openGauss AI特性创新实践课



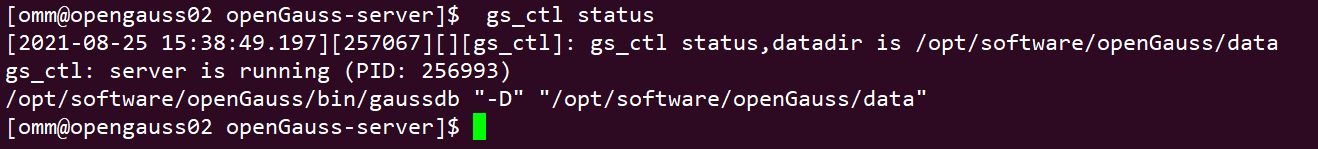
华为技术有限公司

# 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：

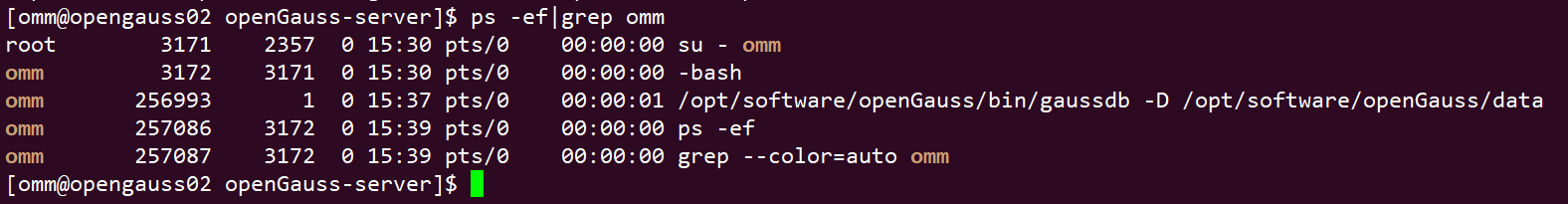
任务一：数据库状态验证

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？

源码是源程序代码，不能直接运行，必须通过对源代码的编译把源代码转化成对应的二进制可 执行程序才能运行。

数据库安装步骤：

1.下载数据库源码

2.下载依赖库

3.下载编译需要的工具

4.进行数据库配置

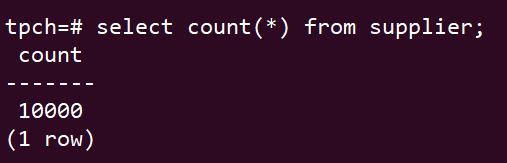
5.编译安装数据库

# 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：

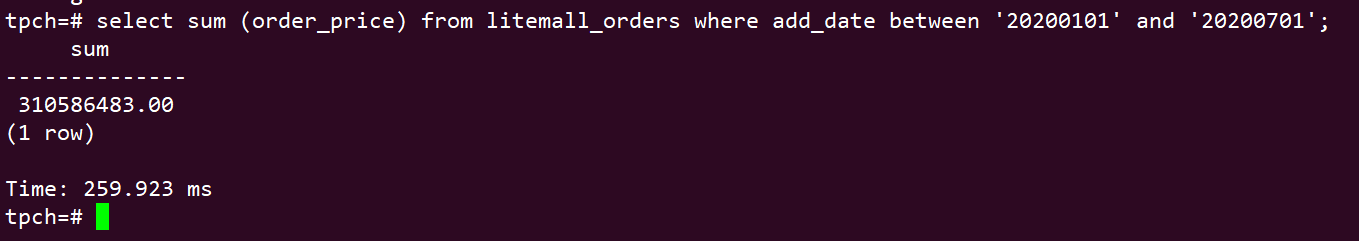
select count(\*) from supplier;;



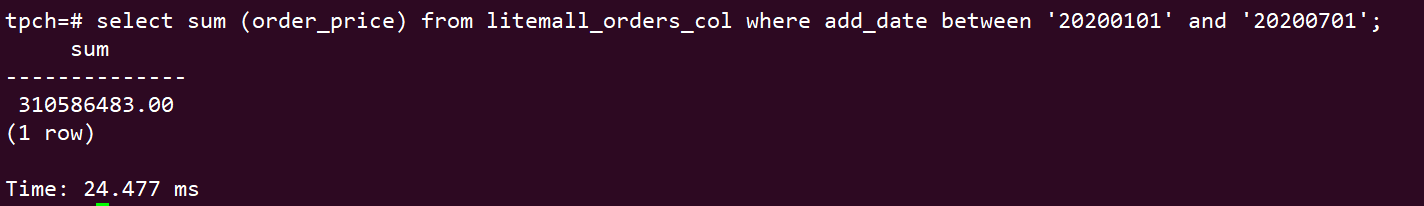
任务二：行存表与列存表执行效率对比

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';

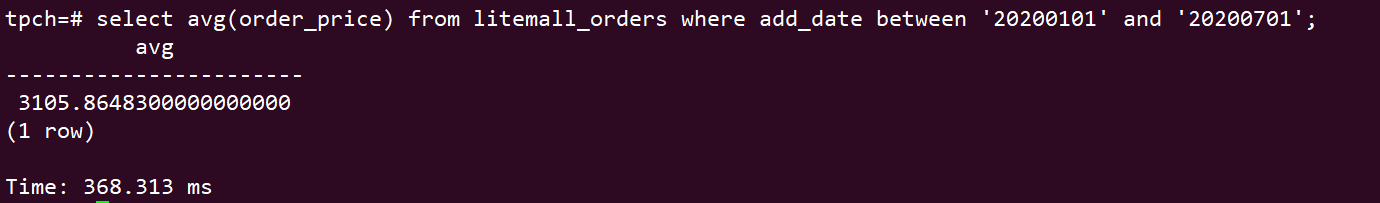


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

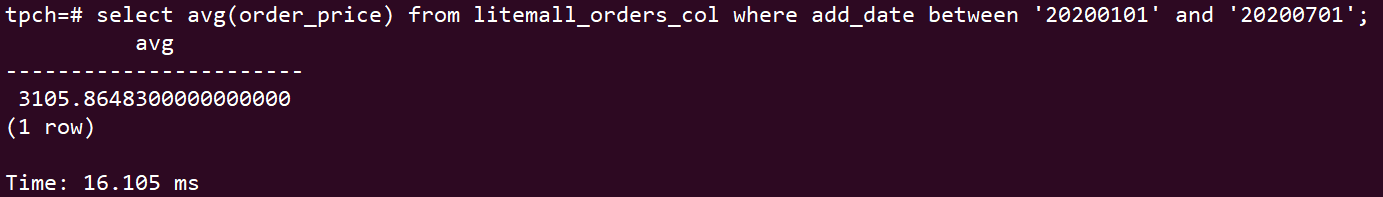


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

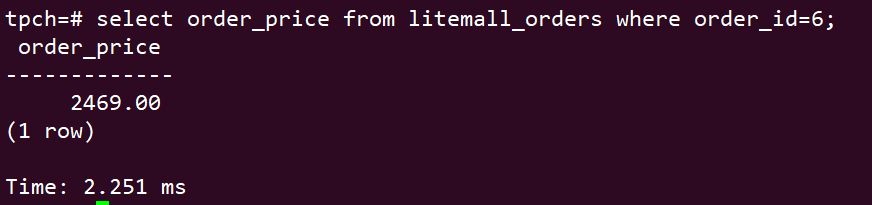


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

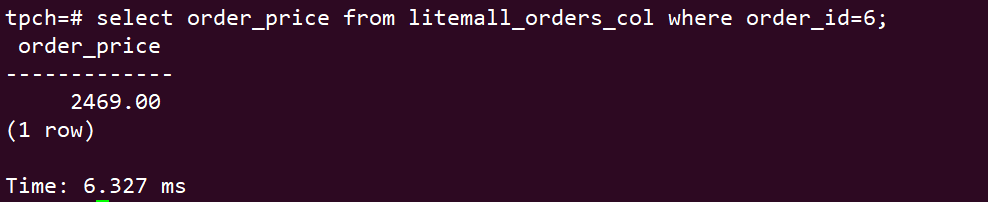


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

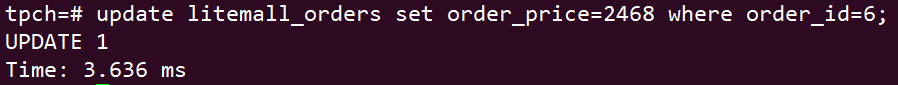


select order\_price from litemall\_orders\_col where order\_id=6;

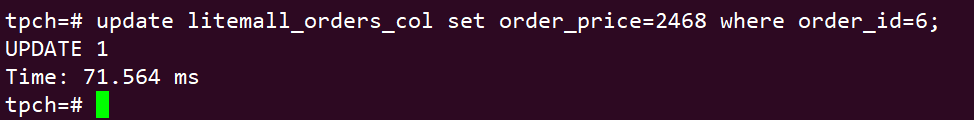


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

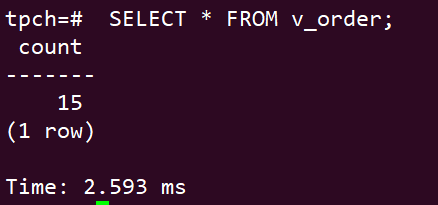
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



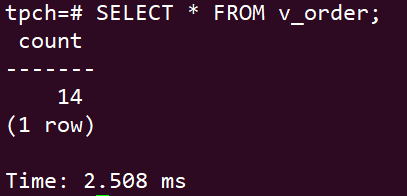
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

行存储表和列存表的数据存储组织方式不同，行存表的一张表的内容都被放在一起，而列存储一张表的内容被分开保存了。这种存储结构的变化导致了在执行相同的增删改查操作的时候行存表和列存表对应的数据处理不一致，因而执行的时间不同。

在执行按照主键操作一个元组的数据时行存表的效率更高，因为行存表的一条记录的字段都被放在了一起，因此效率更高，而列存表因为不同列被存放在不同的地方，查询一个元组的时候需要进行列之间的连接操作，因此效率不高。在执行不需要指定主键按照列进行的运算的时候列存表效率更高，因此同一列的数据被放在了一起，而行存表得扫描一条记录的所有字段。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

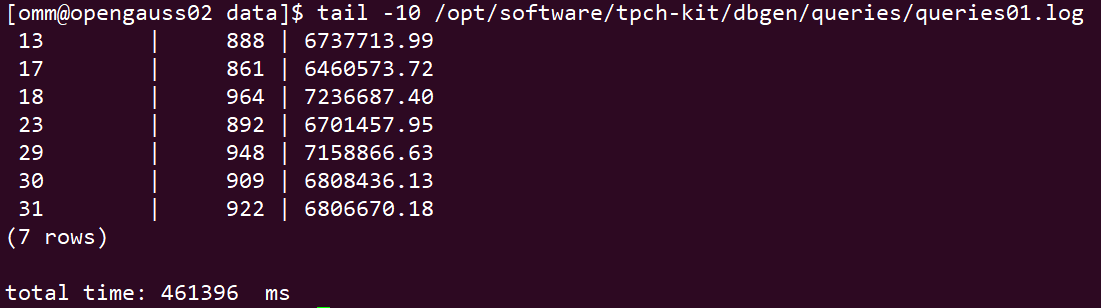
全量物化视图在原表被修改时需要更高的时间进行全量物化视图的生成，而增量物化视图只需 要对原本的数据进行更新，但这也意味着增量物化视图比全量物化视图占用的空间更大，在数 据量大的表中全量物化视图的效率远不如增量物化视图。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

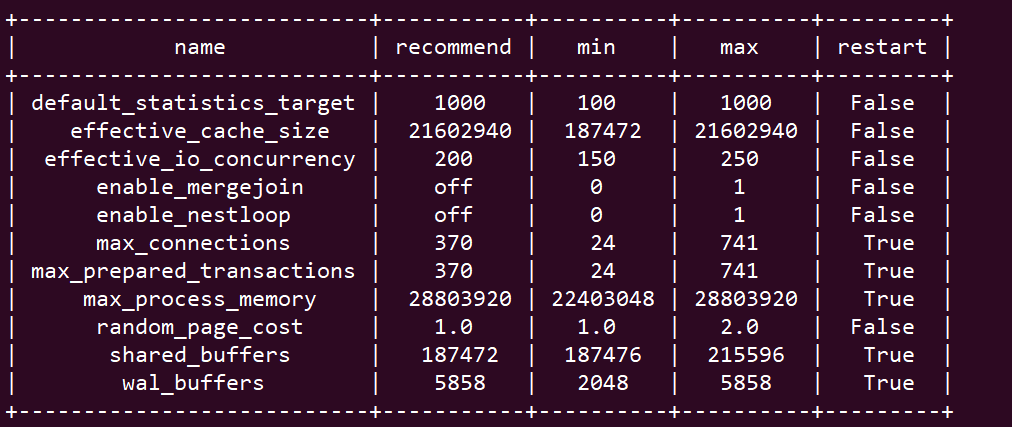
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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



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

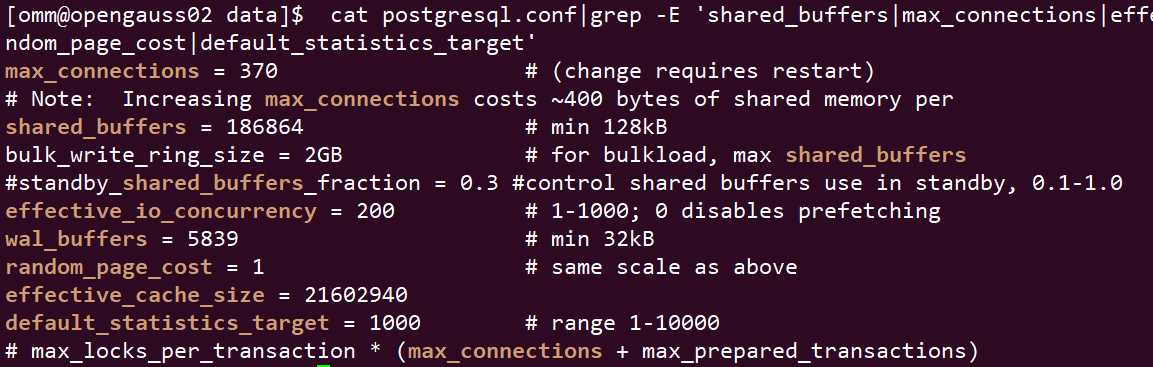
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

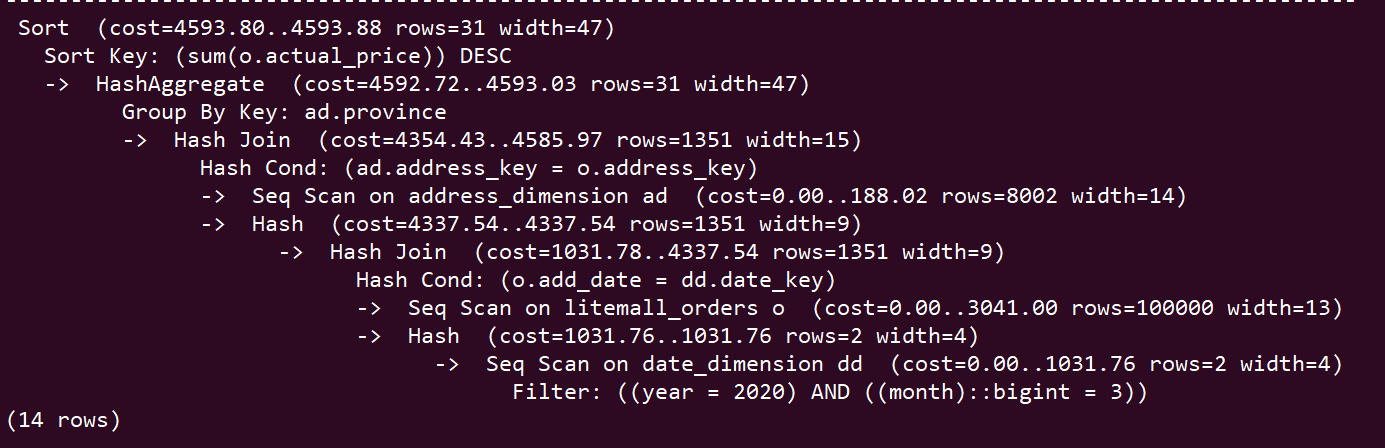
AND o.add\_date = dd.date\_key

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

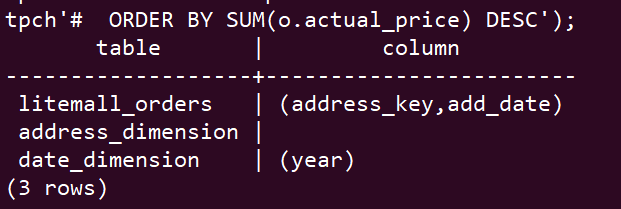
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

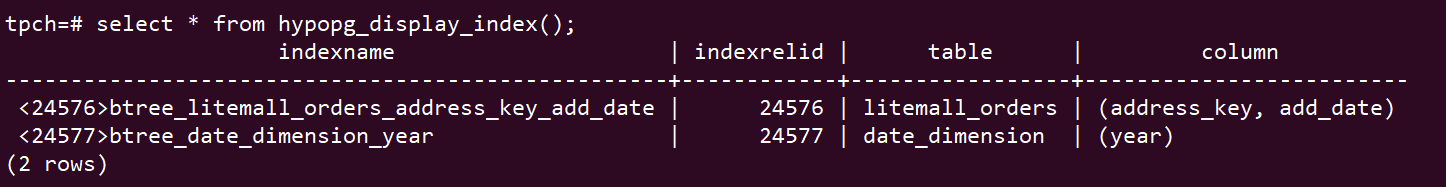
GROUP BY ad.province

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。

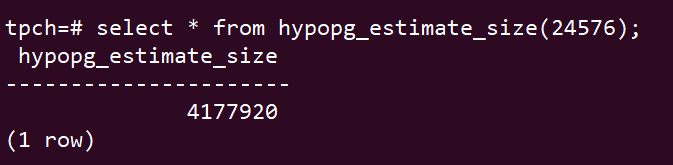
select \* from hypopg\_display\_index();

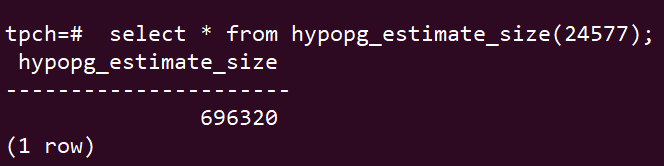


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

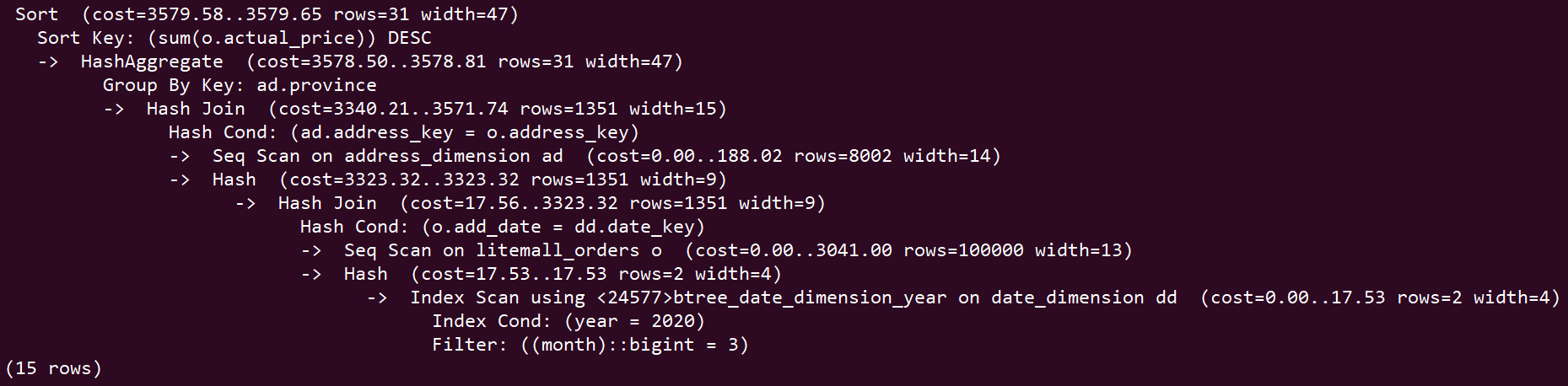
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

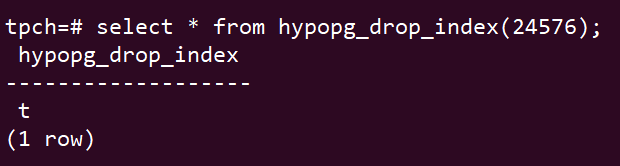
GROUP BY ad.province

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



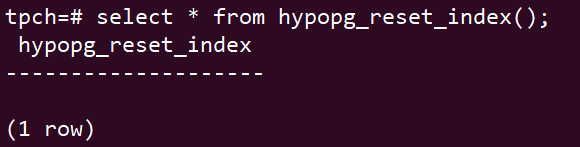
6. 删除某一个索引虚拟列，将执行结果截图。

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



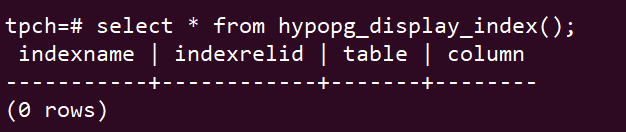
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

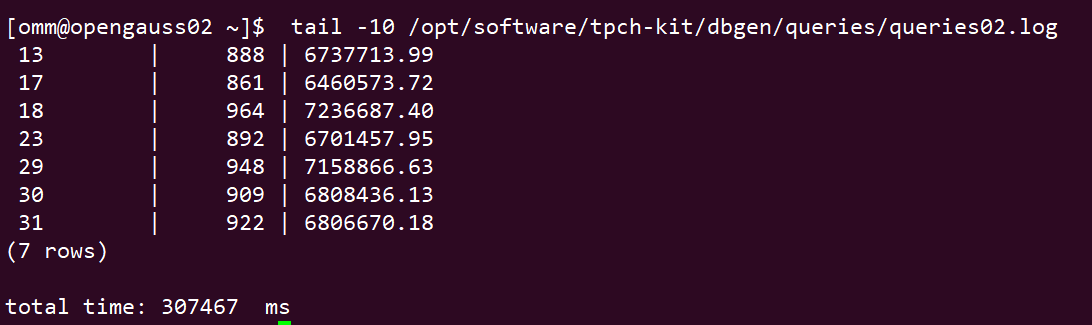
select \* from hypopg\_display\_index();



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

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？

X-Tuner通过对强化学习的方式发现参数的调整对数据库性能的作用，其中X-Tuner在default\_statistics\_target, effective-cache-size, effective\_io\_concurrency, enable\_mergejoin, enable\_nestloop, max\_connections, max\_prepared\_transactions, max\_process\_memory, random\_page\_cost, shared\_buffers, wal\_buffers参数上进行了优化，通过优化参数来提升数据库的性能。

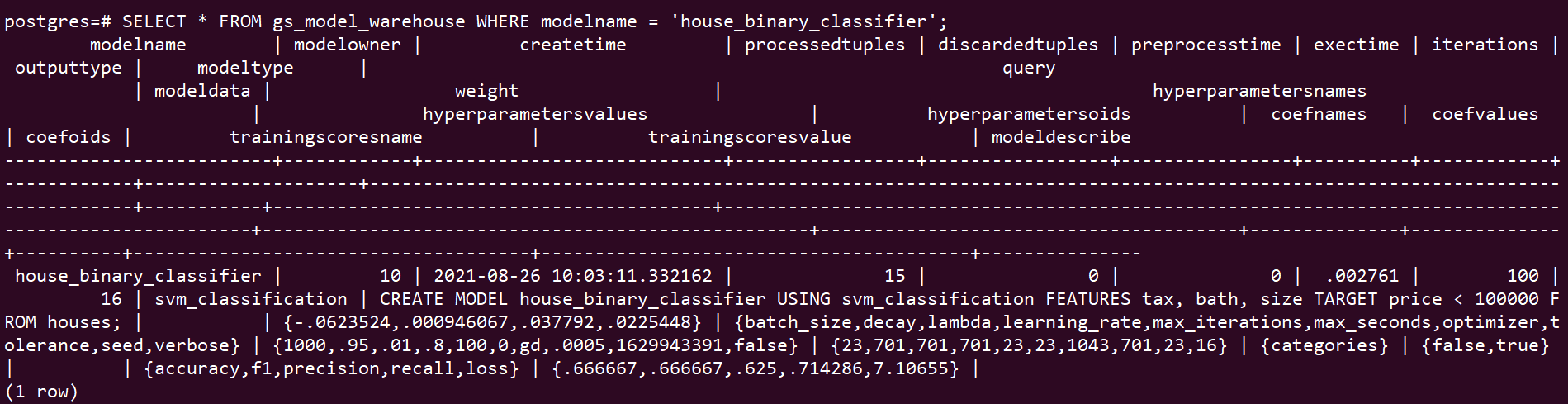
实践思考题2：索引的使用，对于执行SQL有什么好处？除了使用索引和参数外，还有哪些方面可以对数据库进行优化？

索引的使用可以大幅提高数据库查找效率，除了索引和参数外，还可以通过选择适当的数据存 储结构（比如根据数据的操作方式分别选择行存储或者列存储），可以通过分布式存储计算、 优化缓存策略等来优化数据库。

# 关卡四、openGauss的DB4AI特性应用

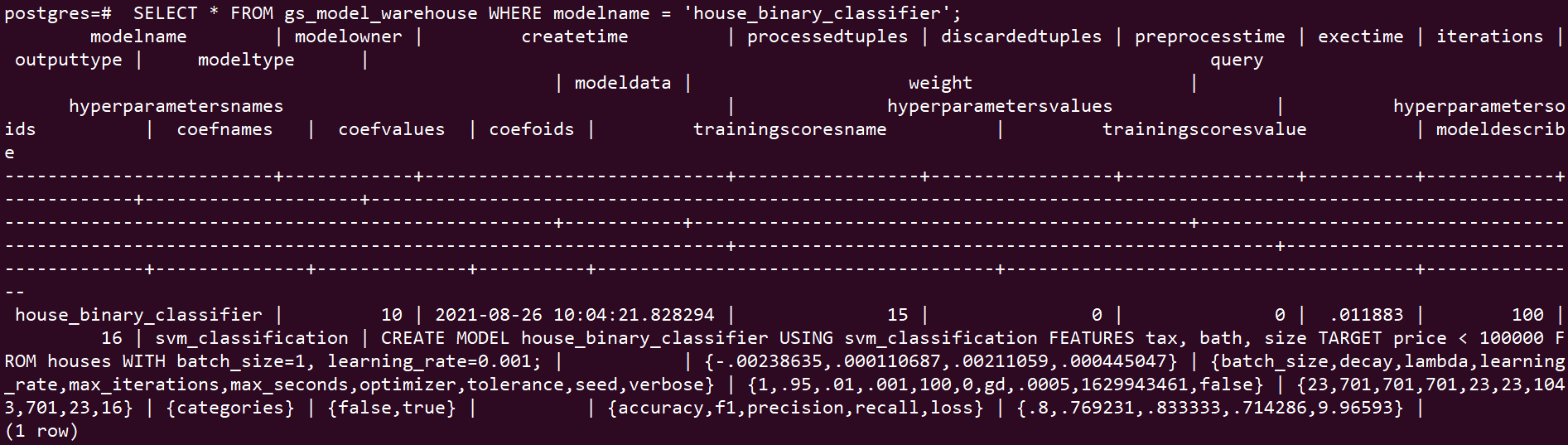
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



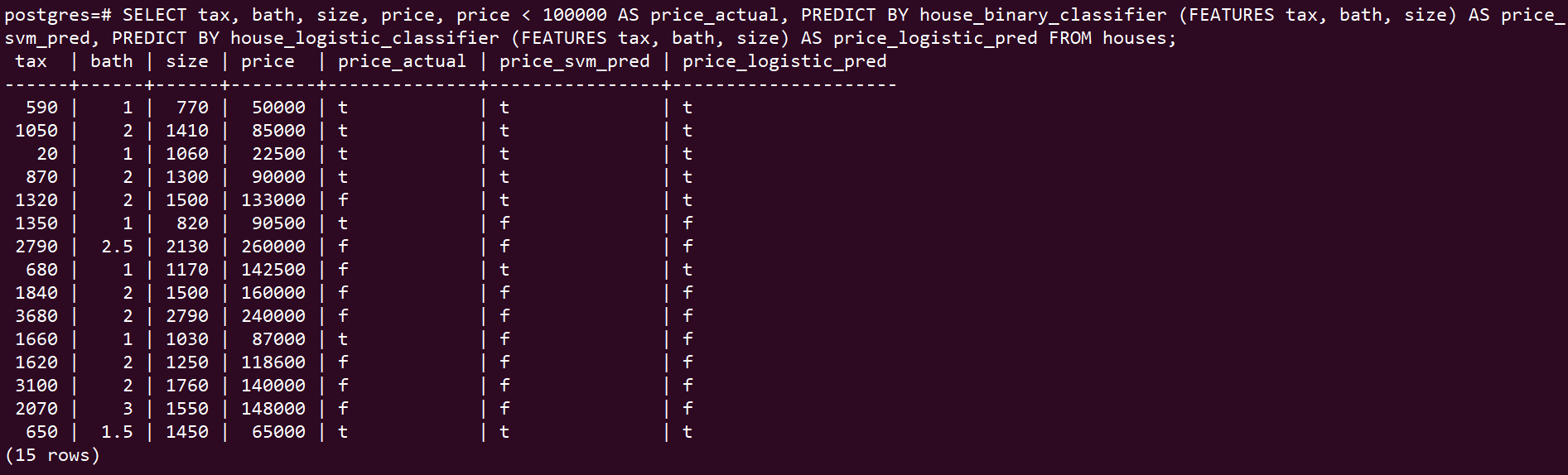
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型可以将回归模型的输出离散化，回归模型也可以将分类模型输出连续化。分类模型用于处理分类问题，回归模型用于处理回归问题。

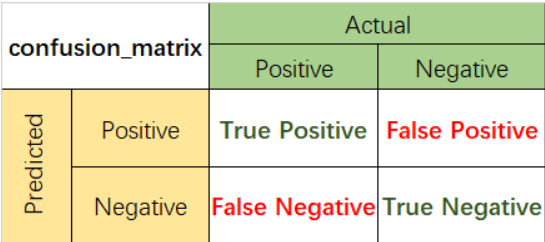
实践思考题2：什么是SVM算法？

SVM是一个二元分类算法，支持线性分类和非线性分类。经过演进，现在也可以支持多元分类，同时经过扩展，也能应用于回归问题。

SVM学习的基本想法是求解能够正确划分训练数据集并且几何间隔最大的分离超平面。

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？

1. 混淆矩阵相关：



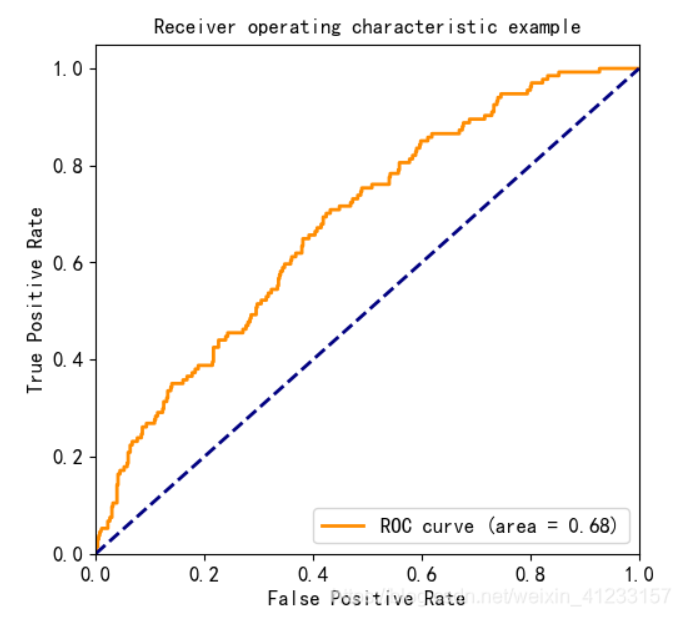
1.1准确率（Accuracy）  
准确率是最常用的分类性能指标。  
Accuracy = (TP+TN)/(TP+FN+FP+TN)  
即正确预测的正反例数 /总数

1.2精确率（Precision）  
精确率容易和准确率被混为一谈。其实，精确率只是针对预测正确的正样本而不是所有预测正确的样本。表现为预测出是正的里面有多少真正是正的。可理解为查准率。  
Precision = TP/(TP+FP)  
即正确预测的正例数 /预测正例总数

1.3召回率（Recall）  
召回率表现出在实际正样本中，分类器能预测出多少。与真正率相等，可理解为查全率。  
Recall = TP/(TP+FN)，即正确预测的正例数 /实际正例总数

1.4F1 score  
F值是精确率和召回率的调和值，更接近于两个数较小的那个，所以精确率和召回率接近时，F值最大。  
2/F1 = 1/Precision + 1/Recall

2. ROC曲线



逻辑回归里面，对于正负例的界定，通常会设一个阈值，大于阈值的为正类，小于阈值为负类。如果我们减小这个阀值，更多的样本会被识别为正类，提高正类的识别率，但同时也会使得更多的负类被错误识别为正类。为了直观表示这一现象，引入ROC。根据分类结果计算得到ROC空间中相应的点，连接这些点就形成ROC curve，横坐标为False Positive Rate(FPR假正率)，纵坐标为True Positive Rate(TPR真正率)。一般情况下，这个曲线都应该处于(0,0)和(1,1)连线的上方。

3.PR曲线

PR曲线的横坐标是精确率P，纵坐标是召回率R。评价标准和ROC一样，先看平滑不平滑（蓝 线明显好些）。一般来说，在同一测试集，上面的比下面的好（绿线比红线好）。当P和R的值 接近时，F1值最大，此时画连接(0,0)和(1,1)的线，线和PRC重合的地方的F1是这条线最大的 F1（光滑的情况下），此时的F1对于PRC就好像AUC对于ROC一样。一个数字比一条线更方 便调型。

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？

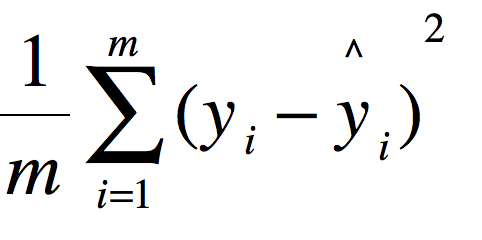
1、SSE 和方差

IMG_256

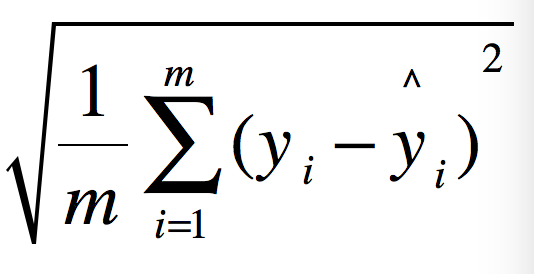
SSE越接近于0，说明模型选择和拟合更好，数据预测也越成功。

2、均方误差（MSE）

该统计参数是预测数据和原始数据对应点误差的平方和的均值，也就是SSE/n



3、均方根误差（RMSE）（Root Mean Squard Error）均方根误差（又称RMSD: root mean square deviation）。



其中，yiyi是第ii个样本的真实值，yi^yi^是第ii个样本的预测值，nn是样本的个数。该评价 指标使用的便是欧式距离。

4、MAE:mean asolute error 平均绝对误差 预测、真实数据

