openGauss AI特性创新实践课



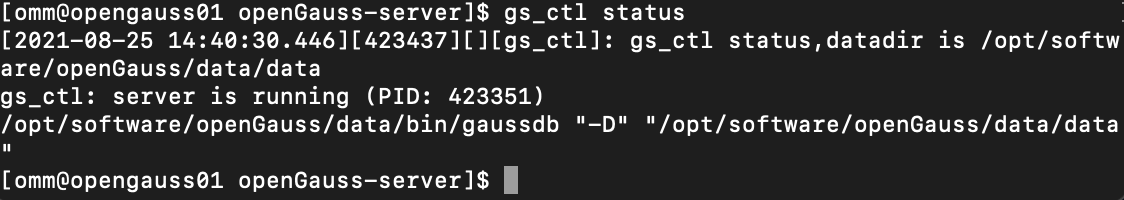
华为技术有限公司

# 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：

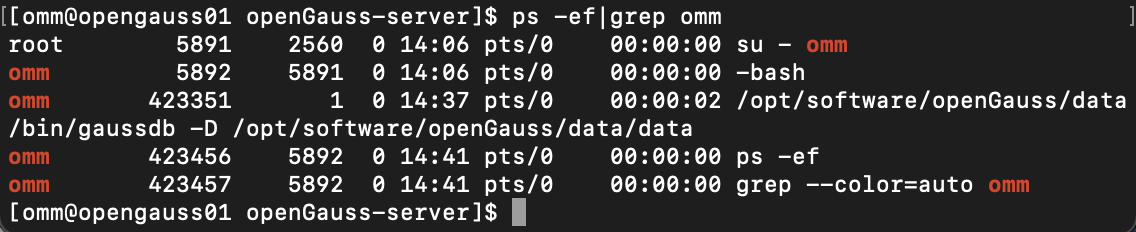
任务一：数据库状态验证

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？

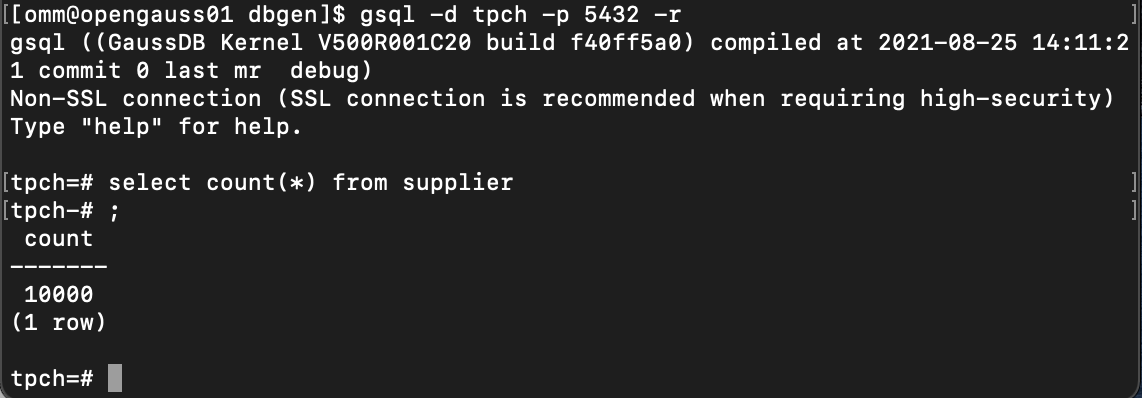
在不同计算机上，Linux内核版本可能不一样，硬件也可能不一样的X86、X64、ARM等，编译器的选项不同，即使内核、硬件配置相同，也没法保证系统中依赖库的版本一致，这样也就无法保证二进制包能正常运行。通过源代码编译可以允许用户选择软件编译配置。

# 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：

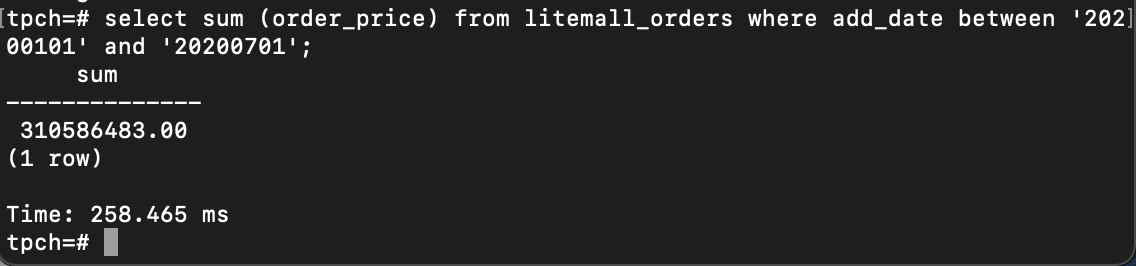
select count(\*) from supplier;;



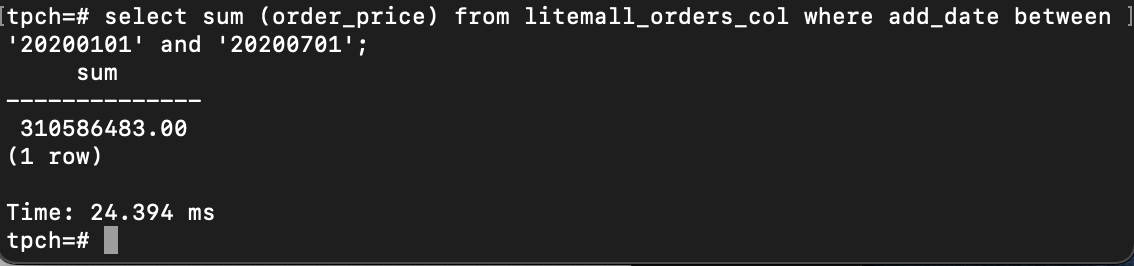
任务二：行存表与列存表执行效率对比

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';

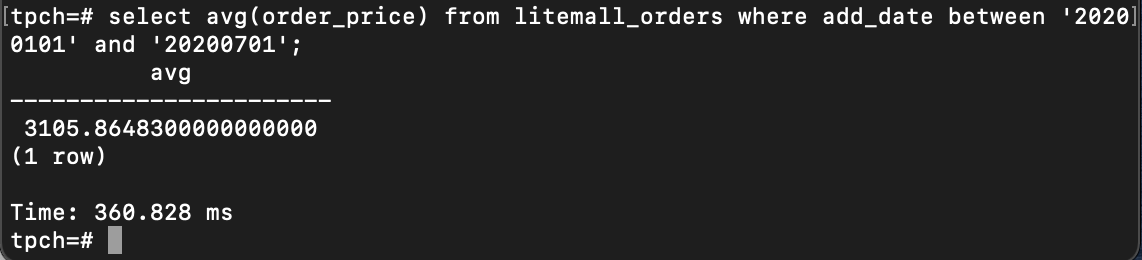


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

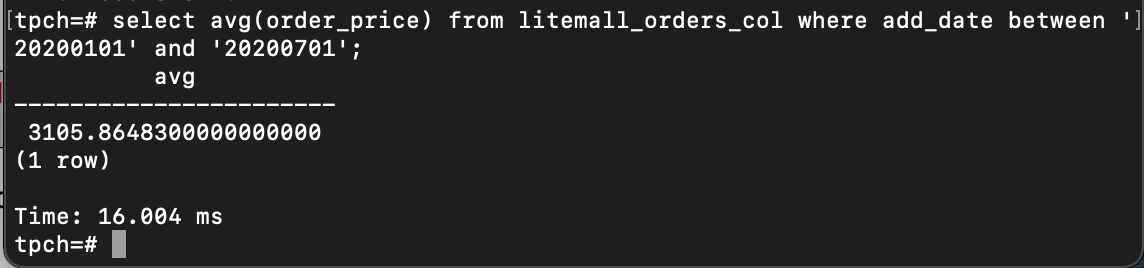


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

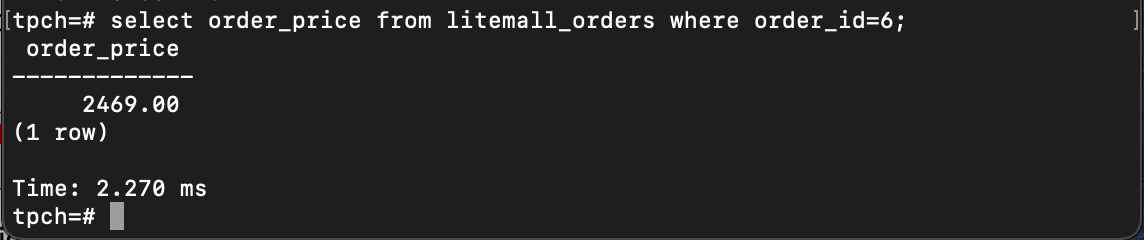


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

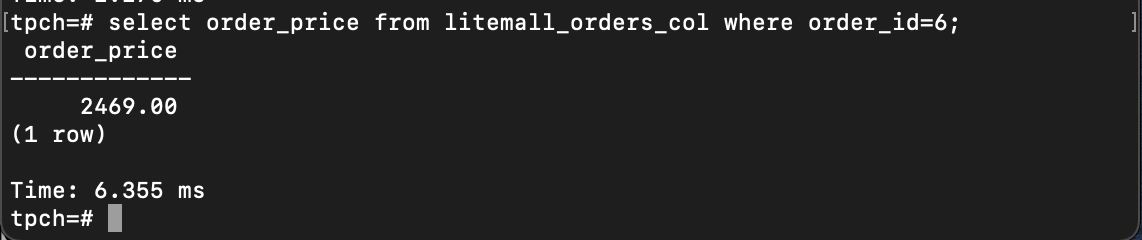


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

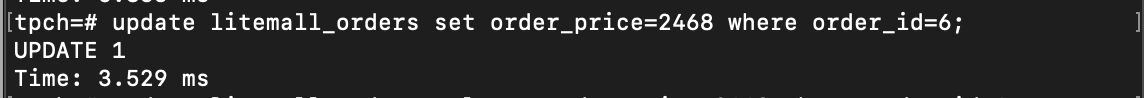


select order\_price from litemall\_orders\_col where order\_id=6;

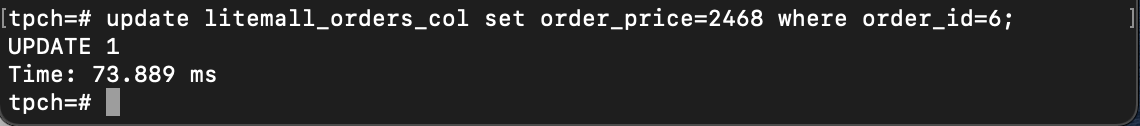


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



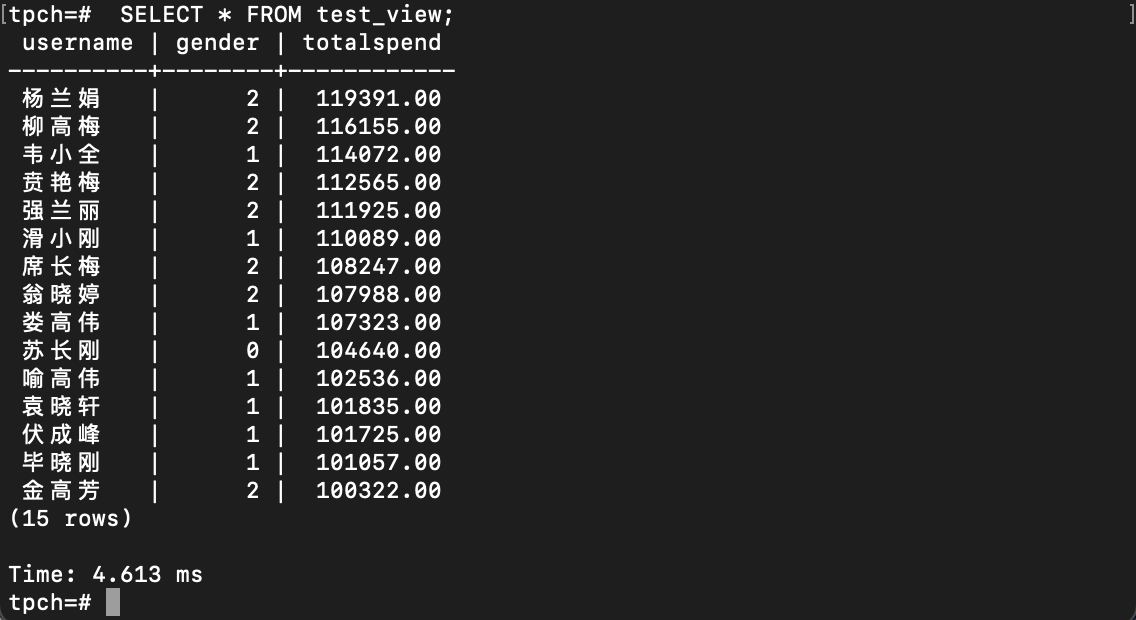
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

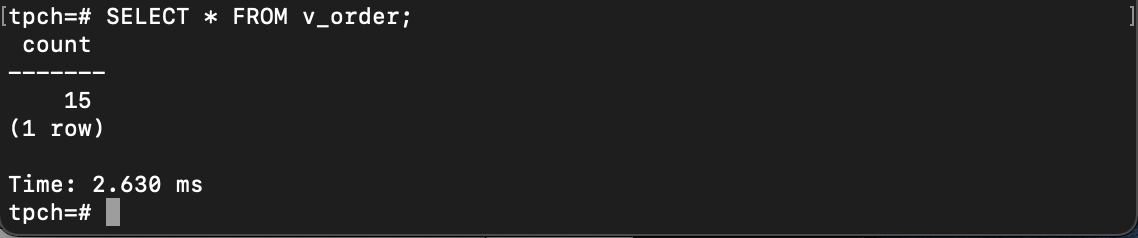
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

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;



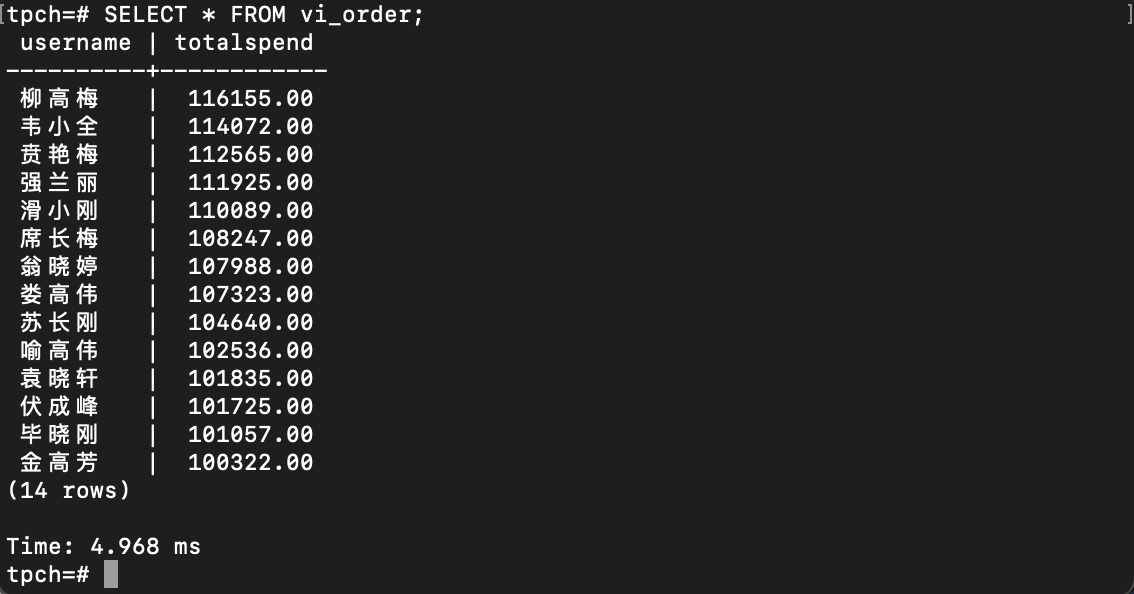
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



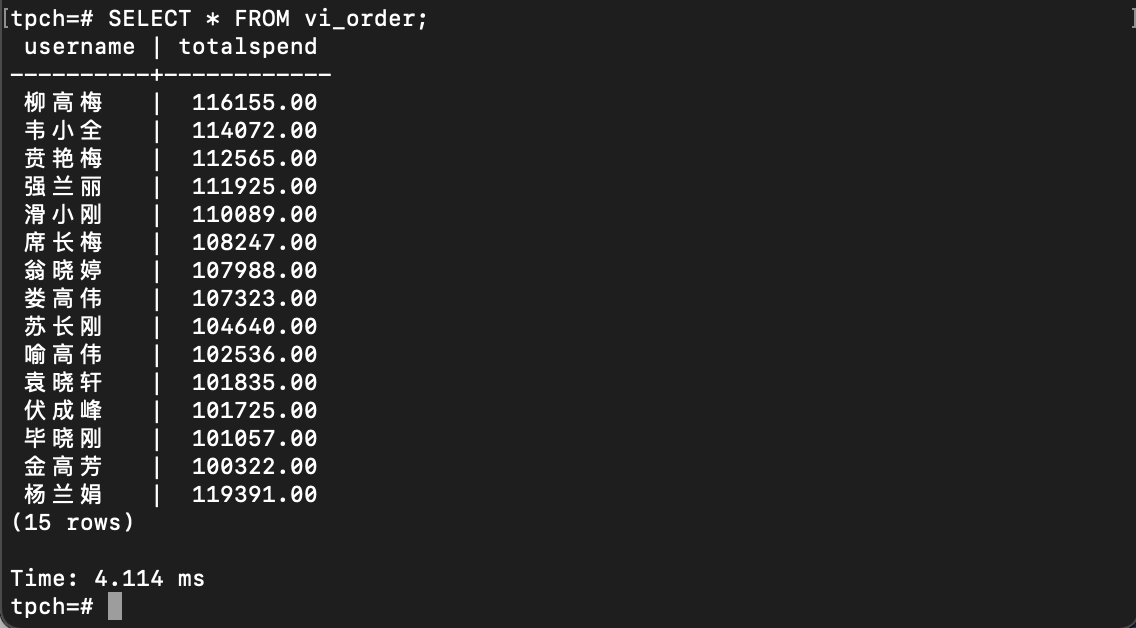
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;

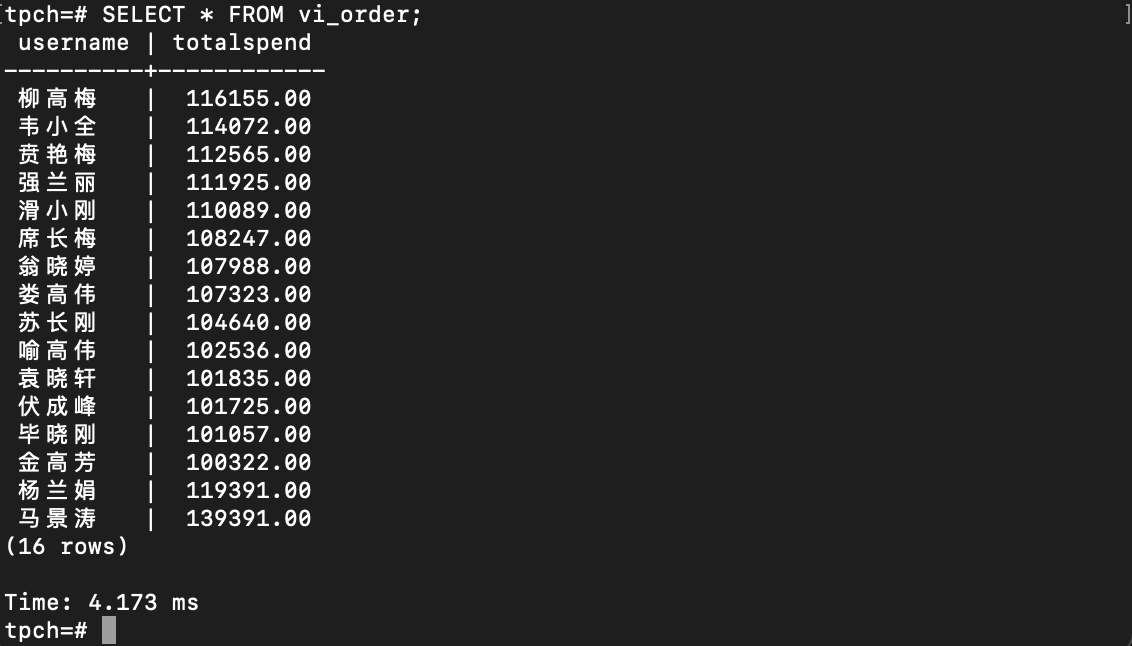


5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



6. 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



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？

因为行存表和列存表存储数据的方式不一样，执行查询的时间也就不一样，行式存储下一张表的数据都是放在一起的，但列式存储下都被分开保存了。行存储1.数据被保存在一起; 2.INSERT/UPDATE容易；而列存储1.查询时只有涉及到的列会被读取; 2.投影(projection)很高效; 3.任何列都能作为索引。行存储的写入是一次性完成，消耗的时间比列存储少，并且能够保证数据的完整性，缺点是数据读取过程中会产生冗余数据，如果只有少量数据，此影响可以忽略；数量大可能会影响到数据的处理效率。列存储在写入效率、保证数据完整性上都不如行存储，它的优势是在读取过程，不会产生冗余数据。

通过上述分析和实验结果，可以看到：在插入和修改数据时，在按键列去查询其他列的信息时，行存表效率高，而在涉及一整列的操作时，列存表会比较快。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

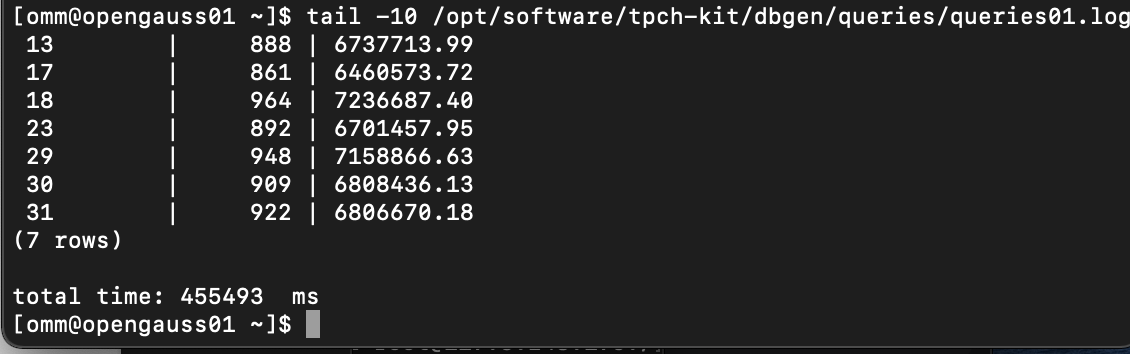
全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新。我理解的意思就是，在用户更新表中记录后，对整个视图刷新。增量物化视图顾名思义就是可以对物化视图增量刷新，需要用户手动执行语句完成仅对物化视图在一段时间内的增量数据进行刷新。与全量创建物化视图不同在于目前增量物化视图所支持场景较小。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

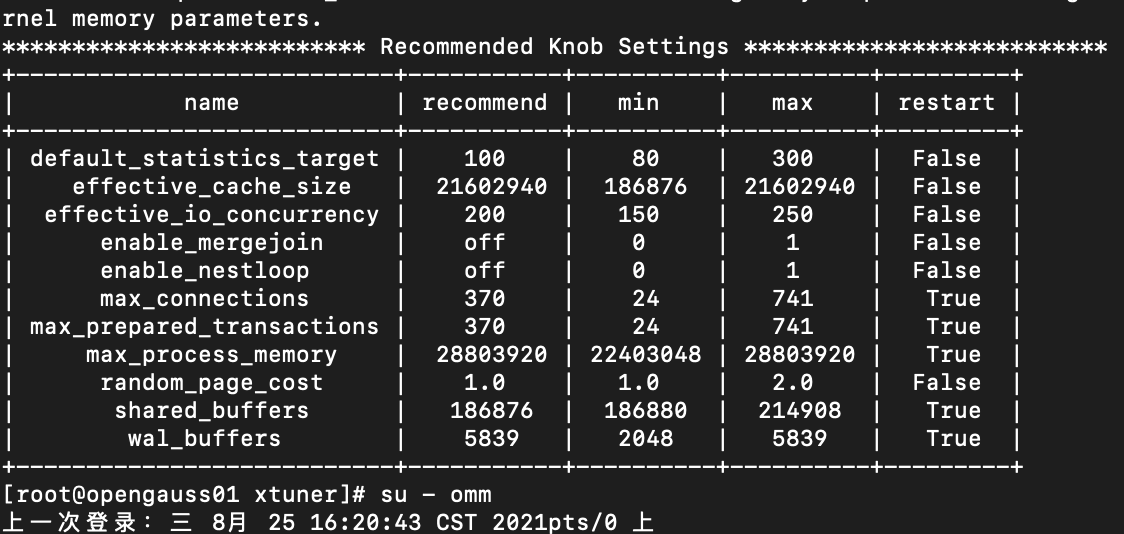
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

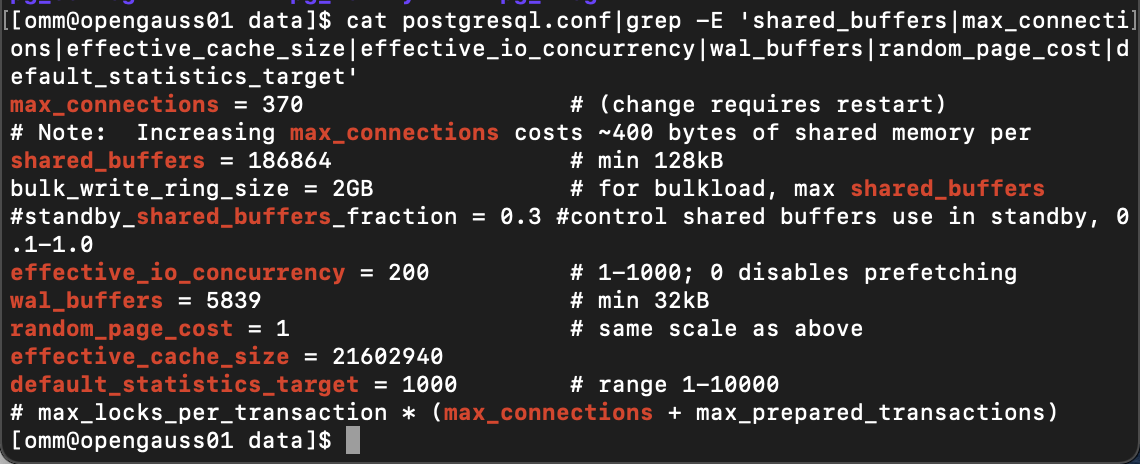
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

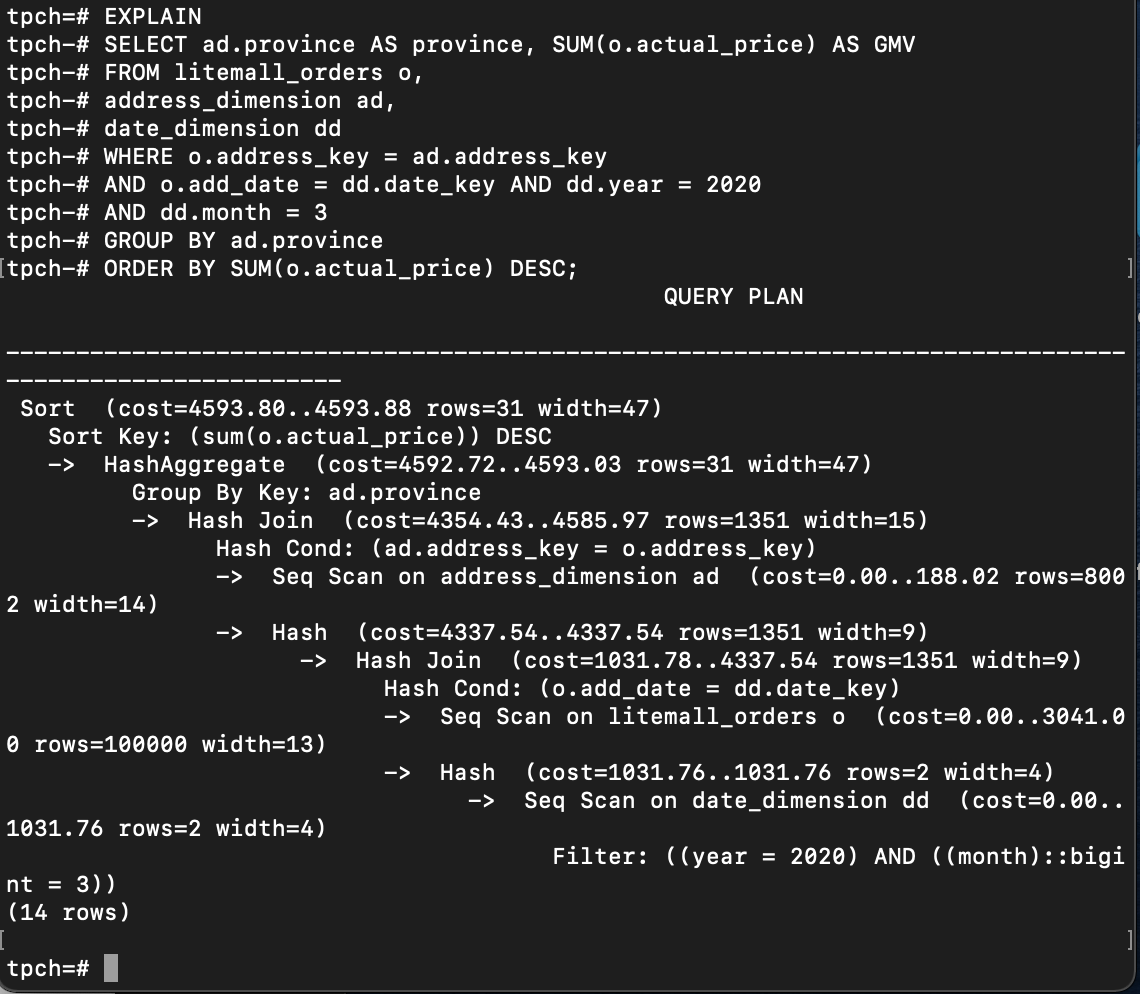
AND o.add\_date = dd.date\_key

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

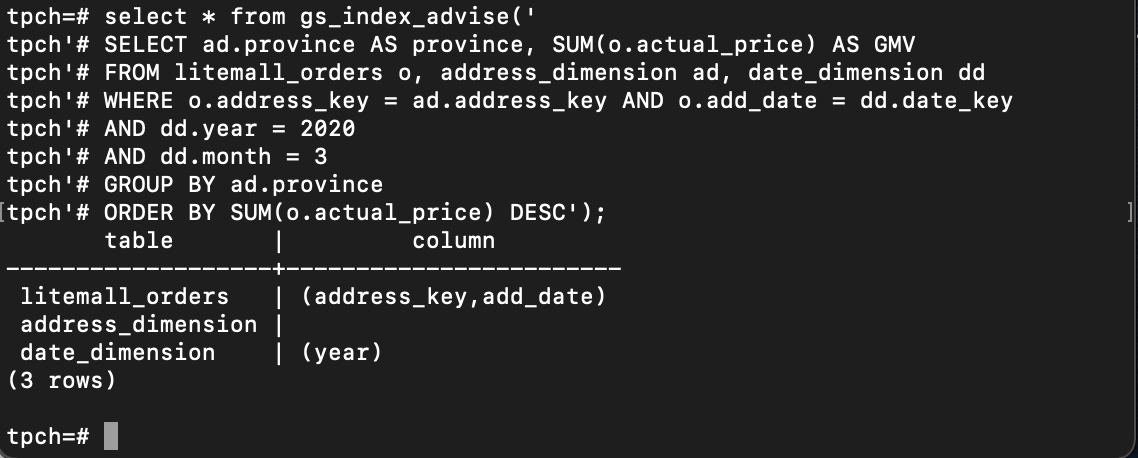
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

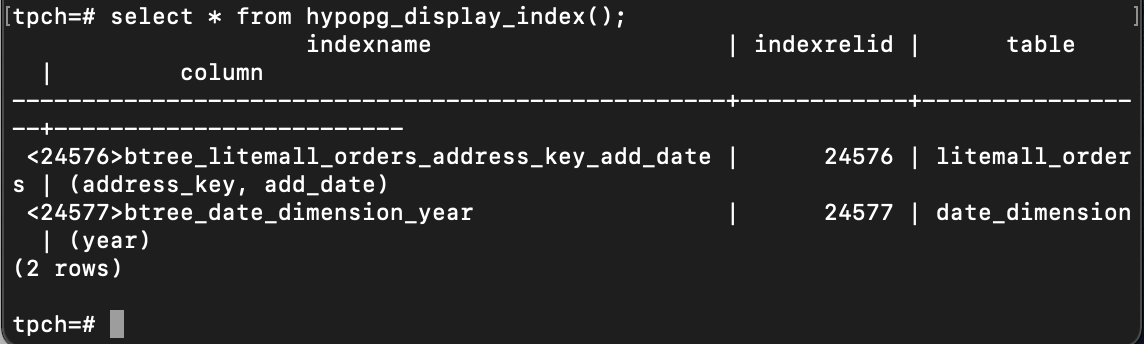
GROUP BY ad.province

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。

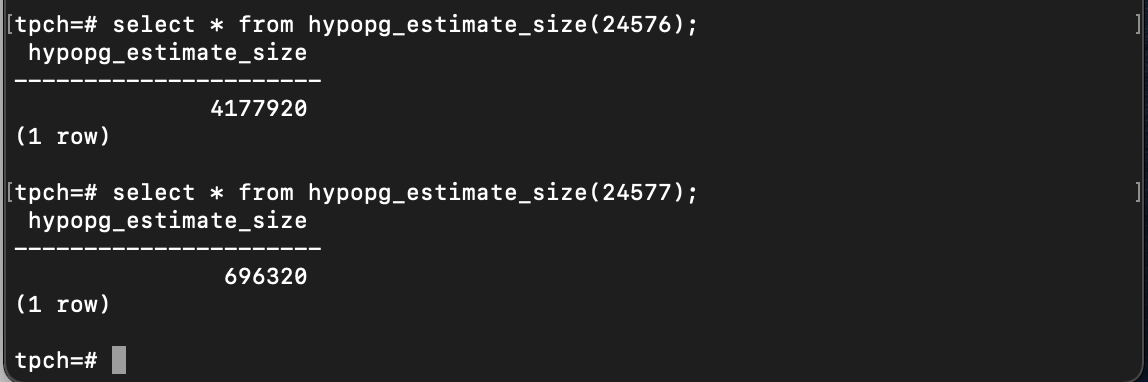
select \* from hypopg\_display\_index();



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

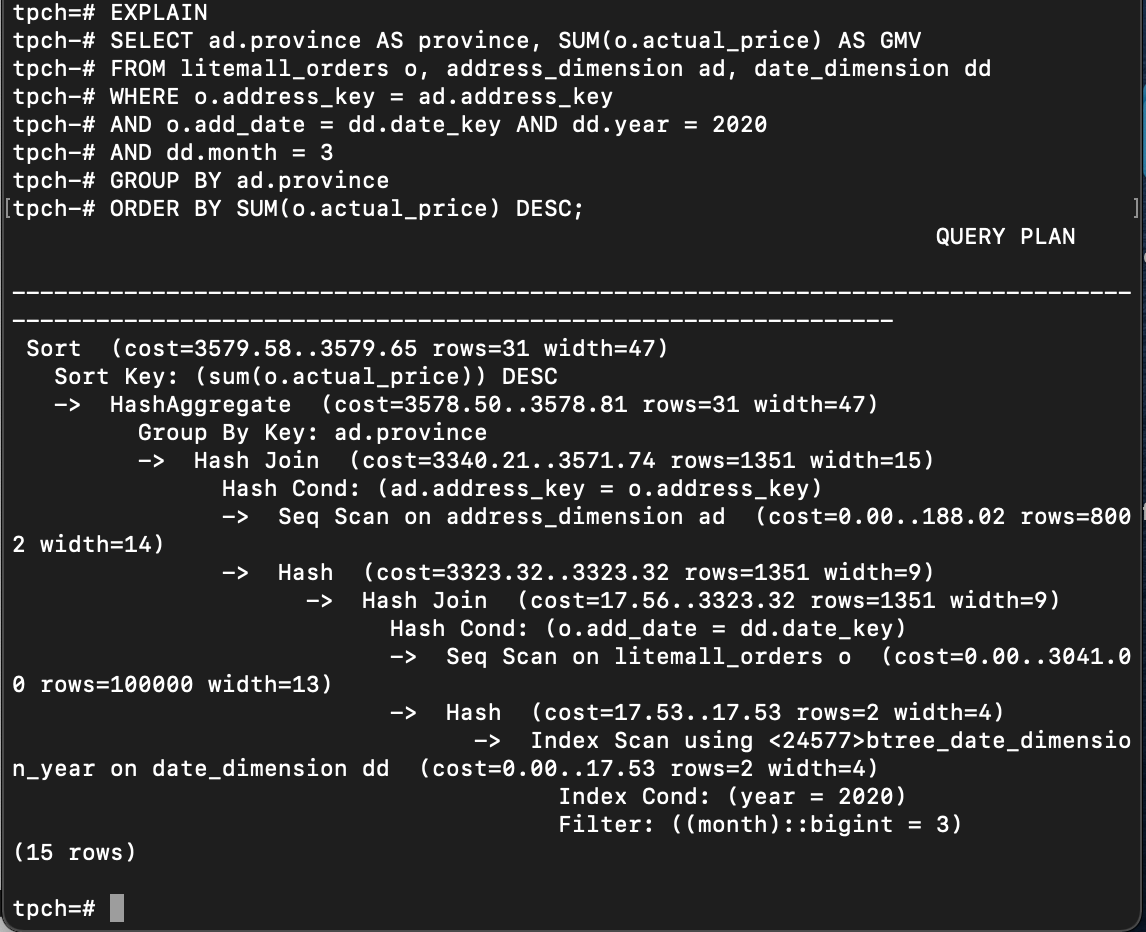
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

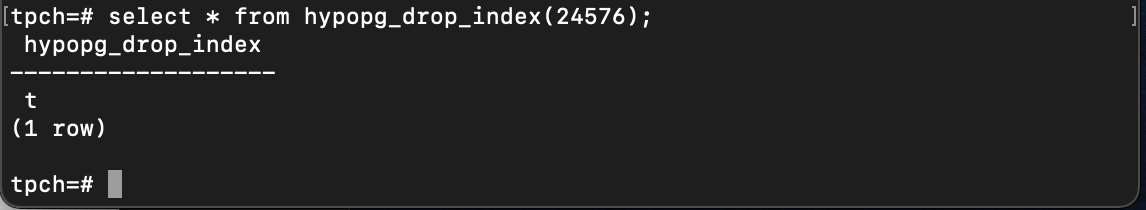
GROUP BY ad.province

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



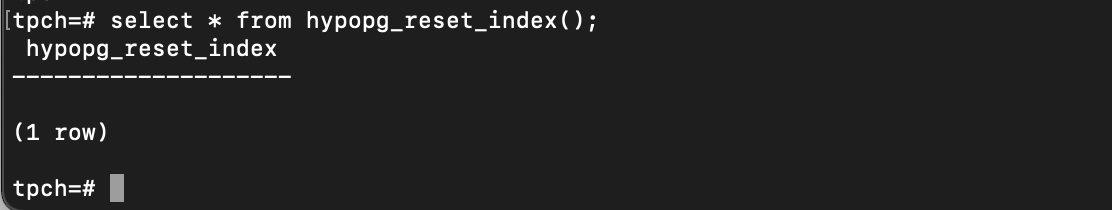
6. 删除某一个索引虚拟列，将执行结果截图。

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



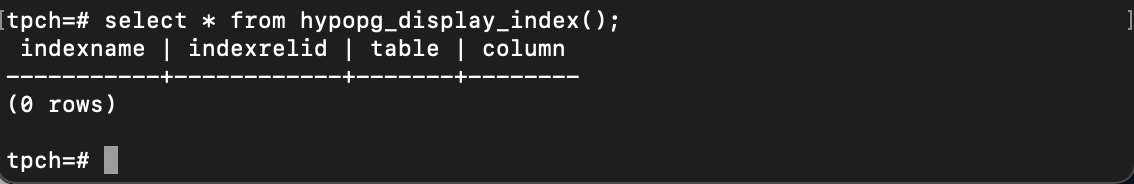
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

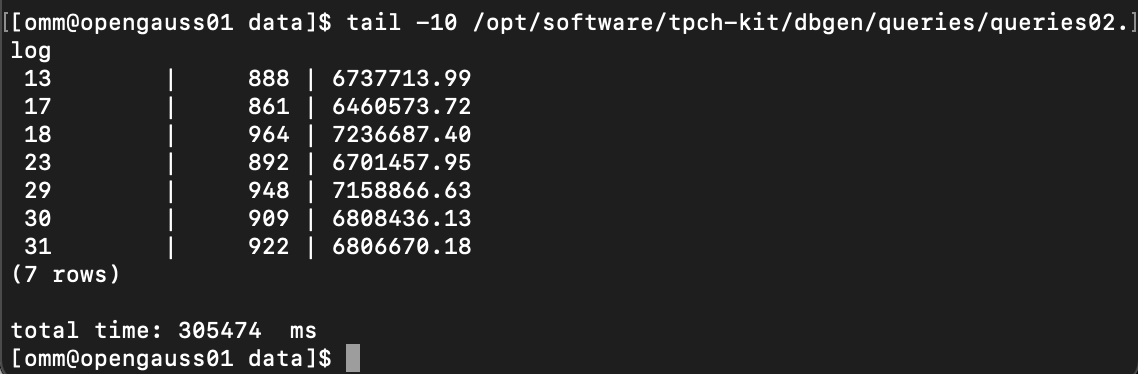
select \* from hypopg\_display\_index();



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

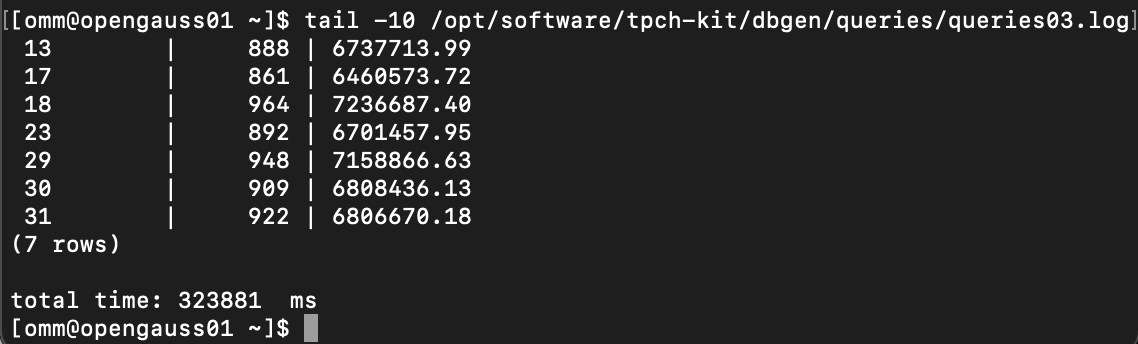


挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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

考察原queries.sql文件，发现其中有大量重复的对lineitem的数据求平均值的操作，可以在开始查询前先创建视图将重复的操作完成，后直接查询即可，理论上可以提高速度。



但是得到的结果与预期相反，很奇怪。初步考虑可能是创建视图的时候sql语句有些地方。由于时间限制，来不及再做更改。

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 = 1000。因为这是openGauss的ai4db特性给出的结果，能提升数据库性能。

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？

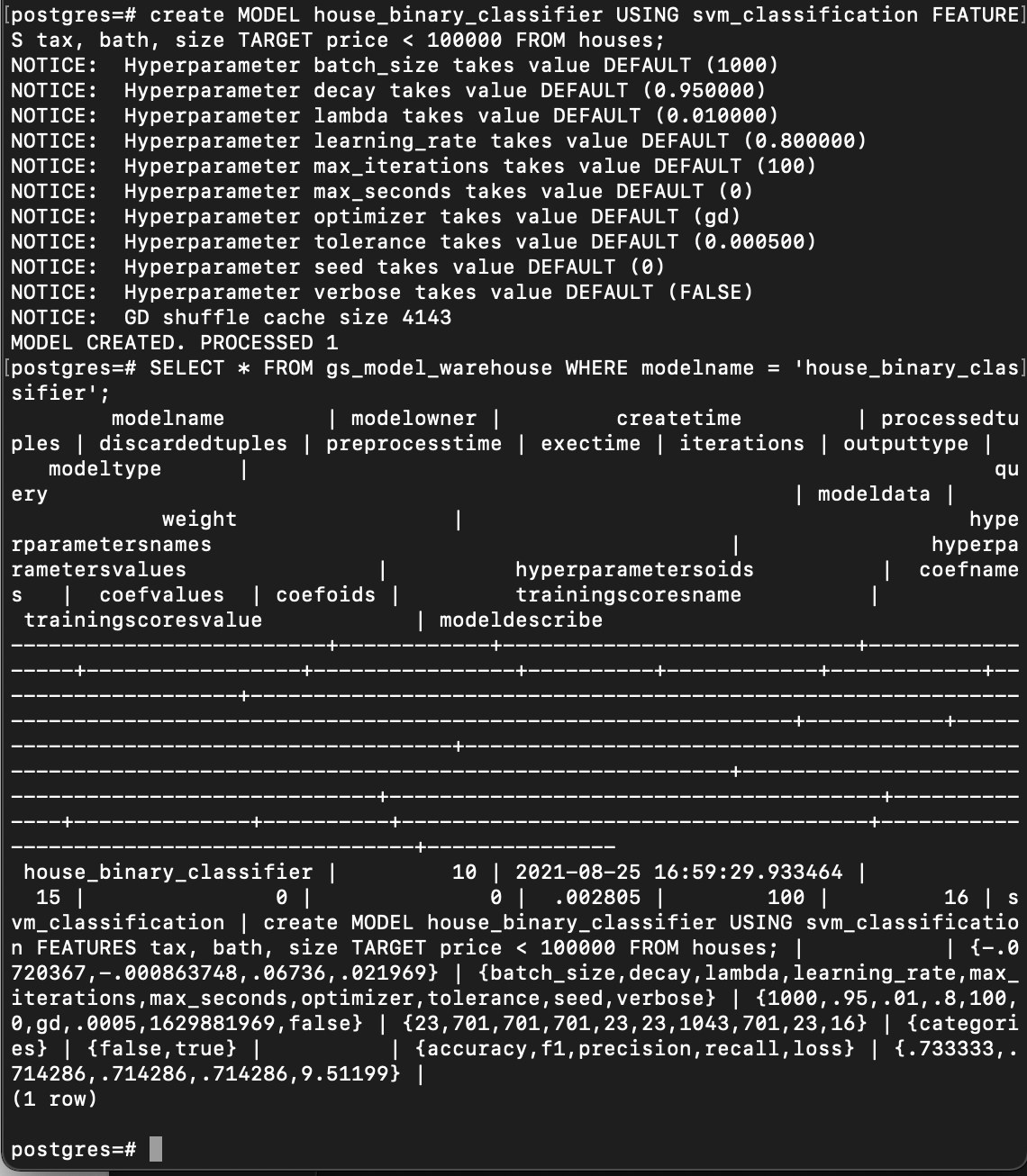
索引是一种特殊的数据库结构，由数据表中的一列或多列组合而成。索引根据表中的一列或若干列按照一定顺序建立的列值与记录行之间的对应关系表，实质上是一张描述索引列的列值与原表中记录行之间一 一对应关系的有序表。可以把索引比作新华字典的音序表。例如，要查“库”字，如果不使用音序，就需要从字典的 400 页中逐页来找。但是，如果提取拼音出来，构成音序表，就只需要从 10 多页的音序表中直接查找。这样就可以大大节省时间。

使用视图，合理设计scheme

# 关卡四、openGauss的DB4AI特性应用

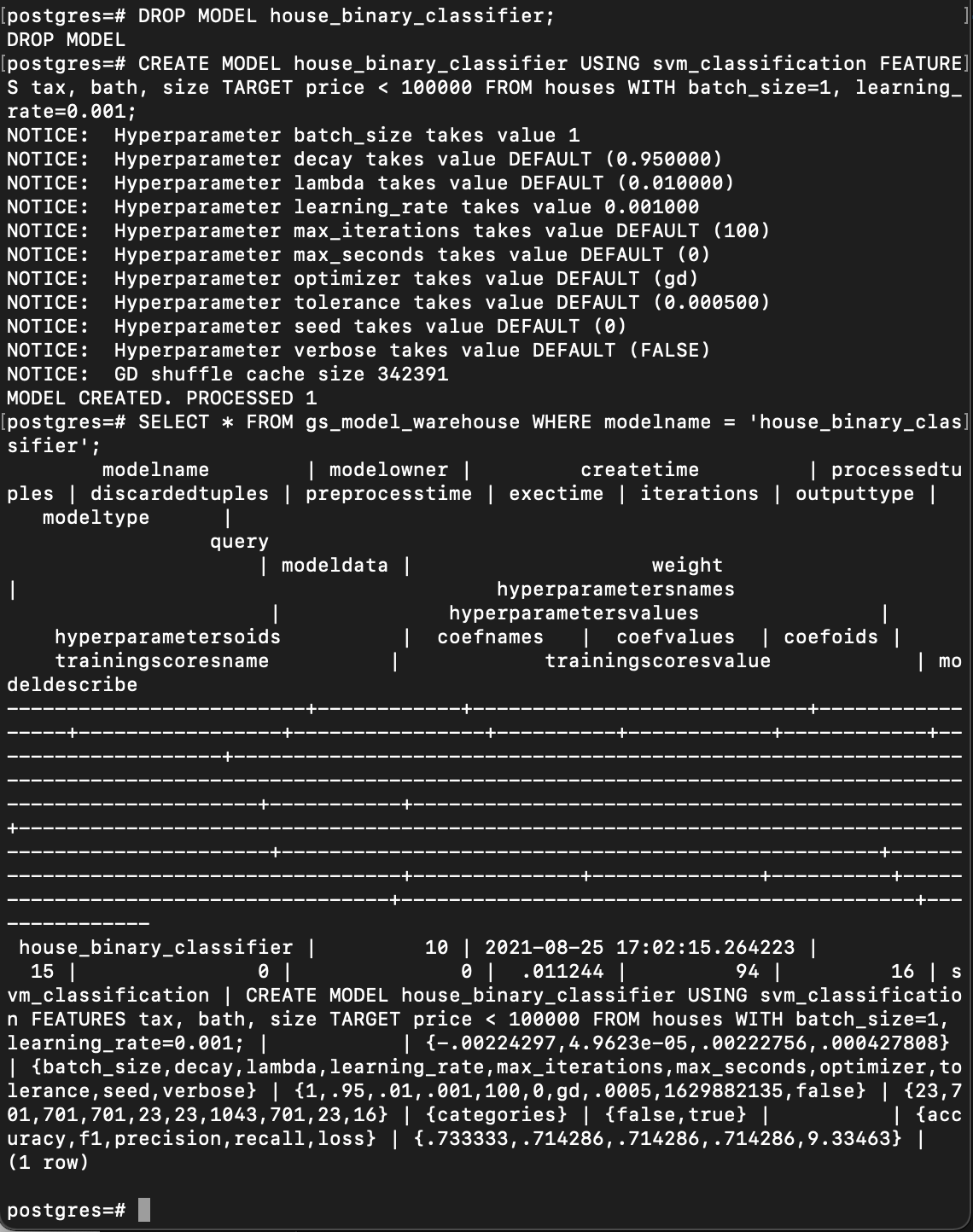
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



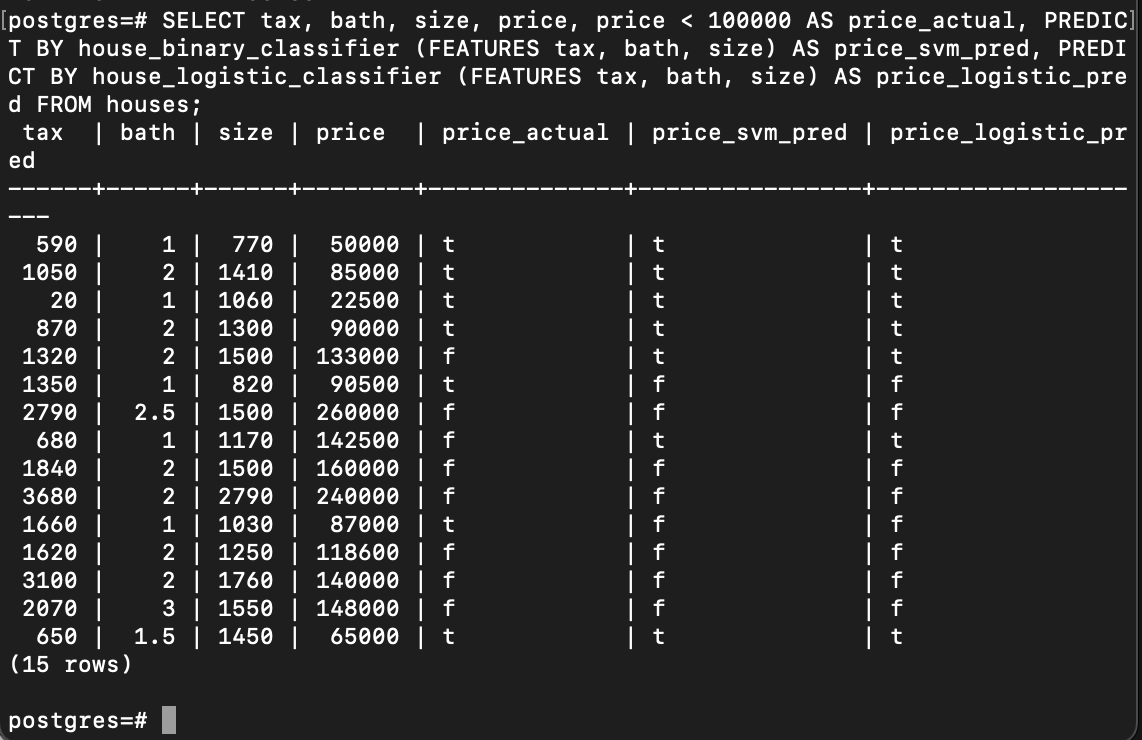
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型预测离散值，回归模型预测连续值

实践思考题2：什么是SVM算法？

SVM(支持向量机)是一种二分类模型，是在分类与回归分析中应用的监督式机器学习算法，它的基本模型是定义在特征空间上的间隔最大的线性分类器。

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：准确率是指分类正确的样本占总样本个数的比例。准确率是针对所有样本的统计量。

精确率：精确率是指分类正确的正样本个数占分类器判定为正样本的样本个数的比例。精确率是对部分样本的统计量，侧重对分类器判定为正类的数据的统计。

召回率：召回率是指分类正确的正样本个数占真正的正样本个数的比例。召回率也是对部分样本的统计量，侧重对真实的正类样本的统计。

F1 score：F1 score是精准率和召回率的调和平均值。是统计学中用来衡量二分类模型精确度的一种指标。 它同时兼顾了分类模型的精确率和召回率

ROC曲线：是一种坐标图式的分析工具，用于选择最佳的分类模型、舍弃次佳的模型。可以通过ROC曲线表达出的AUC面积来判断模型好坏

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？

均方误差：观测值与真值偏差的平方和与观测次数的比值。均方误差的值越小，说明预测模型描述实验数据具有更好的精确度。

均方根误差：均方根误差是均方误差的算术平方根。用来衡量观测值同真值之间的偏差

平均绝对误差：平均绝对误差是绝对误差的平均值。平均绝对误差能更好地反映预测值误差的实际情况。

决定系数：决定系数在统计学中用于度量因变量的变异中可由自变量解释部分所占的比例，以此来判断回归模型的解释力