



openGauss的AI特性 实验指导书

2022秋

华为技术有限公司



目录

1 实验环境介绍.....	4
1.1 实验介绍	4
1.1.1 关于本实验.....	4
1.1.2 读者知识背景.....	4
1.1.3 实验设备介绍	4
2 准备工作：ECS 弹性云服务器购买	5
2.1 实验介绍	5
2.1.1 关于本实验.....	5
2.1.2 实验目的	5
2.2 登录华为云	5
2.3 准备虚拟私有云 VPC 环境.....	6
2.4 购买云服务器 ECS 并登录.....	10
2.5 思考题.....	错误!未定义书签。
3 关卡一：openGauss 数据库的编译和安装.....	17
3.1 实验介绍	17
3.1.1 关于本实验.....	17
3.1.2 实验目的	17
3.2 实验任务及步骤.....	17
3.2.1 编译前准备.....	17
3.2.2 数据库安装编译.....	错误!未定义书签。
3.2.3 关卡验证	22
3.2.4 思考题.....	错误!未定义书签。
4 关卡二：openGauss 数据导入及基本操作	23
4.1 实验介绍	23
4.1.1 关于本实验.....	23
4.1.2 实验目的	23
4.2 实验任务及步骤	23



4.2.1 数据初始化	23
4.2.2 关卡验证	27
4.2.3 思考题	27
5 关卡三：openGauss 的 AI4DB 特性应用	28
5.1 实验介绍	28
5.1.1 关于本实验	28
5.1.2 实验目的	28
5.2 实验任务及步骤	28
5.2.1 将 X-Tuner 安装到系统中	28
5.2.2 使用 X-Tuner 进行参数优化	31
5.2.3 Index-advisor：索引推荐	34
5.2.4 关卡验证	38
5.2.5 思考题	错误!未定义书签。
6 关卡四：openGauss 的 DB4AI 特性应用	39
6.1 实验介绍	39
6.1.1 关于本实验	39
6.1.2 实验目的	39
6.2 实验任务及步骤	39
6.2.1 利用 DB4AI 原生 AI 引擎训练并预测模型	39
6.2.2 关卡验证	43
6.2.3 思考题	错误!未定义书签。
7 清理工作：资源释放	44
7.1 实验介绍	44
7.1.1 关于本实验	44
7.1.2 实验目的	44
7.2 删除弹性云服务器及相关资源	44
7.3 思考题	45
8 缩略语表	46

1 实验环境介绍

1.1 实验介绍

1.1.1 关于本实验

openGauss 是关系型数据库，采用客户端/服务器，单进程多线程架构，支持单机和一主多备部署方式，备机可读，支持双机高可用和读扩展。

本实验主要内容为 openEuler 弹性云服务上单机安装部署 openGauss 数据库，并进行简单的数据库相关操作；完成数据库相关操作后，进行 openGauss 的 AI4DB 的特性实验，提升数据库性能工作；最后完成 openGauss 的 DB4AI 特性实验（选做），完成分类测试和回归测试算法。

1.1.2 读者知识背景

- 具有数据库知识背景，熟悉数据库常用基本操作（查询、插入、删除、更新等）；
- 了解基本 Linux 知识，熟悉常用软件如 vi 等；
- 熟悉华为云，会申请、释放云资源，会基本操作。

1.1.3 实验设备介绍

本实验环境为华为云环境，需要购买 openEuler 弹性云服务。

- 设备介绍

为了满足 openGauss 安装部署实验需要，建议每套实验环境采用以下配置：

表1-1 实验软件配套关系

软件名称	软件版本
Linux操作系统	openEuler 20.3 LTS
工具	WinSCP, TPCH
Python	Python 3.7.X

2 准备工作：ECS 弹性云服务器购买

2.1 实验介绍

2.1.1 关于本实验

本实验通过在华为云上购买弹性云服务器 ECS，了解弹性云服务器 ECS 配置流程，掌握弹性云服务器 ECS 的连接和操作。

2.1.2 实验目的

- 掌握 ECS 弹性云服务器购买流程；
- 掌握 ECS 弹性云服务器配置参数。

2.2 登录华为云

步骤 1 进入华为云官网。

华为云官网：<https://www.huaweicloud.com/>，进入华为云官网，点击登录。



步骤 2 输入账号名和密码，点击登录。



如果还没有注册，点击免费注册，按步骤进行注册后进行登录。

2.3 准备虚拟私有云 VPC 环境

在后续创建 ECS 时，选择该步骤中所配置的 VPC、子网和安全组。

步骤 1 登录华为云账号，在虚拟私有云 VPC 页面下，点击“立即使用”。

<https://www.huaweicloud.com/product/vpc.html>



步骤 2 在网络控制台界面中，点击“创建虚拟私有云”。



步骤3 填写如下配置信息，然后点击“立即创建”。

基本信息

- 区域：华北-北京四
- 名称：vpc-myvpc
- 网段：默认

子网配置

- 可用区：可用区 1
- 名称：subnet-myvpc
- 子网网段：默认

创建虚拟私有云 ?

返回虚拟私有云列表

基本信息

* 区域

华北-北京四

不同区域的资源之间内网不互通。请选择靠近您客户的区域，可以降低网络时延，提高访问速度。

* 名称

vpc-myvpc

* 网段

192 · 168 · 0 · 0 / 16

建议使用网段: 10.0.0.0/8~24, 172.16.0.0/12~24, 192.168.0.0/16~24

标签

如果您需要使用同一标签标识多种云资源，即所有服务均可在标签输入框下拉选择同一标签，建议在TMS中创建预定义标签。[查看预定义标签](#)

标签键

标签值

您还可以添加10个标签。

子网配置

默认子网

* 可用区 ?

可用区1

可用区2

可用区3

* 名称

subnet-myvpc

* 子网网段

192 · 168 · 0 · 0 / 24

可用IP数:251
子网创建完成后，子网网段无法修改

高级配置

默认配置

自定义配置

步骤 4 返回到网络控制台，可看见 VPC 与子网已创建。

虚拟私有云 ?

服务简介

流程引导

评价

使用指南

创建虚拟私有云

通过指定属性的关键字搜索

Q

C

□

名称	IPv4网段	状态	子网个数	路由表	服务器个数	操作
vpc-myvpc	192.168.0.0/16 (主网段)	可用	0	1	0	编辑网段 删除

步骤 5 在“安全组”页面中找到 default 安全组。点击“配置规则”。



步骤 6 选择“入方向规则”页签，点击“添加规则”。



步骤 7 协议端口选择“基本协议/全部协议”，优先级填“1”，点击“确定”，完成安全组规则的添加。

添加加入方向规则
教我设置

安全组方向规则为白名单（允许），放通入方向网络流量。
安全组规则对不同规格的云服务器生效情况不同，为了避免您的安全组规则不生效，请查看[安全组规则限制](#)。

安全组 default

如您要添加多条规则，建议单击 [导入规则](#) 以进行批量导入。

优先级 ?	策略 ?	协议端口 ?	类型	源地址 ?	描述	操作
1	允许	基本协议/全部协议	IPv4	IP地址 0.0.0.0/0		复制 删除

增加1条规则

确定 取消

2.4 购买云服务器 ECS 并登录

步骤 1 登录华为云账号，在弹性云服务器 ECS 页面下，点击“立即购买”。

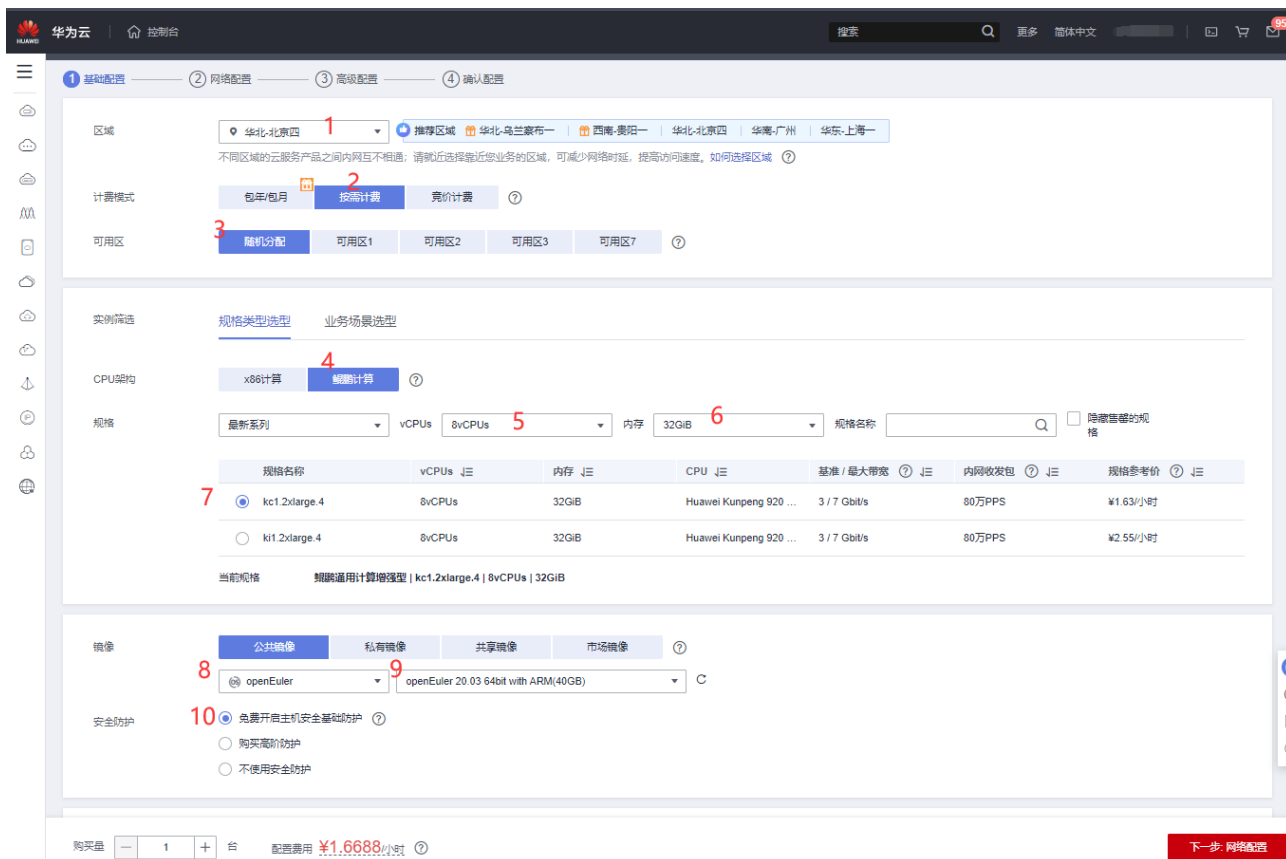
<https://www.huaweicloud.com/product/ecs.html>



步骤 2 填写如下基础配置信息，然后点击“下一步”。

- 区域：华北-北京四
- 计费模式：按需计费
- 可用区：随机分配

- CPU 架构：鲲鹏计算
- 规格：鲲鹏通用计算增强型 | kc1.2xlarge.4 | 8vCPUs | 32B
- 镜像：公共镜像 openEuler 20.03 64bit with ARM(40GB)
- 系统盘：通用型 SSD | 40



华为云 控制台

1 基础配置 2 网络配置 3 高级配置 4 确认配置

区域: 华北-北京四

计费模式: 包年/包月 按需计费 竞价计费

可用区: 随机分配 可用区1 可用区2 可用区3 可用区7

实例筛选: 规格类型选型 业务场景选型

CPU架构: x86计算 鲲鹏计算

规格: 最新系列 vCPUs: 8vCPUs 内存: 32GiB

规格名称	vCPUs	内存	CPU	基准 / 最大带宽	内网收发包	规格参考价
kc1.2xlarge.4	8vCPUs	32GiB	Huawei Kunpeng 920 ...	3 / 7 Gbit/s	80万PPS	¥1.63/小时
ki1.2xlarge.4	8vCPUs	32GiB	Huawei Kunpeng 920 ...	3 / 7 Gbit/s	80万PPS	¥2.55/小时

当前规格: 鲲鹏通用计算增强型 | kc1.2xlarge.4 | 8vCPUs | 32GiB

镜像: 公共镜像 私有镜像 共享镜像 市场镜像

openEuler 20.03 64bit with ARM(40GB)

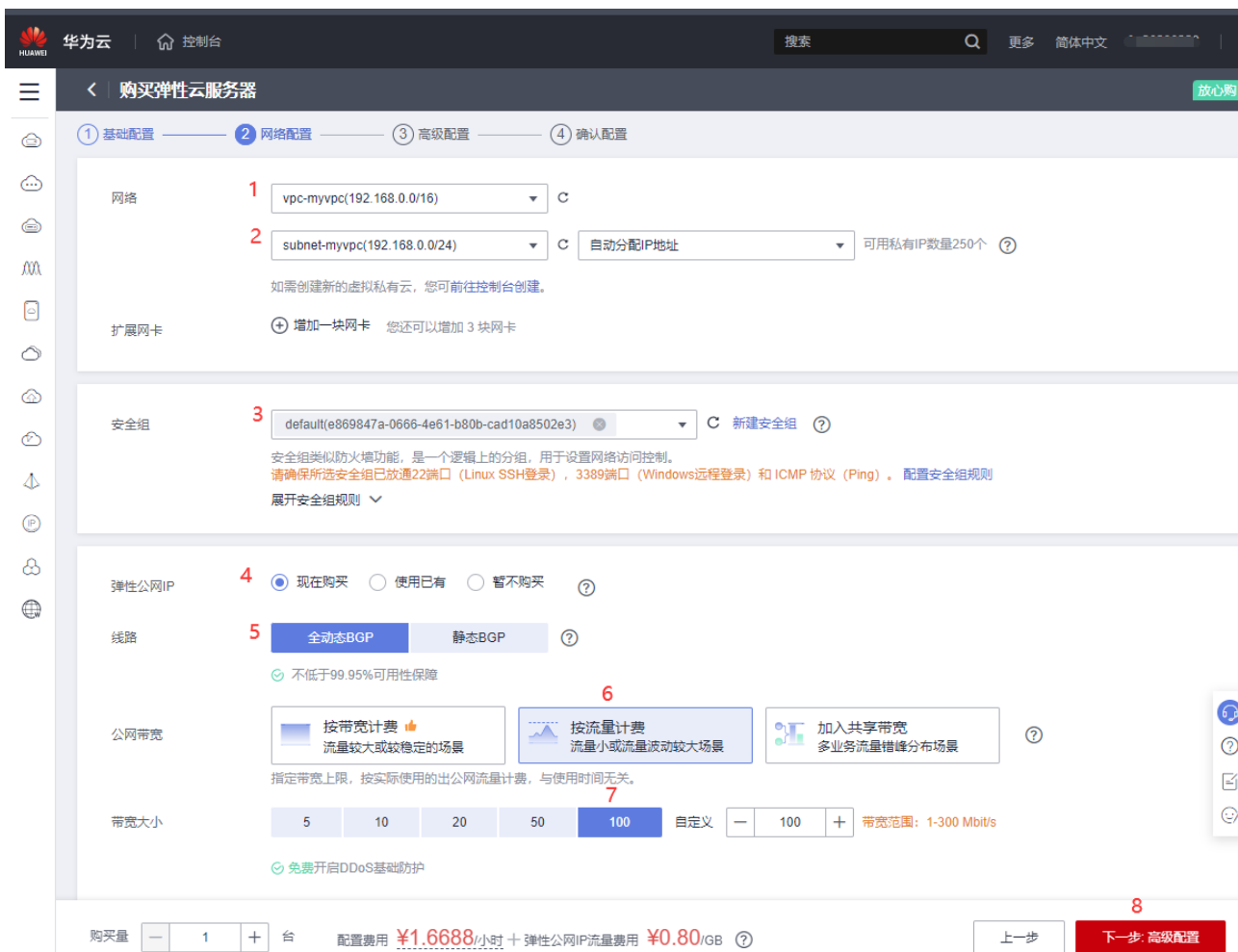
安全防护: 免费开启主机安全基础防护 购买高阶防护 不使用安全防护

购买量: 1 配置费用: ¥1.6688/小时

下一步: 网络配置

步骤3 填写如下网络配置信息，然后点击“下一步”。

- 网络：选择在 2.3 节创建的网络和子网，如 vpc-myvpc 和 subnet-myvpc
- 安全组：default
- 弹性公网 IP：现在购买
- 规格：全动态 BP
- 计费方式：按流量计费
- 带宽：100 Mbit/s



华为云 | 控制台

购买弹性云服务器

1 基础配置 2 网络配置 3 高级配置 4 确认配置

网络 1 vpc-myvpc(192.168.0.0/16) C

2 subnet-myvpc(192.168.0.0/24) C 自动分配IP地址 可用私有IP数量250个 ?

如需创建新的虚拟私有云，您可前往控制台创建。

扩展网卡 + 增加一块网卡 您还可以增加 3 块网卡

安全组 3 default(e869847a-0666-4e61-b80b-cad10a8502e3) C 新建安全组 ?

安全组类似防火墙功能，是一个逻辑上的分组，用于设置网络访问控制。
请确保所选安全组已放通22端口（Linux SSH登录），3389端口（Windows远程登录）和 ICMP 协议（Ping）。配置安全组规则
展开安全组规则

弹性公网IP 4 ☒ 现在购买 ☐ 使用已有 ☐ 暂不购买 ?

线路 5 全动态BGP 静态BGP ?

不低于99.95%可用性保障

公网带宽 6

按带宽计费 流量较大或较稳定的场景

按流量计费 流量小或流量波动较大场景

加入共享带宽 多业务流量错峰分布场景 ?

指定带宽上限，按实际使用的出公网流量计费，与使用时间无关。

带宽大小 7

5 10 20 50 100 自定义 100 + 带宽范围：1-300 Mbit/s

免费开启DDoS基础防护

购买量 - 1 + 台 配置费用 ¥1.6688/小时 + 弹性公网IP流量费用 ¥0.80/GB ?

上一步 下一步: 高级配置 8

步骤 4 填写如下高级配置信息，然后点击“下一步”。

- 云服务器名称：opengauss01（服务器名称建议不要使用下划线）
- 登录凭证：密码
- 设置密码【请记住密码，后面登录要用到】
- 云备份：暂不购买

华为云

控制台

搜索

更多 简体中文 fm20200220

购买弹性云服务器

放心购

1 基础配置

2 网络配置

3 高级配置

4 确认配置

云服务器名称

2

opengauss01

☐ 允许重名

购买多台云服务器时，支持自动增加数字后缀命名或者自定义规则命名。

登录凭证

密码

密钥对

创建后设置

用户名

root

密码

2

请牢记密码，如忘记密码可登录ECS控制台重置密码。

确认密码

3

云备份

使用云备份服务，需购买备份存储库。存储库是存放服务器产生的备份副本的容器。

现在购买

使用已有

4 暂不购买

备份可以帮助您在服务器故障时恢复数据，为了您的数据安全，强烈建议您启用备份。

云服务器组（可选）

反亲和性

—请选择云服务器组—

新建云服务器组

高级选项

☐ 现在配置

5

购买量

— 1 +

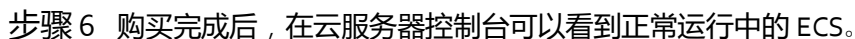
台

配置费用 ¥1.6688/小时 + 弹性公网IP流量费用 ¥0.80/GB

上一步

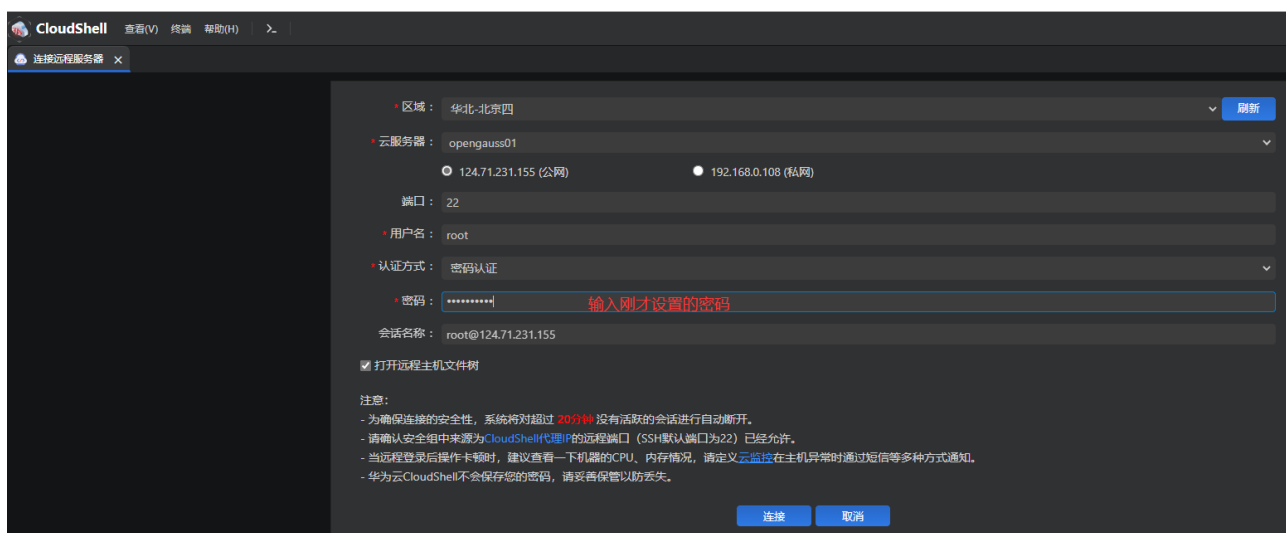
下一步: 确认配置

步骤 5 在确认配置界面，勾选“我已经阅读……”后，点击“立即购买”。

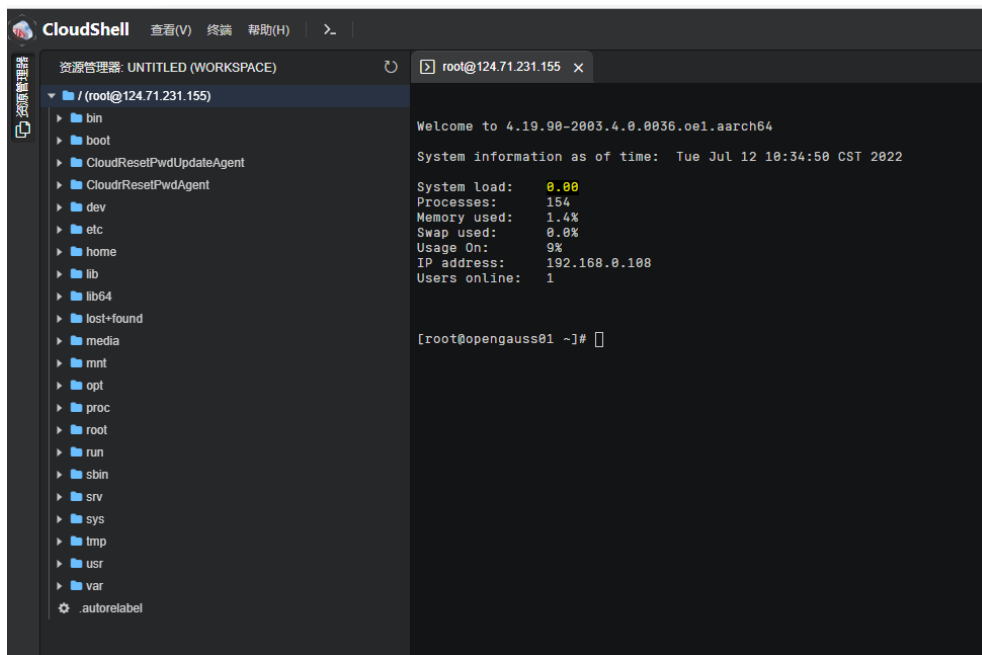




步骤 8 用 root 密码，点击“连接”。



登录成功：



The image shows a CloudShell terminal window with a file explorer on the left and a terminal on the right. The file explorer shows the root directory of a system with various folders like bin, boot, CloudResetPwdUpdateAgent, CloudResetPwdAgent, dev, etc, home, lib, lib64, lost+found, media, mnt, opt, proc, root, run, sbin, srv, sys, tmp, usr, var, and .autorelabel. The terminal window shows the following output:

```
Welcome to 4.19.90-2003.4.0.0036.oe1.aarch64

System information as of time: Tue Jul 12 10:34:50 CST 2022

System load: 0.00
Processes: 154
Memory used: 1.4%
Swap used: 0.0%
Usage On: 9%
IP address: 192.168.0.108
Users online: 1

[root@opengauss01 ~]#
```


3 关卡一：openGauss 数据库的编译和安装

3.1 实验介绍

3.1.1 关于本实验

本实验通过第三方工具，完成 openGauss 的编译和安装。

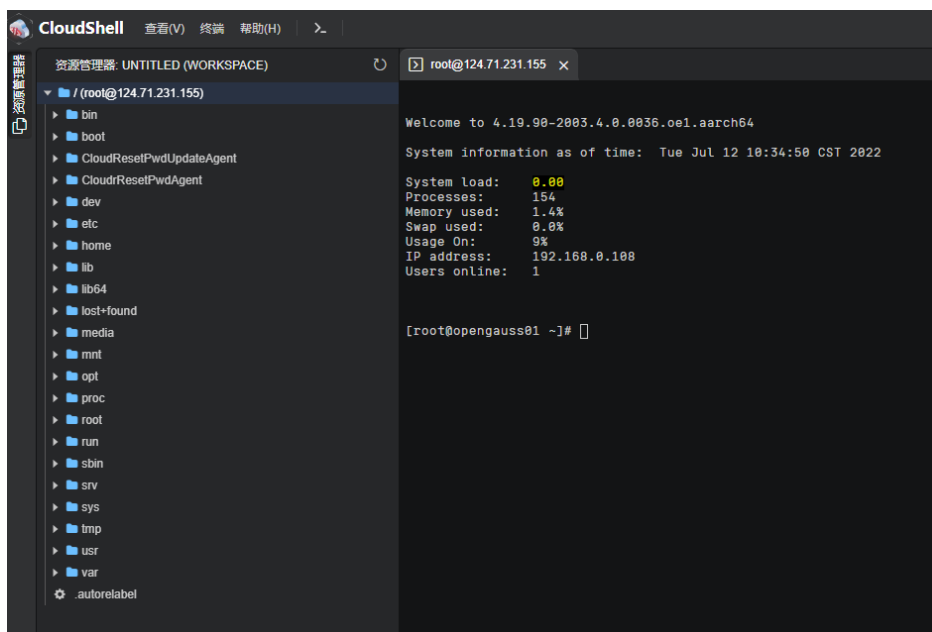
3.1.2 实验目的

- 掌握 openauss 编译和安装。
- 掌握数据库初始化。

3.2 实验任务及步骤

3.2.1 安装准备

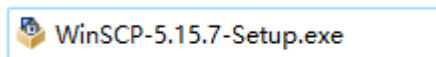
步骤 1 用 root 用户名，使用之前设置的密码登录 ECS。



步骤 2 上传安装脚本到服务器。

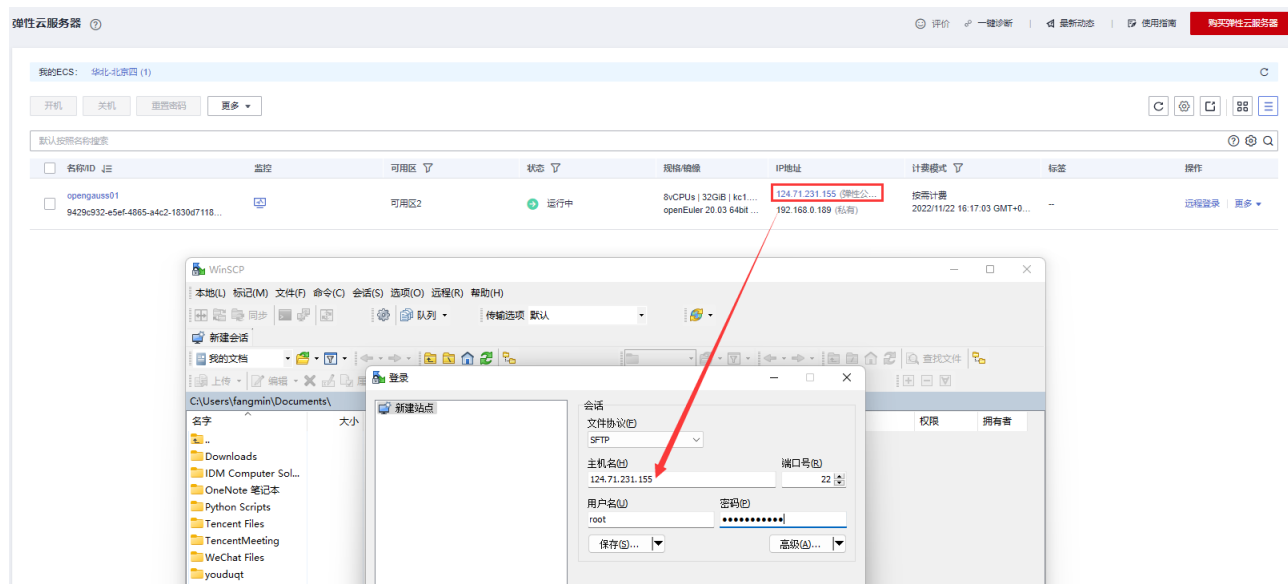
1) 安装 winscp 工具，winscp 安装包已随堂下发，默认安装即可。

* WinSCP 是一款支持 SSH (Secure SHell) 的 SCP (Secure CoPy) 文件传输软件。如果已经有类似软件可以不装。

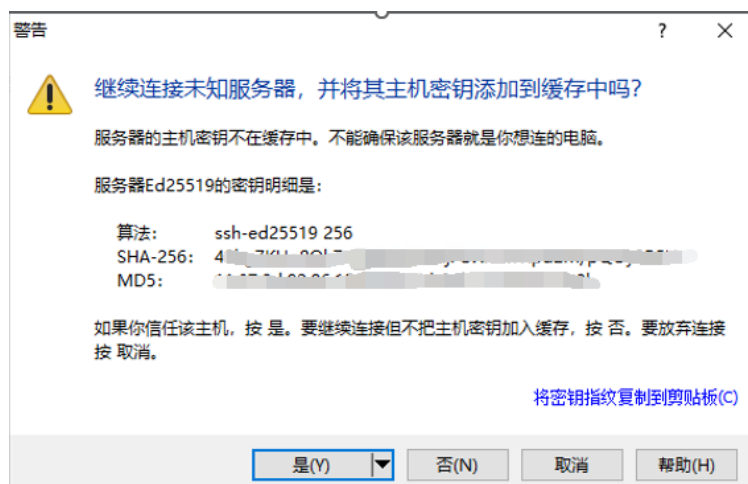


2) 配置 winscp 站点。

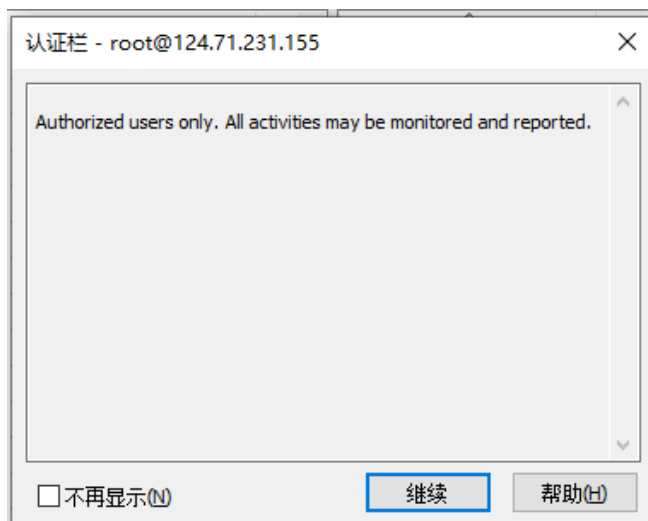
主机名从弹性云服务器列表中复制。用户名、密码就是刚才新建的弹性云服务器的用户名、密码。



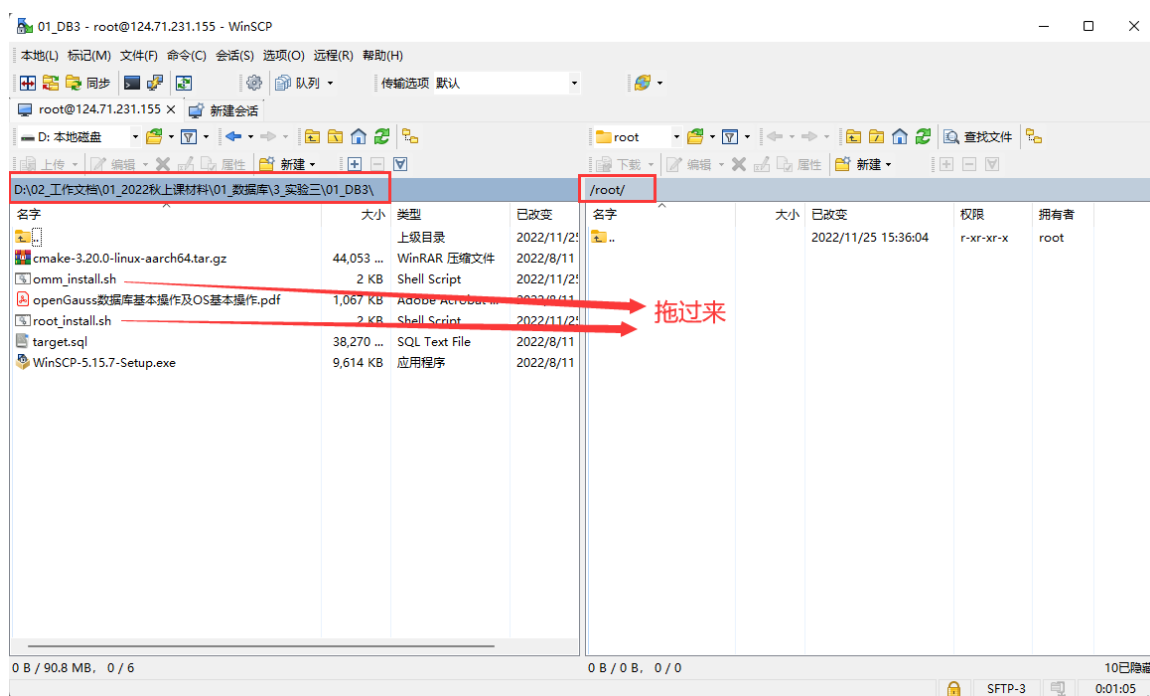
下面点击“是”：



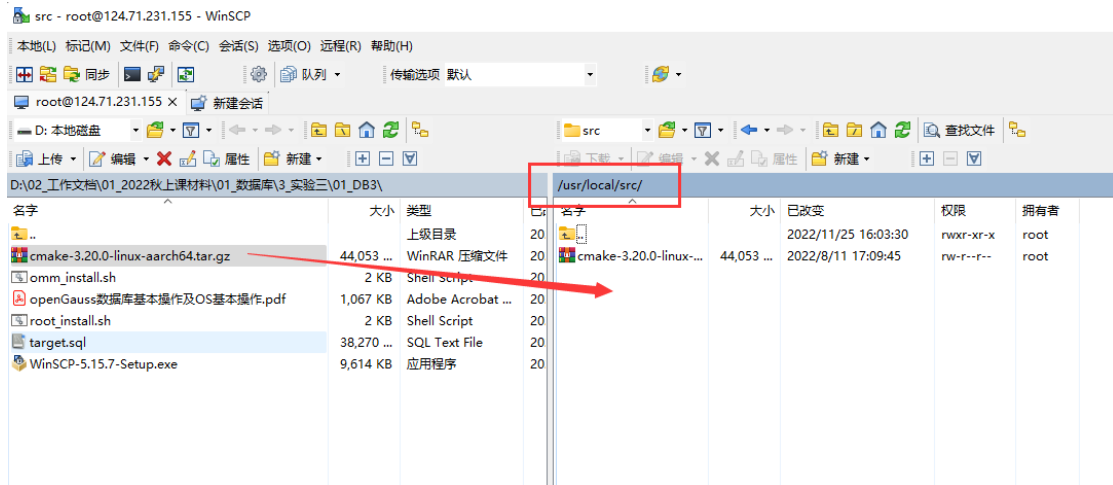
下面点击“继续”：



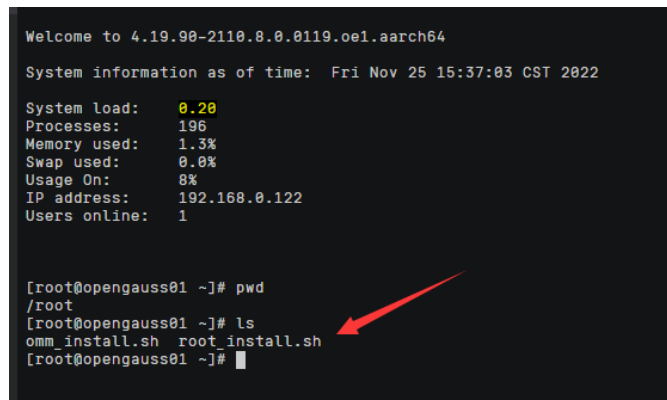
3) 将 root_install.sh 和 omm_install.sh, 上传至服务器/root/下。



4) 将 cmake-3.20.0-linux-aarch64.tar.gz 上传至服务器/usr/local/src 下。



5) 传输完毕后，可以回到 CloudShell 上看到这几个文件。



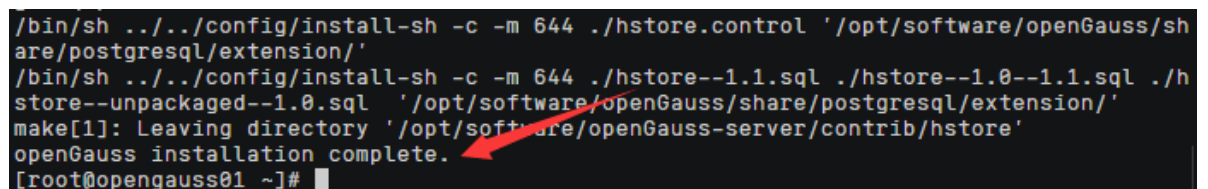
步骤 3 开始执行编译安装脚本

```
[root@opengauss01 ~]# sh root_install.sh
```

脚本运行过程中需要新建一个 omm 用户，初始化 omm 用户的密码。【请记住密码】



这里传输数据和安装需要十五分钟左右，请耐心等待。你可以利用这个时间看看后面的操作。



步骤 4 下面开始初始化数据库，先切换到 omm 用户。

```
[root@opengauss01 bin]# su - omm
```

在初始化数据库时，需要设置数据库密码，并且要使用复杂密码，如下命令。

```
gs_initdb -D $PGDATA --nodename=hostname --locale="en_US.UTF-8" -Atrust -w {password}
```

实际使用中，将{password}部分进行替换。

例如（此处只是作为举例，建议设置为复杂密码，注意密码要用英文的单引号括起来）：

```
[omm@opengauss01 ~]$ gs_initdb -D $PGDATA --nodename=opengauss01 --locale="en_US.UTF-8" -Atrust -w 'Huawei#!13'
```

显示结果如下，表示初始化成功：

```
Success. You can now start the database server of single node using:
    gaussdb -D /opt/software/openGauss/data --single_node
or
    gs_ctl start -D /opt/software/openGauss/data -Z single_node -l logfile
```

这一步如果报错，请确认一下自己当前是 omm 用户执行命令。

步骤 5 启动数据库。

```
[omm@opengauss01 ~]$ gs_ctl start -D /opt/software/openGauss/data -Z single_node -l logfile
```

```
[omm@opengauss01 ~]$ gs_ctl start -D /opt/software/openGauss/data -Z single_node -l logfile
[2022-11-25 16:27:47.455][225282][][gs_ctl]: gs_ctl started,datadir is /opt/software/openGauss/data
[2022-11-25 16:27:47.493][225282][][gs_ctl]: waiting for server to start...
.
[2022-11-25 16:27:48.505][225282][][gs_ctl]: done
[2022-11-25 16:27:48.505][225282][][gs_ctl]: server started (/opt/software/openGauss/data)
```

数据库启动成功。

步骤 6 数据库登录。

```
[omm@opengauss01 openGauss-server]$ gsql -d postgres -p 5432 -r
```

```
[omm@opengauss01 ~]$ gsql -d postgres -p 5432 -r
gsql ((GaussDB Kernel V500R002C00 build b2ff10be) compiled at 2022-11-25 16:13:13 commit 0 last mr debug)
Non-SSL connection (SSL connection is recommended when requiring high-security)
Type "help" for help.
openGauss=#
```

步骤 7 修改 omm 账号密码（可选步骤）。

```
openGauss=# ALTER USER omm identified by 'Huawei@13' replace 'Huawei#!13';
```

说明：ALTER USER omm identified by '新密码' replace '原密码';

步骤 8 查询数据库版本。

```
openGauss=# select version();
version
```

```
-----
(GaussDB Kernel V500R002C00 build b2ff10be) compiled at 2021-11-22 16:34:45 commit 0 last mr debug on
aarch64-unknown-linux-gnu, compiled by g++ (GCC) 7.3.0, 64-bit
(1 row)
```

步骤 9 退出数据库。

```
openGauss=# \q
```

3.2.2 关卡验证

步骤 1 首先需要对数据库状态进行验证，**执行结果截图粘贴至实验报告**。

```
[omm@opengauss01 ~]$ gs_ctl status
```

```
[2022-07-12 14:26:58.567][226159][][gs_ctl]: gs_ctl status,datadir is /opt/software/openGauss/data
gs_ctl: server is running (PID: 226038)
/opt/software/openGauss/bin/gaussdb "-D" "/opt/software/openGauss/data"
[omm@opengauss01 openGauss-server]$
```

步骤 2 其次，查询数据库进程，需包含数据库服务器的主机名，**执行结果截图粘贴至实验报告**。

```
[omm@opengauss01 ~]$ ps -ef|grep omm
```

```
[omm@opengauss01 ~]$ ps -ef|grep omm
root      225088    5233    0 16:24 pts/0    00:00:00 su - omm
omm       225089    225088    0 16:24 pts/0    00:00:00 -bash
omm       225285        1    1 16:27 pts/0    00:00:01 /opt/software/openGauss/bin/gaussdb -D /opt/software/openGauss/data
omm       225349    225089    0 16:30 pts/0    00:00:00 ps -ef
omm       225350    225089    0 16:30 pts/0    00:00:00 grep --color=auto omm
[omm@opengauss01 ~]$
```

4 关卡二：openGauss 数据导入及基本操作

4.1 实验介绍

4.1.1 关于本实验

本实验关卡，在安装完成的 openGauss 数据库上，进行 TPC-H 和 target 数据初始化操作，为后续实验做数据准备。

4.1.2 实验目的

- 掌握 openauss 数据库的基本维护方法；

4.2 实验任务及步骤

4.2.1 数据初始化

步骤 1 使用 root 登录，进入/opt/software 目录下。

```
[omm@opengauss01 ~]$ exit  
[root@opengauss01 ~]# cd /opt/software
```

步骤 2 下载 TPC-H 测试包。

```
[root@opengauss01 software]# git clone https://gitee.com/xzp-blog/tpch-kit.git
```

步骤 3 将 tpch-hit 目录的属组修改为 omm 用户。

```
[root@opengauss01 software]# chown omm:dbgrp -R /opt/software/tpch-kit/
```

步骤 4 切换至用户 omm，并进入测试包的目录。

```
[root@opengauss01 software]# su - omm  
[omm@opengauss01 ~]$ cd /opt/software/tpch-kit/dbgen/
```

步骤 5 生成 MakeFile 文件。

```
[omm@opengauss01 dbgen]$ make -f Makefile
```

步骤 6 连接 openGauss 数据库。

```
[omm@opengauss01 dbgen]$ gsql -d postgres -p 5432 -r
```

步骤 7 创建测试数据库 tpch 并退出。

```
openGauss=# CREATE DATABASE tpch;
openGauss=# \q
```

步骤 8 执行创建对象脚本。

```
[omm@opengauss01 dbgen]$ gsql tpch -f dss.ddl
```

步骤 9 产生测试数据。

```
[omm@opengauss01 dbgen]$ ./dbgen -vf -s 1
```

当返回如下内容，表示数据生产完成。

```
[omm@opengauss01 dbgen]$ ./dbgen -vf -s 1
TPC-H Population Generator (Version 2.17.3)
Copyright Transaction Processing Performance Council 1994 - 2010
Generating data for suppliers table/
Preloading text ... 100%
done.
Generating data for customers tabledone.
Generating data for orders/lineitem tablesdone.
Generating data for part/partsupplier tablesdone.
Generating data for nation tabledone.
Generating data for region tabledone.
[omm@opengauss01 dbgen]$
```

使用 ls 命令查看已经产生的数据：

```
[omm@opengauss01 dbgen]$ ls -l *.tbl
```

回显生成的数据。

```
[omm@opengauss01 dbgen]$ ls -l *.tbl
-rw-rw-r-- 1 omm dbgrp 24196144 Jul 12 15:54 customer.tbl
-rw-rw-r-- 1 omm dbgrp 753862072 Jul 12 15:54 lineitem.tbl
-rw-rw-r-- 1 omm dbgrp 2199 Jul 12 15:54 nation.tbl
-rw-rw-r-- 1 omm dbgrp 170452161 Jul 12 15:54 orders.tbl
-rw-rw-r-- 1 omm dbgrp 118184616 Jul 12 15:54 partsupp.tbl
-rw-rw-r-- 1 omm dbgrp 23935125 Jul 12 15:54 part.tbl
-rw-rw-r-- 1 omm dbgrp 384 Jul 12 15:54 region.tbl
-rw-rw-r-- 1 omm dbgrp 1399184 Jul 12 15:54 supplier.tbl
```

步骤 10 编辑加载脚本 LoadData.sh。

```
[omm@opengauss01 dbgen]$ vi LoadData.sh
```

将以下内容添加进 LoadData.sh 脚本中，然后保存并退出编辑。

```
for i in `ls *.tbl`; do
    table=${i%.tbl}
    echo "Loading $table..."
```



```
sed 's|/|/' $i > /tmp/$i
gsqll tpch -q -c "TRUNCATE $table"
gsqll tpch -c "\\copy $table FROM '/tmp/$i' CSV DELIMITER '|'"
done
```

步骤 11 执行加载脚本 LoadData.sh，将数据加载进数据库中。

```
[omm@opengauss01 dbgen]$ sh LoadData.sh
```

返回结果（这里装载数据需要几分钟时间，请耐心等待）：

```
[omm@opengauss01 dbgen]$ sh LoadData.sh
Loading customer...
Loading lineitem...
Loading nation...
Loading orders...
Loading partsupp...
Loading part...
Loading region...
Loading supplier...
[omm@opengauss01 dbgen]$
```

步骤 12 登录数据库验证，将查询结果截图粘贴至实验报告。

```
[omm@opengauss01 dbgen]$ gsqll -d tpch -p 5432 -r
tpch=# select count(*) from supplier;
```

此处应有返回记录，数据量为 10000。

```
tpch=# select count(*) from supplier;
count
--我是水印
10000
(1 row)
```

步骤 13 退出数据库。

```
tpch=# \q
```

步骤 14 将相关查询拷贝进查询目录中，并进入 queries/目录中。

```
[omm@opengauss01 dbgen]$ cp dists.dss queries/
[omm@opengauss01 dbgen]$ cp qgen queries/
[omm@opengauss01 dbgen]$ cd queries/
```

步骤 15 生成查询语句。

```
[omm@opengauss01 queries]$ vi genda.sh
```

将以下内容复制到 genda.sh 中，保存并退出编辑。

```
for i in {1..22}; do
    ./qgen -d $i>$i_new.sql
    ./qgen -d $i_new | sed 's/limit -1/' | sed 's/limit 100/' | sed 's/limit 10/' | sed 's/limit 20/' | sed 's/day (3)/day/' >
    queries.sql
done
```

步骤 16 执行 genda.sh 脚本，生成查询语句

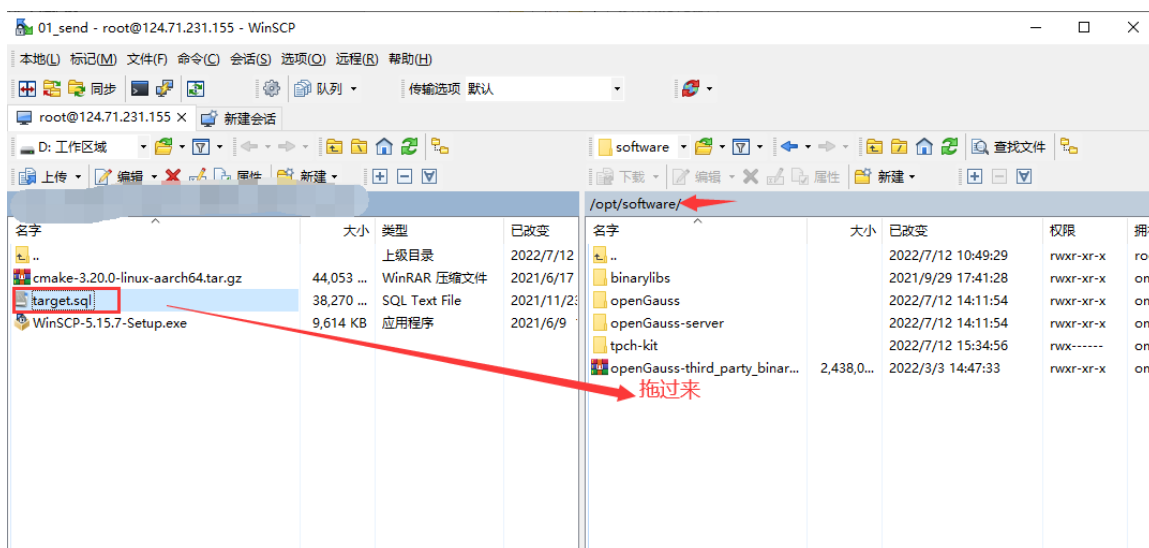
```
[omm@opengauss01 queries]$ sh genda.sh
```

步骤 17 查看生成的查询语句

```
[omm@opengauss01 queries]$ ls -l queries.sql
```

```
[omm@opengauss01 queries]$ ls -l queries.sql
-rw----- 1 omm dbgrp 11400 Jul 12 16:12 queries.sql
[omm@opengauss01 queries]$
```

步骤 18 上传 target.sql 脚本至 /opt/software 目录下（可以使用 winscp 工具传输文件，target.sql 已随堂下发），具体如下图：



步骤 19 切换至 root 用户，将脚本的属组修改为 omm。

```
[omm@opengauss01 queries]$ exit
[root@opengauss01 software]# chown omm:dbgrp /opt/software/target.sql
```

步骤 20 切回 omm 用户，将 target.sql 中的内容导入到 tpch 数据库中

```
[root@opengauss01 software]# su - omm
[omm@opengauss01 ~]$ gsql -d tpch -p 5432 -r -f /opt/software/target.sql > /opt/software/target.log
```

数据量较大，数据导入过程需要较长时间，请耐心等待。当返回如下内容，表示数据导入完成。

```
[omm@opengauss01 ~]$ gsql -d tpch -p 5432 -r -f /opt/software/target.sql > /opt/software/target.log
gsql:/opt/software/target.sql:5: NOTICE: sequence "address_dimension_address_key_seq" does not exist, skipping
gsql:/opt/software/target.sql:16: NOTICE: sequence "user_dimension_user_key_seq" does not exist, skipping
gsql:/opt/software/target.sql:27: NOTICE: table "address_dimension" does not exist, skipping
gsql:/opt/software/target.sql:8056: NOTICE: table "user_dimension" does not exist, skipping
gsql:/opt/software/target.sql:16893: NOTICE: table "date_dimension" does not exist, skipping
gsql:/opt/software/target.sql:45716: NOTICE: table "litmall_orders" does not exist, skipping
gsql:/opt/software/target.sql:145771: NOTICE: ALTER TABLE / ADD PRIMARY KEY will create implicit index "address_dimension_pkey" for table "address_dimension"
gsql:/opt/software/target.sql:145776: NOTICE: ALTER TABLE / ADD PRIMARY KEY will create implicit index "date_dimension_pkey" for table "date_dimension"
gsql:/opt/software/target.sql:145782: NOTICE: ALTER TABLE / ADD PRIMARY KEY will create implicit index "litmall_orders_pkey" for table "litmall_orders"
[omm@opengauss01 ~]$
```

步骤 21 登录数据库进行验证，**将查询结果截图粘贴至实验报告**

```
[omm@opengauss01 ~]$ gsql -d tpch -p 5432 -r
tpch=# \dt
```

步骤 11 产生的 8 张表，加上步骤 20 产生的 4 张表，总共有 12 张表。

```
tpch=# \dt
      List of relations
Schema |      Name      | Type | Owner |      Storage
-----+-----+-----+-----+-----
public | address_dimension | table | omm   | {orientation=row,compression=no}
public | customer         | table | omm   | {orientation=row,compression=no}
public | date_dimension   | table | omm   | {orientation=row,compression=no}
public | lineitem          | table | omm   | {orientation=row,compression=no}
public | litemall_orders  | table | omm   | {orientation=row,compression=no}
public | nation           | table | omm   | {orientation=row,compression=no}
public | orders           | table | omm   | {orientation=row,compression=no}
public | part             | table | omm   | {orientation=row,compression=no}
public | partsupp         | table | omm   | {orientation=row,compression=no}
public | region           | table | omm   | {orientation=row,compression=no}
public | supplier         | table | omm   | {orientation=row,compression=no}
public | user_dimension   | table | omm   | {orientation=row,compression=no}
(12 rows)
```

步骤 22 查询 customer 表的数据，**将查询结果截图粘贴至实验报告**。

```
tpch=# select * from customer limit 10;
```

步骤 23 退出数据库。

```
tpch=# \q
```

4.2.2 关卡验证

本关卡进行数据初始化，需要将步骤 12、21、22 进行**截图粘贴至实验报告**。

4.2.3 思考题

数据初始化中用到的 TPC-H，这是什么？

5 关卡三：openGauss 的 AI4DB 特性应用

5.1 实验介绍

5.1.1 关于本实验

本实验关卡，在安装完成的 openGauss 数据库上，使用 AI4DB 的功能，对数据库参数进行优化；对 TPCB 测试中的 SQL 语句进行优化。

5.1.2 实验目的

- 掌握 openauss 数据库的 AI4DB 的功能；
- 掌握 openauss 数据库的参数修改方法；
- 掌握 openauss 数据库的索引创建与使用方法。

5.2 实验任务及步骤

5.2.1 将 X-Tuner 安装到系统中

步骤 1 切换到 root 用户，安装依赖包。

```
[omm@opengauss01 ~]$ exit
[root@opengauss01 ~]# cd /root
[root@opengauss01 ~]# yum -y groupinstall "Development tools" --nogpgcheck
```

```
urw-base35-p052-fonts-20170801-11.oe1.noarch
urw-base35-z003-fonts-20170801-11.oe1.noarch
webkit2gtk3-jsc-2.22.2-6.oe1.aarch64
xorg-x11-fonts-7.5-24.oe1.noarch
urw-base35-standard-symbols-ps-fonts-20170801-11.oe1.noarch
webkit2gtk3-2.22.2-6.oe1.aarch64
woff2-1.0.2-6.oe1.aarch64
xorg-x11-server-utils-7.7-28.oe1.aarch64

Complete!
[root@opengauss01 ~]#
```

```
[root@opengauss01 ~]# yum -y install zlib-devel bzip2-devel openssl-devel ncurses-devel sqlite-devel readline-
devel tk-devel gdbm-devel libpcap-devel xz-devel --nogpgcheck
```

```
Installed:
bzip2-devel-1.0.8-3.oe1.aarch64      gdbm-devel-1:1.18-2.oe1.aarch64      libpcap-devel-14:1.9.1-4.oe1.aarch64
readline-devel-7.0-13.oe1.aarch64    sqlite-devel-3.24.0-9.oe1.aarch64    tk-devel-1:8.6.8-4.oe1.aarch64
xz-devel-5.2.4-10.oe1.aarch64         expat-devel-2.2.6-5.oe1.aarch64      fontconfig-devel-2.13.1-3.oe1.aarch64
freetype-devel-2.9.1-5.oe1.aarch64    libX11-devel-1.6.9-2.oe1.aarch64      libXau-devel-1.0.9-2.oe1.aarch64
libXft-devel-2.3.2-13.oe1.aarch64     libXrender-devel-0.9.10-10.oe1.aarch64 libpng-devel-2:1.6.36-4.oe1.aarch64
libxcb-devel-1.13.1-2.oe1.aarch64     tcl-devel-1:8.6.8-8.oe1.aarch64      util-linux-devel-2.34-8.oe1.aarch64
xorg-x11-proto-devel-2018.4-3.oe1.noarch
```

```
Complete!
[root@opengauss01 ~]#
```

```
[root@opengauss01 ~]# yum install libffi-devel -y
```

```
[root@opengauss01 ~]# yum install libffi-devel -y
Last metadata expiration check: 1:00:25 ago on Tue 12 Jul 2022 01:37:00 PM CST.
Package libffi-devel-3.3-7.oe1.aarch64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[root@opengauss01 ~]#
```

步骤 2 下载 Python3.8.3 版本

```
[root@opengauss01 ~]# wget https://repo.huaweicloud.com/python/3.8.3/Python-3.8.3.tgz
[root@opengauss01 ~]# tar -zxvf Python-3.8.3.tgz
```

创建编译安装目录

```
[root@opengauss01 ~]# mkdir /usr/local/python3
[root@opengauss01 ~]# cd Python-3.8.3
[root@opengauss01 Python-3.8.3]# ./configure --prefix=/usr/local/python3
[root@opengauss01 Python-3.8.3]# make && make install
```

等待数分钟后，出现 Successfully 回显，表明安装完成。

```
Looking in links: /tmp/tmpz4plauc
Collecting setuptools
Collecting pip
Installing collected packages: setuptools, pip
Successfully installed pip-19.2.3 setuptools-41.2.0
[root@opengauss01 Python-3.8.3]#
```

步骤 3 创建软链接

修改 Python 以及 pip 的软链接。（可一次性 copy 下面的 5 条命令执行）

```
rm -rf /usr/bin/python
ln -s /usr/local/python3/bin/python3 /usr/bin/python
rm -rf /usr/bin/pip
ln -s /usr/local/python3/bin/pip3 /usr/bin/pip
pip install --upgrade pip
```

看到下图表示执行完毕：

```
Downloading https://files.pythonhosted.org/packages/85/73/1ad018e35b0cb5/pip-22.3.1-py3-none-any.whl (2.1MB)
| 2.1MB 20kB/s
Installing collected packages: pip
Found existing installation: pip 19.2.3
Uninstalling pip-19.2.3:
Successfully uninstalled pip-19.2.3
Successfully installed pip-22.3.1
```

步骤 4 修改镜像源地址

```
[root@opengauss01 Python-3.8.3]# mkdir ~/.pip/
[root@opengauss01 Python-3.8.3]# vi ~/.pip/pip.conf
```

添加如下配置

```
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
```

步骤 5 使用 root 用户，进入到 xtuner 的安装目录。

```
[root@opengauss01 Python-3.8.3]# cd /opt/software/openGauss/bin/dbmind/xtuner/
[root@opengauss01 xtuner]# pip install -r requirements-aarch64.txt
[root@opengauss01 xtuner]# cd tuner
[root@opengauss01 tuner]# export PYTHONPATH='..'
```

设置配置文件 setup.cfg。

```
[[root@opengauss01 tuner]# cd ../
[root@opengauss01 xtuner]# vi setup.cfg
```

将以下内容添加进 **setup.cfg** 配置中。

```
[easy_install]
index_url = https://pypi.tuna.tsinghua.edu.cn/simple
```

步骤 6 安装 X-Tuner。

```
[root@opengauss01 xtuner]# python setup.py install
```

安装过程中会出现很多 warning，忽略即可，等待数分钟后，出现 Finished 回显，表明安装完成。

```
Using /usr/local/python3/lib/python3.8/site-packages
Searching for threadpoolctl==3.1.0
Best match: threadpoolctl 3.1.0
Adding threadpoolctl 3.1.0 to easy-install.pth file

Using /usr/local/python3/lib/python3.8/site-packages
Finished processing dependencies for openGauss-xtuner==2.1.0
[root@opengauss01 xtuner]#
```

步骤 7 配置 gs_xtuner 环境变量

```
[root@opengauss01 xtuner]# vi /etc/profile
```

添加如下配置到文件末尾。

```
PATH=$PATH:/usr/local/python3/bin
export PATH
```

```
unset i
unset -f pathmunge

if [ -n "${BASH_VERSION-}" ] ; then
    if [ -f /etc/bashrc ] ; then
        # Bash login shells run only /etc/profile
        # Bash non-login shells run only /etc/bashrc
        # Check for double sourcing is done in /etc/bashrc.
        . /etc/bashrc
    fi
fi

PATH=$PATH:/usr/local/python3/bin
export PATH

-- INSERT --
```

使环境变量生效。

```
[root@opengauss01 xtuner]# source /etc/profile
```

检测是否成功。

```
[root@opengauss01 xtuner]# gs_xtuner --help
```

```
[root@opengauss01 xtuner]# gs_xtuner --help
usage: gs_xtuner [-h] [--db-name DB_NAME] [--db-user DB_USER] [--port PORT] [--host HOST] [--host-user HOST_USER]
                [--host-ssh-port HOST_SSH_PORT] [-f DB_CONFIG_FILE] [-x TUNER_CONFIG_FILE] [-v]
                {train,tune,recommend}

X-Tuner: a self-tuning tool integrated by openGauss.

positional arguments:
  {train,tune,recommend}  Train a reinforcement learning model or tune database by model. And also can recommend best_knobs
                          according to your workload.

optional arguments:
  -h, --help                show this help message and exit
  -f DB_CONFIG_FILE, --db-config-file DB_CONFIG_FILE
                          You can pass a path of configuration file otherwise you should enter database information by
                          command arguments manually. Please see the template file share/server.json.template.
  -x TUNER_CONFIG_FILE, --tuner-config-file TUNER_CONFIG_FILE
                          This is the path of the core configuration file of the X-Tuner. You can specify the path of the
                          new configuration file. The default path is /usr/local/python3/lib/python3.8/site-
                          packages/openGauss_xtuner-2.1.0-py3.8.egg/tuner/xtuner.conf. You can modify the configuration
                          file to control the tuning process.
  -v, --version              show program's version number and exit

Database Connection Information:
  --db-name DB_NAME          The name of database where your workload running on.
  --db-user DB_USER          Use this user to login your database. Note that the user must have sufficient permissions.
  --port PORT                Use this port to connect with the database.
  --host HOST                The IP address of your database installation host.
  --host-user HOST_USER      The login user of your database installation host.
  --host-ssh-port HOST_SSH_PORT
                          The SSH port of your database installation host.
[root@opengauss01 xtuner]#
```

5.2.2 使用 X-Tuner 进行参数优化

步骤 1 切换到 omm 用户，将环境变量加入 omm 用户的.bashrc 文件中。

```
[root@opengauss01 xtuner]# su - omm
[omm@opengauss01 ~]$ vi ~/.bashrc
```

在文件末尾加入以下内容并保存退出。


```
export GAUSSHOME=/opt/software/openGauss
export PATH=$GAUSSHOME/bin:$PATH
export LD_LIBRARY_PATH=$GAUSSHOME/lib:$LD_LIBRARY_PATH
```

```
# Source default setting
[ -f /etc/bashrc ] && . /etc/bashrc

# User environment PATH
PATH="$HOME/.local/bin:$HOME/bin:$PATH"
export PATH

export GAUSSHOME=/opt/software/openGauss
export PATH=$GAUSSHOME/bin:$PATH
export LD_LIBRARY_PATH=$GAUSSHOME/lib:$LD_LIBRARY_PATH
```

使环境变量生效。

```
[omm@opengauss01 ~]$ source ~/.bashrc
```

步骤 2 执行 TPCB 脚本，获得测试时间，**将执行结果截图粘贴至实验报告。**

```
[omm@opengauss01 ~]$ gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql >
/opt/software/tpch-kit/dbgen/queries/queries01.log
```

这里因为数据量大，且数据库是初始状态，未经优化，所以需要等待二十多分钟后才能执行完成，中间可以通过开启另一个 CloudShell 连接，然后使用 tail 工具来监控执行进展。

在另一个 CloudShell 连接窗口中执行如下内容进行监控。

```
[root@opengauss01 ~]$ tail -f /opt/software/tpch-kit/dbgen/queries/queries01.log
```

i_returnflag	i_linestatus	sum_qty	sum_base_price	sum_disc_price	sum_charge	avg_qty	avg_price
A	F	37734107.00	56586554400.73	53758257134.8700	55909065222.827692	25.5220058532573370	38273.129734621672
04998529583839761162		1478493					
N	F	991417.00	1487504710.38	1413082168.0541	1469649223.194375	25.5164719205229835	38284.467760848304
05009342667421629691		38854					
N	O	74476040.00	111701729697.74	106118230307.6056	110367043872.497010	25.5022267695849915	38249.117988908270
04999658605370408037		2920374					
R	F	37719753.00	56568041380.90	53741252684.6040	55889619119.831932	25.5057936126907707	38250.854626099657
05000940583012705647		1478870					

当监控中返回 total time: 字样时，表示执行完成，接着关闭新开启的 CloudShell 连接。

在原来 CloudShell 连接窗口中查看 queries01.log。

```
[omm@opengauss01 ~]$ tail -10 /opt/software/tpch-kit/dbgen/queries/queries01.log
```

得到结果如下，**将结果截图**：

```
cntrycode | numcust | totacctbal
-----+-----+-----
13 | 888 | 6737713.99
17 | 861 | 6460573.72
18 | 964 | 7236687.40
23 | 892 | 6701457.95
29 | 948 | 7158866.63
30 | 909 | 6808436.13
31 | 922 | 6806670.18
(7 rows)

total time: 1231972 ms
```


步骤3 切换至 root 用户，执行 X-Tuner 进行参数建议优化，**执行结果截图粘贴至实验报告。**

需要输入正确的 **omm 数据库用户密码**及 **omm 操作系统用户密码**（密码为关卡一中创建 omm 时设置的用户密码）。

```
[omm@opengauss01 ~]$ exit
[root@opengauss01 xtuner]# gs_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1
--host-user omm
```

返回结果为：

name	recommend	min	max	restart
default_statistics_target	1000	100	1000	False
effective_cache_size	21602940	187388	21602940	False
effective_io_concurrency	200	150	250	False
enable_mergejoin	off	0	1	False
enable_nestloop	off	0	1	False
max_connections	370	50	741	True
max_prepared_transactions	370	50	741	True
max_process_memory	28803920	22403048	28803920	True
random_page_cost	1.0	1.0	2.0	False
shared_buffers	187388	187392	215500	True
wal_buffers	5855	2048	5855	True

步骤4 使用 omm 用户，对以上参数进行优化。

```
[root@opengauss01 xtuner]# su - omm
[omm@opengauss01 ~]$ gs_guc set -D /opt/software/openGauss/data/ -c "shared_buffers = 187388" -c
"max_connections = 370" -c "max_prepared_transactions = 370" -c "effective_cache_size = 21602940" -c
"effective_io_concurrency = 200" -c "wal_buffers = 5855" -c "random_page_cost = 1" -c
"default_statistics_target = 1000"
```

返回结果为：

```
expected instance path: [/opt/software/openGauss/data/postgresql.conf]
gs_guc set: shared_buffers=186864: [/opt/software/openGauss/data/postgresql.conf]
gs_guc set: max_connections=370: [/opt/software/openGauss/data/postgresql.conf]
gs_guc set: effective_cache_size=21602940: [/opt/software/openGauss/data/postgresql.conf]
gs_guc set: effective_io_concurrency=200: [/opt/software/openGauss/data/postgresql.conf]
gs_guc set: wal_buffers=5839: [/opt/software/openGauss/data/postgresql.conf]
gs_guc set: random_page_cost=1: [/opt/software/openGauss/data/postgresql.conf]
gs_guc set: default_statistics_target=1000: [/opt/software/openGauss/data/postgresql.conf]

Total instances: 1. Failed instances: 0.
Success to perform gs_guc!
```

步骤5 使用 omm 用户，重启数据库。（刚才修改的配置有几个需要重启才能生效）

```
[omm@opengauss01 ~]$ gs_ctl stop
[omm@opengauss01 ~]$ gs_ctl start -D /opt/software/openGauss/data -Z single_node -l logfile
```

步骤6 获取参数值，**将执行结果截图粘贴至实验报告。**

```
[omm@opengauss01 ~]$ cd /opt/software/openGauss/data
[omm@opengauss01 data]$ cat postgresql.conf|grep -E
'shared_buffers|max_connections|effective_cache_size|effective_io_concurrency|wal_buffers|random_page_co
st|default_statistics_target'
```

步骤 7 再次执行步骤 2，**将执行结果截图粘贴至实验报告**，对比优化前的执行时间。

步骤 8 **【附加题】**回到步骤 4，gs_guc set 命令只修改了其中几个推荐优化的参数，同学们可以把推荐优化的参数都设置一下，再进行测试，**将执行结果截图粘贴至实验报告**，跟步骤 7 的结果进行对比。（记得修改参数后，如果 restart 是 True 的参数，需要重启才能生效）

5.2.3 Index-advisor：索引推荐

步骤 1 登录数据库。

```
[omm@opengauss01 data]$ gsql -d tpch -p 5432 -r
```

步骤 2 备用知识。

下面会以一个查询来演示索引推荐的用法。这个查询涉及的表有 litemall_orders 订单表、address_dimension 地址表、date_dimension 日期表。

```
tpch=# select count(*) from litemall_orders;
count
-----
100000
(1 row)

tpch=# select count(*) from address_dimension;
count
-----
8002
(1 row)

tpch=# select count(*) from date_dimension;
count
-----
29586
(1 row)
```

每个表的表结构可以用如下命令查看：

```
tpch=# \d litemall_orders
```

```

tpch=# \d litemall_orders
Table "public.litemall_orders"
  Column          |      Type      | Modifiers
-----+-----+-----
 order_id         | integer        | not null
 order_sn         | character varying(63) | default NULL::character varying
 order_status     | smallint       |
 user_key         | integer        |
 address_key      | integer        |
 goods_price      | numeric(10,2)  | default NULL::numeric
 freight_price    | numeric(10,2)  | default NULL::numeric
 coupon_price     | numeric(10,2)  | default NULL::numeric
 integral_price   | numeric(10,2)  | default NULL::numeric
 groupon_price    | numeric(10,2)  | default NULL::numeric
 order_price      | numeric(10,2)  | default NULL::numeric
 actual_price     | numeric(10,2)  | default NULL::numeric
 pay_id           | character varying(63) | default NULL::character varying
 pay_date         | integer        |
 pay_time         | integer        |
 ship_sn          | character varying(63) | default NULL::character varying
 ship_channel     | character varying(63) | default NULL::character varying
 ship_date        | integer        |
 ship_time        | integer        |
 confirm_date     | integer        |
 confirm_time     | integer        |
 message          | character varying(512) | default NULL::character varying
 comments         | smallint       |
 aftersale_status | smallint       |
 refund_amount    | numeric(10,2)  | default NULL::numeric
 refund_type      | character varying(63) | default NULL::character varying
 refund_content   | character varying(127) | default NULL::character varying
 refund_date      | integer        |
 refund_time      | integer        |
 end_date         | integer        |
 end_time         | integer        |
 add_date         | integer        |
 add_time         | integer        |
 update_date      | integer        |
 update_time      | integer        |
 deleted          | tinyint        |
Indexes:
    "litemall_orders_pkey" PRIMARY KEY, btree (order_id) TABLESPACE pg_default

```

步骤 3 使用 SQL 查询 2020 年 3 月每个省的订单收入，并按收入额降序排序。

```

tpch=# SELECT ad.province AS province, SUM(o.actual_price) AS GMV
FROM litemall_orders o,
     address_dimension ad,
     date_dimension dd
WHERE o.address_key = ad.address_key
     AND o.add_date = dd.date_key
     AND dd.year = 2020
     AND dd.month = 3
GROUP BY ad.province
ORDER BY SUM(o.actual_price) DESC;

```

步骤 4 使用 explain，对该 SQL 加以分析，将执行结果截图粘贴至实验报告。

```

tpch=# EXPLAIN
SELECT ad.province AS province, SUM(o.actual_price) AS GMV
FROM litemall_orders o,
     address_dimension ad,

```

```

date_dimension dd
WHERE o.address_key = ad.address_key
AND o.add_date = dd.date_key
AND dd.year = 2020
AND dd.month = 3
GROUP BY ad.province
ORDER BY SUM(o.actual_price) DESC;

```

获得执行计划结果为：

```

QUERY PLAN
-----
Sort (cost=4593.80..4593.88 rows=31 width=47)
  Sort Key: (sum(o.actual_price)) DESC
  -> HashAggregate (cost=4592.72..4593.03 rows=31 width=47)
    Group By Key: ad.province
    -> Hash Join (cost=4354.43..4585.97 rows=1351 width=15)
      Hash Cond: (ad.address_key = o.address_key)
      -> Seq Scan on address_dimension ad (cost=0.00..188.02 rows=8002 width=14)
      -> Hash (cost=4337.54..4337.54 rows=1351 width=9)
        -> Hash Join (cost=1031.78..4337.54 rows=1351 width=9)
          Hash Cond: (o.add_date = dd.date_key)
          -> Seq Scan on litemall_orders o (cost=0.00..3041.00 rows=100000 width=13)
          -> Hash (cost=1031.76..1031.76 rows=2 width=4)
            -> Seq Scan on date_dimension dd (cost=0.00..1031.76 rows=2 width=4)
              Filter: ((year = 2020) AND ((month)::bigint = 3))
(14 rows)

```

步骤 5 使用索引推荐功能，对查询语句进行推荐。

```

tpch=# select * from gs_index_advise('
SELECT ad.province AS province, SUM(o.actual_price) AS GMV
FROM litemall_orders o,
      address_dimension ad,
      date_dimension dd
WHERE o.address_key = ad.address_key
AND o.add_date = dd.date_key
AND dd.year = 2020
AND dd.month = 3
GROUP BY ad.province
ORDER BY SUM(o.actual_price) DESC');

```

获得索引推荐结果如下：

```

 schema |      table      |      column
-----+-----+-----
 public | litemall_orders | (address_key,add_date)
 public | address_dimension |
 public | date_dimension   | (year)
(3 rows)

```

步骤 6 在 litemall_orders 和 date_dimension 表上创建虚拟索引列。

```

tpch=# select * from hypopg_create_index('create index on litemall_orders(address_key,add_date)');
tpch=# select * from hypopg_create_index('create index on date_dimension(year)');

```

步骤 7 查看创建的虚拟索引列。

```
tpch=# select * from hypopg_display_index();
```

获得创建虚拟列的结果：

```
tpch=# select * from hypopg_display_index();
          indexname          | indexrelid | table          | column
-----+-----+-----+-----
<24596>btree_litemall_orders_address_key_add_date |      24596 | litemall_orders | (address_key, add_date)
<24597>btree_date_dimension_year |      24597 | date_dimension  | (year)
(2 rows)
```

步骤 8 获取索引虚拟列大小结果（单位为：字节）。

其中 24596 和 24597 为上一步中查询到的 indexrelid，此处查询需要进行替换成自己系统中查询到的值。

```
tpch=# select * from hypopg_estimate_size(24596);
tpch=# select * from hypopg_estimate_size(24597);
```

返回结果为：

```
tpch=# select * from hypopg_estimate_size(24596);
hypopg_estimate_size
-----
          4177928
(1 row)

tpch=# select * from hypopg_estimate_size(24597);
hypopg_estimate_size
-----
          696328
(1 row)
```

步骤 9 开启 GUC 参数 enable_hypo_index。

该参数控制数据库的优化器进行 EXPLAIN 时是否考虑创建的虚拟索引。通过对特定的查询语句执行 explain，用户可根据优化器给出的执行计划评估该索引是否能够提升该查询语句的执行效率。

```
tpch=# set enable_hypo_index = on;
```

步骤 10 再次使用 explain，对刚才执行的 SQL 加以分析，将执行结果截图粘贴至实验报告。

```
tpch=# EXPLAIN
SELECT ad.province AS province, SUM(o.actual_price) AS GMV
FROM litemall_orders o,
     address_dimension ad,
     date_dimension dd
WHERE o.address_key = ad.address_key
     AND o.add_date = dd.date_key
     AND dd.year = 2020
     AND dd.month = 3
GROUP BY ad.province
ORDER BY SUM(o.actual_price) DESC;
```

获得执行计划：

```

QUERY PLAN
-----
Sort (cost=3579.58..3579.65 rows=31 width=47)
  Sort Key: (sum(o.actual_price)) DESC
  -> HashAggregate (cost=3578.50..3578.81 rows=31 width=47)
    Group By Key: ad.province
    -> Hash Join (cost=3340.21..3571.74 rows=1351 width=15)
      Hash Cond: (ad.address_key = o.address_key)
      -> Seq Scan on address_dimension ad (cost=0.00..188.02 rows=8002 width=14)
      -> Hash (cost=3323.32..3323.32 rows=1351 width=9)
        -> Hash Join (cost=17.56..3323.32 rows=1351 width=9)
          Hash Cond: (o.add_date = dd.date_key)
          -> Seq Scan on litemall_orders o (cost=0.00..3041.00 rows=100000 width=13)
          -> Hash (cost=17.53..17.53 rows=2 width=4)
            -> Index Scan using <24597>btree_date_dimension_year on date_dimension dd (cost=0.00..17.53 rows=2 width=4)
              Index Cond: (year = 2020)
              Filter: ((month)::bigint = 3)
(15 rows)

```

步骤 11 对步骤 5 创建的索引虚拟列进行清理。

```
tpch=# select * from hypopg_reset_index();
```

步骤 12 【附加题】对 queries.sql 里面的语句进行索引推荐，创建虚拟索引列，重新执行 queries.sql 查询，将执行结果截图粘贴至实验报告。

先退出数据库，在操作系统 omm 用户环境下调用 queries.sql 脚本。

```

tpch=# \q
[omm@opengauss01 data]$ gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log

```

等待二十多分钟后才能执行完成（当出现耗时表示执行完成），然后查询 queries02.log 日志文件：

```
[omm@opengauss01 data]$ tail -10 /opt/software/tpch-kit/dbgen/queries/queries02.log
```

5.2.4 关卡验证

1. 使用 X-Tuner 完成参数优化建议，并针对建议值完成 openGauss 数据库参数优化，对步骤 2、3、6、7 结果截图粘贴至实验报告；
2. 完成索引推荐功能的优化，使用索引推荐以及虚拟索引列的创建，对比创建的索引虚拟列前后的执行时间，对步骤 3 和 9 进行截图粘贴至实验报告。

6

关卡四【附加题】：openGauss 的 DB4AI 特性应用

6.1 实验介绍

6.1.1 关于本实验

本实验关卡，在安装完成的 openGauss 数据库上，使用 DB4AI 的功能，利用 openGauss 的 DB4AI 性能进行房价的预测。

6.1.2 实验目的

- 掌握 openGauss 数据库的 DB4AI 的功能。

6.2 实验任务及步骤

6.2.1 利用 DB4AI 原生 AI 引擎训练并预测模型

步骤 1 在数据库中创建一张表，用于数据的训练与预测。

在 OMM 用户环境下，登录数据库。

```
[omm@opengauss01 data]$ gsql -d postgres -p 5432 -r
```

然后用以下语句来创建表。

```
openGauss=# DROP TABLE IF EXISTS houses;
openGauss=# CREATE TABLE houses (id INT, tax INT, bedroom INT, bath FLOAT, price INT, size INT, lot INT);
INSERT INTO houses VALUES
(1, 590, 2, 1, 50000, 770, 22100),
(2, 1050, 3, 2, 85000, 1410, 12000),
(3, 20, 3, 1, 22500, 1060, 3500),
(4, 870, 2, 2, 90000, 1300, 17500),
(5, 1320, 3, 2, 133000, 1500, 30000),
(6, 1350, 2, 1, 90500, 820, 25700),
(7, 2790, 3, 2.5, 260000, 2130, 25000),
```



```
(8, 680, 2, 1, 142500, 1170, 22000),
(9, 1840, 3, 2, 160000, 1500, 19000),
(10, 3680, 4, 2, 240000, 2790, 20000),
(11, 1660, 3, 1, 87000, 1030, 17500),
(12, 1620, 3, 2, 118600, 1250, 20000),
(13, 3100, 3, 2, 140000, 1760, 38000),
(14, 2070, 2, 3, 148000, 1550, 14000),
(15, 650, 3, 1.5, 65000, 1450, 12000);
```

步骤 2 通过 \d 命令观察新创建的表 house 结构。

```
openGauss=# \d houses
```

可以得到表结构信息：

```
openGauss=# \d houses
Table "public.houses"
Column | Type | Modifiers
-----+-----+-----
id | integer |
tax | integer |
bedroom | integer |
bath | double precision |
price | integer |
size | integer |
lot | integer |
```

步骤 3 通过 CREATE MODEL 语句，基于 SVM 算法创建一个二分类模型。

```
openGauss=# CREATE MODEL house_binary_classifier USING svm_classification FEATURES tax, bath, size
TARGET price < 100000 FROM houses;
```

执行 CREATE MODEL 语句后，会开启模型的训练，并输出训练过程中使用的超参数信息：

```
openGauss=# CREATE MODEL house_binary_classifier USING svm_classification FEATUR
ES tax, bath, size TARGET price < 100000 FROM houses;
NOTICE: Hyperparameter batch_size takes value DEFAULT (1000)
NOTICE: Hyperparameter decay takes value DEFAULT (0.950000)
NOTICE: Hyperparameter lambda takes value DEFAULT (0.010000)
NOTICE: Hyperparameter learning_rate takes value DEFAULT (0.800000)
NOTICE: Hyperparameter max_iterations takes value DEFAULT (100)
NOTICE: Hyperparameter max_seconds takes value DEFAULT (0)
NOTICE: Hyperparameter optimizer takes value DEFAULT (gd)
NOTICE: Hyperparameter tolerance takes value DEFAULT (0.000500)
NOTICE: Hyperparameter seed takes value DEFAULT (0)
NOTICE: Hyperparameter verbose takes value DEFAULT (FALSE)
NOTICE: GD shuffle cache size 4143
MODEL CREATED. PROCESSED 1
```

步骤 4 在 gs_model_warehouse 系统表中查看训练后的模型信息。

```
openGauss=# SELECT * FROM gs_model_warehouse WHERE modelname = 'house_binary_classifier';
```



```
openGauss=# SELECT * FROM gs_model_warehouse WHERE modelname = 'house_binary_classifier';
```

modelname	modelowner	createtime	processedtuples	discardedtuples
pre_process_time	exec_time	iterations	outputtype	modeltype
		query		
hyperparametersvalues		hyperparametersnames	weight	
lues	coefoids	trainingscoresname	hyperparametersoids	coefnames
ldescribe			trainingscoresvalue	coefva
(1 row)				

```
house_binary_classifier |          10 | 2022-07-14 16:09:33.614493 |          15 |          0 |  
0 | .002566 |          74 |          16 | svm_classification | CREATE MODEL house_binary  
_classifier USING svm_classification FEATURES tax, bath, size TARGET price < 100000 FROM houses WITH bat  
ch_size=10, learning_rate=0.001; | {-0.00206988,1.87436e-06,.00195499,6.52253e-05} | {batch_s  
ize,decay,lambd,learning_rate,max_iterations,max_seconds,optimize,tolerance,seed,verbose} | {10,.95,.0  
1,.001,100,0,gd,.0005,1657786173,false} | {23,701,701,701,23,23,1043,701,23,16} | {categories} | {false,  
true} | | {accuracy,f1,precision,recall,loss} | {.733333,.714286,.714286,.714286,1.1813} |
```

整体准确率要相对默认超参数有所提升。

步骤 8 使用 PREDICT BY 语句预测样本数据。

```
openGauss=# SELECT tax, bath, size, price, price < 100000 AS price_actual, PREDICT BY
house_binary_classifier (FEATURES tax, bath, size) AS price_pred FROM houses;
```

返回结果为：

tax	bath	size	price	price_actual	price_pred
590	1	770	50000	t	t
1050	2	1410	85000	t	t
20	1	1060	22500	t	t
870	2	1300	90000	t	t
1320	2	1500	133000	f	t
1350	1	820	90500	t	f
2790	2.5	2130	260000	f	f
680	1	1170	142500	f	t
1840	2	1500	160000	f	f
3680	2	2790	240000	f	f
1660	1	1030	87000	t	f
1620	2	1250	118600	f	f
3100	2	1760	140000	f	f
2070	3	1550	148000	f	f
650	1.5	1450	65000	t	t

price_pred 列的输出结果即为预测结果，price_actual 列的结果为真实值。

步骤 9 通过 CREATE MODEL 语句创建一个逻辑回归（logistic regression）模型。

```
openGauss=# CREATE MODEL house_logistic_classifier USING logistic_regression FEATURES tax, bath, size
target price < 100000 FROM houses WITH learning_rate=0.001;
```

步骤 10 利用训练好的逻辑回归模型预测数据，并与 SVM 算法进行比较，**执行结果截图粘贴至实验报告。**

```
openGauss=# SELECT tax, bath, size, price, price < 100000 AS price_actual, PREDICT BY
house_binary_classifier (FEATURES tax, bath, size) AS price_svm_pred, PREDICT BY house_logistic_classifier
(FEATURES tax, bath, size) AS price_logistic_pred FROM houses;
```

输出结果如下：

tax	bath	size	price	price_actual	price_svm_pred	price_logistic_pred
590	1	770	50000	t	t	t
1050	2	1410	85000	t	t	t
20	1	1060	22500	t	t	t
870	2	1300	90000	t	t	t
1320	2	1500	133000	f	t	t
1350	1	820	90500	t	f	f
2790	2.5	2130	260000	f	f	f
680	1	1170	142500	f	t	t
1840	2	1500	160000	f	f	f
3680	2	2790	240000	f	f	f
1660	1	1030	87000	t	f	f
1620	2	1250	118600	f	f	f
3100	2	1760	140000	f	f	f
2070	3	1550	148000	f	f	f
650	1.5	1450	65000	t	t	t

(15 rows)

退出数据库。

```
openGauss=# \q
```

6.2.2 关卡验证

完成模型训练和预测过程，将 svm 算法和逻辑回归模型预测结果对比**截图粘贴至实验报告**。

7 清理工作：资源释放

7.1 实验介绍

7.1.1 关于本实验

本实验通过在华为云上删除弹性云服务器 ECS 实例，帮助学员和读者掌握在华为云上清除弹性云服务器 ECS 资源。

7.1.2 实验目的

- 掌握弹性云服务器资源释放操作；
- 熟悉华为云操作。

7.2 删除弹性云服务器及相关资源

完成实验后请务必删除华为云上的收费资源，以免造成不必要的收费。找到创建的弹性云服务器 ECS，按照如下步骤进行删除。

步骤 1 打开云服务器控制台，在需要删除的云服务器后面选择“更多>删除”。



步骤 2 在弹出对话框中勾选“释放云服务器绑定的弹性公网 IP 地址”和“删除云服务器挂载的数据盘”，然后点击“是”。



步骤3 查看到列表中已没有资源时，表示弹性云服务器已删除，**执行结果截图粘贴至实验报告。**



7.3 关卡验证

完成资源释放，**执行结果截图粘贴至实验报告。**

8 缩略语表

缩略语	英文全称	中文全称
PC	Personal Computer	个人电脑
EIP	Elastic IP	弹性IP
VPC	Virtual Private Cloud	虚拟私有云
SG	Security Groups	安全组
ECS	Elastic Cloud Server	弹性云服务器