openGauss AI特性创新实践课



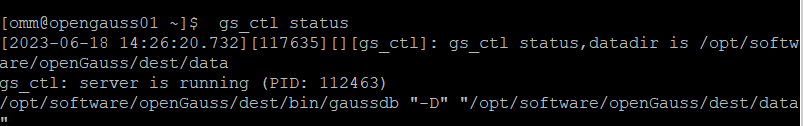
华为技术有限公司

# 关卡一、openGauss数据安装及基本操作

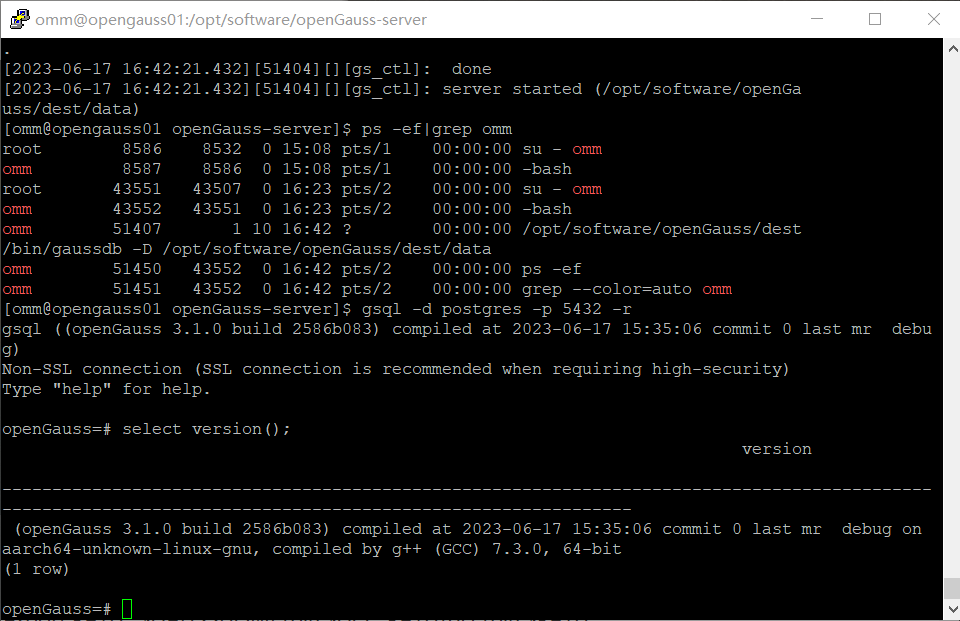
openGauss数据安装及基本操作, 作业提交任务如下：

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？

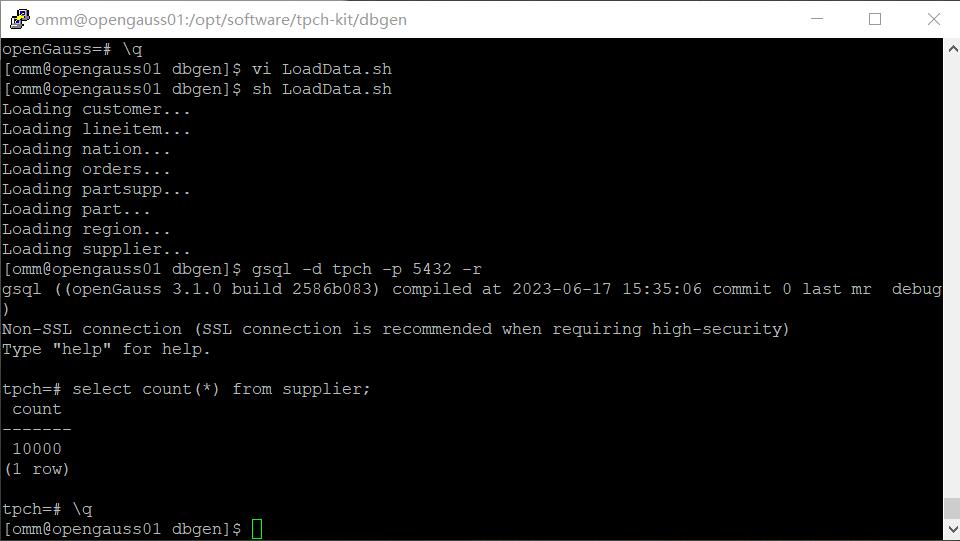
源码安装的优点，编译安装过程，可以设定参数，按照需求，进行安装，并且安装的版本，可以自己选择，灵活性比较大。

# 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：

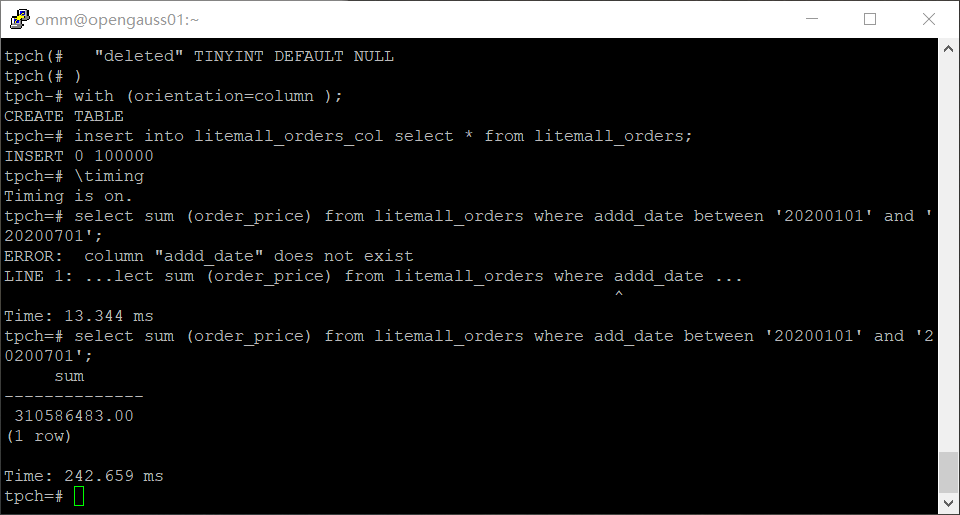
select count(\*) from supplier;;



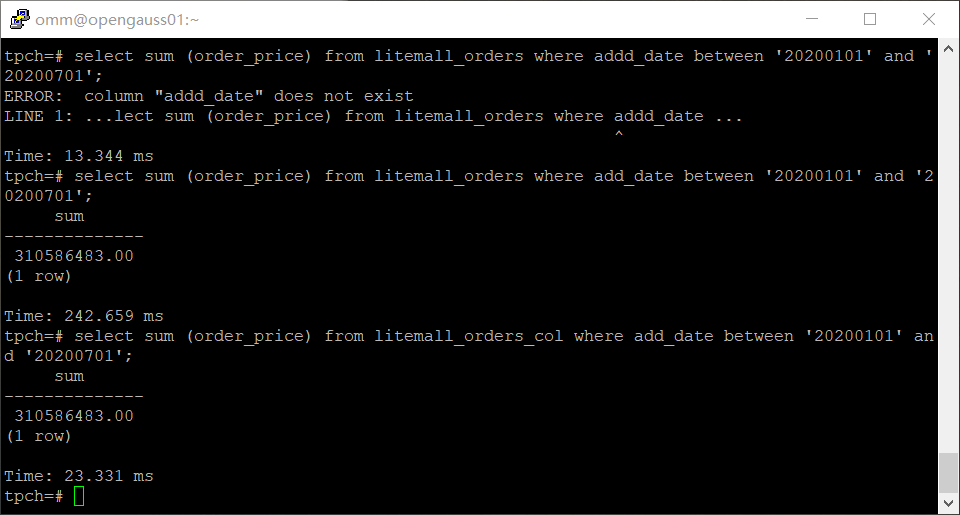
任务二：行存表与列存表执行效率对比

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';

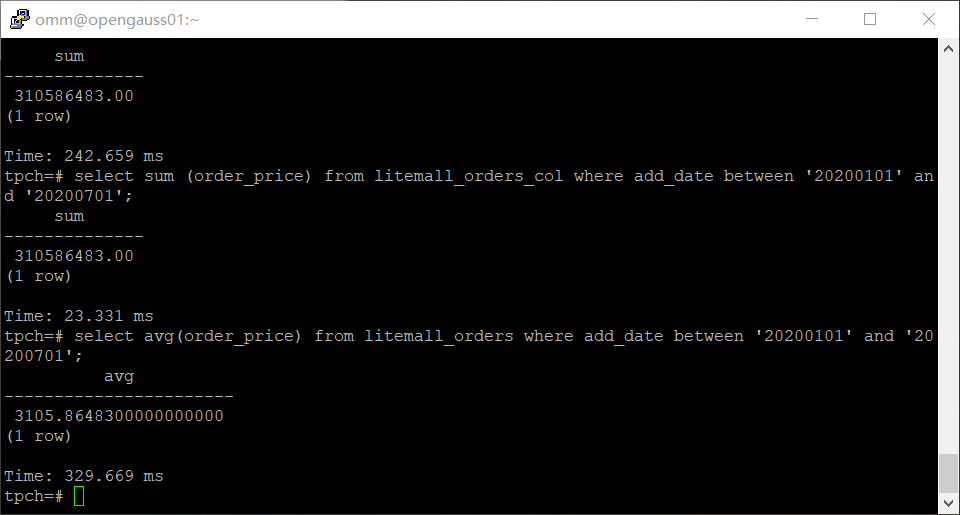


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

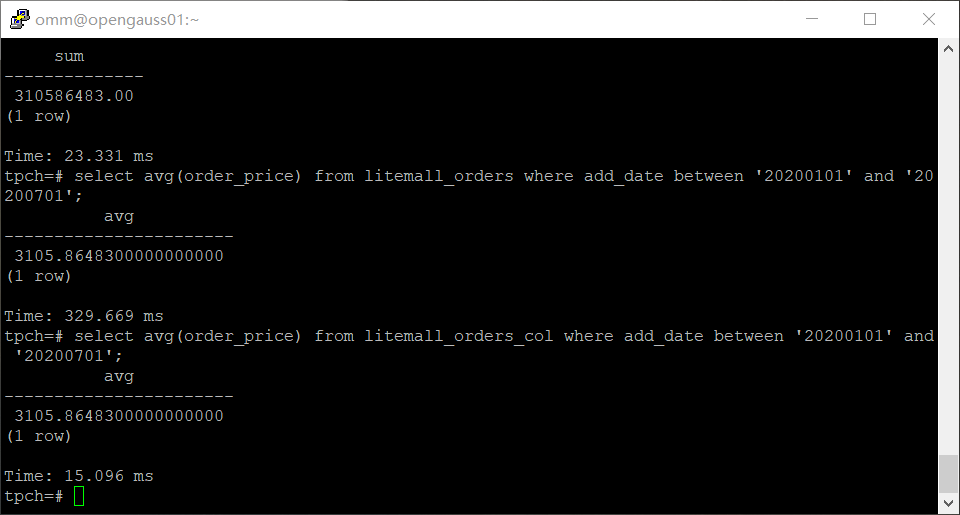


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

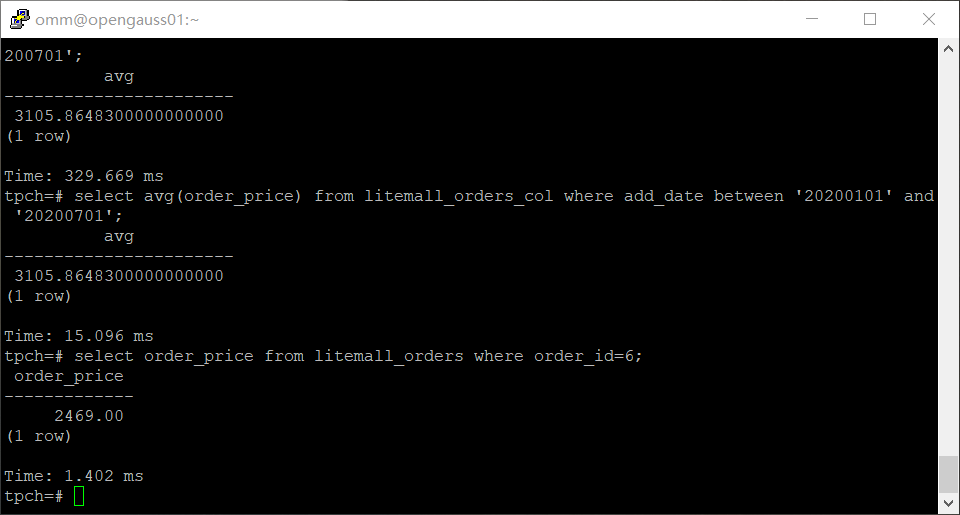


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

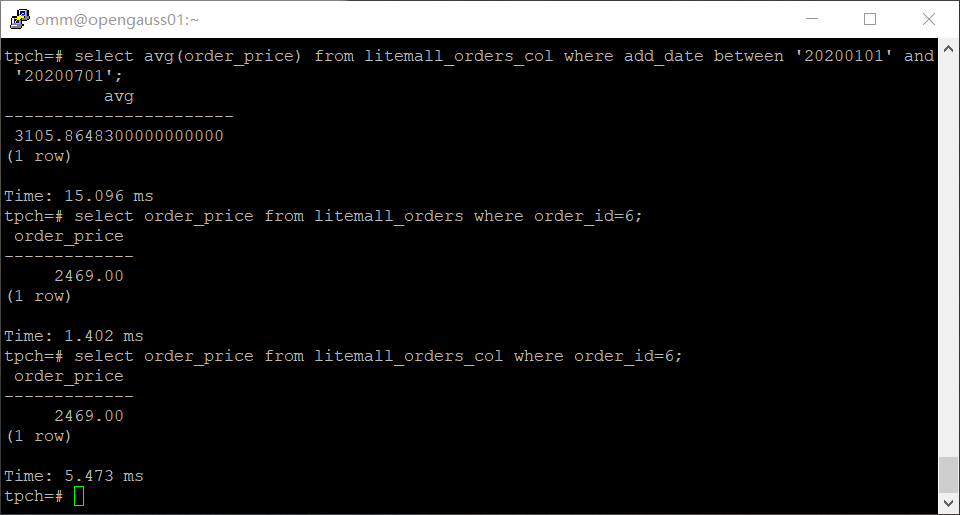


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

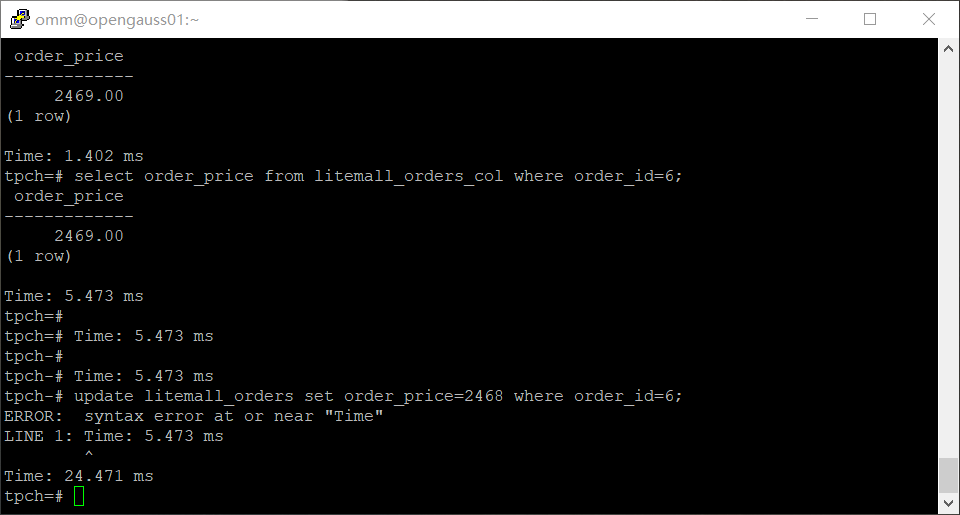


select order\_price from litemall\_orders\_col where order\_id=6;

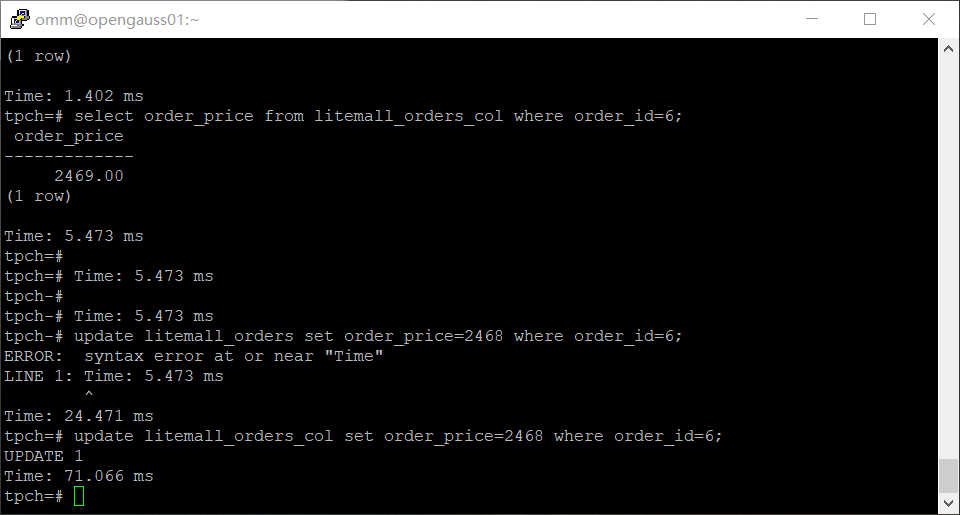


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



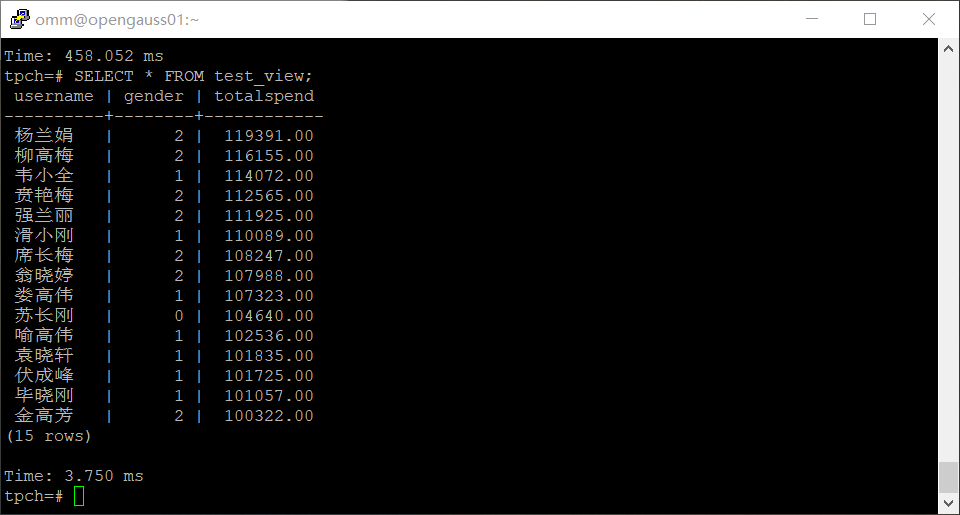
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

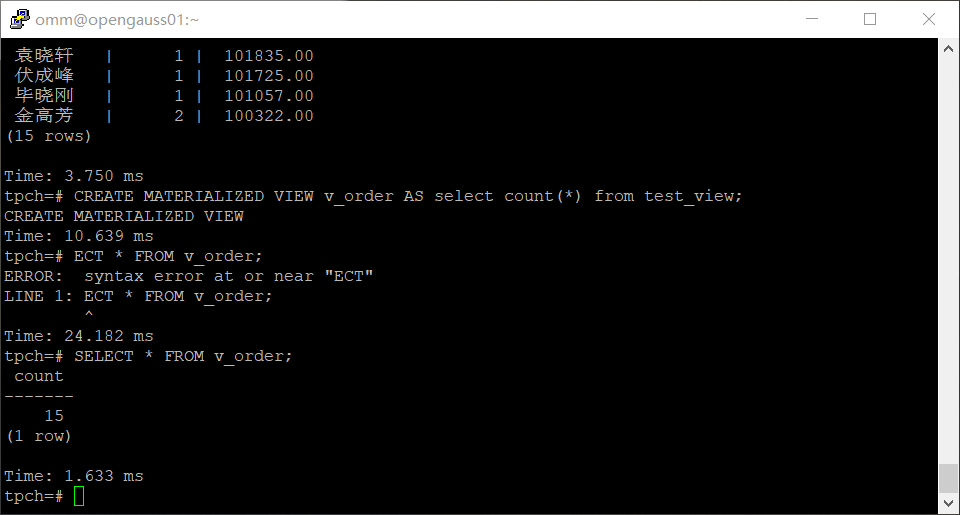
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



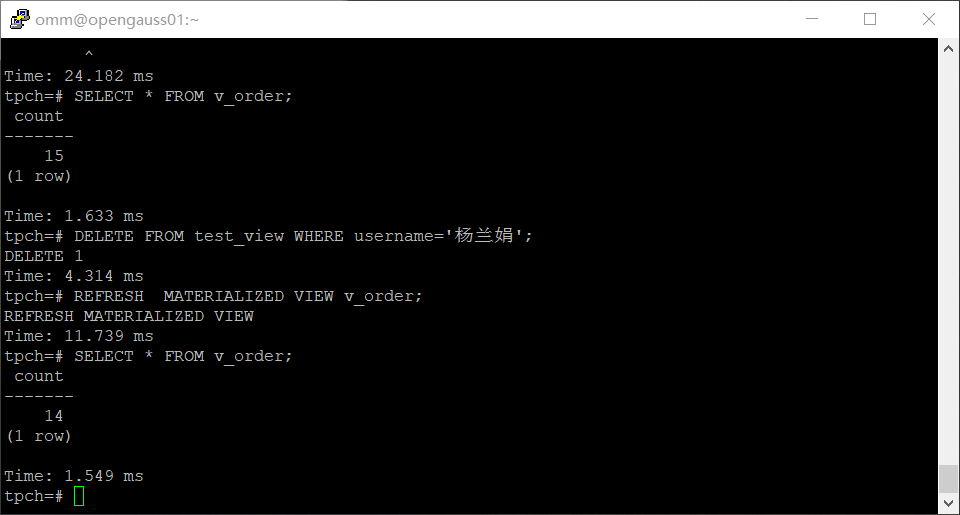
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



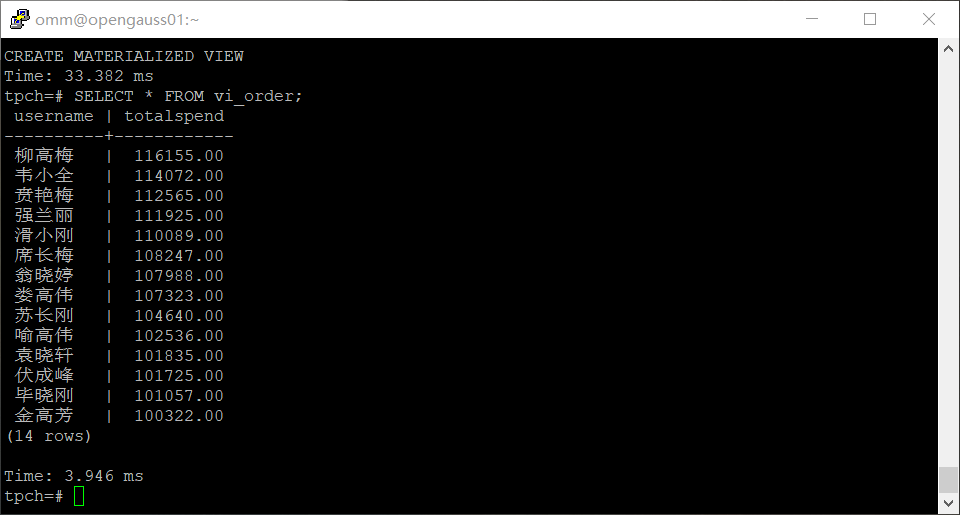
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



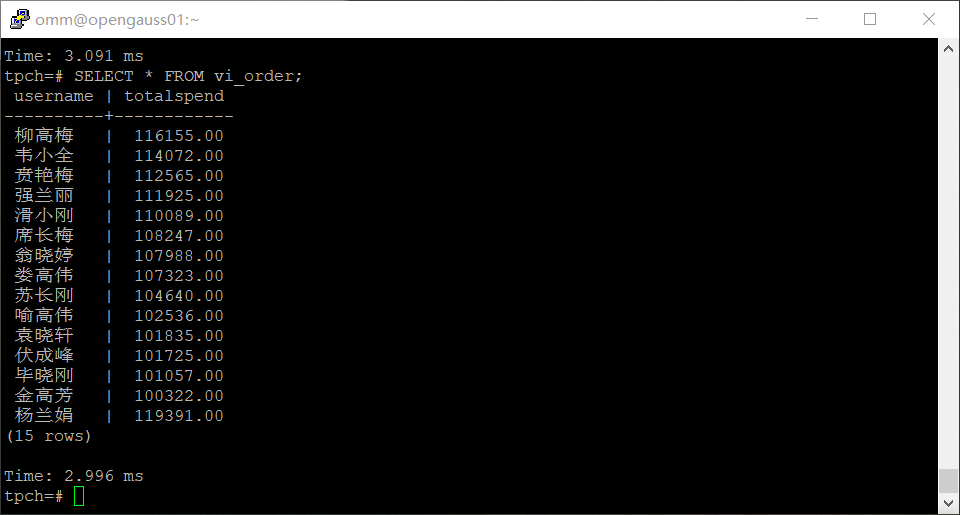
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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;



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？

行存表与列存表在执行相同的SQL语句时,执行时间不同的主要原因是:- 它们的数据存储结构不同。行存表将同一行的数据存储在一起,而列存表将同一列的数据存储在一起。- 它们适合不同类型的查询。行存表更适合进行全行扫描和行级别的查询,而列存表更适合聚合函数和列级别的查询。当执行以下类型的SQL时,行存表效率更高:- 全表扫描,需要读取整个表的数据行  
- 根据行键(主键)进行查找,只需要找到某一行  
- 需要访问表中大部分列的数据时这是因为行存表将同一行的数据存储在一起,所以读取整行数据更加高效。当执行以下类型的SQL时,列存表效率更高:- 包含GROUP BY、ORDER BY、DISTINCT等列级别运算的查询  
- 只需要访问表中部分列的数据时  
- 包含聚合函数如COUNT()、MAX()、AVG()等这是因为列存表将同一列的数据存储在一起,所以执行列级别的查询和聚合更加高效。总之,行存表和列存表由于数据存储结构的不同,在不同类型的查询下表现也不同:- 行存表在行级别的查询下性能更好,更适合全表扫描等  
- 列存表在列级别的查询和聚合运算下性能更好理解它们的区别和各自的优点,可以更好地在数据库设计和SQL优化中选择合适的表。许多数据库也提供了行列混合的表,既有行存又有列存的特性,这可以在一定程度上兼顾两者的优点,应对更广范围的查询场景。但这也增加了设计和优化的难度。所以,深入理解各种表结构的特点和应用场景是成为一名高级数据库开发人员的必要技能。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量物化视图和增量物化视图的主要差别有:

1. 数据范围:全量物化视图包含所有数据,增量物化视图只包含新增或更新的数据。全量物化视图的数据范围更广,数据量更大。

2. 数据更新频率:全量物化视图的更新频率低,一般每天或每周更新一次。增量物化视图的更新频率高,可以每分钟更新一次。

3. 查询延迟:由于数据范围和更新频率的差异,全量物化视图的查询延迟通常高于增量物化视图。全量物化视图的查询需要扫描更多数据,而增量物化视图只需要扫描新增的数据。

4. 空间消耗:全量物化视图占用的存储空间通常远大于增量物化视图,因为全量物化视图包含全部历史数据。

5. 构建难度:全量物化视图的构建难度和复杂度高于增量物化视图。全量物化视图需要处理全部历史数据,而增量物化视图只需要处理新产生的数据。

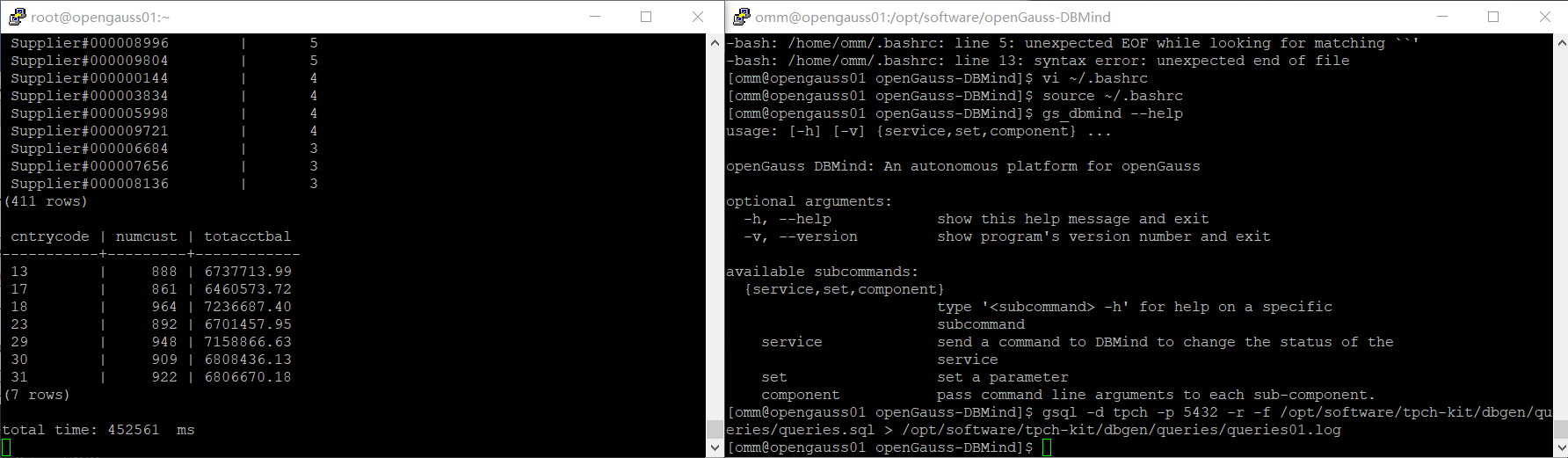
总之,全量物化视图和增量物化视图各有优缺点。实际应用中,可以根据业务需求选择合适的物化视图,或者将两者结合使用。全量物化视图可以提供历史数据支持,增量物化视图可以提供低延迟的实时数据查询。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

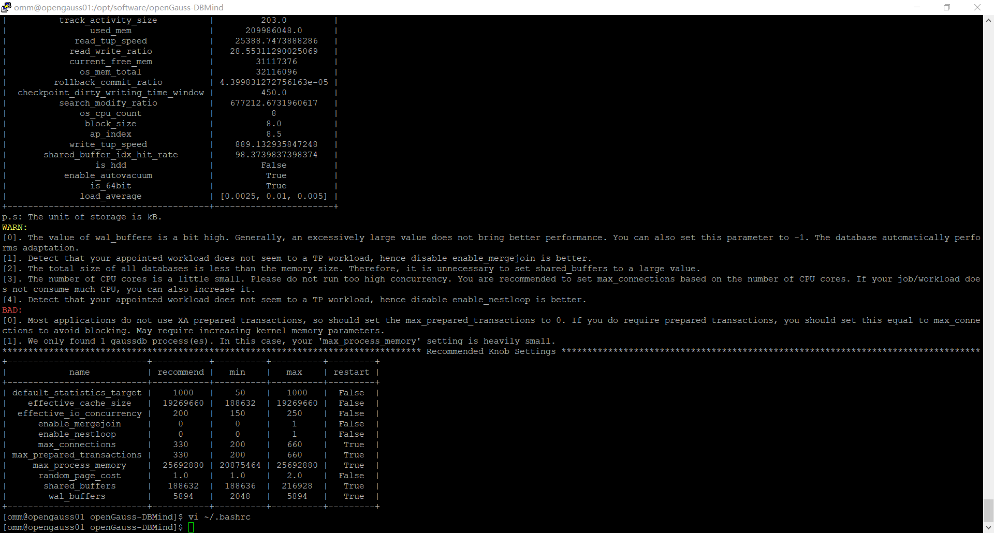
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

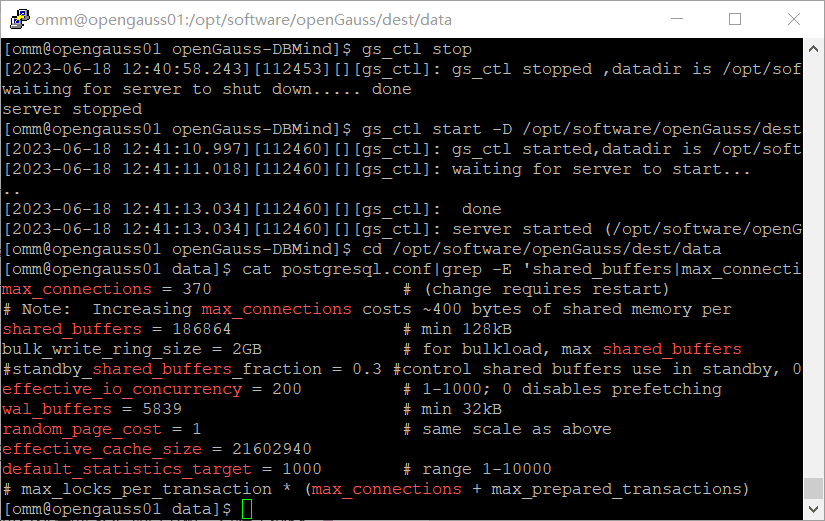
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

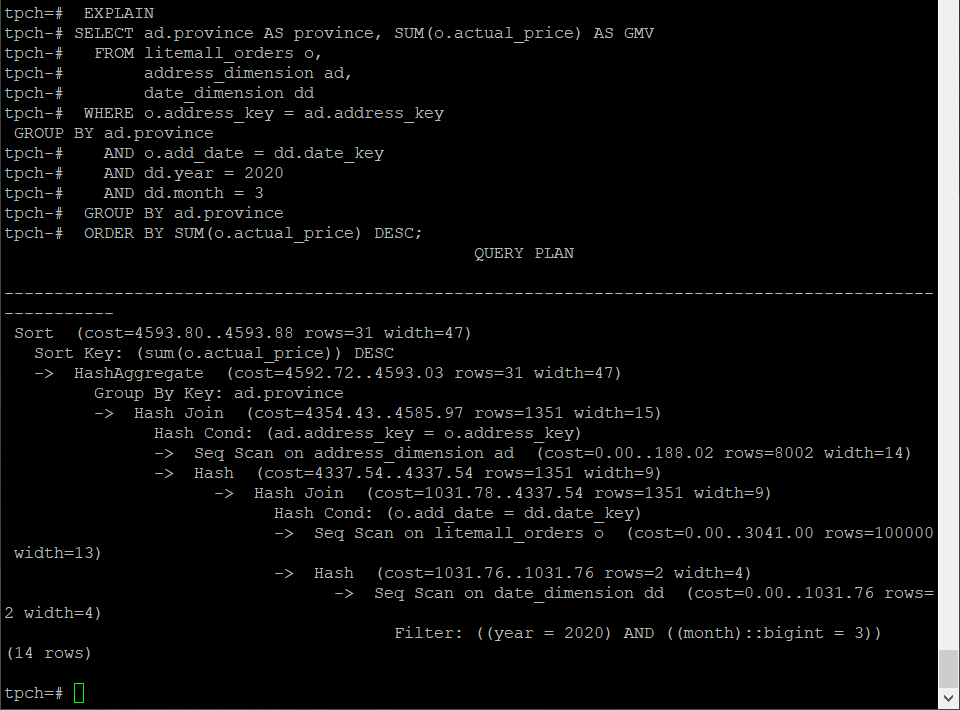
AND o.add\_date = dd.date\_key

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

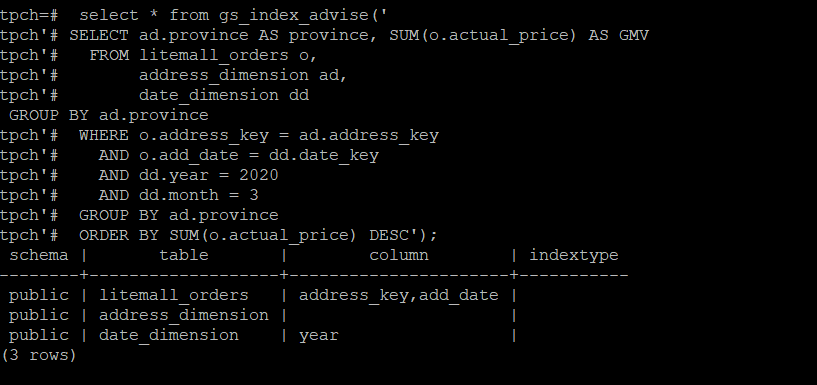
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

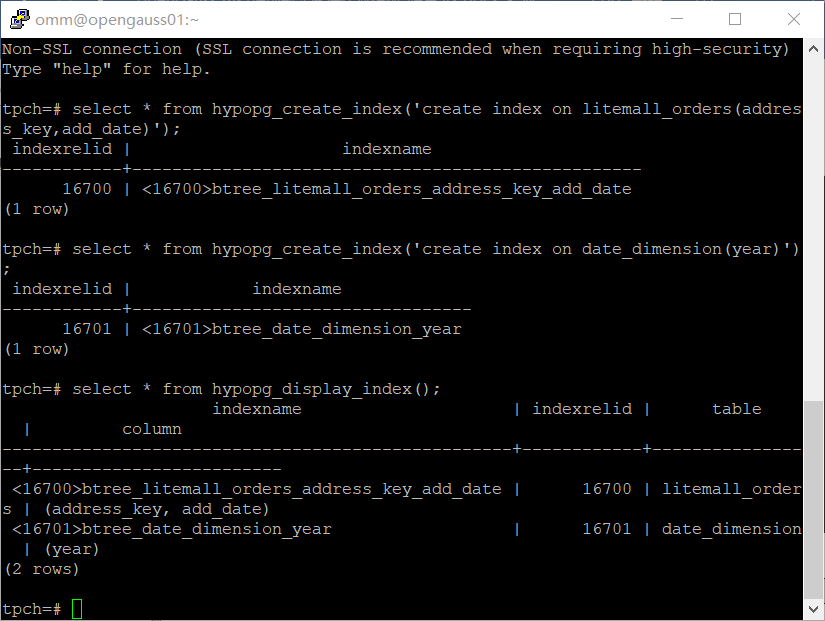
GROUP BY ad.province

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。

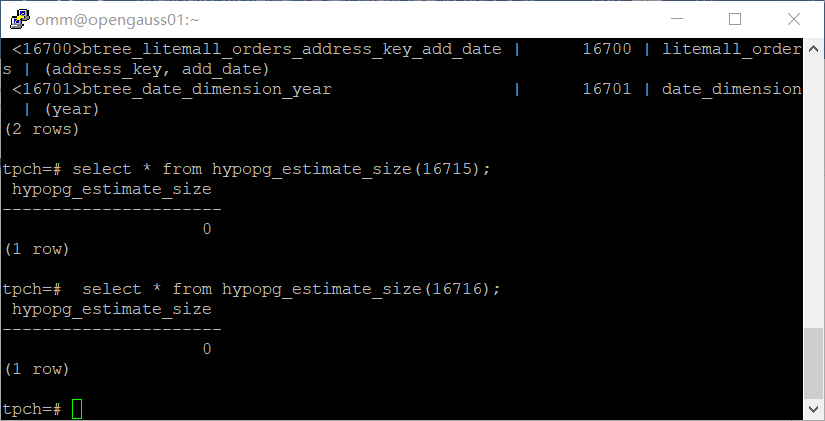
select \* from hypopg\_display\_index();



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

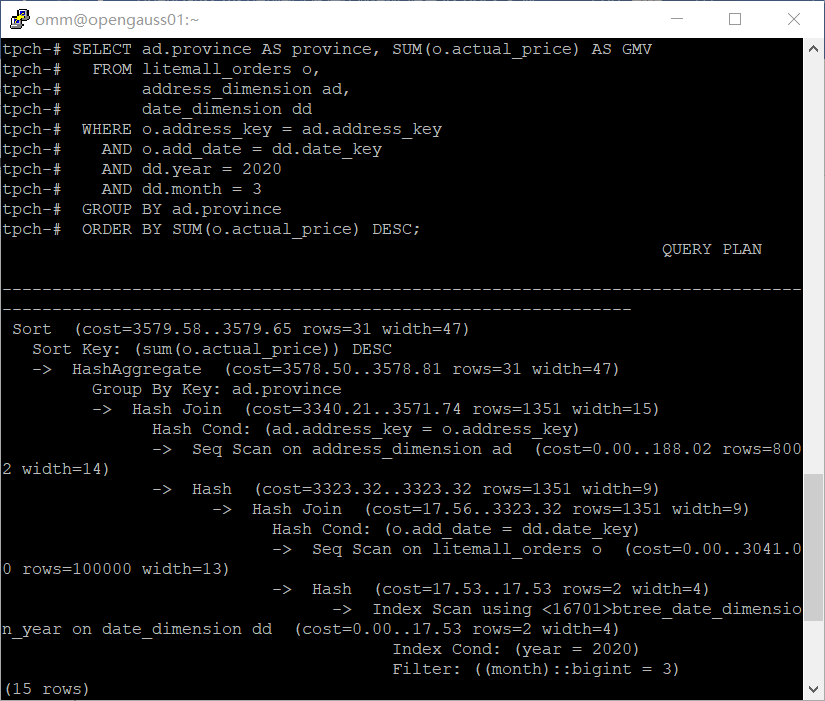
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

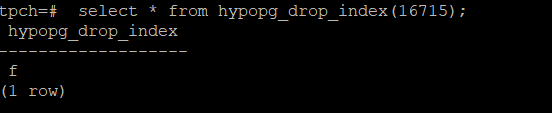
GROUP BY ad.province

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



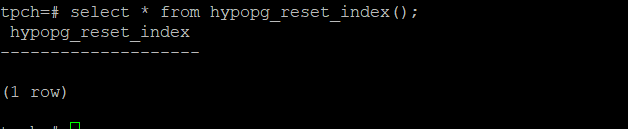
6. 删除某一个索引虚拟列，将执行结果截图。

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



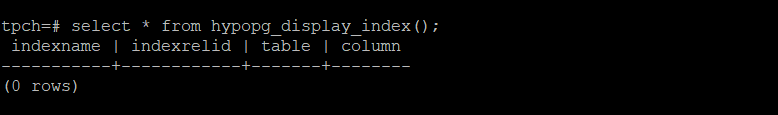
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

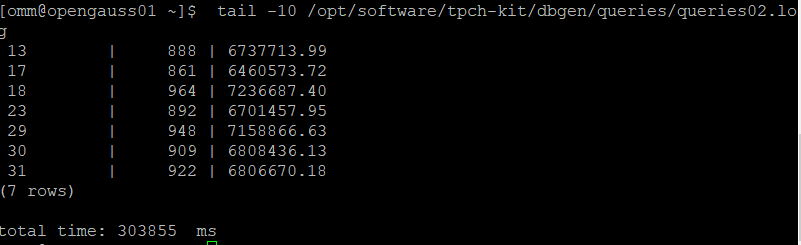
select \* from hypopg\_display\_index();



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

这里给出一些优化 queries.sql 中查询语句的建议 使用索引。如果查询中使用了条件过滤或排序的字段,且该字段无索引,应建立索引。索引可以大幅提高查询效率。 避免使用 select \* 。应只查询需要的字段,避免带出不必要的字段。使用表分区。如果表数据量很大,应考虑使用分区来提高查询效率。分区可以过滤掉不必要读取的分区数据。避免在 where 子句中使用 != 或 <> 。应使用更高效的 in 或 not in 。避免在 where 子句中使用 or 。or 条件导致引擎无法使用索引,应改写成 union 或子查询。在 group by 或 order by 子句中使用索引列。否则会引起文件排序或重排序,降低效率。 避免在 where 子句的后面使用 select 字段。应先使用 select 字段过滤,再应用 where 条件。 使用连接查询代替子查询。子查询要执行两次,效率较低。应改写成 inner join 或左连接查询。合理使用缓存。对频繁查询且不经常修改的数据,应设置合理的缓存策略,利用缓存能大幅提升查询效率。 考虑数据库的查询优化器参数。根据数据库不同,设置相关的参数,让查询优化器生成更高效的执行计划。 避免锁竞争。在高并发下,查询应避免锁表或行锁,影响其他事务或查询。

综上,通过索引、表分区、缓存、优化SQL语句结构等手段,可以显著提高查询的执行效率,达到倍数级的提升。但需要根据具体的查询场景和数据库,选择最适合的优化方案。

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default\_statistics\_target，effective\_cache\_sizeeffective\_io\_concurrencyenable\_mergejoin，enable\_nestloopmax\_connections，max\_Prepared\_transactions，max\_process\_memory，random\_page\_costshared\_bufferswal buffers参数进行了优化。

通过-Tuner给出的参数优化，可以获取最佳数据库参数配置。

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？

使用SQL索引的优点：

 1、通过唯一性索引可确保数据的唯一性

  2、加快数据的检索速度

  3、加快表之间的连接

  4、减少分组和排序时间

  5、使用优化隐藏器提高系统性能

数据库优化还可以：

  1、重建索引(dbcc dbreindex)

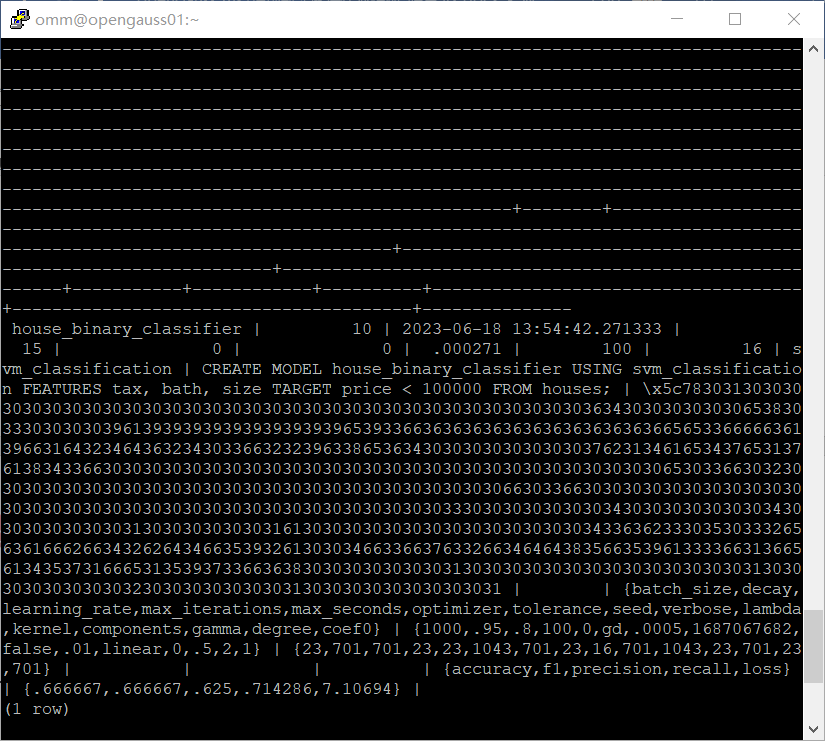
  2、索引优化向导

  3、整理指定的表或视图的聚集索引和辅助索引碎片

# 关卡四、openGauss的DB4AI特性应用

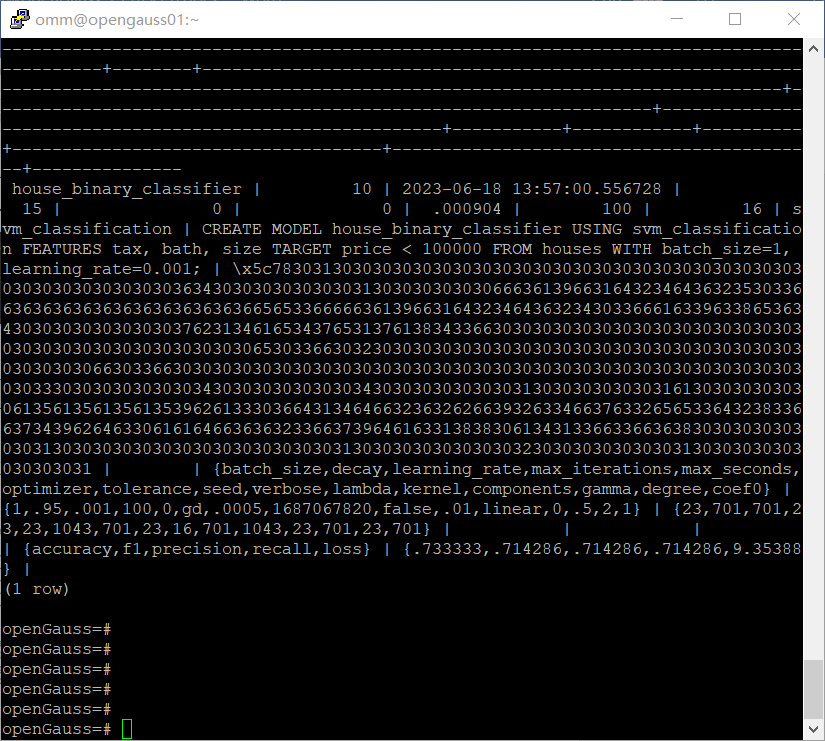
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



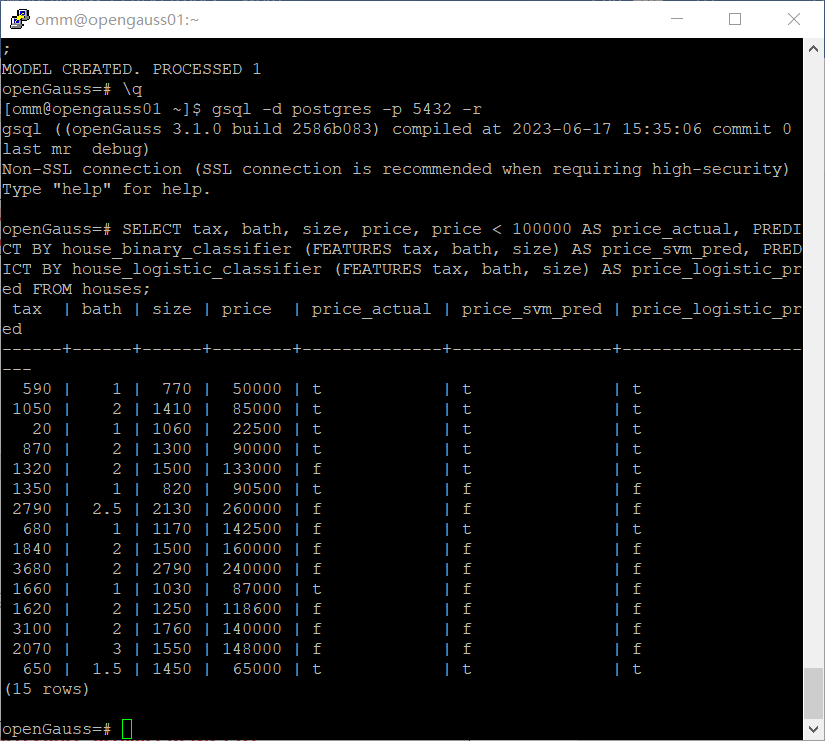
任务二：观察新模型的信息，将执行结果截图。

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？

分类与回归最主要的区别是输出变量的类型：连续变量的预测叫回归，离散变量的预测是分类。

回归的主要作用是了解两个或多个变量之间是否相关，相关的方向，相关的强度，并利用数学模型以便观察特定变量来预测研究者感兴趣的变量。分类模型是将回归模型输出离散化，分类和回归模型之间存在重要差异。

根本上来说分类是关于预测标签，而回归是关于预测数量。

实践思考题2：什么是SVM算法？

SVM算法是一种机器学习算法,属于监督学习方法中的分类算法。它的基本模型是定义在特征空间中的间隔最大的线性分类器。SVM算法的主要思想是:在特征空间中找寻一个最优的分离超平面,使不同类别的样本点在超平面两侧,且两侧间隔最大。最大间隔原则使分类决定边界最大程度地离群值的影响。使分类器具有较强的泛化能力。通过使用核技巧,SVM可以非线性地映射输入空间到特征空间,实现在输入空间中线性不可分的数据的线性可分。SVM还可以用于回归分析,称为SVR。

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.准确率：对于给定的测试数据集，分类器正确分类的样本数与总样本数之比。

2.精确率是检索出相关文档数与检索出的文档总数的比率（正确分类的正例个数占分类为正例的实例个数的比例），衡量的是检索系统的查准率。

3.召回率是指检索出的相关文档数和文档库中所有的相关文档数的比率（正确分类的正例个数占实际正例个数的比例），衡量的是检索系统的查全率

4. Precision和Recall指标有时候会出现矛盾的情况，这样就需要综合考虑他们，最常见的方法就是F-Measure。F-Score是Precision和Recall的加权调和平均。

5.TPR、FPR&TNR 真正类率，也称为sensitivity，刻画的是被分类器正确分类的正实例占所有正实例的比例。 负正类率，也称为1-specificity，计算的是被分类器错认为正类的负实例占所有负实例的比例。 真负类率，也称为specificity，刻画的是被分类器正确分类的负实例占所有负实例的比例。

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平均绝对误差 这个指标是对绝对误差损失的预期值

2平均绝对百分比误差这个指标是对相对误差损失的预期值.所谓相对误差,就是绝对误差和真值的百分比.

3均方误差该指标对应于平方(二次)误差的期望.

3均方误差根或均方根误差 该指标对应于平方(二次)误差的期望.

4 均方误差对数 该指标对应平方对数(二次)差的预期

5中位绝对误差 通过取目标和预测之间的所有绝对差值的中值来计算损失.