实验三报告

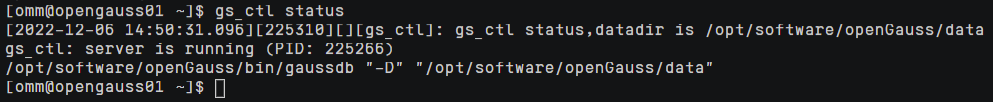
## 关卡一：openGauss数据库的编译和安装

### 关卡验证

步骤1 首先需要对数据库状态进行验证。

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

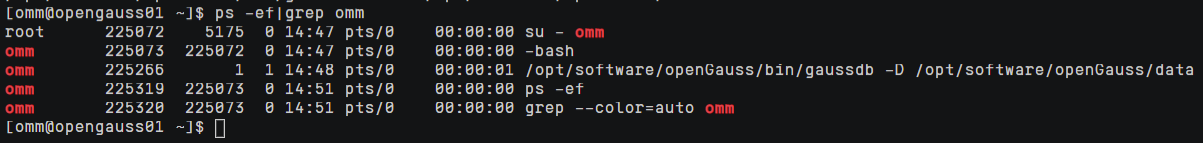
（截图语句和执行结果）



步骤2 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**

（截图语句和执行结果）



## 关卡二：openGauss数据导入及基本操作

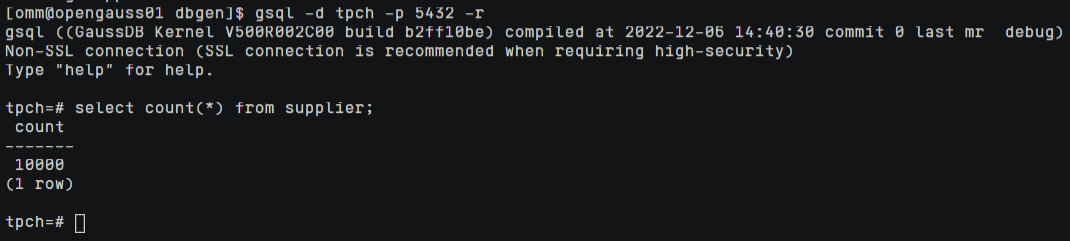
### 关卡验证

步骤12 登录数据库验证

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

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

（截图语句和执行结果）

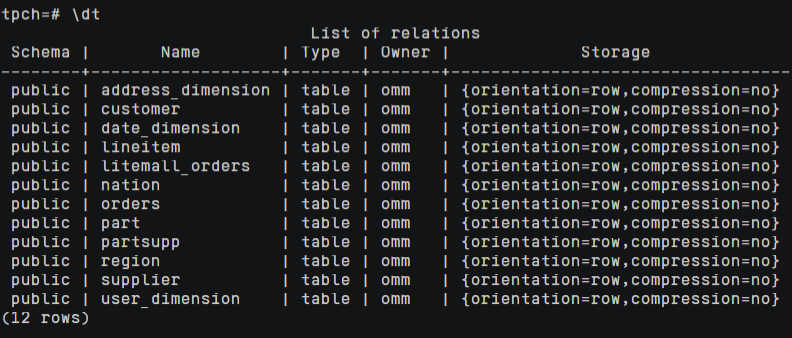


步骤21 登录数据库进行验证

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

tpch=# **\dt**

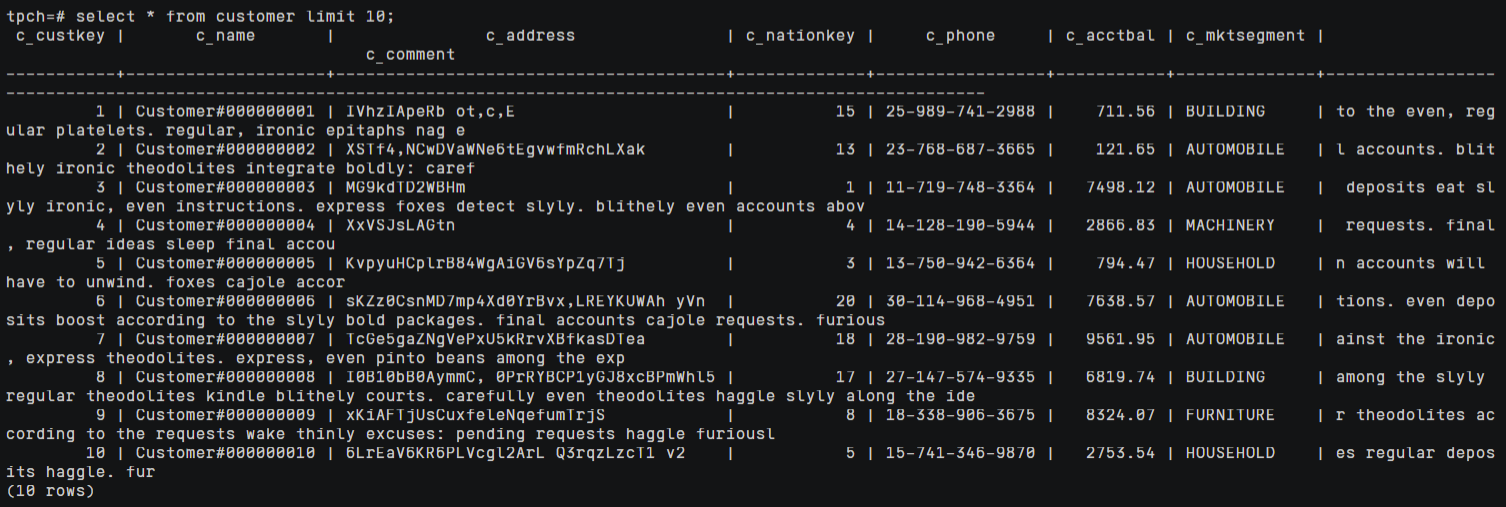
（截图语句和执行结果）



步骤22 查询customer表的数据

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

（截图语句和执行结果）



### 思考题

数据初始化中出现了TPC-H，这是什么？

TPC-H是事务处理性能委员会制定的基准程序之一，是针对随机查询/商业智能处理能力的决策支持基准，其主要目的是评价特定查询的决策支持能力，强调服务器在数据挖掘、分析处理方面的能力。该基准由一套面向业务的即时查询（ad-hoc）和并发数据修改组成，模拟了决策支持系统中的数据库操作，测试数据库系统复杂查询的响应时间，以每小时执行的查询数作为度量指标。

## 关卡三：openGauss的AI4DB特性应用

### 关卡验证

（1）使用X-Tuner进行参数优化

步骤2 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**

（截图执行语句和结果）

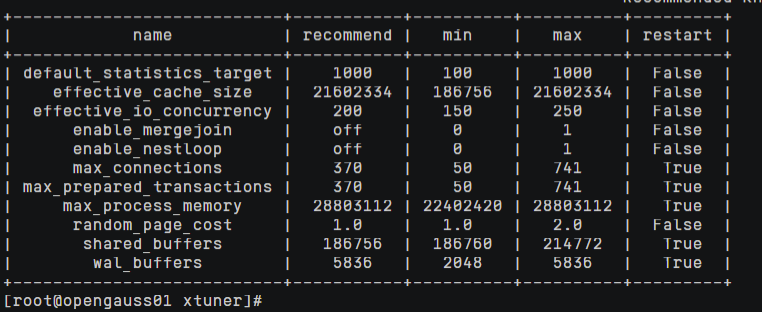


步骤3 切换至root用户，执行X-Tuner进行参数建议优化

[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**

（截图执行语句和结果）

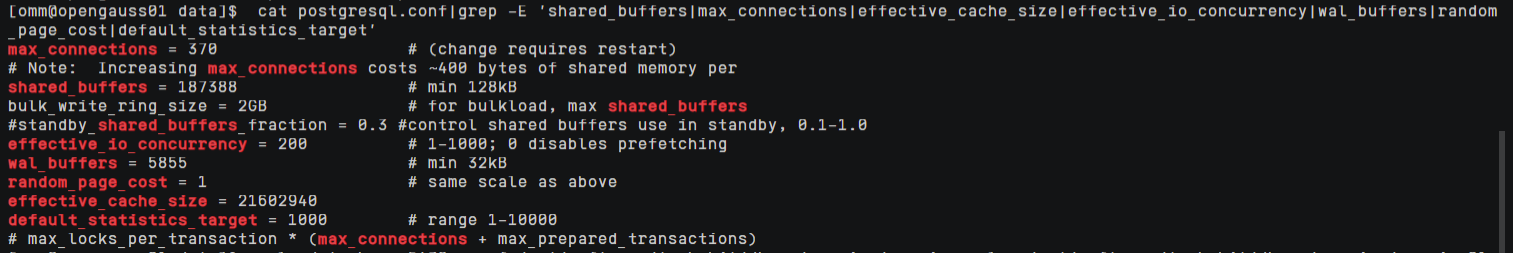


步骤6 获取参数值

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

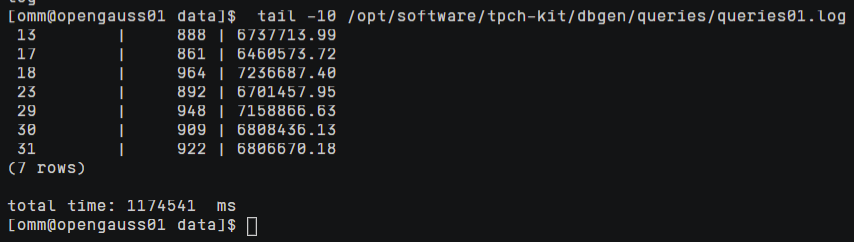
[omm@opengauss01 data]$ **cat postgresql.conf|grep -E 'shared\_buffers|max\_connections|effective\_cache\_size|effective\_io\_concurrency|wal\_buffers|random\_page\_cost|default\_statistics\_target'**

（截图执行语句和结果）



步骤7 再次执行步骤2，对比优化前的执行时间。

（截图执行语句和结果）



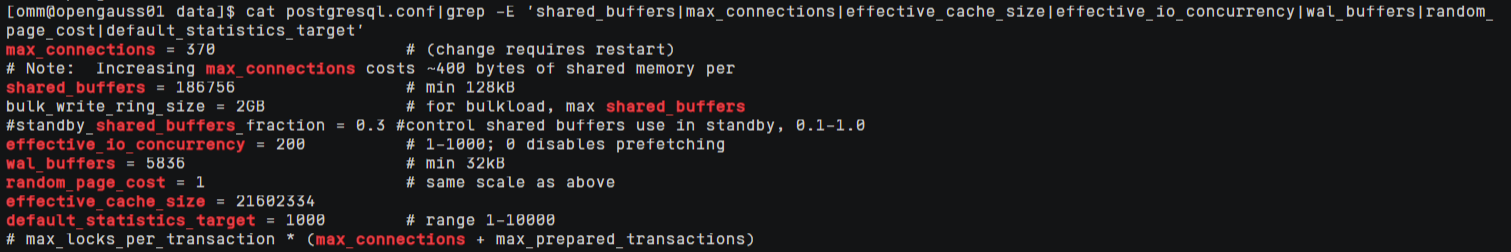
优化前的时间为1207247ms，优化后的时间为1174541ms，比优化前少了32706ms。

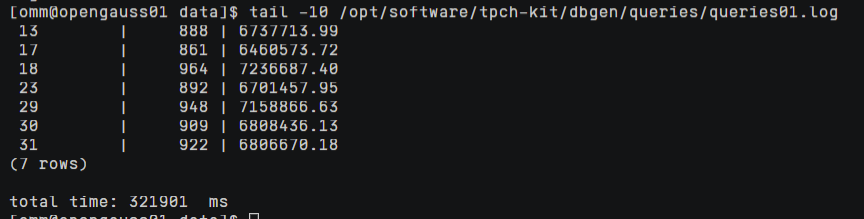
步骤8 【附加题】有兴趣的同学可以尝试并截图记录于此。

（截图执行语句和结果）

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

[omm@opengauss01 ~]$ **gs\_guc set -D /opt/software/openGauss/data/ -c "default\_statistics\_target = 1000" -c "effective\_cache\_size = 21602334" -c "effective\_io\_concurrency = 200" -c "max\_connections = 370" -c "max\_prepared\_transactions = 370" -c "max\_process\_memory=28803112" -c "random\_page\_cost = 1" -c "shared\_buffers = 186756" -c "wal\_buffers = 5836" -c "enable\_mergejoin=off" -c "enable\_nestloop=off"**





优化前的时间为1207247ms，优化后的时间为321901ms，比优化前少了885346ms。

（2）Index-advisor：索引推荐

步骤4 使用explain，对该SQL加以分析

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**

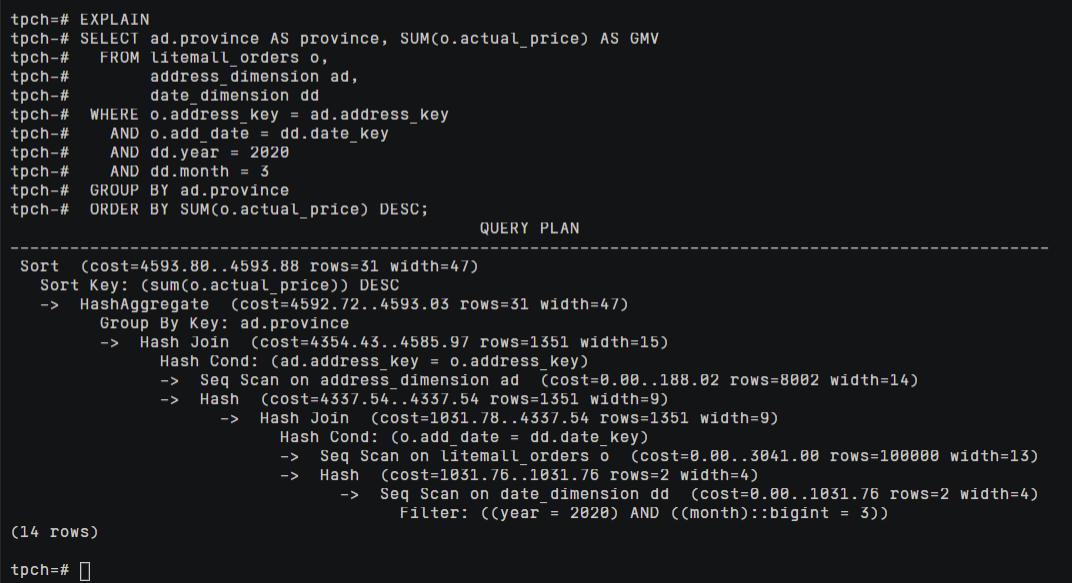
**AND dd.year = 2020**

**AND dd.month = 3**

**GROUP BY ad.province**

**ORDER BY SUM(o.actual\_price) DESC;**

（截图执行语句和结果）



步骤10 使用explain，对该SQL加以分析

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**

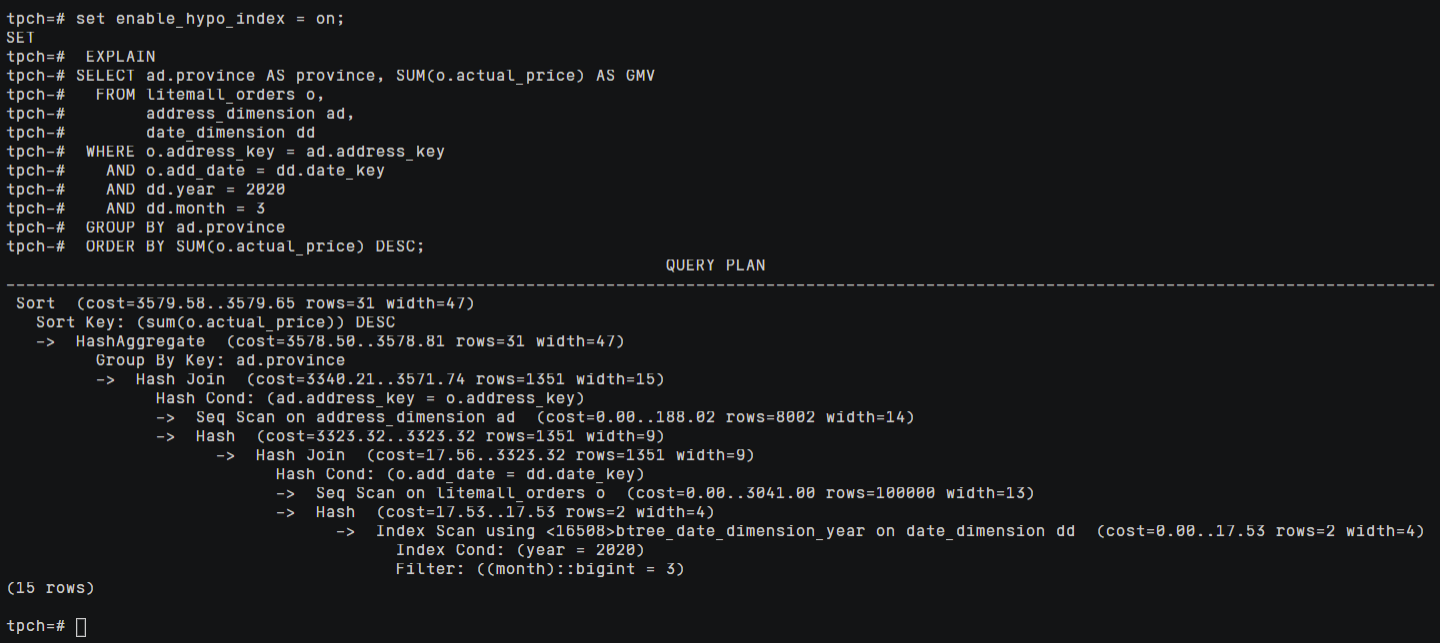
**AND dd.year = 2020**

**AND dd.month = 3**

**GROUP BY ad.province**

**ORDER BY SUM(o.actual\_price) DESC;**

（截图执行语句和结果）



步骤11 【附加题】有兴趣的同学可以尝试并截图记录于此。仅需要从queries.sql文件里选择一条或多条进行索引优化即可。

（截图执行语句和结果）

选择queries.sql中的如下语句：

tpch=# **select**

**l\_returnflag,**

**l\_linestatus,**

**sum(l\_quantity) as sum\_qty,**

**sum(l\_extendedprice) as sum\_base\_price,**

**sum(l\_extendedprice \* (1 - l\_discount)) as sum\_disc\_price,**

**sum(l\_extendedprice \* (1 - l\_discount) \* (1 + l\_tax)) as sum\_charge,**

**avg(l\_quantity) as avg\_qty,**

**avg(l\_extendedprice) as avg\_price,**

**avg(l\_discount) as avg\_disc,**

**count(\*) as count\_order**

**from**

**lineitem**

**where**

**l\_shipdate <= date '1998-12-01' - interval '90' day**

**group by**

**l\_returnflag,**

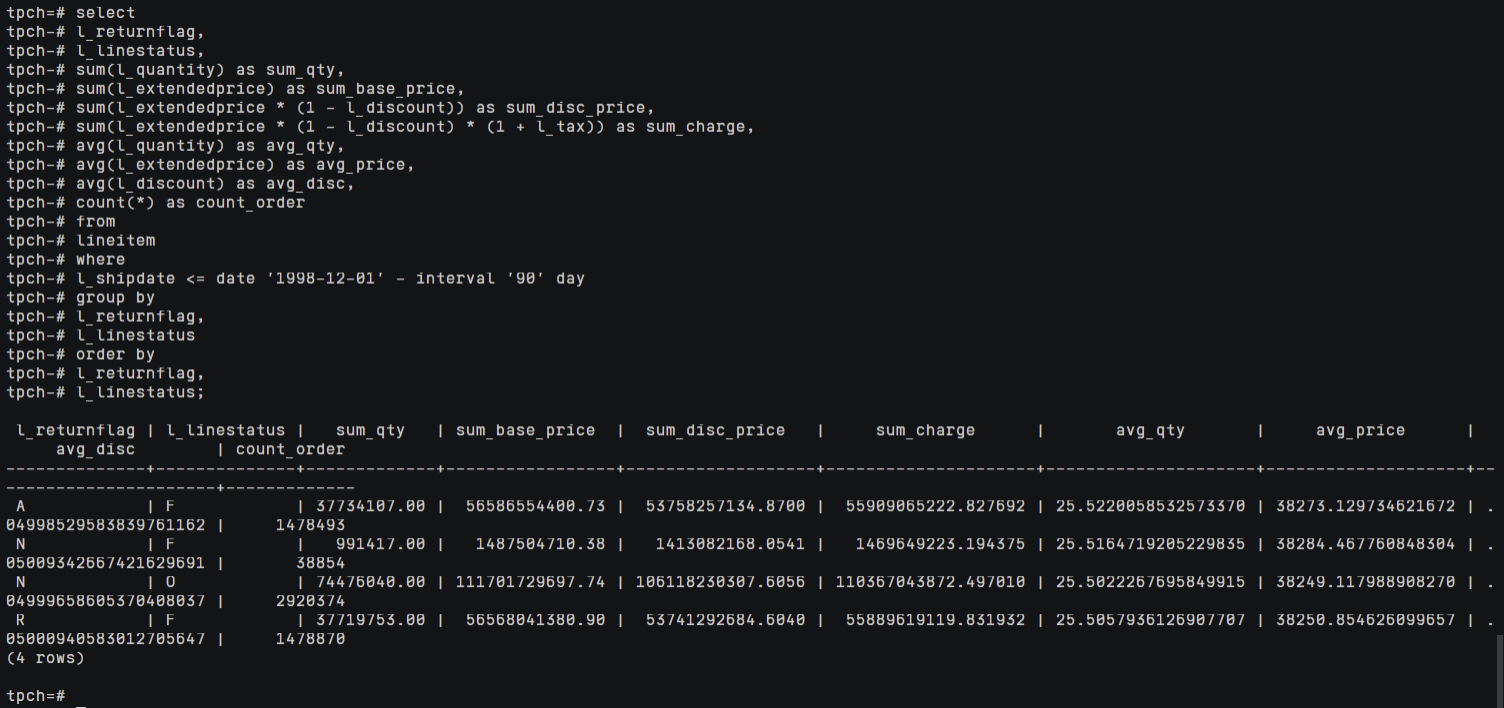
**l\_linestatus**

**order by**

**l\_returnflag,**

**l\_linestatus;**

耗时较长，执行结果如下：



使用explain，对该SQL加以分析

tpch=# **EXPLAIN**

**select**

**l\_returnflag,**

**l\_linestatus,**

**sum(l\_quantity) as sum\_qty,**

**sum(l\_extendedprice) as sum\_base\_price,**

**sum(l\_extendedprice \* (1 - l\_discount)) as sum\_disc\_price,**

**sum(l\_extendedprice \* (1 - l\_discount) \* (1 + l\_tax)) as sum\_charge,**

**avg(l\_quantity) as avg\_qty,**

**avg(l\_extendedprice) as avg\_price,**

**avg(l\_discount) as avg\_disc,**

**count(\*) as count\_order**

**from**

**lineitem**

**where**

**l\_shipdate <= date '1998-12-01' - interval '90' day**

**group by**

**l\_returnflag,**

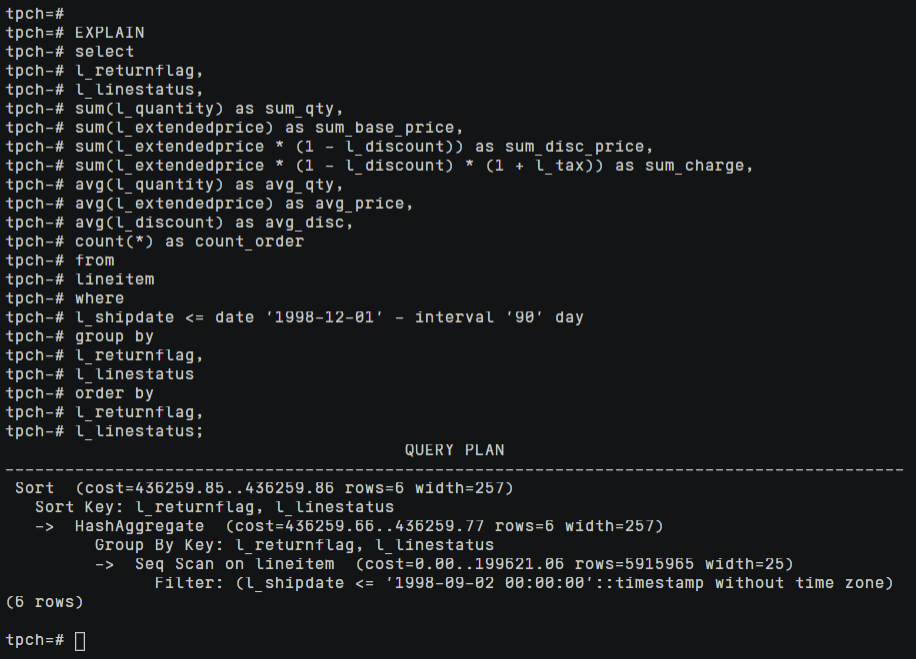
**l\_linestatus**

**order by**

**l\_returnflag,**

**l\_linestatus;**

获得执行计划结果为：



使用索引推荐功能，对查询语句进行推荐：

tpch=# **select \* from gs\_index\_advise('**

**select**

**l\_returnflag,**

**l\_linestatus,**

**sum(l\_quantity) as sum\_qty,**

**sum(l\_extendedprice) as sum\_base\_price,**

**sum(l\_extendedprice \* (1 - l\_discount)) as sum\_disc\_price,**

**sum(l\_extendedprice \* (1 - l\_discount) \* (1 + l\_tax)) as sum\_charge,**

**avg(l\_quantity) as avg\_qty,**

**avg(l\_extendedprice) as avg\_price,**

**avg(l\_discount) as avg\_disc,**

**count(\*) as count\_order**

**from**

**lineitem**

**where**

**l\_shipdate <= date ''1998-12-01'' - interval ''90'' day**

**group by**

**l\_returnflag,**

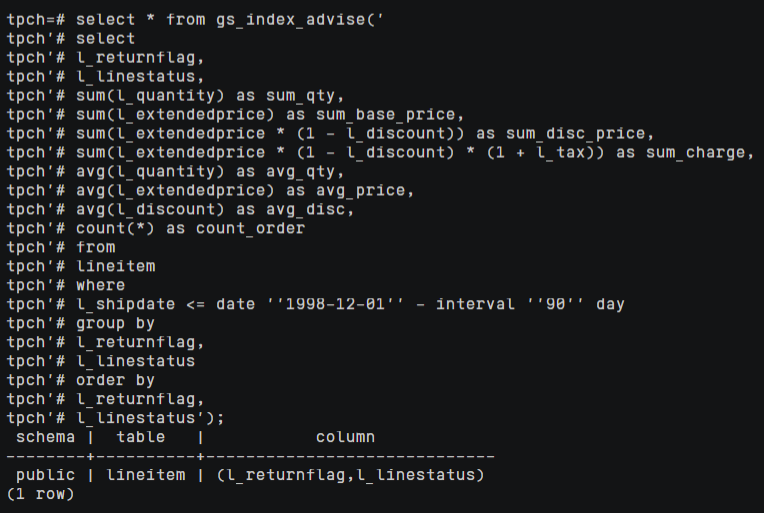
**l\_linestatus**

**order by**

**l\_returnflag,**

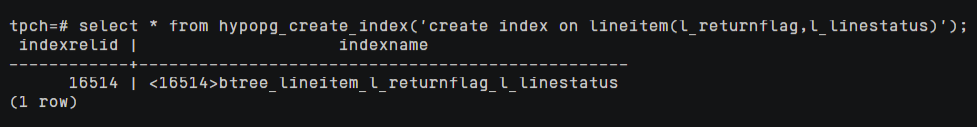
**l\_linestatus');**

获得索引推荐结果如下：



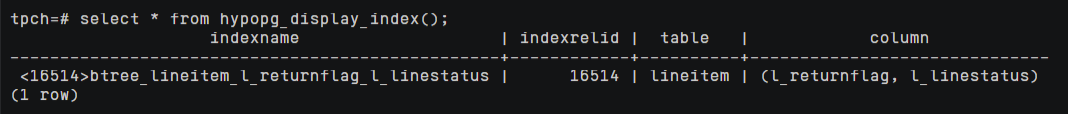
在lineitem表上创建虚拟索引列。

tpch=# **select \* from hypopg\_create\_index('create index on lineitem(l\_returnflag,l\_linestatus)');**



查看创建的虚拟索引列。

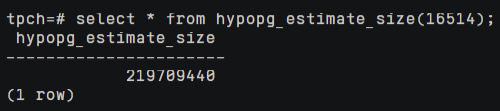
tpch=# **select \* from hypopg\_display\_index();**



获取索引虚拟列大小结果

tpch=# **select \* from hypopg\_estimate\_size(16514);**

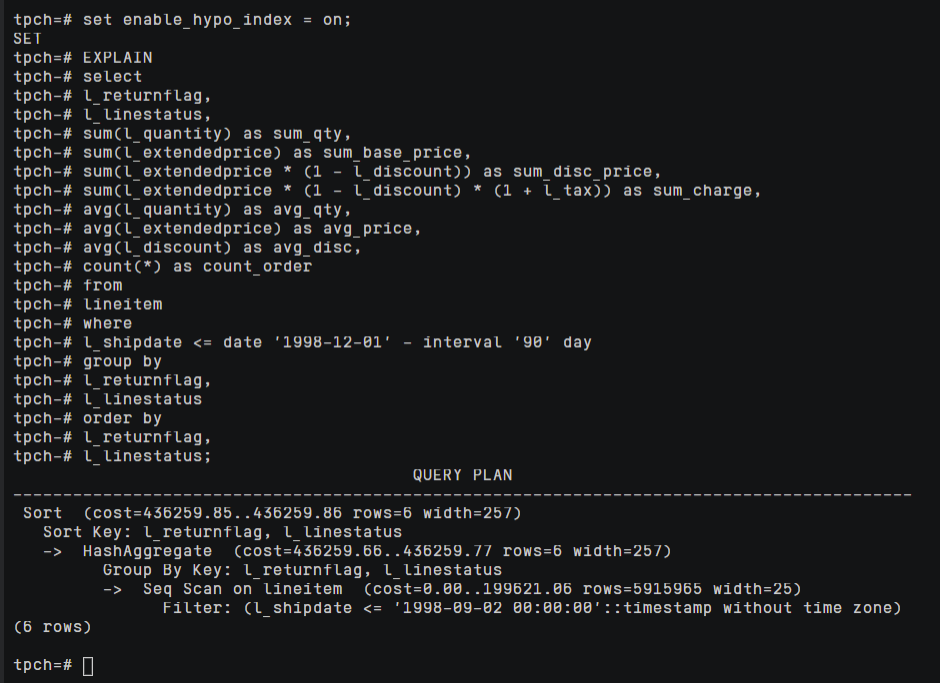
返回结果为：



开启GUC参数enable\_hypo\_index。

tpch=# **set enable\_hypo\_index = on;**

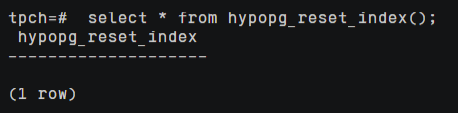
再次使用explain，对刚才执行的SQL加以分析



可以看出该语句并未奏效，分析发现，这是由于该语句仅是单表操作。

清理创建的索引虚拟列：

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



因此挑选多表操作的语句：

tpch=# **select**

**s\_acctbal,**

**s\_name,**

**n\_name,**

**p\_partkey,**

**p\_mfgr,**

**s\_address,**

**s\_phone,**

**s\_comment**

**from**

**part,**

**supplier,**

**partsupp,**

**nation,**

**region**

**where**

**p\_partkey = ps\_partkey**

**and s\_suppkey = ps\_suppkey**

**and p\_size = 15**

**and p\_type like '%BRASS'**

**and s\_nationkey = n\_nationkey**

**and n\_regionkey = r\_regionkey**

**and r\_name = 'EUROPE'**

**and ps\_supplycost = (**

**select**

**min(ps\_supplycost)**

**from**

**partsupp,**

**supplier,**

**nation,**

**region**

**where**

**p\_partkey = ps\_partkey**

**and s\_suppkey = ps\_suppkey**

**and s\_nationkey = n\_nationkey**

**and n\_regionkey = r\_regionkey**

**and r\_name = 'EUROPE'**

**)**

**order by**

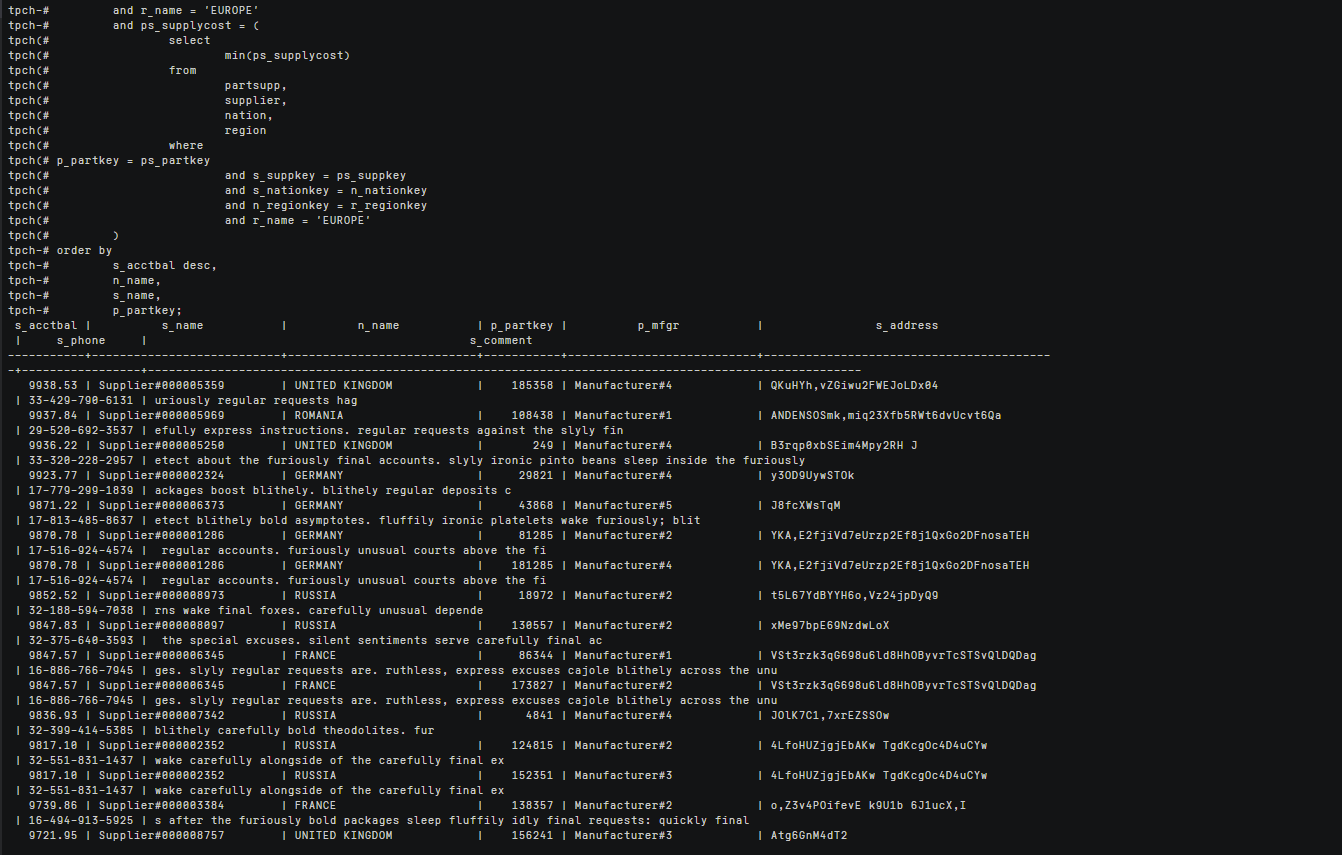
**s\_acctbal desc,**

**n\_name,**

**s\_name,**

**p\_partkey;**

**;**



使用explain，对该SQL加以分析

tpch=# **EXPLAIN**

**select**

**s\_acctbal,**

**s\_name,**

**n\_name,**

**p\_partkey,**

**p\_mfgr,**

**s\_address,**

**s\_phone,**

**s\_comment**

**from**

**part,**

**supplier,**

**partsupp,**

**nation,**

**region**

**where**

**p\_partkey = ps\_partkey**

**and s\_suppkey = ps\_suppkey**

**and p\_size = 15**

**and p\_type like '%BRASS'**

**and s\_nationkey = n\_nationkey**

**and n\_regionkey = r\_regionkey**

**and r\_name = 'EUROPE'**

**and ps\_supplycost = (**

**select**

**min(ps\_supplycost)**

**from**

**partsupp,**

**supplier,**

**nation,**

**region**

**where**

**p\_partkey = ps\_partkey**

**and s\_suppkey = ps\_suppkey**

**and s\_nationkey = n\_nationkey**

**and n\_regionkey = r\_regionkey**

**and r\_name = 'EUROPE'**

**)**

**order by**

**s\_acctbal desc,**

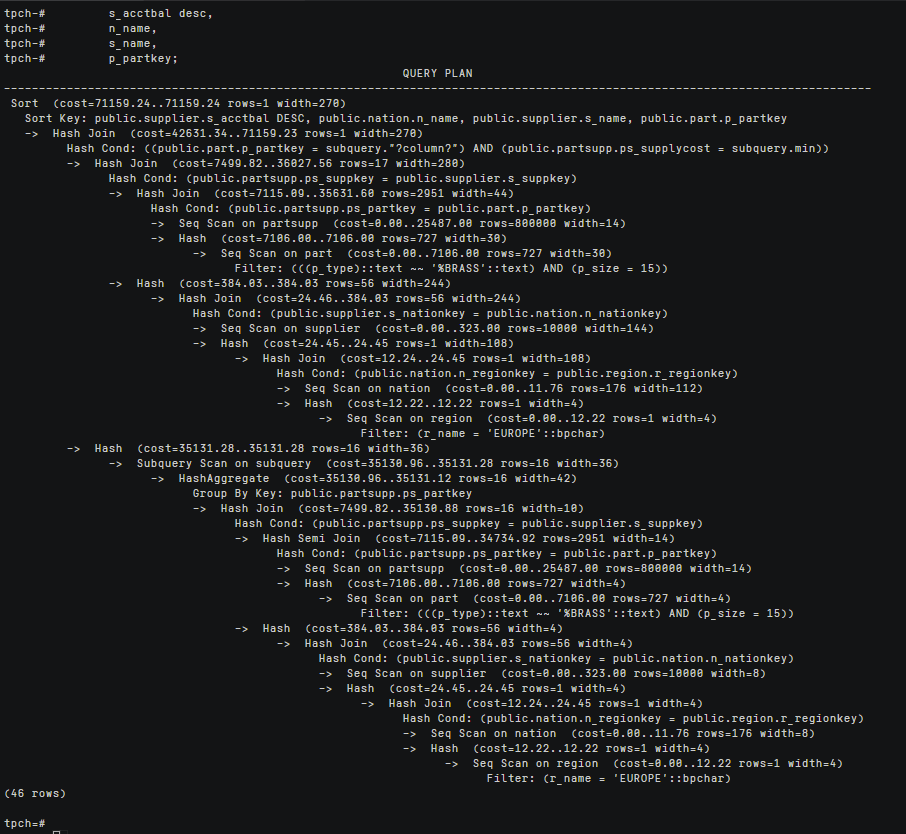
**n\_name,**

**s\_name,**

**p\_partkey;**

**;**

获得执行计划结果为：



使用索引推荐功能，对查询语句进行推荐：

tpch=# **select \* from gs\_index\_advise(' select**

**s\_acctbal,**

**s\_name,**

**n\_name,**

**p\_partkey,**

**p\_mfgr,**

**s\_address,**

**s\_phone,**

**s\_comment**

**from**

**part,**

**supplier,**

**partsupp,**

**nation,**

**region**

**where**

**p\_partkey = ps\_partkey**

**and s\_suppkey = ps\_suppkey**

**and p\_size = 15**

**and p\_type like ''%BRASS''**

**and s\_nationkey = n\_nationkey**

**and n\_regionkey = r\_regionkey**

**and r\_name = ''EUROPE''**

**and ps\_supplycost = (**

**select**

**min(ps\_supplycost)**

**from**

**partsupp,**

**supplier,**

**nation,**

**region**

**where**

**p\_partkey = ps\_partkey**

**and s\_suppkey = ps\_suppkey**

**and s\_nationkey = n\_nationkey**

**and n\_regionkey = r\_regionkey**

**and r\_name = ''EUROPE''**

**)**

**order by**

**s\_acctbal desc,**

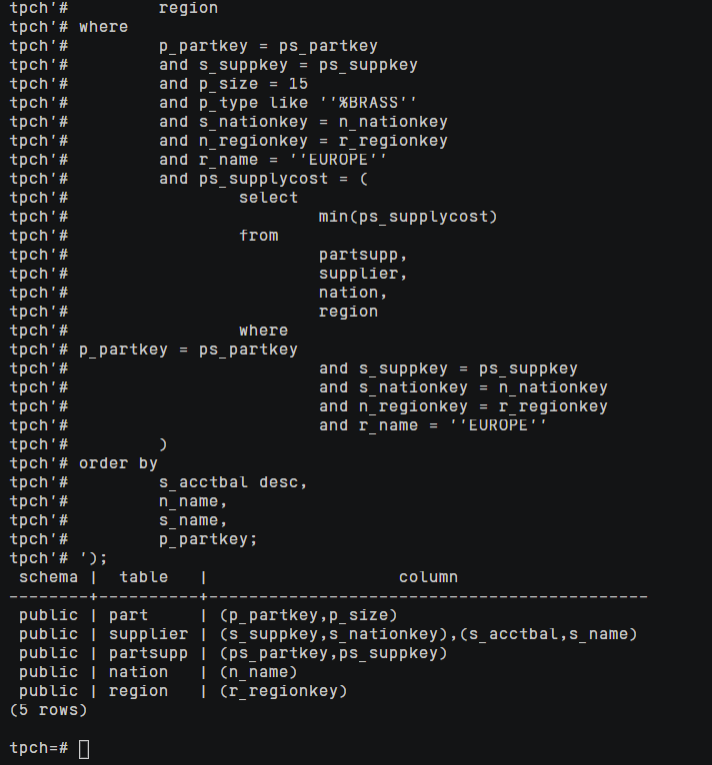
**n\_name,**

**s\_name,**

**p\_partkey;**

**');**

获得索引推荐结果如下：



在part、supplier、partsupp、nation和region表上创建虚拟索引列：

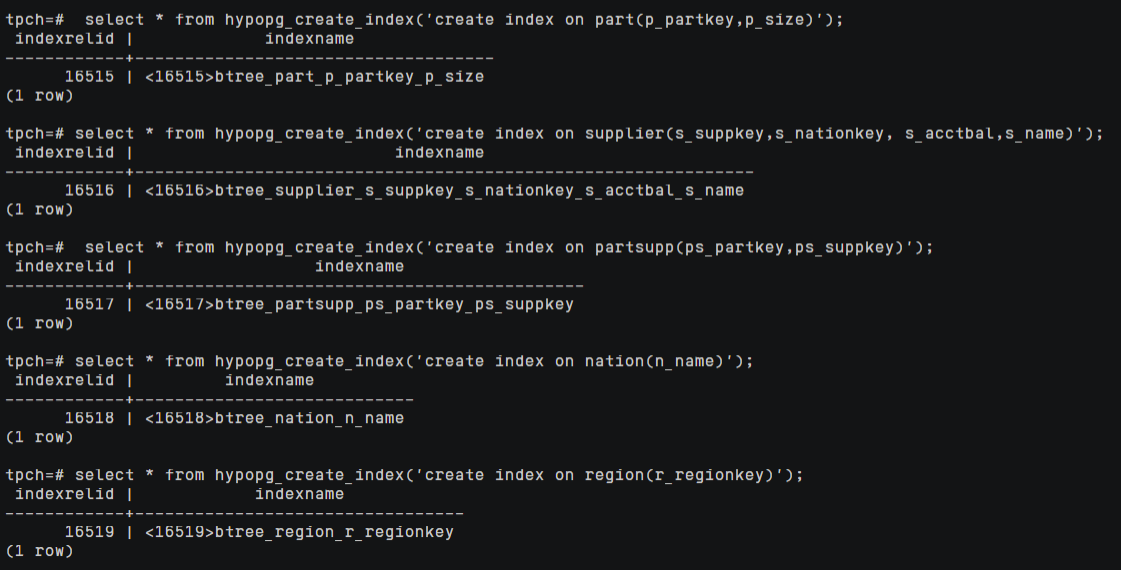
tpch=# **select \* from hypopg\_create\_index('create index on part(p\_partkey,p\_size)');**

tpch=# **select \* from hypopg\_create\_index('create index on supplier(s\_suppkey,s\_nationkey, s\_acctbal,s\_name)');**

tpch=# **select \* from hypopg\_create\_index('create index on partsupp(ps\_partkey,ps\_suppkey)');**

tpch=# **select \* from hypopg\_create\_index('create index on nation(n\_name)');**

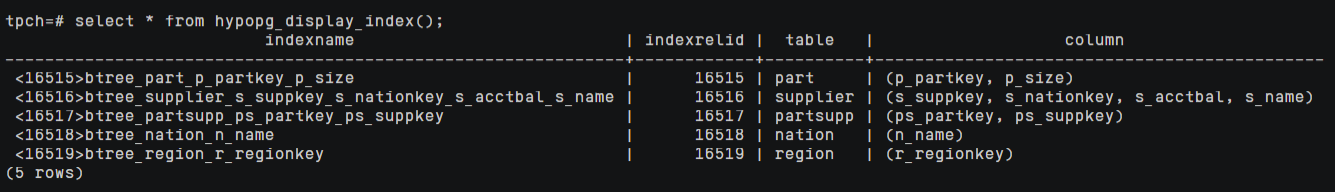
tpch=# **select \* from hypopg\_create\_index('create index on region(r\_regionkey)');**



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：



获取索引虚拟列大小结果

tpch=# **select \* from hypopg\_estimate\_size(16515);**

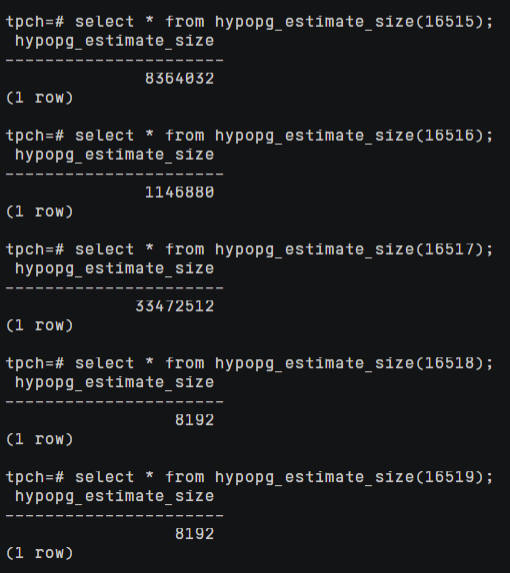
tpch=# **select \* from hypopg\_estimate\_size(16516);**

tpch=# **select \* from hypopg\_estimate\_size(16517);**

tpch=# **select \* from hypopg\_estimate\_size(16518);**

tpch=# **select \* from hypopg\_estimate\_size(16519);**

返回结果为：



开启GUC参数enable\_hypo\_index：

tpch=# **set enable\_hypo\_index = on;**

再次使用explain，对刚才执行的SQL加以分析

tpch=# **EXPLAIN**

**select**

**s\_acctbal,**

**s\_name,**

**n\_name,**

**p\_partkey,**

**p\_mfgr,**

**s\_address,**

**s\_phone,**

**s\_comment**

**from**

**part,**

**supplier,**

**partsupp,**

**nation,**

**region**

**where**

**p\_partkey = ps\_partkey**

**and s\_suppkey = ps\_suppkey**

**and p\_size = 15**

**and p\_type like '%BRASS'**

**and s\_nationkey = n\_nationkey**

**and n\_regionkey = r\_regionkey**

**and r\_name = 'EUROPE'**

**and ps\_supplycost = (**

**select**

**min(ps\_supplycost)**

**from**

**partsupp,**

**supplier,**

**nation,**

**region**

**where**

**p\_partkey = ps\_partkey**

**and s\_suppkey = ps\_suppkey**

**and s\_nationkey = n\_nationkey**

**and n\_regionkey = r\_regionkey**

**and r\_name = 'EUROPE'**

**)**

**order by**

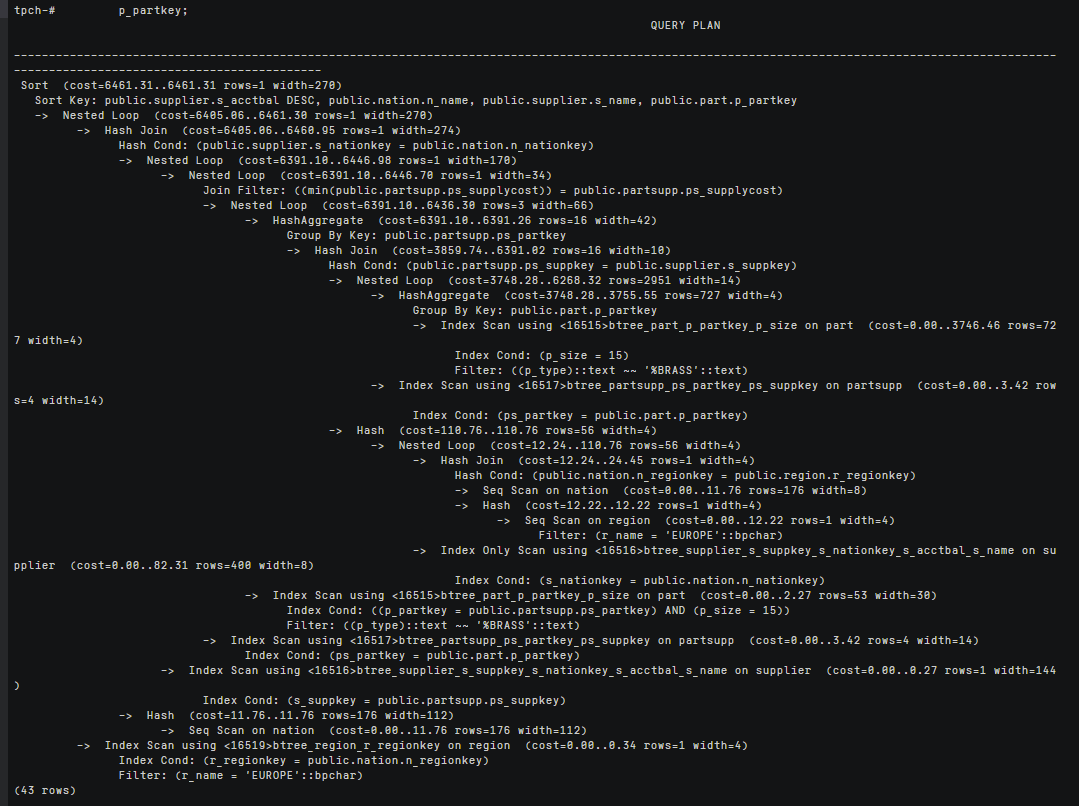
**s\_acctbal desc,**

**n\_name,**

**s\_name,**

**p\_partkey;**

**;**



结果中出现：

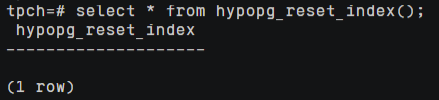
1670329124242

1670327699984

说明索引推荐已经奏效。

清理创建的索引虚拟列：

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



## 关卡四【附加题】：openGauss的DB4AI特性应用

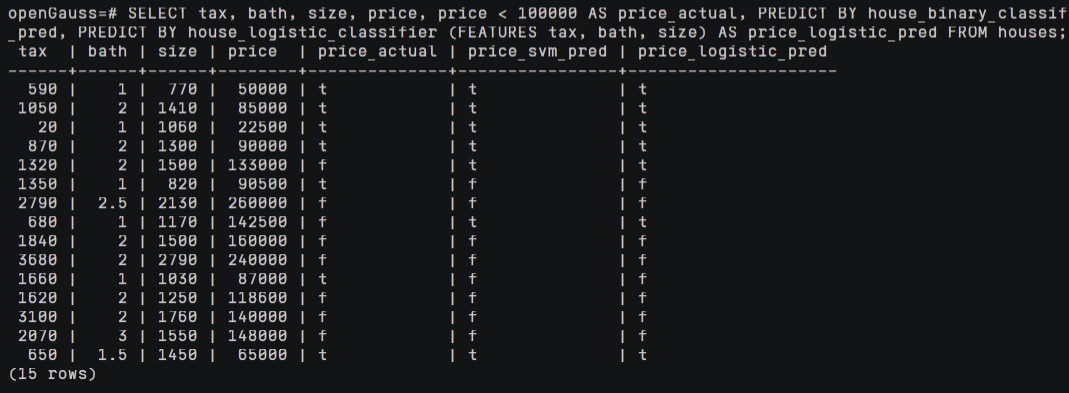
\*本关卡为附加题，有兴趣的同学可以尝试实验并记录于此。

### 关卡验证

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，将执行结果截图。

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;**

（截图执行语句和结果）



可以看出，逻辑回归模型与SVM算法的结果类似，说明某些样例较为困难，这两种方法都无法预测正确。

## 清理工作：资源释放

### 关卡验证

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。

（截图执行语句和结果）

