openGauss AI特性创新实践课

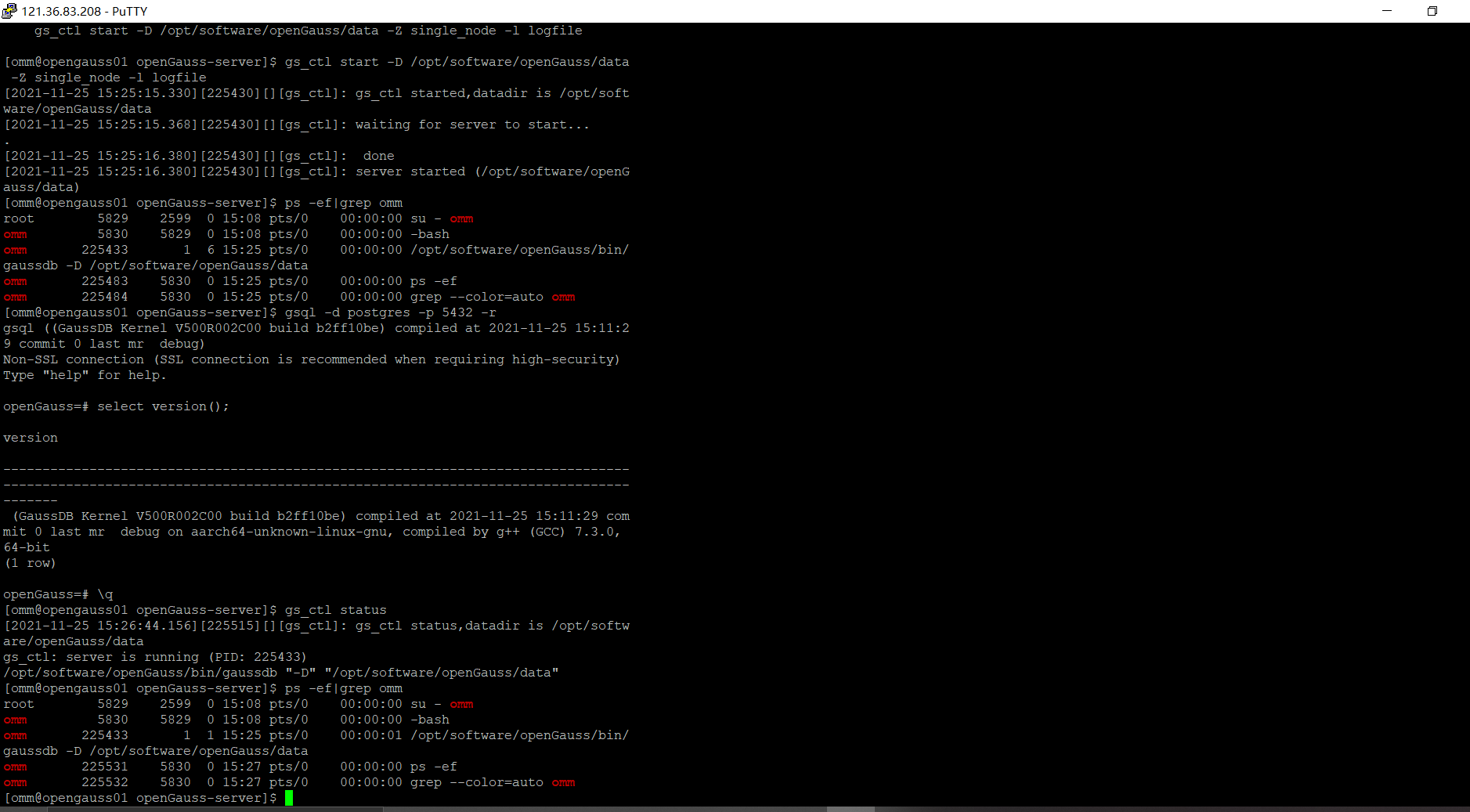


华为技术有限公司

# 关卡一、openGauss数据安装及基本操作

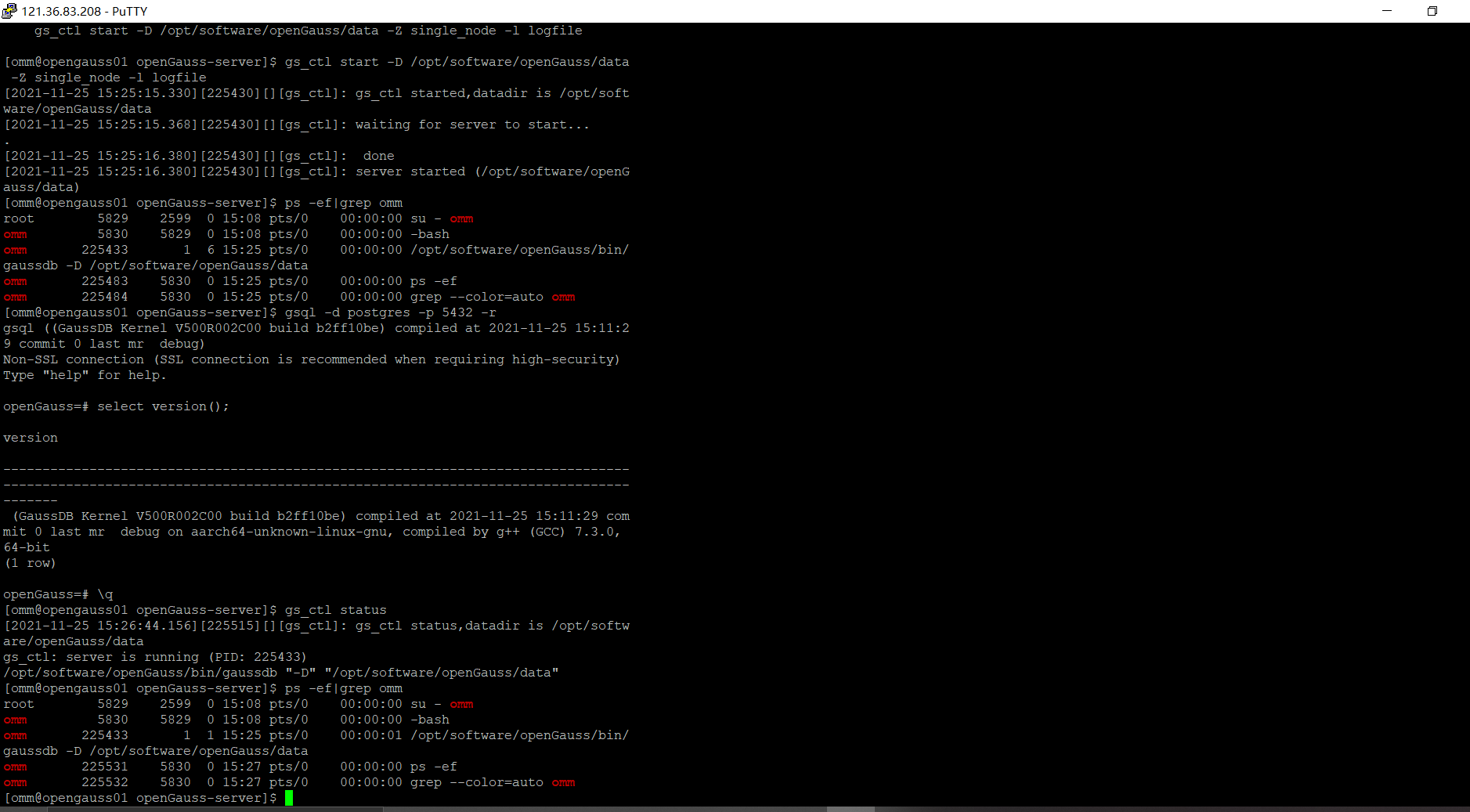
openGauss数据安装及基本操作, 作业提交任务如下：

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？

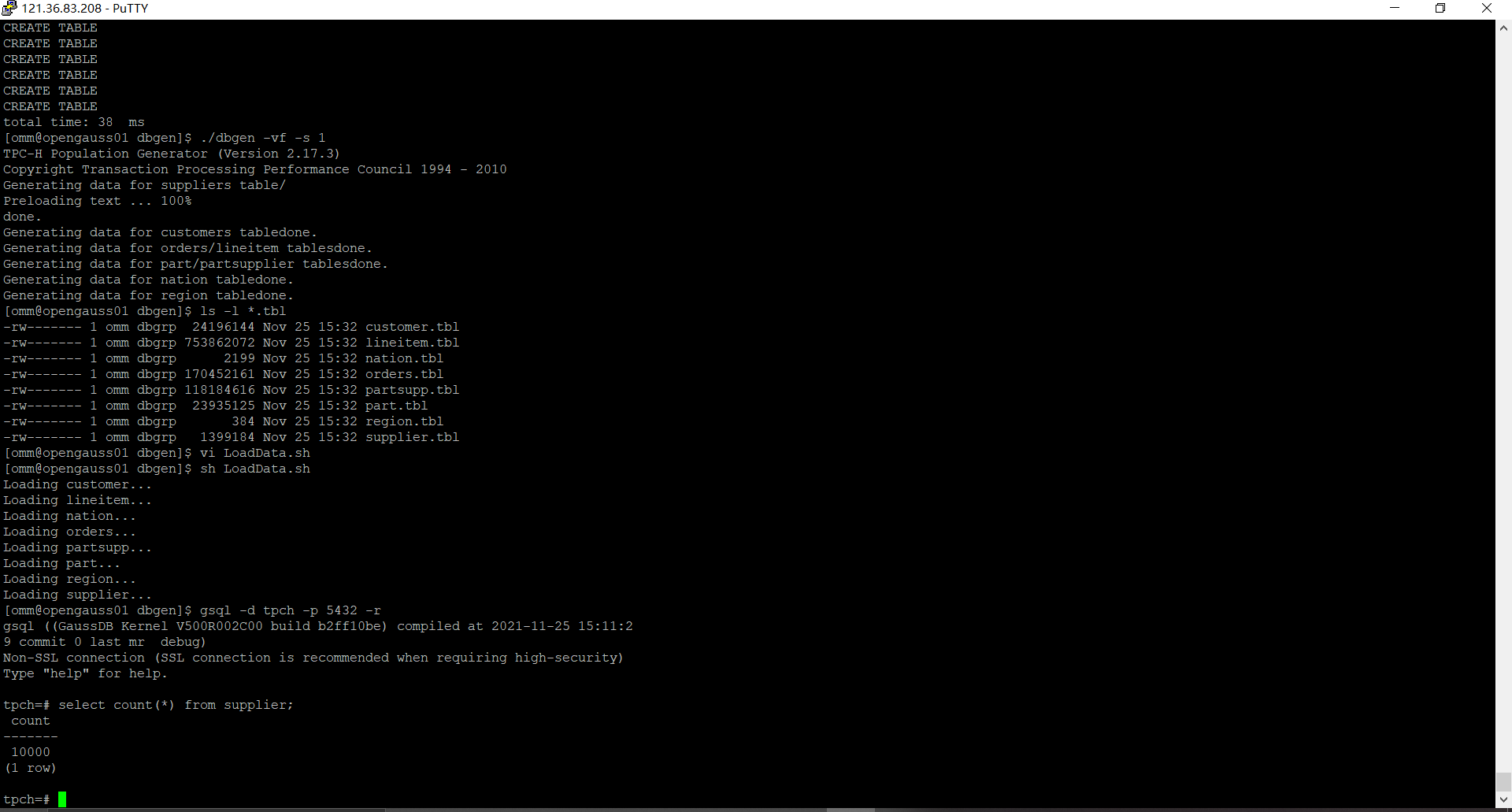
源码安装用户可以根据自己的需要对源码进行修改，然后再编译生成二进制文件。方便于对数据库进行修改和更新。如果采用二进制文件安装，用户则无法对程序作任何修改。编译安装过程，可以设定参数，按照需求，进行安装，并且安装的版本，可以自己选择，灵活性比较大。

# 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：

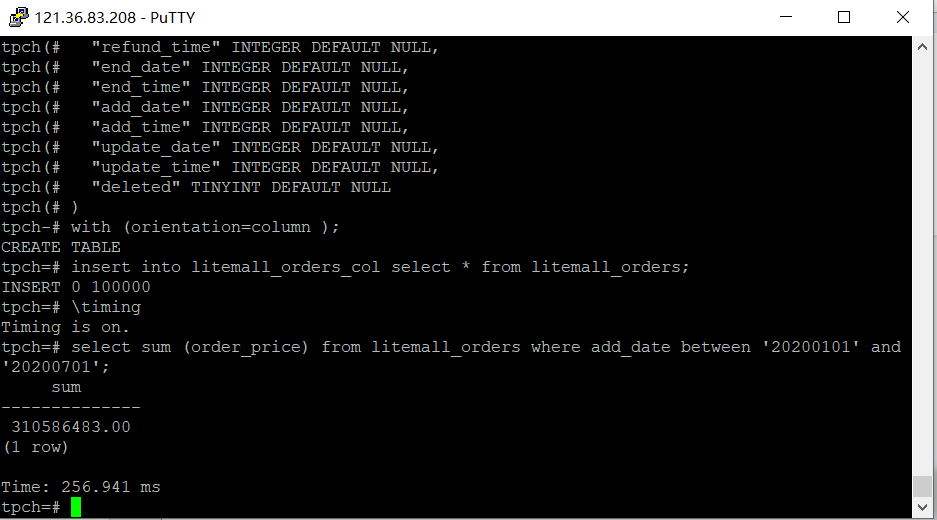
select count(\*) from supplier;;

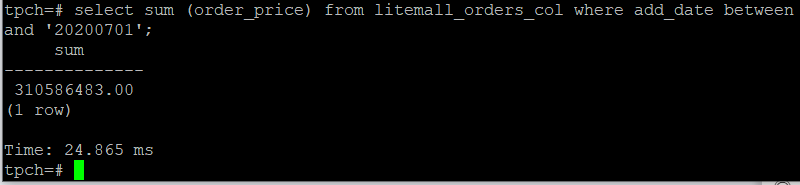


任务二：行存表与列存表执行效率对比

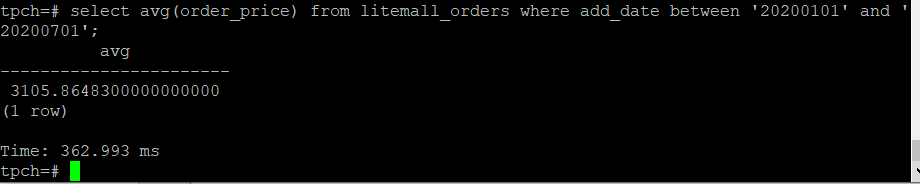
1. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的总和查询，并对比执行效率截图

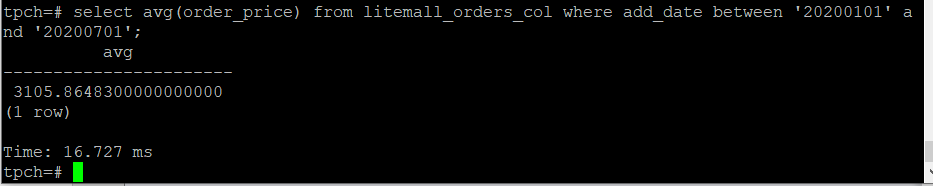
select sum (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';



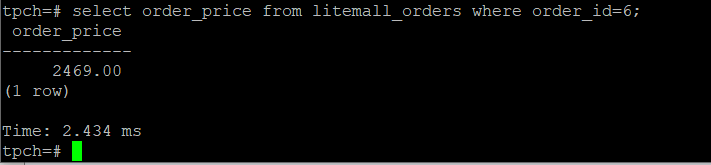
select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

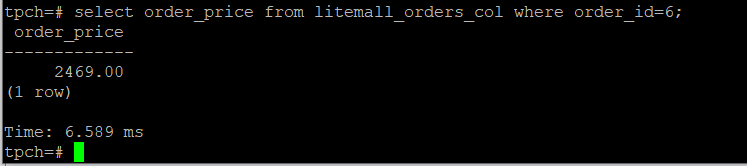
2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

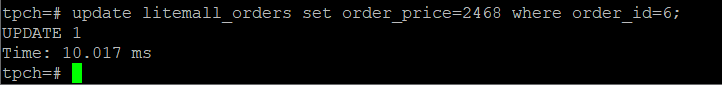
select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

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。

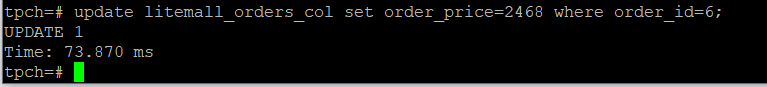
select order\_price from litemall\_orders where order\_id=6;

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;



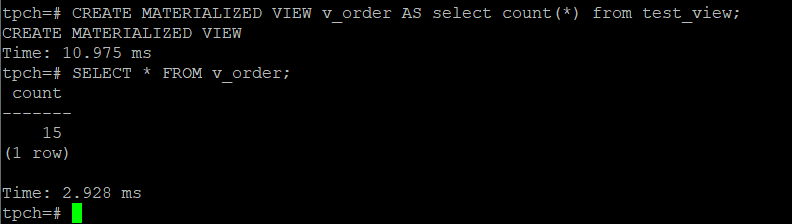
任务三：物化视图的使用

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;

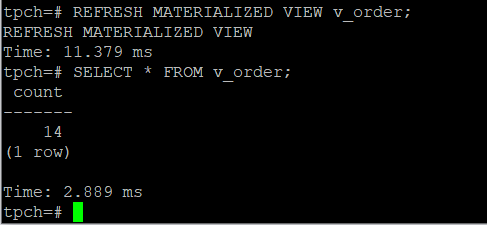
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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;



6.补充：刷新**全量**物化视图，查询物化视图结果，将执行结果截图。



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？

行存表是按行存储的，列存表是按列存储的，如果SQL语句主要查询少数几列，多列查询较多，使用列存储的效率更高。当SQL语句涉及的属性列个数过多时，由于行存表按一行存在一个tuple里，效率更高。所以执行相同的SQL语句时，执行时间不同。比如在上面的第一个对一列求和的示例中，行存用的时间是256.941ms，列存用的时间是24.865ms，列存比行存快了一个数量级，因为只涉及到了对一行的操作，采用列存更有效。又比如在示例修改order\_id = 6的order\_price 为2468中，行存的·时间是10.017ms，列存的时间是73.87ms，因为涉及到了多列，采用行存时一行的数据都在一个tuple中，这时行存效率更高。

在执行 行中选取所有属性的查询操作(select \*)、随机的增删改查操作 适合于行存表

在执行数据量大，查询的列数比较少的情况，适合于列存表。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

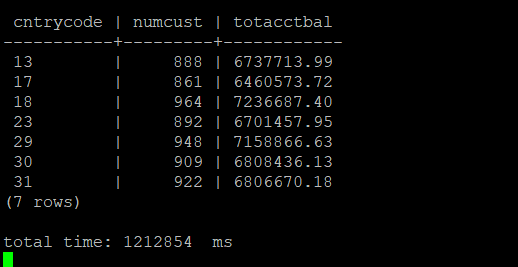
全量刷新机制是首先物化视图对应表中的数据采用delete全部删除，然后再从原表中使用insert把数据重新插入。增量刷新：增量刷新是基于主表上的物化视图日志进行刷新的，主表上每插入或删除一条数据，对应物化视图日志中同样会插入一条数据（主表更新一条数据，对应物化视图中会插入两条记录数据），物化视图刷新后主表上物化视图日志记录信息会被清空，重新开始记录后面的更新。

增量物化视图，可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。与全量创建物化视图不同在于目前增量物化视图所支持场景较小，目前物化视图创建语句仅支持基表扫描语句或者UNION ALL语句。

# 关卡三、openGauss的AI4DB特性应用

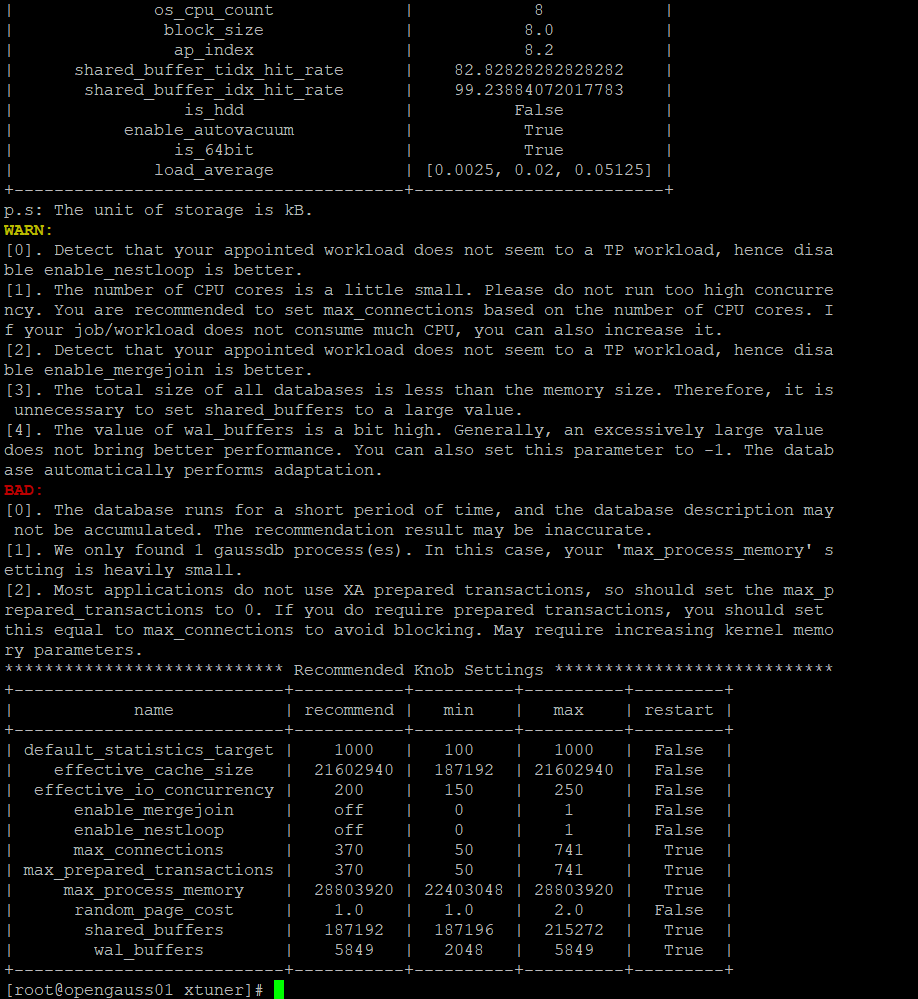
任务一：使用X-Tuner进行参数优化

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

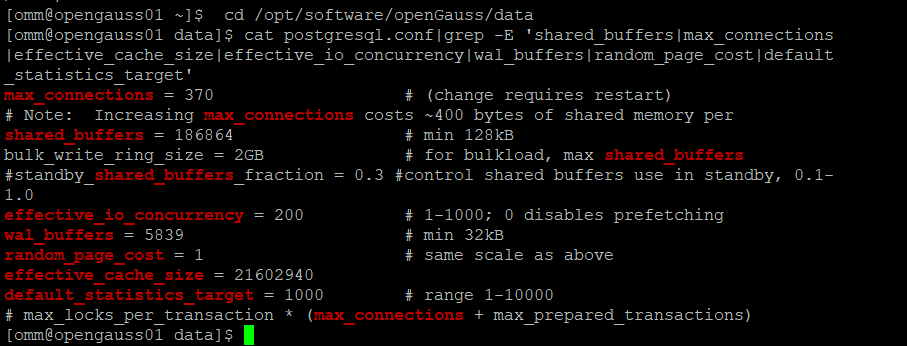
2. 使用root用户，执行X-Tuner进行参数建议优化，将执行结果截图

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

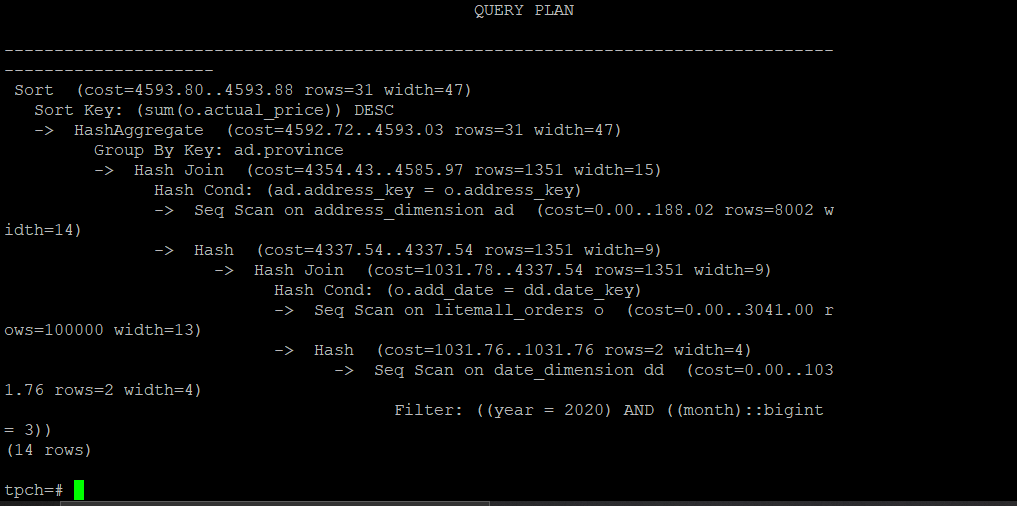
WHERE o.address\_key = ad.address\_key

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

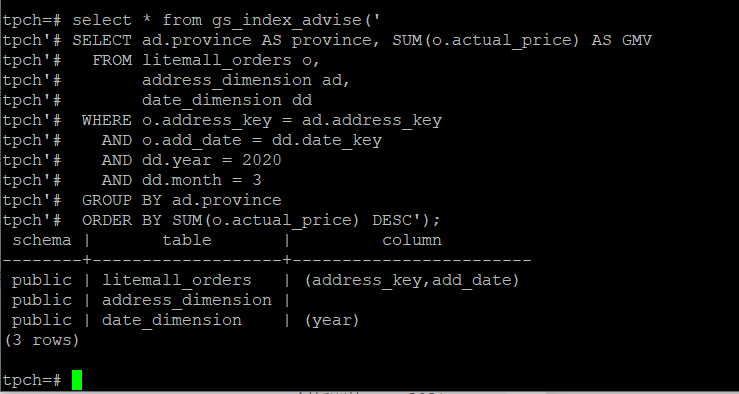
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

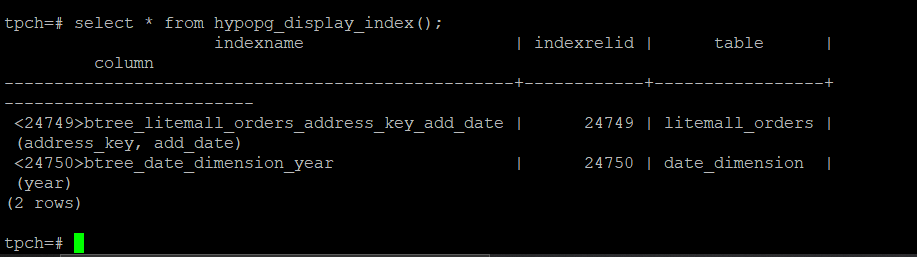
GROUP BY ad.province

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



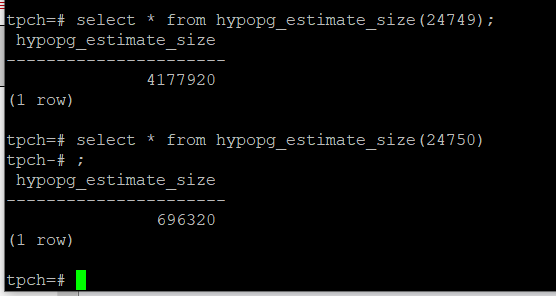
3. 查看创建的虚拟索引列，将执行结果截图。

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

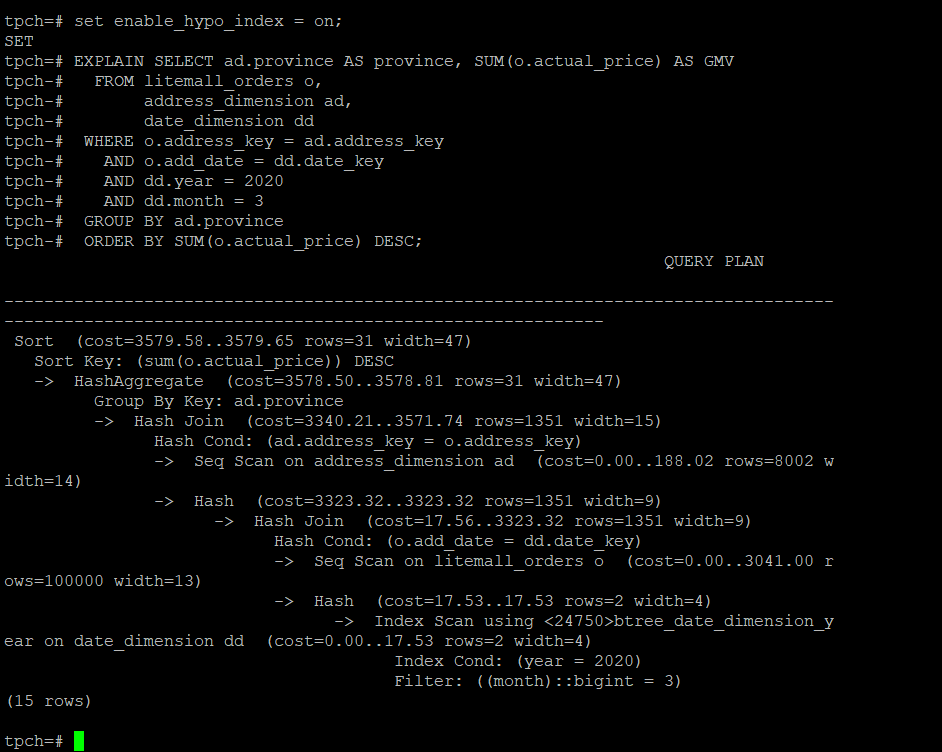
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

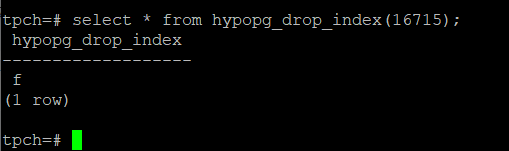
GROUP BY ad.province

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



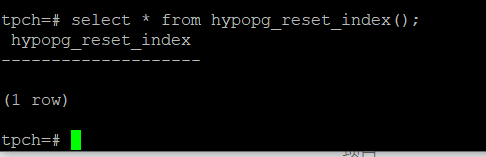
6. 删除某一个索引虚拟列，将执行结果截图。

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

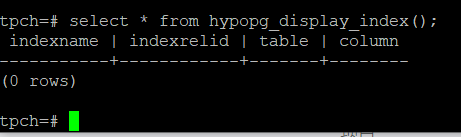


7. 删除某一个索引虚拟列，将执行结果截图。

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

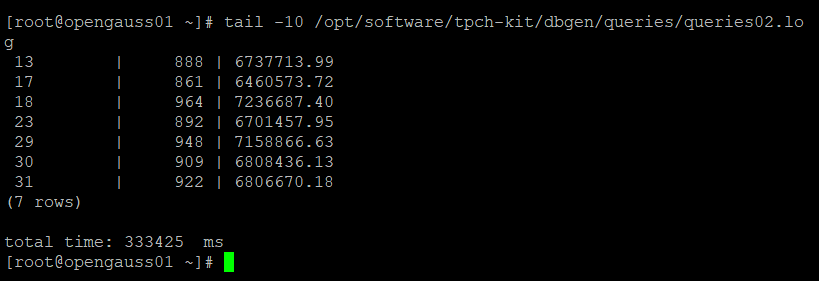


8. 查看索引虚拟列，将执行结果截图。

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。

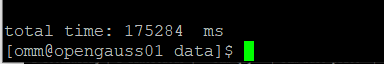
1. 重新执行queries.sql查询，将执行结果截图：

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

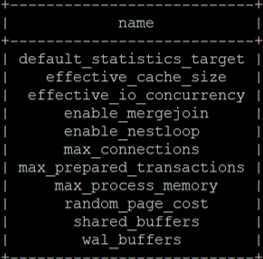
挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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

已经通过X\_Tuner进行参数优化，建立主键索引，然后考虑对queries.sql里的每一条SQL语句进行索引推荐，然后进行汇总。经过尝试发现，如果所有推荐的索引都采用，发现性能变差了，然后经过不断筛选，最后执行时间缩短了一倍左右。

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？

优化的参数：

原因：比如max\_connections 是针对所有的账号所有的客户端并行连接到MYSQL服务的最大并行连接数。并行的用户过少时，系统利用率比较低，用户过多时，系统资源紧张。

通过对参数的优化，使数据库的效率达到最高的同时，减小系统各种资源的消耗。

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？

好处：索引相当于字典的目录，可以先从目录中查找,看看我们需要的找字在字典中具体页码是多少,然后再直接翻到对应的页码,从而快速的找我们需要的内容。如果没有索引，只能从头开始查找。所以索引能够极大地提高查询的效率。索引可以分为聚簇索引和非聚簇索引，聚簇索引只能有一个，非聚簇索引可以有多个。

其他方法：

优化SQL语句，不要任何地方都使用select \* from t ，用具体的字段列表代替“\*”，不要返回用不到的任何字段。不在索引列做运算或者使用函数。查询尽可能使用 limit 减少返回的行数，减少数据传输时间和带宽浪费。

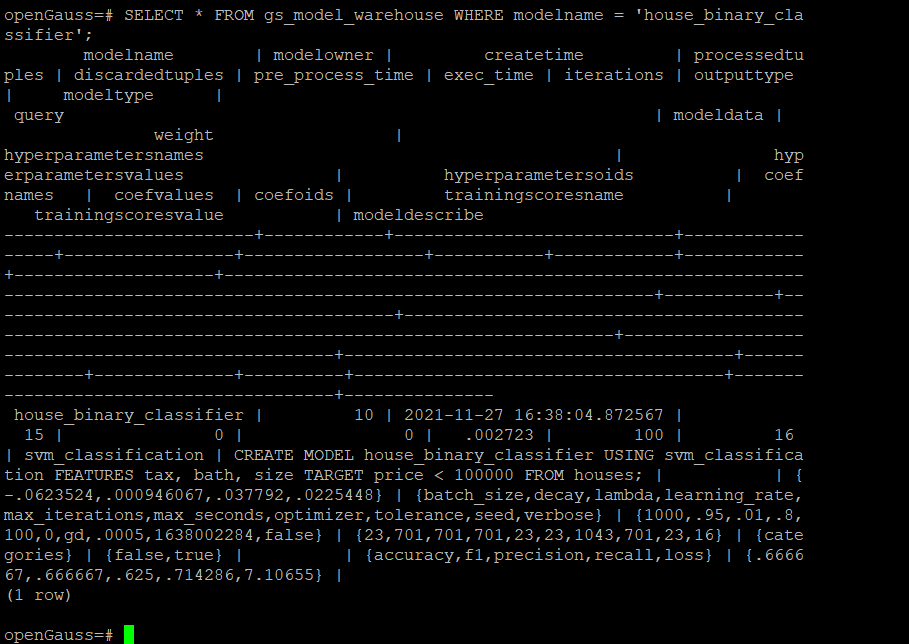
优化表的数据类型，使用能正确地表示和存储数据的最短类型，这样可以减少对磁盘空间、内存、CPU缓存的使用。将表按照功能模块、关系密切程度划分出来，部署到不同的库上。

对于磁盘I/O的优化：使用磁盘阵列，调整磁盘调度算法，减少磁盘寻道时间。

应用优化包括使用数据库连接池和实用查询缓存。

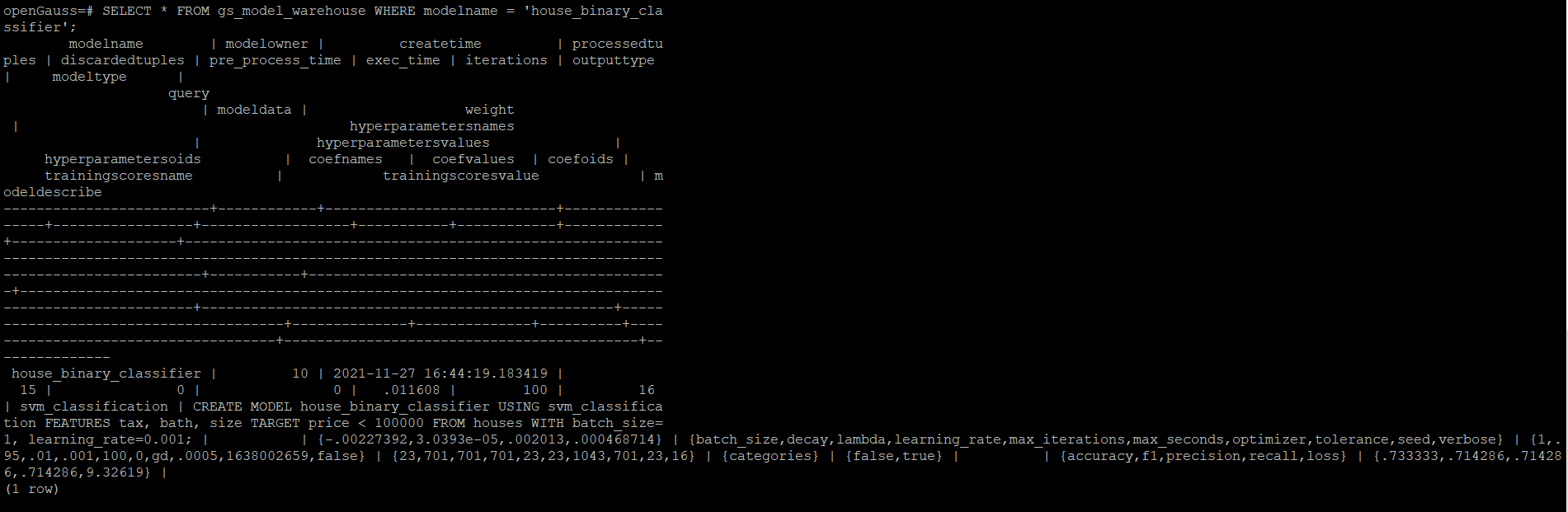
# 关卡四、openGauss的DB4AI特性应用

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';

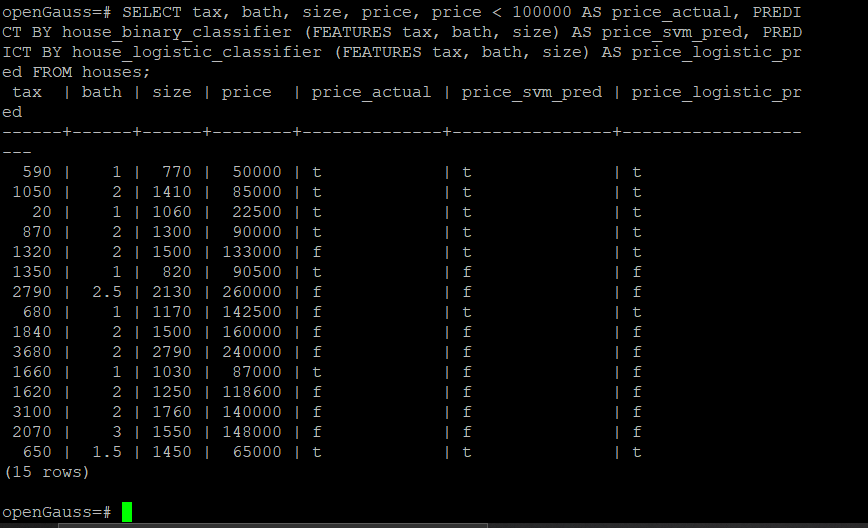
任务二：观察新模型的信息，将执行结果截图。

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：处理定性问题，输出的是离散数据，目的是为了寻找决策边界，评价方式是精度和混淆矩阵等

回归模型：处理定量问题，处理连续变量，目的是找到最优拟合，评价方法是拟合优度、SSE等。

实践思考题2：什么是SVM算法？

SVM通常用于处理二分类问题，思想是最大化间隔思想，找到一个超平面，把两个不同类别尽可能分开，两个类别的差距最大，这就变成了一个优化问题，两个类别到超平面的距离最大化，经过处理等价转变为一个最小化问题。通过数学中的优化理论得到最佳的分类平面。对于非线性二分类问题，可以通过核函数处理转化为线性二分类问题。

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？

混淆矩阵中有：

准确度(accuracy)：预测正确的样本个数占总样本个数的比例

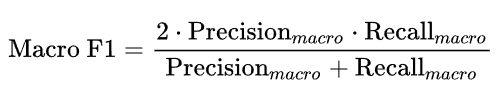
精度（precision）：预测为正样本中，预测正确的样本的比例。

召回率：真实为正样本中，模型预测为正样本的样本比例。

F1-score：它是精度和召回率的综合。F1-score = 2\*Precison\*Recall/(Precision+Recall)

MCC: 马修斯相关系数，MCC的取值范围在 [-1, 1]， 取值为1 表示预测与实际完全一致，取值为0表示预测的结果还不如随机预测的结果， -1 表示预测结果与实际的结果完全不一致

Macro F1： 宏平均

计算公式为：

Macro F1 本质上是所有类别的统计指标的算术平均值来求得的，这样单纯的平均忽略了样本之间分布可能存在极大不平衡的情况。

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？

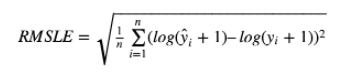
平均绝对误差MAE：计算每一个样本的预测值和真实值的差的绝对值，然后求和再取平均值。MAE是一个线性的指标，所有个体差异在平均值上均等加权，所以它更加凸显出异常值，相比MSE。

MSE（Mean Squared Error）均方误差：用 真实值-预测值 然后平方之后求和平均。线性回归用MSE作为损失函数。

RMSE：是MSE的平方根

R-Squared：在线性回归时，等于相关系数的平方，衡量了一个模型中能够被解释的部分大小。

RMSLE：均方根对数误差



RMSLE主要针对数据集中有一个特别大的异常值，这种情况下，数据会被skew，RMSE会被明显拉大，这时候就需要先对数据log下，再求RMSE，这个过程就是RMSLE