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： | CS2007 |
| 学　　号： | U202015512 |
| 姓　　名： | 孙溪 |
| 指导教师： | 赵小松 |

| | |
|---|---|
| 分数 | |
| 教师签名 | |

2022 年 12 月 30 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

课程的重点是将数据库系统原理和数据库系统原理实践，理论与实践相结合，使用 OpenGauss 作为例子。课程包括一系列实践任务，涵盖以下内容：

1. 管理和编程数据库对象，如表、索引、视图、约束、存储过程、函数、触发器和游标。
2. 与数据处理相关的数据查询、插入、删除和修改任务。
3. 对数据库安全控制、完整性控制、恢复机制以及并发控制机制等核心系统进行实验。
4. 数据库的设计和实现。
5. 数据库应用系统的开发（Java 版）。
6. 数据库内核实验（B+树）。

该课程使用头歌实践教学平台，实践课程网址可在相关教室教师发布的链接及邀请码中找到。实验环境为 Linux 操作系统下的 OpenGauss 2.1。在数据库应用开发阶段，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)

关卡主要关于数据库和表的创建，包括创建数据库、创建表以及在表中建立主键、外键、CHECK、DEFAULT 和 UNIQUE 约束的任务。

### 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

在指定的数据库中创建一个表，并为表指定主码。使用 create table 关键语句进行表创建，并用 primary key 指定主码。代码如下：

```
create table t_emp(
    id INT PRIMARY KEY,
    name VARCHAR(32),
    deptld INT,
    salary FLOAT
);
```

### 2.1.3 创建外码约束

外码约束主要通过 foreign key 来指定，创建两表之间的关联。关键代码为 staff 中的 CONSTRAINT 语句，指定外码与 dept 表的关联。代码如下：

```
create table dept(
    deptNo int primary key,
    deptName varchar(32)
);
create table staff(
    staffNo int primary key,
    staffName varchar(32),
    gender char(1),
    dob date,
    salary numeric(8,2),
```

```
    deptNo int,
    CONSTRAINT FK_staff_deptNo foreign key(deptNo) references dept(deptNo)
);
```

### 2.1.4 CHECK 约束

为表创建 CHECK 约束，创建表 products，并分别实现对品牌和价格的约束，两个 CHECK 约束的名称分别为 CK_products_brand 和 CK_products_price，主码约束不显示命名。通过 constraint 语句设置约束，关键代码如下：

```
create table products(
    pid char(10) primary key,
    name varchar(32),
    brand char(10),
    price int,
    constraint CK_products_brand check(brand in ('A','B')),
    constraint CK_products_price check(price > 0)
);
```

### 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)

这一节包含四个任务，重点是数据库中表的基本修改操作。这些任务涉及使用 alter 语句来修改表的定义，如更换/修改表名、列名、列的类型、列约束、表约束、添加或删除列和约束。

### 2.2.1 修改表名

该关卡任务已完成，实施情况本报告略过。

### 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

包括删除和添加主码约束；删除和添加外码约束；删除和添加 CHECK 约束；删除和添加 UNIQUE 约束。代码如图 2.2.1 所示：

```
---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 2.2.1 添加、修改与删除约束

## 2.3 数据查询 (Select)

本实训采用的是某银行的一个金融场景应用的模拟数据库，数据库中表，表结构以及所有字段的说明。

### 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

该关卡任务已完成，实施情况本报告略过。

### 2.3.3 既买了保险又买了基金的客户

本关任务： 请用一条语句查询既买了保险又买了基金的客户的名称和邮箱。通过 select 语句和嵌套查询、多条件查询。通过限制 pro_type 来找到对应的 c_id，进而获得对应的客户信息，最后按照 c_id 排序。代码如图 2.2.2 所示：

```
-- 3) 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
-- 请用一条SQL语句实现该查询:
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 2.3.1 既买了保险又买了基金的客户

### 2.3.4 办理了储蓄卡的客户信息

本关任务： 查询办理了储蓄卡的客户名称、手机号、银行卡号。

本关的关键在多表的连接，将 client 和 bank_card 表进行内连接，查询对应客户办理了储蓄卡的个人信息。代码如图 2.2.3 所示：

```
select
    c_name,
    c_phone,
    b_number
    from client INNER JOIN bank_card
    ON client.c_id = bank_card.b_c_id AND b_type = '储蓄卡'
    order by c_id;
```

图 2.3.2 办理了储蓄卡的客户信息

### 2.3.5 每份金额在 30000~50000 之间的理财产品

本关任务： 查询每份金额在 30000～50000 之间的理财产品，并对结果进行排序。

关键在于通过 between 语句进行条件限定，降序排列的效果。代码如图一 2.2.4 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 每份金额在 30000~50000 之间的理财产品

### 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。

通过 pro_icome 进行分组，用 count()函数来统计出现次数，需要找出众数则需要找到出现次数最多的一个，可以通过 ALL 关键词来与所有的数据进行比较，筛选出出现次数最多的一个。代码如图 2.2.5 所示：

```sql
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
    )
```

图 2.3.4 商品收益的众数

### 2.3.7 未购买任何理财产品的武汉居民

本关任务： 查询未购买任何理财产品的武汉居民的信息。

本关关键在于通过 like 函数用表达式匹配来查找符合条件的居民，并且通过 not in()来筛选。代码如图 2.3.5 所示：

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

图 2.3.5 未购买任何理财产品的武汉居民

### 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。

本关通过 sum 函数计算投资总收益，将 client 和 property 两表右连接。并且通过 c_id 和 pro_status 进行分类，筛选 pro_status 状态为‘可用’，最后通过 total_icome 进行排序，只取 top3。代码如图 2.3.6 所示：

```sql
select c_name, c_id_card, sum(pro_income) as total_income
    from client right JOIN property
        ON client.c_id=property.pro_c_id
    group by c_id, pro_status
    having property.pro_status='可用'
    order by total_income DESC limit 3;
```

图 2.3.6 投资总收益前三名的客户

### 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

本关任务： 查询客户理财、保险、基金投资金额的总和，并排序。

本关的关键在于通过 full join 和 union all 的多表连接，选出三类资产的购买客户，计算出资金总和作为依据从而排序。代码如图 2.3.7 所示：

```sql
select
    c_name, c_id_card, COALESCE(sum(pro_account), 0) AS total_amount
    from client
        full join (
            select
                pro_c_id,
                pro_quantity * p_amount as pro_account
            from property, finances_product where pro_pif_id = p_id and pro_type = 1
            union all
            select
                pro_c_id,
                pro_quantity * i_amount as pro_account
            from property, insurance where pro_pif_id = i_id and pro_type = 2
            union all
            select
                pro_c_id,
                pro_quantity * f_amount as pro_account
            from property, fund where pro_pif_id = f_id and pro_type = 3
        ) as pro on (pro.pro_c_id = c_id)
    group by c_id
    order by total_amount DESC;
```

图 2.3.7 客户理财、保险与基金投资总额

### 2.3.13 客户总资产

该关卡任务已完成，实施情况本报告略过。

### 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。

本关关键在于通过 distinct 消重，通过 limit off, count 来取特定排序的值。代码如图 2.3.8 所示：

```sql
select i_id, i_amount
    from insurance
    where i_amount = (
        select distinct i_amount
        from insurance order by i_amount DESC limit 3,1
    )
```

图 2.3.8 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。

本关通过两种排序方式分别进行排序，通过 rank() over()来统计名次不连续的排名，通过 dense_rank() over()统计名次连续的排名。代码如图 2.3.9 所示：

```sql
-- (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 total_revenue desc) as rank
    from property
    where pro_type = 3
    group by pro_c_id
    order by total_revenue desc;


-- (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 total_revenue desc) as rank
    from property
    where pro_type = 3
    group by pro_c_id
    order by total_revenue desc;
```

图 2.3.9 基金收益两种方式排名

### .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。

思路：查询 property 表中 pro_type 字段值为 3 的记录，并将它们按 pro_c_id 分组。对于每组记录，使用 string_agg 函数将 pro_pif_id 值组合成一个字符串。

然后，它连接两个子查询，并使用 WHERE 子句筛选出 pifset 和 pifset2 字段相同且 pro_c_id 字段小于 pro_c_id2 字段的记录。最后，使用 ORDER BY 子句将结果按 pro_c_id 字段排序。代码如图 2.3.10 所示：

```
select
    p1.pro_c_id as c_id1, pro_c_id2 as c_id2
    from(
        select
            pro_c_id, string_agg(pro_pif_id , ' ' order by pro_pif_id) as pifset
        from property
        where pro_type = 3
        group by pro_c_id
    ) as p1,
    (
        select
            pro_c_id, string_agg(pro_pif_id, ' ' order by pro_pif_id) as pifset
        from property where pro_type = 3 group by pro_c_id
    ) as p2(pro_c_id2, pifset2)
    where p1.pifset = p2.pifset2 and p1.pro_c_id < p2.pro_c_id2
    order by p1.pro_c_id;
```

图 2.3.10 持有完全相同基金组合的客户

### 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。

此关采用了暴力解法，将 2 月所有可能成为高峰期的日期列举出来，进行查找。需要限定资产为基金类型和至少三个连续交易日。代码过长，穷举思路简单。

### 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。

查询 property 表中 pro_type 是基金且 pro_purchase_time 字段在 2022 年 2 月份的记录，并将它们与 fund 表连接。然后，它使用 DATE_PART 函数将 pro_purchase_time 字段的星期数减去 5，并使用 extract 函数提取 pro_purchase_time 字段的星期几。查询使用 SUM 函数计算总额，使用 CASE 语句按星期几分类，最后使用 GROUP BY 和 ORDER BY 子句将结果按 wk 字段分组和排序。代码如图 2.3.11 所示：

```
select
    wk week_of_trading,
    sum(CASE WHEN dayId=1 THEN amount ELSE null END) as Monday,
    sum(CASE WHEN dayId=2 THEN amount ELSE null END) as Tuesday,
    sum(CASE WHEN dayId=3 THEN amount ELSE null END) as Wendnesday,
    sum(CASE WHEN dayId=4 THEN amount ELSE null END) as Thursday,
    sum(CASE WHEN dayId=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 dayId,
        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
) t
group by wk order by week_of_trading;
```

图 2.3.11 以日历表格式显示每日基金购买总金额

### 2.3.20 查询销售总额前三的理财产品

本关任务：查询销售总额前三的理财产品。

思路：使用 to_char 函数将 pro_purchase_time 字段转换为 yyyy 格式的字符串，并计算每个产品在每年的销售额。接着，使用 rank 函数为每年的每个产品分配排名，并使用 order by 子句将结果按 sumamount 字段降序排序。最后，使用 UNION 连接两个子查询，并使用 WHERE 子句筛选出每年排名前 3 的产品。代码如图 2.3.12 所示：

```
select pyear,rank as rk,p_id,sumamount
from(
  (
  select  pyear,rank,p_id,sumamount
  from(
    select pyear,rank()over(order by sumamount desc),p_id,sumamount
    from (
      select to_char(pro_purchase_time,' yyyy') as pyear,p_id,(p_amount*pro_quantity)as sumamount
      from finances_product,property
      where pro_type=1 and pro_pif_id=p_id and pyear=2010
    ) order by rank
  ) where rank<=3
)
union
(
  select  pyear,rank,p_id,sumamount
  from(
    select pyear,rank()over(order by sumamount desc),p_id,sumamount
    from (
      select to_char(pro_purchase_time,' yyyy') as pyear,p_id,(p_amount*pro_quantity)as sumamount
      from finances_product,property
      where pro_type=1 and pro_pif_id=p_id and pyear=2011
    ) order by rank
  )where rank<=3
))
```

图 2.3.12 查询销售总额前三的理财产品

### 2.3.21 投资积极且偏好理财类产品的客户

本关任务：投资积极且偏好理财类产品的客户。若该客户持有基金产品种类数小于其持有的理财产品种类数，则认为该客户为投资积极且偏好理财产品的客户。

查询 property 表中 pro_type 字段值为 1 或 3 的记录，并将它们与 finances_product 和 fund 表连接。然后，使用 COUNT(DISTINCT pro_pif_id) 函数统计每个客户购买的不同产品/基金的数量，并使用 GROUP BY 子句将结果按 pro_c_id 字段分组。接着，使用 HAVING 子句筛选出购买产品数量大于 3 的客户，用 INNER JOIN 连接两个子查询，并使用 WHERE 子句筛选出购买产品数量大于基金数量的客户。代码如图 2.3.13 所示：

```
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类产品的客户

### 2.3.22 查询购买了所有畅销理财产品的客户

本关任务：查询购买了所有畅销理财产品的客户。持有人数超过 2 的理财产品称为畅销理财产品。

子查询来找出所有类型为 1 的记录，并去重，再找出找出所有记录数大于 2 的 pro_pif_id，将所有类型为 1 的记录分组，并使用 HAVING 子句来筛选出所有记录数等于"sumid" 的 pro_c_id。代码如图 2.3.14 所示：

```sql
select pro_c_id
    from(select distinct pro_c_id,pro_pif_id from property where pro_type=1
        and pro_pif_id in (
            select distinct pro_pif_id--查找所有畅销理财产品的id
            from(
                select distinct pro_c_id,pro_pif_id from property where pro_type=1)
                group by pro_pif_id having count(*)>2
        ))a,
        (
            select count(*)as sumid
            from(
                select distinct pro_pif_id--查找所有畅销理财产品的id
            from(
                select distinct pro_c_id,pro_pif_id from property where pro_type=1)
                    group by pro_pif_id having count(*)>2
                )
        )b
    group by pro_c_id,b.sumid having count(*)=b.sumid;
```

图 2.3.14 查询购买了所有畅销理财产品的客户

### 2.3.23 查找相似的理财产品

本关任务：查找相似的理财产品。结果输出要求：按照相似度值降序排列，相同相似度的理财产品之间则按照产品编号的升序排列。

使用三层嵌套的子查询来实现。通过对 pro_type=1 的记录进行分组并计数，得到每个 pro_pif_id 的数量。然后，在外层查询中使用 dense_rank() 函数按照 pro_pif_id 的数量排序，并得到每个 pro_pif_id 的排名（rank）。最后，通过在外层查询中筛选出 rank<=3 的结果，取出前三名的 pro_pif_id 和对应的数量。

```sql
select pro_pif_id,cc,prank
from(
    select pro_pif_id,count(*) as cc,dense_rank() over(order by cc desc) as prank
    from(
        select * from property where pro_type=1
        and pro_pif_id in(
            select pro_pif_id
            from (select * from property where pro_type=1)
            where pro_pif_id <>14 and pro_c_id in(
                select pro_c_id --购买14号理财产品前三名的人的id
                from(
                    select pro_c_id,dense_rank() over(order by rk desc) as rank
                    --购买14号理财产品的人的id和排名
                    from(
                        select pro_c_id,count(*) as rk
                        from property where pro_pif_id=14 and pro_type=1
                        group by (pro_c_id))p1
                    )p2
                where rank<=3)
            )
        )group by (pro_pif_id)
    )where prank<=3;
```

图 2.3.15 查找相似的理财产品

### 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

该小节内容关于 Insert, Update, Delete 语句在不同场景下的应用。

### 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

该关卡任务已完成，实施情况本报告略过。

### 2.4.3 批量插入数据

本关任务：向客户表 client 批量插入数据。向 client 表中插入所有 new_client 表中的记录。代码如下：

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

### 2.4.4 删除没有银行卡的客户信息

本关任务：删除在本行没有银行卡的客户信息。

使用 NOT EXISTS 子句检查是否存在与 client 表 c_id 字段值匹配的 bank_card 表 b_c_id 字段值。如果不存在，则删除 client 表中的该记录。代码如下：

```
delete from client
    where not exists (
        select 1
        from bank_card
        where client.c_id = bank_card.b_c_id
    );
```

### 2.4.5 冻结客户资产

该关卡任务已完成，实施情况本报告略过。

### 2.4.6 连接更新

本关任务：根据客户表的内容修改资产表的内容。

用一条 update 语句，根据 client 表中提供的身份证号(c_id_card)，填写 property 表中对应的身份证号信息(pro_id_card)。通过 set 语句将 property

表中的 pro_id_card 字段更新为与 pro_c_id 字段匹配的 client 表中 c_id_card 字段的值。代码如下：

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

## 2.5 视图

此小节关于视图的创建与使用。

### 2.5.1 创建所有保险资产的详细记录视图

该关卡任务已完成，实施情况本报告略过。

### 2.5.2 基于视图的查询

本关任务：基于视图 v_insurance_detail 查询每位客户保险资产的总额和保险总收益。

从视图 "v_insurance_detail" 中查询数据，并按照客户的姓名和身份证号分组。对于每组数据，计算保险总金额 (由产品数量和产品单价组成) 和保险总收益，最后按照保险总金额降序排列。代码如图 2.5.1 所示：

```
SELECT
    c_name,
    c_id_card,
    sum(pro_quantity * i_amount) as insurance_total_amount,
    sum(pro_income) as insurance_total_revenue
FROM v_insurance_detail
GROUP BY c_name,
    c_id_card
ORDER BY insurance_total_amount DESC
```

图 2.5.1 基于视图的查询

## 2.6 存储过程与事务

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。

存储过程的目的是在 "fibonacci" 表中生成一个长度为 "m" 的斐波那契数列。首先将数列的第一和第二个数插入到 "fibonacci" 表中，使用一个 WHILE 循环，计算数列中的剩余数。每次循环，计算斐波那契数列中的下一个数，并插

入到 "fibonacci" 表中。代码如图 2.6.1 所示：

```
create procedure sp_fibonacci(in m int)
as
declare
 n int := 2;
 a int := 0;
 fibn int;
 b int := 1;
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 n<m loop
    fibn := a + b;
    insert into fibonacci values(n,fibn);
    a := b;
    b := fibn;
    n := n + 1;
    end loop;

end;
/
```

图 2.6.1 使用流程控制语句的存储过程

### 2.6.2 使用游标的存储过程

本关任务：使用游标编程存储过程为医院的某科室排夜班值班表。

使用两个游标来遍历员工表，第一个游标遍历所有类型为护士的员工，第二个游标遍历所有类型不为护士的员工。使用 WHILE 循环，遍历每一天。每次循环，把两个护士的名字和一个医生的名字插入到 "night_shift_schedule" 表中。通过 EXTRACT 函数和 CAST 函数来获取每一天是星期几（0 表示星期天，1 表示星期一，以此类推）。因为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。所以如果这一天是星期天，则使用之前保存的 "head" 变量；如果这一天不是星期天，则使用游标 cur2 遍历的医生。代码如图 2.6.2 所示：

```
create procedure sp_night_shift_arrange(in start_date date, in end_date
date)
AS
    declare tp int default false;
    declare wk int default false;
    declare doc char(30);
    declare nur1 char(30);
    declare nur2 char(30);
    declare head char(30);
    cursor cur1 for select e_name from employee where e_type = 3;
    cursor cur2 for select e_type, e_name from employee where e_type < 3;
begin

    open cur1;
    open cur2;
    while start_date <= end_date loop
        fetch cur1 into nur1;
        if not found then
            close cur1;
            open cur1;
            fetch cur1 into nur1;
        end if;
        fetch cur1 into nur2;
        if not found then
            close cur1;
            open cur1;
            fetch cur1 into nur2;
        end if;
         wk := (extract(DOW FROM cast(start_date as TIMESTAMP)) + 7) % 8;
        if wk = 0 and head is not null then
            doc := head;
            head := null;
        else
            fetch cur2 into tp, doc;
            if not found then
                close cur2;
                open cur2;
                fetch cur2 into tp, doc;
            end if;
            if wk > 4 and tp = 1 then
                head := doc;
                fetch cur2 into tp, doc;
                if not found then
                    close cur2;
                    open cur2;
                    fetch cur2 into tp, doc;
                end if;
            end if;
        end if;
        insert into night_shift_schedule values (start_date, doc, nur1,
nur2);
        SELECT start_date + INTERVAL '1 day' into start_date;
    end loop;
end;
```

图 2.6.2 使用游标的存储过程

### 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。

存储过程执行两条 update 语句，用于将转账金额从源账户和目标账户中分别扣除。然后进行判断，如果源账户余额不足或目标账户不存在，则回滚事务并将 return_code 赋值为 0；否则提交事务并将 return_code 赋值为 1。
这样的话，在调用这个存储过程时，可以通过 return_code 的值来判断转账是否成功。代码如图 2.6.3 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_number = source_card_id and b_c_id = applicant_id and b_type = '储蓄卡';
    update bank_card set b_balance = b_balance + amount
        where b_number = dest_card_id and b_c_id = receiver_id and b_type = '储蓄卡';
    update bank_card set b_balance = b_balance - amount
        where b_number = dest_card_id and b_c_id = receiver_id and b_type = '信用卡';

    if not exists(select * from bank_card
        where b_number = source_card_id and b_c_id = applicant_id and b_type = '储蓄卡' and b_balance >= 0) then
        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 2.6.3 使用事务的存储过程

## 2.7 触发器

触发器的创建和使用场景。

### 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： 为表 property(资产表)编写一个触发器，以实现以下完整性业务规则：

- 如果 pro_type = 1， 则 pro_pif_id 只能引用 finances_product 表的 p_id;
- 如果 pro_type = 2， 则 pro_pif_id 只能引用 insurance 表的 i_id;
- 如果 pro_type = 3， 则 pro_pif_id 只能引用 fund 表的 f_id;
- pro_type 不接受(1,2,3)以外的值。

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
    --此处用declare语句声明你所需要的变量
    declare tp int default new.pro_type;
    declare id int default new.pro_pif_id;
    declare msg varchar(50);
BEGIN
    --此处插入触发器业务
     if tp = 1 then
        if id not in (select p_id from finances_product) then
            msg := concat('finances product #', id, ' not found!');
        end if;
    elseif tp = 2 then
        if id not in (select i_id from insurance) then
            msg := concat('insurance #', id, ' not found!');
        end if;
    elseif tp = 3 then
        if id not in (select f_id from fund) then
            msg := concat('fund #', id, ' not found!');
        end if;
    else
        msg = concat('type ', tp, ' is illegal!');
    end if;
    if msg is not null then
        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;


-- 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

该小节需要创建符合要求的用户自定义函数和使用。

### 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。

用 create function 语句创建函数，利用创建的函数，仅用一条 SQL 语句查询存款总额在 100 万(含)以上的客户身份证号，姓名和存款总额(total_deposit)，结

果依存储总额从高到低排序。代码如图 2.8.1 所示：

```
CREATE OR REPLACE FUNCTION get_deposit(client_id integer)
 returns numeric(10,2)
 LANGUAGE plpgsql
AS
$$
begin
    return (
        select
            sum(b_balance)
        from bank_card
        where b_type = '储蓄卡'
        group by b_c_id
        having b_c_id = client_id
    );
end;
$$;
```

图 2.8.1 创建函数并在语句中使用它

## 2.9 安全性控制

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。

通过 create role 创建角色，grant 语句授予权限。代码如图 2.9.1 所示：

```
-- 请填写语句，完成以下功能：
--  (1) 创建角色client_manager和fund_manager；
create role client_manager WITH PASSWORD NULL;
create role fund_manager WITH PASSWORD NULL;
-- CREATE ROLE client_manager WITH LOGIN PASSWORD 'Hust_12345678';
-- CREATE ROLE fund_manager WITH LOGIN PASSWORD 'Hust_12345678';
--  (2) 授予client_manager对client表拥有select,insert,update的权限；
grant select, insert, update on client to client_manager;
--  (3) 授予client_manager对bank_card表拥有查询除银行卡余额外的select权限；
grant select (b_c_id, b_number, b_type) on bank_card to client_manager;
--  (4) 授予fund_manager对fund表的select,insert,update权限；
grant select, insert, update on 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 用户、角色与权限

## 2.10 并发控制与事务的隔离级别

有两个涉及该表的并发事务 t1 和 t2，分别定义在 t1.sql 和 t2.sql 代码文件中。平台会让两个事务并发执行，通过修改代码文件来达到题目预期的并发执行效果。

### 2.10.1 不可重复读

本关任务： 选择合适的事务隔离级别，构造两个事务并发执行时，发生“不可重复读”现象。

不可重复读指在一个事务中两次查询之中，数据不一致。选择将事务的隔离级别设置为 read uncommitted，根据任务要求的执行顺序利用 pg_sleep（）函数保证执行逻辑的准确，实现程序内部的等待。

### 2.10.2 幻读

本关任务：在 read committed 事务隔离级别，构造两个事务并发执行时，发生“幻读”现象。

两次查询余票超过 300 张的航班信息；在第 1 次查询之后，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,发生“幻读”。该任务复现了这一场景。

### 2.10.3 主动加锁保证可重复读

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

OpenGauss 的 select 语句支持 for share 和 for update 短语，分别表示对表加共享(Share)锁和写(write)锁，在事务结束时才释放。保证事务 t1 可重复读，在等待 t2 正常提交(commit)后，再查询一次全部航班的余票,MU2455 的余票减少 1 张。

### 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）

结合 java 和数据库语言实现一个简单的数据库应用系统。

### 2.11.1 JDBC 体系结构和简单的查询

本关任务：正确使用 JDBC 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。

调用 DriverManager 对象的 getConnection（）方法来建立实际的数据库连接。旦获得了连接，就可以与数据库进行交互。通过 statement 接口发送 SQL 命令并从数据库接收数据的方法和属性。通过 ResultSet executeQuery(String SQL)获取结果集，代码如图 2.11.1 所示：

```
import java.sql.*;

public class Client {

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

        try {
            // String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
            String URL = "jdbc:postgresql://localhost:5432/postgres";
            String USER = "gaussdb";
            String PASS = "Passwd123@123";
            Class.forName("org.postgresql.Driver");
            connection = DriverManager.getConnection(URL, USER, PASS);
            statement = connection.createStatement();
            resultSet = statement.executeQuery("select c_name, c_mail, c_phone from client where c_mail is not null");
            System.out.println("姓名\t邮箱\t\t\t\t电话");
            while (resultSet.next()) {
                System.out.print(resultSet.getString("c_name") + "\t");
                System.out.print(resultSet.getString("c_mail") + "\t\t");
                System.out.println(resultSet.getString("c_phone"));
            }

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

                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException throwables) {
                throwables.printStackTrace();
            }
        }
    }
}
```

图 2.11.1 基于视图的查询

### 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

该关卡任务已完成，实施情况本报告略过。

### 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

补充代码，实现一个银行卡转账的方法，方法返回 boolean 值，true 表示转帐成功，false 表示转账失败，并不需要细分或解释失败的原因。

理清转账失败可能原因：转出或转入帐号不存在，转出账号是信用卡，转出帐号余额不足。处理对应情况返回转账状态为 false，其余情况则返回 true。main()不需要修改。代码如图 2.11.2 所示（节选）：

```java
public class Transfer {
    static final String JDBC_DRIVER = "org.postgresql.Driver";
    static final String DB_URL = "jdbc:postgresql://127.0.0.1:5432/postgres?";
    static final String USER = "gaussdb";
    static final String PASS = "Passwd123@123";
    public static boolean transferBalance(Connection connection,
                                          String sourceCard,
                                          String destCard,
                                          double amount){
PreparedStatement pstmt = null;
        ResultSet resultSet = null;
        boolean n = true;
        try {
            connection.setAutoCommit(false);
            connection.setTransactionIsolation(4);
            String sql = "update bank_card set b_balance = b_balance - ? where b_number = ?;";
            pstmt = connection.prepareStatement(sql);
            pstmt.setDouble(1,amount);
            pstmt.setString(2,sourceCard);
            pstmt.executeUpdate();
            sql = "update bank_card set b_balance = b_balance + ? where b_number = ? and b_type = '储蓄卡';";
            pstmt = connection.prepareStatement(sql);
            pstmt.setDouble(1,amount);
            pstmt.setString(2,destCard);
            pstmt.executeUpdate();
            sql = "update bank_card set b_balance = b_balance - ? where b_number = ? and b_type = '信用卡';";
            pstmt = connection.prepareStatement(sql);
            pstmt.setDouble(1,amount);
            pstmt.setString(2,destCard);
            pstmt.executeUpdate();

            sql = "select * from bank_card where b_number = ? and b_type = '储蓄卡';";
            pstmt = connection.prepareStatement(sql);
            pstmt.setString(1,sourceCard);
            resultSet = pstmt.executeQuery();
            if(resultSet.next()==false) {
                n = false;
                connection.rollback();
            }
            else {
                if(resultSet.getDouble("b_balance") < 0){
                    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();
                }
            }

        } catch (SQLException throwables) {
            throwables.printStackTrace();
            n = false;
        } finally {
            try {
                if (pstmt != null) {
                    pstmt.close();
                }
                if (resultSet != null) {
                    resultSet.close();
                }
            } catch (SQLException throwables) {
                throwables.printStackTrace();
                n = false;
            }
        }
        return n;

    }
```

图 2.11.2 转账与事务操作

### 2.11.7 把稀疏表格转化为键值对存储

本关任务：将一个稀疏的表中有保存数据的列值，以键值对(列名，列值 ）的形式转存到另一个表中，这样可以直接丢失没有值列。

依规则将 entrance_exam 表的值转写到 sc 表。对每一行，从左至右依次考察每一列，转存非空列。转换为转存在大量的三元组中。代码如图 2.11.3 所示（节选）：

```
/**
 * 向sc表中插入数据
 *
 */
public static int insertSC(Connection conn,int sno, String type, int score){
    PreparedStatement pstmt = null;
    int n = 0;
    try {
        String sql = "insert into sc values (?,?,?);";
        pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1,sno);
        pstmt.setString(2,type);
        pstmt.setInt(3,score);
        n = pstmt.executeUpdate();
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    } finally {
        try {
            if (pstmt != null) {
                pstmt.close();
            }
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
    }
    return n;
}

public static void main(String[] args) {
    Connection conn = null;
    Statement stmt = null;
    ResultSet rest = null;

    try {
        Class.forName(JDBC_DRIVER);
        conn = DriverManager.getConnection(DB_URL, USER, PASS);

        stmt = conn.createStatement();
        String sql = "select * from entrance_exam;";
        rest = stmt.executeQuery(sql);
        while(rest.next()){
            int sno = rest.getInt("sno");
            int chinese = rest.getInt("chinese");
            int math = rest.getInt("math");
            int english = rest.getInt("English");
            int physics = rest.getInt("physics");
            int chemistry = rest.getInt("chemistry");
            int biology = rest.getInt("biology");
            int history = rest.getInt("history");
            int geography = rest.getInt("geography");
            int politics = rest.getInt("politics");
            if(chinese != 0){
                insertSC(conn,sno,"chinese",chinese);
            }
            if(math != 0){
                insertSC(conn,sno,"math",math);
            }
            if(english != 0){
                insertSC(conn,sno,"english",english);
            }
            if(physics != 0){
                insertSC(conn,sno,"physics",physics);
            }
            if(chemistry != 0){
                insertSC(conn,sno,"chemistry",chemistry);
            }
            if(biology != 0){
                insertSC(conn,sno,"biology",biology);
            }
            if(history != 0){
                insertSC(conn,sno,"history",history);
            }
            if(geography != 0){
                insertSC(conn,sno,"geography",geography);
            }
            if(politics != 0){
                insertSC(conn,sno,"politics",politics);
            }
        }
```

图 2.11.3 把稀疏表格转化为键值对存储

## 2.12 备份+日志：介质故障与数据库恢复

本节涉及数据库的备份与恢复，需要了解 OpenGauss 的恢复机制，提供的备份与恢复种类，gs_dump 和 ds_restore 的使用。

### 2.12.1 备份与恢复

本关任务：备份数据库，然后再恢复它。用 gs_dump 工具为该数据库做一次静态的(你一个人独享服务器)海量备份，备份文件为.tar 格式文件； 然后再用 gs_restore 工具，利用前述备份文件恢复数据库 residents。

结合 gs_dump 数据导出工具和 gs_restore 数据导入工具，代码如图 2.12.1 所示：

```PostgreSQL
# 你写的命令将在linux的命令行运行(test1_1.sh)
# 对数据库postgres作海量备份,备份至文件residents_bak.sql:

gs_dump -U gaussdb -W'Passwd123@123' -h localhost -p5432 residents -Ft -f
residents_bak.tar

# 你写的命令将在linux的命令行运行(test1_2.sh)
# 利用备份文件residents_bak.sql还原数据库:

gs_restore -U gaussdb -W'Passwd123@123' -h localhost -Ft -p5432 -d residents
residents_bak.tar
```

图 2.12.1 备份与恢复

## 2.13 数据库设计与实现

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。

### 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，并转换成关系式。

E-R图如图2.13.1所示：

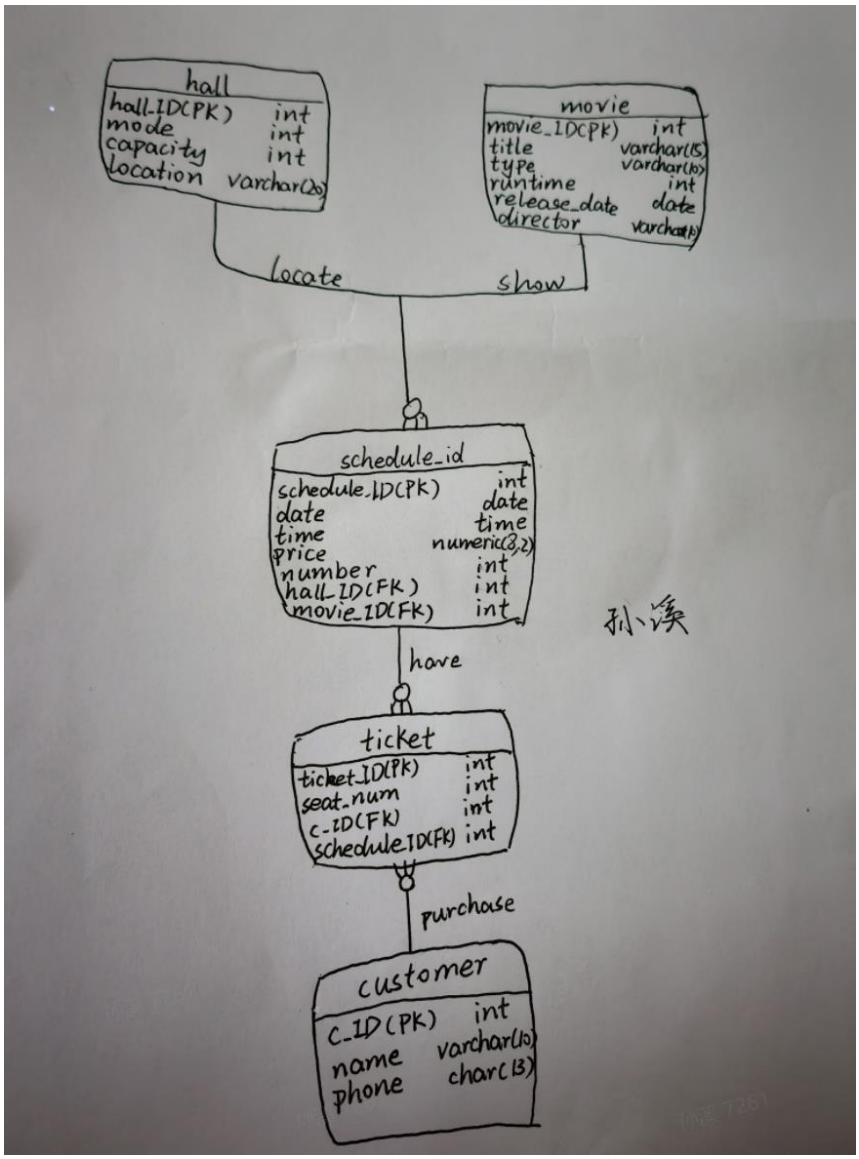


图 2.13.1 E-R 图

### 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

建模是指将实际问题抽象成数据库能够表示的形式。这个过程分为两个步骤：概念建模和逻辑建模。

在概念建模阶段，我们需要考虑的制约因素包括：业务要求、数据要求、安全性、可扩展性、性能。

在逻辑建模阶段，我们需要考虑的制约因素包括：数据类型（需要确定每个属性的数据类型，并确定是否需要限制属性值的范围）、主键（以便在数据库中

区分不同的记录）、外键（以便在数据库中维护数据之间的关系）、索引（需要考虑是否需要为表中的某些列建立索引，以提高查询效率）、数据冗余（需要考虑如何避免或最小化数据冗余，以保证数据的一致性和准确性）、数据分区（需要考虑如何将数据分区，以提高查询效率和系统的可扩展性）、数据约束（需要考虑如何通过设置数据约束来保证数据的一致性和准确性）。

在从实际问题到建立数据库模型的过程中，应该考虑若干制约因素，以机票订票系统为例，应考虑旅客的实际情况和系统的权限需求。在转账系统中需要考虑到实际场景下转账成功的必要条件。在把稀疏表格转化为键值对存储任务中，我们从数据结构的角度进行了存储结构优化，也进而提高了查询效率和空间利用率。

### 2.13.5 工程师责任及其分析

工程师应该能够根据工程相关背景知识进行合理分析，评估专业工程实践和复杂工程问题解决方案对社会、健康、安全、法律和文化的影响，并理解应承担的责任。工程师应尽可能考虑系统中的安全漏洞，并使用科学方法对复杂工程问题进行研究，包括设计实验、分析和解释数据，并通过信息综合获得合理有效的结论。

具体来说，工程师应该：

1. 能够基于工程相关背景知识进行合理分析，评价专业工程实践和复杂工程问题解决方案对社会、健康、安全、法律以及文化的影响，并理解应承担的责任。

2. 能够尽可能考虑系统中存在的安全漏洞，保证系统的安全性，以免造成人身伤害和财产损失。

3. 能够基于科学原理并采用科学方法对复杂工程问题进行研究，包括设计实验、分析与解释数据、并通过信息综合得出合理有效的结论。

4. 能够考虑工程系统的可扩展性，以便在未来满足新的需求。

5. 能够考虑工程系统的性能，以便满足用户的需求。

工程师应该遵守所在国家/地区的法律法规和行业准则，并承担相应的责任。同时，工程师还应该遵守职业道德，尊重他人的知识产权和财产权，并不得在未经授权的情况下使用他人的知识产权和财产。

在解决实际问题时，工程师应该努力了解问题的背景和环境，并根据相关条件进行分析和决策。在实际实施过程中，工程师应该负责协调各方面的工作，保证工程项目的顺利实施。

在服务的过程中，工程师应该尽可能提供优质的服务，并对自己的工作负责。如果发生任何问题，工程师应该尽快解决问题，并对自己的工作负责。

## 2.14 数据库的索引 B+树实现

进行 B+数基本数据结构的实现和简单操作。

### 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。

完成主要功能，判断页类型是否为叶子结点、根结点；设置/获取页的类型；设置/获取页的大小；增加页的大小；设置/获取页的最大大小；获取页的最小大小；设置/获取页的父页 id；设置/获取页的本身 id；设置/获取页的层级。主要用于在 B+树的各种操作中（插入、删除、查找等）维护页的相关信息，并能通过这些信息对 B+树进行调整以保证 B+树性质的维护。代码如图 2.14.1 所示：

```cpp
 * 函数功能：
 *  判断页类型是否为叶子结点
 * 建议：
 *  enum class IndexPageType { INVALID_INDEX_PAGE = 0, LEAF_PAGE, INTERNAL_PAGE }
 *  （定义于b_plus_tree_page.h头文件中）
 */
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
 * 建议：
 *  static constexpr int INVALID_PAGE_ID = -1;// invalid page id（定义与config.h中）
 *  父节点pageId为INVALID_PAGE_ID时即为根结点
 */
bool BPlusTreePage::IsRootPage() const {
  return parent_page_id_ == INVALID_PAGE_ID;
}
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个数)
 */
int BPlusTreePage::GetSize() const { return size_; }
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
 * 建议：
 *  1.如果此时为根节点：根节点可能是内部节点也可能是叶子节点
 *  内部节点：此时至少存在两个索引
 *  叶子节点：此时至少存在一条记录
 *  2.非根结点正常处理
 */
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; }
```

图 2.14.1 BPlusTreePage 的设计

### 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

该任务关卡跳过。

### 2.14.4 B+树索引：Insert

该任务关卡跳过。

### 2.14.5 B+树索引：Remove

该任务关卡跳过。

# 3 课程总结

在这次实验中完成了 14 个不同的实验，涵盖了多种主题，包括创建和修改表及其完整性约束，数据查询和操作，视图，存储过程和事务，触发器，用户定义函数，安全控制，并发和事务隔离，数据库应用开发，备份和日志数据库，数据库设计和实现，以及 B+ 树实现。课程收获和体会如下：

1. 使用 OpenGauss 数据库管理系统进行数据库管理。创建和修改表，以及如何在表中插入、修改和删除数据。

2. 学会了如何使用视图和存储过程来简化数据库操作，以及如何使用事务来管理数据库操作。

3. 学会了如何使用触发器和用户定义函数来扩展数据库功能。

4. 学会了如何在数据库应用开发中使用 JDBC 来连接数据库，以及如何保证代码的安全性。

5. 学会了如何管理数据库的备份和日志，以及如何设计和实现数据库。

6. 基于 B+树来实现数据库。

另外还学会了、如何使用 OpenGauss 复杂的语法来简化编程。还有在数据库应用开发中代码安全的重要性，以及 SQL 注入攻击可能造成的潜在后果。总的来说，这门课程可能会让你对数据库管理有更深入的了解，并且能够熟练使用 OpenGauss 数据库管理系统来实现各种数据库操作。