openGauss AI特性创新实践课



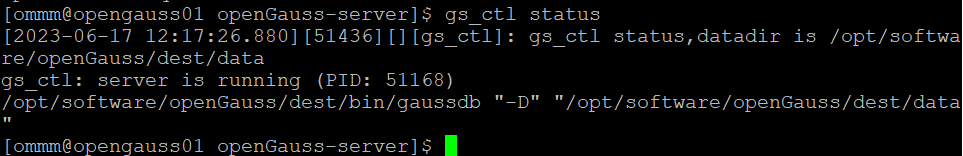
华为技术有限公司

# 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：

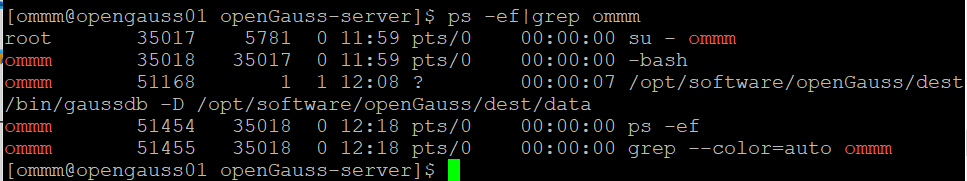
任务一：数据库状态验证

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？

通过源码变异安装数据库，可以实现定制化数据库打的安装选项，比如数据库的安装路径、编译选项、支持的插件等，从而满足不同的需求。

另外可以针对特定的硬件和操作系统平台进行优化，比如使用特定的编译器、优化选项等。

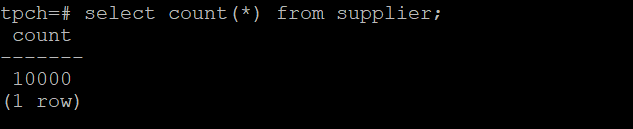
通过源码编译可以自定义编译选项，比如启用或禁用某些安全特性，从而提高数据库的安全性。

# 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：

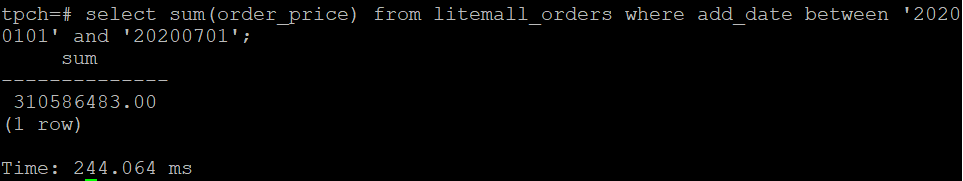
select count(\*) from supplier;;



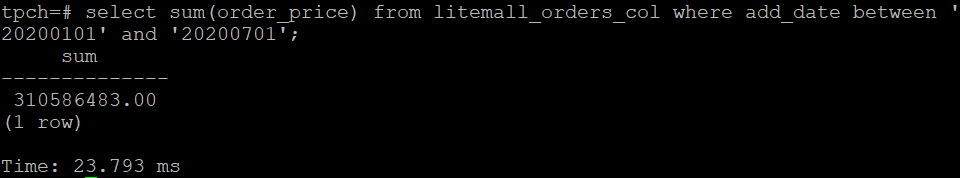
任务二：行存表与列存表执行效率对比

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';

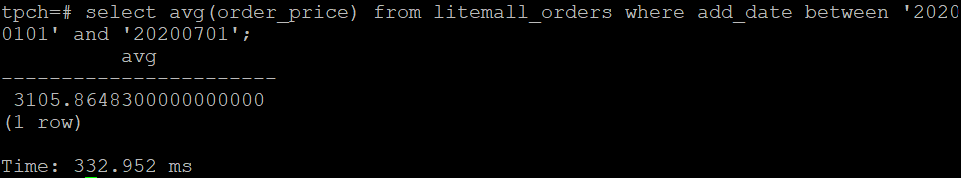


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

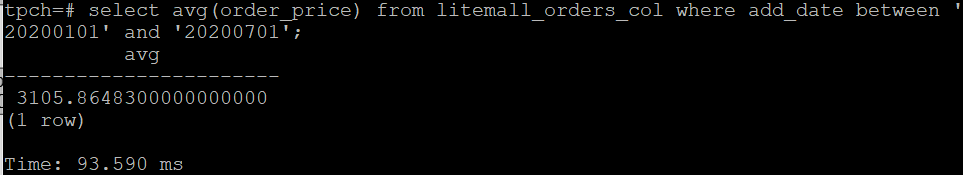


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

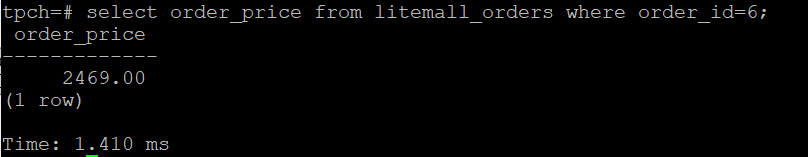


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

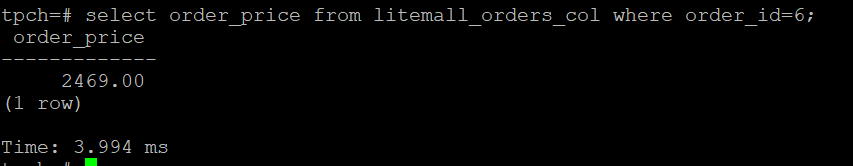


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

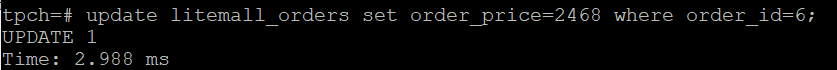


select order\_price from litemall\_orders\_col where order\_id=6;

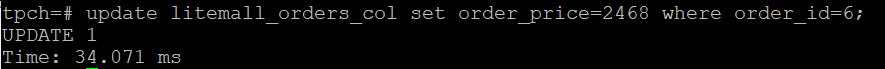


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



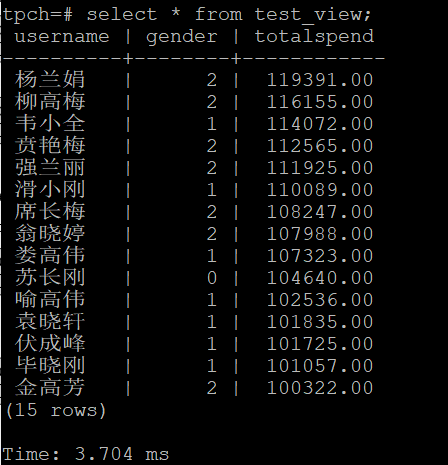
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

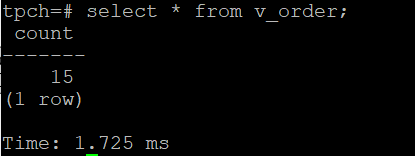
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



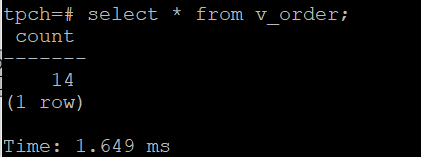
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

第一小问：1.数据存储方式方面，行存表按照行存储数据，而列存表按列存储数据。在查询时，列存表只需要加载所需的列数据，而无需加载其他列的数据。2.I/O开销方面，对于某些查询，行存表需要读取加载更多的数据页，其中包含查询中未使用的列。而列存表只需要加载查询所需的列数据，减少了磁盘I/O操作,从而降低了数据检索的开销。3.数据压缩方面，列存表通常使用更高效的压缩算法来存储列数据，同一列数据往往具有更高的数据相似性和重复性。数据压缩可以减少磁盘存储空间的占用，并且减少了磁盘I/O从而提高查询性能。

第二小问：行存表适用于以下方面，1.频繁进行更新或插入的表；2.存储小型表的情况；3.需要经常进行全表扫描的表。

第三小问：列存表适用于以下方面，1.需要进行大量查询和分析打的表；2.需要进行复杂聚合计算的表；3.需要进行大批量数据加载的表；4.需要进行大量数据压缩的表。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量物化视图是一种包含完整数据集的物化视图，当基础表中的数据发生变化时，需要重新计算整个物化视图，因此需要更多的计算资源和时间。全量物化视图适合于基础表数据变化较少的情况。

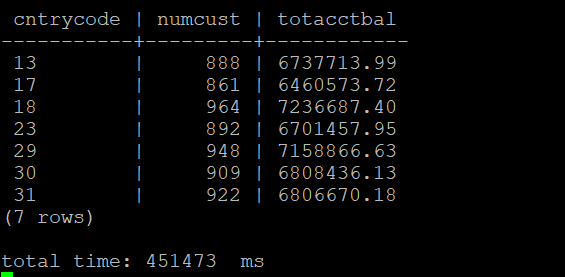
增量物化视图是一种只包含基础表中改变数据的物化视图，当基础表中的数据发生改变时，只需要计算改变部分，因此需要较少的计算资源和时间。增量物化视图适合于基础表数据变化较频繁的情况。

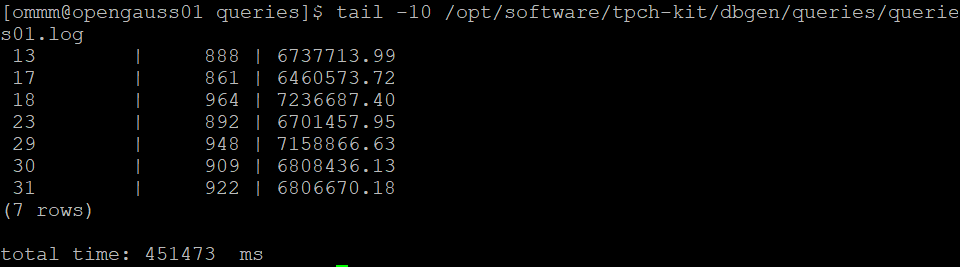
# 关卡三、openGauss的AI4DB特性应用

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：

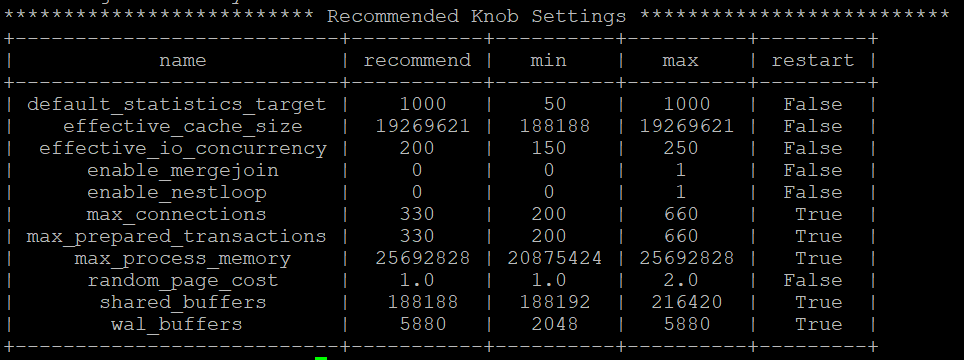
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log





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

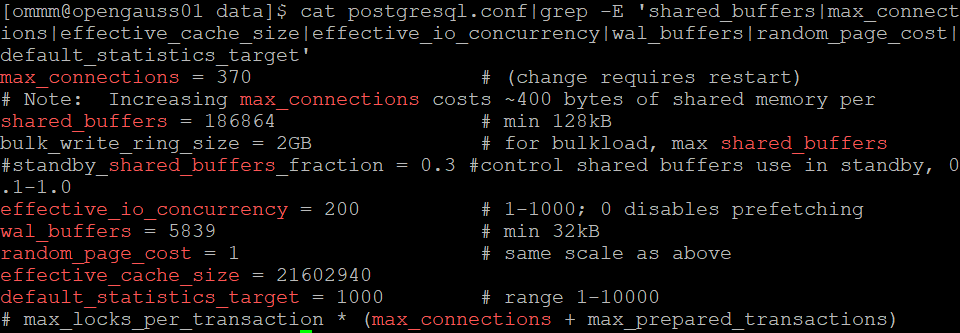
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

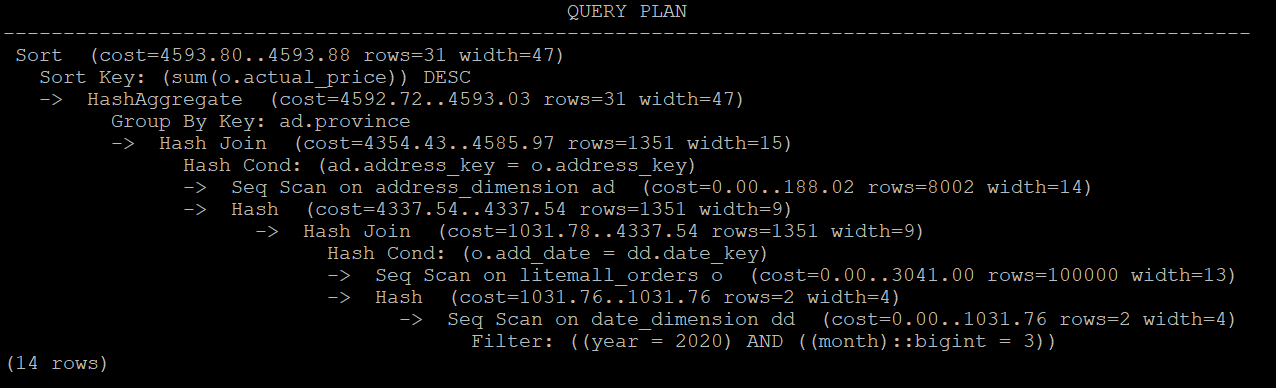
AND o.add\_date = dd.date\_key

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

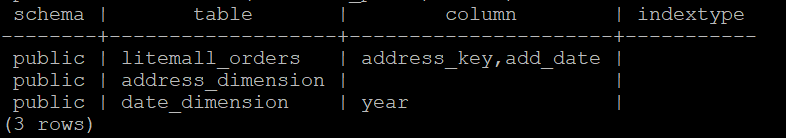
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

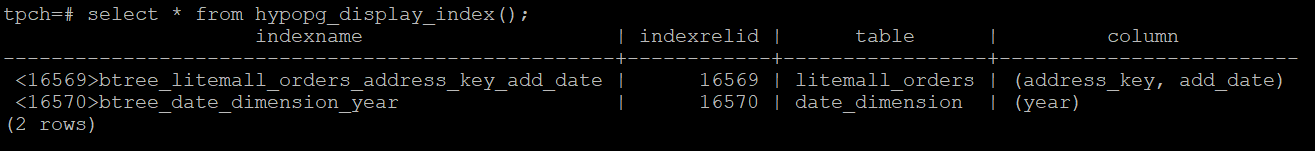
GROUP BY ad.province

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。

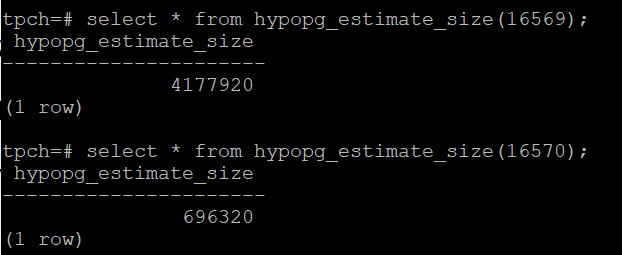
select \* from hypopg\_display\_index();



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

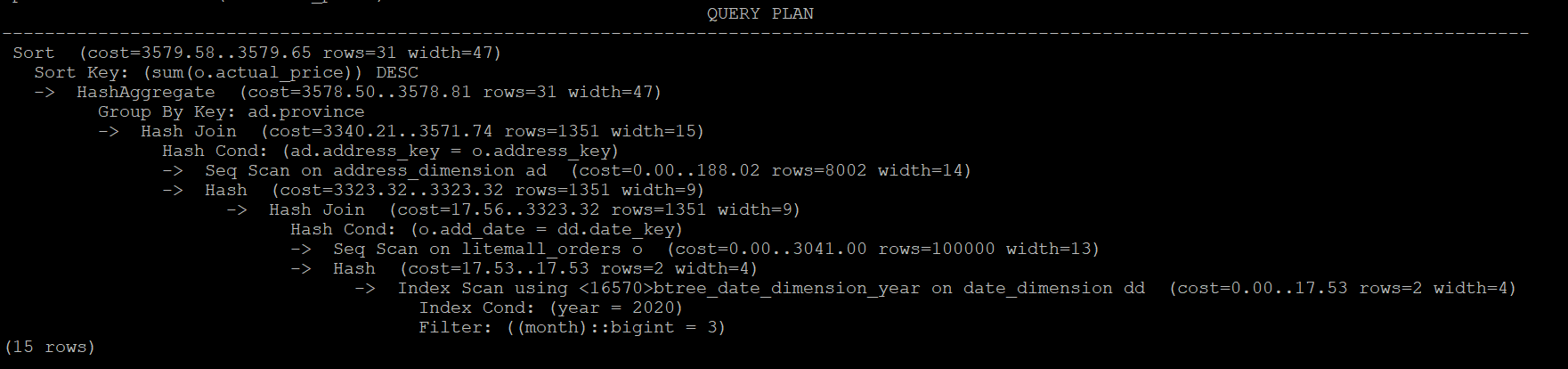
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

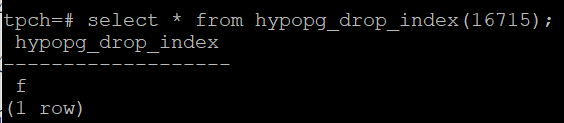
GROUP BY ad.province

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



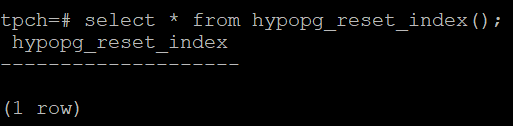
6. 删除某一个索引虚拟列，将执行结果截图。

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



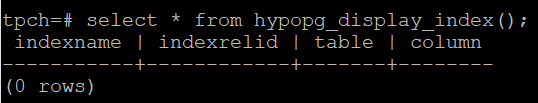
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

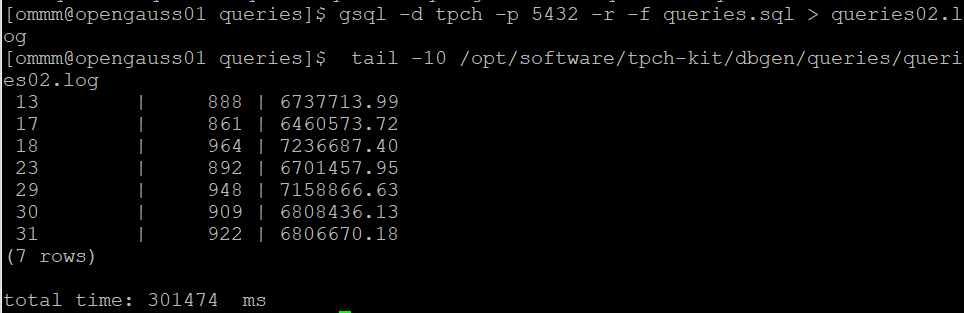
select \* from hypopg\_display\_index();



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

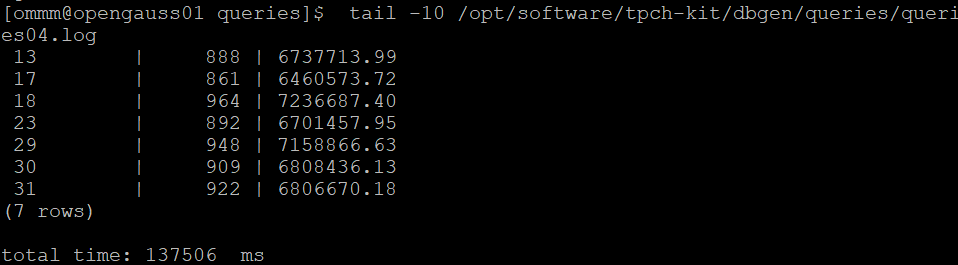


挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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/queries04.log

通过对表中的数据添加较多的列索引，提升表的查询效率



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：shared\_buffers、max\_connection、effective\_cache\_size、effective\_io\_concurrency、wal\_buffers,random\_page\_cost、default\_statistics\_target

原因：在初始化的数据库中，存在不合理的参数配置和潜在风险，如果参数没有得到优化，会导致数据库执行命令的效率变差，耗费额外的资源。

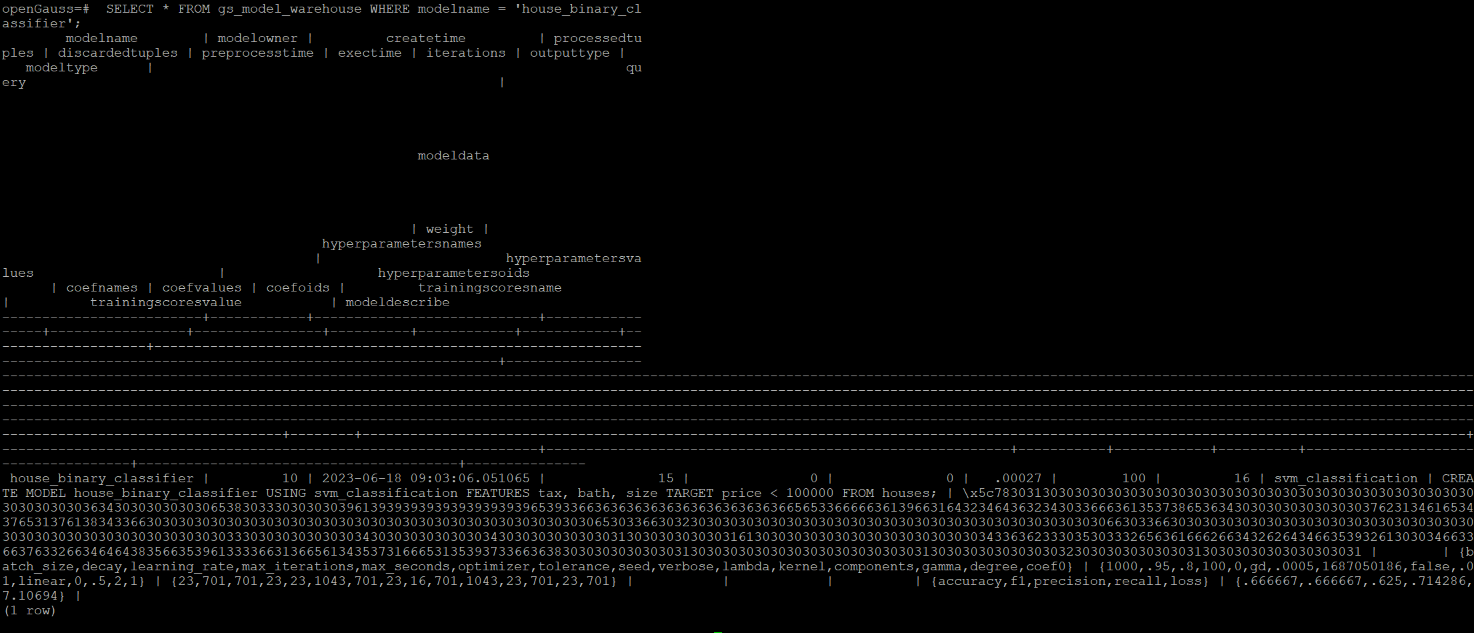
实践思考题2：索引的使用，对于执行SQL有什么好处？除了使用索引和参数外，还有哪些方面可以对数据库进行优化？

好处：1.能够提升查询的速度，通过在索引列上创建索引，可以快速定位到符合查询条件的记录，避免全表扫描的开销；2.减少数据的排序与聚合操作，使用合适的索引可以避免在查询过程中对数据进行排序与聚合操作从而减少查询时间与计算资源消耗；3.提供唯一性约束：索引可以作为唯一性约束，保证数据库表的某些列值的唯一性；4.支持快速连接和关联操作，在多表连接和复杂查询时，索引可以显著减少连接操作的执行时间。5.减少存储空间的需求，通过使用索引可以有效地组织和存储数据，减少冗余和重复存储，提高存储空间利用率。

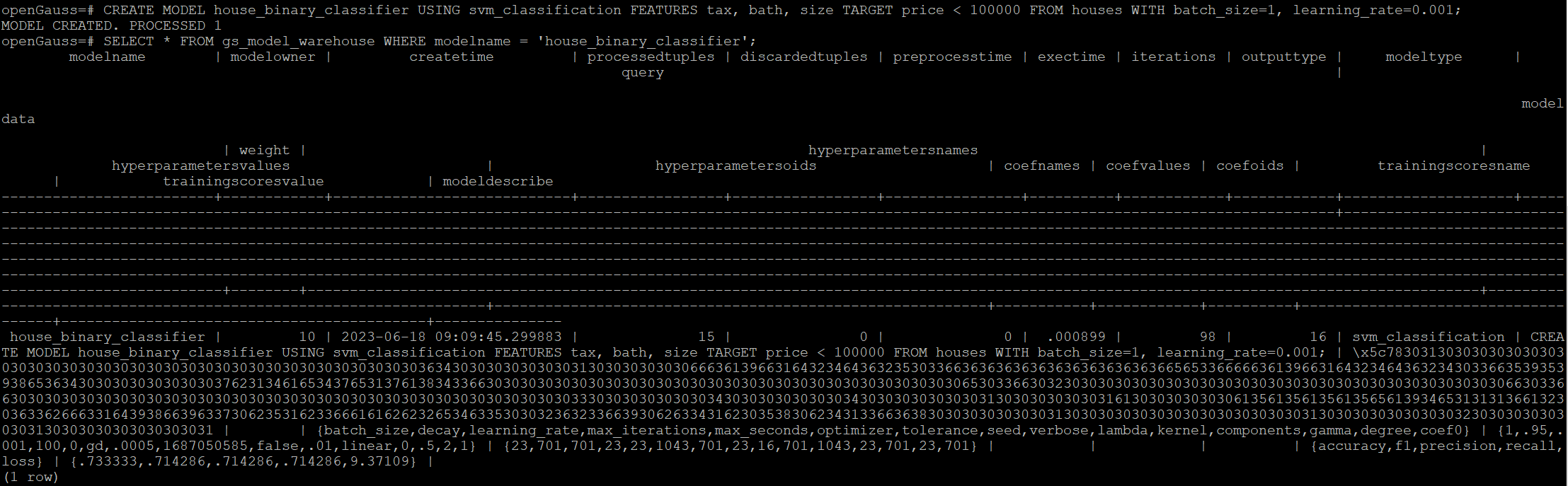
优化方法：1.设计合理的表结构，正确的数据类型；2.查询优化，避免不必要的表连接，使用合适的过滤条件，避免使用全表扫描等；3.缓存优化，合理使用缓存可以减少数据库访问次数。可以使用数据库本身的缓存机制也可以使用应用程序层面的缓存。4.建立分区与分表，对于数据量较大的表，可以使用分区分表技术来提高查询性能，减少查询范围，提高查询速度。

# 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？

预测目标方面，分类模型主要用于预测离散的类别或标签，而回归模型用于预测连续的数值。

输出类型方面，分类模型的输出是离散的类别标签或类别概率，表示输入数据属于每个类别的概率。回归模型输出的是连续的数值，可以是实数或整数。

损失函数方面，分类模型通常使用交叉熵等损失函数来度量预测结果与真是标签之间的差异。回归模型则使用均方误差、平均绝对误差等作为损失函数。

输出解释方面，分类模型的输出易于解释为不同类别的概率或标签。回归模型的输出则需要根据具体问题进行解释和理解。

实践思考题2：什么是SVM算法？

SVM算法即为支持向量机算法，该算法是一种常用的监督学习算法，用于分类和回归问题。它的主要思想是在特征空间中找到一个最优的超平面或者曲面，将不同类别的样本分隔开来。支持向量机存在四个关键概念，分别为支持向量，软间隔、核函数以及凸优化问题。支持向量机具有良好的泛化能力以及鲁棒性。

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：分类正确的样本数与总样本数之比。

精确率：分类为正例的样本中，实际为正例的比例。

召回率：实际为正例的样本中，被正确预测为正例的比例。

F1 score：精确率与召回率的调和平均值，综合考虑了模型的准确率和找回能力。

特异度：实际为负例的样本中，被正确预测为负例的比例。

AUC-ROC：ROC是以真正例率为纵轴，假正例率为横轴，绘制的曲线。而AUC值为ROC曲线与横轴之间包围的面积。AUC-ROC衡量了模型在不同阈值下的分类能力

对数损失：用于测量分类模型在概率估计方面的准确性。对数损失越小，模型的预测概率分布越接近真实分布。

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：预测值与真实值之间差异的平方均值。

均方根误差：均方误差的平方根。

平均绝对误差：预测值与真实值之间差异的绝对值均值。

R2 score：即决定系数，反映了因变量的变异程度可以由模型解释的比例。

相对平均误差：预测误差与真实值的平均绝对误差的比值。