openGauss的AI特性实践

实验指导手册

学生版



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验环境介绍 3](#_Toc80259567)

[1.1 实验介绍 3](#_Toc80259568)

[1.1.1 关于本实验 3](#_Toc80259569)

[1.1.2 读者知识背景 3](#_Toc80259570)

[1.1.3 实验设备介绍 3](#_Toc80259571)

[2 关卡一：openGauss数据安装及基本操作 4](#_Toc80259572)

[2.1 实验介绍 4](#_Toc80259573)

[2.1.1 关于本实验 4](#_Toc80259574)

[2.1.2 实验目的 4](#_Toc80259575)

[2.2 实验任务及步骤 4](#_Toc80259576)

[2.2.1 编译前准备 4](#_Toc80259577)

[2.2.2 数据库安装编译 6](#_Toc80259578)

[2.2.3 关卡验证 8](#_Toc80259579)

[2.2.4 思考题 8](#_Toc80259580)

[3 关卡二：openGauss数据导入及基本操作 9](#_Toc80259581)

[3.1 实验介绍 9](#_Toc80259582)

[3.1.1 关于本实验 9](#_Toc80259583)

[3.1.2 实验目的 9](#_Toc80259584)

[3.2 实验任务及步骤 9](#_Toc80259585)

[3.2.1 数据初始化 9](#_Toc80259586)

[3.2.2 列存表的创建与使用 13](#_Toc80259587)

[3.2.3 物化视图的使用 15](#_Toc80259588)

[3.2.4 关卡验证 16](#_Toc80259589)

[3.2.5 思考题 16](#_Toc80259590)

[4 关卡三：openGauss的AI4DB特性应用 17](#_Toc80259591)

[4.1 实验介绍 17](#_Toc80259592)

[4.1.1 关于本实验 17](#_Toc80259593)

[4.1.2 实验目的 17](#_Toc80259594)

[4.2 实验任务及步骤 17](#_Toc80259595)

[4.2.1 将X-Tuner 安装到系统中 17](#_Toc80259596)

[4.2.2 使用X-Tuner进行参数优化 18](#_Toc80259597)

[4.2.3 Index-advisor：索引推荐 20](#_Toc80259598)

[4.2.4 关卡验证 22](#_Toc80259599)

[4.2.5 思考题 23](#_Toc80259600)

[5 关卡四：OpenGauss的原生DB4AI引擎 24](#_Toc80259601)

[5.1 实验介绍 24](#_Toc80259602)

[5.1.1 关于本实验 24](#_Toc80259603)

[5.1.2 实验目的 24](#_Toc80259604)

[5.2 实验任务及步骤 24](#_Toc80259605)

[5.2.1 利用DB4AI原生AI引擎训练并预测模型 24](#_Toc80259606)

[5.2.2 关卡验证 28](#_Toc80259607)

[5.2.3 思考题 28](#_Toc80259608)

# 实验环境介绍

## 实验介绍

### 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，进行列存表的使用与物化视图的使用；后续进行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，完成分类测试和回归测试算法。

### 读者知识背景

本课程为创新实践课，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

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。

### 实验设备介绍

* 组网说明

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：

软件类型和配置描述如下：

实验设备配套关系

|  |  |
| --- | --- |
| 设备名称 | 设备型号 |
| Linux操作系统 | openEuler 20.3LTS |
| 工具 | Putty WinSCP TPCH TensorFlow |
| Python | Python 3.7.X |

# 关卡一：openGauss数据安装及基本操作

## 实验介绍

### 关于本实验

本实验通过第三方工具，完成openGauss的编译安装。

### 实验目的

掌握第三方工具库编译；

掌握openGauss编译安装。

## 实验任务及步骤

### 编译前准备

用root用户名，使用之前设置的密码登录ECS。



创建openGauss数据库的安装用户omm及其属组dbgrp。

[root@opengauss01 ~]# **groupadd -g 1000 dbgrp**

[root@opengauss01 ~]# **useradd -g dbgrp -u 1000 -d /home/omm omm**

修改omm用户密码。

[root@opengauss01 ~]# **passwd omm**

输入修改的omm用户密码，建议设置成复杂密码。

创建openGauss源码存放及openGauss安装路径。

[root@opengauss01 ~]# **mkdir -p /opt/software/openGauss/data**

下载第三方编译库，社区针对 centos\_7.6\_x86\_64、openEuler20.03 LTS\_arm、openEuler20.03 LTS\_ x86\_64 三种架构及操作系统已经提供了编译好的二进制，对于这三种系统架构，可以直接使用社区提供的编译好的文件openGauss-third\_party\_binarylibs.tar.gz。

[root@opengauss01 ~]# **cd /opt/software**

[root@opengauss01 software]# **wget** [**https://opengauss.obs.cn-south-1.myhuaweicloud.com/1.1.0/openGauss-third\_party\_binarylibs.tar.gz**](https://opengauss.obs.cn-south-1.myhuaweicloud.com/1.1.0/openGauss-third_party_binarylibs.tar.gz)

下载时间比较长大概10分钟，请耐心等待。

解压下载好的第三方编译库，并重命名为binarylibs。

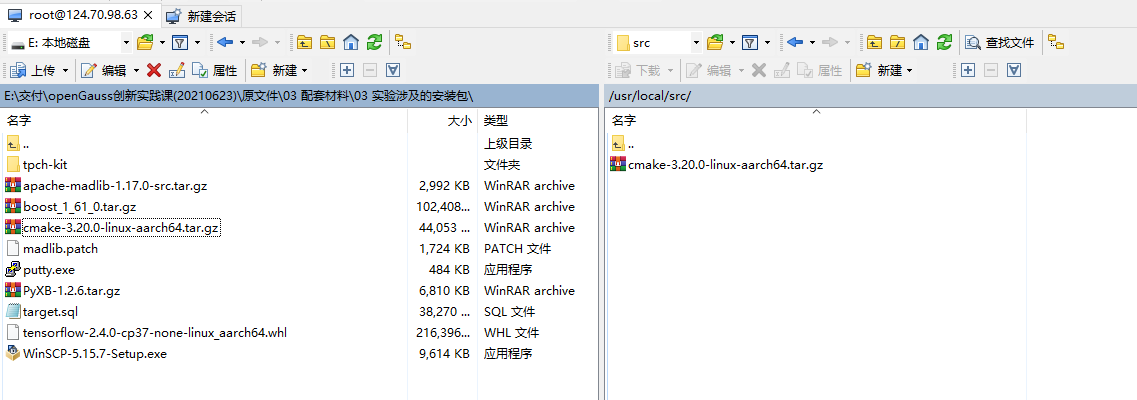
[root@opengauss01 software]# **tar -zxvf openGauss-third\_party\_binarylibs.tar.gz**

[root@opengauss01 software]# **mv openGauss-third\_party\_binarylibs/ binarylibs/**

下载openGauss源码。

[root@opengauss01 software]# **git clone -b master https://gitee.com/opengauss/openGauss-server.git**[root@opengauss01 software]# **cd openGauss-server**  
[root@opengauss01 openGauss-server]# **git checkout f40ff5a0e9dcde6178ac116b391c662f3a516bbb**

将[cmake-3.20.0-linux-aarch64.tar.gz](https://github.com/Kitware/CMake/releases/download/v3.20.0/cmake-3.20.0-linux-aarch64.tar.gz)包，上传至服务器/usr/local/src/下（可使用winscp工具）,如下图：



然后进行进行解压，具体如下：

[root@opengauss01 software]# **cd /usr/local/src/**

[root@opengauss01 src]# **tar -zxvf cmake-3.20.0-linux-aarch64.tar.gz**

[root@opengauss01 src]# **chmod 755 /usr/local/src/cmake-3.20.0-linux-aarch64/**

使用yum安装依赖包。

[root@opengauss01 src]# **yum install -y libaio-devel ncurses-devel pam-devel libffi-devel libtool libtool-devel libtool-ltdl openssl-devel bison golang flex dkms-2.6.1-5.oe1.noarch python3-devel patch**

替换python版本，将python3的链接设置为python的。

[root@opengauss01 src]# **cd /usr/bin**

[root@opengauss01 bin]# **mv python python.bak**

[root@opengauss01 bin]# **ln -s python3 /usr/bin/python**

检查python的版本。

[root@opengauss01 bin]# **python -V**

返回值为：Python 3.7.4

修改/opt/software路径的用户所属组及权限。

[root@opengauss01 bin]# **chown omm:dbgrp -R /opt/software**

[root@opengauss01 bin]# **chmod 755 -R /opt/software**

### 数据库安装编译

切换omm用户。

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

设定omm用户的环境变量。

[omm@opengauss01 ~]$ **vi ~/.bash\_profile**

添加环境变量。

将以下环境变量内容，红色加粗部分根据实际安装环境进行设置，并写入.bash\_profile文件中。

# GaussDB Code-

export GAUSSHOME=/opt/software/openGauss #the path for openGauss

export PGDATA=$GAUSSHOME/data #the path for data

export BINARY\_PATH=/opt/software/binarylibs #the path for binarylibs

# GCC 7.3 compiler path

export CMAKEROOT=/usr/local/src/cmake-3.20.0-linux-aarch64/ #the path for cmake

export GCC\_PATH=$BINARY\_PATH/buildtools/openeuler\_aarch64/gcc7.3

export CC=$GCC\_PATH/gcc/bin/gcc

export CXX=$GCC\_PATH/gcc/bin/g++

export LD\_LIBRARY\_PATH=$GCC\_PATH/gcc/lib64:$GCC\_PATH/isl/lib:$GCC\_PATH/mpc/lib/:$GCC\_PATH/mpfr/lib/:$GCC\_PATH/gmp/lib/:$LD\_LIBRARY\_PATH

export PATH=$GCC\_PATH/bin:$CMAKEROOT/bin:$PATH

# Gauss DB install path

export PATH=$GAUSSHOME/bin:$PATH

export LD\_LIBRARY\_PATH=$GAUSSHOME/lib:$LD\_LIBRARY\_PATH

export GAUSS\_WARNING\_TYPE=2

export GS\_CLUSTER\_NAME=single

使环境变量生效。

[omm@opengauss01 ~]$ **source ~/.bash\_profile**

进入openGauss源码下，生成配置文件。

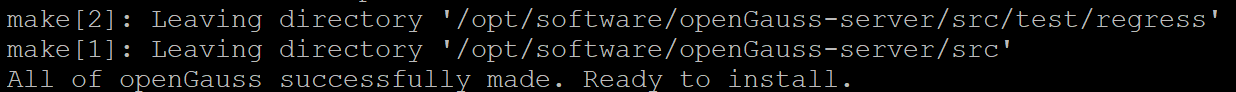
[omm@opengauss01 ~]$ **cd /opt/software/openGauss-server/**

[omm@opengauss01 openGauss-server]$ **./configure --gcc-version=7.3.0 --prefix=$GAUSSHOME CFLAGS="-O0" --enable-debug --without-readline --without-zlib --enable-cassert --enable-thread-safety CC=g++ --3rd=$BINARY\_PATH --with-python**

执行make命令进行编译。

[omm@opengauss01 openGauss-server]$ **make -j8**

当返回All of openGauss successfully made. Ready to install.时，表明编译成功。



执行make install安装。

[omm@opengauss01 openGauss-server]$ **make install**

当返回openGauss installation complete.时，表明安装完成。

C:\Users\hwx559043\AppData\Roaming\eSpace_Desktop\UserData\hwx559043\imagefiles\E785BC09-F9C7-477E-8076-2E22F5647950.png

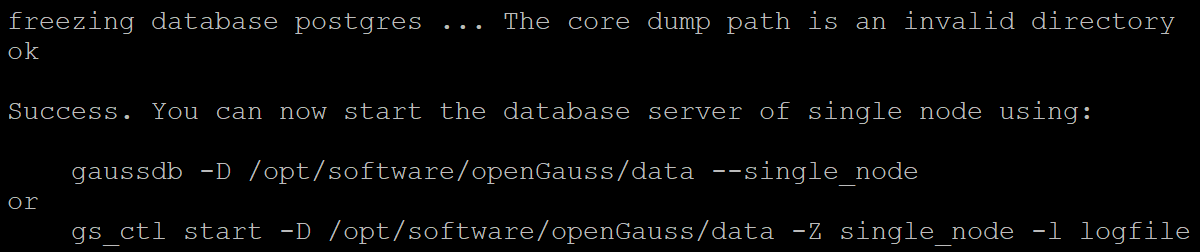
初始化数据库，需要设置数据库密码，请使用复杂密码，将{password}部分替换。

gs\_initdb -D $PGDATA --nodename=**hostname** --locale="en\_US.UTF-8" -Atrust -w **{password}**

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 openGauss-server]$ **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：



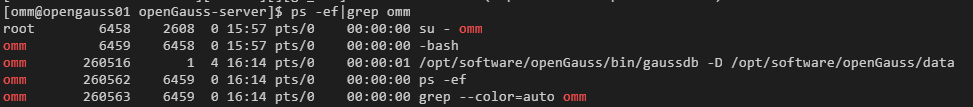
启动数据库。

[omm@opengauss01 openGauss-server]$ **gs\_ctl start -D /opt/software/openGauss/data -Z single\_node -l logfile**

查看数据库是否启动成功。

[omm@opengauss01 openGauss-server]$ **ps -ef|grep omm**

显示结果如下，表示启动成功：



数据库登录。

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

修改omm账号密码（可选步骤）。

postgres=# **ALTER USER omm identified by 'Huawei@13' replace 'Huawei#!13';**

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

查询数据库版本。

postgres=# **select version();**

version

-------------------------------------------------------------------------------------------------------------------------

PostgreSQL 9.2.4 (GaussDB Kernel V500R001C20 build 4c77c0bb) compiled at 2021-03-30 17:12:43 commit 0 last mr debug on aarch64-unknown-linux-gnu, compiled by g++ (GCC) 7.3.0, 64-bit

(1 row)

退出数据库。

postgres=# **\q**

### 关卡验证

**此处需要通过截图**，来完成关卡的验证，首先需要对数据库状态进行验证。

[omm@opengauss01 openGauss-server]$ **gs\_ctl status**

其次，对数据库进程**进行截图**验证，需包含数据库服务器的主机名。

[omm@opengauss01 openGauss-server]$ **ps -ef|grep omm**

简单描述关卡一中，安装数据库所需要的步骤。完成以上信息，表明关卡一完成。

### 思考题

为什么需要通过源码编译，完成数据库的安装？简单描述安装数据库所需要的步骤。

# 关卡二：openGauss数据导入及基本操作

## 实验介绍

### 关于本实验

本实验关卡，在安装完成的openGauss数据库上，进行TPCH和target数据初始化操作；完成openGauss中的列存表的创建、修改与查询；物化视图的创建与使用。

### 实验目的

掌握openGauss数据库的基本维护方法；

掌握openGauss数据库列存表的创建与使用；

掌握物化视图的创建与使用。

## 实验任务及步骤

### 数据初始化

使用root登录，进入/opt/software目录下。

[omm@opengauss01 openGauss-server]$ **exit**

[root@opengauss01 bin]# **cd /opt/software**

下载TPC-H测试包。

[root@opengauss01 software]# **git clone https://gitee.com/wang-tq/tpch-kit.git**

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/**

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/**

生成MakeFile文件。

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

连接openGauss数据库。

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

创建测试数据库tpch并退出。

postgres=# **CREATE DATABASE tpch;**

postgres=# **\q**

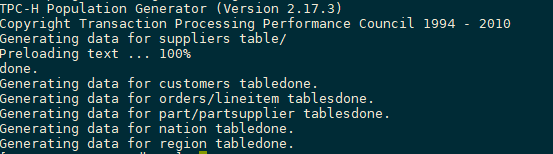
执行创建对象脚本。

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

产生测试数据。

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

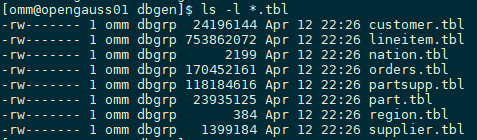
当返回如下内容，表示数据生产完成。



使用ls命令查看：

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

回显生成的数据。



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。

**for i in `ls \*.tbl`; do**

**table=${i/.tbl/}**

**echo "Loading $table..."**

**sed 's/|$//' $i > /tmp/$i**

**gsql tpch -q -c "TRUNCATE $table"**

**gsql tpch -c "\\copy $table FROM '/tmp/$i' CSV DELIMITER '|'"**

**done**

使用:wq退出并保存。

**:wq**

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：

Loading customer...

Loading lineitem...

Loading nation...

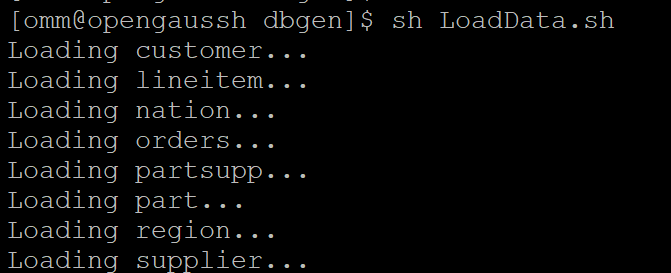
Loading orders...

Loading partsupp...

Loading part...

Loading region...

Loading supplier...

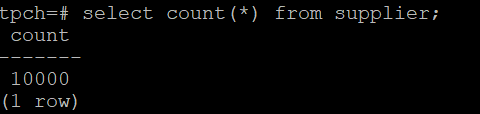


登录数据库验证，**将查询结果截图**。

[omm@opengauss01 dbgen]$ **gsql -d tpch -p 5432 -r**

tpch=# **select count(\*) from supplier;**

此处应有返回记录，数据量为10000。



退出数据库。

tpch=# **\q**

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/**

生成查询语句。

[omm@opengauss01 queries]$ **vim genda.sh**

将以下内容复制到genda.sh中。

**for i in {1..22}; do**

**./qgen -d $i>$i\_new.sql**

**./qgen -d $i\_new | sed 's/limit -1//' | sed 's/day (3)/day/' > queries.sql**

**done**

使用:wq退出并保存。

**:wq**

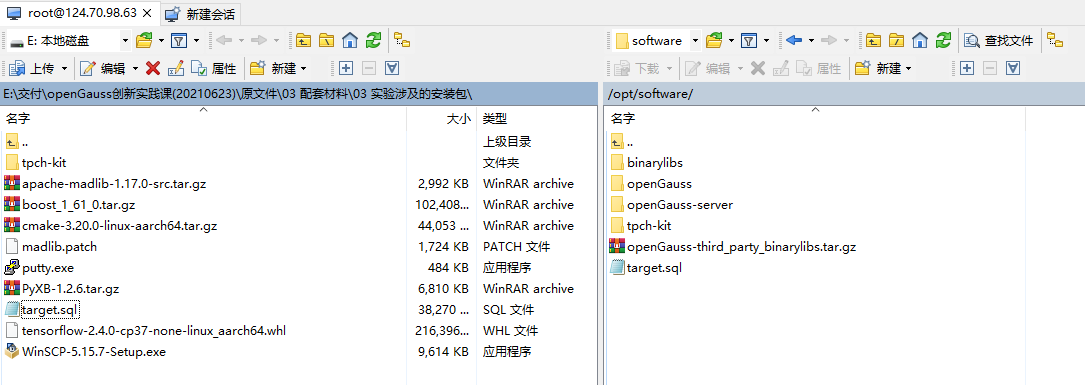
执行genda.sh脚本，生成查询语句

[omm@opengauss01 dbgen]$ **sh genda.sh**

验证生成的查询语句

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

上传target.sql脚本至/opt/software目录下（可以使用winscp等工具），具体如下图：



切换至root用户，将脚本的属组修改为omm。

[omm@opengauss01 dbgen]$ **exit**

[root@opengauss01 software]# **chown omm:dbgrp /opt/software/target.sql**

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**

等待数分钟。

登录数据库进行验证

[omm@opengauss01 ~]$ **gsql -d tpch -p 5432 -r**

tpch=# **\dt**

此处加上TPCH测试的表，总共有12张表。

### 列存表的创建与使用

创建与litemall\_orders表表结构相同的列存表。

tpch=# **CREATE TABLE "public"."litemall\_orders\_col" (**

**"order\_id" INTEGER NOT NULL DEFAULT NULL,**

**"order\_sn" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,**

**"order\_status" SMALLINT DEFAULT NULL,**

**"user\_key" INTEGER DEFAULT NULL,**

**"address\_key" INTEGER DEFAULT NULL,**

**"goods\_price" numeric(10,2) DEFAULT NULL,**

**"freight\_price" numeric(10,2) DEFAULT NULL,**

**"coupon\_price" numeric(10,2) DEFAULT NULL,**

**"integral\_price" numeric(10,2) DEFAULT NULL,**

**"groupon\_price" numeric(10,2) DEFAULT NULL,**

**"order\_price" numeric(10,2) DEFAULT NULL,**

**"actual\_price" numeric(10,2) DEFAULT NULL,**

**"pay\_id" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,**

**"pay\_date" INTEGER DEFAULT NULL,**

**"pay\_time" INTEGER DEFAULT NULL,**

**"ship\_sn" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,**

**"ship\_channel" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,**

**"ship\_date" INTEGER DEFAULT NULL,**

**"ship\_time" INTEGER DEFAULT NULL,**

**"confirm\_date" INTEGER DEFAULT NULL,**

**"confirm\_time" INTEGER DEFAULT NULL,**

**"message" varchar(512) COLLATE "pg\_catalog"."default" DEFAULT NULL,**

**"comments" SMALLINT DEFAULT NULL,**

**"aftersale\_status" SMALLINT DEFAULT NULL,**

**"refund\_amount" numeric(10,2) DEFAULT NULL,**

**"refund\_type" varchar(63) COLLATE "pg\_catalog"."default" DEFAULT NULL,**

**"refund\_content" varchar(127) COLLATE "pg\_catalog"."default" DEFAULT NULL,**

**"refund\_date" INTEGER DEFAULT NULL,**

**"refund\_time" INTEGER DEFAULT NULL,**

**"end\_date" INTEGER DEFAULT NULL,**

**"end\_time" INTEGER DEFAULT NULL,**

**"add\_date" INTEGER DEFAULT NULL,**

**"add\_time" INTEGER DEFAULT NULL,**

**"update\_date" INTEGER DEFAULT NULL,**

**"update\_time" INTEGER DEFAULT NULL,**

**"deleted" TINYINT DEFAULT NULL**

**)**

**with (orientation=column );**

将litemall\_orders表中的数据插入litemall\_orders\_col中。

tpch=# **insert into litemall\_orders\_col select \* from litemall\_orders;**

使用\timing开启SQL执行时间的返回

tpch=# **\timing**

使用SQL语句查询，2020年上半年litemall\_orders行存表中的order\_price的总和，并**对结果和耗时截图**。

tpch=# **select sum (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';**

使用SQL语句查询，2020年上半年litemall\_orders\_col列存表中的order\_price的总和，**并对结果和耗时截图**。

tpch=# **select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';**

使用SQL语句查询，2020年上半年litemall\_orders行存表中的order\_price的平均值，**并对结果和耗时截图**。

tpch=# **select avg(order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';**

使用SQL语句查询，2020年上半年litemall\_orders\_col列存表中的order\_price的平均值，**并对结果和耗时截图**。

tpch=# **select avg(order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';**

使用SQL语句查询，litemall\_orders行存表中order\_id为6的order\_price的值，**并对结果和耗时截图**。

tpch=# **select order\_price from litemall\_orders where order\_id=6;**

使用SQL语句查询，litemall\_orders\_col列存表中order\_id为6的order\_price的值，**并对结果和耗时截图**。

tpch=# **select order\_price from litemall\_orders\_col where order\_id=6;**

使用SQL语句，将litemall\_orders行存表中order\_id为6的order\_price修改为2468，**并对结果和耗时截图**。

tpch=# **update litemall\_orders set order\_price=2468 where order\_id=6;**

使用SQL语句，将litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，**并对结果和耗时截图**。

tpch=# **update litemall\_orders\_col set order\_price=2468 where order\_id=6;**

对比在行存表与列存表中的操作，根据每种SQL执行的时间，得出结论。

### 物化视图的使用

#### 全量物化视图

创建物化视图时需要的表，内容为查看消费金额大于100000的用户。

tpch=# **CREATE TABLE test\_view AS SELECT \* FROM (**

**SELECT ud.username,**

**gender,**

**SUM(o.actual\_price) AS totalspend**

**FROM litemall\_orders o, user\_dimension ud**

**WHERE o.user\_key = ud.user\_key**

**GROUP BY ud.username, ud.weixin\_openid, gender**

**ORDER BY SUM(o.actual\_price) DESC) WHERE totalspend > 100000**

**;**

对表内容进行查询，**将执行结果截图**。

tpch=# **SELECT \* FROM test\_view;**

创建全量物化视图。

tpch=# **CREATE MATERIALIZED VIEW v\_order AS select count(\*) from test\_view;**

使用物化视图统计人数，查询物化视图结果，**将执行结果截图**。

tpch=# **SELECT \* FROM v\_order;**

删除表中的数据。

tpch=# **DELETE FROM test\_view WHERE username='杨兰娟';**

对全量物化视图做全量刷新。

tpch=# **REFRESH MATERIALIZED VIEW v\_order;**

使用物化视图统计人数，查询物化视图结果，**将执行结果截图**。

tpch=# **SELECT \* FROM v\_order;**

删除物化视图。

tpch=# **DROP MATERIALIZED VIEW v\_order;**

#### 增量物化视图

创建增量物化视图。

tpch=# **CREATE INCREMENTAL MATERIALIZED VIEW vi\_order AS SELECT username,totalspend FROM test\_view;**

查询物化视图结果，**将执行结果截图**。

tpch=# **SELECT \* FROM vi\_order;**

插入数据。

tpch=# **INSERT INTO test\_view VALUES('杨兰娟',2 , 119391.00);**

增量刷新物化视图。

tpch=# **REFRESH INCREMENTAL MATERIALIZED VIEW vi\_order;**

查询物化视图结果，**将执行结果截图**。

tpch=# **SELECT \* FROM vi\_order;**

再次插入数据。

tpch=# **INSERT INTO test\_view VALUES('马景涛',2 , 139391.00);**

全量刷新物化视图。

tpch=# **REFRESH MATERIALIZED VIEW vi\_order;**

查询物化视图结果，**将执行结果截图**。

tpch=# **SELECT \* FROM vi\_order;**

删除物化视图。

tpch=# **DROP MATERIALIZED VIEW vi\_order;**

tpch=# **\q**

对比全量物化视图与增量物化视图，输出结论。

### 关卡验证

本关卡需要将行存与列存执行时间对比进行截图，同时通过对比，得出使用两种不同的存储所适用的场景；需要将全量与增量物化视图的查询结果进行截图，得出全量物化视图与增量物化视图的区别。

### 思考题

行存表与列存表在执行相同的SQL语句时，为何执行的时间不同？在执行哪些类型SQL时，行存表效率更高？在执行哪些类型SQL时，列存表效率更高？全量物化视图与增量物化视图有哪些差别？

# 关卡三：openGauss的AI4DB特性应用

## 实验介绍

### 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TPCH测试中的SQL语句进行优化。

### 实验目的

掌握openGauss数据库的AI4DB的功能；

掌握openGauss数据库的参数修改方法；

掌握openGauss数据库的SQL优化能力；

掌握openGauss数据库的索引创建与使用方法。

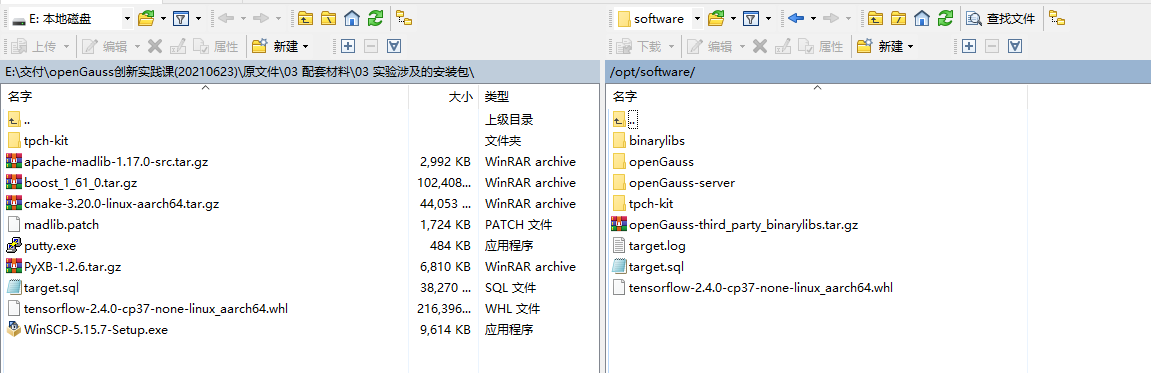
## 实验任务及步骤

### 将X-Tuner 安装到系统中

使用root用户，安装hdf5依赖包。

[root@opengauss01 ~]# **yum -y install hdf5-devel**

将tensorflow-2.4.0-cp37-none-linux\_aarch64.whl文件上传至服务器/opt/software下(可用WinSCP)，具体如下：



然后使用root用户，安装TensorFlow。

[root@opengauss01 ~]# **cd /opt/software**

[root@opengauss01 software]# **pip3 install tensorflow-2.4.0-cp37-none-linux\_aarch64.whl -i "https://pypi.doubanio.com/simple/"**

等待数分钟后，出现Successfully回显，表明安装完成。

使用root用户，进入到xtuner的安装目录。

[root@opengauss01 software]# **cd /opt/software/openGauss/bin/dbmind/xtuner/**

设置配置文件setup.cfg。

[root@opengauss01 software]# **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**

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。



### 使用X-Tuner进行参数优化

使用omm用户登录，将环境变量加入omm用户的.bashrc文件中。

[root@opengauss01 xtuner]# **su - omm**

[omm@opengauss01 ~]$ **vi ~/.bashrc**

在文件末尾将以下内容加入并保存退出。

**export GAUSSHOME=/opt/software/openGauss**

**export PATH=$GAUSSHOME/bin:$PATH**

**export LD\_LIBRARY\_PATH=$GAUSSHOME/lib:$LD\_LIBRARY\_PATH**

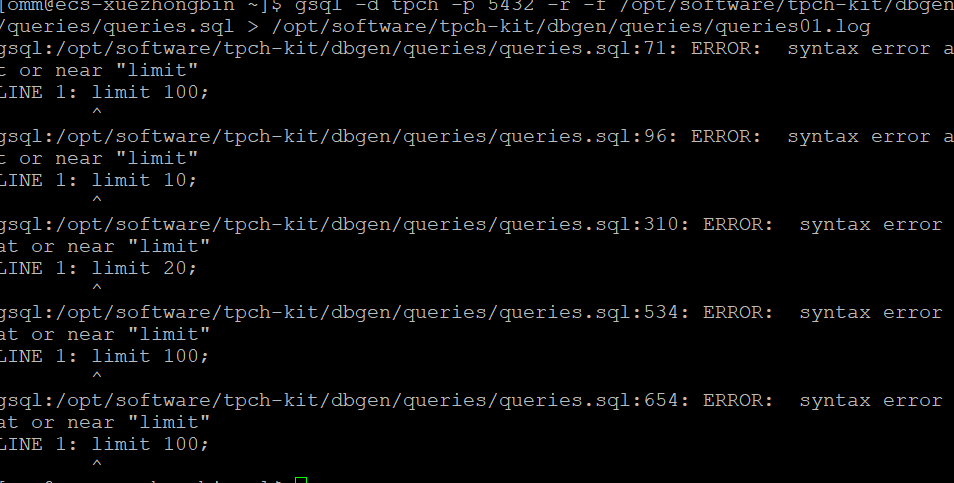
使环境变量生效。

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

执行TPCH脚本，获得测试时间，**将执行结果截图**。

[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**

等待数分钟后才能执行完成。



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**：

total time: 581509 ms

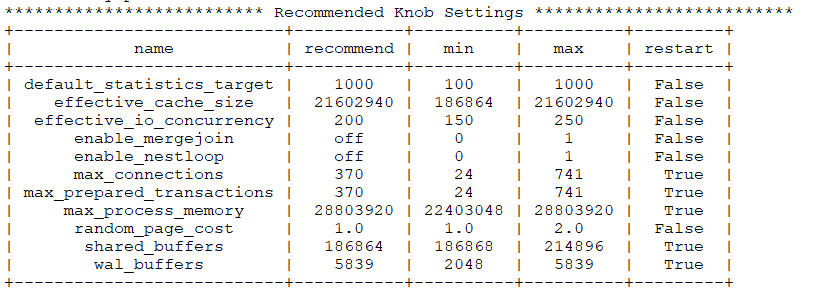
使用root用户，执行X-Tuner进行参数建议优化，**将执行结果截图**。

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：

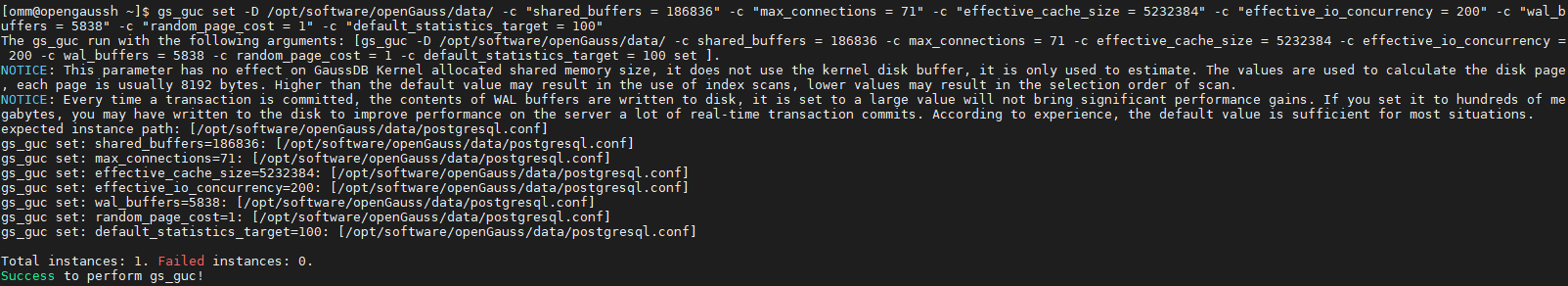


使用omm用户，对以上参数进行优化。

[root@opengauss01 xtuner]# **su - omm**

[omm@opengauss01 ~]$ **gs\_guc set -D /opt/software/openGauss/data/ -c "shared\_buffers = 186864" -c "max\_connections = 370" -c "effective\_cache\_size = 21602940" -c "effective\_io\_concurrency = 200" -c "wal\_buffers = 5839" -c "random\_page\_cost = 1" -c "default\_statistics\_target = 1000"**

返回结果为：



使用omm用户，重启数据库。

[omm@opengauss01 ~]$ **gs\_ctl stop**

[omm@opengauss01 ~]$ **gs\_ctl start -D /opt/software/openGauss/data -Z single\_node -l logfile**

获取参数值，**将结果截图**。

[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\_cost|default\_statistics\_target'**

### Index-advisor：索引推荐

登录数据库。

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

使用SQL查询2020年3月订单收入，并进行排序。

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;**

使用explain，对该SQL加以分析，**将结果截图**。

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;**

使用索引推荐功能，对查询语句进行推荐，**将执行结果截图**。

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');**

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)');**

查看创建的虚拟索引列，**将执行结果截图**。

tpch=# **select \* from hypopg\_display\_index();**

获取索引虚拟列大小结果（单位为：字节），**将执行结果截图**。

其中16715和16716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(16715);**

tpch=# **select \* from hypopg\_estimate\_size(16716);**

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;**

再次使用explain，对该SQL加以分析，**将执行结果截图**。

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;**

删除某一个索引虚拟列，**将执行结果截图**。

tpch=# **select \* from hypopg\_drop\_index(16715);**

其中16715为前面查询到的indexrelid。

删除所有索引虚拟列，**将执行结果截图**。

tpch=# **select \* from hypopg\_reset\_index();**

查看索引虚拟列，**将执行结果截图**。

tpch=# **select \* from hypopg\_display\_index();**

针对执行的queries.sql进行优化，为customer、order、part表增加主键索引，学员也可以针对queries.sql中的查询内容进行进一步优化。

tpch=# **ALTER TABLE "public"."customer" ADD CONSTRAINT "pk\_customer" PRIMARY KEY ("c\_custkey");**

tpch=# **ALTER TABLE "public"."orders" ADD CONSTRAINT "pk\_orders" PRIMARY KEY ("o\_orderkey");**

tpch=# **ALTER TABLE "public"."part" ADD CONSTRAINT "pk\_part" PRIMARY KEY ("p\_partkey");**

在order表的o\_orderdate列，以及lineitem表的l\_shipdate列上创建索引，可以通过分析queries.sql查询，进行更深入的优化。

tpch=# **CREATE INDEX idx\_o\_orderdate ON orders(o\_orderdate);**

tpch=# **CREATE INDEX idx\_l\_shipdate ON lineitem(l\_shipdate);**

重新执行queries.sql查询，**将执行结果截图**。

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**

查询queries02.log文件：

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

### 关卡验证

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，对结果进行截图；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，对创建的索引虚拟列前后的执行时间进行对比截图；完成对queries查询的优化，对比优化前后的区别。

### 思考题

根据X-Tuner给出的参数优化，在哪些参数上进行了优化，为何要对这些参数进行优化？索引的使用，对于执行SQL有什么好处？除了使用索引和参数外，还有哪些方面可以对数据库进行优化？

# 关卡四：openGauss的原生DB4AI引擎

## 实验介绍

### 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。

### 实验目的

掌握openGauss数据库的DB4AI的功能。

## 实验任务及步骤

### 利用DB4AI原生AI引擎训练并预测模型

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。

postgres=# **DROP TABLE IF EXISTS houses;**

postgres=# **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),**

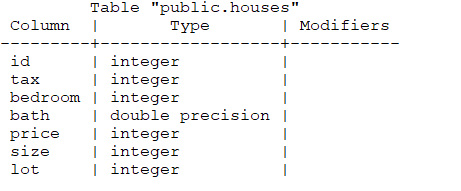
**(14 , 2070 , 2 , 3 , 148000 , 1550 , 14000),**

**(15 , 650 , 3 , 1.5 , 65000 , 1450 , 12000);**

通过 \d 命令观察新创建的表 house 结构。

postgres=# **\d houses**

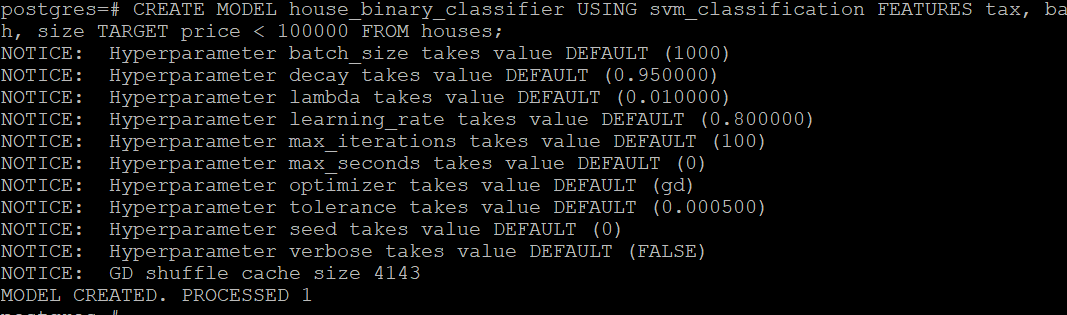
可以得到表结构信息：



通过CREATE MODEL语句，基于SVM算法创建一个二分类模型。

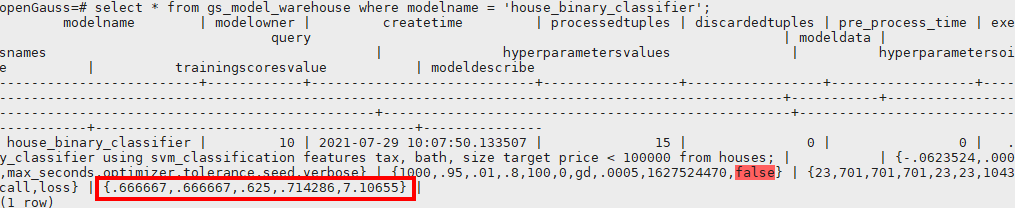
postgres=# **CREATE MODEL house\_binary\_classifier USING svm\_classification FEATURES tax, bath, size TARGET price < 100000 FROM houses;**

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：



在gs\_model\_warehouse系统表中查看训练后的模型信息，**将执行结果截图**。

postgres=# **SELECT \* FROM gs\_model\_warehouse WHERE modelname = 'house\_binary\_classifier';**



通过系统表的返回信息可以看到训练过程的预测准确率为 0.666667。

这对于二分类来说准确率并不高，我们后面将尝试通过超参数来提升模型表现。

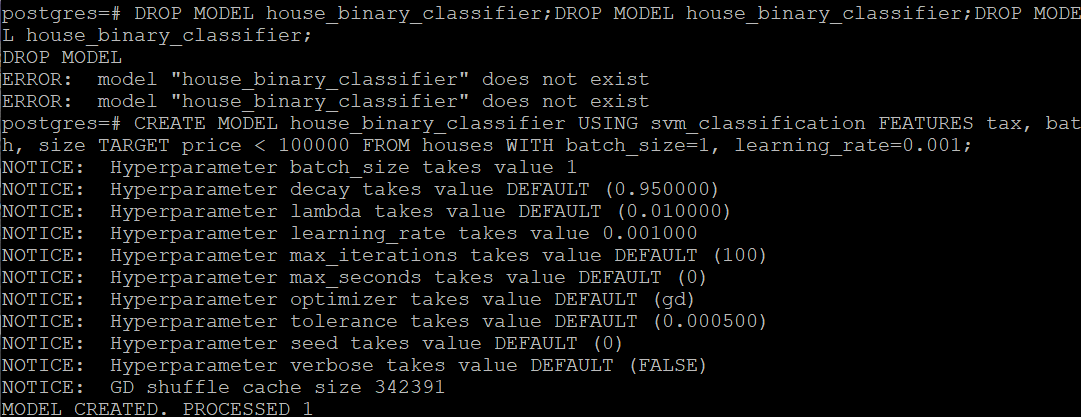
使用 DROP MODEL 语句删除模型。

postgres=# **DROP MODEL house\_binary\_classifier;**

使用更优的超参数提高SVM算法的训练表现。

postgres=# **CREATE MODEL house\_binary\_classifier USING svm\_classification FEATURES tax, bath, size TARGET price < 100000 FROM houses WITH batch\_size=1, learning\_rate=0.001;**

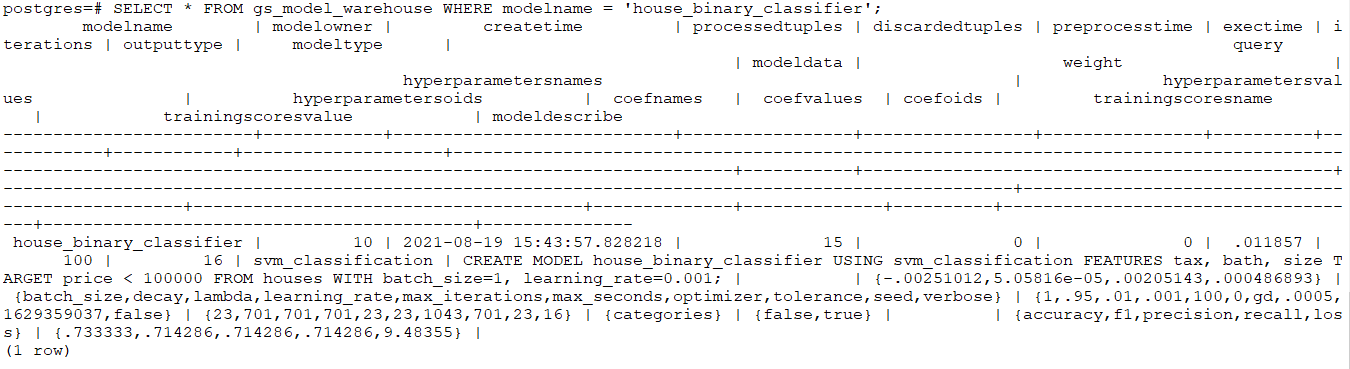
在该步骤中，修改训练时的学习率（learning\_rate）为0.001，批大小（batch\_size）设置为1。



观察新模型的信息，**将执行结果截图**。

postgres=# **SELECT \* FROM gs\_model\_warehouse WHERE modelname = 'house\_binary\_classifier';**

在返回信息中，我们可以看到新模型的表现：

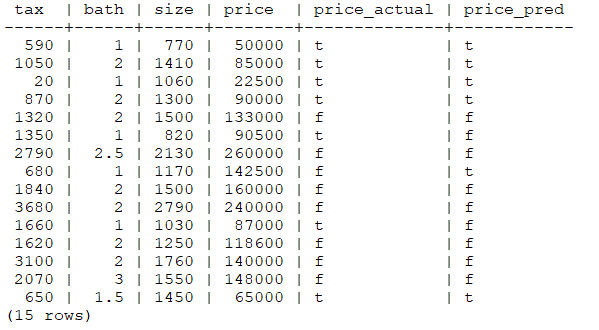


整体准确率要相对默认超参数有所提升。

使用PREDICT BY 语句预测样本数据。

postgres=# **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：



Price\_pred 列的输出结果即为预测结果，price\_actual 列的结果为真实值。

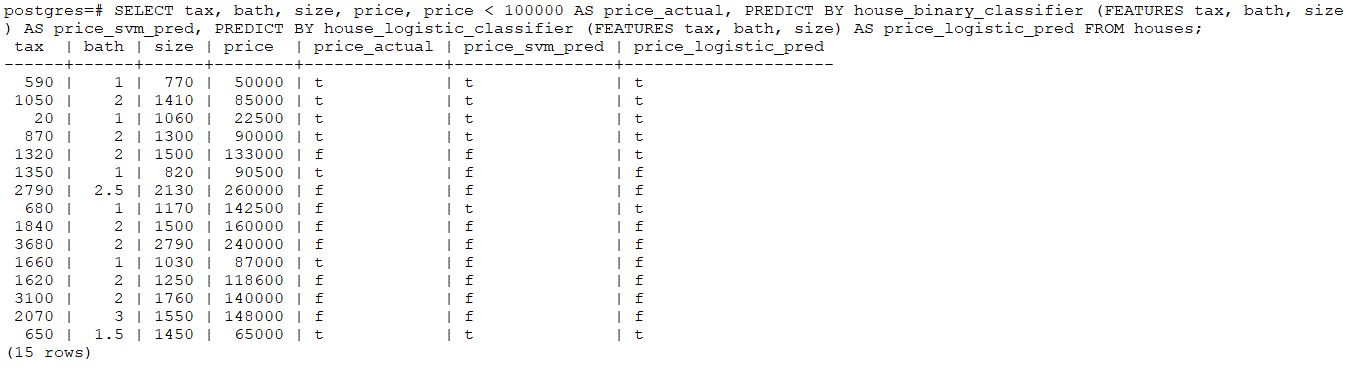
通过 CREATE MODEL 语句创建一个逻辑回归（logistic regression）模型。

postgres=# **CREATE MODEL house\_logistic\_classifier USING logistic\_regression FEATURES tax, bath, size target price < 100000 FROM houses WITH learning\_rate=0.001;**

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，**将执行结果截图**。

postgres=# **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：



### 关卡验证

完成模型训练和预测过程的截图。

### 思考题

1、分类模型与回归模型有何不同？什么是SVM算法？

2、分类问题有哪些评价指标，请分别说明他们的含义？

3、回归问题有哪些评价指标，请分别说明他们的含义？