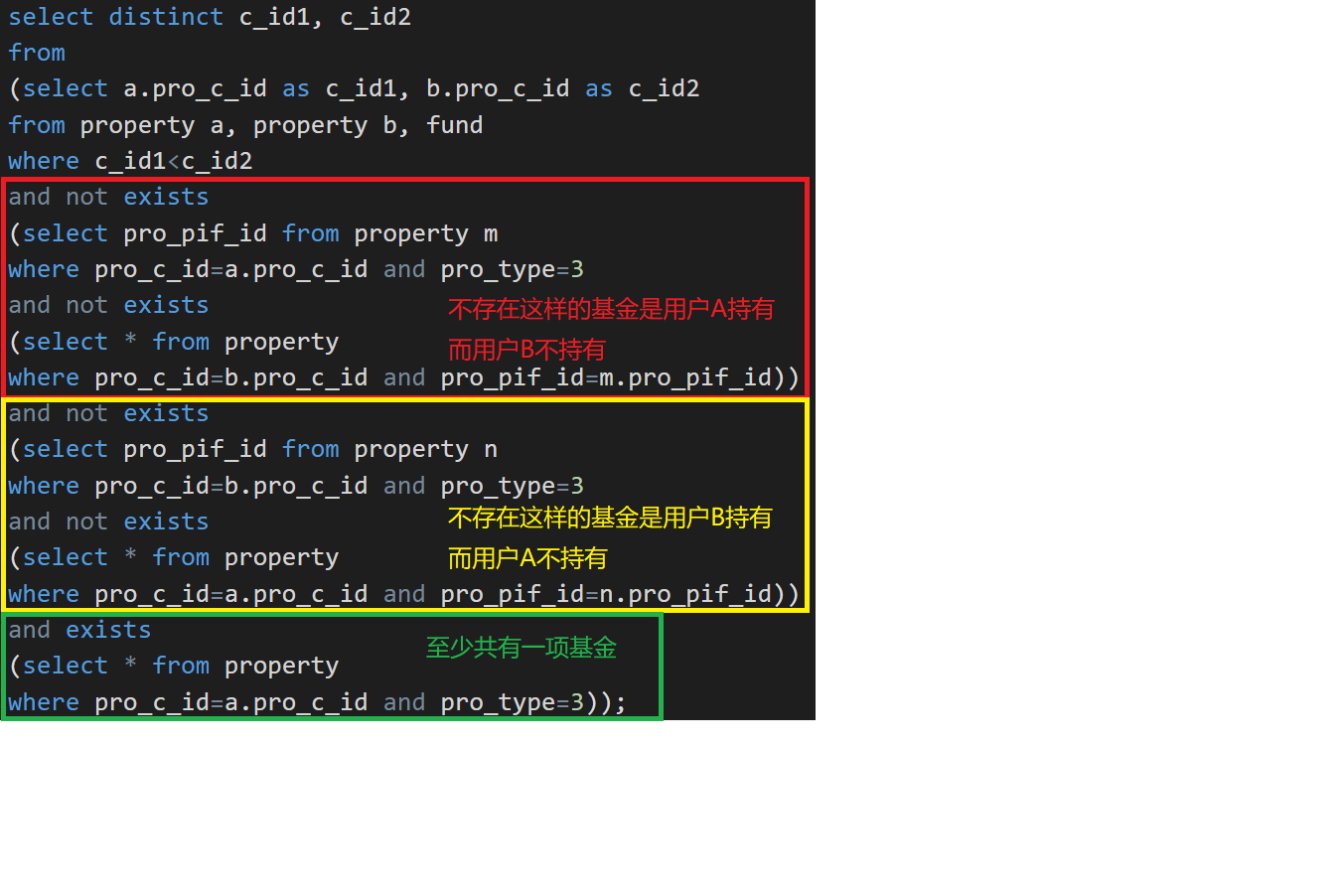


图2.3.2：持有完全相同基金组合的客户代码解释

**2.3.7 以日历表格式显示每日基金购买总金额（第19关）**

**本关任务**：以日历表格式列出2022年2月每周每个交易日（周一到周五）基金购买的总金额，输出格式如表2.3.1所示。

表2.3.1：日历表格式输出每周前五天基金购买的总金额

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| week\_of\_trading | Monday | Tuesday | Wednesday | Thursday | Friday |
| 1 | … | … | … | … | … |
| 2 | … | … | … | … | … |
| 3 | … | … | … | … | … |
| 4 | … | … | … | … | … |

**实验过程**：对2022年2月7日开始的每一天进行处理。使用date\_part函数获取每一个交易日的周次，由于2月的周次为6-9，故需要将周次减去5作为week\_of\_trading值。同样地，利用extract函数从交易时间中提取处日期标记daypos，daypos从1到5分别对应周一到周五。这样处理后，对周一到周五的每一列，只会选取相应日期的基金购买总金额，否则相应位置设为空值NULL。最终按照周次进行分组并排序。代码如下。

select wk as week\_of\_trading,

    sum(case when daypos = 1 then amount else null end) as monday,

    sum(case when daypos = 2 then amount else null end) as tuesday,

    sum(case when daypos = 3 then amount else null end) as wendnesday,

    sum(case when daypos = 4 then amount else null end) as thursday,

    sum(case when daypos = 5 then amount else null end) as friday

from (

    select date\_part('week', pro\_purchase\_time) -5 as wk,

    extract(DOW FROM cast(pro\_purchase\_time as TIMESTAMP)) as daypos, sum(pro\_quantity \* f\_amount) as amount

    from property

join fund on pro\_pif\_id = f\_id

    where pro\_purchase\_time like '2022-02-%' and pro\_type = 3

    group by pro\_purchase\_time)

group by wk

order by wk;

实现的日历表效果如图2.3.3所示。

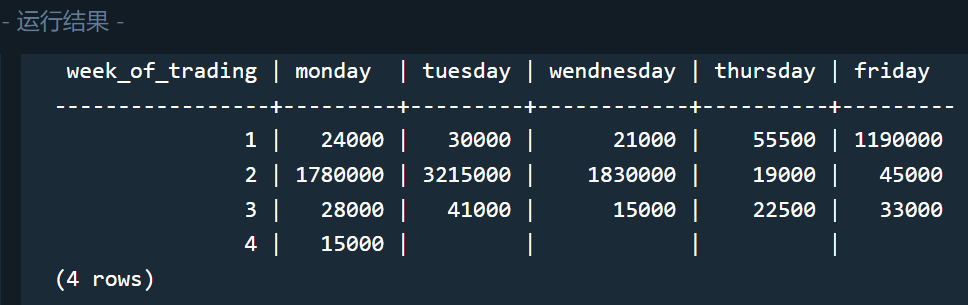


图2.3.3：以日历表格式显示每日基金购买总金额

**2.3.8 查询购买了所有畅销理财产品的客户（第22关）**

**本关任务**：查询购买了所有畅销理财产品的客户编号(pro\_c\_id)。其中，畅销理财产品是指持有人数超过2的理财产品。结果按客户编号升序排列，且去除重复结果。

**实验过程**：这一关和前面2.3.6中“持有完全相同基金组合的客户”类似，也用到了二重NOT EXISTS条件判断语句。“一个用户购买了所有畅销理财产品”可以转化为“不存在这样一个畅销理财产品是这个用户没有购买的”。在第一层NOT EXISTS条件中，利用having count(\*)>2筛选出所有畅销理财产品，并在第二层NOT EXISTS条件中逐个判断即可。代码如下。

select distinct pro\_c\_id from property a

where not exists

(select \* from property b

    where b.pro\_pif\_id in

    (select pro\_pif\_id from property where pro\_type=1

     group by pro\_pif\_id

     having count(\*)>2)

and not exists

(select \* from property c

     where c.pro\_c\_id=a.pro\_c\_id

     and c.pro\_pif\_id=b.pro\_pif\_id and c.pro\_type=1))

order by pro\_c\_id;

**2.3.9 查找相似的理财客户（第25关）**

**本关任务**：查询每位客户(列名：pac)的相似度排名值小于3的相似客户(列名：pbc)列表，以及该客户和其相似客户共同持有的理财产品数(列名：common)和相似度排名值(列名：crank)。相似度的定义是：对于A客户，其购买的理财产品集合为{P}，另所有买过{P}中至少一款产品的其他客户集合为{B}，则{B}中每位用户购买的{P}中产品的数量为他与A客户的相似度值。将{B}中客户按照相似度值降序排列，得到A客户的相同相似度值。按照客户编号升序排列，取前两位客户即为A客户的相似理财客户列表。注意结果先按左边客户编号升序排列，同一个客户的相似客户则按客户相似度排名值顺序排列。

**实验过程**：按照题意逐步实现用户之间相似度的获取与排序。我创建了两个派生表，分别命名为p1和p2，用于方便地两两枚举客户之间的组合。同时，利用count(\*)统计客户之间共同持有的理财产品数，并使用rank() over函数以共有的理财产品数为关键字进行降序排序，得到相似度排名值crank。代码如下。

select \* from

(select pro\_c\_id1 as pac, pro\_c\_id2 as pbc, count(\*) as common, rank() over(partition by pac order by common desc, pbc) as crank

   from

(select pro\_c\_id as pro\_c\_id1, pro\_pif\_id as pro\_pif\_id1

  from property where pro\_type = 1)p1,

(select pro\_c\_id as pro\_c\_id2, pro\_pif\_id as pro\_pif\_id2

    from property where pro\_type = 1)p2

where pro\_c\_id1<>pro\_c\_id2 and pro\_pif\_id1=pro\_pif\_id2

group by pro\_c\_id1, pro\_c\_id2)

where crank<3;

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

本小节主要是在不同场景下对数据库进行数据的插入、修改与删除操作。插入信息时，有完整和不完整两种情形；同时，还涉及到批量插入数据、修改数据属性、连接更新等。

本任务已完成全部6个关卡。

**2.4.1 插入不完整的客户信息（第2关）**

**本关任务**：向客户表client插入一条数据不全的记录。

**实验过程**：观察题目的输入数据，发现缺少了客户表的邮箱(c\_mail)，因此在插入时用NULL赋空值。代码如下。

Insert into client

values(33,'蔡依婷',NULL,'350972199204227621','18820762130','MKwEuc1sc6');

## 2.5 视图

本小节围绕视图的创建和基于视图的查询展开。

本任务已完成全部2个关卡。

**2.5.1 插入不完整的客户信息（第2关）**

**本关任务**：向客户表client插入一条数据不全的记录。

**实验过程**：利用视图的查询只需要将from语句后填写视图view的名称即可，代码较为简单，此处不再展示。

## 2.6 存储过程与事务

本小节围绕视图的创建和基于视图的查询展开。

本任务已完成全部3个关卡。

**2.6.1 使用流程控制语句的存储过程（第1关）**

**本关任务**：数据库中有表fibonacci，用来储存斐波拉契数列的前n项，推导公式为：fib(n) = fib(n-1) + fib(n-2)。现在需要创建存储过程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。

**实验过程**：首先创建存储过程，其语法格式是：create procedure 存储过程名(参数)。存储过程sp\_fibonacci(in m int)包含一个由SQL语句组成的主体，它是由以分号字符分隔的多个语句组成的复合语句。

具体来说，首先在as和begin之间的区域定义存储过程需要用到的变量，其中a、b、temp是递归过程中的三个数据变量，countnum用于记录循环的轮数。对于m取值的不同，函数实现的路径不同。若m小于3，则直接向fibonacci表中插入前两项（0,0）和（1,1）即可；否则，对更大的m，则根据fib(n) = fib(n-1) + fib(n-2)计算下一项fibonacci值即可。代码如下。

create procedure sp\_fibonacci(in m int) as

    declare a int default 0;

    declare b int default 1;

    declare countunm int default 2;

    declare temp int default 0;

begin

    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 > countunm loop

        insert into fibonacci values (countunm, a + b);

        temp := a + b; a:= b; b:= temp; countunm:= countunm+1;

end loop;

end;

**2.6.2 使用游标的存储过程（第2关）**

**本关任务**：实现基于游标的存储过程，其背景信息是：医院的某科室有科室主任1名(亦为医生)，医生若干(至少2名，不含主任)，护士若干(至少4人)，现在需要编写一存储过程自动安排某连续期间的大夜班(即每天00:00-8:00)的值班表，排班规则为：每个夜班安排1名医生、2名护士；值班顺序依工号顺序循环轮流安排(即排至最后1名后再从第1名接着排)；科室主任参与轮值夜班，但不安排周末的夜班，当周末轮至科主任时，主任的夜班调至周一，由排在主任后面的医生依次递补值周末的夜班。本关卡中存储过程的名称为sp\_night\_shift\_arrange，输入参数为start\_date和end\_date，分别指排班的起始时间和结束时间。排班结果直接写入表night\_shift\_schedule。

**实验过程**：这是一个比较复杂的存储过程，主要是要弄清楚游标在整个过程中是怎样发挥作用的。

首先，在as和begin之间的区域声明一些变量：cur\_date表示当前日期，leader\_name表示每晚轮值负责人的姓名，leader\_adjustment表示轮值负责人是否需要调整，d\_name表示医生的姓名，n\_name1和n\_name2分别表示两名护士的姓名，d\_type表示人员类型（1表示轮值负责人，2表示医生，3表示护士）。

接下来，使用游标d\_cursor和n\_cursor从employee表中获取所有类型不为3的人员和类型为3的人员。根据输入的起始日期和结束日期，依次处理每一天直到结束。在此过程中，需要判断当前选定的人员类型是否为1（轮值负责人）且当前日期为周六或周日。这里可能出现三种情况：

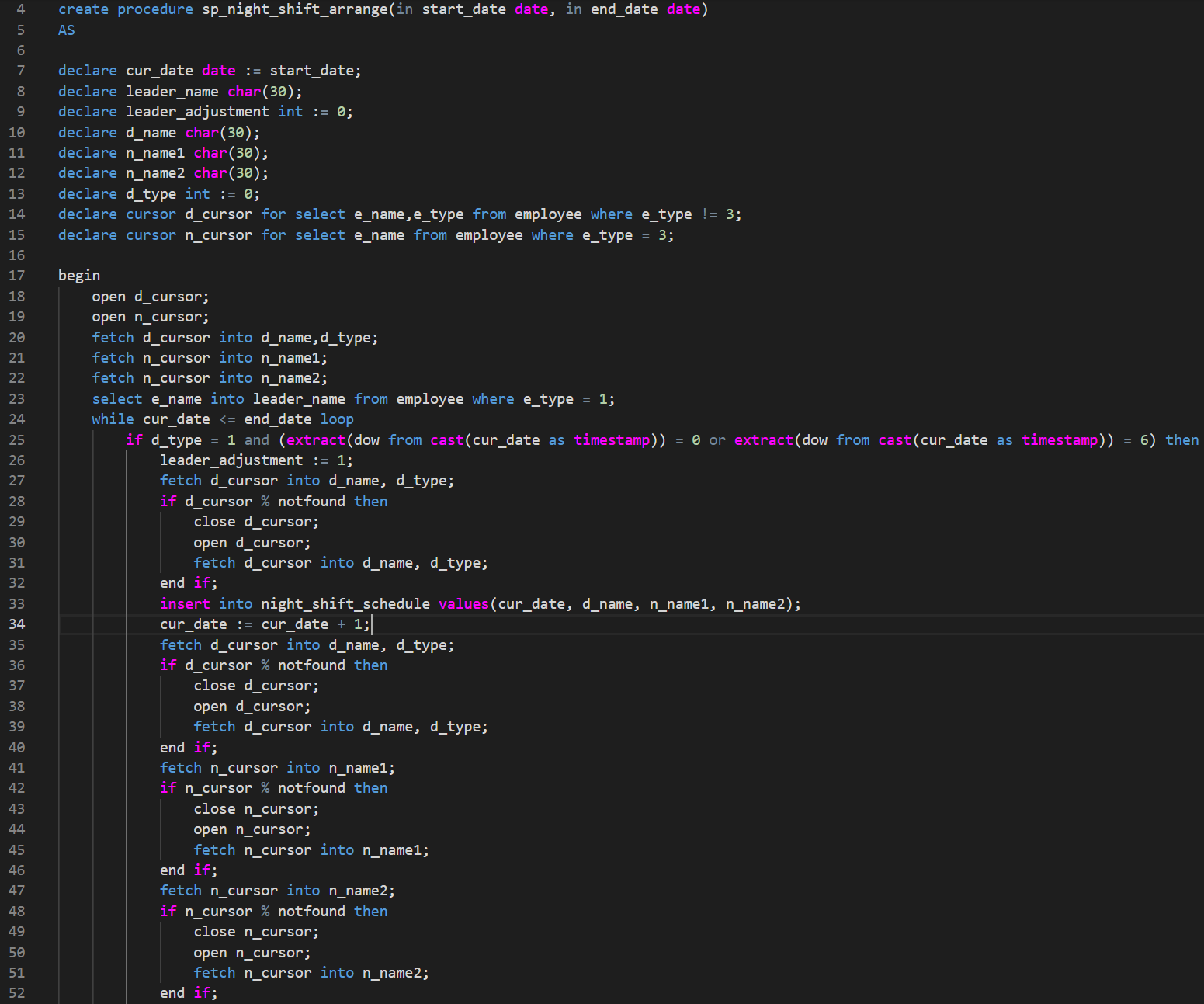
①如果是，则将轮值负责人调整标志设置为1，并获取下一人员插入值班表中，同时将当前日期加1，并获取下一个护士的信息。

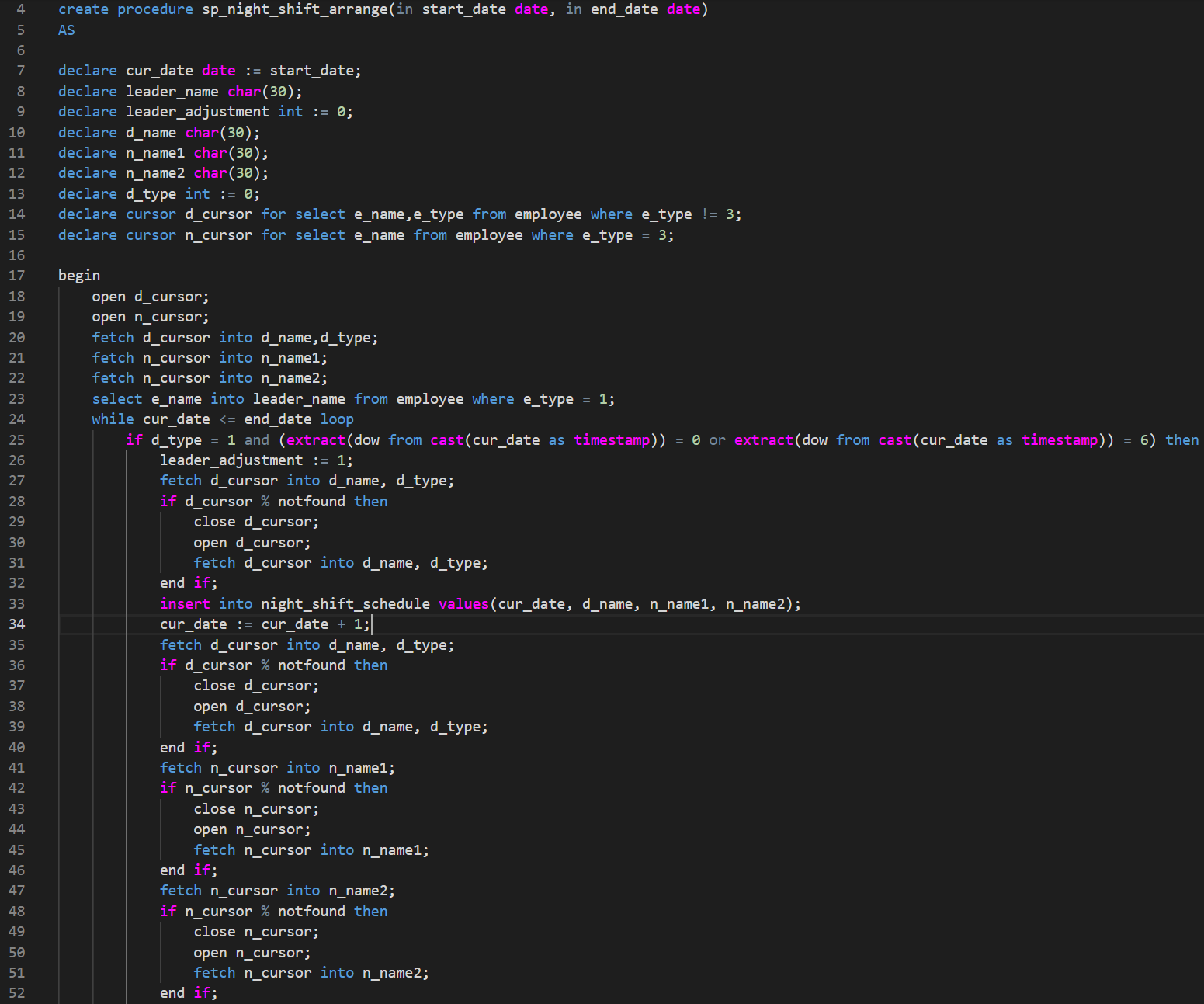
②如果当前日期为周一且轮值负责人调整标志为1，则将调整标志设置为0并将轮值负责人插入值班表中，同时将当前日期加1，并获取下两个护士的信息。

③如果当前日期不为周末且当前人员类型不为1，则将当前人员插入值班表中，同时将当前日期加1，并获取下一个医生和两个护士的信息。

如果游标d\_cursor或n\_cursor已经遍历完，则重新打开游标，继续获取人员信息，直到存储过程执行完毕。

本题最终通关的代码较长，此处使用图片展示，如图2.6.1所示。





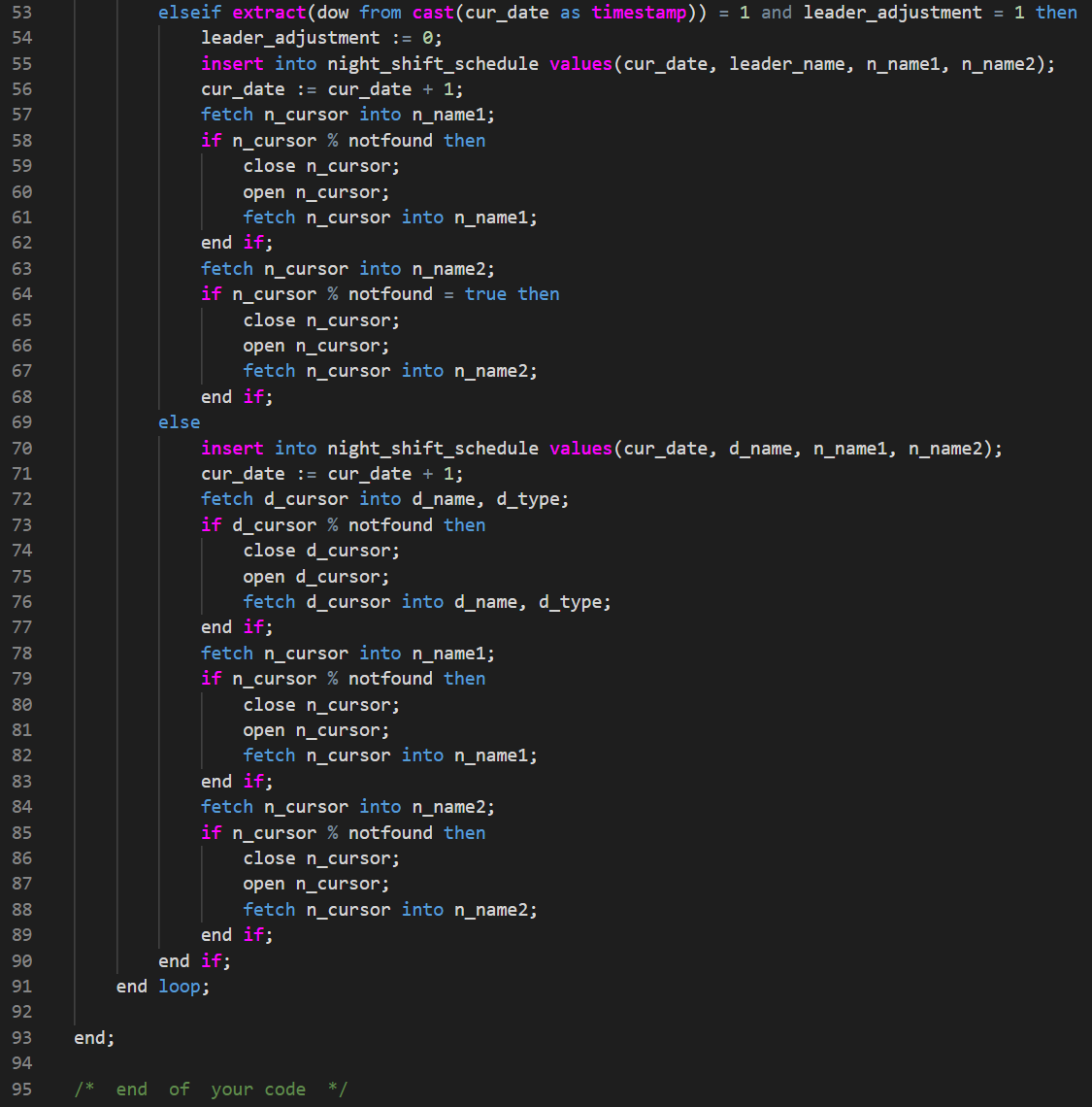


图2.6.1：使用游标的存储过程代码

**2.6.3 使用事务的存储过程（第3关）**

**本关任务**：在金融应用场景数据库中编写一个转账操作的存储过程sp\_transfer，实现从一个帐户向另一个帐户转账的功能。

**实验过程**：根据题意写出存储过程的判断条件，若合法则将返回值置为1并使用COMMIT提交事务，若不合法则将返回值置为0并使用ROLLBACK回滚对数据库的修改。具体条件是：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储蓄卡之间可相互转账；④允许储蓄卡向信用卡转账，称为信用卡还款(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。

实现时只需要判断收款人和转款人的卡类别，并根据上述规则进行条件判断，若合法则使用update对双方的卡余额进行更新。代码如下。

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)

as

begin

    update bank\_card set b\_balance = b\_balance - amount where b\_type = '储蓄卡' and b\_number = source\_card\_id and b\_c\_id = applicant\_id;

    update bank\_card set b\_balance = b\_balance + amount where b\_type = '储蓄卡' and b\_number = dest\_card\_id and b\_c\_id = receiver\_id;

    update bank\_card set b\_balance = b\_balance - amount where b\_type = '信用卡' and b\_number = dest\_card\_id and b\_c\_id = receiver\_id;

    if not exists (select \* from bank\_card where b\_type = '储蓄卡' and b\_number = source\_card\_id and b\_c\_id = applicant\_id and b\_balance >= 0) then

       return\_code := 0; rollback;

    elseif not exists

    (select \* from bank\_card where b\_number = dest\_card\_id and b\_c\_id = receiver\_id) then return\_code := 0; rollback;

    Else return\_code := 1; commit;

    end if; end;

## 2.7 触发器

本小节涉及了触发器的创建和使用。

本任务已完成全部1个关卡。

**2.7.1 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（第1关）**

**本关任务**：为表property(资产表)编写一个触发器，以实现以下完整性业务规则：①如果pro\_type = 1，则pro\_pif\_id只能引用finances\_product表的p\_id；②如果pro\_type = 2，则pro\_pif\_id只能引用insurance表的i\_id；③如果pro\_type = 3，则pro\_pif\_id只能引用fund表的f\_id；④pro\_type不接受(1,2,3)以外的值。

**实验过程**：首先定义msg为出错信息（不超过128个字符），当数据不合法时用raise exception语句抛出异常，并设置出错信息：raise exception '%', msg。接下来根据题目条件构造出错语句格式即可实现触发器业务。代码如下。

CREATE OR REPLACE FUNCTION TRI\_INSERT\_FUNC() RETURNS TRIGGER AS

$$ DECLARE --此处用declare语句声明你所需要的变量

   msg varchar(128);

BEGIN --此处插入触发器业务

if new.pro\_type <> 1 and new.pro\_type <> 2 and new.pro\_type <> 3 then msg := concat('type ',new.pro\_type,' is illegal!');

    raise exception '%',msg; end if;

if new.pro\_type = 1 and not exists(select \* from finances\_product where finances\_product.p\_id = new.pro\_pif\_id) then

    msg := concat('finances product #',new.pro\_pif\_id,' not found!'); raise exception '%',msg; end if;

if new.pro\_type = 2 and not exists(select \* from insurance where insurance.i\_id = new.pro\_pif\_id) then

    msg := concat('insurance #',new.pro\_pif\_id,' not found!');

    raise exception '%',msg; end if;

if new.pro\_type = 3 and not exists(select \* from fund where fund.f\_id = new.pro\_pif\_id) then

    msg := concat('fund #',new.pro\_pif\_id,' not found!');

    raise exception '%',msg; end if;

   --触发器业务结束 return new;--返回插入的新元组

END; $$ LANGUAGE PLPGSQL;

## 2.8 并发控制与事务的隔离级别

本小节包括并发控制中的不可重复读、幻读、主动加锁保证可重复读、可串行化等内容。

本任务已完成全部4个关卡。

**2.8.1 不可重复读（第1关）**

**本关任务**：现有两个并发事务t1和t2，分别定义在t1.sql和t2.sql代码文件中，需要构造“不可重复读”现象。t2是发生不可重复读的事务，t1在t2的两次连续读之间修改了数据。

**实验过程**：首先要弄清楚“不可重复读”的概念。不可重复读是指一个事务读取到某数据后，另一个事务修改了该事务并未修改的数据，但当第一个事务再次读取该数据时发现两次读取的结果不一样。实验中只需将事务隔离级别设置为read uncommitted，并根据任务给定的执行顺序，利用pg\_sleep()函数实现程序内部等待即可。具体来说，只需确保t2的第一次读取在t1读后和修改前，t1在t2提交后读取。两个事务t1和t2的代码如下。

-- 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(t,tickets)

select 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

select pg\_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(t,tickets)

select 1 t, tickets from ticket where flight\_no = 'CZ5525';

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

select pg\_sleep(1); insert into result(t,tickets)

select 1 t, tickets from ticket where flight\_no = 'CZ5525';

-- 事务2

set session transaction isolation level read uncommitted;

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

select pg\_sleep(1); insert into result(t,tickets)

select 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

select pg\_sleep(2); insert into result(t,tickets)

select 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）:

insert into result(t,tickets)

select  2 t, tickets from ticket where flight\_no = 'CZ5525';

commit;

**2.8.2 幻读（第2关）**

**本关任务**：构造两个事务并发执行时的“幻读”现象。

**实验过程**：首先要弄清楚“幻读”的概念。幻读是指一个事务读取到某数据后，另一个事务作了insert或delete操作，导致第一个事务再次读取该数据时发现数据变多了或者变少了(记录数量不一致)。本题中在第1次查询后，事务t2插入了一条航班信息并提交，第2次查询的记录数增多，才能发生“幻读”。因此只需要在事务1中补充一句select pg\_sleep(2)即可。

## 2.9 备份+日志：介质故障与数据库恢复

本小节包括数据库的备份和恢复的具体操作。

本任务已完成全部1个关卡。

**2.9.1 备份和恢复（第1关）**

**本关任务**：给定数据库residents，先运行test1\_1.sh作备份，然后会drop数据residents，最后运行test1\_2.sh来恢复数据，并检查恢复是否成功。

**实验过程**：备份过程中，使用gs\_dump工具对数据库的内容进行导出。gs\_dump的常见参数有：“-U”为连接数据库的用户名，“-W”指定用户连接的密码，“-f”将导出文件发送至指定目录文件，“-p”指定服务器所侦听的TCP端口或本地Unix域套接字后缀以确保连接，“-F”选择导出文件格式。据此即可写出备份的指令语句，同理也可以写出恢复的指令语句，代码如下。

gs\_dump -U gaussdb -W 'Passwd123@123'  -p 5432 residents -f residents\_bak.sql -F t

gs\_restore -U gaussdb -W 'Passwd123@123' residents\_bak.sql -p 5432 -d residents

## 2.10 数据库设计与实现

本小节主要涉及了数据库设计与实现的相关任务，包括从概念模型到OpenGauss实现、从需求分析到逻辑模型等。

本任务已完成2个关卡。

**2.10.1 从概念模型到MySQL实现（第1关）**

**本关任务**：给定一个机票订票系统，系统需要考虑用户（user）、旅客（passenger）、机场（airport）、航空公司（airline）、民航飞机（airplane）、航班常规调度表（flightschedule）、航班表、机票等实体以及它们之间的联系。具体细节E-R图如图所示。



图2.10.1：机票订票系统E-R图

现在需要根据上述信息和所给E-R图，在OpenGauss下实现flight\_booking的语句，包括建表、创建主码、外码、索引、指定缺省、不能为空等约束。所有索引采用B TREE，所有约束的名字不作要求，所有的外码与主码同名，但有两处例外：计划航班和飞行航班都涉及出发机场和到达机场，外码与主码同名会导致同一表有两个同名列。故这两处外码例外处理：出发机场命名为from，到达机场命名为to。

**实验过程**：这是一个很庞大的系统，但实现时一步步按照题目给定的信息编写代码即可逐步完成。

第一步，要完成各个表的建立。建表时，要注意设置主码约束（PRIMARY KEY）、非空约束（NOT NULL）、不可重复约束（UNIQUE）、缺省值约束（DEFAULT）。

第二步，根据题目要求创建所有外码约束（foreign key），并自定义约束的名称。

第三步，依次创建全部的索引，并使用B TREE的形式，以方便查询等操作。

本关的代码较长，此处只列出部分，如图2.10.2所示。

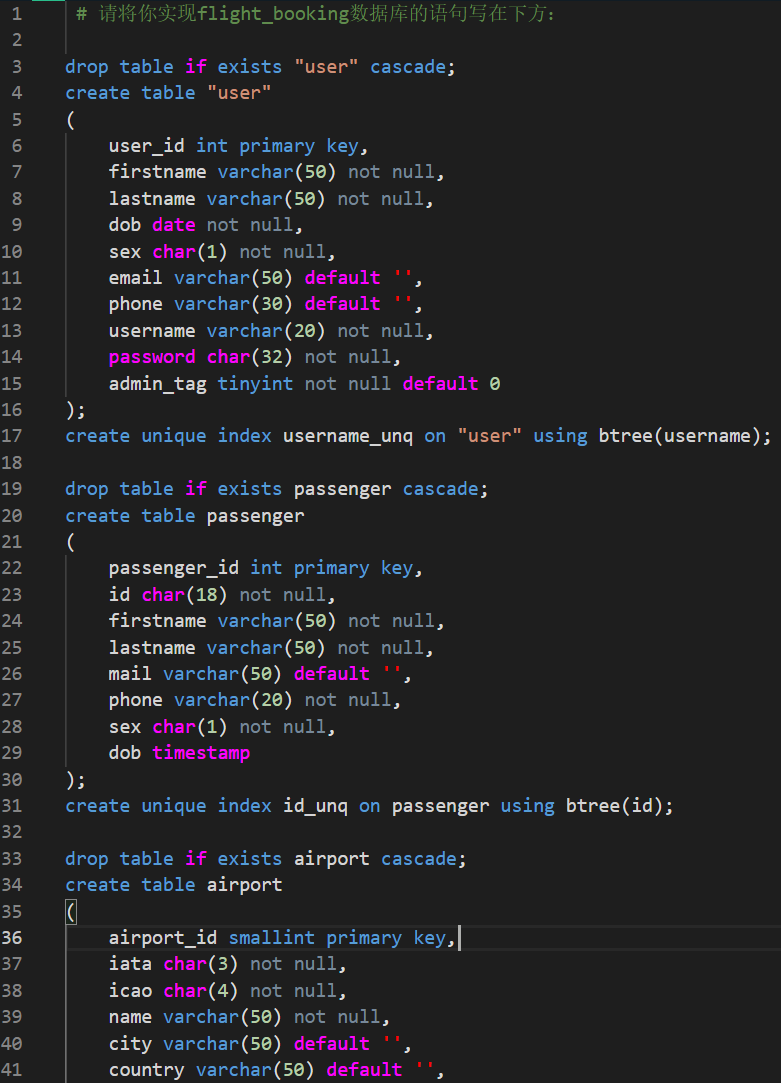


图2.10.2：从概念模型搭配MySQL实现代码（节选）

**2.10.2 从需求分析到逻辑模型（第2关）**

**本关任务**：设计一个影院管理系统。影院对当前的放映厅和电影进行排片，顾客可购买任一排场的电影票，进入对应放映厅观看。系统中有以下实体集：电影、顾客、放映厅、排场、电影票。实体间的关系是：①顾客和电影票有一对多的购买关系；②电影票和排场有多对一的属于关系；③排场和电影有一对多的放映关系；④排场和放映厅有一对多的位于关系。

**实验过程**：根据题意绘制E-R图即可，并分析对应的关系模式。其中，E-R图如图2.10.3所示。

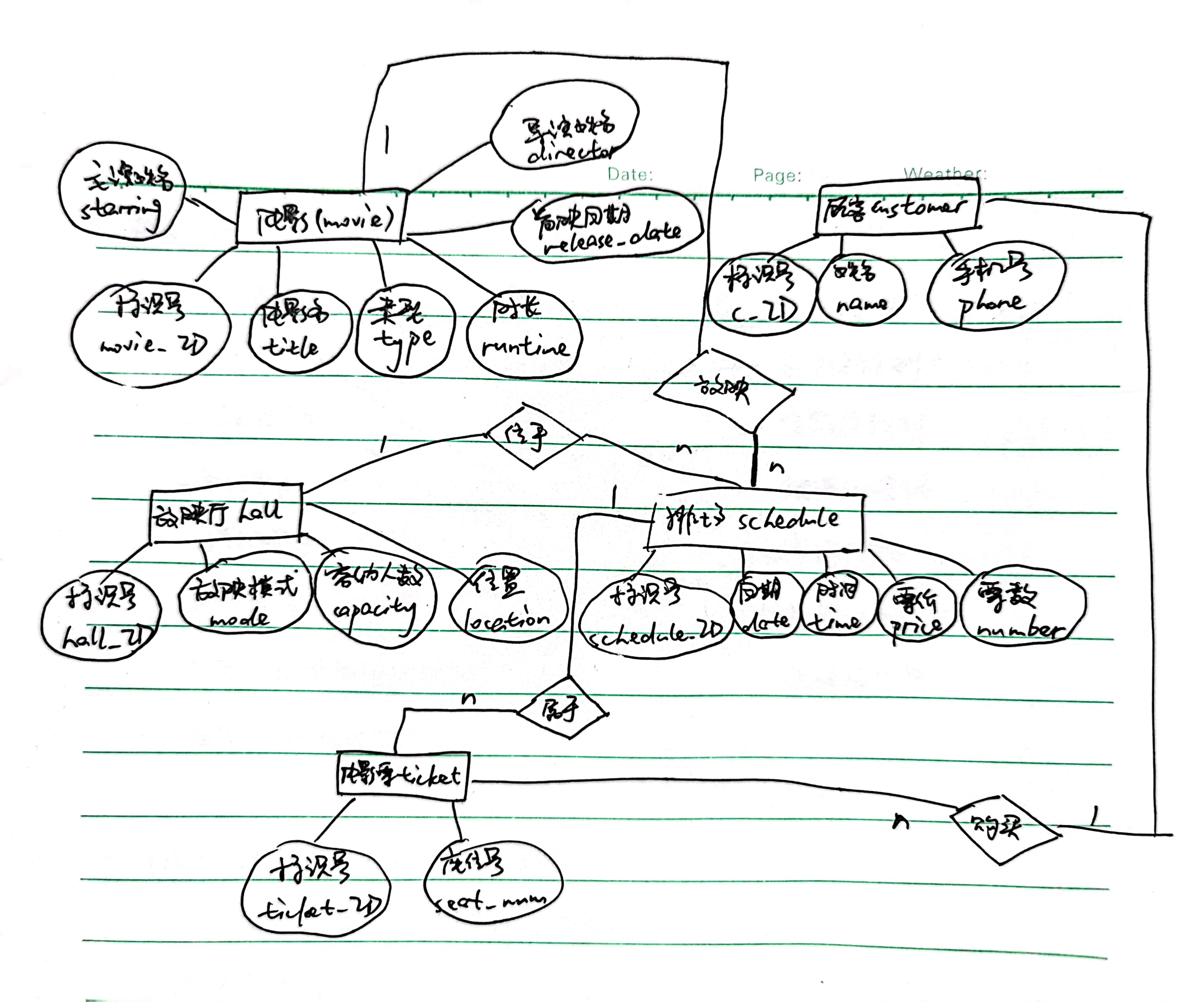


图2.10.3：影院管理系统E-R图

图片链接和关系模式如下。

 请给出ER图文件存放的URL:

https://gitee.com/GOODGAP/database/raw/master/QQ%E5%9B%BE%E7%89%8720230530124437.jpg    //ersolution.jpg

 以下给出关系模式：

电影(movie\_ID, title, type, runtime, release\_date, director, starring)，主码：movie\_ID；

顾客(c\_ID, name, phone)，主码：c\_ID；

放映厅(hall\_ID, mode, capacity, location)，主码：hall\_ID；

排场(schedule\_ID, date, time, price, number)，主码：schedule\_ID；

电影票(ticket\_ID, seat\_num, movie\_ID, phone)，主码：ticket\_ID，外码：movie.movie\_ID, customer.phone

**2.10.3** **制约因素分析与设计**

在进行数据库的设计与实现时，总是会碰到大量的制约因素，这时需要结合实际情况设计符合要求的表结构与数据项，并构建出合理的数据库概念模型、逻辑模型并使用代码实现（例如OpenGauss）。

具体来说，在第一关“机票订票系统”中，首先要设置好主码约束（PRIMARY KEY）、非空约束（NOT NULL）、不可重复约束（UNIQUE）、缺省值约束（DEFAULT）、外码约束（foreign key）等。这些是数据库表的最基本约束，是数据结构的基本属性。其次，在业务流程设计上，同样也有相关的约束，例如每个航班属于一家航空公司，航空公司可以很多航班，这体现了1:n联系；用户可以多次订票，旅客可以多次乘坐飞机，这体现了m:n联系。

又比如在第二关“影院管理系统”中，涉及了用户操作权限的制约。普通用户只能购买电影票，而管理员则可以查看后台数据，包括电影信息、顾客信息、排场信息等。

**2.10.4 工程师责任及其分析**

在本小节中，我深刻认识到了数据库应用所涉及的社会、健康、安全、法律以及文化等方面的重要性。我认为工程师应该有高度的责任心和使命感。

首先，数据库系统的设计和管理直接关系到社会信息化进程的发展，尤其是大数据时代下各行各业都需要运用数据库技术来存储、管理和分析数据，从而提高工作效率和服务质量。例如上面两关分别涉及到的机票订票系统和影院管理系统，都需要使用数据库来管理客户信息、购票结果等敏感数据，确保用户隐私得到保障。

其次，数据库系统的应用也对健康、安全和法律等因素产生了深远的影响。例如在食品安全方面，相关部门可以通过数据库技术收集和分析食品生产、流通、销售等环节的信息，及时发现问题并采取有效措施。同时，数据库系统也需要遵守国家相关法律法规，防止数据泄露和滥用，例如机票信息中包含的个人身份证号就是非常敏感的数据。

综上所述，工程师在从事数据库应用开发和管理时，必须具备责任心，考虑各种因素对数据库系统的影响，并积极采取措施保障数据安全、稳定和可靠性。

## 2.11 数据库的索引B+树实现

本小节主要是对B+树基本数据结构的实现，以及在建立好的B+树上实现insert和remove等操作。

本任务已完成2个关卡。

**2.11.1 BPlusTreePage的设计**

**本关任务**：实现BPlusTreePage类，该类是B+树叶结点类型和内部结点类型的父类，提供B+树结点的基本功能。

**实验过程**：首先分析BPlusTreePage类。它作为BPlusTreeInternalPage与BPlusTreeInternalPage的父类，包含了B+树结点的基本信息和功能，比如结点类型、包含元素存储最大值以及现存元素个数、父结点id、当前结点id等。

接下来按照题意依次实现各个函数的功能。

①判断页类型是否为叶子结点或根结点，通过判断当前页的类型和父节点的page\_id是否为INVALID\_PAGE\_ID实现。

②设置当前页的类型，通过SetPageType函数设置。

③获取或设置当前页存放的元素（键值对）个数，通过GetSize、SetSize和IncreaseSize函数实现。

④获取或设置当前页允许的最大元素个数，通过GetMaxSize和SetMaxSize函数实现。

⑤获取当前页允许的最少元素个数，通过GetMinSize函数实现。如果当前页是根节点，则根节点可能是内部节点或叶子节点。内部节点至少存在两个索引，叶子节点至少存在一条记录。对于非根节点，则至少要求是半满以上，如果是叶结点，至少需要满足[(max\_size\_-1)/2]个元素，如果是内部结点，至少需要满足第一个key不使用，(max\_size\_-1-1+1)/2+1个元素。

⑥获取或设置当前页的父节点page\_id，通过GetParentPageId和SetParentPageId函数实现。

⑦获取或设置当前页的page\_id，通过GetPageId和SetPageId函数实现。

⑧设置LSN（Log Sequence Number），用于恢复和日志管理。

这些操作函数是B+树页的基本操作，支撑B+树的插入、删除、查询等操作。最终各个函数的代码如下。

 \* 函数功能：判断页类型是否为叶子结点

bool BPlusTreePage::IsLeafPage() const {

  return page\_type\_ == IndexPageType::LEAF\_PAGE; }

 \* 函数功能：判断页类型是否为根结点

bool BPlusTreePage::IsRootPage() const {

  return parent\_page\_id\_ == INVALID\_PAGE\_ID; }

 \* 函数功能：设置索引页类型

void BPlusTreePage::SetPageType(IndexPageType page\_type) { page\_type\_ = page\_type; }

 \* 函数功能：get/set size (size：当前结点中存放的元素（键值对）个数)

int BPlusTreePage::GetSize() const { }

void BPlusTreePage::SetSize(int size) { size\_ = size; return; }

void BPlusTreePage::IncreaseSize(int amount) { size\_ += amount; }

 \* 函数功能：get/set max size

int BPlusTreePage::GetMaxSize() const { return max\_size\_; }

void BPlusTreePage::SetMaxSize(int size) { max\_size\_ = size; }

 \* 函数功能：获取当前结点允许的最少元素个数

int BPlusTreePage::GetMinSize() const {

  if (IsRootPage()) return IsLeafPage() ? 1 : 2;

  return (max\_size\_ + 1)/ 2; }

 \*函数功能：get/set parent page id

page\_id\_t BPlusTreePage::GetParentPageId() const { return parent\_page\_id\_; }

void BPlusTreePage::SetParentPageId(page\_id\_t parent\_page\_id) { parent\_page\_id\_ = parent\_page\_id; }

 \*函数功能：get/set self page id

page\_id\_t BPlusTreePage::GetPageId() const { return page\_id\_; }

void BPlusTreePage::SetPageId(page\_id\_t page\_id) { page\_id\_ = page\_id; }

 \*函数功能：set lsn

void BPlusTreePage::SetLSN(lsn\_t lsn) { lsn\_ = lsn; }

# 3 课程总结

本次数据库系统原理实践课是理论与实践的结合、课内与课外的延伸。其总体任务包括对数据库的基本操作、对数据对象的管理与编程、数据库的安全性控制、完整性控制、并发控制机制、数据库的设计与实现，以及数据库的索引B+树的实现。在实验中，我最终通过了66关，获得了133分的总分，对数据库多个维度的操作拥有了更深刻的认识和更熟练的技能。

通过对OpenGauss下SQL语法的学习，我第一次掌握了关系数据库的操作、维护和管理方法。我理解了如何创建数据库、表、索引、视图并定义其中数据项的类型，同时在面对一些实际要求时还需要创建各类约束，如主码约束、外码约束、非空约束等。面对实际情境下的数据库，我学会了对数据库进行更新、查询、备份与恢复，以满足具体的需要。我还学习了函数、触发器的定义与使用方法，探究了事务的并发控制机制，并对B+树这一较复杂的数据结构略有涉足。同时，我还学会了从抽象的概念模型、逻辑模型中建立一个能真实运行的数据库。如果要用一句话来概括我的主要工作，那应该是在浅层学习了数据库操作的各类语法知识和常用操作，并在深层了解了数据库的内核处理思想。

本次数据库实验课给我带来了很丰富的收获。具体来说，我的心得体会有以下几点。在做数据查询时，我发现同一个查询需求可能对应了很多种不同的写法，也就是查询的路径多样化，十分灵活。通过创建索引和视图，可以使查询更方便。存储过程可以用来完成很多功能，例如实现日程的自动化安排。其中，利用游标的存储过程实现起来非常复杂，但最后还是在不断改进中成功通过了评测，让我学会了游标可以对一组数据进行逐行处理，增加了灵活性和精确性。在利用E-R图编写数据库代码时，我再次体会到了概念模型和实际需求分析的紧密联系。

本学期的数据库实验课程就到此结束了，由衷地感谢赵老师和潘老师在课上对我的耐心答疑与指导，同时也要感谢无数个夜晚坚持调bug的自己！