openGauss AI特性创新实践课



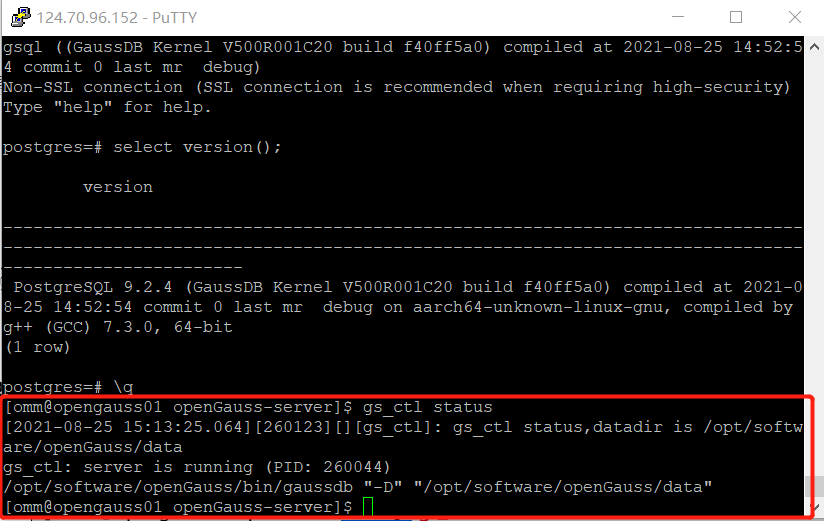
华为技术有限公司

# 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：

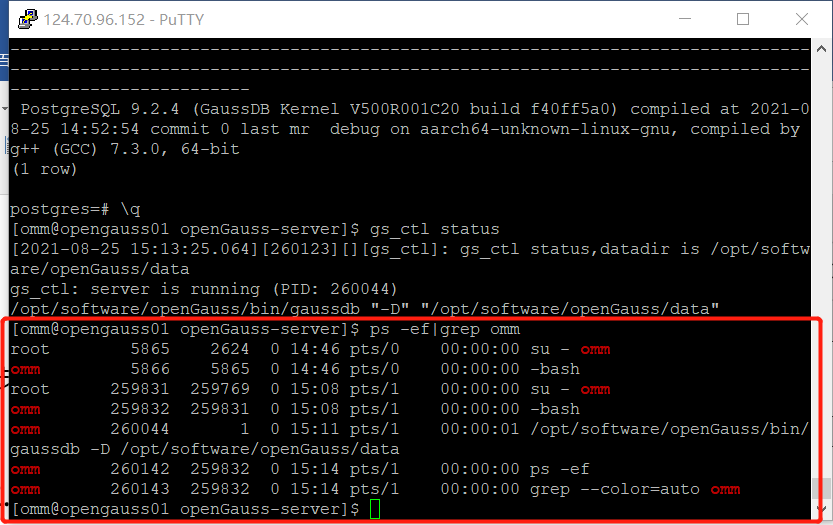
任务一：数据库状态验证

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？

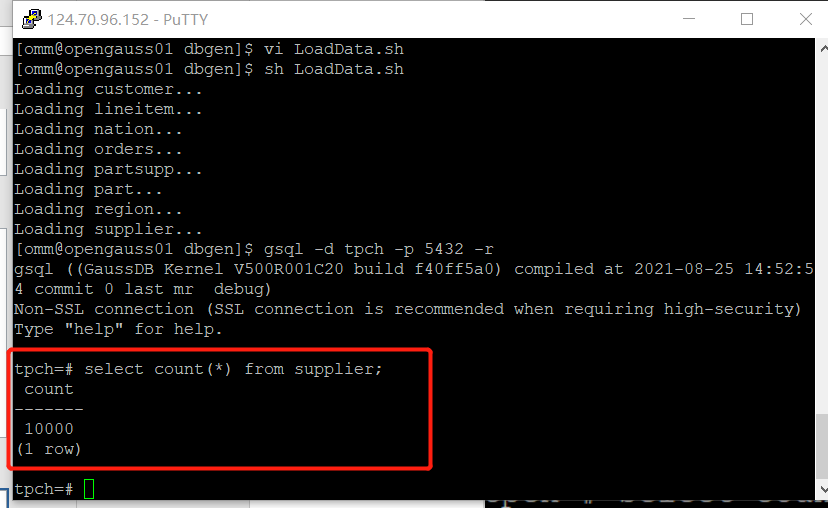
灵活性较大，可以自行制定安装路径和选择安装的数据库版本。编译安装过程中，可以自行调整优化编译参数和配置，最大化地定制安装结果，提高性能。通过源码编译的数据库，卸载和迁徙时既方便又安全。

# 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：

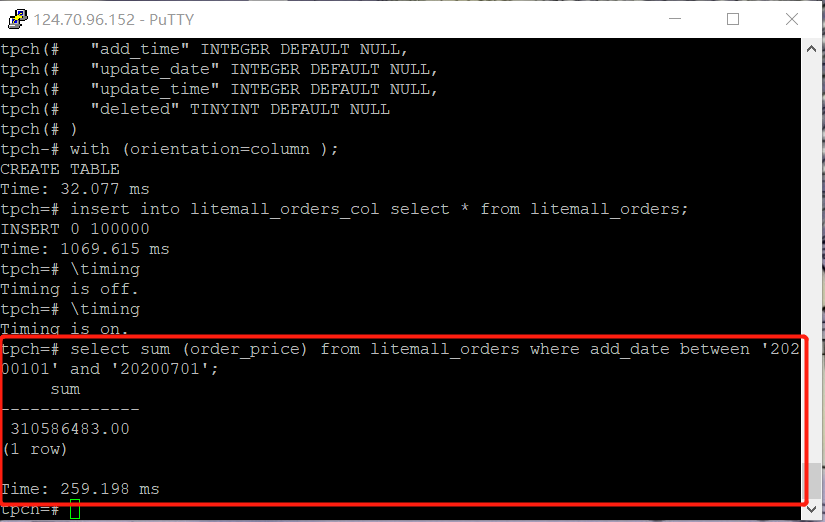
select count(\*) from supplier;;



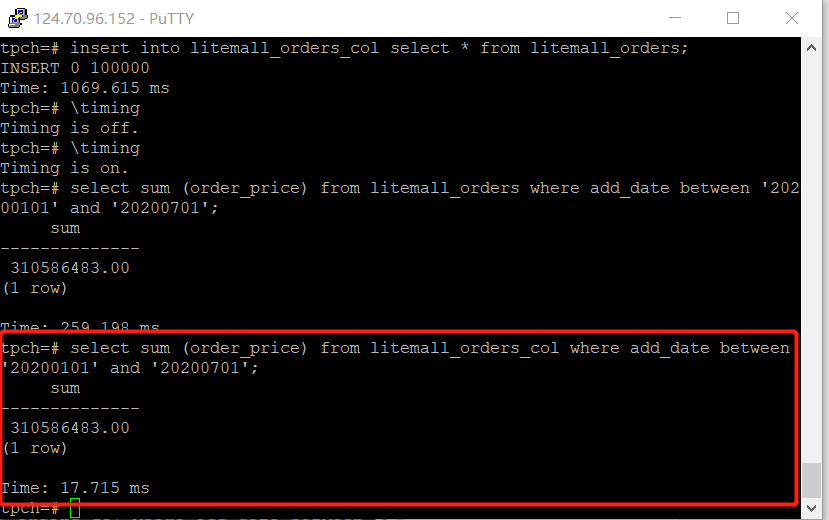
任务二：行存表与列存表执行效率对比

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';

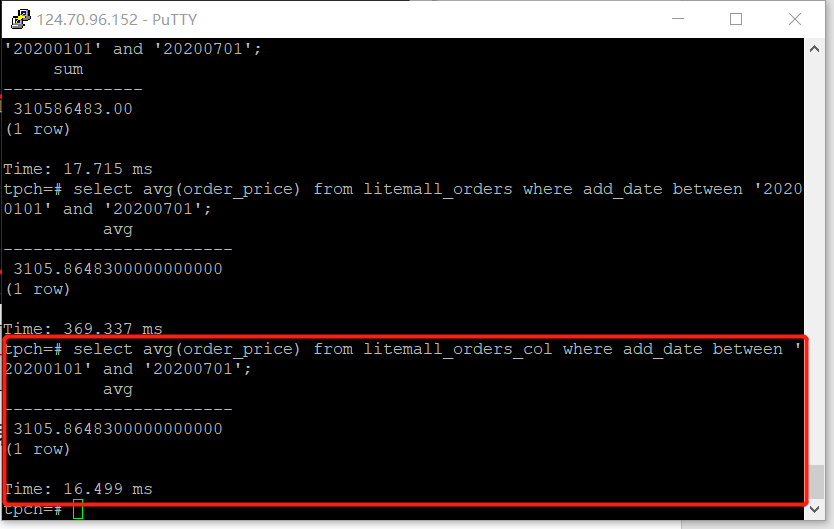


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

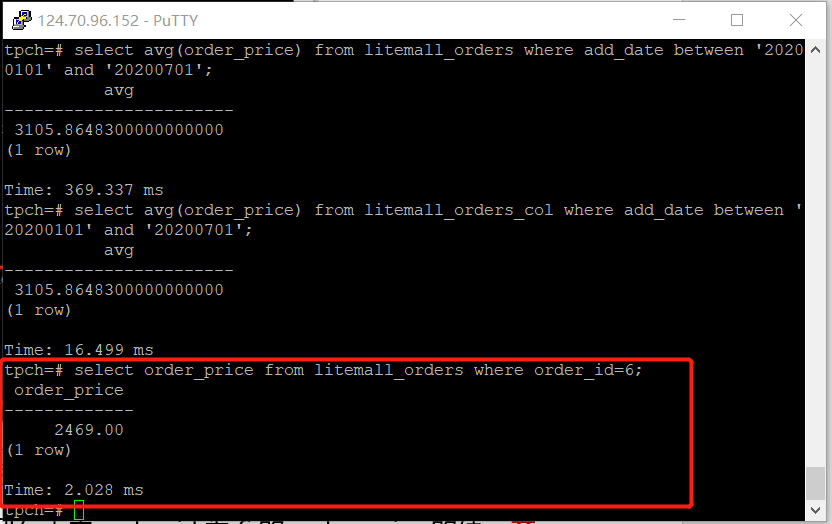


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

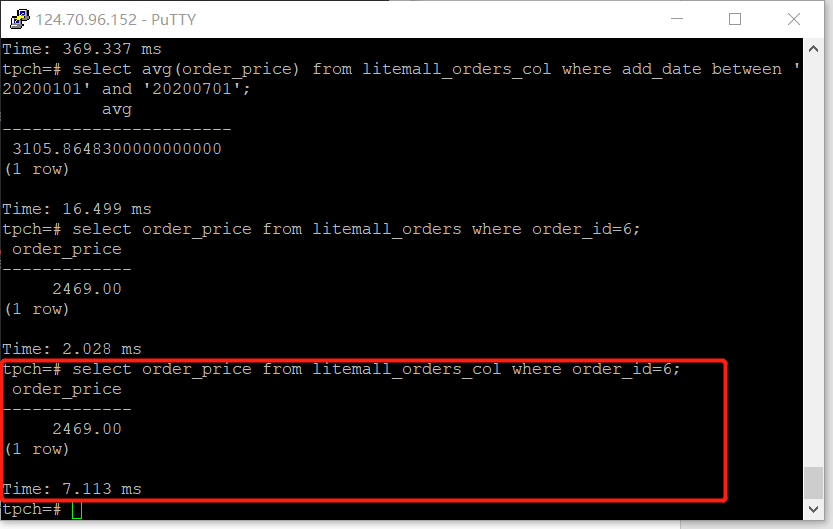


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

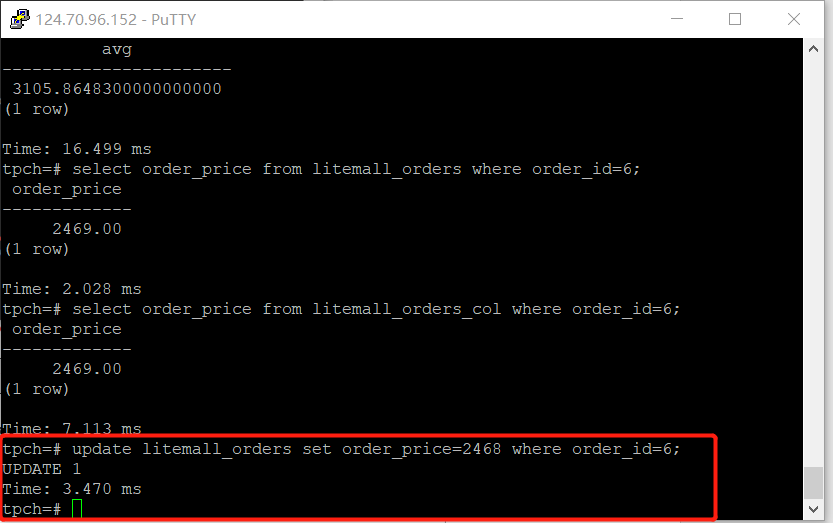


select order\_price from litemall\_orders\_col where order\_id=6;

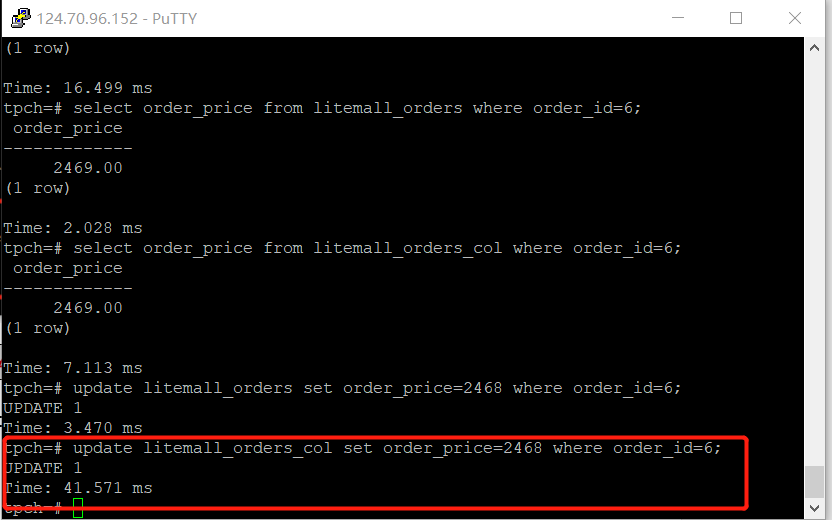


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



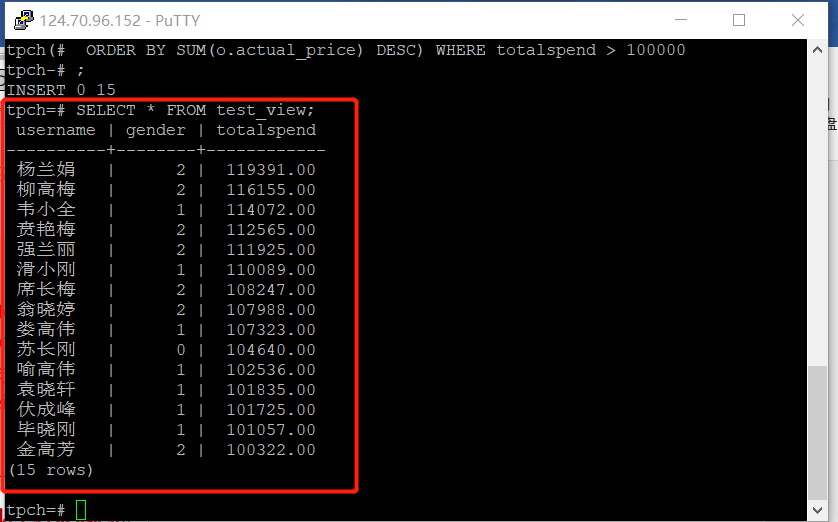
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

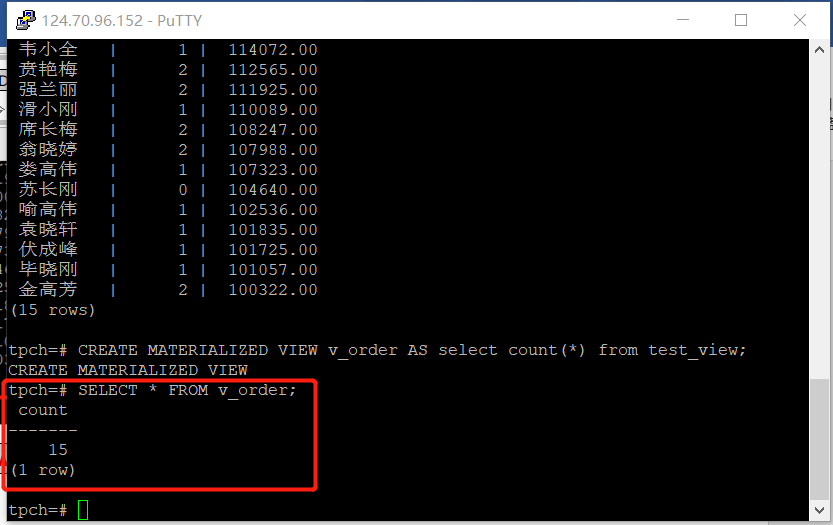
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



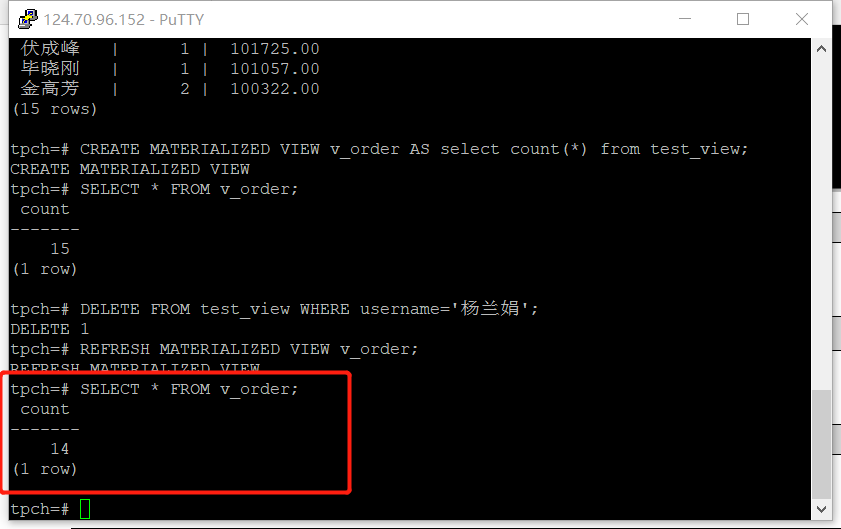
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



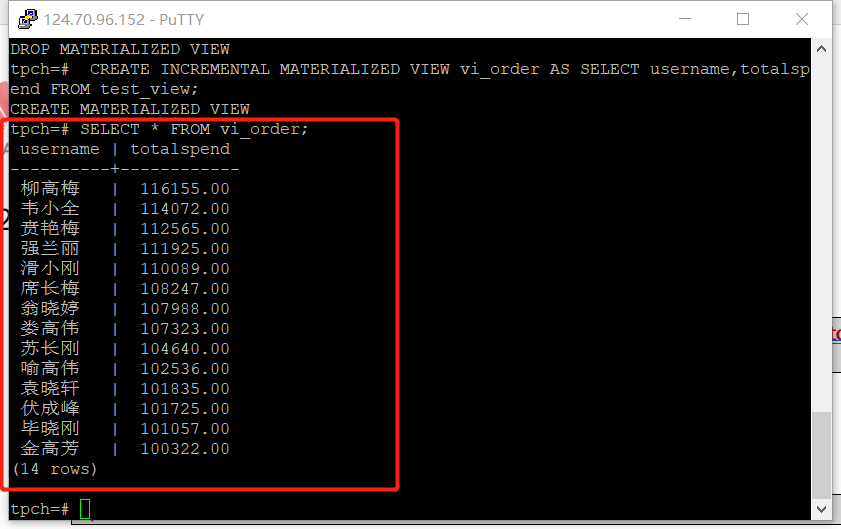
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



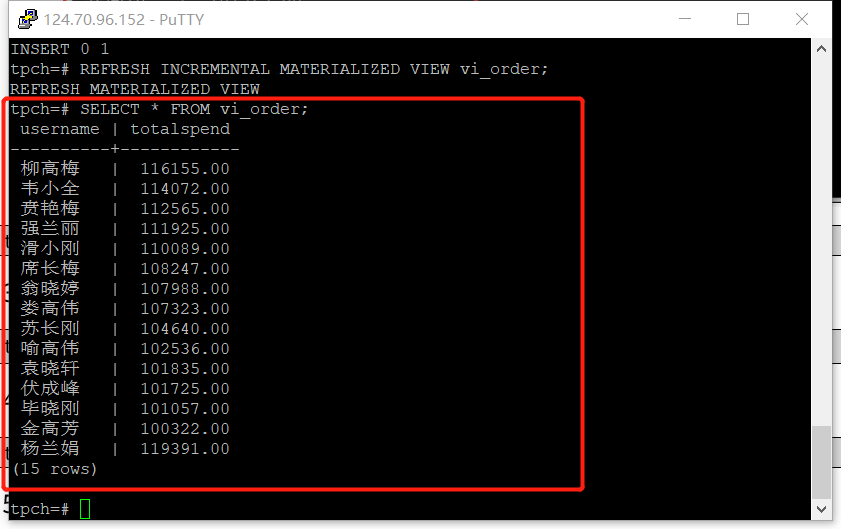
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



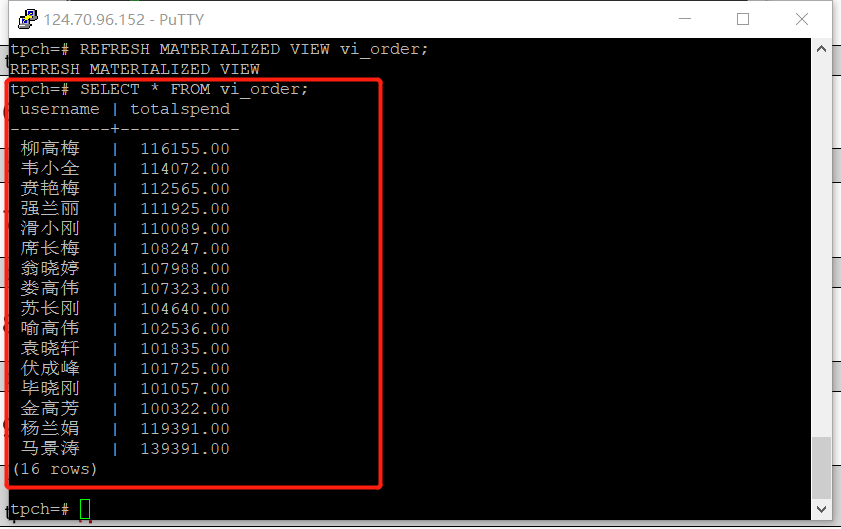
5. 对表进行操作后，增量刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



6. 全量刷新物化视图。

tpch=# REFRESH MATERIALIZED VIEW 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？

两种表对应两种存储方式，行存储和列存储两种数据存储方式有差异，查询时因为存储结构的不同，执行时间也会不同。

行存储是在指定位置写入一次，就能够保证数据的完整性；列存储需要把一行记录拆分成多个单列进行保存，写入次数比行存储多。行存储通常将一行数据完全读出，列存储每次读取的数据是集合中的一段或者全部。因此读写数据时行存表效率更高。

因为列存表的的数据是同质的，它让数据解析变得容易；而行存表在一行记录中保存了多种类型的数据，数据解析需要在多种数据类型之间频繁地进行转换，因此进行统计查询和数据解析时列存表效率更高。

执行单个值修改和查询时，行存表效率更高；执行总和与平均值等统计查询时，列存表效率更高。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量物化视图仅支持对创建好的物化视图做全量刷新，不支持增量刷新；而增量物化视图支持两种刷新。

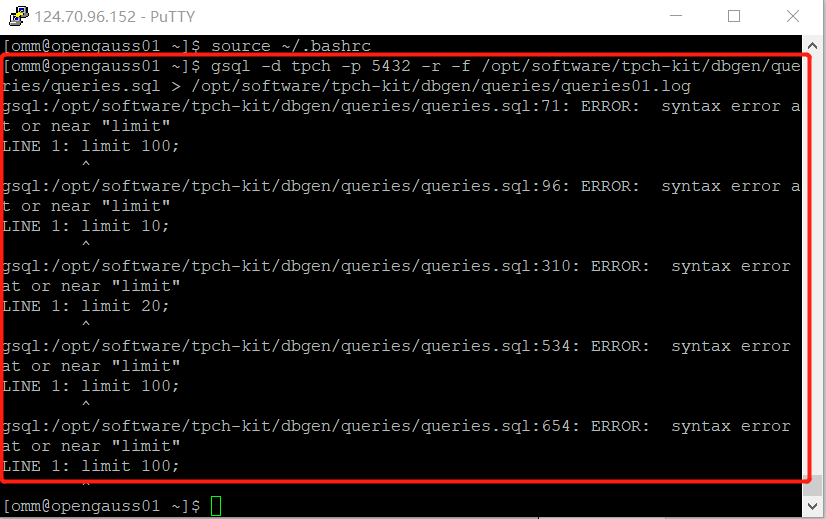
增量物化视图支持场景较小，而全量物化视图支持场景较多。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

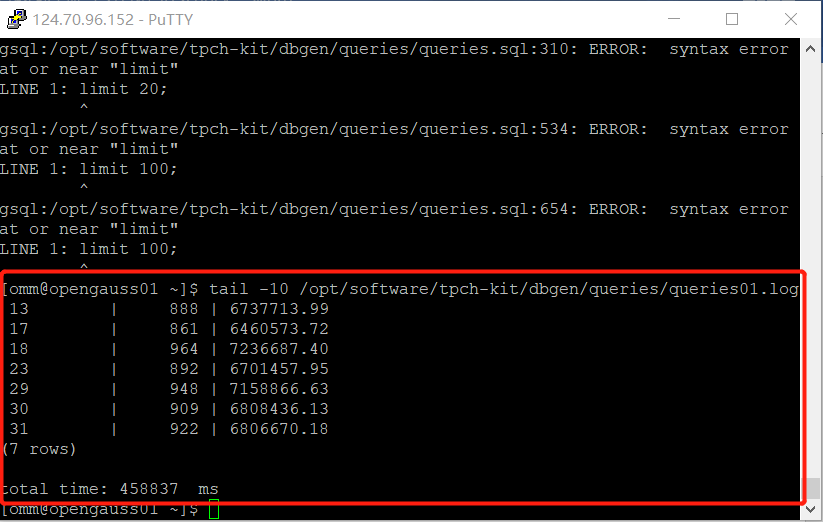
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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



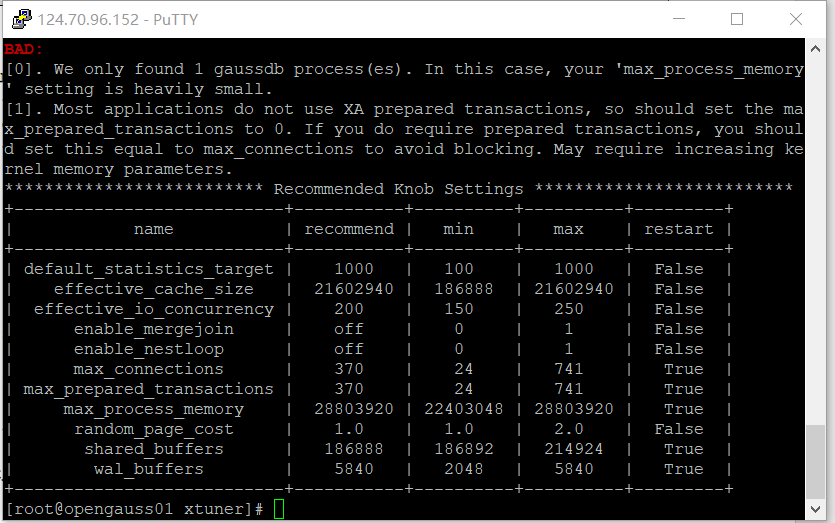
查看queries01.log。

[omm@opengauss01 ~]$ **tail -10 /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

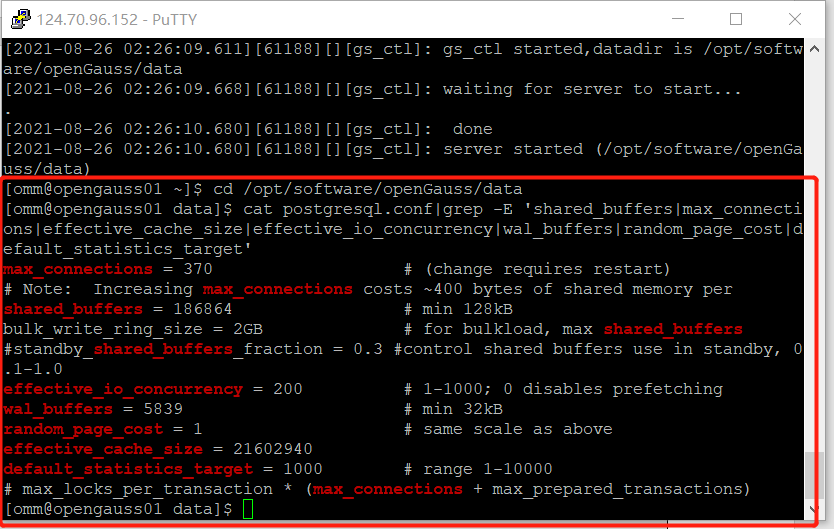
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

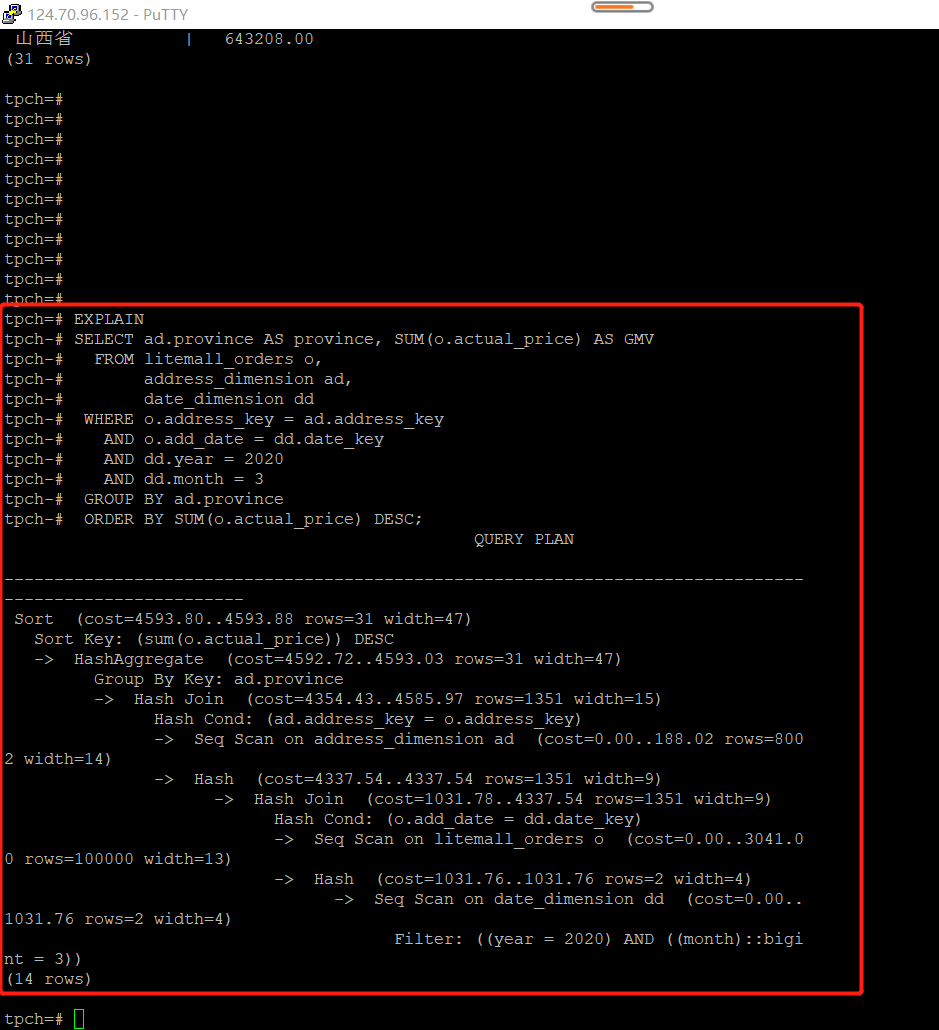
AND o.add\_date = dd.date\_key

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

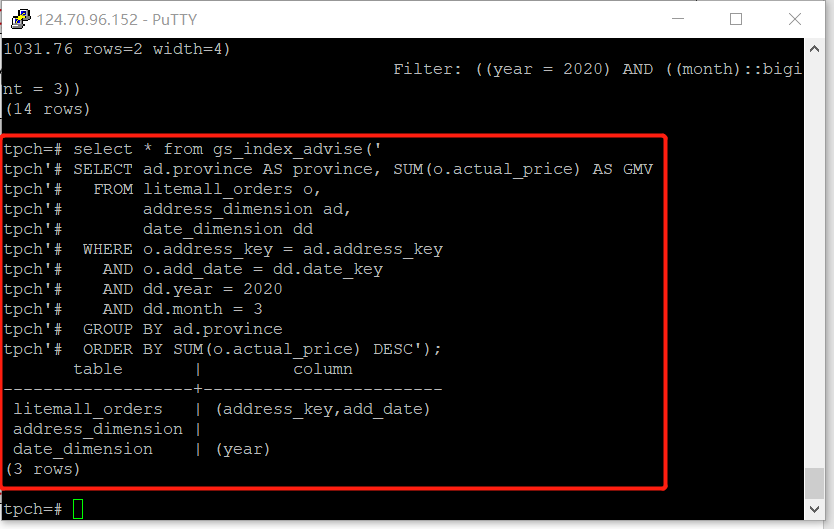
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

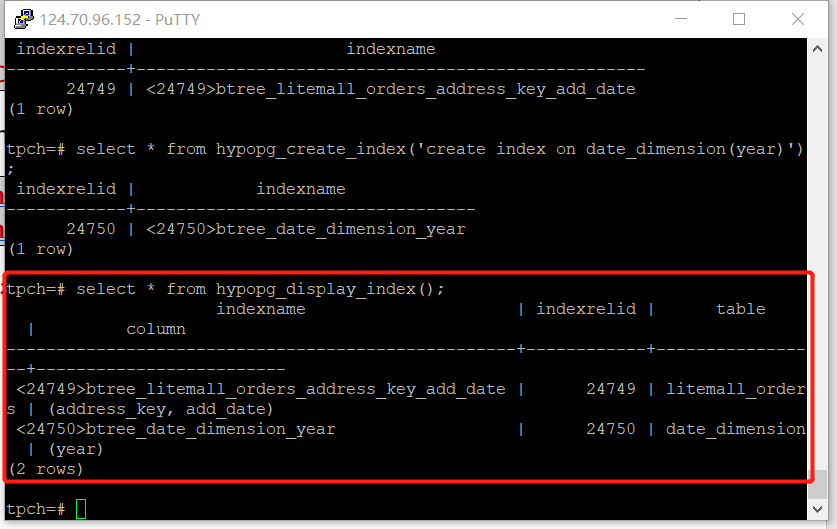
GROUP BY ad.province

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。

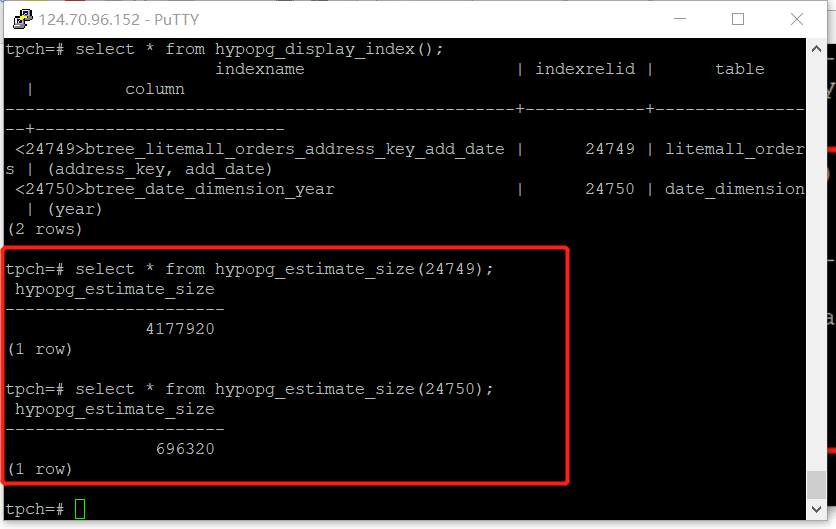
select \* from hypopg\_display\_index();



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

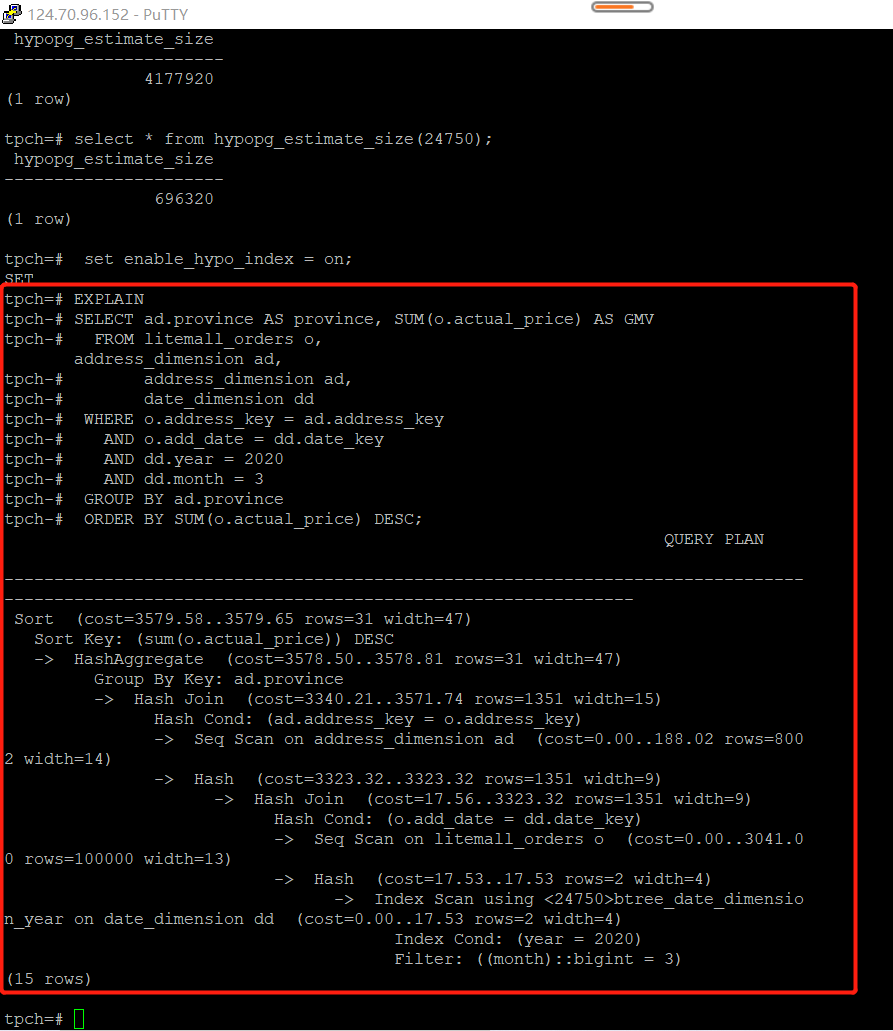
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

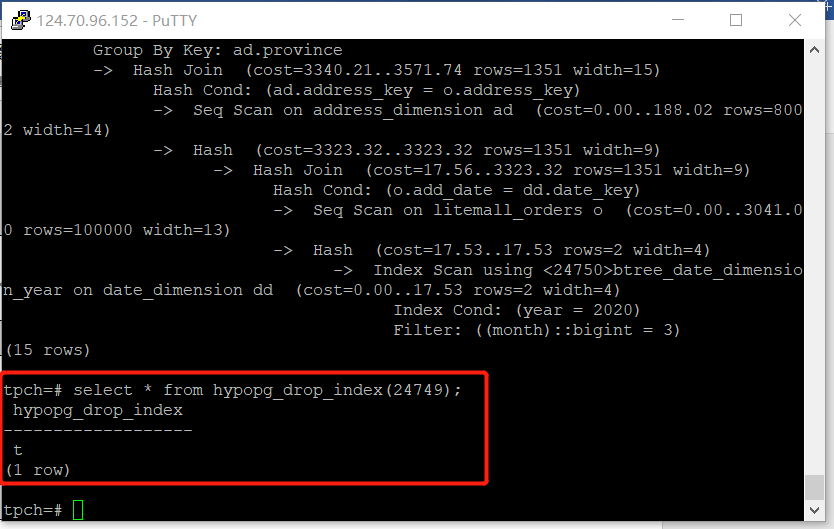
GROUP BY ad.province

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



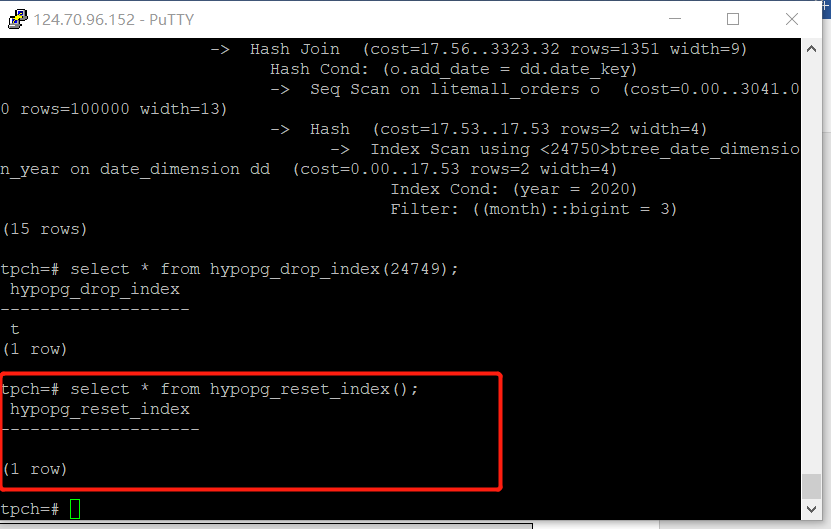
6. 删除某一个索引虚拟列，将执行结果截图。

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



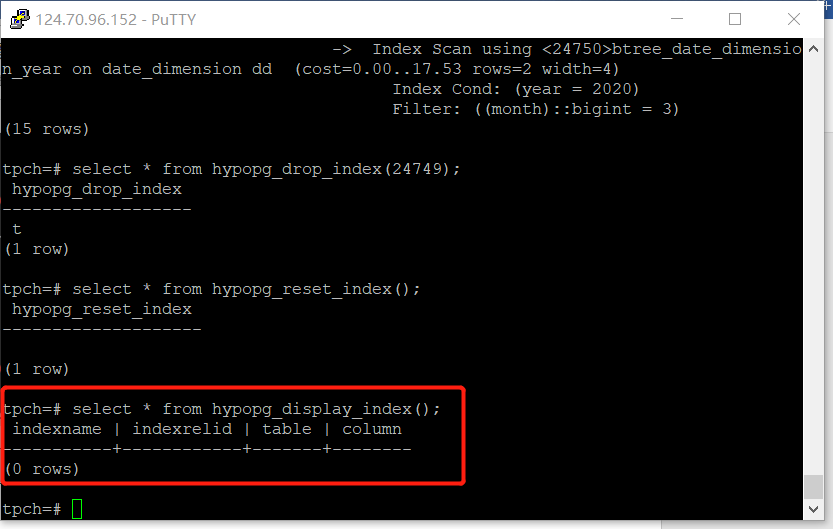
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

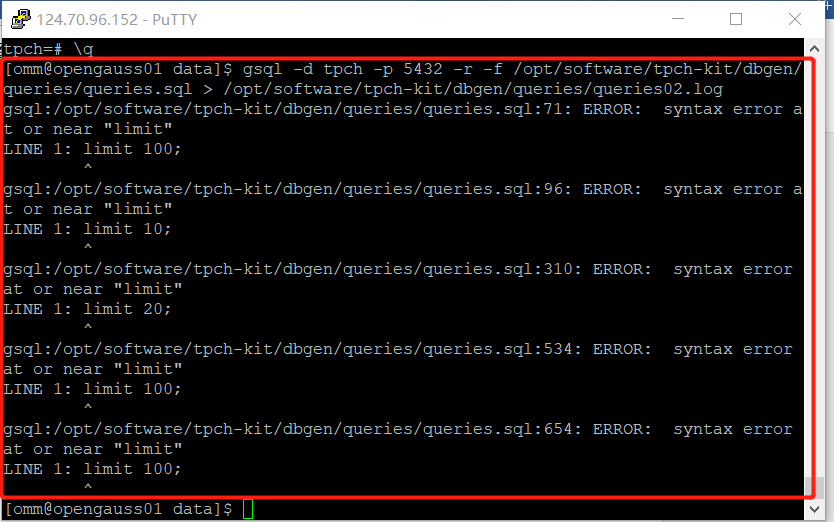
select \* from hypopg\_display\_index();



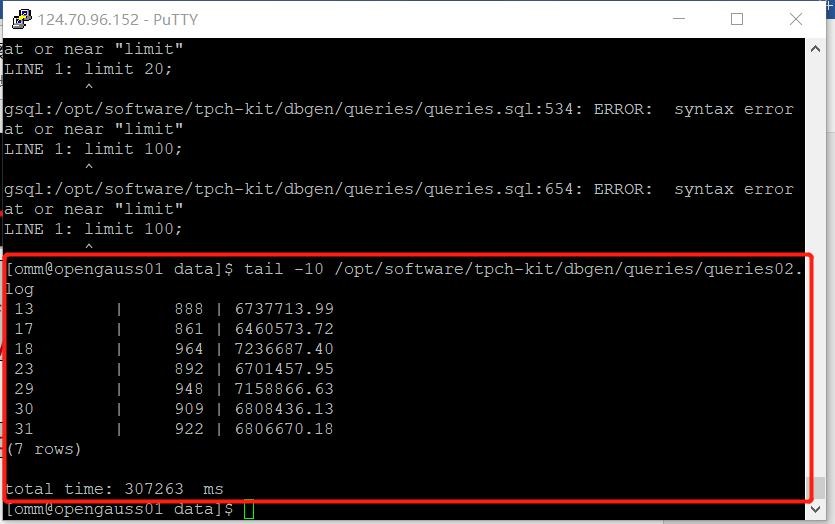
任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

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



查询queries02.log文件：



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？

对shared\_buffers、max\_connections、effective\_cache\_size 、effective\_io\_concurrency、wal\_buffers、random\_page\_cost、default\_statistics\_target进行了优化。

增大shared\_buffers使整个工作数据集可以保留在缓存中，max\_connections用来设置最大连接用户数，effective\_cache\_size提供可用于磁盘高速缓存的内存量的估计值，effective\_io\_concurrency 提高IO性能，wal\_buffers定义缓冲区的大小，random\_page\_cost设置从数据文件上随机读取一个数据块的执行成本，default\_statistics\_target设置默认的收集优化器统计数据的目标值，值越大，分析操作的执行的时间越长，扫描的数据行的个数越多，得到的优化器统计数据就越准确。

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？

索引的使用能在执行SQL时提高数据查询的效率,降低数据库的IO成本，除此之外，通过索引对数据进行排序,能降低数据排序的成本。

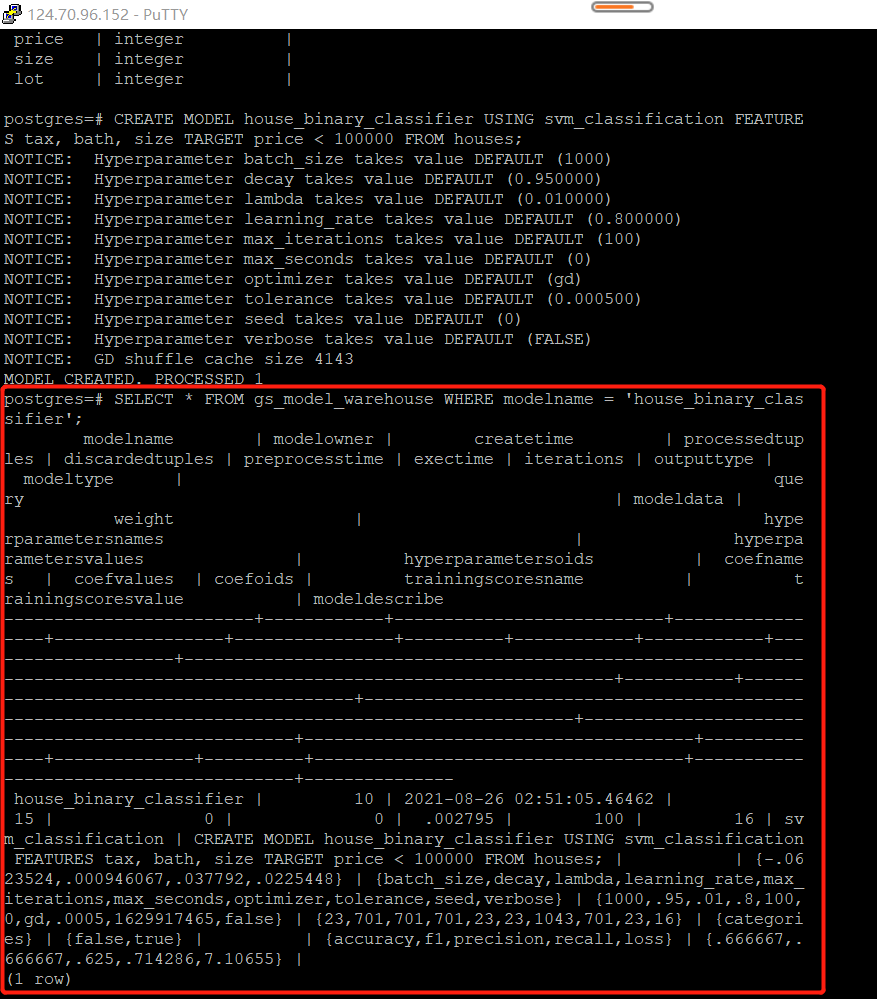
除此之外，分库分表、读写分离也能对数据库进行优化。把一个库拆分为多个库，部署在多个数据库服务上，主库承载写入请求，而每个主库都挂载至少一个从库，由从库来承载读请求。

对写少读多的请求，可以引入缓存集群，即在写数据库的时候同时写一份数据到缓存集群里，然后用缓存集群来承载大部分的读请求，就可以用更少的机器资源承载高并发。

# 关卡四、openGauss的DB4AI特性应用

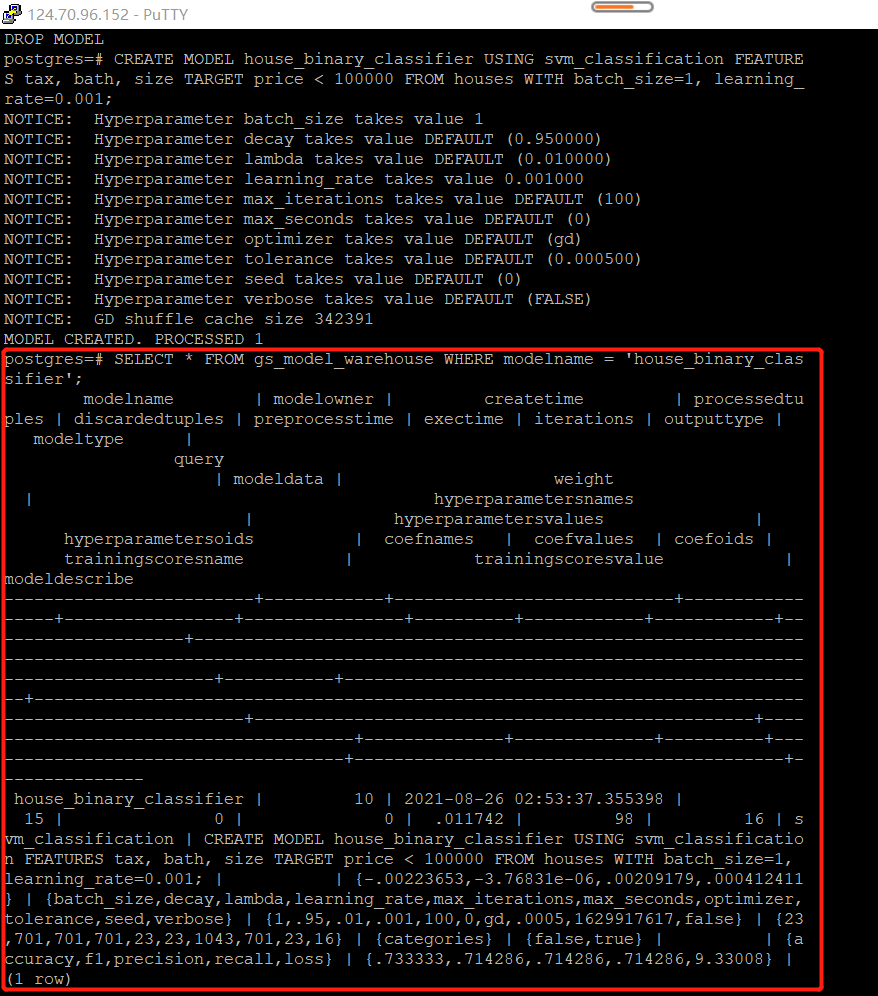
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



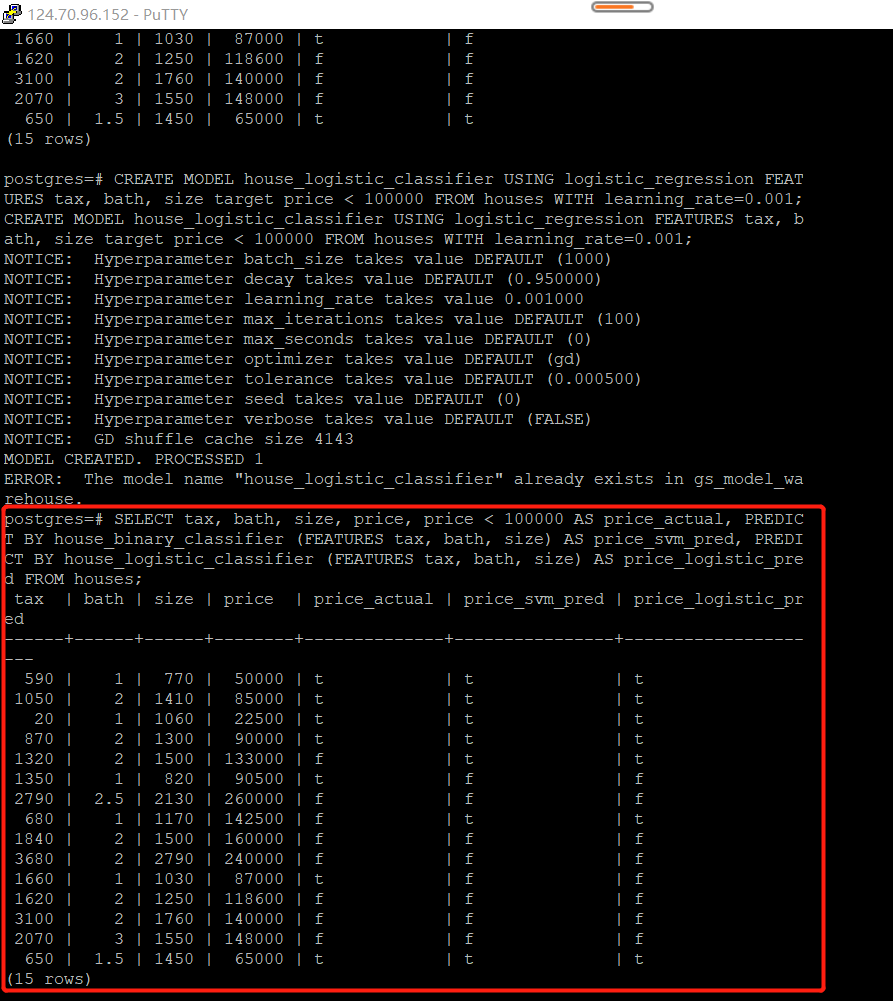
任务二：观察新模型的信息，将执行结果截图。

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？

回归模型预测的值是对真实值的一种逼近，分类模型分类模型将输出离散化，结果通常为离散值，只有一个，没有回归模型逼近的概念。

实践思考题2：什么是SVM算法？

SVM算法是分类与回归分析中分析数据的监督式学习模型算法。在训练中，一组训练实例会被标记为属于两个类别中的一个或者另一个，而SVM算法创建一个将新的实例分配给两个类别之一的模型，使其成为非概率二元线性分类器，它将实例表示为空间中的点，这样映射就使得单独类别的实例被尽可能明显的分开，然后，将新的实例映射到同一空间，并根据落在间隔的哪一侧来预测所属类别。

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？

准确率：预测正确的结果占总样本的百分比

精确率：在所有被预测为正的样本中实际为正的样本的概率

召回率：在实际为正的样本中被预测为正样本的概率

P-R曲线：描述精确率/召回率变化的曲线

F1 分数：精确率和召回率的调和平均数

混淆矩阵：以矩阵形式将数据集中的记录按照真实的类别与分类模型作出的分类判断两个标准进行汇总。

ROC：受试者工作特征曲线

AUC：ROC曲线下的面积

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：用真实值减去预测值，然后平方之后求和平均，线性回归的损失函数

RMSE 均方根误差：与MSE本质一样，用于数据更好的描述

MAE：平均绝对误差

R-square：判断模型拟合效果