openGauss AI特性创新实践课



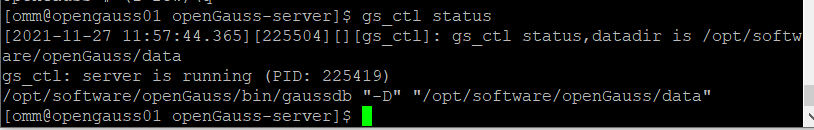
华为技术有限公司

# 关卡一、openGauss数据安装及基本操作

openGauss数据安装及基本操作, 作业提交任务如下：

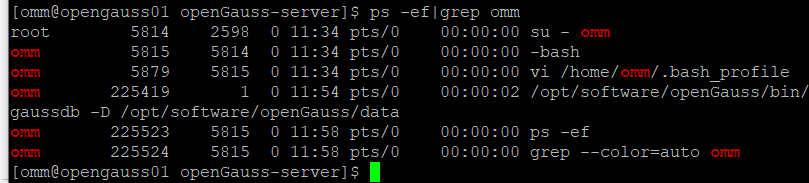
任务一：数据库状态验证

1. 查询数据库状态成功截图



任务二：数据库服务进程验证

1. 查看数据库服务进程截图（包含数据库服务器的主机名）



**实验思考题：为什么需要通过源码编译，安装数据库？**

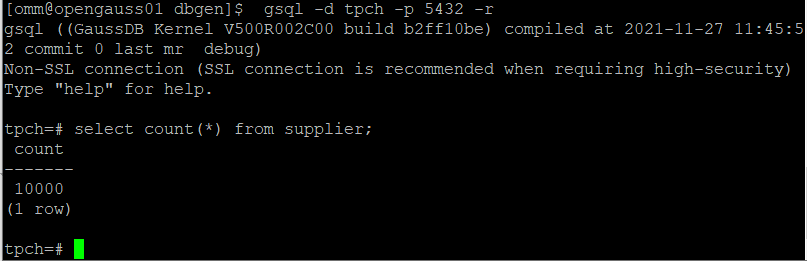
如果不通过源码编译安装数据库，则该数据库的参数都是确定的。我们想要根据自己的需求定制数据库的相关参数和功能，所以要通过源码编译安装可定制的数据库。

# 关卡二、openGauss数据导入及基本操作

任务一：数据初始化验证

1. 查询supplier表的行数，并将结果进行图：

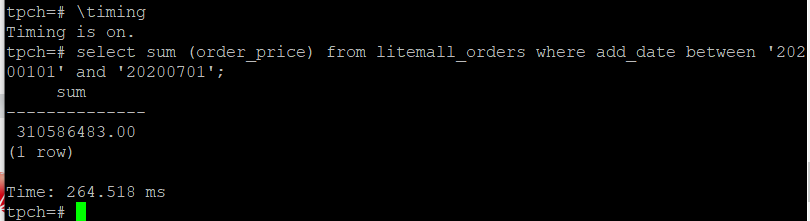
select count(\*) from supplier;;



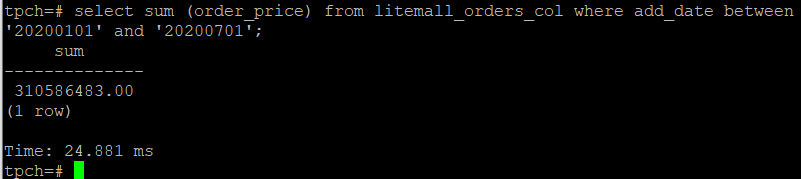
任务二：行存表与列存表执行效率对比

1. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的总和查询，并对比执行效率截图

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

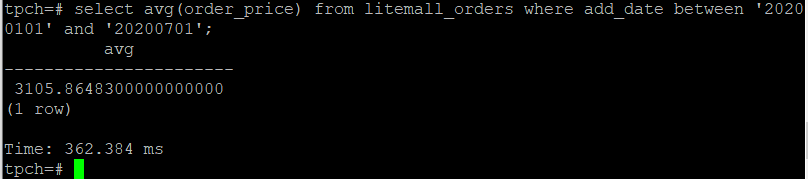


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

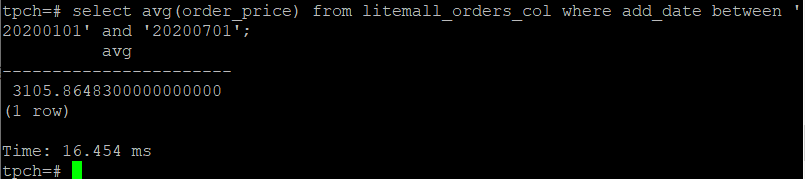


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

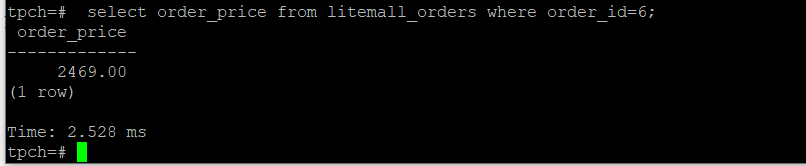


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

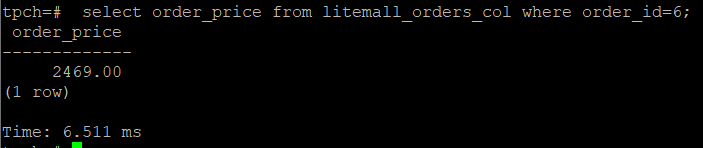


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

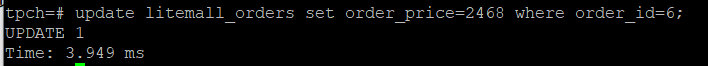


select order\_price from litemall\_orders\_col where order\_id=6;

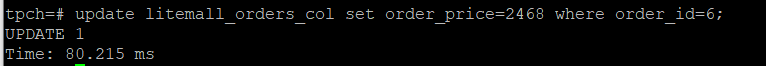


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



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



任务三：物化视图的使用

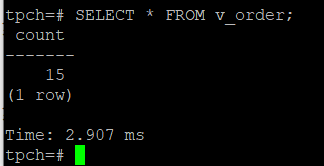
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



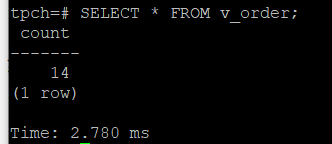
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



**实践思考题1：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？**

|  |  |  |
| --- | --- | --- |
| 1 | a | 1 |
| 2 | b | 2 |

行存：(1.a.1)(2,b,2)；列存：(1,2)(a,b)(1,2)

执行时间不同的原因是存储方式不同，行存是按顺序来存储；每一列数据都是同一类型的，以列存储，一次读取一列；

行存储关注整张表内容，更加适合需要经常更新数据、需要经常读取整行数据，而且不需要聚集运算，或者快速查询需求、或者数据表本身数据行并不多的情况，在这些情况下行存储效率较高。如果只有少量数据，那么读取过程中产生冗余数据的缺点就可以忽略，可能查询效率还会高于列存储，但数据量大时会使效率大幅降低，会存在数据冗余，时间变长。

列存储更加适合关注一张表某几列、基于一列或比较少的列计算、数据表列的数量多、数据表有非常多行数据并且需要聚集运算的时候、数据表列中有非常多重复数据的情况，因为有利于高度压缩。列存储在写入效率、保证数据完整性上不如行存储，但在读取过程不会产生冗余数据，适合对数据完整性要求不高的情况。

**实践思考题2：全量物化视图与增量物化视图有哪些差别？**

主要体现在刷新方式的区别。

全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新。

增量刷新是指以页面显示数据为基础到表中查是否有更新数据，有则更新，没有则不做操作。也就是说，只传更新过的数据，并且将他们进行更新，存储起来。

全量更新，会将所有参数都传给后台，全部入库。

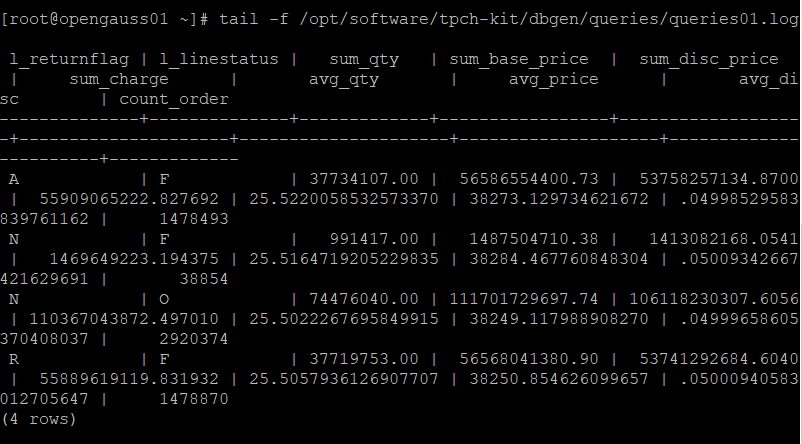
# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

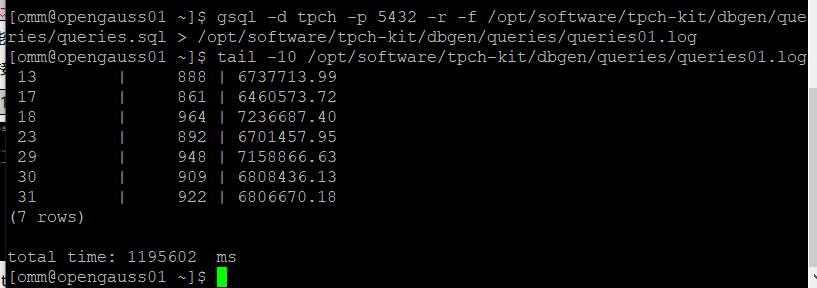
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log

新PuTTY连接窗口的截图：

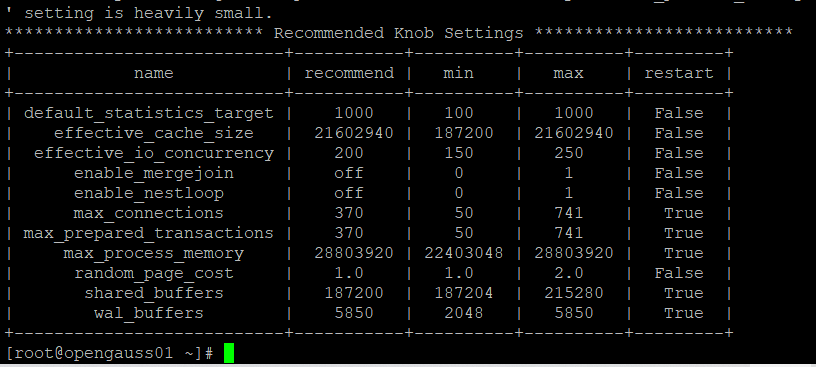


测试时间的运行结果截图：



2. 使用root用户，执行X-Tuner进行参数建议优化，将执行结果截图

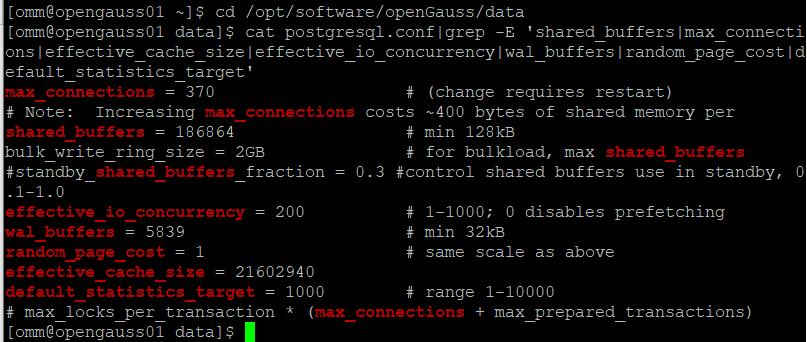
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



3.重启完成后，获取参数值：

cd /opt/software/openGauss/data

cat postgresql.conf|grep -E 'shared\_buffers|max\_connections|effective\_cache\_size|effective\_io\_concurrency|wal\_buffers|random\_page\_cost|default\_statistics\_target'



任务二：使用Index-advisor对select 查询语句进行优化，并通过对比执行计划，得到优化前后的不同。

1. 使用explain，对查询2020年3月订单表收入并进行排序的SQL加以分析，将结果截图。

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

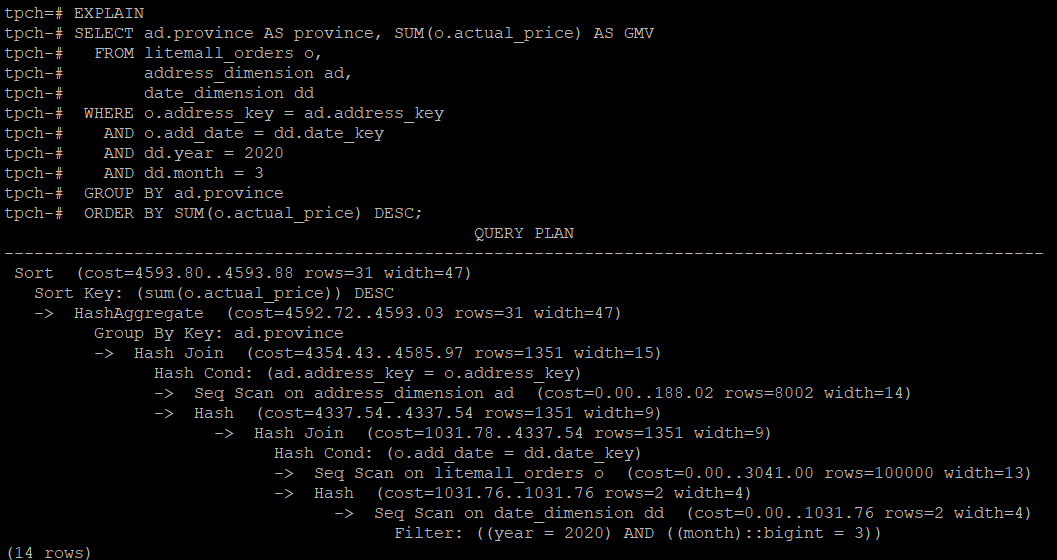
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

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



2. 使用索引推荐功能，对查询语句进行推荐，将执行结果截图。

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

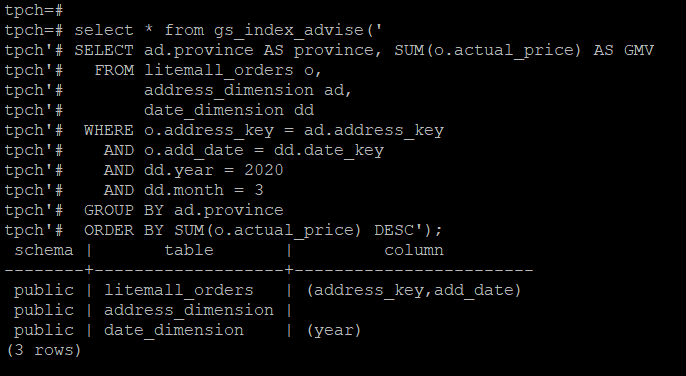
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

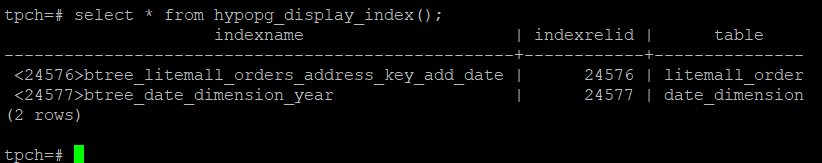
GROUP BY ad.province

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



3. 查看创建的虚拟索引列，将执行结果截图。

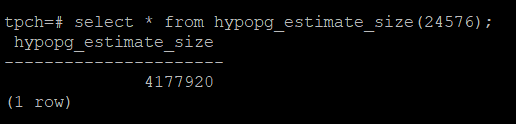
select \* from hypopg\_display\_index();

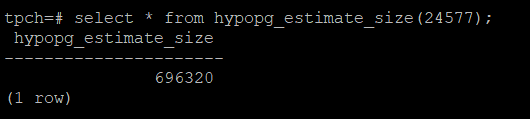


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

select \* from hypopg\_estimate\_size(16715);

select \* from hypopg\_estimate\_size(16716);





5.再次使用explain，对该SQL加以分析，将执行结果截图。

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

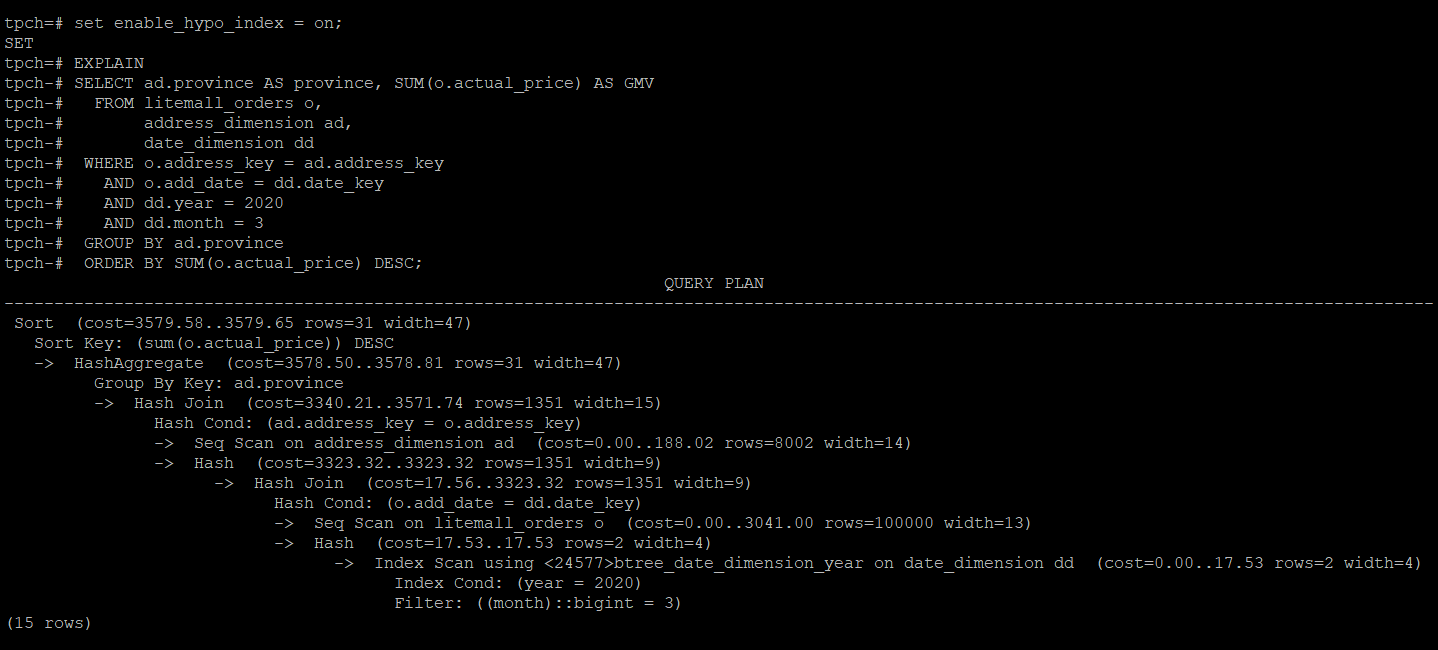
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

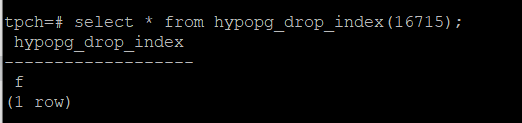
GROUP BY ad.province

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



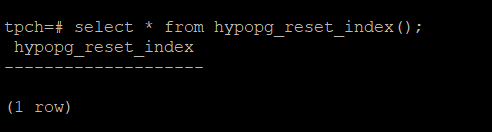
6. 删除某一个索引虚拟列，将执行结果截图。

select \* from hypopg\_drop\_index(16715);



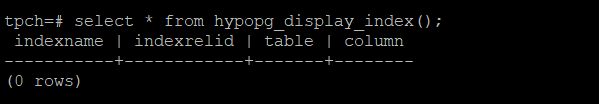
7. 删除某一个索引虚拟列，将执行结果截图。

select \* from hypopg\_reset\_index();



8. 查看索引虚拟列，将执行结果截图。

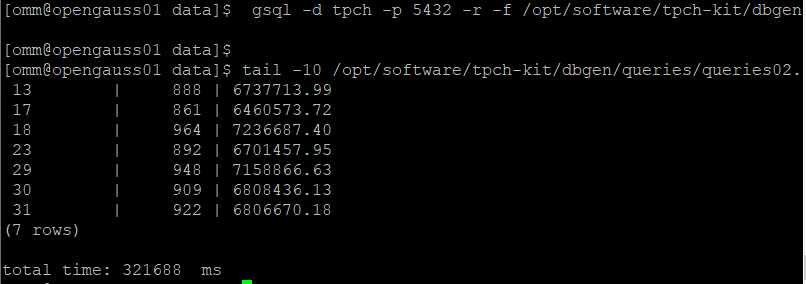
select \* from hypopg\_display\_index()



任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

1. 重新执行queries.sql查询，将执行结果截图：

gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

1. 重新执行queries.sql查询，将执行结果截图：

gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries03.log

暂时只提交基础部分；

**实践思考题1：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？**

shared\_buffers，max\_connections，effective\_cache\_size，effective\_io\_concurrency，wal\_buffers，random\_page\_cost，default\_statistics\_target.

因为我们在上一条语句执行了X-Tuner进行参数建议优化，得出了推荐进行优化的参数和推荐优化的值。我们根据gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm语句的结果进行优化。

**实践思考题2：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？**

索引提供的数据顺序不同于数据在磁盘上的物理存储顺序。索引的特殊作用是在表内重新排列记录的物理位置。索引可建立在数据表的一列上，或建立在表的几列的组合上。虽然索引增加了占用的存储空间，但当在数据量大的表中进行检索操作时，索引能够大大提高检索效率。

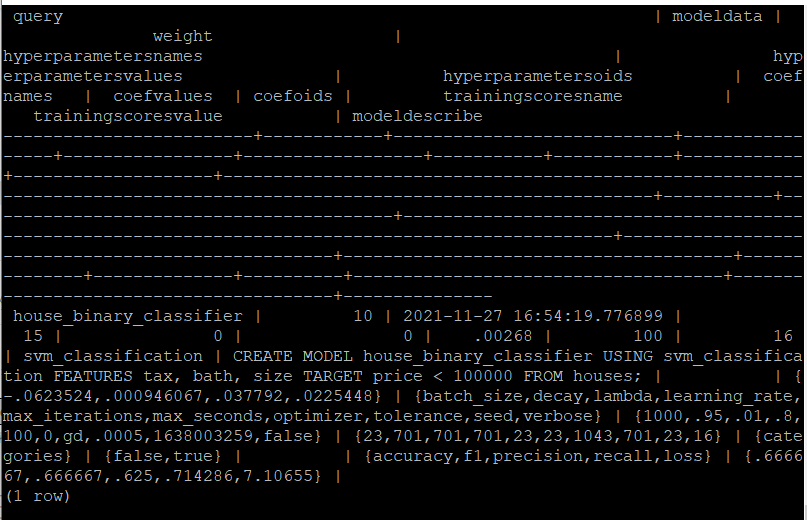
除了创建索引，我们还可以进行复合索引。由于查询每次只能使用一个索引，虽然这样已经相对不做索引时全表扫描提高了很多效率，但是如果在两列上创建复合索引的话将带来更高的效率，这也被称为最佳左前缀特性。在创建复合索引时要将最常用作限制条件的列放在最左边，依次递减。

除此之外，索引中不能有包含NULL值的列。复合索引中只要有一列含有NULL值，那么这一列对于此复合索引就是无效的。所以在对数据库设计约束时，不要让字段的默认值为NULL，以免影响效率。

# 关卡四、openGauss的DB4AI特性应用

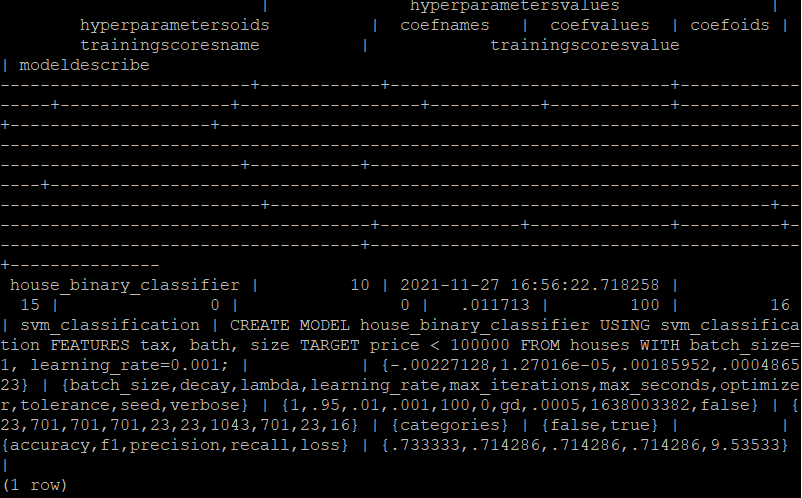
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



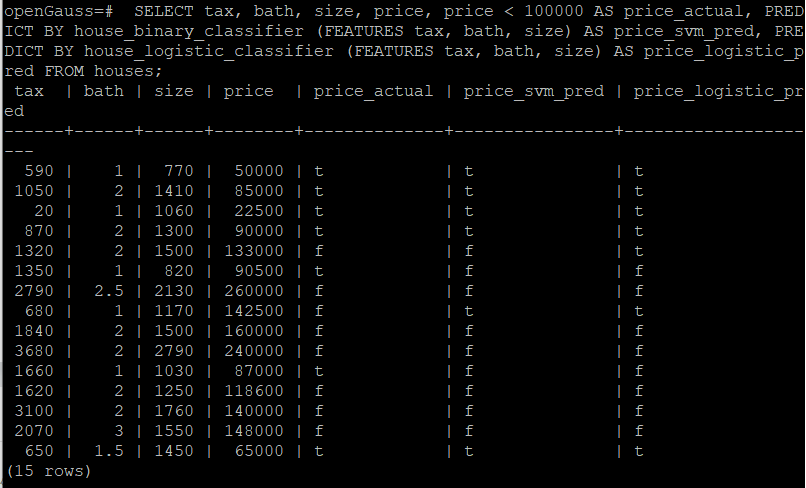
任务二：观察新模型的信息，将执行结果截图。

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



任务三：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;



**实践思考题1：分类模型与回归模型有何不同？**

1. 分类的输出是离散值，回归的输出是连续值，他们的输出类型不同
2. 分类要找到最优决策面，而回归要使得学习到的函数曲线跟真实值曲线拟合最好。
3. 分类常用的损失函数有log loss、hinge loss等，而回归一般是平方损失函数等。

**实践思考题2：什么是SVM算法？**

我们想要一条直线将样本标记为1和-1的两类样本区分开来，我们想找到这样一条最优直线。

我们先假设一条直线为W•X+b =0为最优的分割线把两类分开。所以我们就要确定这条最优直线，也就是确定W和b的值。方法就是找到SVM中的最优分割面(超平面)，也就是能使支持向量和超平面最小距离的最大值。找到这样一条直线，我们就能解决SVM问题。

**实践思考题3：分类问题有哪些评价指标，请分别说明他们的含义？**

TP：True Positives，表示实际为正例且被分类器判定为正例的样本数

FP：False Positives，表示实际为负例且被分类器判定为正例的样本数

FN：False Negatives，表示实际为正例但被分类器判定为负例的样本数

TN：True Negatives，表示实际为负例且被分类器判定为负例的样本数

以及Accuracy,Precision和Recall值：

Accuracy：对于给定的测试数据集，分类器正确分类的样本数与总样本数之比。

Precision：所有"正确被检索的item(TP)"占所有"实际被检索到的(TP+FP)"的比例。

Recall：所有"正确被检索的item(TP)"占所有"应该检索到的item(TP+FN)"的比例。

**实践思考题4：回归问题有哪些评价指标，请分别说明他们的含义？**

MSE均方误差：均方误差描述了样本真实值与预测值差方求和的平均值。

RMSE：即对MSE开平方根。

MAE平均绝对误差：真实值与预测值差的绝对值和求平均。

MAPE：求所有样本真实值与预测值差绝对值与真实值的比例的和求平均。

MSPE：求所有样本的真实值与预测值的差与真实值的比例求平方的和求平均。