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：

学　　号：

姓　　名：

指导教师：　赵小松

分数	
教师签名	

2023 年 06 月 26 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》是一个与《数据库系统原理》课程配套的实践课程，旨在将理论与实践相结合。本课程以 OpenGauss 为例，通过系统性的设计一系列实训任务，涵盖以下几个主要内容：

1）数据对象的管理与编程：包括数据库、表、索引、视图、约束、存储过程、函数、触发器、游标等数据对象的创建、修改和管理。

2）数据处理相关任务：重点关注数据查询以及数据插入、删除和修改等数据处理操作，通过实践任务来加深对这些操作的理解和掌握。

3）数据库的安全性控制、完整性控制、恢复机制和并发控制机制等系统内核的实验：通过实践任务，学习数据库的安全性控制方法、数据完整性保护策略、故障恢复机制以及并发控制机制等核心概念和技术。

4）数据库的设计与实现：通过实践任务，学习数据库设计的基本原则和方法，了解关系数据库模型的设计过程，并进行实际的数据库设计和实现。

5）数据库应用系统的开发（JAVA 篇）：在数据库应用开发环节，使用 JAVA 1.8 进行实践任务，掌握基于数据库的应用系统开发方法和技巧。

6）数据库内核实验（B+树）：通过实践任务，深入了解数据库内部的数据结构和算法，重点研究 B+树索引的原理、实现和优化方法。

本课程借助头歌实践教学平台进行教学，相关课程的 URL 和邀请码将由教师发布。实验环境为基于 Linux 操作系统的 OpenGauss 2.1 版本，而在数据库应用开发环节，使用 JAVA 1.8 作为开发语言。

通过《数据库系统原理实践》课程的学习，我将深入理解数据库系统的原理和实践，掌握数据库的设计、管理和应用开发的技能，为将来在数据库领域的工作和研究打下坚实的基础。

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头歌平台上的全部关卡，下面将重点针对其中的部分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)

本节的六个关卡围绕数据库和表的创建展开，分别完成了数据库的创建、表的创建以及表中主码、外码、CHECK、DEFAUT 和 UNIQUE 等约束的建立。

2.1.1 创建数据库

该关卡任务已完成，实施情况本报告略过。

2.1.2 创建表及主码约束

本关任务：在指定的数据库中创建一个表，并为表指定主码。

使用 create table 命令创建表 t_emp ，并对主码 id 附加 primary key 约束。如图 2.1.1 所示。

```
CREATE TABLE t_emp
(
    id INT PRIMARY KEY,
    name VARCHAR(32),
    deptid INT,
    salary FLOAT
);
```

图2. 1. 1 创建表和主码约束

2.1.3 创建外码约束

本关任务：创建外码约束（参照完整性约束）。

按关卡要求创建 dept 表和 staff 表，并给表 staff 创建外码，代码如图 2.1.2 所示。

```
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)
);
```

图2.1.2 外码约束

代码中的第 146 行创建了表 staff 的 deptNo 到表 dept 的 deptNo 的外码约束，约束名为 FK_staff_deptno。

2.1.4 CHECK 约束

本关任务：为表创建 CHECK 约束

根据任务要求，创建表 products 并添加 check 约束，代码如图 2.1.3 所示。代码的 10、11 两行对 price 和 brand 的取值进行了约束。

```
CREATE TABLE products
(
    pid CHAR(10) PRIMARY KEY,
    name VARCHAR(32),
    brand CHAR(10),
    price INT,
    CONSTRAINT CK_products_price CHECK(price>0),
    CONSTRAINT CK_products_brand CHECK(brand in ('A', 'B'))
);
```

图 2.1.3 CHECK约束

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)

本节的四个关卡围绕数据库中表的基本修改操作展开，设计用 alter 语句

对表的定义进行修改，如更换/修改表名、列名、列的类型、列约束、表约束； 添加或删除列、约束等。

2.2.1 修改表名

本关任务：修改表的名称，将表名 your_table 更改为 my_table 。 ALTER TABLE 的简化的语法为：

ALTER TABLE 表名[修改事项 [, 修改事项] ...]

根据任务要求选择“RENAME [TO|AS] 新表名” 修改事项，代码如图 2.2.1 所示。

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

图 2.2.1 修改表名

2.2.2 添加与删除字段

该关卡任务已完成，实施情况本报告略过。

2.2.3 修改字段

该关卡任务已完成，实施情况本报告略过。

2.2.4 添加、删除与修改约束

本关任务：添加、删除与修改约束。

对每个要求修选取合适的修改事项即可，代码如图 2.2.2 所示。图 2.2.2 添加、修改与删除约束

```
---请在以下空白处填写适当的语句，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);
```

图 2.2.2 添加、修改与删除约束

2.3 数据查询 (Select)

本节的 25 个关卡由浅到深围绕 select 语句在不同场景下的应用展开。

2.3.1 金融应用场景介绍，查询客户主要信息

该关卡任务已完成，实施情况本报告略过。

2.3.2 邮箱为 null 的客户

本关任务：查询客户表 (client) 中没有填写邮箱的客户的编号、名称、身 份证号、手机号。使用 where 字句即可完成该条件查询，代码如图 2.3.1 所示。

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

图 2.3.1 邮箱为 null

2.3.3 既买了保险又买了基金的客户

本关任务：查询既买了保险又买了基金的客户的名称和邮箱。使用 IN 子查询和 and 来同时满足两个条件即可，代码如图 2.3.2 所示。

```
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 2.3.2 既买了保险又买了基金

2.3.4 办理了储蓄卡的客户信息

该关卡任务已完成，实施情况本报告略过。

2.3.5 每份金额在 30000~50000 之间的理财产品

本关任务：查询金额在 30000 和 50000 之间的理财产品，使用 and 语句即可 实现条件查询，还需要掌握 order 语句进行结果排序，用 desc 关键字使结果降 序排序，代码如图 2.3.3 所示。

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

图 2.3.3 理财产品金额在 30000~5000

2.3.6 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或者数量最大的分组，则为众数，此处用到嵌 套查询和 group by 子句。代码如图 2.3.4 所示。

```
SELECT pro_income, COUNT(pro_income) presence
FROM property
GROUP BY pro_income
HAVING COUNT(*) >= all(SELECT COUNT(*) FROM property GROUP BY pro_income);
```

图 2.3.4 商品收益众数

2.3.7 未购买任何理财产品的武汉居民

该关卡任务已完成，实施情况本报告略过。

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

本关任务：查询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 分组统计收益和并降序排序，利用“limit 3” 输出前 3 名用户即可。注 意 sum（）的使用和限制资产状态不为“冻结”。

代码如图 2.3.5 所示。

```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!='冻结'
GROUP BY c_id
ORDER BY total_income DESC
LIMIT 3;
```

图 2.3.5 总收益前三名

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，实施情况本报告略过。

2.3.13 客户总资产

本关任务：查询客户在本行的总资产。

对于每个客户每种类型的资产总和，可以用一条 select 语句获取，将全部的结果利用 union all 取可重并集，再与用户表 client 做自然连接，最后再按 c_id 分组，统计资产和并降序排序即可。值得注意的一个常识：如果银行卡类 型的信用卡，那么总资产要减去信用卡的 b_balance，即透支金额。还要学习 coalesce（）的使用用来处理空值。最终代码如图 2.3.6 所示。

```sql
select c_id,c_name,coalesce(bc_cx_property,0)-coalesce(bc_xy_property,0)+coalesce
(total_income,0)+coalesce(total_amount,0) as total_property

from client left outer join
    (
        select b_c_id as bc_cx_id,sum(b_balance) as bc_cx_property
        from bank_card
        where b_type='储蓄卡'
        group by b_c_id
    ) as cx on(bc_cx_id = c_id)
    left outer join
    (
        select b_c_id as bc_xy_id,sum(b_balance) as bc_xy_property
        from bank_card
        where b_type='信用卡'
        group by b_c_id
    ) as xy on(bc_xy_id = c_id)
    left outer join
    (
        select pro_c_id as pro_sy_id,sum(pro_income) as total_income
        from property
        group by pro_c_id
    ) as sy on(pro_sy_id = c_id)
    left outer join
    (
        select c_id as pro_amount_id,coalesce(sum(total),0) as total_amount
        from client left outer join
        (
            select pro_c_id,sum(pro_quantity*p_amount) as total
            from property left outer join finances_product on(pro_pif_id = p_id)
            where pro_type=1
            group by pro_c_id

            union all

            select pro_c_id,sum(pro_quantity*i_amount) as total
            from property left outer join insurance on(pro_pif_id = i_id)
            where pro_type=2
            group by pro_c_id

            union all

            select pro_c_id,sum(pro_quantity*f_amount) as total
            from property left outer join fund on(pro_pif_id = f_id)
            where pro_type=3
            group by pro_c_id
        ) as P on (c_id = pro_c_id)
        group by c_id
    ) as amount on(pro_amount_id = c_id)

order by c_id;
```

图 2.3.6 客户总资产

2.3.14 第 N 高问题

本关任务：查询每份保险金额第 4 高保险产品的编号和保险金额。

在保险表 insurance 中利用 dense_rank（）选对不同的保险金额进行排序，再利用 “rank=4”判断 即可获取 select 结果第 4 高的保险金额。最后在外 部查询中判断保险金额是否等于该金额即可，代码如图 2.3.7 所示。

```
select i_id,i_amount
from (
    select i_id,i_amount,dense_rank() over (order by i_amount desc)   as rank
    from insurance
    )
where rank=4;
```

图 2.3.7 第 N 高问题

2.3.15 基金收益两种方式排名

本关任务：对客户基金投资收益实现两种方式的排名次。

以降序的基金投资收益和为关键字，直接调用库函数即可：函数 rank() 是名次不连续的排名，函数 dense_rank() 是名次连续的排名。

名次不连续代码和名次连续代码分别如图 2.3.8，图 2.3.9 所示：

```
-- (1) 基金总收益排名(名次不连续)
select pro_c_id, sum(pro_income) as total_revenue,rank() over (ORDER BY sum
(pro_income) DESC) as rank
from property
where pro_type=3
group by pro_c_id
order by total_revenue desc,pro_c_id;
```

图 2.3.8 名次不连续

```
-- (2) 基金总收益排名(名次连续)

select pro_c_id, sum(pro_income) as total_revenue,DENSE_RANK() over (ORDER BY sum
(pro_income) DESC) as rank
from property
where pro_type=3
group by pro_c_id
order by total_revenue desc,pro_c_id;
```

图 2.3.9 名次连续

2.3.16 持有完全相同基金组合的客户

本关任务：查询持有完全相同基金组合的客户。

首先最内层两个相同的查询，把客户 c_id 和其购买的基金 f_id 的结果保存 到表中，之后进行自然连接得到若干三元组，存储着两个不同客户购买同一

个基 金的信息，根据客户 c_id 进行分组，与另一个查询每个客户购买基金数量的表 进行对比，确认购买同一基金组合的两个客户均满足：这个基金组合是二人购买 10 的全部基金。代码如图 2.3.10 所示。

```
select c1.c_id1,c2.c_id2 from
    (select pro_c_id as c_id1, sum(pro_pif_id) as total,count(*) as c from property
where pro_type='3' group by pro_c_id)as c1,
    (select pro_c_id as c_id2, sum(pro_pif_id) as total,count(*) as c from property
where pro_type='3' group by pro_c_id)as c2
where c1.total=c2.total and c1.c_id1<c2.c_id2 and c1.c=c2.c;
```

图 2.3.10 完全相同基金组合

2.3.17 购买基金的高峰期

本关任务：查询 2022 年 2 月购买基金的高峰期，如果连续三个交易日，投资者购买基金的总金额超过 100 万，则称这连续的几日为投资者购买基金的 高峰期。

在最外层查询的 where 条件中，已经查询了所有当日购买金额大于 1000000 的日期，只需查询以下三种情况，当前日期分别为三个连续金额超过 1000000 的交易日的第一天、第二天和第三天。代码如图 2.3.11 所示。

```
select pro_purchase_time, total_amount as amount
from(
    select pro_purchase_time,total_amount,rn
    from(
        select pro_purchase_time,sum(f_amount*pro_quantity) as total_amount,(rank()
over (order by property.pro_purchase_time)) as rn
        from property,fund
        where pro_type=3 and pro_pif_id=f_id
        group by pro_purchase_time
    )prf1
    where total_amount>=1000000
)
where
rn in(
    select rn+1
    from(
        select pro_purchase_time,sum(f_amount*pro_quantity) as total_amount,(rank()
over (order by property.pro_purchase_time)) as rn
        from property,fund
        where pro_type=3 and pro_pif_id=f_id
        group by pro_purchase_time
    )prf1
    where total_amount>=1000000
)
and
rn in(
    select rn-1
    from(
        select pro_purchase_time,sum(f_amount*pro_quantity) as total_amount,(rank()
over (order by property.pro_purchase_time)) as rn
        from property,fund
        where pro_type=3 and pro_pif_id=f_id
        group by pro_purchase_time
    )prf1
    where total_amount>=1000000
)
or
rn in(
    select rn+1
    from(
        select pro_purchase_time,sum(f_amount*pro_quantity) as total_amount,(rank()
over (order by property.pro_purchase_time)) as rn
        from property,fund
        where pro_type=3 and pro_pif_id=f_id
        group by pro_purchase_time
    )prf1
    where total_amount>=1000000
)
and
rn in(
    select rn+2
    from(
        select pro_purchase_time,sum(f_amount*pro_quantity) as total_amount,(rank()
over (order by property.pro_purchase_time)) as rn
        from property,fund
        where pro_type=3 and pro_pif_id=f_id
        group by pro_purchase_time
    )prf1
    where total_amount>=1000000
)
or
rn in(
    select rn-2
    from(
        select pro_purchase_time,sum(f_amount*pro_quantity) as total_amount,(rank()
over (order by property.pro_purchase_time)) as rn
        from property,fund
        where pro_type=3 and pro_pif_id=f_id
        group by pro_purchase_time
    )prf1
    where total_amount>=1000000
)
and
rn in(
    select rn-1
    from(
        select pro_purchase_time,sum(f_amount*pro_quantity) as total_amount,(rank()
over (order by property.pro_purchase_time)) as rn
        from property,fund
        where pro_type=3 and pro_pif_id=f_id
        group by pro_purchase_time
    )prf1
    where total_amount>=1000000
)


/* end of your code */
```

图 2.3.11 购买基金高峰期

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

本关任务：以日历表格式显示 2022 年 2 月每周每日基金购买总金额。

事先已经得知 2022.2.7 是第六周，因此对 date_part 取得“week”结果减 5 就可以得到在 2 月的周次。对资产表 property 和基金表 fund 做自然连接，并按交易时间是星期几分别计算出当日购买总金额，再按照周次相等 left join 到一起。代码如图 2.3.12 所示。

```
sum(f_amount*pro_quantity) as Monday
from property, fund where pro_type=3 and pro_pif_id=f_id
and pro_purchase_time between '2022-2-1' and '2022-2-28'
group by pro_purchase_time
having to_char(pro_purchase_time,'dd')%7+1=1) mon left join

(select (date_part('week',pro_purchase_time)-5) as week_of_trading ,
sum(f_amount*pro_quantity) as Tuesday
from property, fund where pro_type=3 and pro_pif_id=f_id
and pro_purchase_time between '2022-2-1' and '2022-2-28'
group by pro_purchase_time
having to_char(pro_purchase_time,'dd')%7+1=2) tue
on mon.week_of_trading= tue.week_of_trading
left join

(select (date_part('week',pro_purchase_time)-5) as week_of_trading ,
sum(f_amount*pro_quantity) as Wednesday
from property, fund where pro_type=3 and pro_pif_id=f_id
and pro_purchase_time between '2022-2-1' and '2022-2-28'
group by pro_purchase_time
having to_char(pro_purchase_time,'dd')%7+1=3) wed
on mon.week_of_trading= wed.week_of_trading
left join

(select (date_part('week',pro_purchase_time)-5) as week_of_trading ,
sum(f_amount*pro_quantity) as Thursday
from property, fund where pro_type=3 and pro_pif_id=f_id
and pro_purchase_time between '2022-2-1' and '2022-2-28'
group by pro_purchase_time
having to_char(pro_purchase_time,'dd')%7+1=4) thu
on mon.week_of_trading= thu.week_of_trading
left join

(select (date_part('week',pro_purchase_time)-5) as week_of_trading ,
sum(f_amount*pro_quantity) as Friday
from property, fund where pro_type=3 and pro_pif_id=f_id
and pro_purchase_time between '2022-2-1' and '2022-2-28'
group by pro_purchase_time
having to_char(pro_purchase_time,'dd')%7+1=5) fri
on mon.week_of_trading= fri.week_of_trading;
```

图 2.3.12 日历表格式

2.3.20 查询销售总额前三的理财产品

该关卡任务已完成，实施情况本报告略过。

2.3.21 投资积极且偏好理财类产品的客户

本关任务：找到投资积极且偏好理财类产品的客户。

利用嵌套的两个子查询分别查找用户购买的理财产品数目和基金数目，用客户编号相等连接两个表，找出理财产品数目大于基金数目的客户即可。代码如图 2.3.13 所示。

```sql
select a.pro_c_id
from(
   select count(distinct pro_pif_id) as fi_num,pro_c_id
   from property,finances_product
   where pro_pif_id=p_id and pro_type=1
   group by pro_c_id
   having count(distinct pro_pif_id)>3
)a,
(
   select count(distinct pro_pif_id) as f_num,pro_c_id
   from property,fund
   where pro_pif_id=f_id and pro_type=3
   group by pro_c_id

)b
where  a.pro_c_id=b.pro_c_id and fi_num>f_num
```

图 2.3.13 投资积极且偏好理财的客户

2.3.22 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。

2.3.23 查找相似的理财产品

该关卡任务已完成，实施情况本报告略过。

2.3.24 查询任意两个客户的相同理财产品数

该关卡任务已完成，实施情况本报告略过。

2.3.25 查找相似的理财客户

该关卡任务已完成，实施情况本报告略过。

2.4 数据的插入、修改与删除

本节的 6 个关卡围绕 Insert, Update, Delete 语句在不同场景下的应用 展开。

2.4.1 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。

2.4.2 插入不完整的客户信息

本关任务：向客户表 client 插入一条数据不全的记录。

插入不完整的数据记录需要在 values 前指定插入的列或者把对应空值的列设置为 NULL，代码如图 2.4.1 所示。

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

图 2.4.1 插入不完整的客户信息

2.4.3 批量插入数据

本关任务：向客户表 client 批量插入数据。

将 insert 语句中的 value 字段更换为 select 查询语句即可，因为两个表结构完全相同，代码如图 2.4.2 所示。

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

图 2.4.2 批量插入

2.4.4 删除没有银行卡的客户信息

该关卡任务已完成，实施情况本报告略过。

2.4.5 冻结客户资产

本关任务：冻结客户的投资资产。

使用一条 update 语句结合 exists 子查询即可实现。代码如图 2.4.3 所示。

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

图 2.4.3 冻结客户资产

2.4.6 连接更新

该关卡任务已完成，实施情况本报告略过。

2.5 视图

本节的 2 个关卡围绕视图的创建与使用展开。

2.5.1 创建所有保险资产的详细记录视图

本关任务：创建所有保险资产的详细记录视图。

根据任务要求编写 select 语句，并在前面加上“create view ... as” 即 可，代码如图 2.5.1 所示。

```
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 pro_type='2' AND c_id=pro_c_id AND pro_pif_id=i_id;
```

图 2.5.1 创建视图

2.5.2 基于视图的查询

该关卡任务已完成，实施情况本报告略过。

2.6 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。

2.6.1 使用流程控制语句的存储过程

本关任务：创建一个存储过程，向表 fibonacci 插入斐波拉契数列的前 n 项。

利用“create procedure ... as...begin...end” 定义一个存储过程，在 as 和 begin 之间定义变量并且赋初值，begin 和 end 之间是流程语句，算法的思想是共通的只是语法不太一样，代码如图 2.6.1 所示。

```
create procedure sp_fibonacci(in m int)
as
    DECLARE a INT DEFAULT 0;
    DECLARE b INT DEFAULT 1;
    DECLARE n 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>n LOOP
        INSERT INTO fibonacci VALUES (n,a+b);
        temp := a+b;
        a:=b;
        b:=temp;
        n:=n+1;
    END LOOP;

end;
```

图 2.6.1 流程控制

2.6.2 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。

2.6.3 使用事务的存储过程

本关任务：编写实现转账功能的存储过程。

根据题意编写判断条件，如果不合法则将返回值置为 0 ，并且执行 rollback 恢复对数据库的修改。如果合法，则先判断付款方和收款方的卡类型， 如果是信用卡则要把转账金额设置为负数，最后使用 update 更新即可，然后 commit 提交更改。代码如图 2.6.2 所示。

```
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;
```

图 2.6.2 事务

2.7 触发器

本节的 1 个关卡涉及触发器的使用。

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

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则。

根据任务要求声明“BEFORE INSERT ON property” 类型的触发器， 具体用触发器函数 TRI_INSERT_FUNC（）实现功能，如果执行过程中发现有不满 足条件的情况则利用 concat（）函数编辑报错信息 msg，并通过“raise exception ‘%’,msg”抛出报错信息。代码如图 2.7.1 所示。

```
--创建触发器函数TRI_INSERT_FUNC()
CREATE OR REPLACE FUNCTION TRI_INSERT_FUNC() RETURNS TRIGGER AS
$$
DECLARE
   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;

   return new;--返回插入的新元组
END;
$$ LANGUAGE PLPGSQL;


-- 创建before_property_inserted触发器，使用函数TRI_INSERT_FUNC实现触发器逻辑：
CREATE  TRIGGER before_property_inserted BEFORE INSERT ON property
FOR EACH ROW
EXECUTE PROCEDURE TRI_INSERT_FUNC();
```

图 2.7.1 触发器

2.8 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。

2.8.1 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储总额的函数，并在 SELECT 语句使用这个函数。

根据任务要求和实验指导编写代码，函数的语法格式是固定的很简单。代码

如图 2.8.1 所示。

```
/*
   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:*/

CREATE OR REPLACE FUNCTION get_deposit(client_id integer)
 returns numeric(10,2)
 LANGUAGE plpgsql
AS
$$
declare result numeric(10,2);
begin
   select sum(b_balance) INTO result
   from bank_card
   where b_c_id=client_id and b_type='储蓄卡';
   return result;

end;
$$;

/*  应用该函数查询存款总额在100万(含)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 get_deposit(c_id) desc;



/*  代码文件结束     */
/*  代码文件结束     */
```

图 2.8.1 自定义函数

2.9 安全性控制

本节的 2 个关卡涉及数据库中的用户、角色和权限等内容。

2.9.1 用户和权限

该关卡任务已完成，实施情况本报告略过。

2.9.2 用户、角色与权限

本关任务：创建角色，授予角色一组权限，并将角色代表的权限授予指定的一组用户。

根据任务要求合理使用 create...with password, grant...on...to 语句编写指令即可，代码如图 2.9.1 所示。

```
-- (1) 创建角色client_manager和fund_manager;
create role client_manager with password 'hust_1234';
create role fund_manager with password 'hust_1234';
-- (2) 授予client_manager对client表拥有select,insert,update的权限;
grant select,insert,update on table 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_type,b_number,b_c_id) on table bank_card to client_manager;
-- (4) 授预fund_manager对fund表的select,insert,update权限;
grant select,insert,update on table fund to fund_manager;
-- (5) 将client_manager的权限授予用户tom和jerry;
grant client_manager to tom,jerry;
-- (6) 将fund_manager权限授予用户Cindy.
grant fund_manager to cindy;
```

图 2.9.1 安全性控制

2.10 并发控制与事务的隔离级别

本节的 4 个关卡涉及数据库中并发控制与事务的隔离级别相关内容，包括隔 离级别的设置等，还通过添加等待代码实现了不可重复读、幻读等出错场景。

2.10.1 不可重复读

本关任务：设置事务的隔离级别。

数据库是共享资源，允许多个用户同时访问同一数据库，特别是在互联网应用成为主流的当下，高可用性、高并发是所有应用追求的目标。但并发操作不加控制，便会产生数据的不一致性。 并发操作可能带来的数据不一致性包括：

1) 丢失修改(lost update)

2) 读脏数据(dirty read)

3) 不可重复读(non-repeatable read)

4) 幻读(phantom read)

最低的隔离级别不能避免读脏、不可重复读和幻读，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，但其代价是并发度最低。

根据任务要求将事务的隔离级别设置为 read uncommitted，并且根据任务要求的执行顺序利用 pg_sleep（）函数实现程序内部的等待，t1.sql 和 t2.sql 的代码分别如图 2.10.1 和图 2.10.2 所示。

```sql
-- 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';
--select pg_sleep(3);

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';
```

图 2.10.1

```sql
-- 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 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;
```

图 2.10.2

2.10.2 幻读

本关任务：本关要求大家在较高隔离级别，即仅次于 serializable 的 repeatable read 隔离级别下重现“幻读”现象，这样，可更好地体验不可重复 读与幻读的区别。

两次查询余票超过 300 张的航班信息(第 2 次查询已替你写好)；在第 1 次查 询之后，事务 t2 插入了一条航班信息并提交(t2.sql 已替你写下好)；第 2 次查 询的记录数增多,发生“幻读”。代码如图 2.10.3 所示。

```
-- 事务1（采用事务隔离级别- read committed）：
set session transaction isolation level repeatable read;

-- 开启事务
start transaction;

-- 第1次查询余票超过300张的航班信息
insert into result
select *,1 t from ticket where tickets > 300;

-- 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 pg_sleep(2);

insert into result
select *,2 t from ticket where tickets > 300;
commit;



-- 事务2（采用事务隔离级别- read committed）：
set session transaction isolation level repeatable read;

-- 开启事务
start transaction;

select  pg_sleep(1);
insert into ticket values('MU5111','A330-200',311);
commit;
```

图 2.10.3 幻读

2.10.3 主动加锁保证可重复读

该关卡任务已完成，实施情况本报告略过。

2.10.4 可串行化

该关卡任务已完成，实施情况本报告略过。

2.11 数据库应用开发（JAVA 篇）

本实训 7 个关卡要求结合 java 和数据库语言实现一个简单的数据库应用。

2.11.1 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。

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, 26 String user, String password) 方法建立连接。在使用 Statement 对象执行 SQL 语 句之前，需要使用 Connection 对象的 createState- ment() 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 有三个执行方法：

1.boolean execute(String SQL) ：如果可以检索到 ResultSet 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。当希望获得结果集时，使用此方法，就像使用 SELECT 语句一样。

本关需要选用第 3 个执行方法，使用 resultSet.next() 遍历 ResultSet 输出相应的信息即可，代码如图 2.11.1 所示。

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

        try {
            Class.forName("org.postgresql.Driver");
            String URL = "jdbc:postgresql://localhost:5432/postgres";
            String USER = "gaussdb";
            String PASS = "Passwd123@123";
            Connection conn = DriverManager.getConnection(URL, USER, PASS);
            statement = conn.createStatement();
            String sql = "select c_name,c_mail,c_phone from client where c_mail i
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.print(resultSet.getString("c_name")+"\t");
                System.out.print(resultSet.getString("c_mail")+"\t\t");
                System.out.print(resultSet.getString("c_phone")+"\n");
            }

         } 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 2.11.1 JDBC体系结构和简单查询

2.11.2 用户登录

该关卡任务已完成，实施情况本报告略过。

2.11.3 添加新客户

本关任务：编程完成向客户表 client 插入记录的方法。 编写 SQL 语句，用 PreparedStatement 类执行即可，代码如图 2.11.2 所示。

```
/**
 * 向Client表中插入数据
 *
 * @param connection 数据库连接对象
 * @param c_id 客户编号
 * @param c_name 客户名称
 * @param c_mail 客户邮箱
 * @param c_id_card 客户身份证
 * @param c_phone 客户手机号
 * @param c_password 客户登录密码
 */
public static int insertClient(Connection connection,
                               int c_id, String c_name, String c_mail,
                               String c_id_card, String c_phone,
                               String c_password){
        String sql = "insert into client values(?,?,?,?,?,?)";
        PreparedStatement statement = null;
        int n = -1;
        try{
            statement = connection.prepareStatement(sql);;
            statement.setInt(1,c_id);
            statement.setString(2,c_name);
            statement.setString(3,c_mail);
            statement.setString(4,c_id_card);
            statement.setString(5,c_phone);
            statement.setString(6,c_password);
            n = statement.executeUpdate();
        }catch(Exception e){

        }
        return n;
}
```

图 2.11.2 添加新客户

2.11.4 银行卡销户

该关卡任务已完成，实施情况本报告略过。

2.11.5 客户修改密码

该关卡任务已完成，实施情况本报告略过。

2.11.6 转账与事务操作

本关任务：编写一个银行卡转账的方法。根据题意编写判断条件，如果不合法则直接 return false ，然后判断收款方卡 类型是否为信用卡决定是否要把收款金额设置为负数。代码如图 2.11.3 所示。（main 函数不做修改已省略）

```java
    /**
     * 转账操作
     *
     * @param connection 数据库连接对象
     * @param sourceCard 转出账号
     * @param destCard 转入账号
     * @param amount  转账金额
     * @return boolean
     *   true  - 转账成功
     *   false - 转账失败
     */
    public static boolean transferBalance(Connection connection,
                                          String sourceCard,
                                          String destCard,
                                          double amount){
        try{
            String type = null;
            connection.setAutoCommit(false);
            boolean rollback = false;
            PreparedStatement query = connection.prepareStatement("select * from
bank_card where b_number=?");
            query.setString(1,sourceCard);
            ResultSet resultSet = query.executeQuery();
            if(resultSet.next()){
                type = resultSet.getString("b_type");
                int balance = resultSet.getInt("b_balance");
                if(balance < amount || "信用卡           ".equals(type)){
                    rollback = true;
                }
            }else{
                rollback = true;
            }
            query.setString(1,destCard);
            resultSet = query.executeQuery();
            if(!resultSet.next()) rollback = true;
            else type = resultSet.getString("b_type");
            if(rollback) return false;
            PreparedStatement statement1 = connection.prepareStatement("update
bank_card set b_balance = b_balance - ? where b_number = ?");
            statement1.setDouble(1,amount);
            statement1.setString(2,sourceCard);
            PreparedStatement statement2 = null;
            if("信用卡           ".equals(type)){
                statement2 = connection.prepareStatement("update bank_card set
b_balance = b_balance - ? where b_number = ?");
            }else{
                statement2 = connection.prepareStatement("update bank_card set
b_balance = b_balance + ? where b_number = ?");
            }
            statement2.setDouble(1,amount);
            statement2.setString(2,destCard);
            statement1.executeUpdate();
            statement2.executeUpdate();
            connection.commit();
            return !rollback;
        }catch (Exception e){
            try {
                connection.rollback();
            } catch (SQLException ex) {
                ex.printStackTrace();
            }
            e.printStackTrace();
        }
        return false;
    }
```

图 2.11.3 转账与事务操作

2.11.7 把稀疏表格转化为键值对存储

该关卡任务已完成，实施情况本报告略过。

2.12 备份 + 日志：介质故障与数据库恢复

本节的 1 个关卡涉及数据库的备份与恢复方法。

2.12.1 备份与恢复

本关任务：备份数据库，然后再恢复它。

使用 gs_dump 将数据库备份到文件中，代码如图 2.12.1 所示。

```
gs_dump -U gaussdb -W 'Passwd123@123'  -p 5432 residents -f residents_bak.tar -F t
```

图 2.12.1 备份

使用 gs_restore 从文件中恢复数据库，代码如图 2.12.2 所示。

```
gs_restore -U gaussdb -W 'Passwd123@123' residents_bak.tar -p 5432 -d residents
```

图 2.12.2 恢复

2.13 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 OpenGauss 实现、E-R 图的构建、建模工具的使用等。

2.13.1 从概念模型到 OpenGauss 实现

该关卡任务已完成，实施情况本报告略过。

2.13.2 从需求分析到逻辑模型

本关任务：根据应用场景业务需求描述，完成 E-R 图，并转换成关系模式。根据任务要求描述转换得到的 E-R 图如图 2.13.1，代码如图 2.13.2 所示。

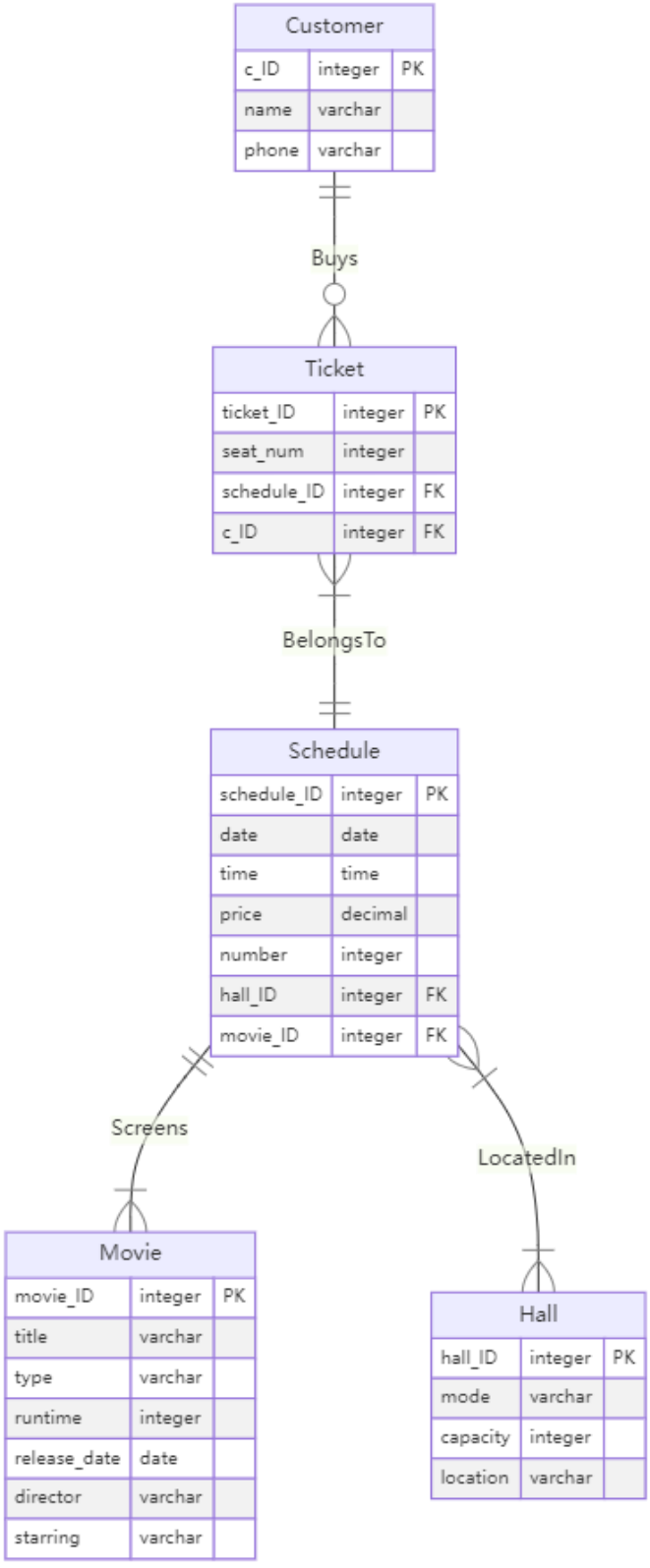
Customer
c_ID integer PK
name varchar
phone varchar
Buys
Ticket
ticket_ID integer PK
seat_num integer
schedule_ID integer FK
c_ID integer FK
BelongsTo
Schedule
schedule_ID integer PK
date date
time time
price decimal
number integer
hall_ID integer FK
movie_ID integer FK
Screens
LocatedIn
Movie
movie_ID integer PK
title varchar
type varchar
runtime integer
release_date date
director varchar
starring varchar
Hall
hall_ID integer PK
mode varchar
capacity integer
location varchar

图 2.13.1 E-R 图

```
请给出ER图文件存放的URL:
https://raw.githubusercontent.com/thezmmm/Gallery/master/other/ersolution.jpg
以下给出关系模式:
顾客(c_ID, name, type, phone), primary key:(c_ID);
电影票(ticket_ID, seat_num, schedule_ID), primary key(ticket_ID), foreign key
(schedule_ID);
排场(schedule_ID, date, time, price, number, movie_ID, hall_ID), primary key:
(schedule_ID), foreign key(movie_ID, hall_ID);
放映厅(hall_ID, mode, capacity, location), primary key:(hall_ID);
电影(movie_ID, title, type, runtime, release_date, director, starring), primary key:
(movie_ID);
```

图 2.13.2 逻辑模型

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。

2.14 数据库的索引 B+树实现

本实训的 5 个关卡目的在于进行 B+数基本数据结构的实现和简单操作比如 insert 和 remove 的实现。

2.14.1 BPlusTreePage 的设计

本关任务：作为 B+树索引结点类型的数据结构设计的第一部分：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。

BPlusTreePage 作为 BPlusTreeInternalPage 与 BPlusTreeInternalPage 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，如图 2.14.1 所示。

Variable Name	Size	Description
page_type_	4	Page Type (internal or leaf)
lsn_	4	Log sequence number (Used in Project 4)
size_	4	Number of Key & Value pairs in page
max_size_	4	Max number of Key & Value pairs in page
parent_page_id_	4	Parent Page Id
page_id_	4	Self Page Id

图 2.14.1

2.14.2 BPlusTreeInternalPage 的设计

该关卡任务已完成，实施情况本报告略过。

2.14.3 BPlusTreeLeafPage 的设计

该关卡任务已完成，实施情况本报告略过。

2.14.4 B+树索引：Insert

该关卡任务已完成，实施情况本报告略过。

2.14.5 B+树索引：Remove

本关任务：实现 B+树索引的删除操作。 一颗完整的 B+树由内部节点和叶子节点组成，随着记录的删除，节点不断 合并减少，但仍需保持 B+树的完整性和一致性。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。首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。

此处进行几个关键函数的展示：

1. void BPLUSTREE_TYPE::Remove(const KeyType &key)

删除功能的主函数，主要依照删除逻辑设计，当 B+树为空时直接返回；不为空时首先调用 FindLeafPage（）函数找到删除值所在的页；删除记录后，根据 不同情况，要么与兄弟节点合并，要么进行元素调动。代码如图 2.14.2 所示。

```
/*
 * 函数功能：
 *  在B+树中删除key对应的记录
 * 建议：
 *  1.若B+树为空直接返回
 *  2.通过上一个关卡实现的FindLeafPage(key)查找key值所在页并删除记录
 *  3.删除后需判断结点元素是否小于min_size,并调用CoalesceOrRedistribute()进行后续处理
 *  4.注意unpinPage，避免缓冲区内存泄露
 */
INDEX_TEMPLATE_ARGUMENTS
void BPLUSTREE_TYPE::Remove(const KeyType &key) {
  std::lock_guard<std::mutex> lock(latch_);
  if (IsEmpty()) return;
  Page *page = FindLeafPage(key);
  assert(page != nullptr);
  B_PLUS_TREE_LEAF_PAGE_TYPE *leaf = reinterpret_cast<B_PLUS_TREE_LEAF_PAGE_TYPE *>
(page->GetData());
  //printf("Remove key:%ld\n", key.ToString());
  int size = leaf->RemoveAndDeleteRecord(key, comparator_);
  if (size < leaf->GetMinSize()) CoalesceOrRedistribute(leaf);
  //printf("%s\n", this->ToString(true).c_str());
  buffer_pool_manager_->UnpinPage(page->GetPageId(),true);
  return;
}
```

图 2.14.2 Remove（）函数

2. bool BPLUSTREE_TYPE::CoalesceOrRedistribute(N *node)

判断是否可以合并，可以的话执行操作，如不行则改为调用相应函数进行元素调动。代码如图 2.14.3 所示。

```
/*
 * 函数功能:
 *  对元素个数小于Min_size的结点进行处理
 *      @param N *node 表示内部结点或叶结点
 *      @return: 若有结点合并，返回true，否则返回false
 * 建议:
 *  1.当node为根结点时，需要调用AdjustRoot()进行调整
 *  2.首先要通过调用FindSibling()找到兄弟结点
 *  3.如果两个结点的元素可以合并，则调用Coalesce()进行合并
 *  4.如果元素过多，无法合并，则调用Redistribute()进行结点元素的调整
 */
INDEX_TEMPLATE_ARGUMENTS
template <typename N>
bool BPLUSTREE_TYPE::CoalesceOrRedistribute(N *node) {
  //如果可以合并：进行页的选择和合并
  //如果无法合并，则选择从其他页抽取一个结点，先找兄弟FindSibling
  if (node->IsRootPage()) {
    //该函数可能递归调用至根结点，故需要对根结点进行特殊处理
    return AdjustRoot(node);
  }
  N *sibling;
  // result 为true代表此时node为父页的首结点
  bool result = FindSibling(node, sibling);
  Page *parent = buffer_pool_manager_->FetchPage(node->GetParentPageId());
  B_PLUS_TREE_INTERNAL_PAGE *parentPage = reinterpret_cast<B_PLUS_TREE_INTERNAL_PAGE
*>(parent->GetData());
  if (node->GetSize() + sibling->GetSize() <= node->GetMaxSize()) {
    //进行融合
    int removedIndex;
    //进行if判断，保证在父页删除的key为父页的非首结点
    if (result) {
      //此时需要把后继页合并到首结点上
      removedIndex = parentPage->ValueIndex(sibling->GetPageId());
      Coalesce(&node, &sibling, &parentPage, removedIndex);
    } else {
      //当前结点合并到前驱结点上
      removedIndex = parentPage->ValueIndex(node->GetPageId());
      Coalesce(&sibling, &node, &parentPage, removedIndex);
    }

    buffer_pool_manager_->UnpinPage(parentPage->GetPageId(), true);
    return true;
  }
  int nodeIndex = parentPage->ValueIndex(node->GetPageId());
  Redistribute(sibling, node, nodeIndex);  // unpin node,node2
  buffer_pool_manager_->UnpinPage(parentPage->GetPageId(), false);
  return false;
}
```

图 2.14.3 CoalesceOrRedistribute（）函数

3. bool BPLUSTREE_TYPE::Coalesce(N **neighbor_node, N **node, BPlusTreeInternalPage **parent， int index)

本函数具体实现合并功能，并且在执行过程中，如果父节点也变得不满足

min_size_的条件，则还需递归调用 CoalesceOrRedistribute（）函数，直到调整 好所有结点。代码如图 2.14.4 所示。

```
/*
 * 函数功能：
 *  将node中的元素全部合并到neighbor_node中
 *      @param   neighbor_node      幸存结点
 *      @param   node               待合并结点
 *      @param   parent             二者的父结点
 *      @param   index              待合并结点在父结点中的索引
 *      @return: 若有结点被删除，返回true，否则返回false
 * 建议：
 *  1.node合并后需对父结点进行相关调整
 *  2.父结点元素被删除后，若小于Min_size则调用CoalesceOrRedistribute处理
 *  3.注意unpinPage和deletePage
 */
INDEX_TEMPLATE_ARGUMENTS
template <typename N>
bool BPLUSTREE_TYPE::Coalesce(N **neighbor_node, N **node,
                              BPlusTreeInternalPage<KeyType, page_id_t,
KeyComparator> **parent, int index) {
  assert((*node)->GetSize() + (*neighbor_node)->GetSize() <= (*node)->GetMaxSize());
  (*node)->MoveAllTo(*neighbor_node,buffer_pool_manager_);
  page_id_t pId = (*node)->GetPageId();
  buffer_pool_manager_->UnpinPage(pId, true);
  buffer_pool_manager_->DeletePage(pId);
  buffer_pool_manager_->UnpinPage((*neighbor_node)->GetPageId(), true);
  (*parent)->Remove(index);
  //当内部页中size过小时或者仅剩余一个有效结点时，递归触发CoalesceOrRedistribute函数，对当
前结点进行处理
  if ((*parent)->GetSize() < (*parent)->GetMinSize() || (*parent)->GetSize() == 2) {
    return CoalesceOrRedistribute(*parent);
  }
  return false;
}
```

图 2.14.4 Coalesce（）函数

3 课程总结

在本次课程中，我完成了 14 个实训实验，涵盖了数据库的多个方面，包括数据库和表的定义、完整性约束的创建和修改、数据查询、数据的插入、修改和删除、视图的创建和使用、存储过程与事务、触发器、用户自定义函数、安全性控制、并发控制与事务隔离级别、数据库应用开发、数据库的备份与日志、数据库的设计与实现以及 B+树实现，总计涉及 70 个关卡。

在实施课程实验的过程中，我合理利用搜索引擎查阅相关资料，学习了 OpenGauss 的一些复杂语法和一些经典问题的简单处理方法，从而简化编程过程。我们能够熟练地使用 OpenGauss 实现一些复杂的查询，并且深刻感受到 OpenGauss 的灵活性。在许多关卡中，实现的方法并不是固定的，熟悉内置函数有时会带来意想不到的简化。编程的关键在于逻辑设计，首先要在脑海中构建一个完整的查询框架，不能一步一步进行思考。

在基于 JDBC 的数据库应用开发中，我深刻体会到了代码安全的重要性。如果不遵守规范，就可能会面临 SQL 注入攻击带来的严重后果。本次课程实验内容充实而完整，具有很强的引导性，完成实验后收获颇丰。尤其是最后的 B+树实现，难度较大，具有很大的挑战性，同时也让我学习了一种新的数据结构，对于学习操作系统等其他课程也有很大的帮助。

最后，我衷心感谢课程组的老师们对实验内容的精心打造，以及在实验过程中给予的悉心指导。他们的辛勤工作和指导让我们能够深入理解数据库系统的原理和实践，并获得宝贵的经验和知识。