华中科技大学

数据库系统原理实践报告

专　　业：	计算机科学与技术专业
班　　级：	CS2009
学　　号：	U202015562
姓　　名：	徐雨梦
指导教师：	丁晓锋

分数	
教师签名	

2022 年 12 月 7 日

教师评分页

子目标	子目标评分
1	
2	
3	
4	
5	
6	

总分	

目　录

1 课程任务概述

“数据库系统原理实践”是配合“数据库系统原理”课程独立开设的实践课，注重理论与实践相结合，并且依托头歌实践教学平台为大家提供了一个方便的实验环境：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。

实训内容涉及以下几个部分：

1) 数据库、表、索引、视图、约束、存储过程、函数、触发器、游标等数据对象的管理与编程；
2) 数据查询，数据插入、删除与修改等数据处理相关任务；
3) 数据库的安全性控制，完整性控制，恢复机制，并发控制机制等系统内核的实验；
4) 数据库的设计与实现；
5) 数据库应用系统的开发(JAVA 篇)；
6) 数据库内核实验(B+树)。

针对这六部分内容，课程设置了 14 个实训，共 70 关：

1. 数据库、表与完整性约束的定义（create）
2. 表结构和完整性约束的修改（ALTER）
3. 数据查询（select）
4. 数据的插入、修改与删除（insert、update、delete）
5. 试图
6. 存储过程与事务
7. 触发器
8. 用户自定义函数
9. 安全性控制
10. 并发控制与事务的隔离级别
11. 数据库应用开发（Java 篇）
12. 备份+日志：介质故障与数据库恢复
13. 数据库的设计与实现
14 数据库的索引 B+数实现

2 任务实施过程与分析

2.1 数据库、表与完整性约束定义 Create

该实训需要①创建一个北京冬奥会信息系统的数据库；②在创建的数据库中创建一个表并为其指定主码；③创建两个新表并定义主键，再给其中一个表创建外键；④创建一个表并对其实现 check 约束；⑤创建一个表并对其实现 default 约束；⑥创建一个表并对其实现 unique 约束。

该实训需要注意的是，本关卡内测评环境与命令行为同一环境，这也意味着测试环境不会在每次提交时进行重置，因此每次 create 之前需要在命令行将原本创建的表删掉才行，该操作只针对第一个实训，其他实训的测评环境与命令行都是独立环境，不需要考虑上次的操作所带来的影响。

在该实训中，需要首先打开命令行手动连接到数据库，命令为：

gsql -h 127.0.0.1 -d postgres -U gaussdb -W'Passwd123@123'

2.1.1 创建数据库

该关卡任务已完成，但实施情况本报告略过。

2.1.2 创建表以主码约束

该关卡任务已完成，但实施情况本报告略过。

2.1.3 创建外码约束

该关卡任务已完成，但实施情况本报告略过。

2.1.4 创建 check 约束

该关卡任务已完成，但实施情况本报告略过。

2.1.5 创建 default 约束

该关卡任务已完成，但实施情况本报告略过。

2.1.6 创建 unique 约束

该关卡任务已完成，但实施情况本报告略过。

2.2 表结构与完整性约束的修改(ALTER)

该实训需要①修改表名；②添加与删除字段；③修改字段；④添加、删除与修改约束。

从实训内容来看，该实训的所有内容都需要对表进行修改，对表进行增删改操作都需要用到 alter 指令，指明需要修改的表并对其进行相关操作：

alter table **（table_name） add/drop/modify/rename to ***。

2.2.1 修改表名

该关卡任务已完成，但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，但实施情况本报告略过。

2.2.3 修改字段

该关卡任务已完成，但实施情况本报告略过。

2.2.4 添加、删除与修改约束

该关卡任务已完成，但实施情况本报告略过。

2.3 数据查询(Select)

该实训提供了六个表，分别为：client（客户表）、bank_card（银行卡）、finances_product（理财资产表）、insurance（保险表）、fund（基金表）、property（资产表），来对某银行的一个金融场景应用的数据库进行模拟。

在本实训中大多采用 select 语句来对表进行筛选来查询满足条件的表项数据，涉及了子查询、数据库函数、数据库特殊操作等知识点。

2.3.1 查询客户主要信息

该关卡任务已完成，但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，但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

1. 代码思路

首先分别找到买了保险的客户表和买了基金的客户表，再对两个表取交集。

2. 代码过程

①通过 with 操作生成两个子表 BAOXIAN、LICAI；

②通过 where exists 进行条件筛选，条件为既在子表 BAOXIAN 中存在又在子表 LICAI 中存在的客户。

3. 代码注释

```
WITH BAOXIAN AS (SELECT pro_c_id FROM property WHERE pro_type=2),--生成买了保险的客户表
LICAI AS (SELECT pro_c_id FROM property WHERE pro_type=3) --生成买了基金的客户表
SELECT c_name,c_mail,c_phone FROM client WHERE --从全部客户中筛选
EXISTS (SELECT pro_c_id FROM BAOXIAN WHERE pro_c_id=client.c_id)--买了保险，在保险这个表中存在
```

AND --同时满足

EXISTS (SELECT pro_c_id FROM LICAI WHERE pro_c_id=client.c_id);--买了基金，在基金这个表中存在

2.3.4 办理了储蓄卡的客户信息

该关卡任务已完成，但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

该关卡任务已完成，但实施情况本报告略过。

2.3.6 商品收益的众数

1. 代码思路：

找到每一个收益的数量，再比较得到众数。

2. 代码过程：

①通过 GROUP 对不同收益进行分组，并通过 COUNT 得到每个收益的数量，采用 with 生成一个子表保存这些信息；

②对该子表通过 max 找到数量最大值；

③通过 where 筛选等于该最大值的表项。

3. 代码注释：

WITH RESULT AS(SELECT pro_income,COUNT(pro_income) AS presence FROM property GROUP BY pro_income)--生成收益和它的数量，其中数量命名为 presence

SELECT pro_income,presence FROM RESULT WHERE presence=(SELECT MAX(presence)FROM RESULT);--采用聚合函数 max 找到最大的数，并筛选出众数

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，但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，但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，但实施情况本报告略过。

2.3.10 投资总收益前三名的客户

该关卡任务已完成，但实施情况本报告略过。

2.3.11 黄姓客户持卡数量

1. 代码思路：

对银行卡按照人进行分组并计数，得到每个人的持卡数量，再筛选姓名为“黄%”的用户。

2. 代码过程

①将客户信息表和银行卡信息表合并；

②对合并后的表分组得到每个客户的银行卡信息；

③采用 count 聚合函数对分组进行计数得到每个用户的持卡数量；

④通过 where 筛选黄姓用户。

3. 代码注释

```
select c_id,c_name, count(b_c_id)number_of_cards
from client left outer join bank_card on (b_c_id=c_id)--将银行的信息和客户的信息合并方便查询
group by c_id--按照用户分组
having c_name LIKE '黄%'--筛选黄姓用户
order by c_id,number_of_cards DESC;
```

2.3.12 客户理财、保险与基金投资总额

1.代码思路：

需要对基金表、理财产品表、保险表三个表进行合并来生成一个不区分任何资产类型的资产表，并对该资产表针对用户进行分组，计算其资产数。

2.代码过程

①通过 union 将三个表合并，并通过 with 生成一个新的子表以便后续使用；

②对新表分组并通过 sum 聚合函数计算总资产数并生成一个新的表；

③对计算出来的资产数与客户表合并得到客户的资产数并从中选择需要输出的信息；

④对没有资产的客户采用 not exists 筛选出来并将其资产数设置为 0；

⑤将③④筛选出来的表项通过 union 合并得到全部的信息表；

⑥对该表进行排序得到新的表。

2.3.13 客户总资产

1.代码思路：

将理财表、保险表、基金表、银行卡表按照其性质更新为真实拥有资产数量，更新后将这些资产按照用户分组求和得到最后的总资产。

2.代码过程：

①从 property 中生成一个不区分任何资产类型的资产表 ZICHAN；

②从 ZICHAN 中生成客户的投资总金额表，该思路与 2.3.12 中阐述的一致；

③从银行卡表中筛选用户的余额，其中储蓄卡按照正数保存，信用卡由于是透支的金额，因此将其更新为原金额的负数；

④对 ZICHAN 和更新后的银行卡表结合用户表进行资产相加得到总资产；

⑤通过 not exists 筛选出来资产数为 0 的用户将其资产数设置为 0 通过 union 联合④产生的表得到所有人的总资产数。

2.3.14 第 N 高问题

该关卡任务已完成，但实施情况本报告略过。

2.3.15 基金收益两种方式排名

该关卡任务已完成，但实施情况本报告略过。

2.3.16 持有完全相同基金组合的客户

1. 代码思路：

如果 A 用户和 B 用户的基金持有相同，那么对用户 A 来说，自己的基金在 B 的基金表中一定存在，也就是 A 的基金中不存在一条基金，它在 B 基金表中不存在；对于用户 B 来说，同样，其含有的基金中也不存在在 A 基金表中不存在的基金。因此可以通过两层 not exists 进行筛选。

2. 代码过程：

①筛选出两个客户表方便后续进行比较，命名为 T1、T2；

②对于这两个表，首先筛选 T1 的 id 比 T2 的 id 小的组合表项，该步骤用来去重；

③采用 and 筛选思路中阐述的条件：从 A 所拥有的基金表中筛选 B 中不存在的基金，再采用 not exists 判断；

④采用 and 对 B 同样采取的措施；

⑤对 T1、T2 客户表采取 where 来实施②③④的措施。

3. 代码注释：

```
WITH T1 AS(SELECT c_id c_id1 FROM client),T2 AS(SELECT c_id c_id2 FROM client)
SELECT c_id1,c_id2 FROM T1,T2 WHERE c_id1<c_id2
AND EXISTS(SELECT pro_c_id FROM property WHERE pro_c_id=c_id1 AND pro_type=3)
AND EXISTS(SELECT pro_c_id FROM property WHERE pro_c_id=c_id2 AND pro_type=3) --筛选买了基金的人
AND NOT EXISTS(
(SELECT FUND1 FROM (SELECT  pro_pif_id FUND1 FROM property WHERE pro_type=3 AND pro_c_id=c_id1)AS T3(FUND1) WHERE NOT EXISTS (SELECT pro_pif_id  FROM property WHERE pro_type=3 AND pro_c_id=c_id2 AND pro_pif_id=T3.FUND1))
)--t1 的用户基金 t2 中用户都包含
AND NOT EXISTS(
(SELECT FUND2 FROM (SELECT  pro_pif_id FUND2 FROM property WHERE pro_type=3 AND pro_c_id=c_id2)AS T4(FUND2) WHERE NOT EXISTS (SELECT pro_pif_id  FROM
```

property WHERE pro_type=3 AND pro_c_id=c_id1 AND pro_pif_id=T4.FUND2)));--t2 的用户基金 t1 中用户都包含，取交集就是相等

2.3.17 购买基金的高峰期

1. 代码思路：

首先先筛选出来二月的基金购买金额和日期，对每一条筛选出来的金额，对存在的三种情况进行判断：①该天的前两天；②该天的前后各一天；③该天的后两天，其中至少有一种情况满足要么是周六日要么超过 1000000。

2. 代码过程：

①采用 with 得到两个二月购买基金的时间与金额的表 T1、T2；

②对 T1 表筛选三种情况：筛选出该天的前两天是日期为周六日 or 基金金额超过 1000000 的表项构成一张表，同理筛选出其他两种情况的表，再采用 union 将所有的满足条件的表项集中在一张表中。

2.3.18 至少有一张信用卡余额超过 5000 元的客户信用卡总余额

该关卡任务已完成，但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

1. 代码思路：

首先筛选出二月每一天的购买时间和购买金额，之后调用相关函数判断出每一天的日期，即今日是周几，判断后形成一个冗余的日历表，表中的每一条数据都只包含一周中的某一天的金额，其他为空。之后再对冗余表去除冗余，将其中的 null 空填上数据去重。去除冗余的方法是筛选出每一个周一/二/三/四/五的表再对其合并。

2. 代码过程：

①筛选出二月的每一天的购买时间和每天的购买基金总金额；

②生成冗余表，通过 date_part 函数得到每一天的日期，并采用 case-when 语法为周一/二/三/四/五填入数额；该冗余表的表头信息为：周次、周一、周二、周三、周四、周五；

③针对冗余表筛选出每一个周*的表；

④通过 left outer join 以及将周次相等作为条件得到初步的日历表；

⑤通过 right outer join 以及将周次相等作为条件得到初步的日历表；

⑥将④⑤生成的表通过 union 合并得到该月的基金金额日历表。

3. 代码注释：

WITH GOUMAI AS(SELECT pro_purchase_time,SUM(pro_quantity*f_amount) total_amount FROM property,fund WHERE pro_type=3 AND pro_pif_id=f_id GROUP BY pro_purchase_time HAVING TO_CHAR(pro_purchase_time,'yyyy-mm-dd') LIKE '2022-02-%' ORDER BY

```
pro_purchase_time),--把购买时间和金额筛选出来
T1 AS(SELECT DATE_PART('WEEK',pro_purchase_time)-5 week_of_trading,
(CASE WHEN extract(dow from pro_purchase_time)=1 THEN total_amount END) AS Monday,
(CASE WHEN extract(dow from pro_purchase_time)=2 THEN total_amount END) AS Tuesday,
(CASE WHEN extract(dow from pro_purchase_time)=3 THEN total_amount END) AS Wendnesday,
(CASE WHEN extract(dow from pro_purchase_time)=4 THEN total_amount END) AS Thursday,
(CASE WHEN extract(dow from pro_purchase_time)=5 THEN total_amount END) AS Friday FROM GOUMAI),
--把日期集中在 T1 这个表中
MONDAY AS(SELECT week_of_trading,Monday FROM T1 WHERE Monday IS NOT NULL),
--找星期一的金额
TUESDAY AS(SELECT week_of_trading,Tuesday FROM T1 WHERE Tuesday IS NOT NULL),
--找星期二的金额
WEDNESDAY AS(SELECT week_of_trading,Wendnesday FROM T1 WHERE Wendnesday IS NOT NULL),
--找星期三的金额
THURSDAY AS(SELECT week_of_trading,Thursday FROM T1 WHERE Thursday IS NOT NULL),
--找星期四的金额
FRIDAY AS(SELECT week_of_trading,Friday FROM T1 WHERE Friday IS NOT NULL)
--找星期五的金额
SELECT MONDAY.week_of_trading,Monday,Tuesday,Wendnesday,Thursday,Friday FROM
MONDAY LEFT OUTER JOIN TUESDAY ON MONDAY. week_of_trading=TUESDAY.
week_of_trading LEFT OUTER JOIN WEDNESDAY ON MONDAY.week_of_trading=WED
NESDAY.week_of_trading LEFT OUTER JOIN THURSDAY ON MONDAY.week_of_tradin
g=THURSDAY.week_of_trading LEFT OUTER JOIN FRIDAY ON MONDAY.week_of_tradi
ng=FRIDAY.week_of_trading;
--周 1、2、3、4、5 合起来，这个样例中用 left outer join 即可，但普遍情况应该 left outer join
和 right outer join 筛选出来两个表然后 union 一下
```

2.3.20 查询销售总额前三的理财产品

该关卡任务已完成，但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

该关卡任务已完成，但实施情况本报告略过。

2.3.22 查询购买了所有畅销理财产品的客户

1. 代码思路：

如果一个人购买了所有的畅销理财产品，那么对于畅销产品来讲，不存在一个产品，该产品不在该客户的拥有的理财产品表中。

2. 代码过程：

①with 筛选出一个子表 CX，保存畅销产品的信息；

②with 筛选出一个子表 CXCLIENT，该表保存了购买了理财产品的客户 id；

③对 CXCLIENT 的客户进行筛选：采用 not exists 从 CX 表中查找是否存在一个产品，该产品在 property 表中不存在该客户购买了该产品，不存在仍采用 not exists 进行判断。

3. 代码注释：

```
WITH CX AS(SELECT DISTINCT pro_pif_id FROM property  WHERE pro_type=1 GROUP BY pro_pif_id HAVING COUNT(*)>2)--CX 保存了畅销产品的 id
CXCLINET AS(SELECT DISTINCT pro_c_id FROM property WHERE pro_type=1)--CXCLIENT 保存了购买了理财产品的 id
SELECT pro_c_id FROM CXCLINET WHERE NOT EXISTS(
    SELECT pro_pif_id FROM CX WHERE NOT EXISTS(
        SELECT pro_pif_id FROM property WHERE pro_type=1 AND property.pro_c_id=CXCLINET.pro_c_id AND property.pro_pif_id=CX.pro_pif_id
    )--查找该 pro_c_id 下没有购买的畅销理财产品的表
);--该表如果不存在那么就是购买了所有的畅销理财产品
```

2.3.23 查找相似理财产品

该关卡任务已完成，但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，但实施情况本报告略过。

2.3.25 查找相似的理财客户

1. 代码思路：

对两个资产表进行笛卡尔积并筛选出买了相同投资产品的数据，再按照两个客户的 id 进行分组得到买的相同的产品数，统计产品数再排序。

2. 代码过程：

①with 筛选出一个子表 T1：保存客户 id 和理财产品 id；

②将该子表与资产表通过理财产品 id 进行交叉得到一个新的子表 T2；

③在子表 T2 中针对两客户 id 进行分组，采用 count 得到数量，用 rank 得到排名。

2.4 数据的插入、修改与删除(Insert,Update,Delete)

该实训主要是对表进行修改，包括增删改等操作，直接对数据库中的表进行修改得到一个新的表。增删改操作通常采用 inset/delete/update 三条指令完成，包括对单条数据和多条数据的修改。

2.4.1 插入多条完整的客户信息

该关卡任务已完成，但实施情况本报告略过。

2.4.2 插入不完整客户信息

该关卡任务已完成，但实施情况本报告略过。

2.4.3 批量插入数据

该关卡任务已完成，但实施情况本报告略过。

2.4.4 删除没有银行卡的客户信息

该关卡任务已完成，但实施情况本报告略过。

2.4.5 冻结客户资产

该关卡任务已完成，但实施情况本报告略过。

2.4.6 连接更新

该关卡任务已完成，但实施情况本报告略过。

2.5 视图

View（视图）是一张假表，只不过是通过相关的名称存储在数据库中的一个 PostgreSQL 语句，实际上是一个以预定义的 PostgreSQL 查询形式存在的表的组合，因此主要执行的是查询功能，可以采用 create 创建，在 select 中使用试图时和使用一张表的用法相同。

2.5.1 创建所有保险资产的详细记录试图

该关卡任务已完成，但实施情况本报告略过。

2.5.2 基于视图的查询

该关卡任务已完成，但实施情况本报告略过。

2.6. 存储过程与事务

该实训通过创建存储过程来进行多次插入的存储，该方法可以类比于 C 语言的过程性代码，其将存储过程通过循环或者条件判断的形式实施在数据库中，从而不需要数据库工作人员对每条数据进行插入存储而是通过将存储的过程代码化将数据插入存入数据库。

2.6.1 使用流程控制语句的存储过程

1. 代码思路：

对于斐波那契数列：a2=a1+a0，需要三个变量，因此在实现的时候需要三个声明三个变量保存相关的值，而每计算一个值都需要存入数据库的表中，因此再循环过程中每次循环都要通过 insert 对表进行插入。这里的表就相当于在代码过程中的数组分配的空间。

2. 关键代码：

```
while i<m loop
    insert into fibonacci values(i,f0);
    i:=i+1;
    f0:=f1;
    f1:=f2;
    f2:=f0+f1;
end loop;
```

2.6.2 使用游标的存储过程

1. 代码过程：

①初始化一些信息：创建医生游标和护士游标，通过声明 cursor 创建；选出主任的名字，便于后续比较使用；初始化当前日期为排班的起始日期；

②如果当前日期没有达到结束日期，就进入循环执行③，否则结束过程；

③对于护士，日期无影响，因此每次循环都进行两个单位的游标移动，如果达到尽头就重启游标；

④对于医生，主任有影响，因此获取当前日期为周*，如果是周六或者周日，先找到一个医生，如果游标到达尽头，就重启游标；判断该医生是否是主任，如果是，就跳过将游标下移找到下一个医生并设置 flag 为 1 表示主任跳过周一值班，否则该日的值班医生即为该医生；

⑤如果当天是周一，判断 flag 是否为 1，如果是，医生游标不移动，将主任写入值班表并恢复 flag 为 0，否则游标下移一位找到值班医生；

⑥如果不是周一、六、日，正常的对医生进行游标下移得到当日的值班医生；

⑦对当前日期加一天回到②。

2.6.3 使用事务的存储过程

该关卡任务已完成，但实施情况本报告略过。

2.7. 触发器

触发器是与某个表绑定的命名存储对象，由一组语句组成，当这个表上发生

某个操作(insert,delete,update)时，触发器将会被触发执行，它一般用于实现业务完整性规则。触发器通过 create 创建并在语句中大多采用条件判断进行相关约束，如果不满足将会进入异常处理状态，通过 raise exception 实现。

2.7.1 为投资表 property 实现业务约束规则

该关卡任务已完成，但实施情况本报告略过。

2.8. 用户自定义函数

在数据库中可以写自己的函数实现并封装某些功能，类似于聚合函数、排序函数等等，函数采用 create function 定义，在使用的时候也可以直接调用使用实现功能得到想要的输出，在 select 使用函数的时候，直接通过函数名调用即可，并将适当的参数传递给函数得到结果插入到表中。

2.8.1 创建函数并在语句中使用它

该关卡任务已完成，但实施情况本报告略过

2.9. 安全性控制

安全性控制引入了角色和权限两个概念和机制，对于具有相同权限或者特点的数据，可以对其进行角色赋值来分组，在后续进行权限赋予或者修改的时候，直接对角色操作可以避免多次对不同的数据进行相同的操作，更加方便。权限操作可以直接对个人也可以对角色，采用 grant 赋予权限和 revoke 收回权限。角色采用 create 创建，和创建客户类似。

2.9.1 用户与权限

该关卡任务已完成，但实施情况本报告略过。

2.9.2 用户、角色与权限

该关卡任务已完成，但实施情况本报告略过。

2.10. 并发控制与事务的隔离级别

该实训提供了一个飞机票购买场景，将数据库并发操作不一致性实例化，数据库中提供了航班信息表，有两个涉及该表的并发事务 t1 和 t2，两个事务会并发执行，每一关需要实现相关代码解决某个数据不一致性的问题。

这些关卡需要设置合适的隔离级别达到目的，同时需要设置合适的等待时长从而使得事务可以在适当的时间执行。

2.10.1 不可重复读

1. 代码思路：

该关卡需要构造不可重复读现象，也就是另一个事务修改要在两次读票之间，

因此两事务执行起来的时间安排可以设置为：

表 2.1 事务的时间安排表

时刻	事务 1	事务 2
1	读票(0)	等待
2	等待	读票(0.5)
3	修改(1)	等待
4	等待	读票(1.5)
5	等待	修改(2)
6	读票(3)	等待

对于该现象，可以将隔离级别设置为 READ COMMITTED。

2.10.2 幻读

该关卡需要构造幻读现象，也就是另一个事务修改要在两次提交之间，因此两事务执行起来的时间安排可以设置为：

表 2.2 关卡 2 的事务时间安排表

time	thing1	thing2
1	read(0)	wait
2	wait	update(1)
3	read(1.5)	wait

对于该现象，需要在另一个事务提交的时候才可以共享数据读取信息，因此需要将隔离级别设置为 READ COMMITTED。

2.10.3 主动加锁保证可重复读

该关卡任务已完成，但实施情况本报告略过。

2.10.4 可串行化

该关卡任务已完成，但实施情况本报告略过。

2.11. 数据库应用开发（Java 篇）

该实训的数据库应用开发采用 java 作为开发工具，使用了 JDBC 包来将 java 与数据库进行衔接，可以使得 Java 应用程序与各种不同数据库之间进行对话。因此本实训的各个关卡就是通过调用 JDBC 提供的包来实现要求功能，具体的实现就是通过各个包的接口来调用数据库。

2.11.1 JDBC 体系结构和简单的查询

该关卡任务已完成，但实施情况本报告略过。

2.11.2 用户登录

1. 代码思路：

①初始化数据库链接、用户、密码，通过包连接到数据库；

②对查询语句 0 初始化并通过"?"保留一个参数的位置;

③通过 setString 得到完整的语句；

④执行语句返回结果；

⑤比较得到的结果，相同就是登录成功，否则登录失败。

2. 关键代码：

```
String sql="select c_password from client where c_mail=? ;";
statement=connection.prepareStatement(sql);
statement.setString(1,loginName);
resultSet = statement.executeQuery();
String sqlpassword=null;
while(resultSet.next()){
        sqlpassword=resultSet.getString("c_password");
}
if(sqlpassword!=null&&sqlpassword.contains(loginPass))
   System.out.println("登录成功。");
else
   System.out.println("用户名或密码错误！");
```

2.11.3 添加新客户

该关卡任务已完成，但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，但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，但实施情况本报告略过。

2.11.6 事务与转账操作

1. 代码思路：

①通过连接、声明、传参、运行 select 语句得到转款人的信息；

②判断转款人的账号的类型，如果是信用卡，返回 false；如果余额小于转款数量，返回 false；

③通过传参、运行 select 语句得到收款人的信息；

④如果查找失败，返回 false；

⑤对转款人的余额进行更新，通过 update 语句对转款人信息进行更新，将余额修改为减少了转款数量后的金额；

⑥判断收款人的银行卡信息，如果是信用卡，需要将金额减少，如果是储蓄卡，需要将余额增加。

2.11.7 把稀疏表格转为键值对存储

1. 代码思路：

main 函数:

①连接数据库，并通过 JDBC 包调用函数筛选出每个同学的学习成绩；

②对筛选出来的表进行游标遍历：

③对于每一条数据，将其与科目名称进行匹配，匹配后将得到一条数据；

④调用 insertSC 函数将该信息插入到表中。

insertSC 函数:

①初始化插入语句为 insert 语句，含有三个需要灵活变化的参数；

②根据传入的参数对语句中的参数赋值；

③运行该语句。

2.12. 备份+日志：介质故障与数据库恢复

该实训需要实现命令行语句，对数据库进行整体的操作。在数据库中，数据有可能发生损坏，因此需要备份并恢复，另外需要记录下数据库的操作，形成日志文件，有利于后续恢复数据库。OpenGauss 提供了 gs_dump 工具用于数据导出，gs_restore 工具用于数据导入。

2.12.1 备份与恢复

1. 代码实现：

```
--备份
gs_dump -U gaussdb -W'Passwd123@123' -h localhost -p5432 residents -F t -f residents_bak.tar
--恢复
gs_restore -U gaussdb -W'Passwd123@123'    residents_bak.tar -p5432 -d residents
```

2.13. 数据库的设计与实现

该实训需要实现一个完整的数据库，也就是将数据库中的表完备的创建出来，除了定义还有各种约束，即为数据库中添加表构建一个完整的数据库，其中提供了一个具体的完整的应用场景方便实现一个具体功能，这里提供的是航空系统。此外还需要在数据库和建模两个维度实现该数据库的创建。

2.13.1 从概念模型到 OpenGauss 实现

该关卡任务已完成，但实施情况本报告略过。

2.13.2 从需求分析到逻辑模型

在创建关系模式的时候，需要注明表头信息和主键信息。

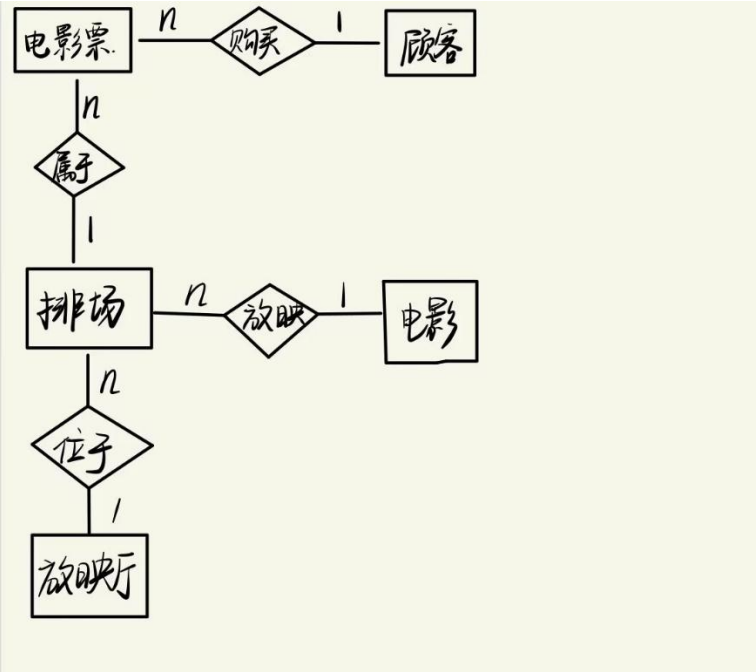

图 2.1 ER 图

2.13.3 建模工具的使用

该关卡任务已完成，但实施情况本报告略过。

2.13.4 制约因素分析与设计

在设计的过程中，需要考虑多重因素，其中最重要的就是实用性，即客户的需求是否能够得到满足，是否能够为人们解决某些问题，这才是该项技术得以发展的基础，然而在需求层面之外，还有其他的各种制约因素，包括但不限于是否满足道德层面的要求、社会需要的其他用处、相关法律政策的要求、对环境是否产生了影响等等。

在社会层面，设计的成果需要符合大势，也就是说，其设计一方面需要满足大众的需求，并且具有持久性和发展性，在需求发生变化的时候，该数据库可以不变或者只需要小范围变动即可以满足变化的需求。

在安全层面，需要注重数据安全，包括但不限于数据保密性、数据的备份等方面，尤其是数据保密性，在当前的网络时代，大数据时代，任何一个网民的几乎一切的数据都可以从网络中查询到，而数据库的运营人员应该保证用户数据的隐私，这一点可以有数据库设计人员在某些方面实现，设置好数据的权限等等。

在法律层面，设计出来的工具需要合法合规，不可以利用数据库来实现一些非法的事情，比如数据库设计涉及到非必要的用户隐私数据，导致用户的关键信息泄露。

在文化层面，该设计需要能够满足一些其他的需求，比如弘扬中华传统文化等等，这些东西在底层也可以潜移默化的影响到。

在环境层面，数据的物理实现需要大量的存储工具，这些工具的运行需要耗电等等，因此在设计层面就要考虑硬件成本以及所产生的环境成本等各种因素。

2.13.5 工程师责任及其分析

在实现过程中，社会、安全、法律以及文化等各种因素对任务的解决方案起到了制约和调整作用，而任务的解决方案对这些因素起到了服务作用，即技术是为许多上层需求服务的，需要在实现过程中考虑这些因素。同时解决方案也在推动着这些影响因素发生改变，技术的发展和革新将会将社会带入一个新的局面，给大家带来对各种层面新的思考，从而发展出当下的时代最合适的情形。

工程师在设计、实现、维护数据库的过程中，需要考虑上一节所阐述的各种制约因素，设计出一款在各个方面全部满足需要和要求的数据库。同时工程师需要牢记涉及的法律常识，谨慎行事，避免实现了一些技术被别人用于非法途径，从而造成巨大损失，特别是信息技术行业，所产生的损失将是巨大并有时候不可控的。同时，我们国家的工程设计师还要秉承从技术上发展中国以及提升国家综合实力的思想不断的进行创新设计。

2.14 数据库的索引 B+数实现

索引是数据库中的重要的组成部分，能够免于遍历数据的耗时，快速的定位记录。而 B+树作为数据库索引所用数据结构，能够使得数据库在较小的磁盘访问次数内获取记录，有着稳定且高效的速度。

该实训需要实现 B+树索引的几种类型。

2.14.1 BPlusTreePage 的设计

1. bool IsLeafPage() const;

函数功能：判断页类型是否为叶子结点。

函数实现：判断 page_type_ 是否等于 IndexPageType::LEAF_PAGE，是返回 true，否则，返回 false。

2. bool IsRootPage() const;

函数功能：判断页类型是否为根结点。

函数实现：判断 parent_page_id_是否等于 INVALID_PAGE_ID，是返回 true，否则，返回 false。

3. void SetPageType(IndexPageType page_type);

函数功能：设置索引页类型。

函数实现：直接将 page _type_=page_type。

4. int GetSize() const;

函数功能：获取结点中存放的元素（键值对）个数。

函数实现：返回 size_。

5. void IncreaseSize(int amount);

函数功能：增加结点中存放的元素（键值对）大小。

函数实现：对 size_增加 amount 的量。

6. int GetMaxSize() const;

函数功能：获取最大存放的元素（键值对）个数。

函数实现：返回 max_size_。

7. void SetMaxSize(int max_size);

函数功能：设置最大存放的元素（键值对）个数。

函数实现：max_size_设置为参数 max_size。

8. int GetMinSize() const;

函数功能：获取当前结点允许的最少元素个数

函数实现：判断当前是否为父节点，如果是，判断是否为叶子节点，如果是，返回 1，否则返回 2；如果不是，判断是否为叶子节点，如果是返回最大值的一半。

9. page_id_t GetParentPageId() const;

函数功能：获取父节点。

函数实现：返回 parent_page_id_。

10. void SetParentPageId(page_id_t parent_page_id);

函数功能：设置父节点。

函数实现：将 parent_page_id_=parent_page_id。

11. page_id_t GetPageId() const;

函数功能：获取节点号。

函数实现：返回 page_id_。

12. void SetPageId(page_id_t page_id);

函数功能：设置节点号。

函数实现：将 page_id_=page_id。

13. void SetLSN(lsn_t lsn = INVALID_LSN);

函数功能：设置 lsn_的值。

函数实现：将 lsn_ = lsn。

14. void SetSize(int size);

函数功能：设置存放的元素（键值对）个数。

函数实现：size_设置为参数 size。

2.14.2 BPlusTreeInternalPage 的设计

1. Init

函数功能：初始化 this 页。

函数实现：调用父类的函数设置，分别为：SetPageType、SetSize、SetPageId、SetParentPageId、SetMaxSize。

2. KeyAt

函数功能：返回 index 处的 key 值

函数实现：返回保存数据的数组中的索引对应的值 array_[index].first。

3. SetKeyAt

函数功能：设置 index 处的 key 值。

函数实现：设置键值对 array_[index].first=key。

4. ValueIndex

函数功能：寻找当前结点所有的键值（key）对中值为 value 的元素的索引。

函数实现：对保存键值对的数据进行遍历，查看 array_[i].second 是否等于 value，如果等于就是该键值对，返回该索引。

5. ValueAt

函数功能：返回索引 index 处的 value。

函数实现：直接返回 array_[index].second。

6. Lookup

函数功能：在 key 值有序排列的数组中，找到特定 key 值对应的 value。

函数实现：由于键值对的键是顺序的，因此可以采用二分法找到该键的位置，采用二分法找到该键的索引位置，返回该索引位置的值。

7. PopulateNewRoot

函数功能：对新的根结点进行初始化填充

函数实现：对第一个键值对的值初始化为 old_value;再插入一个键值对，即对数组的第二个键值对进行赋值。

8. InsertNodeAfter

函数功能：在当前 page 中找到 old_value 的位置，然后将 new_key 和 new_value 插入其中。

函数实现：通过 valueindex 函数找到该值得索引位置，从该索引位置遍历到最后，将键值对向后移一位，将新的键值对赋值给索引位置。

9. Remove

函数功能：移除 this 结点中的 index 处的键值对。

函数实现：根绝索引位置，从该处的下一个到最后，将所有键值对前移一位。

10. RemoveAndReturnOnlyChild

函数功能：返回根结点含有的唯一元素 ValueAt(0)。

函数实现：将该节点的 size 减少一个删除，直接返回 ValueAt(0)。

11. MoveAllTo

函数功能：合并 this 结点的元素至 recipient 结点中，即将 this 结点中的全部元素移至 recipient 结点的尾部。

函数实现：获取 recipient 的 size，如果合并后大于 max_size，将进行异常处理。找到当前节点的父节点，在父节点中找到 this 的 id 对应的索引，将 this 节点的第一个键值对的值设置为该索引。遍历 this 节点的数组，将该数组加入到 recipient 的数组之后，遍历的过程中，获取值的界面，将其父节点设置为 recipient 节点。遍历完毕更新 size 值，将 recipient 加上 size，将 this 节点的 size 设置为 0。

12. MoveHalfTo

函数功能：移动 this 结点中一半（向上取整）的元素到一个新生成的 recipient 结点中。

函数实现：调用 recipient 的 CopyNFrom 将后一半的数据赋值给 recipient 数组中，更新 this 的键值对数量为一半。

13. MoveFirstToEndOf

函数功能：移动 this 结点的首元素至 recipient 结点的末尾。

函数实现：

①获取父节点的 id 和界面信息；

②获取首节点的键值对；

③调用 CopyLastFrom 将 recipient 更新；

④对 this 节点采用 memmove 更新键值对数组；

⑥将父节点的 this 结点索引更新，更新为新的首结点的 key 值；

⑦更新 this 的 size。

14. MoveLastToFrontOf

函数功能：移动 this 结点的尾部元素至 recipient 结点的头部。

函数实现：找到结尾的键值对，将 this 结点的 size 减一。recipient 调用 CopyFirstFrom 函数实现节点更新。

15. CopyNFrom

函数功能：从 item 处开始，截取 size 个键值对并入到结点的尾部。

函数实现：对要添加的数据遍历，将其加入到 this 的数组末尾，遍历的过程中，通过 value 值获取到子节点的 id，索引到子界面，将子界面的父节点设置为 this 的 id 值。

16. CopyLastFrom

函数功能：将 pair 元素添加到 this 结点的尾部。

函数实现：将数组的末尾加一个键值对，更新该结点的 size 为增一；根据键值对的值索引子界面，将子界面的父节点更新为当前界面，并设置该界面为 true。

17. CopyFirstFrom

函数功能：移动 pair 元素至 recipient 结点的首部。

函数实现：

①获取父节点的 id 和界面信息；

②对 this 节点采用 memmove 更新键值对数组；

③对首元素进行更新，更新为传参的键值对；

④更新 this 的 size；

⑥将父节点的 this 结点索引更新，更新为新的首结点的 key 值；

⑦根据该键值对的 value 值找到子界面，将子界面的父节点设置为当前结点。

2.14.3 BPlusTreeLeafPage 的设计

该关卡任务已完成，但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，但实施情况本报告略过。

2.14.5 B+树索引：remove

1. bool IsEmpty() const;

函数功能：判断 B+树是否为空。

函数实现：如果根节点的值等于根节点，返回 true，否则，false。

2. void Remove(const KeyType &key, Transaction *transaction = nullptr);

函数功能：在 B+树中删除 key 对应的记录.

函数实现：

①判断 B+树是否为空，如果为空，直接返回；

②调用 FindLeafPage 查找该值对应的界面，如果不存在这样的界面，直接返回；

③获取该界面的信息，对该界面调用 RemoveAndDeleteRecord 函数删除 key 对应的记录；

④如果删除后的 size 小于最小 size，调用 CoalesceOrRedistribute 处理；

⑤将界面 pin 为 true。

3. bool CoalesceOrRedistribute(N *node, Transaction *transaction = nullptr);

函数功能：对元素个数小于 Min_size 的结点进行处理。

函数实现：

①判断 node 是否为根节点，如果是，调用 AdjustRoot()进行调整并返回；

②找到兄弟结点，并获取其界面信息；

③判断两节点的 size 和是否超过最大 size，如果没超过，进行合并，判断当前的 node 是否处于首结点，如果处于首节点，将兄弟结点合并到 node 上，否则将 node 合并到兄弟结点上；

④如果超过，无法合并，采用 Redistribute 解决。

4. bool FindSibling(N *node, N *&sibling);

函数功能：寻找 node 结点的兄弟结点 sibling。

函数实现：

①找到父节点的界面及信息；

②找到 node 的索引；

③如果 node 是首节点，设置兄弟结点索引为后一个，否则为前一个；

④根据索引找到兄弟结点的 id 再查询相关界面；

⑤根据 node 结点的索引值返回相应值，index 为 0 返回 true，否则返回 false。

5. bool Coalesce(N **neighbor_node, N **node, BPlusTreeInternalPage<KeyType, page_id_t, KeyComparator> **parent,int index, Transaction *transaction = nullptr);

函数功能：将 node 中的元素全部合并到 neighbor_node 中。

函数实现：

①如果两个结点的 size 之和大于最大 size 树，调用异常处理；

②调用 node 的 MoveAllTo 将元素合并；

③将父节点的 node 的索引键值对删掉；

④将 node 界面删除；

⑤判断父节点的 size 值是否小于最小 size 值，如果是，将父节点传参并调用 CoalesceOrRedistribute 进行处理。

6. void Redistribute(N *neighbor_node, N *node, int index);

函数功能：将 node 中的元素移动到 neighbor_node 中。

函数实现：

①若 index 为 0，调用 MoveFirstToEndOf 将 neighbor_node 首元素移动到 node 尾部；否则调用 MoveLastToFrontOf 将 neighbor_node 的尾元素移动到 node 首部；

②调用 unpinPage 将两节点设置为 true。

7. bool AdjustRoot(BPlusTreePage *node);

函数功能：对元素个数小于 Min_size 的根结点进行更新。

函数实现：

①如果 root 中 size 为 0，且树中只有 root，则直接删除结点，更新相关数据，返回 true；

②如果该根节点 size 为 1，将直接替换根节点为子节点；

③返回 false。

8. void UpdateRootPageId(int insert_record = 0);

函数功能：当 B+树根结点发生变化时，调用该函数在 header_page 中对根结点 ID 进行更新。

函数实现：获取后页面的信息，如果该头页面中没有根节点信息，调用 InsertRecord 插入当前的根节点；否则调用 UpdateRecord 更新根节点；同时需要注意 unpinPage，避免缓冲区内存泄露。

3 课程总结

在这次的数据库实验中，我完成了课程提供的 70 关，拿到了 157 分。学习并巩固了实训提供的内容：包括但不限于数据查询，数据插入、删除与修改等数据处理以及数据库的应用开发、内核实验等等。

整个实验的难度是不小的，特别是数据查询和 B+树索引的实现这两部分。

数据查询这一部分涉及的知识点比较多而杂，比如聚合函数的使用、日期函数的使用、多层嵌套的使用以及子表查询等，需要比较灵活的思路和对数据库语法以及函数的熟练。同时设置的问题许多需要多种方法相结合，但由于只能使用一条语句，导致写代码的时候容易将自己绕进去，这个时候就需要通过流程图等方式辅助自己完成查询，通过流程图，从内到外将一层层嵌套实现，这样不仅正确率高，而且花费的时间也会短不少。4

B+树这一实训比较难，首先是 C++对功能的实现，其次是对其他类的使用，这里需要学习 B+树的知识并将其与数据库结合，进而利用类的知识。这里需要熟练掌握 C++并且理清功能实现的步骤以及在实现这些功能的时候有没有什么其他细枝末节需要考虑，不然就会出现错误从而需要 debug 很久很久。在一开始我写 B+树的时候也是比较懵的，但是和同学讨论了以后收获了很多并且效率页提高了很多，起到了事半功倍的效果。

对于其他几个实训，虽然不及这两个实训难以及花费的时间多，但是也考察了数据库的其他方面的实操知识，也需要熟悉数据库的语法。

在写代码的过程中，最印象深刻的过程就是 debug 的过程了，尤其是数据查询这一章节，往往很快就可以把实现的思路理清楚并且写出来一版代码，但是总是会出现各种各样的错误，这个时候就需要耐下心来对代码进行调试，这次的平台是 educoder，因此我的调试方式就是先试试某个思路和某一步下的语句执行出来的结果，对该步骤下的结果进行验证，如果正确了就继续流程图中的下一步。会出错的一个重要原因就是对数据库的知识不够熟悉，尤其是函数的使用和某些复杂语法的使用，这个需要上网查询学习再去动手写代码。

这个实验在整体上来说还算比较顺利。我也学到了很多知识，尤其是将理论学习通过实验验证了一下，以及区分 MySQL 和 open gauss 的区别。同时我的顺利也离不开老师和同学的帮助，在这里我也非常感谢向我提供帮助的各位老师和同学！