openGauss AI特性创新实践课



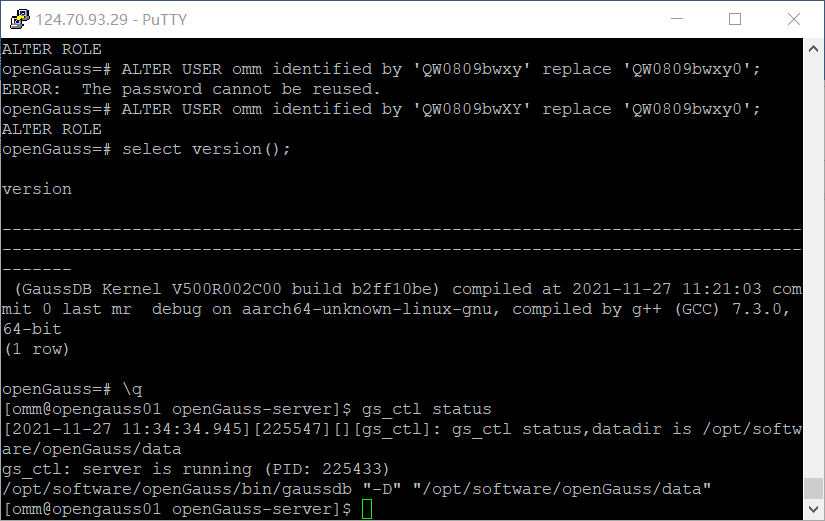
华为技术有限公司

# 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：

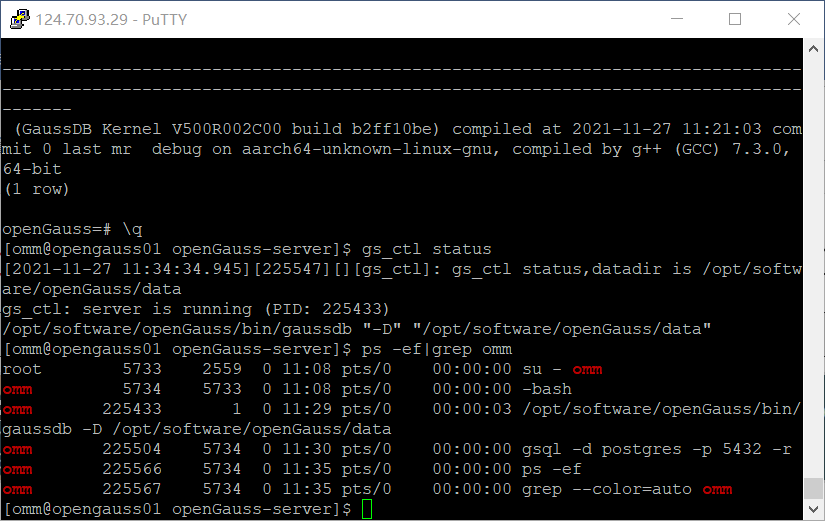
任务一：数据库状态验证

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）



任务三：简单描述关卡一中，安装数据库所需要的步骤

针对数据库源码添加存放的文件路径，下载第三方编译库并解压使用，下载opengauss源码至相应路径，创建用户和设置相关权限。添加环境变量并生效，在源码下生成配置文件编译并安装，完成初始化数据库的操作。

实验思考题：为什么需要通过源码编译，安装数据库？

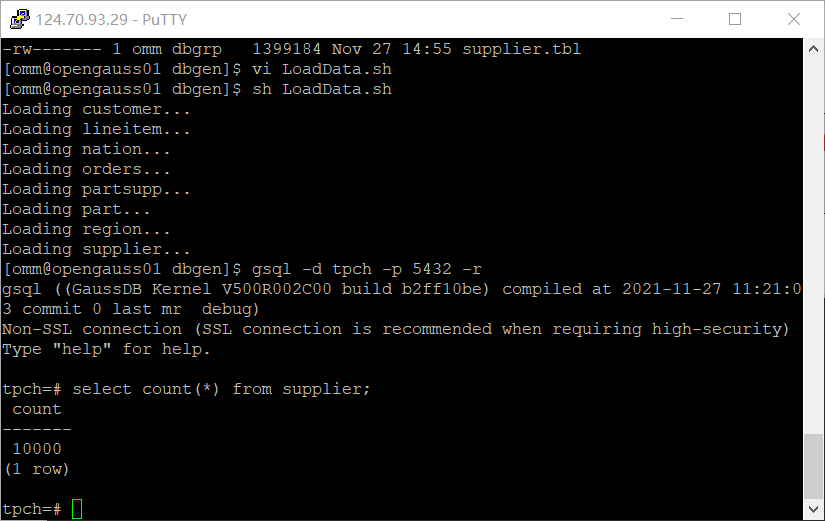
通过源码编译可以自由选择配置地点，比较灵活。也能加深对数据库配置的了解，方便对数据库做出各种配置升级调整，针对主机的情况灵活使用。

# 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：

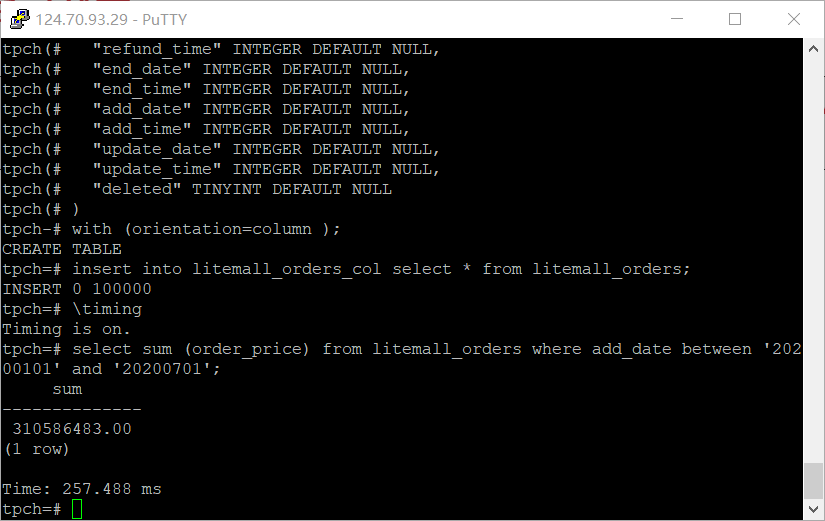
select count(\*) from supplier;;



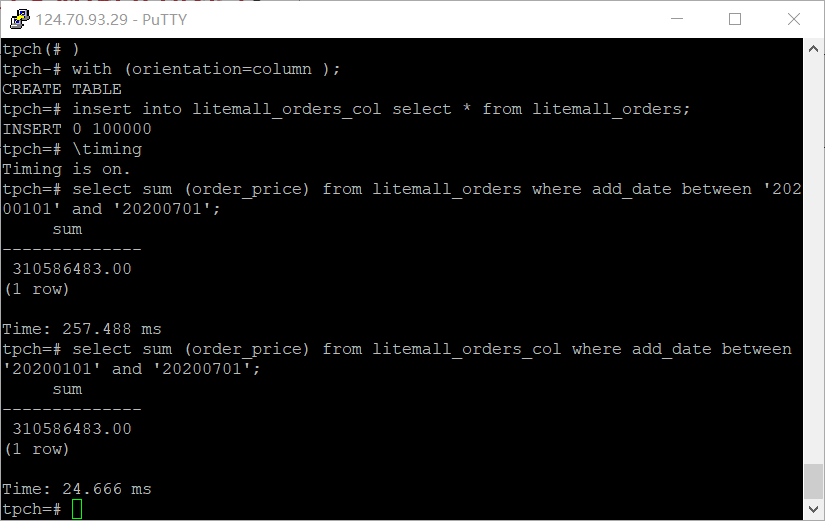
任务二：行存表与列存表执行效率对比

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';

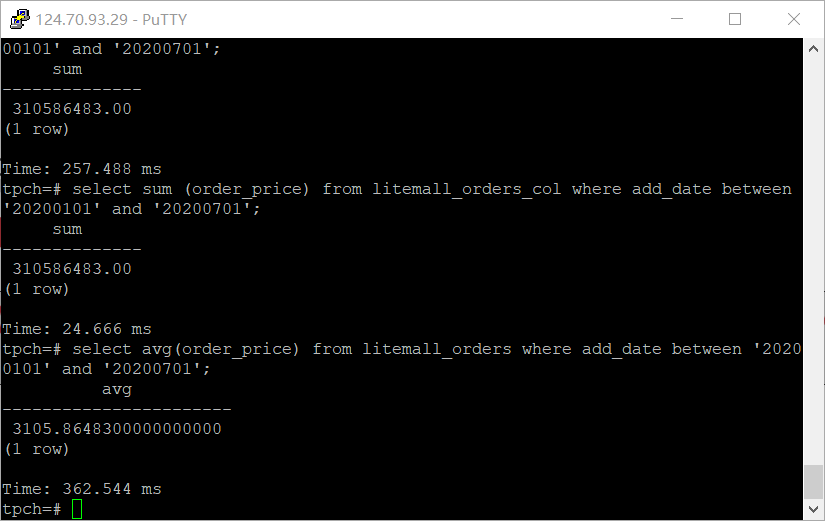


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

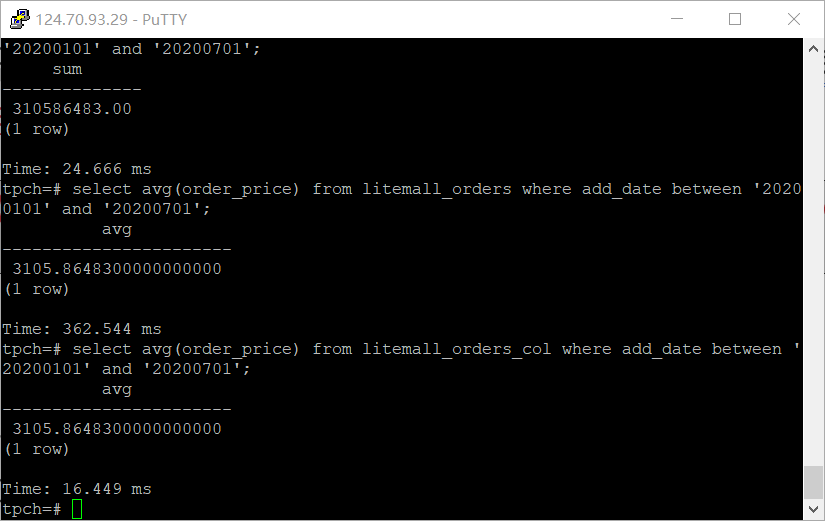


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

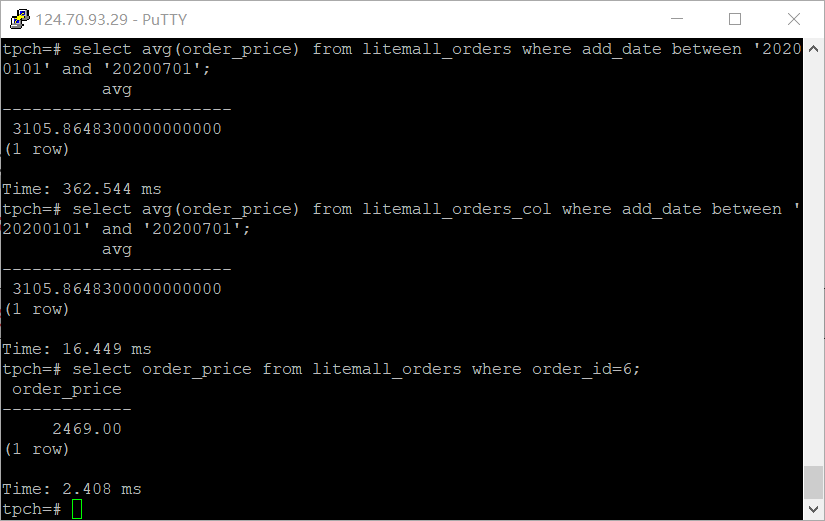


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

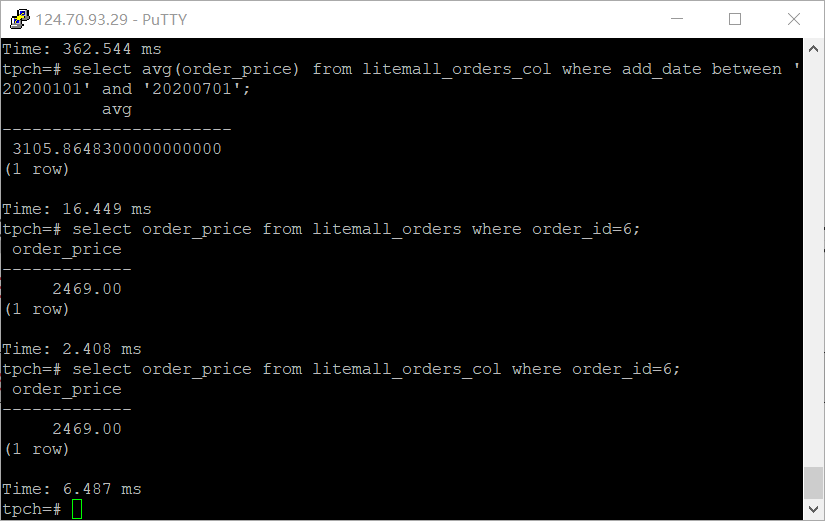


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

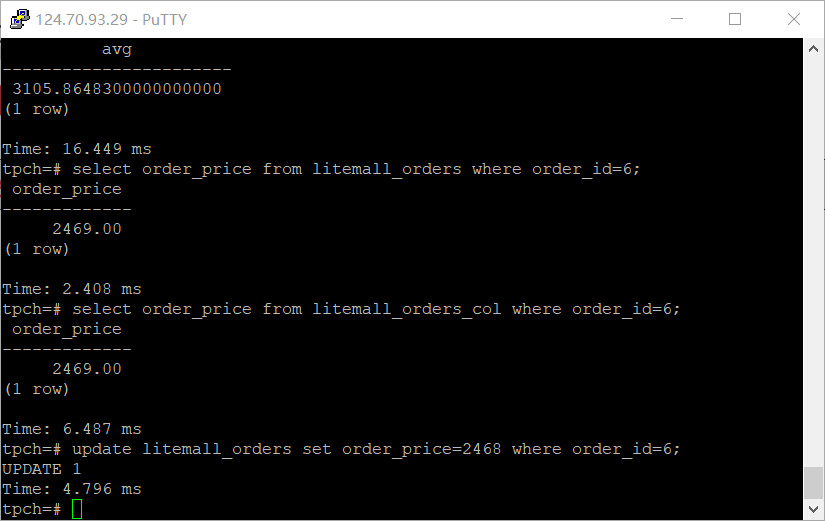


select order\_price from litemall\_orders\_col where order\_id=6;

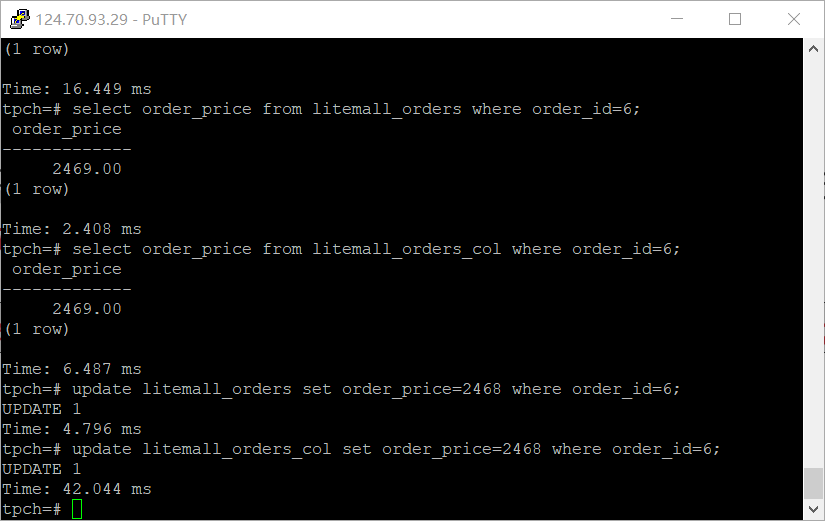


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



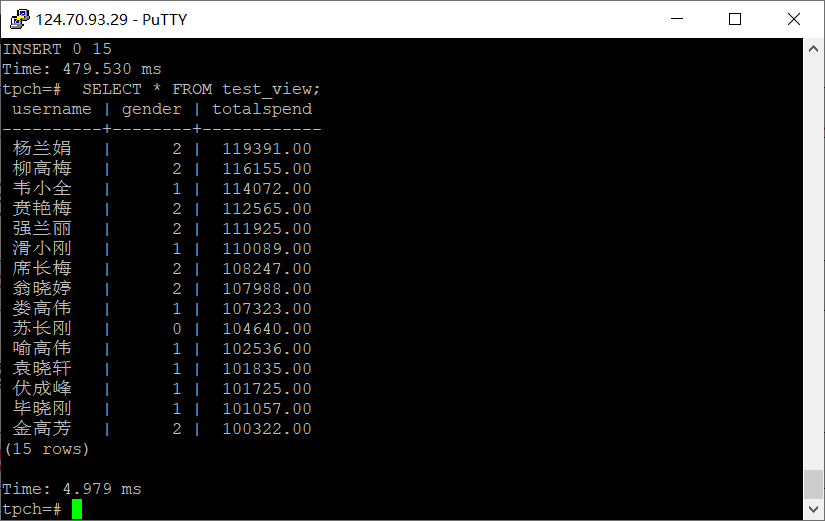
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

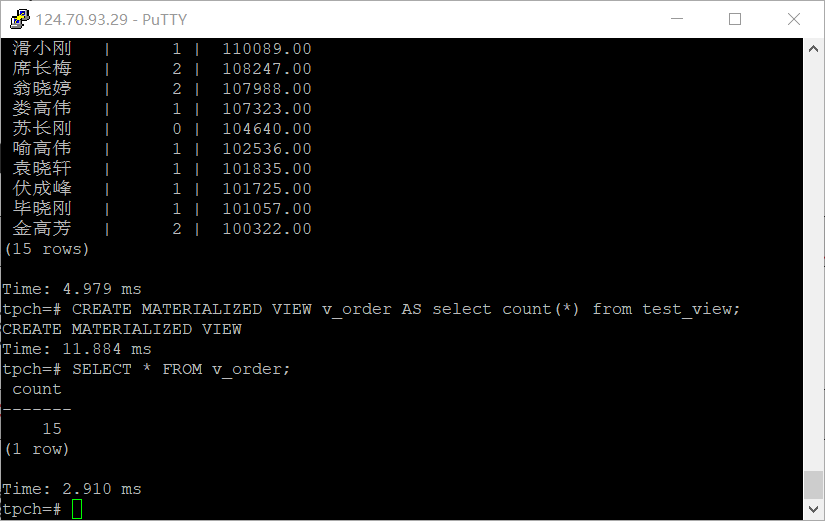
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



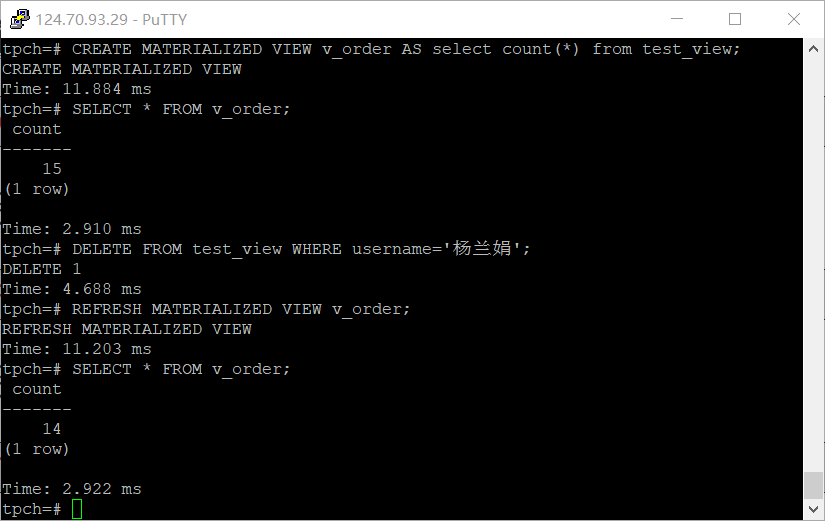
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



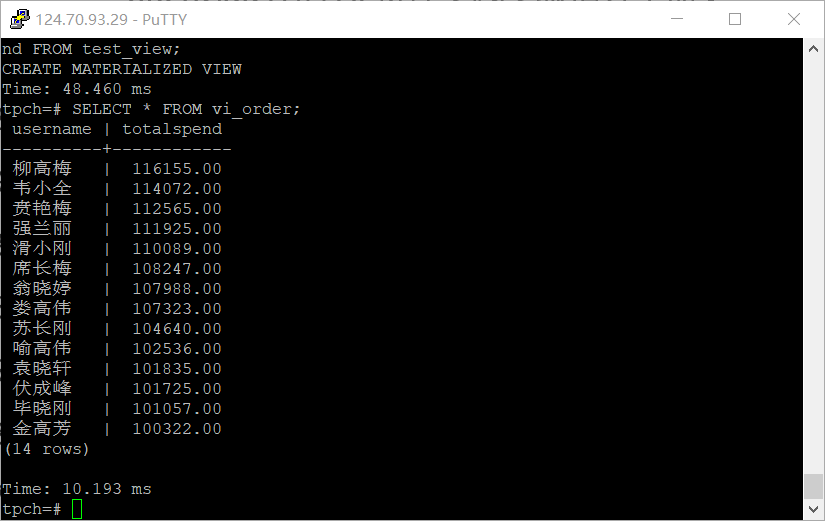
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



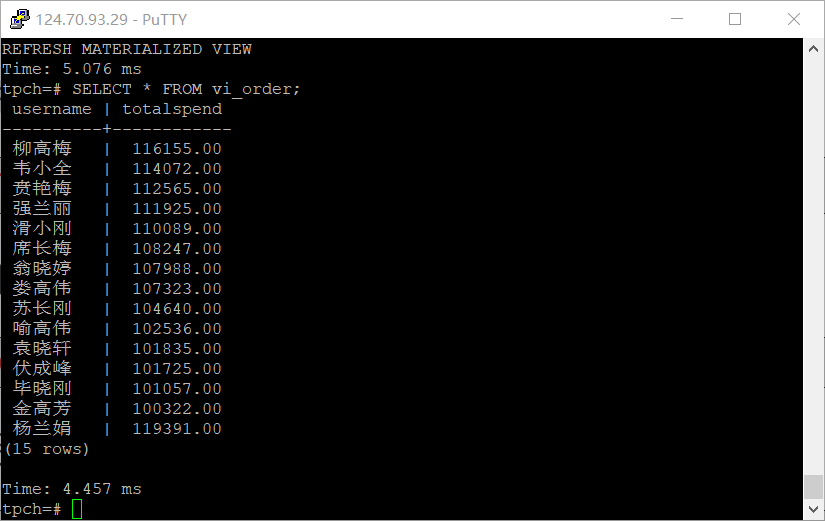
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



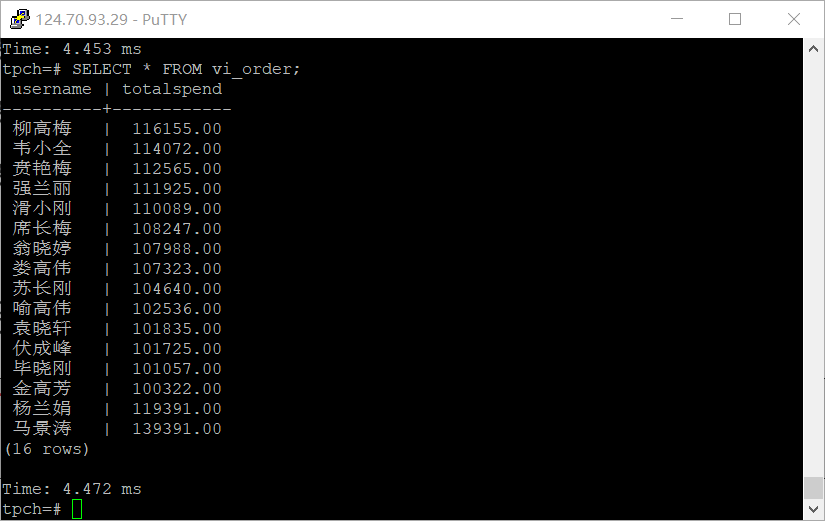
5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



6. 再次对表进行操作后，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？

使用的数据结构不同，分别以行或列为组织形式。

面对更多的OLTP需求时，在更新和删除操作时，行存表的效率更高，读取列的成本不易。

面对更多的OLAP需求时，在统计操作或需要数据有较高的压缩比时，列存表的效率更高，读取列的成本一样。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

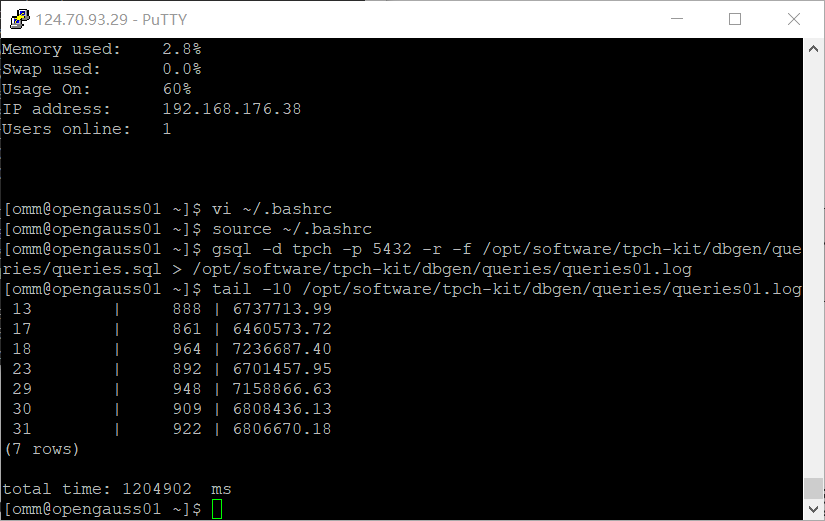
全量物化视图只支持全量刷新，不支持增量更新。增量物化视图同时也支持对数据的增量更新，但其支持场景较少。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

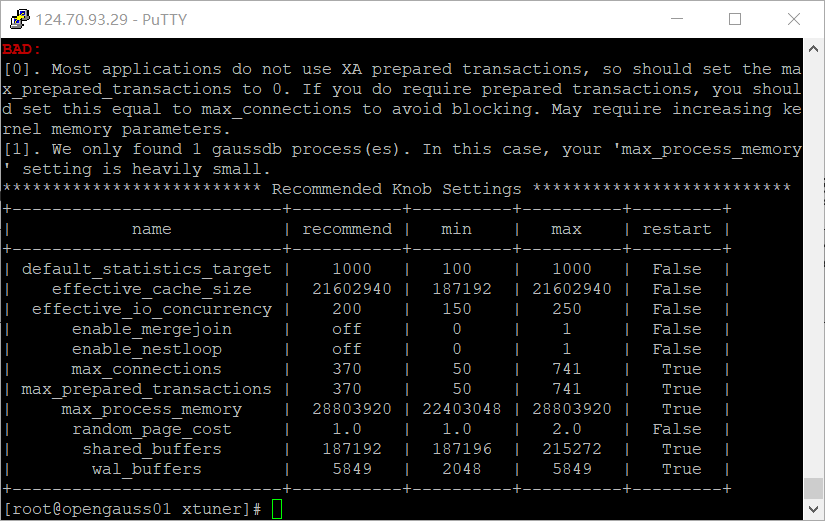
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

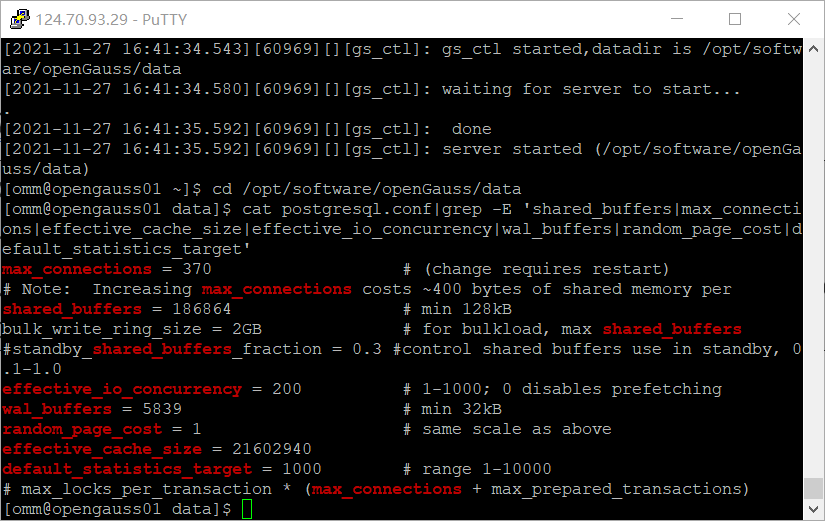
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

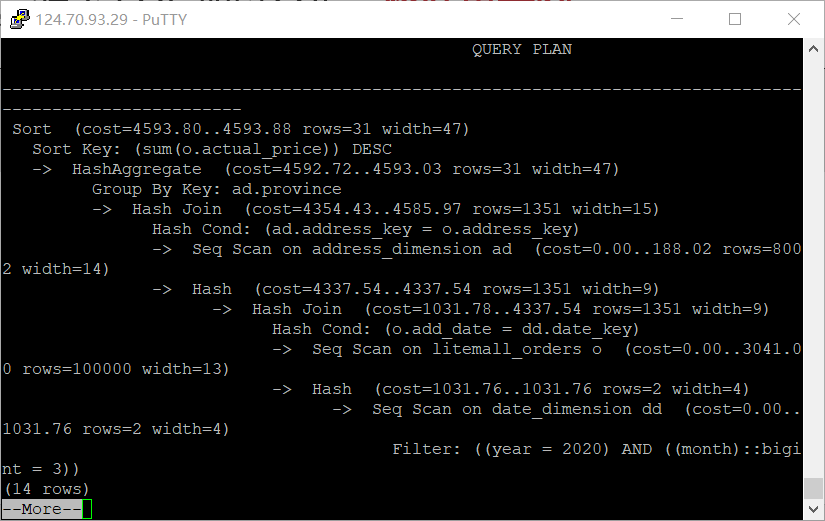
AND o.add\_date = dd.date\_key

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

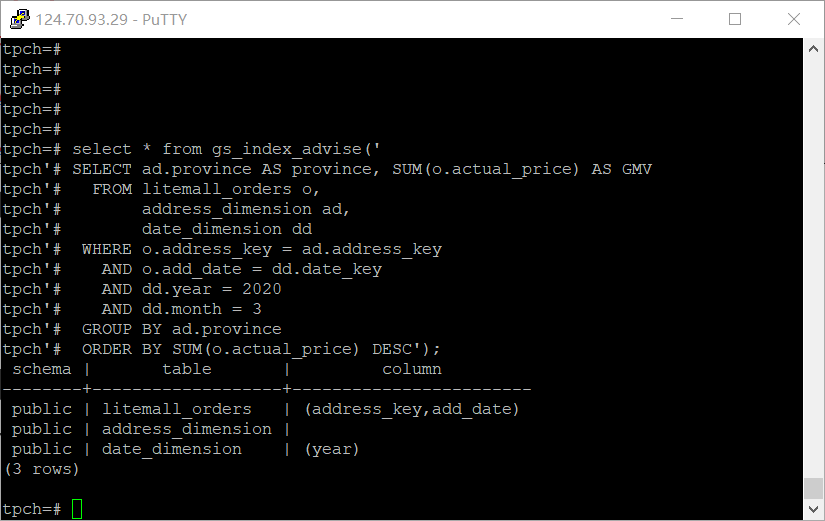
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

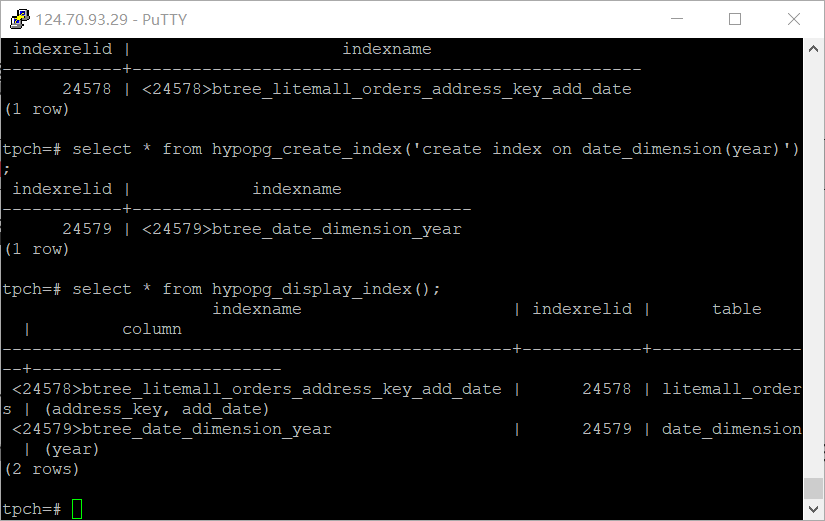
GROUP BY ad.province

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。

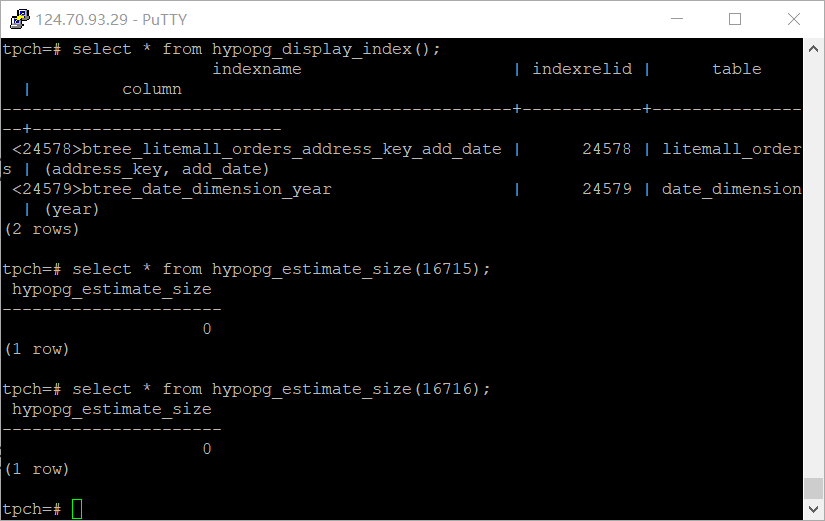
select \* from hypopg\_display\_index();



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

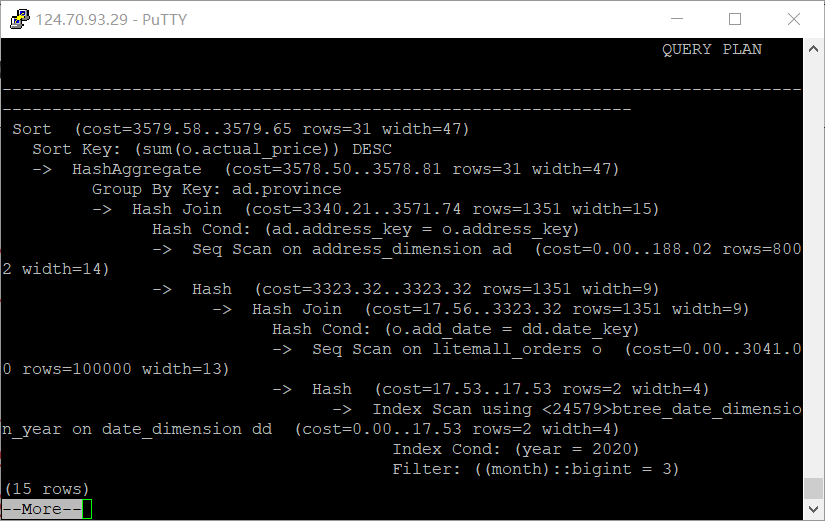
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

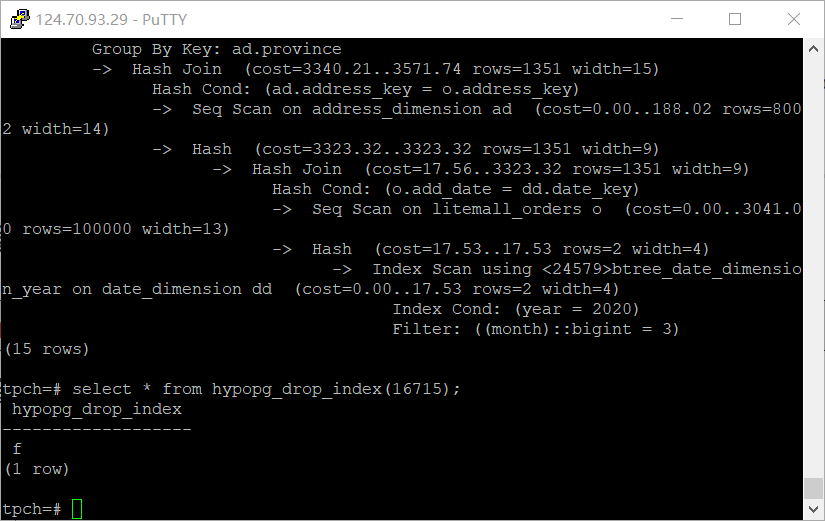
GROUP BY ad.province

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



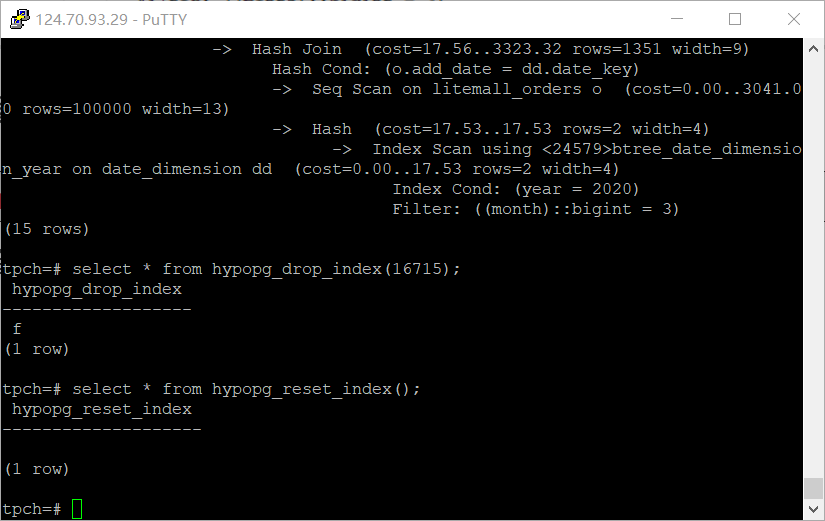
6. 删除某一个索引虚拟列，将执行结果截图。

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



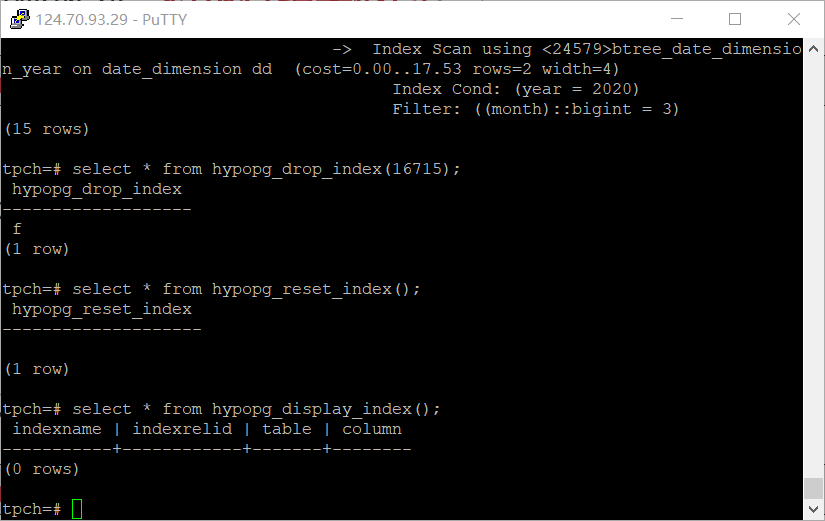
7. 删除所有索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

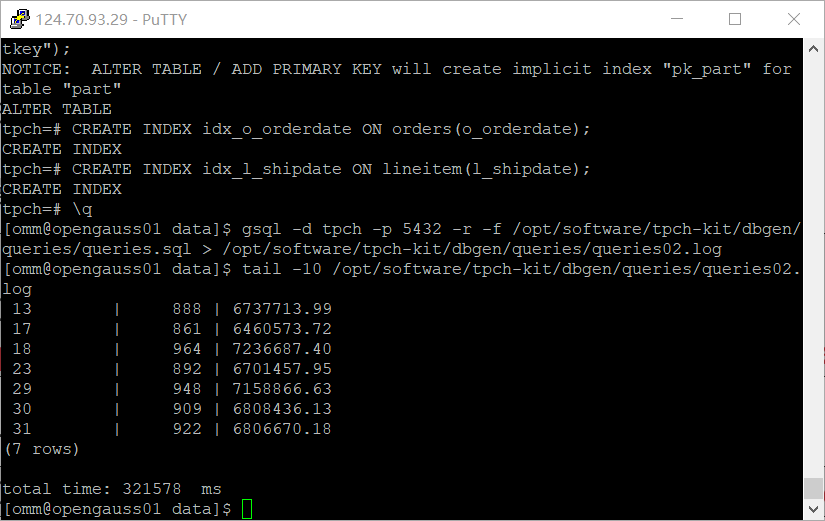
select \* from hypopg\_display\_index();



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

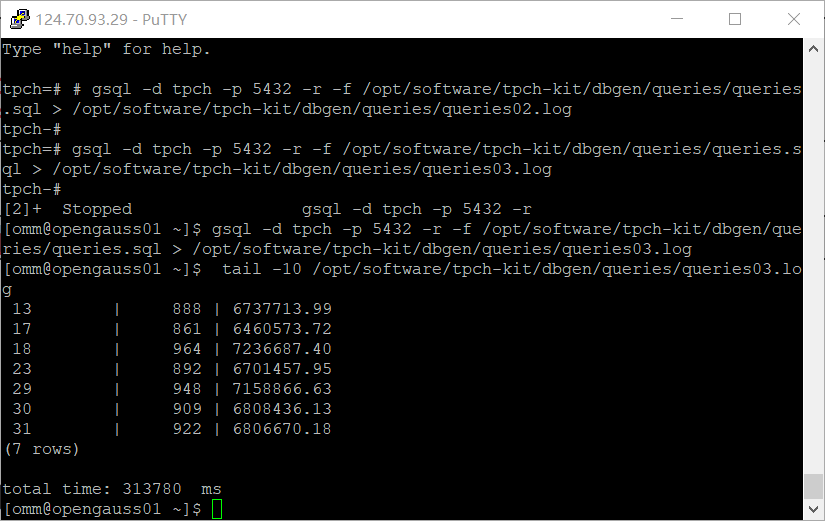


挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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

又为表添加了两条索引。



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？

Max\_connections,shared\_buffers,effective\_io\_concurrency,wal\_buffers,random\_page\_cost,

Effective\_cache\_size,default\_statistics\_target等。可以提高语句的执行效率，减少缓存溢出等事件的可能型，提高连接的稳定性和数据的传输效率。

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？

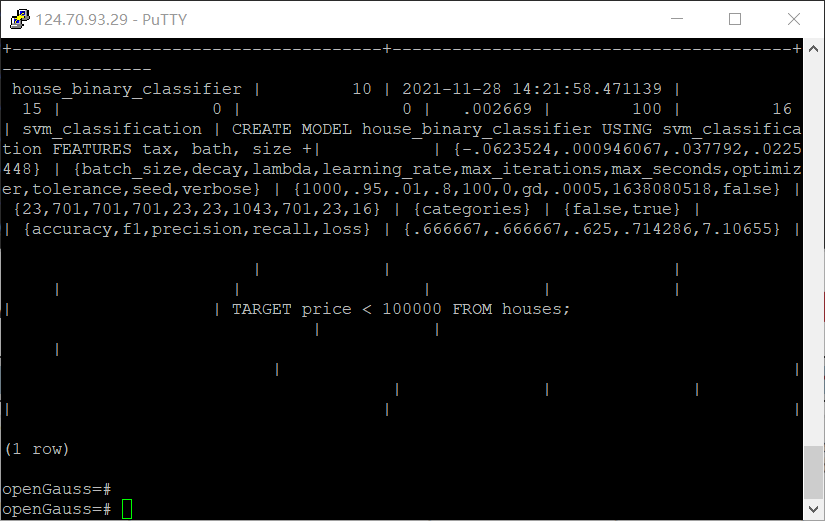
索引可以提高sql语句的执行效率，减少不必要的检索行数。有利于表与表之间的连接操作的（JOIN）快速执行。可以减少分组和排序中所需要的时间，可以优化聚合查询。避免索引覆盖。利用唯一性约束，保证数据的唯一性。

可优化：配备高级的硬件设施，提高高速缓存的命中率，使用更为高效合理的数据结构的设计（比如表）。定时进行碎片整理，清除掉不需要的数据，使用分表技术。对sql语句进行优化等。

# 关卡四、openGauss的DB4AI特性应用

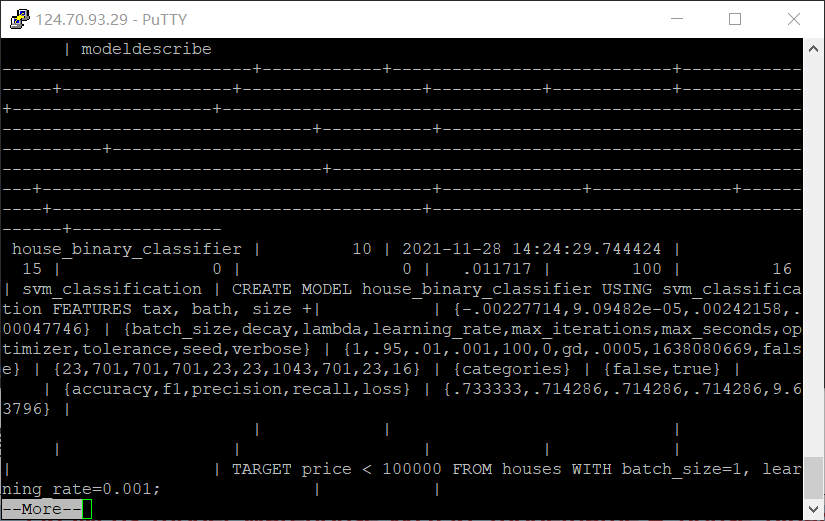
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



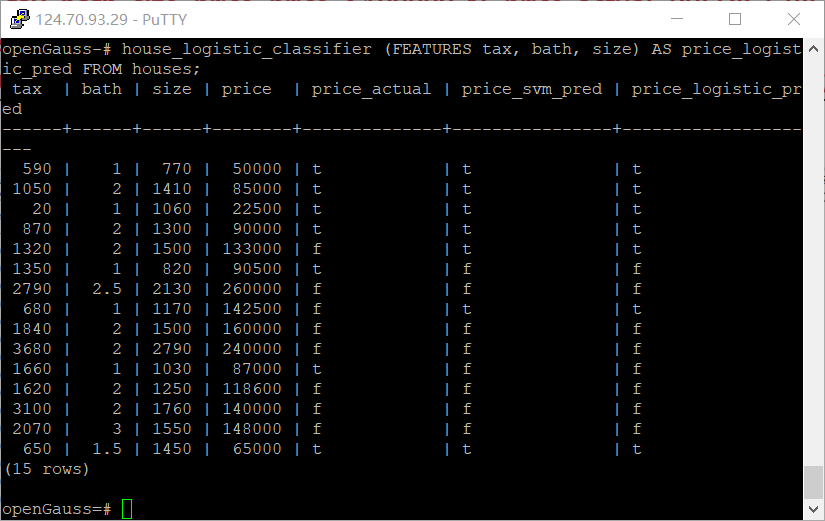
任务二：观察新模型的信息，将执行结果截图。

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？

分类的结果是具体的类，可以出现的结果往往是离散的。

回归的结果不是具体的分类，而是具体的，可以是连续的值。

实践思考题2：什么是SVM算法？

SVM（support vector machine）为支持向量机。在平面上可以用线性函数进行直接分类，

但复杂的数据及其模型难以在普通的平面上分类，于是把数据投影到超平面上去找可以

找到分离超平面，并且是几何距离最大的超平面达到分类的效果。

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？

使用混淆矩阵整体体现分类效果：使用如下参数

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

Accuracy = (TN + TP)/(TP + FP + FN + TN),为正确判断的概率

Precision = TP/(FP + TP)，判断为阳性时的准确率

Recall = TP/(TP + FN)，对真阳性时准确判断的概率

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？

均方根误差：

为观测值与真实值间的偏差。

标准差：

方差的算术平均根。可以用于衡量一组数值的离散程度。

均方误差：

MSE是真实值与预测值的差值的平方然后求和平均。

因为通过平方的形式便于求导，所以常被用作线性回归的损失函数。

平均绝对误差：

为绝对误差的平均值，可以更好地反映预测值误差的实际情况。