
2023 CCF CHINASOFT
中国软件大会
智能化软件创新推动数字经济与社会发展
NASAC
第22届全国软件与应用学术会议
The 22nd National Software and Application Conference
FMAC
第8届全国形式化方法与应用会议
The 8th National Conference on Formal Method and Application
中国·上海 2023年12月1-3日
SHANGHAI·CHINA December 1-3, 2023

2023 CCF CHINASOFT 中国软件大会

基于openGauss开源数据库的系统软件实践教学探索

王 鑫
天津大学

2023.12.01

目录

CONTENTS

目录 CONTENTS

系统软件的国产化攻坚战

习近平总书记在中共中央政治局第三次集体学习时强调
(2023.2.21)

- **加强基础研究，是实现高水平科技自立自强的迫切要求**
- **要打好科技仪器设备、操作系统和基础软件国产化攻坚战**

系统软件人才培养现状问题

- 系统软件研发人才需要牢固坚实的计算机与软件工程理论文化素养
- 系统软件研发人才需要非常强的系统能力和编码开发能力
- 目前计算机、软件工程等专业培养方案与企业、开源社区需求脱节
- 目前培养系统软件研发人才的教学环境滞后（教材、实验）

系统软件实验教学的困难

◆ 系统软件：控制和协调计算机及外部设备,支持应用软件开发和运行的系统

- 是无需用户干预的各种程序的集合，主要功能是调度，监控和维护计算机系统
- 负责管理计算机系统中各种独立的硬件，使得它们可以协调工作

■ 主要类别

- **操作系统**：计算机系统的控制和管理中心
- **语言处理程序**：编译器、汇编器、链接器
- **数据库管理**：有组织地、动态地存贮大量数据

◆ 系统软件特点：代码规模大、系统复杂性高、教学难度大

系统软件实验教学的困难

◆教师 与 学生

- 教师
  - 仅就系统软件理论进行讲解
  - 缺少系统软件项目开发经验
- 学生
  - 仅停留在书本理论学习
  - 编程即ACM/程序设计比赛
  - 没有接触系统软件源码的机会
  - 缺少系统软件开发经验积累

系统软件实验教学的有效途径：开源教育

◆培养系统软件研发人才：需要以严肃的大型开源系统软件源代码为培养环境

1. **验证型**：复现和验证系统软件涉及的理论知识
2. **探索型**：通过设置断点、单步调试、修改源码等方式探索系统软件的运行机理
3. **创造型**：有独立的想法，编写源代码，为开源系统软件增添新功能

◆开源教育之于系统软件实验教学：必要性、不可代替性、紧迫性

数据库实验教学中使用开源软件：早期探索

◆数据库管理系统 DBMS

- 数据库理论方法需要在数据库管理系统（DBMS）中加以验证，才能使学生加深对数据库原理的理解
  - 在国内，计算机及相关本科专业数据库原理课程的实验环节一般使用某个商业数据库产品（如 Oracle、SQL Server、DB2 等）作为实验教学的软件环境
    1. 学生易被商业数据库产品提供的繁杂功能分散注意力，花费大量时间进行用户界面学习和文档查阅，而非数据库原理本身的实验验证
    2. 学生无法从内部架构和源代码层面理解数据库理论方法背后的实现机制，不利于与本科生高年级或研究生的数据库实现技术课程进行衔接

第 11 期
54 2015 年 6 月 10 日
计 算 机 教 育
Computer Education

文章编号：1672-5913(2015)11-0054-04 中图分类号：G642

基于开源软件的数据库原理课程实验教学改革探索

王 鑫，刘宝林，张 钢，戴维迪
（天津大学 计算机科学与技术学院，天津 300072）

摘 要：针对数据库原理课程实验教学中使用商业数据库产品作为实验环境带来的问题，分析数据库原理课程的传统实验教学模式，构建基于开源软件的数据库原理实验教学环境，介绍开源软件的选取原则，阐述基于 C/S 架构的单机实验环境和基于 B/S 架构的 Web 实验环境的搭建方法，并给出基于开源软件的实验教学方案。

关键词：数据库原理；实验教学；开源软件；实验环境

0 引 言

数据库原理是计算机及相关专业的核心基础课程。随着大数据时代的到来，数据库原理课程对于学生掌握数据理论和技术的重要作用更加凸显。同时，数据库是一个理论和实践并重的知识领域。数据库理论方法需要在数据库管理系统（DBMS）中加以验证，才能使学生加深对数据库原理的理解。在国内，计算机及相关本科专业数据库原理课程的实验环节一般使用某个商业数据库产品（如 Oracle、SQL Server、DB2 等）作为实验教学的软件环境。此做法主要存在两方面的问题：①学生易被商业数据库产品提供的繁杂功能分散注意力，花费大量时间进行用户界面学习和文档查阅，而非数据库原理本身的实验验证；时，掌握一个商业数据库产品的操作方法。形成这种认识的原因是学生在学习完数据库原理课程之后，应能胜任某商业数据库产品的数据库管理员（DBA）职位。然而实际情况是，以 Oracle、SQL Server 和 DB2 这三大数据库为代表的商业产品，为适应市场需求，功能和界面频繁更新且呈现复杂化趋势，难以通过相对简单的操作验证数据库原理知识点，相反会增加学生的学习负担；即使能够展示某些效果，由于产品的商业封闭性，也无法从内部结构和源代码级别说明实现机理，例如关系表的存储、索引的建立、SQL 查询的执行等。不使用商业数据库的另一个原因是，作为一门本科课程应保持理论和实践教学的中立性，没有理由选择一种产品作为实验环境而不选择另一种。至于培养 DBA，实际上并不应

王鑫, 刘宝林, 张钢等. 基于开源软件的数据库原理课程实验教学改革探索. 计算机教育, 2015年11期, 54-57+70页, 2015.

数据库实验教学中使用开源软件：早期探索

◆数据库原理实验教学环境

- MariaDB
- HeidiSQL
- phpMyAdmin

据库后台[8]。不过，我们没有直接选用 MySQL，而是选用了目前与 MySQL 完全兼容的开源数据库 MariaDB，原因有两方面：一是 MariaDB 在多项性能上超越了 MySQL；二是 MySQL 被 Oracle 公司收购后未来有被闭源的风险，而 MariaDB 是由 MySQL 原班人马开发的，保证其开源性。

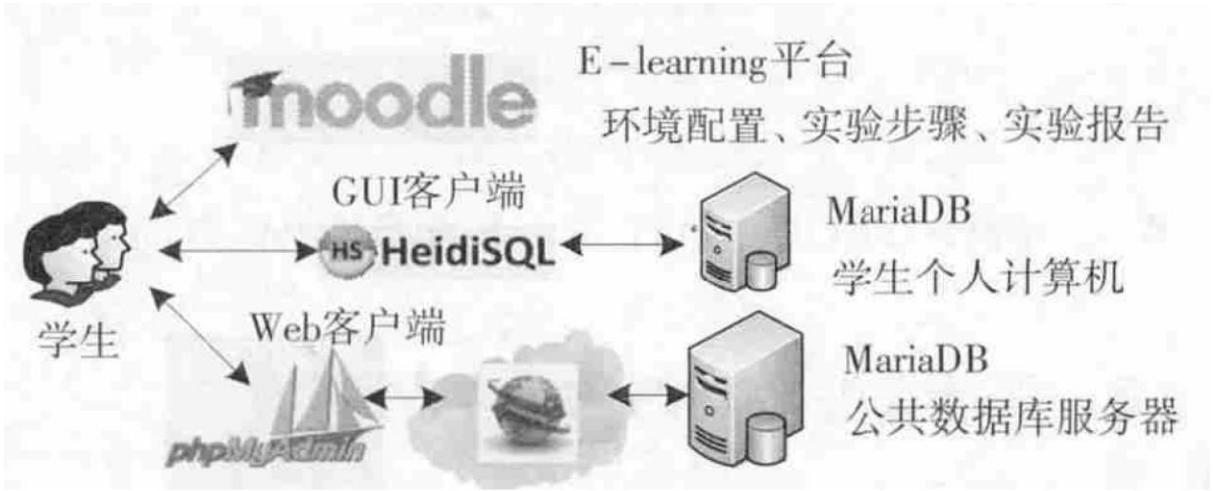


图 1　基于开源软件的数据库原理实验教学环境的架构图

王鑫, 刘宝林, 张钢等. 基于开源软件的数据库原理课程实验教学改革探索. 计算机教育, 2015年11期, 54-57+70页, 2015.

目录 CONTENTS

早期的数据库原理实验教学

◆ 数据库原理实验教学：验证性实验

表 1　数据库原理课程验证性实验

实验序号	实验名称	课上学时	课下估计学时	理论知识点
实验 1	数据库和表	3	2	关系数据库模型
实验 2	SQL 基本功能	3	3	简单查询、多关系查询
实验 3	SQL 高级功能	3	4	子查询、全关系操作、更新、事务
实验 4	数据库设计	3	4	E/R 模型、设计原则、E/R 图到关系
实验 5	完整性、视图和索引	3	4	约束、触发器、视图、索引
实验 6	存储过程和安全性	3	4	存储过程、SQL 中的安全机制
实验 7	JDBC 编程	3	5	连接数据库、执行查询和更新
实验 8	恢复和并发控制	3	4	日志和数据库恢复、封锁和并发控制

◆数据库原理实验教学：开源系统软件

数据库原理实验教学：基于openGauss

◆ 数据库原理实验教学：开源系统软件

- 天津大学是首批特色化示范性软件学院，肩负信创软件人才培养使命。
  数据库系统是重要的系统软件，是建设融入国产软件系统的软件工程专业系列教材的重点

- 天津大学智能与计算学部大类平台基础课程改革：《数据库原理》（48学时 2学分 教材 斯坦福 全书）
  > 4个专业：软件工程、计算机科学与技术、人工智能、网络空间安全

- **痛点：目前《数据库原理》课程实验教学主要侧重于数据库使用与SQL层面，原理与实现脱节很少从系统软件开发实现角度，透过数据库管理系统内部代码实现层面，巩固数据库原理学习**

- 亟需基于华为鲲鹏生态openGauss国产开源数据库软件，进行《数据库原理》课程综合改革，建设原理与实验系列教材，以实验教学教材建设为突破点

基于openGauss的数据库原理实验教学

◆ 建设思路

1. 基于openGauss数据库培养学生系统能力

现有的数据库实验教材多侧重于数据库应用方面，没有将数据库作为系统软件，对学生的系统能力进行培养，而本项目就是将数据库管理系统作为系统软件，让学生学习于动手改写数据库底层源代码，基于openGauss数据库，实现数据库原理理论教学落地

2. 基于openGauss数据库建设交互式实验教学教程

目前已有数据库实验教程多为命令式而非交互式，即按照教程相应步骤顺序操作便可获得相应结果，但此方式的缺点在于学生在使用的过程中没有真正理解步骤的含义与原理，容易使学生陷入到机械的复制粘贴操作。而本教程则通过在实验步骤中设置思考性问题，以启发性的方式帮助学生在实验过程中思考与应用所学知识，并且本教程通过“试错” -“修正” -“反馈”的方式充分细化实验步骤，迭代式提高学生系统软件开发能力

3. 基于openGauss数据库开发体验式实验教学平台

基于openGauss 3.0版本的源代码，即包括存储，查询，索引，日志，事务和并发控制等主要环节在内的不同数据库实现层面代码，构建体验式实验教学平台。本平台通过结合华为云资源，使学生利用VSCode进行远程开发openGauss源代码，并对改写后的代码进行编译，让学生体验实际生产中的开发流程，使学生身临其境的感受系统软件的开发

◆建设内容

- **《openGauss数据库实验教程》**
  - > 第1章 实验1：openGauss初探
  - > 第2章 实验2：openGauss编译与安装
  - > 第3章 实验3：openGauss调试
  - > 第4章 实验4：表的创建与系统表
  - > 第5章 实验5：表的页面存储结构
  - > 第6章 实验6：查询执行：嵌套循环连接
  - > 第7章 实验7：索引的作用与代价
  - > 第8章 实验8：日志与恢复
  - > 第9章 实验9：并发控制与锁

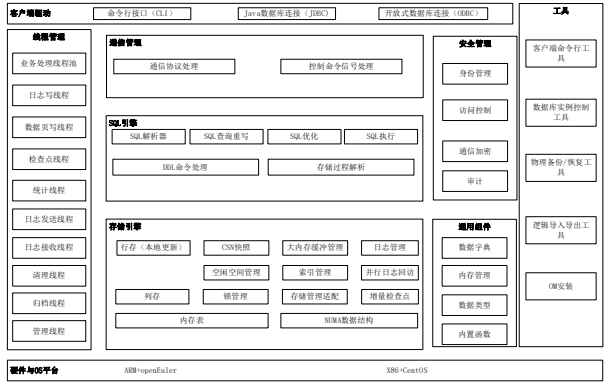
第 1 章 实验 1：openGauss 初探

1.1 实验介绍

openGauss 是华为开源发布的关系型数据库管理系统，具有多种优越的特性。openGauss 数据库在内核架构上具有多方面的创新与改进。在本实验中，我们将进行 openGauss 数据库的初探。首先介绍 openGauss 的特性与架构，学习如何准备 openGauss 数据库的安装环境，掌握 openGauss 数据库服务器的启动控制操作，学习 gsql 客户端连接数据库的基本命令用法。本实验是本系列 openGauss 数据库“实验之旅”的开端。

1.2 实验目的

1. 了解 openGauss 数据库的特性。
2. 了解 openGauss 数据库的架构。
3. 掌握 openGauss 数据库极简版的安装方法。
4. 掌握 openGauss 数据库服务器的控制方法。
5. 掌握 openGauss 数据库客户端 gsql 连接数据库的基本使用方法。



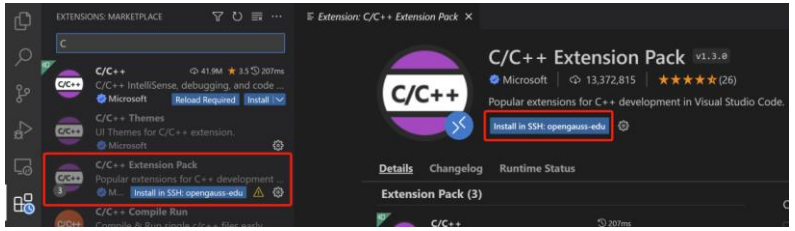
2.1　实验介绍

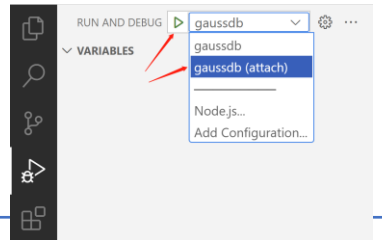
openGauss 数据库源代码的编译与调试是开展后续其他实验的工作基础。基于源代码的开发工作实际上就是按照“编辑——编译——测试——调试”这样的步骤循环进行的。“编辑”（edit）是指对源代码就行编辑修改，以达到预定目标；“编译”（compile）是将编辑修改后的源代码整体构建为可执行文件的过程；“测试”（test）包括了安装过程，对安装好的可执行文件进行测试，看是否达到了编辑代码要做到的预定目标；“调试”（debug）是根据测试中发现的问题，查找为什么没有达到预定目标的过程，定位查找到问题原因后，进行编辑修正，开启新一轮的上述循环过程，直到达到要编码实现的预定目标为止。openGauss 作为开源数据库管理系统，其开发过程遵循同样的流程。我们这套实验也是按此步骤，到 openGauss 源代码中去验证数据库系统原理的实现机制。因而，openGauss 的编译和调试是进行后续实验必须掌握的技能。

本章将通过实验介绍如何对 openGauss 源代码进行编译，并对编译后的 openGauss 数据库进行安装；关于 openGauss 的调试将在下一章涉及。

2.2　实验目的

1. 了解 openGauss 数据库的编译环境与配置要求。
2. 掌握 openGauss 源代码的编译和安装过程。
3. 掌握 openGauss 数据库的初始化、启动和连接方法。
4. 通过编译和安装过程深入理解 openGauss 数据库的服务器/客户端运行原理。
5. 掌握 openGauss 的编译和安装过程，为后续实验操作打下基础。





3.1　实验介绍

本实验将实践通过 VS Code 进行 openGauss 源代码调试的步骤。首先，介绍如何进行 VS Code 开发环境配置，对 openGauss 进行远程开发；然后，配置在 VS Code 中 openGauss 的调试环境，实践通过在源代码内设置断点进行单步调试；最后，实践以 attach 方式进行 openGauss 的调试过程。本实验通过添加代码，实现在执行 CREATE TABLE 语句时，输出一个循环变量值，以掌握 VS Code 开发环境以及 openGauss 单步调试过程。本实验践行了“编辑——编译——测试——调试”的迭代开发过程。读者可以在后续实验中不断通过实践体会该过程，为系统软件开发积累经验。

3.2　实验目的

1. 掌握 VS Code 远程开发环境的配置操作，能够满足 openGauss 源代码开发要求。
2. 掌握 VS Code 中 openGauss 调试环境的配置。
3. 掌握 VS Code 中 openGauss 源代码单步调试操作过程。
4. 掌握 VS Code 中以 attach 方式进行 openGauss 源代码调试操作过程。
5. 理解 openGauss 的“编辑——编译——测试——调试”迭代开发过程。

3.4.2 openGauss 调试环境配置

1. 确保远程主机上已安装了 gdb 调试器。可以执行命令“gdb -v”输出 gdb 版本信息进行测试，如果系统中已安装了 gdb，则如图 3.14 所示。

```
[dblab@eduog ~]$ gdb -v
GNU gdb (GDB) EulerOS 8.3.1-11.oe1
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

图 3.14 输出 gdb 版本信息

2. 在远程主机上进行 GDB 配置。确保 VS Code 已经通过 SSH 成功连接上远程主机，在 VS Code 中，在 openGauss 源代码目录下，新建.vscode 文件夹，在该文件夹下新建 launch.json 文件，文件内容如下：

```
{
    "configurations": [
        {
            "name": "gaussdb",
            "type": "cppdbg",
            "request": "launch",
            "program": "${workspaceFolder}/dest/bin/gaussdb",
            "args": [ "-D", "$GAUSSHOME/data" ],
            "cwd": "${workspaceFolder}",
```

3.4.3 openGauss 单步调试

下面在 openGauss 的一个源代码文件中添加几行代码来测试上一小节配置的 VS Code 调试环境。这里我们选择在 heap.cpp 文件的 heap_create_with_catalog 函数末尾添加一个简单的 for 循环语言，在 gsql 客户端输出循环变量。同时演示在 VS Code 中设置断点进行单步调试的过程。

1. 在 VS Code 中，打开文件 src/common/backend/catalog/heap.cpp。在函数 heap_create_with_catalog 末尾，“return relid;”语句之前，增加如下代码：

```
/* [ DBLAB ========================================== */
for (int i = 0; i < 3; i++) {
    ereport(INFO, (0, errmsg("i = %d", i)));
}
/* DBLAB ] ========================================== */
```

添加代码后，如图 3.15 所示。

```
3006    if (pg_partition_desc) {
3007        heap_close(pg_partition_desc, RowExclusiveLock);
3008    }
3009
3010    /* [ DBLAB ========================================================== */
3011    for (int i = 0; i < 3; i++) {
3012        ereport(INFO, (0, errmsg("i = %d", i)));
3013    }
3014    /* DBLAB ] ========================================================== */
3015
```

5. 在 VS Code 中，执行“Run”菜单的“Start Debugging”命令，以 gdb 调试模式启动 openGauss 数据库服务器进程 gaussdb，如图 3.16 所示。可以看到，以调试模式启动 gaussdb 完成后，VS Code 下方的状态栏变为橙黄色

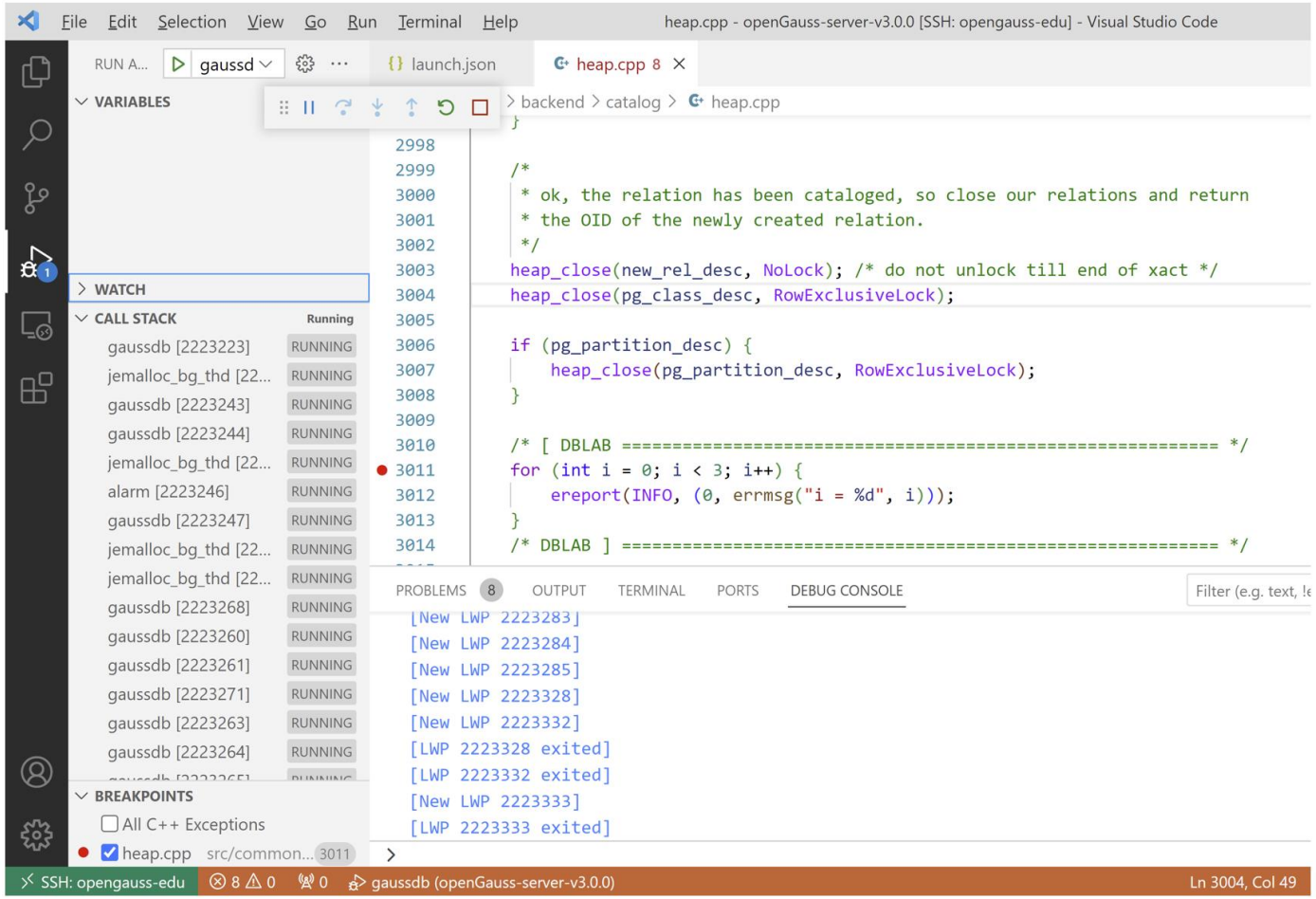


图 3.16 VS Code 以调试模式启动 openGauss 服务器进程 gaussdb

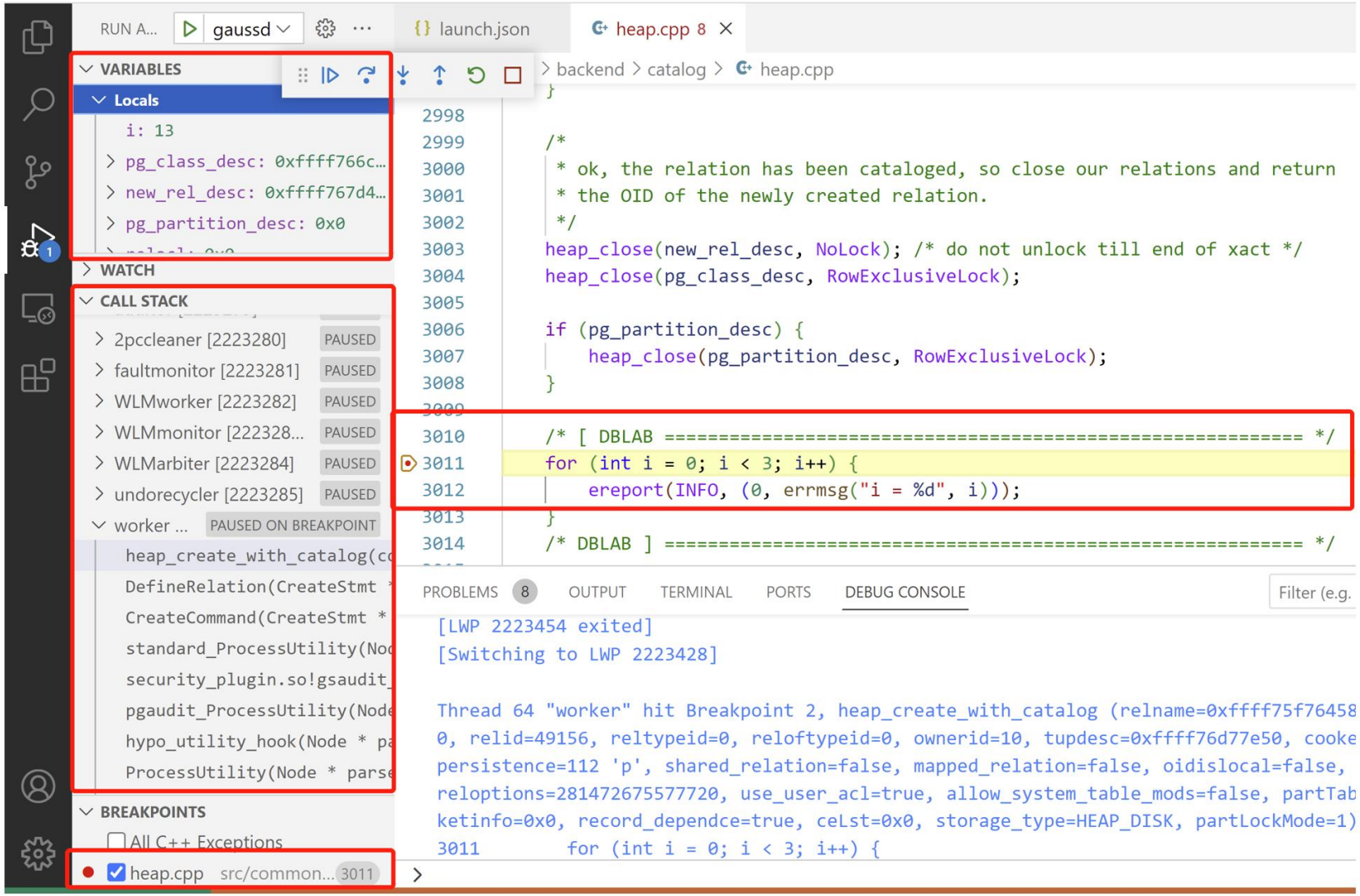


图 3.17 VS Code 调试 openGauss 设置断点生效

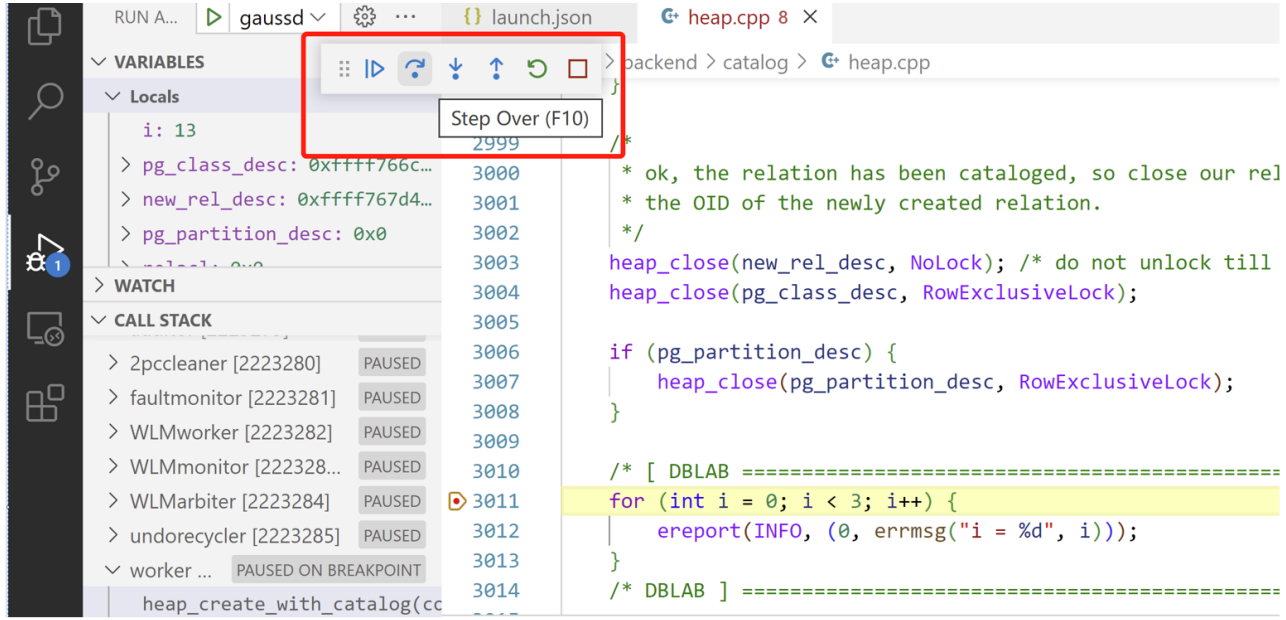


图 3.18 在 VS Code 中进行单步执行调试代码

10. 继续单击“Step Over”按钮，进行单步执行。观察变量 i 的变化以及 gsql 中输出的变化。如图 3.19 所示。

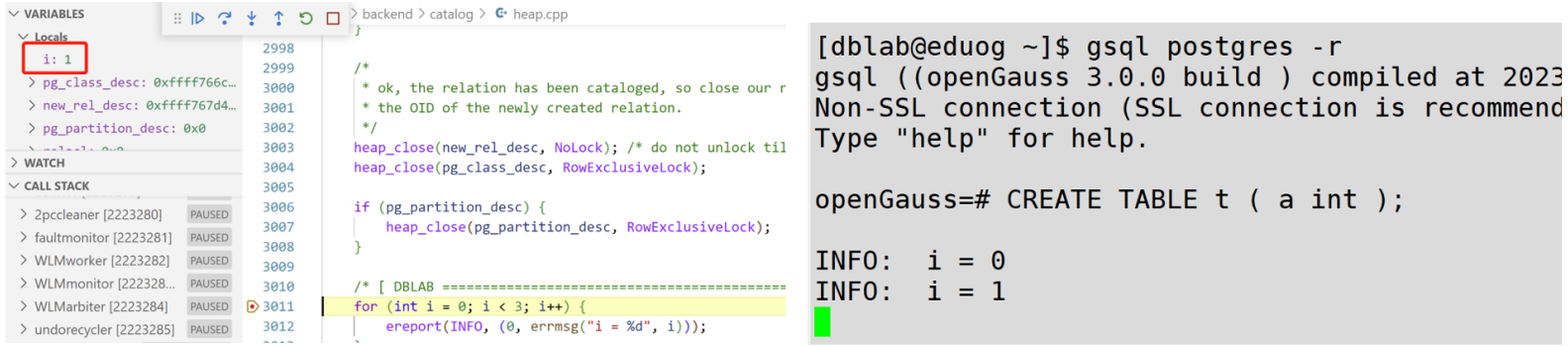


图 3.19 单步执行调试代码时变量的变化以及 gsql 的输出

1. 本实验中，要添加代码的位置是 3.4.3 节所添加代码（已注释掉）的后面，函数 heap_create_with_catalog 末尾最后一行语句“return relid;”之前。

- 文件：src/common/backend/catalog/heap.cpp
- 函数：heap_create_with_catalog

```
3014    /* DBLAB ] ============================================================ */
3015
3016    /* [ DBLAB ============================================================ */
3017    Form_pg_class pg_class = new_rel_desc->rd_rel;
3018    TupleDesc t_desc = new_rel_desc->rd_att;
3019    const int buf_size = 512;
3020    char info[buf_size];
3021    int idx = sprintf_s(info, buf_size, "\n"
3022                "relation's object id : %u\n"
3023                "pg_class->relname : %s\n"
3024                "pg_attribute->attname : \n",
3025                new_rel_desc->rd_id,
3026                pg_class->relname.data);
3027    for (int i = 0; i < t_desc->natts; i++) {
3028        if (idx + 1 == buf_size)
3029            break;
3030        Form_pg_attribute pg_attribute = t_desc->attrs[i];
3031        char* attname = pg_attribute->attname.data;
3032        Oid oid_type = pg_attribute->atttypid;
3033        HeapTuple tup = SearchSysCache1(TYPEOID, ObjectIdGetDatum(oid_type));
3034        Form_pg_type type_tup = (Form_pg_type) GETSTRUCT(tup);
3035        char* typname = type_tup->typname.data;
3036        int typlen = type_tup->typlen;
3037        int len = sprintf_s(&info[idx], buf_size - idx,
3038                    "attr[%d].attname : %s typname : %s typlen : %d\n",
3039                    i, attname, typname, typlen);
3040        idx += len;
3041    }
3042    info[idx] = '\0';
3043    ereport(INFO, (0, errmsg("%s", info)));
3044    /* DBLAB ] ============================================================ */
3045
3046    return relid;
3047 }
```

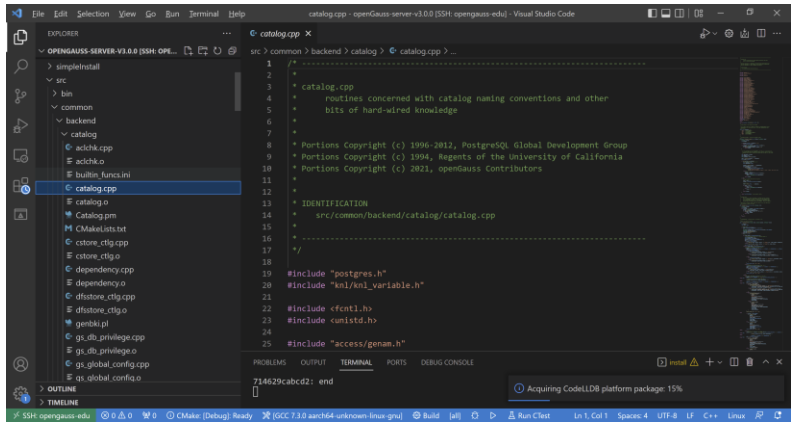
图 3.29 添加代码输出 CREATE TABLE 的相关信息

2. 编译添加的代码并安装，启动数据库服务器，gsql 客户端连接数据库服务器，输入 3.4.4 节中的 CREATE TABLE 语句，创建 users 表。gsql 的执行结果如下：

```
openGauss=# CREATE TABLE users
openGauss-# (
openGauss(# u_uid varchar(20),
openGauss(# u_passwd varchar(20),
openGauss(# u_name varchar(10),
openGauss(# u_idtype int,
openGauss(# u_idnum varchar(20),
openGauss(# u_regtime timestamp
openGauss(# );
INFO:
relation's object id : 57354
pg_class->relname : users
pg_attribute->attname :
attr[0].attname : u_uid typname : varchar typlen : -1
attr[1].attname : u_passwd typname : varchar typlen : -1
attr[2].attname : u_name typname : varchar typlen : -1
attr[3].attname : u_idtype typname : int4 typlen : 4
attr[4].attname : u_idnum typname : varchar typlen : -1
attr[5].attname : u_regtime typname : timestamp typlen : 8

CREATE TABLE
openGauss=#
```

可以看到，我们添加的代码被成功执行。通过访问 CREATE TABLE 语句新创建的 users 表的 Relation 中的 pg_class 和 pg_attribute，获得并输出了表的信息和表中各属性列的信息，包括：表的的对象 id、表的名称、各属性的名称、各属性的类型及

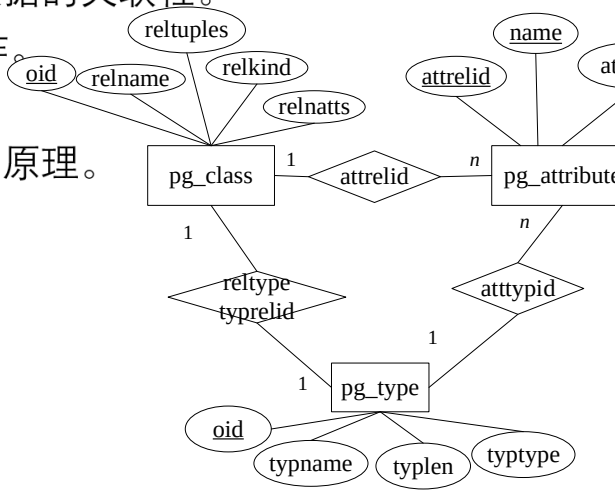


4.1　实验介绍

本实验通过探索 openGauss 数据库服务器中 CREATE TABLE 语句执行所对应的相关源代码，引出 openGauss 的系统表机制。在 openGauss 数据库系统中，每创建一张表都会自动在系统表中存储相关信息。本实验以 pg_class、pg_attribute 和 pg_type 这三个重要的系统表为牵引，通过源代码解析深入了解关系表的定义以及属性列的详细信息。通过查询系统表，理解与表定义相关的元数据组织结构；通过浏览重要结构体与系统表源代码，了解代码层面的细节内容。

4.2　实验目的

1. 掌握 CREATE TABLE 语句执行与三个系统表中元数据的关联性。
2. 掌握三个系统表的关键属性、表间关联与查询操作。
3. 了解与本实验相关的重要结构体与系统表源代码。
4. 理解通过添加代码输出 CREATE TABLE 相关系统的原理。



【请按照要求完成实验操作并粘贴截图】

1. 完成实验步骤 3.4.1 节，打开源代码文件 src/common/backend/catalog/heap.cpp，将操作成功的截图粘贴在下面。

2. 完成实验步骤 3.4.2 和 3.4.3，获得图 3.19 的效果，将操作成功的截图粘贴在下面。

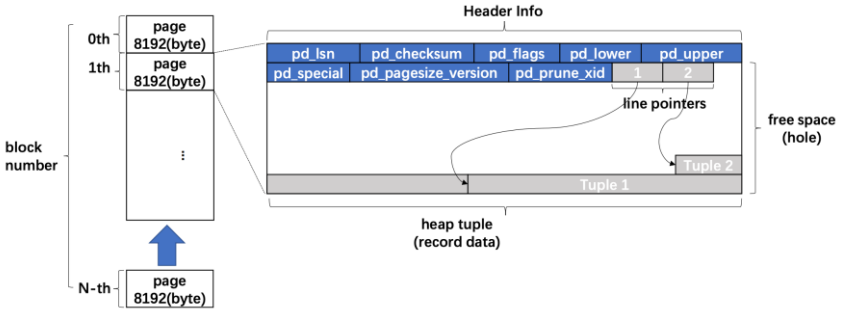
3. 完成实验步骤 3.4.4，获得创建 users 表成功的 gsql 命令行客户端截图，将操作成功的截图粘贴在下面。

4. 完成实验步骤 3.4.5，就其中给出的系统表属性，画出系统表 pg_class、pg_attribute、pg_type 之间的关系的 ER 图。

5. 完成实验步骤 3.4.7，验证添加代码的执行效果，将 gsql 中 CREATE TABLE users 语句执行的截图粘贴在下面。

3.6 讨论与总结

【请将实验中遇到的问题描述、解决办法与思考讨论列在下面，并对本实验进行总结。】



5.1　实验介绍

本实验将介绍和实践 openGauss 数据库中表的页面存储结构，即数据在 openGauss 数据库中是以何种方式被存储和组织的。在 openGauss 中，每个表的存储文件都被分成若干个页面，每个页面存储一定量的数据。每个页面的开头都有一个 PageHeaderData 结构体，该结构体包含了关于页面的元数据，例如页面的版本号，检查和标志等。随后的数据存储在页面的主体中，这些数据被存储为一个个元组（tuple），每个元组都有一个对应的 ItemIdData 结构体。该结构体包含了元组的元数据，例如该元组的大小，是否被删除等。每个元组的实际数据存储在 HeapTupleHeaderData 结构体中，该结构体包含了关于该元组的元数据，例如元组的版本号，多元组事务 ID 等。openGauss 表的页面存储结构是一种关系数据的高效物理存储方式，它通过组织元数据和实际数据来实现快速存储和读取。

本实验将首先介绍 openGauss 中的数据文件和堆表存储结构以及元组更新和删除过程；接着通过 pageinspect 插件对表的页面结构进行具体实验与分析。同时，在本实验中，还将建立 railway 示例数据库，用于后续的实验操作。

5.2　实验目的

1. 理解 openGauss 数据文件组织与堆表存储结构、元组更新和删除过程。
2. 了解与堆表数据页面结构相关的重要结构体及其源代码。
3. 掌握使用 pageinspect 插件分析表的页面结构。
4. 掌握实验示例数据库 railway 的表模式、外键约束与数据构成。
5. 理解关系数据库中关系表的物理存储结构。

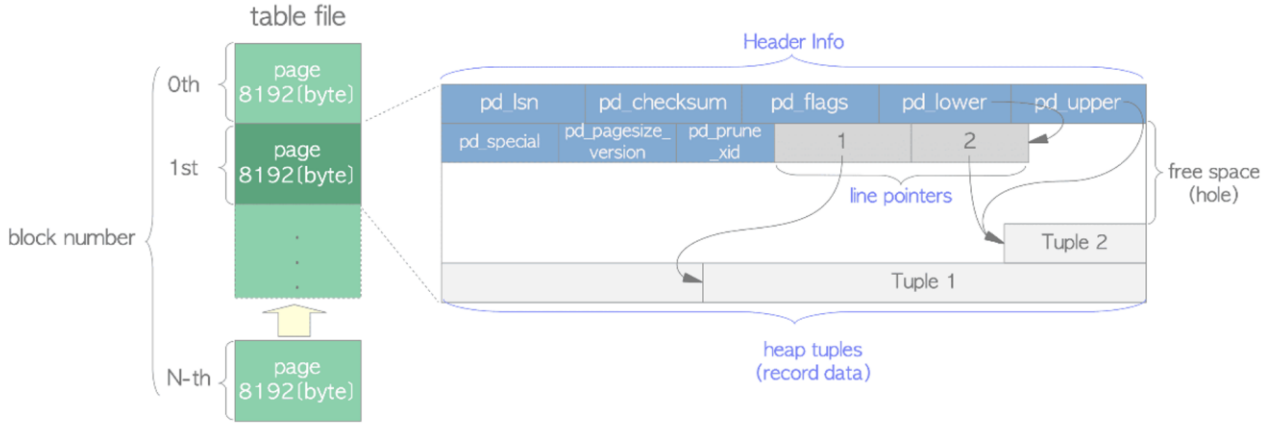


图 4.1 openGauss 数据文件示意图

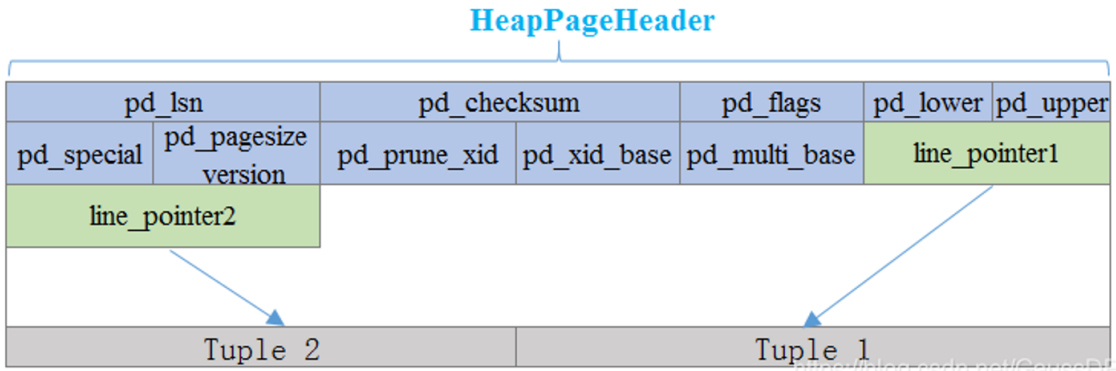


图 4.2 堆表页面示意图

▪ 5.3.5 相关结构体

本实验涉及到的几个重要的结构体如下：

1. 结构体 PageHeaderData 及其指针 PageHeader

PageHeaderData 结构体表示数据库文件中一个数据页面的页头。页头是数据库文件中数据页的第一部分，包含关于该页的信息，包括其大小，该页上可用的空闲空间以及存储在页上的项目数。PageHeaderData 结构提供了一种访问和操纵存储在页头中的信息的方法。

PageHeaderData 结构体的字段说明：

- pd_lsn：记录最后一次对页面修改的 xlog 日志记录 id
- pd_checksum：页面的检查和
- pd_flags：标志位 flag.
- pd_lower：空闲空间的起始处（距离页头）
- pd_upper：空闲空间的结尾处（距离页头）
- pd_special：页面预留空间的开始处（距离页头）
- pd_pagesize_version：页面大小及版本号
- pd_prune_xid：页面清理辅助事务 id（32 位），通常为该页面内现存最老的删除或更新操作的事务 id，用于判断是否要触发页面级空闲空间整理。
- pd_linp：行（元组/项）指针数组。

【源码】src/include/storage/buf/bufpage.h：

```
/*
 * disk page organization
 *
 * space management information generic to any page
 *
 *      pd_lsn      - identifies xlog record for last change to this page.
 *      pd_checksum - page checksum, if set.
 *      pd_flags    - flag bits.
```

▪ 5.4.3 使用 pageinspect 插件分析表的页面结构

为了使用 pageinspect 插件分析表的页面结构，在 railway 数据库中创建一个新的表 users2，与 users 表结构完全相同，只是表名不同。

```
railway=# CREATE TABLE users2
(
   u_id varchar(20),              -- 用户 id，用于系统登录账户名（主键）
   u_passwd varchar(20),          -- 密码，用于系统登录密码
   u_name varchar(10),            -- 真实姓名
   u_idnum varchar(20),           -- 证件号码
   u_regtime timestamp,           -- 注册时间
   CONSTRAINT pk_users2 PRIMARY KEY (u_id)
);
NOTICE:  CREATE TABLE / PRIMARY KEY will create implicit index "pk_users2" for table "users2"
CREATE TABLE
```

1. 查看表页面的页头结构。

   查看空表的页头结构：

   在 users2 表创建后，但还未插入数据之前，执行：

```
railway=# SELECT * FROM page_header(get_raw_page('users2', 0));
ERROR:  block number 0 is out of range for relation "users2"
```

   返回错误信息“block number 0 is out of range for relation "users2"”，说明空表没有添加任何数据，也就没有创建任何页面，编号为 0 的页面不存在。

   执行一条插入语句：

```
railway=# INSERT INTO users2 VALUES(1, 'qweasd', '张三', '123456789', '2000-06-23
12:00:00');
INSERT 0 1
```

   再执行下列语句并查看返回结果：

```
railway=# SELECT* FROM page_header(get_raw_page('users2', 0));
    lsn    | tli | flags | lower | upper | special | pagesize | version | prune_xid
-----------+-----+-------+-------+-------+---------+----------+---------+-----------
 0/4F095C0 |   0 |     0 |    44 |  8120 |    8192 |     8192 |       6 |     22470
(1 row)
```

图 4.9 插入第 1 条数据后查看表页面的页头结构

观察可见，此时页面 0 已经创建。页头所占字节数分析如下：

lsn：8 字节

tli：2 字节

flags: 2 字节

lower: 2 字节

upper: 2 字节

special: 2 字节

pagesize、version：2 字节

prune_xid：4 字节

---------- 24 字节 ----------

pd_xid_base：8 字节

pd_multi_base：8 字节

---------- 40 字节 ----------

一个 ItemIdData pd_linp 元组指针数组项：4 字节

---------- 44 字节 ----------

可以看到，lower 列的值恰为 44。页面大小为 8192 字节，upper 列的值为 8120，即刚插入的第 1 条元组占用了 72 字节。

```
ENL1_printf("testing qualification");

/* [ DBLAB ============================================================ */
TupleTableSlot* outer_slot = econtext->ecxt_outertuple;
TupleTableSlot* inner_slot = econtext->ecxt_innertuple;
HeapTuple outer_ht = ExecFetchSlotTuple(outer_slot);
HeapTuple inner_ht = ExecFetchSlotTuple(inner_slot);
Relation outer_rel = RelationIdGetRelation(outer_ht->t_tableOid);
Relation inner_rel = RelationIdGetRelation(inner_ht->t_tableOid);
char* outer_relname = RelationGetRelationName(outer_rel);
char* inner_relname = RelationGetRelationName(inner_rel);
if (strcmp(outer_relname, "users") == 0 && strcmp(inner_relname, "orders") == 0 ||
    strcmp(outer_relname, "orders") == 0 && strcmp(inner_relname, "users") == 0) {
    TupleDesc outer_td = outer_slot->tts_tupleDescriptor;
    TupleDesc inner_td = inner_slot->tts_tupleDescriptor;
    // users.u_id (1st att) or orders.o_uid (2nd att)
    int outer_num = strcmp(outer_relname, "users") == 0 ? 1 : 2;
    int inner_num = strcmp(inner_relname, "orders") == 0 ? 2 : 1;
    char* outer_attname = outer_td->attrs[outer_num - 1]->attname.data;
    char* inner_attname = inner_td->attrs[inner_num - 1]->attname.data;
    bool isnull;
    Datum outer_attr = heap_getattr(outer_ht, outer_num, outer_td, &isnull);
    Datum inner_attr = heap_getattr(inner_ht, inner_num, inner_td, &isnull);
    ereport(INFO, (0, errmsg("outer attr: %s = %s <--> inner attr: %s = %s",
            outer_attname,
            text_to_cstring(DatumGetVarCharP(outer_attr)),
            inner_attname,
            text_to_cstring(DatumGetVarCharP(inner_attr)))));
}
/* DBLAB ] ============================================================ */

if (ExecQual(joinqual, econtext, false)) {
```

6.1　实验介绍

本实验通过阅读和分析 openGauss 数据库服务器中嵌套循环连接算法实现源代码，理解和验证嵌套循环连接算法的实现机制。首先介绍嵌套循环连接算法的原理，通过浏览 ExecNestLoop 函数源代码和相应的流程图，实践嵌套循环连接算法的具体工程实现。在浏览阅读相关结构体源代码之后，以 railway 数据库中的 users 表和 orders 表的连接查询作为示例，通过添加源代码进行宏重定义的方式，尝试输出 ExecNestLoop 函数调试信息。通过配置优化器参数改变查询执行计划，引发 ExecNestLoop 函数执行。最后，通过添加代码，输出嵌套循环算法中进行连接属性值的比较信息，进一步理解循环嵌套连接的查询执行计划中索引扫描与顺序扫描的区别。

本实验的实践内容涉及到数据库服务器的查询执行与查询优化机制，相关 openGauss 源代码关联的细节众多，具有一定的复杂性和挑战性。在添加代码过程中，引入错误 (bug) 也是不可避免的一件事情。在诸如 openGauss 这样大型、复杂、服务器架构的系统软件中，通过贯彻“编辑——编译——测试——调试”的迭代步骤，采取系统性的调试手段修正错误，可以有效地提升解决复杂系统工程问题的能力并增长相应的系统开发与调试经验。

6.2　实验目的

1. 理解嵌套循环连接算法的原理。
2. 理解 ExecNestLoop 函数代码实现与嵌套循环连接算法的对应关系。
3. 掌握通过宏重定义方式输出 ExecNestLoop 函数调试信息的方法。
4. 掌握通过在 ExecNestLoop 函数添加代码输出嵌套循环算法中字段值的方法。
5. 理解通过 EXPLAIN 语句查看的查询执行计划。
6. 理解通过优化器参数配置改变查询执行计划的方式。
7. 了解与本实验相关的结构体与源代码。

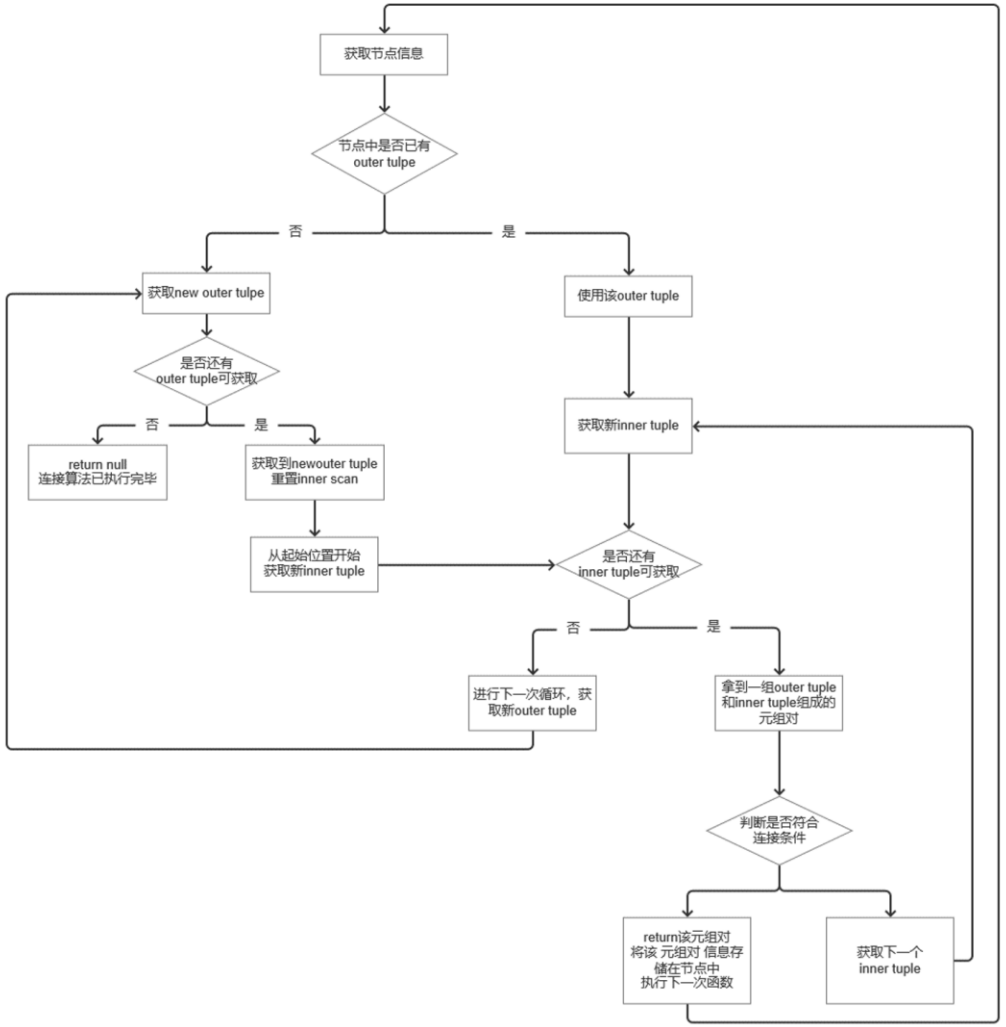


图 5.1 openGauss 中的嵌套循环算法源代码的主要流程图

```
/* [ DBLAB ============================================================ */
#undef ENL1_printf
#define ENL1_printf(message) ereport(INFO, (0, errmsg("ExecNestLoop: %s", message)))
/* DBLAB ] ============================================================ */
```

该段代码的添加位置如图 5.2 所示。

```
27  #include "executor/exec/execStream.h"
28  #include "utils/memutils.h"
29  #include "executor/node/nodeHashjoin.h"
30
31  /* [ DBLAB ============================================================ */
32  #undef ENL1_printf
33  #define ENL1_printf(message) ereport(INFO, (0, errmsg("ExecNestLoop: %s", message)))
34  /* DBLAB ] ============================================================ */
35
36
37  static void MaterialAll(PlanState* node)
38  {
39      if (IsA(node, MaterialState)) {
40          ((MaterialState*)node)->materalAll = true;
```

图 5.2 添加 ENL1_printf 宏定义以输出 ExecNestLoop 函数调试信息

2. 停止 openGauss 服务：

```
[dblab@eduog openGauss-server-v3.0.0]$ gs_ctl stop -D $GAUSSHOME/data -Z single_node -l logfile
```

3. 对 openGauss 进行编译和安装：

```
[dblab@eduog openGauss-server-v3.0.0]$ make -j4
[dblab@eduog openGauss-server-v3.0.0]$ make install
```

4. 启动 openGauss：

```
[dblab@eduog openGauss-server-v3.0.0]$ gs_ctl start -D $GAUSSHOME/data -Z single_node -l logfile
```

5. 使用 gsql 再次连接 railway 数据库，执行 users 表与 orders 表的连接查询：

```
railway=# SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid;
 u_id | u_name | o_id | o_uid | o_tid
------+--------+------+-------+-------
 1    | 张三   |    1 | 1     | C2002
 3    | 王五   |    2 | 3     | G321
 3    | 王五   |    3 | 3     | G1709
(3 rows)
```

但发现没有输出预期的 ExecNestLoop 中的调试信息！那只有一种可能，即 ExecNestLoop 函数根本没有被执行。查看一条 SQL 语句是具体如何被执行的，即查看具体的查询执行计划，可使用 EXPLAIN 语句。

6. 执行 EXPLAIN 语句，查看上述连接查询的执行计划：

```
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id
= o_uid;
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=16.30..33.12 rows=287 width=196)
   Hash Cond: ((orders.o_uid)::text = (users.u_id)::text)
   ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
   ->  Hash  (cost=12.80..12.80 rows=280 width=96)
         ->  Seq Scan on users  (cost=0.00..12.80 rows=280 width=96)
(5 rows)
```

经观察，果然查询执行计划显示使用的是 Hash 连接（hash join），而不是嵌套循环连接。采用传统的表连接语法，发现 EXPLAIN 的结果是一样的。

（关于 EXPLAIN 语言输出的查询执行计划的解释，可参见文档 https://www.postgresql.org/docs/current/using-explain.html）

```
railway=# SELECT u_id, u_name, o_id, o_uid, o_tid FROM users, orders WHERE u_id = o_uid;
 u_id | u_name | o_id | o_uid | o_tid
------+--------+------+-------+-------
 1    | 张三   |    1 | 1     | C2002
 3    | 王五   |    2 | 3     | G321
 3    | 王五   |    3 | 3     | G1709
(3 rows)

railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users, orders WHERE u_id = o_uid;
                              QUERY PLAN
-----------------------------------------------------------------------
 Hash Join  (cost=16.30..33.12 rows=287 width=196)
   Hash Cond: ((orders.o_uid)::text = (users.u_id)::text)
   ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
   ->  Hash  (cost=12.80..12.80 rows=280 width=96)
         ->  Seq Scan on users  (cost=0.00..12.80 rows=280 width=96)
(5 rows)
```

openGauss 选择了使用基于哈希的连接（Hash Join）算法，而非嵌套循环连接（Nested Loop Join）算法。

7. 通过 SET 配置参数 enable_hashjoin 为 off，禁止使用 Hash Join 连接算法。（关于查询执行计划相关的参数选项，可参见文档 https://www.postgresql.org/docs/current/runtime-config-query.html）

```
railway=# SET enable_hashjoin = off;
SET
```

8. 再次执行查询：

```
railway=# SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid;
 u_id | u_name | o_id | o_uid | o_tid
------+--------+------+-------+-------
 1    | 张三   |    1 | 1     | C2002
 3    | 王五   |    2 | 3     | G321
 3    | 王五   |    3 | 3     | G1709
(3 rows)
```

发现还是没有输出预期的 ExecNestLoop 中的调试信息！

9. 执行 EXPLAIN 语句，查看查询执行计划：

```
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id
= o_uid;
                              QUERY PLAN
-----------------------------------------------------------------------
 Merge Join  (cost=48.77..54.47 rows=287 width=196)
   Merge Cond: ((users.u_id)::text = (orders.o_uid)::text)
   ->  Sort  (cost=24.18..24.88 rows=280 width=96)
         Sort Key: users.u_id
         ->  Seq Scan on users  (cost=0.00..12.80 rows=280 width=96)
   ->  Sort  (cost=24.59..25.30 rows=287 width=100)
         Sort Key: orders.o_uid
         ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
(8 rows)
```

这次，openGauss 选择了使用排序归并连接（Sort-Merge Join）算法，还不是嵌套循环连接（Nested Loop Join）算法。

（关于 Merge Join 连接算法的详细介绍，参见教材“Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom. Database Systems: The Complete Book. Pearson; 2nd edition (June 5, 2008)” 15.4 节 Two-Pass Algorithms Based on Sorting）

10. 通过 SET 配置参数 enable_mergejoin 为 off，禁止使用 Merge Join 连接算法：

```
railway=# SET enable_mergejoin = off;
SET
```

10. 通过 SET 配置参数 enable_mergejoin 为 off，禁止使用 Merge Join 连接算法：

```
railway=# SET enable_mergejoin = off;
SET
```

11. 再次执行连接查询：

```
SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid;
```

成功输出了 ExecNestLoop 函数中的调试信息，如下：

```
railway=# SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id = o_uid;
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting info from node
INFO:  ExecNestLoop: entering main loop
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: no inner tuple, need new outer tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: saving new outer tuple information
INFO:  ExecNestLoop: rescanning inner plan
INFO:  ExecNestLoop: getting new inner tuple
INFO:  ExecNestLoop: testing qualification
INFO:  ExecNestLoop: qualification succeeded, projecting tuple
INFO:  ExecNestLoop: getting new outer tuple
INFO:  ExecNestLoop: no outer tuple, ending join
 u_id | u_name | o_id | o_uid | o_tid
------+--------+------+-------+-------
 1    | 张三   |    1 | 1     | C2002
 3    | 王五   |    2 | 3     | G321
 3    | 王五   |    3 | 3     | G1709
(3 rows)
```

12. 执行 EXPLAIN 语句，查看查询执行计划：

```
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id
= o_uid;
                                QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop  (cost=0.00..141.23 rows=287 width=196)
   ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
   ->  Index Scan using pk_users on users  (cost=0.00..0.44 rows=1 width=96)
         Index Cond: ((u_id)::text = (orders.o_uid)::text)
(4 rows)
```

发现这次执行 openGauss 终于使用了 Nested Loop 连接算法！通过观察，发现此查询执行计划中，对于 orders 表进行了顺序扫描（Seq Scan）操作，而对于 users 表进行的是索引扫描（Index Scan）操作。使用了名为 pk_users 索引，该索引正是 users 表的主键对应的默认索引。

- **6.4.4 添加代码：输出嵌套循环算法的比较信息**

通过在 ExecNestLoop 函数中添加代码，对内外层两个表的连接字段的值进行输出。

1. 在嵌套循环算法代码 ExecNestLoop 函数中，添加以下代码：

```
/* [ DBLAB ========================================================== */
TupleTableSlot* outer_slot = econtext->ecxt_outertuple;
TupleTableSlot* inner_slot = econtext->ecxt_innertuple;
HeapTuple outer_ht = ExecFetchSlotTuple(outer_slot);
HeapTuple inner_ht = ExecFetchSlotTuple(inner_slot);
Relation outer_rel = RelationIdGetRelation(outer_ht->t_tableOid);
Relation inner_rel = RelationIdGetRelation(inner_ht->t_tableOid);
char* outer_relname = RelationGetRelationName(outer_rel);
char* inner_relname = RelationGetRelationName(inner_rel);
if (strcmp(outer_relname, "users") == 0 && strcmp(inner_relname, "orders") == 0 ||
    strcmp(outer_relname, "orders") == 0 && strcmp(inner_relname, "users") == 0) {
    TupleDesc outer_td = outer_slot->tts_tupleDescriptor;
    TupleDesc inner_td = inner_slot->tts_tupleDescriptor;
    // users.u_id (1st att) or orders.o_uid (2nd att)
    int outer_num = strcmp(outer_relname, "users") == 0 ? 1 : 2;
    int inner_num = strcmp(inner_relname, "orders") == 0 ? 2 : 1;
    char* outer_attname = outer_td->attrs[outer_num - 1]->attname.data;
    char* inner_attname = inner_td->attrs[inner_num - 1]->attname.data;
    bool isnull;
    Datum outer_attr = heap_getattr(outer_ht, outer_num, outer_td, &isnull);
    Datum inner_attr = heap_getattr(inner_ht, inner_num, inner_td, &isnull);
    ereport(INFO, (0, errmsg("outer attr: %s = %s <--> inner attr: %s = %s",
            outer_attname,
            text_to_cstring(DatumGetVarCharP(outer_attr)),
            inner_attname,
            text_to_cstring(DatumGetVarCharP(inner_attr)))));
}
/* DBLAB ] ========================================================== */
```

添加代码的具体位置如图 6.3 所示。

```
INFO:  outer attr: o_uid = 1 <--> inner attr: o_id = 1
INFO:  outer attr: o_uid = 3 <--> inner attr: o_id = 3
INFO:  outer attr: o_uid = 3 <--> inner attr: o_id = 3
```

7. 通过 SET 配置参数 enable_indexscan 为 off，禁止使用索引扫描。

```
railway=# SET enable_indexscan = off;
SET
```

8. 执行 EXPLAIN 语句，查看查询执行计划：

```
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id
= o_uid;
                              QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop  (cost=0.00..1231.77 rows=287 width=196)
   Join Filter: ((users.u_id)::text = (orders.o_uid)::text)
   ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
   ->  Materialize  (cost=0.00..14.20 rows=280 width=96)
         ->  Seq Scan on users  (cost=0.00..12.80 rows=280 width=96)
(5 rows)
```

对比此时查询执行计划与前面查询执行计划的区别。可以看到，orders 表和 users 表均为 Seq Scan 即顺序扫描操作。但在 users 表的 Seq Scan 上面，有一步 Materialize 即物化操作。

9. 通过 SET 配置参数 enable_material 为 off，禁止使用物化操作。

```
railway=# SET enable_material = off;
SET
```

10. 执行 EXPLAIN 语句，查看查询执行计划：

```
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id
= o_uid;
                                QUERY PLAN
```

11. 通过 SET 配置参数 enable_ bitmapscan 为 off，禁止使用位图扫描操作。

```
railway=# SET enable_ bitmapscan = off;
SET
```

12. 执行 EXPLAIN 语句，查看查询执行计划：

```
railway=# EXPLAIN SELECT u_id, u_name, o_id, o_uid, o_tid FROM users INNER JOIN orders ON u_id
= o_uid;
                              QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop  (cost=0.00..4620.90 rows=287 width=196)
   Join Filter: ((users.u_id)::text = (orders.o_uid)::text)
   ->  Seq Scan on users  (cost=0.00..12.80 rows=280 width=96)
   ->  Seq Scan on orders  (cost=0.00..12.87 rows=287 width=100)
(4 rows)
```

观察发现，此时的查询执行计划中只有 Seq Scan 操作和 Join Filter 操作了。其中 Join Filter 就是两个连接属性值是否相等的条件判断。但是与之前使用 Index Scan 的查询执行计划相比，外表和内表发生了互换。这是 openGuass 的优化器做出的动态调整，用户无法自行改变。不过我们之前添加的代码是考虑了外表和内表交换的情况的，能够根据表名确定对应的属性编号。

13. gsql 连接数据库 railway，并禁用之前提及的优化器参数：

```
[dblab@eduog openGauss-server-v3.0.0]$ gsql railway -r
gsql ((openGauss 3.0.0 build ) compiled at 2023-01-29 11:22:50 commit 0 last mr  debug)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.

railway=#

SET enable_hashjoin = off;
SET enable_mergejoin = off;
SET enable_indexscan = off;
SET enable_material = off;
SET enable_bitmapscan = off;
```

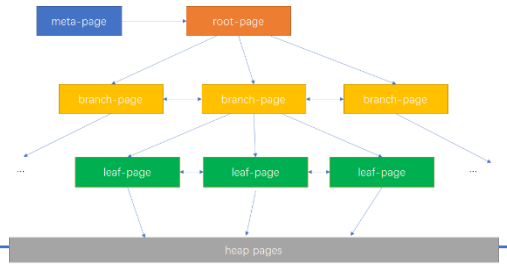
此次，我们终于可以看到了，该查询按照嵌套循环算法的流程进行了连接属性值的比较：

```
INFO:  outer attr: u_id = 1 <--> inner attr: o_uid = 1
INFO:  outer attr: u_id = 1 <--> inner attr: o_uid = 3
INFO:  outer attr: u_id = 1 <--> inner attr: o_uid = 3
INFO:  outer attr: u_id = 2 <--> inner attr: o_uid = 1
INFO:  outer attr: u_id = 2 <--> inner attr: o_uid = 3
INFO:  outer attr: u_id = 2 <--> inner attr: o_uid = 3
INFO:  outer attr: u_id = 3 <--> inner attr: o_uid = 1
INFO:  outer attr: u_id = 3 <--> inner attr: o_uid = 3
INFO:  outer attr: u_id = 3 <--> inner attr: o_uid = 3
```

观察输出信息，就是按照嵌套循环算法步骤，外表 users 的当前元组与内表 orders 的当前元组进行了 9 次比较，即对于 users 表的每行，orders 表进行一轮遍历，每轮遍历进行 3 次比较。

至此，从实验的角度验证了嵌套循环算法原理。

如果在执行上述连接查询 SQL 语句时数据库服务器崩溃或 gsql 连接断开，或没有实现预期的输出效果，是由于编写代码过程中引入了错误（bug）。此时，可通过使用第 3 章中已实践的单步调试方法对添加代码进行调试（debug），以排除错误。通过“编辑——编译——测试——调试”这样的步骤循环，直到成功执行为止。



7.1　实验介绍

本实验通过阅读和分析 openGauss 中与 B+树索引构建相关的实现源代码，理解和验证索引的构建流程、使用方法与系统开销。首先回顾索引结构与 B+树索引的原理，总结 CREATE INDEX 语句的用法，通过浏览 openGauss 中 btree 索引构建的相关源代码，实践 btree 索引的具体工程实现。通过一系列实验，对 btree 索引构建、索引与查询执行计划的关联、索引相关系统表、索引的作用与开销有更加深刻的认识和理解。通过添加代码的方法，对 btree 索引构建过程中的关键环节信息进行输出，进一步理解索引结构的原理与实现。

本实验的实践内容涉及到 B+树索引在 openGauss 中的具体实现，实验内容较多且综合性较强。如添加的代码引入错误，需要通过“编辑——编译——测试——调试”的迭代步骤进行耐心排错，查找问题所在，增强系统软件的调试能力。

7.2　实验目的

1. 理解索引结构与 B+树索引的原理。
2. 掌握在 openGauss 中用 CREATE INDEX 语句构建索引的基本方法。
3. 理解在 openGauss 中 btree 索引的构建流程及其在源代码中的实现。
4. 理解索引构建与查询执行计划的关联。
5. 了解在 openGauss 中与索引构建相关的系统表信息。
6. 理解索引的作用与开销。
7. 掌握在 openGauss 中添加代码输出 btree 索引相关信息的方法。
8. 掌握使用 pageinspect 插件分析索引页面的方法。
9. 了解与本实验相关的函数与结构体的源代码。

添加代码：分析btree索引构建过程

▪ 7.4.4 添加代码：分析 btree 索引构建过程

在本步骤中，在构建 btree 索引的相关底层函数中添加代码，向 gsql 客户端输出 btree 节点构建过程中的相关信息。

根据前面的源代码分析，相关函数的调用顺序为：

(1) DefineIndex (indexcmds.cpp)

(2) index_create (index.cpp)

(3) index_build (index.cpp)

(4) index_build_storage (index.cpp)

(5) btbuild (nbtree.cpp)

(6) _bt_leafbuild (nbtsort.cpp)

(7) _bt_load (nbtsort.cpp)

(8) _bt_buildadd (nbtsort.cpp)

下面我们在 btbuild、_bt_leafbuild 和_bt_buildadd 函数中添加代码。

1. 在函数 btbuild 中添加代码

在函数_bt_leafbuild 执行完之后，输出构建索引的关系表中的元组数量以及插入到索引中的元组数量。

【源码】src/gausskernel/storage/access/nbtree/nbtree.cpp

```
/*
 *  btbuild() -- build a new btree index.
 */
Datum btbuild(PG_FUNCTION_ARGS)
```

【源码】src/gausskernel/storage/access/nbtree/nbtree.cpp

```
/*
 *  btbuild() -- build a new btree index.
 */
Datum btbuild(PG_FUNCTION_ARGS)
{
... 省略若干行
        // Return statistics
        result = (IndexBuildResult *)palloc(sizeof(IndexBuildResult));
        result->heap_tuples = reltuples;
        result->index_tuples = buildstate.indtuples;
        result->all_part_tuples = allPartTuples;

        /* [ DBLAB ============================================================ */
        ereport(INFO, (0, errmsg("btbuild: [# of tuples seen in the relation] = %.0f [# of tuples
inserted into index] = %.0f",
            result->heap_tuples, result->index_tuples)));
        /* DBLAB ] ============================================================ */

    PG_RETURN_POINTER(result);
}
```

2. 在函数_bt_leafbuild 中添加代码

在函数_bt_load 执行完之后，输出为构建索引而分配的页面数量以及写出磁盘的页面

添加代码：分析btree索引构建过程

向 users 表中插入 100 行随机生成的元组。

```
SELECT setseed(0);
INSERT INTO users VALUES (
    to_char(generate_series(100, 199)),
    to_char(random() * 100000000, '09999999'),
    gen_hanzi(3),
    to_char(10000000000 + random() * 10000000000, '09999999999'),
    CURRENT_DATE + floor((random() * 15))::int);
```

在 users 表上对于 u_name 列构建 btree 索引。

```
CREATE INDEX usersidx ON users(u_name);
```

gsql 输出为：

```
INFO:  _bt_buildadd: [rel] blocknum =    0, offsetnum =  97 | [index] key = 侔鏕埜, level =  0, blocknum =    1
INFO:  _bt_buildadd: [rel] blocknum =    0, offsetnum =  55 | [index] key = 倩瓏簨, level =  0, blocknum =    1
... 省略若干行
INFO:  _bt_buildadd: [rel] blocknum =    0, offsetnum =  25 | [index] key = 霵嫞蛋, level =  0, blocknum =    1
INFO:  _bt_leafbuild: [# pages allocated] = 2 [# pages written out] = 2
INFO:  btbuild: [# of tuples seen in the relation] = 100 [# of tuples inserted into index] = 100
CREATE INDEX
```

可以看到，gsql 输出的这些信息正是我们添加的代码而输出的相关信息。请结合 gsql 中的全部输出信息，在源代码中分析这些输出与 btree 索引构建流程的关系。

7. 使用 pageinspect 插件：函数 bt_page_items。

函数 bt_page_items 返回指定 btree 索引的指定页面上索引项 (item) 的详细信息。

```
railway=# SELECT * FROM bt_page_items('usersidx', 1);
itemoffset |  ctid   | itemlen | nulls | vars |                    data
-----------+---------+---------+-------+------+-------------------------------------------------
         1 | (9,18)  |      24 | f     | t    | 15 e6 89 be e5 9f bc e5 8e 89 00 00 00 00 00 00
         2 | (5,100) |      24 | f     | t    | 15 e4 b8 8d e7 a8 a1 e5 95 8f 00 00 00 00 00 00
         3 | (4,44)  |      24 | f     | t    | 15 e4 b8 9e e8 8b ac e7 a3 a4 00 00 00 00 00 00
         4 | (8,45)  |      24 | f     | t    | 15 e4 b8 ae e5 83 8c e7 8e b1 00 00 00 00 00 00
         5 | (4,38)  |      24 | f     | t    | 15 e4 b8 af e7 af 99 e7 ad b4 00 00 00 00 00 00
         6 | (3,54)  |      24 | f     | t    | 15 e4 b9 89 e8 af ab e7 b1 a6 00 00 00 00 00 00
         7 | (6,3)   |      24 | f     | t    | 15 e4 b9 8f e5 86 97 e8 8d 9e 00 00 00 00 00 00
         8 | (2,48)  |      24 | f     | t    | 15 e4 ba 88 e7 9a 97 e9 8f 99 00 00 00 00 00 00
         9 | (3,2)   |      24 | f     | t    | 15 e4 ba 88 e9 b4 b1 e6 b7 ba 00 00 00 00 00 00
        10 | (8,40)  |      24 | f     | t    | 15 e4 ba 9e e9 9a a5 e6 9d a2 00 00 00 00 00 00
...
       253 | (6,48)  |      24 | f     | t    | 15 e6 87 90 e4 be a3 e6 97 b6 00 00 00 00 00 00
       254 | (7,105) |      24 | f     | t    | 15 e6 87 92 e9 b4 b9 e8 b2 a2 00 00 00 00 00 00
       255 | (6,97)  |      24 | f     | t    | 15 e6 87 aa e7 b1 85 e6 a7 85 00 00 00 00 00 00
       256 | (8,79)  |      24 | f     | t    | 15 e6 87 ac e7 8a a0 e8 ae 9a 00 00 00 00 00 00
       257 | (4,95)  |      24 | f     | t    | 15 e6 87 ac e9 ac ad e6 a9 a2 00 00 00 00 00 00
       258 | (8,11)  |      24 | f     | t    | 15 e6 88 9c e9 aa b1 e8 92 87 00 00 00 00 00 00
       259 | (6,101) |      24 | f     | t    | 15 e6 89 86 e5 98 a3 e9 89 a7 00 00 00 00 00 00
```

8.1　实验介绍

日志与恢复是 openGauss 数据库实现事务处理和确保 ACID 特性的重要组成部分。本实验尝试打通目前数据库日志与恢复模块在原理学习与系统实现上的鸿沟。通过 openGauss 数据库中日志与恢复部分的实现源代码，分析与验证相关原理与机制，包括：WAL 日志文件、XLOG 日志记录、日志写入过程、检查点机制、数据库恢复、数据库备份与 PITR 恢复。

首先，通过实验查看 WAL 日志文件的基本信息与命名方式；然后，通过“立即”关闭模式验证数据库在重启时的恢复过程；通过添加代码的方法，在数据库恢复过程中输出调试信息，结合源代码阅读，更加详细地分析 WAL 日志恢复过程；最后，验证数据库备份与 PITR 恢复机制。

日志与恢复机制是数据库系统最为繁杂的功能模块之一，数据库日志与恢复功能的实现需要考虑众多原理中忽略的细节，大量涉及较为底层的系统机制。本实验的实践内容包括较多的源代码阅读与分析，具有较大的挑战性。

8.2　实验目的

1. 理解 WAL 日志文件的工作原理。
2. 理解 XLOG 日志记录的组织。
3. 理解 WAL 日志写入过程。
4. 理解 WAL 日志检查点机制。
5. 理解利用 WAL 日志重做 XLOG 记录进行数据库恢复的原理。
6. 掌握数据库备份与 PITR 恢复方法。
7. 了解与本实验相关的函数与结构体的源代码。

WAL日志与数据库恢复

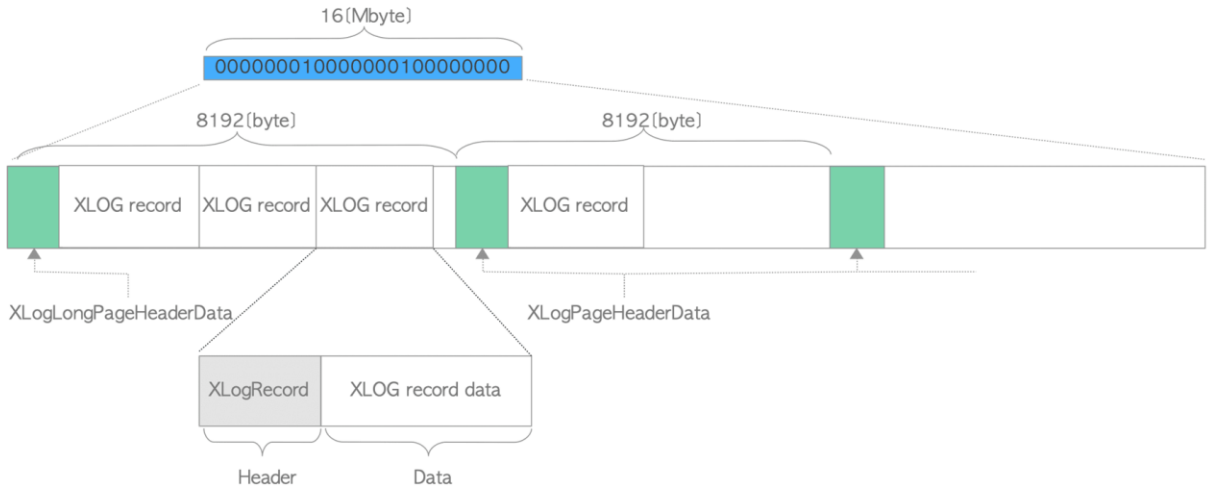


图 7.7 WAL 段文件的内部结构

3. 结构体 XLogPageHeaderData

【源码】src/include/access/xlog_basic.h

```
typedef struct XLogPageHeaderData {
        uint16 xlp_magic;       /* magic value for correctness checks */
        uint16 xlp_info;        /* flag bits, see below */
        TimeLineID xlp_tli;     /* TimeLineID of first record on page */
        XLogRecPtr xlp_pageaddr; /* XLOG address of this page */

        /*
         * When there is not enough space on current page for whole record, we
         * continue on the next page.  xlp_rem_len is the number of bytes
```

▪ 8.4.3 分析代码：数据库恢复过程

openGauss 数据库发生非正常停止后，重启时会进入到恢复模式。这时，会自动重做 XLOG 日志记录。恢复过程调用的主要函数是 StartupXLOG，该函数非常长，我们只关注关键步骤。

【源码】src/gausskernel/storage/access/transam/xlog.cpp

```
/*
 * This must be called ONCE during postmaster or standalone-backend startup
 */
void StartupXLOG(void)
{
    XLogCtlInsert *Insert = NULL;
    CheckPoint checkPoint;
    CheckPointNew checkPointNew;    /* to adapt update and not to modify the storage format */
    CheckPointPlus checkPointPlus; /* to adapt update and not to modify the storage format for
global clean */
    CheckPointUndo checkPointUndo;
    uint32 recordLen = 0;
    bool wasShutdown = false;
    bool DBStateShutdown = false;
    bool reachedStopPoint = false;

    bool haveBackupLabel = false;
    bool haveTblspcMap = false;
    XLogRecPtr RecPtr, checkPointLoc, EndOfLog;
    XLogSegNo endLogSegNo;
    XLogRecord *record = NULL;
... 省略
```

在数据库恢复过程中输出信息

▪ 8.4.4 添加代码：在数据库恢复过程中输出信息

在本节中，我们进行代码实验，通过注释掉 xlog.cpp 文件中的一些条件编译宏变量 WAL_DEBUG 和代码，打开一些日志调试信息输出代码。

1. 注释掉条件编译，打开函数 xlog_outrec

   注释掉两处条件编译 WAL_DEBUG，打开函数 xlog_outrec。该函数用于输出一条 XLOG 记录的相关信息。

【源码】src/gausskernel/storage/access/transam/xlog.cpp

```
/* [ DBLAB ============================================================ */
// #ifdef WAL_DEBUG
/* DBLAB ] ============================================================ */
static void xlog_outrec(StringInfo buf, XLogReaderState *record);
/* [ DBLAB ============================================================ */
// #endif
/* DBLAB ] ============================================================ */
```

【源码】src/gausskernel/storage/access/transam/xlog.cpp

```
/* [ DBLAB ============================================================ */
// #ifdef WAL_DEBUG
/* DBLAB ] ============================================================ */

static void xlog_outrec(StringInfo buf, XLogReaderState *record)
{
    int block_id;

     appendStringInfo(buf, "prev %X/%X; xid %u", (uint32)(XLogRecGetPrev(record) >> 32),
(uint32)XLogRecGetPrev(record),
                    XLogRecGetXid(record));
```

如上述实验步骤无误，将产生以下日志输出（具体时间不同）。请对照 8.4.3 节中的代码，分析这里的日志输出。

```
...省略
LOG:  database system timeline: 66
LOG:  database system was interrupted; last known up at 2023-02-23 20:33:35 CST
...省略
LOG:  redo record is at 0/162B85E8; shutdown FALSE
...省略
LOG:  database system was not properly shut down; automatic recovery in progress
LOG:         StartupXLOG     PrintCkpXctlControlFile:     [checkPoint]     oldCkpLoc:0/162B8668,
oldRedo:0/162B85E8, newCkpLoc:0/162B8668, newRedo:0/162B85E8, preCkpLoc:0/162B6CD8
...省略
LOG:  redo starts at 0/162B85E8
...省略
...以下是通过本节实验操作所产生的重做 XLOG 记录的输出
LOG:  REDO @ 0/162B85E8; LSN 0/162B8618: prev 0/162B8408; xid 0; len 8 -  XLOG_STANDBY_CSN
LOG:  REDO @ 0/162B8618; LSN 0/162B8668: prev 0/162B85E8; xid 0; len 40 -  XLOG_RUNNING_XACTS
LOG:  REDO @ 0/162B8668; LSN 0/162B8708: prev 0/162B8618; xid 0; len 120 - checkpoint: redo
0/162B85E8; len 120; next_csn 2575; recent_global_xmin 34896; tli 1; fpw false; xid 34897; oid
90272; multi 2; offset 0; oldest xid 12772 in DB 15552; oldest running xid 34897; oldest xid
with epoch having undo 0; online at Thu Feb 23 20:33:35 2023; remove_seg 0/6
LOG:  REDO @ 0/162B8708; LSN 0/162B8760: prev 0/162B8668; xid 34897; len 11; blkref #0: rel
1663/57357/82080, blk 0 - XLOG_HEAP_INSERT insert(init): off 34894
LOG:    REDO  @  0/162B8760;  LSN  0/162B87A0:  prev  0/162B8708;  xid  34897;  len  24  -
XLOG_STANDBY_CSN_COMMITTING, xid 34897, csn 2575
```

```
LOG:    REDO  @  0/162B87A0;  LSN  0/162B87E8:  prev  0/162B8760;  xid  34897;  len  32  -
XLOG_XACT_COMMIT_COMPACT commit: 2023-02-23 20:33:45.17921+08; csn:2575; RecentXmin:34897
...通过本节实验操作所产生的重做 XLOG 记录的输出结束
```

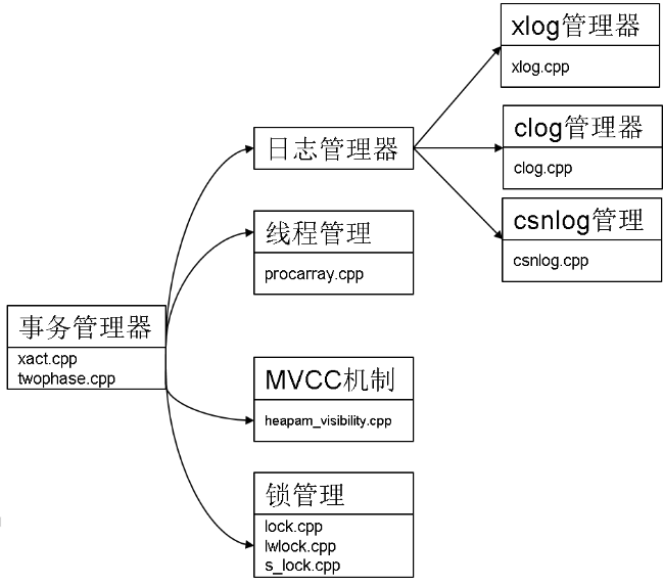
9.1　实验介绍

与日志和恢复机制相同，并发控制与锁机制是 openGauss 数据库实现事务处理 ACID 特性的另一重要部分。目前数据库原理教学中对于并发控制与锁机制模块缺乏行之有效的实践教学手段，部分原因归咎于并发控制与锁机制在数据库管理系统中涉及到的相关代码复杂繁复。本实验旨在借助 openGauss 的开源代码，一方面对于并发控制与锁机制的原理进行落地的代码工程实践，另一方面最大限度地减小源代码繁复程度的影响。本章实验原理主要包括 openGauss 的事务处理和锁机制。首先，通过实验查看 openGauss 数据库中表上的加锁信息；然后，通过复现的方式分别验证 Share 锁、Access Share 锁、Row Exclusive 锁和 Access Exclusive 锁。

并发控制与锁机制也是数据库管理系统实现中最为复杂和精妙部分之一。与日志和恢复部分类似，会涉及到若干系统层面的实现代码。希望通过本实验的实践内容，提升对于并发控制与锁机制的理解程度，更好地认识系统软件在工程实现方面的复杂性。

9.2　实验目的

1. 理解 openGauss 事务处理原理。
2. 了解 openGauss 事务处理模块的实现机制。
3. 理解 openGauss 锁机制原理。
4. 了解 openGauss 锁机制的若干实现过程。
5. 掌握 openGauss 中 Share 锁的复现方法。
6. 掌握 openGauss 中 Access Share 锁的复现方法。
7. 掌握 openGauss 中 Row Exclusive 锁的复现方法。
8. 掌握 openGauss 中 Access Exclusive 锁的复现方法。
9. 了解与本实验相关的函数与结构体的源代码。



锁机制的复现

▪ 9.4.3 复现 Access Share 锁

从实验原理中了解到，普通的查询命令都会添加 Access Share 锁.

1. 生成锁

```
BEGIN;
SELECT * FROM users WHERE u_id='100';
SELECT locktype,database,relation,pid,mode FROM pg_locks WHERE pid= 139628707247872;
```

可以看到，在 users 表上添加了 Access Share 锁。由于 Access Share 锁与 Access exclusive 锁模式冲突，所以此时无法进行 ALTER 或 DROP 等表级操作。但是可以执行 SELECT、INSERT、UPDATE 等操作。执行结果如下：

```
openGauss=# Begin;
Begin

openGauss=# SELECT * FROM users WHERE u_id="100';
 u_id  |   u_passwd |  u_name |    u_idnum    |       u_regtime
-------+------------+---------+--------------+----------------------------
  100  |   27371846 |   12400 |   18015805623|   2022-05-18 00:00:00
(1 row)
openGauss=# SELECT locktype,database,relation,pid,mode FROM pg_ locks WHERE pid= 139628707247872;
locktype   |  database |   relation   |     pid          |     mode
-----------+-----------+-------------+-----------------+--------------------
relation   |     14553 |       12009 | 139628707247872 | AccessShareLock
relation   |     14553 |       24587 | 139628707247872 | AccessShareLock
relation   |     14553 |       24585 | 139628707247872 | AccessShareLock
relation   |     14553 |       24582 | 139628707247872 | AccessShareLock
virtualxid |           |             | 139628707247872 | ExclusiveLock
(5 rows)
```

2. 进行 SELECT 操作

打开另一个会话窗口，进行 SELECT 操作，顺利执行完成。执行结果如下

```
openGauss=# SELECT * FROM users WHERE u_id = '120';
 u_id  |   u_passwd |  u_name |    u_idnum    |       u_regtime
-------+------------+---------+--------------+----------------------
   120 |   88119327 |  17967  | 19510355243  |   2022-05-30 00:00:00
 (1 row)
```

3. 进行 ALTER 操作

打开另一个会话窗口，进行 ALTER 操作，发现该命令**被堵塞**，无法执行完成。

执行结果如下：

```
openGauss=# ALTER TABLE users ADD(address varchar(10));
WARNING: Session unused timeout
FATAL: terminating connection due to administrator command
Could not send data to server: Broken pipe
The connection to the server was lost. Attempting reset: Succeeded.
openGauss=# ALTER TABLE users ADD(address warchar(10));
```

4. 提交事务，释放锁

在第一个会话中提交事务，发现第三个会话中的 ALTER 成功执行。执行结果如下：

```
openGauss=# COMMIT;
COMMIT
openGauss=# ALTER TABLE users ADD(address warchar(10));
ALTER TABLE
```

10.1 实验介绍

本实验将综合数据库课程所学知识以及前 9 章实验内容在 openGauss 数据库上进行实践。首先给定用户需求构建数据库，围绕该关系模型设计 ER 图，以具体任务需要学习运用 SQL 语言增删改查等基本操作。紧接着在所建数据库上建立触发器、索引以及创建存储过程，学习如何创建和使用数据库中的各种对象。本实验首次使用 Data Studio 客户端，它是华为推出的一款数据分析和可视化工具，其提供了丰富的图形、图表和仪表板等可视化元素，使用户可以方便地分析、探索和呈现数据。最后，使用 JDBC 连接数据库进行增删改查操作，熟悉并掌握 JDBC 编程过程。

本实验内容综合性较强，在实践过程中可能会遇到操作失败或结果不符合预期的情况。通过正确理解失败原因，能够有效提高对 SQL 语言和其特性的理解，增加数据库使用经验。

10.2 实验目的

1. 复习并巩固关系数据模型及其基本概念。
2. 巩固关系数据库语言 SQL。
3. 掌握并练习关系数据库设计方法。
4. 掌握关系数据库编程技术。

SQL综合实验与Data Studio工具使用

10.4 实验步骤

10.4.1 构建数据库

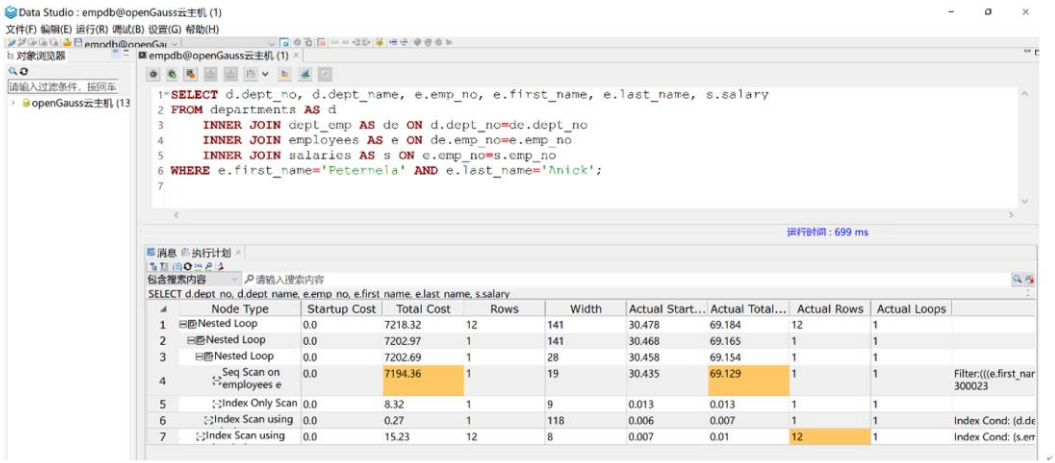
需要构建的数据库信息如下：

- 某公司为管理员工相关数据需要设计名为 empdb 的数据库。该数据库中要管理的数据包括：员工数据（employees）、职称数据（titles）、工资数据（salaries）、部门数据（departments）等。
- 员工数据包括：员工编号（emp_no）、出生日期（birth_date）、名字（first_name）、姓氏（last_name）、性别（gender）、入职日期（hire_date）。
- 职称数据包括：职称名称（title）、起始时间（from_date）、终止时间（to_date）。一条职称数据记录了某员工从起始时间到终止时间这个时间段内的职称名称。
- 工资数据包括：工资数额（salary）、起始时间（from_date）、终止时间（to_date）。一条工资数据记录了某员工从起始时间到终止时间这个时间段内的工资数额。
- 部门数据包括：部门编号（dept_no）、部门名称（dept_name）。
- 部门和员工间的关系 1（dept_emp）：一个部门下属有多名员工，一名员工可隶属于多个部门。需要记录某员工为某部门工作的起始时间和终止时间。
- 部门和员工间的关系 2（dept_manager）一个部门有多位经理（不用区分正副职），经理也是一名员工，一名员工可同时担任多个部门的经理。需要记录某员工担任某部门经理的起始时间和终止时间。

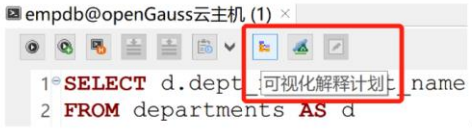
1. 分析用户需求，画出 Employees 数据库的 E/R 模型图。

2. 将 E/R 模型转换为关系模型，用 SQL 创建关系表，写出 CREATE TABLE 语句。
（连接 openGauss、创建数据库的命令见 empdb_TODO.sql）

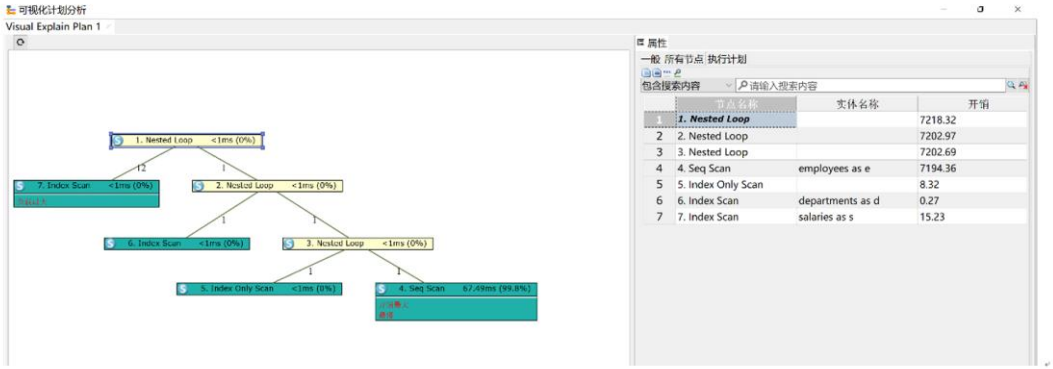
查询将被实际执行，得到包括准确开销代价的查询执行计划，如下图所示：



单击“可视化查询计划”工具按钮，



得到如下图所示的查询执行计划树图示：



适用于教学的高铁数据库h-railway

面向课程思政：设计适用于教学的**高铁数据库h-railway**

课程思政与事务型数据库结合

```
-- ------------------
-- 高铁数据库
-- ------------------

CREATE DATABASE railwaydb;

-- ------------------
-- 用户
-- ------------------

CREATE TABLE users
(
   u_id varchar(20),              -- 用户id，用于系统登录账户名
   u_passwd varchar(20),          -- 密码，用于系统登录密码
   u_name varchar(10),            -- 真实姓名
   u_idtype smallint,             -- 证件类型
   u_idnum varchar(20),           -- 证件号码
   u_tel varchar(11),             -- 手机号码
   u_regtime timestamp,           -- 注册时间
   CONSTRAINT pk_users PRIMARY KEY (u_id)
);
```

```
-- ------------------
-- 订单
-- ------------------

CREATE TABLE orders
(
  o_id int,                       -- 订单id（主键）
  o_uid varchar(20),              -- 用户id（外键：参照 users(u_id)）
  o_tdate date,                   -- 发车日期
  o_tid varchar(10),              -- 车次（外键：参照train(t_id)）
  o_sstation varchar(20),         -- 上车站（外键：参照station(s_name)）
  o_estation varchar(20),         -- 下车站（外键：参照station(s_name)）
  o_seattype smallint,            -- 座位类型：一等1、二等2
  o_carriage smallint,            -- 车厢号
  o_seatnum smallint,             -- 座位号（排）
  o_seatloc char(1),              -- 座位位置：ABCEF
  o_price money,                  -- 订单金额
  o_ispaid boolean,               -- 是否已支付
  o_ctime timestamp,              -- 订单创建时间
  CONSTRAINT pk_orders PRIMARY KEY (o_id)
);
```

```
-- ------------------
-- 列车停靠车站
-- ------------------

CREATE TABLE trainstop
(
  ts_tid varchar(10),                    -- 车次id（外
  ts_sname varchar(20),                  -- 车站名称（
  ts_atime time,                         -- 到达时间
  ts_dtime time,                         -- 出发时间
  CONSTRAINT pk_trainstop PRIMARY KEY (ts_tid, ts_
);
```

```
-- ------------------
-- 车站
-- ------------------

CREATE TABLE station
(
  s_name varchar(20),             -- 车站名称（主键）
  s_city varchar(20),             -- 车站所在城市
  CONSTRAINT pk_station PRIMARY KEY (s_name)
);

INSERT INTO station VALUES('北京南','北京市');
INSERT INTO station VALUES('天津','天津市');
INSERT INTO station VALUES('重庆西','重庆市');
INSERT INTO station VALUES('长沙南','长沙市');
INSERT INTO station VALUES('天津西','天津市');
INSERT INTO station VALUES('天津南','天津市');
INSERT INTO station VALUES('福州','福州市');
INSERT INTO station VALUES('沧州西','沧州市');
INSERT INTO station VALUES('郑州东','郑州市');
```

```
-- ------------------
-- 列车车次
-- ------------------

CREATE TABLE train
(
  t_id varchar(10),               -- 车次id（主键）
  t_dstation varchar(20),         -- 始发站（外键：参照station(s_name)）
  t_astation varchar(20),         -- 到达站（外键：参照station(s_name)）
  t_dtime time,                   -- 出发时间
  t_atime time,                   -- 到达时间
  CONSTRAINT pk_train PRIMARY KEY (t_id)
);

INSERT INTO train VALUES('G321','北京南','厦门北','2022-04-29 08:47','2022-04-29 19:58');
INSERT INTO train VALUES('G2608','天津西','北京南','2022-04-30 05:42','2022-04-30 06:22');
INSERT INTO train VALUES('C2002','天津','北京南','2022-04-29 05:58','2022-04-29 06:28');
INSERT INTO train VALUES('G1709','天津西','重庆西','2022-04-29 08:05','2022-04-29 19:54');
INSERT INTO train VALUES('G305','天津西','香港西九龙','2022-04-29 10:57','2022-04-29 21:07');
```

```
-- ------------------
-- 车次运行
-- ------------------

CREATE TABLE trainrun
(
  tr_date date,                          -- 车次日期
  tr_tid  varchar(10),                   -- 车次id（
  tr_seat1 smallint,                     -- 剩余座位
  tr_seat2 smallint,                     -- 剩余座位
  CONSTRAINT pk_trainrun PRIMARY KEY (tr_date,
);
```

◆ 数据库原理知识覆盖

- **进行基于openGauss数据库的《数据库原理》课程实验设计**

> 系统软件开发环境：实验1

> 系统软件开发环境、数据库基本概念：实验2 —> 数据库基本概念

> 数据库存储：实验4、实验5 —> 存储

> 数据库查询：实验6 —> 查询

> 数据库索引：实验7 —> 索引

> 数据库恢复：实验8 —> 恢复日志

> 数据库并发控制：实验9 —> 并发控制

> 数据库基本综合：实验10 —> SQL综合操作

- 涵盖《数据库原理》本科专业核心课程主要知识体系

第1章 实验1：openGauss初探.docx

第2章 实验2：openGauss编译与安装.docx

第3章 实验3：openGauss调试.docx

第4章 实验4：表的创建与系统表.docx

第5章 实验5：表的页面存储结构.docx

第6章 实验6：查询执行：嵌套循环连接.docx

第7章 实验7：索引的构建与使用.docx

第8章 实验8：日志与恢复.docx

第9章 实验9：并发控制与锁.docx

第10章 实验10：SQL综合实验.docx

附录：实验7-1000行元组索引构建记录.docx

◆ 系统软件人才培养思考

- 提升系统软件教育师资队伍水平
- 以开源教育理念改造学生学习模式
- 建设面向系统软件能力培养的课程与教材体系
- 面向系统软件的社区生态文化培养

谢谢！

敬请批评指正！

THANKS