openGauss AI特性创新实践课



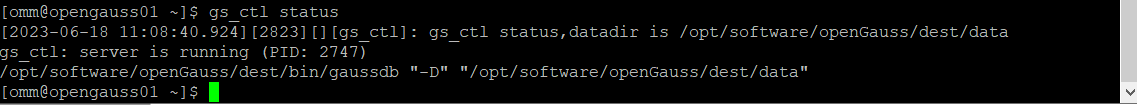
华为技术有限公司

# 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：

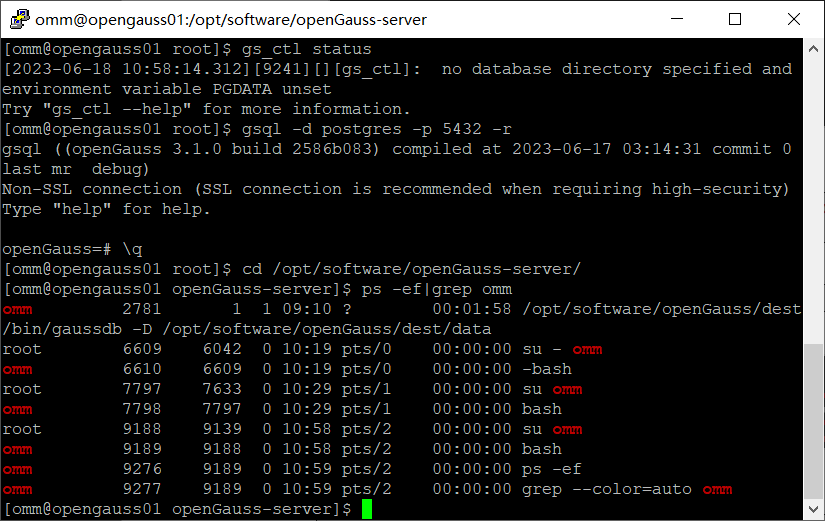
任务一：数据库状态验证

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？

1.可以定制自己的配置：通过源码编译的话，可以更改其中一部分的配置以适应自己的需求。也可以通过自己的技术手段对其进行优化或者删减。源码编译为用户提供了灵活性和精确性，让他们能够根据自己的实际情况对数据库进行优化和定制。

2.可以提升系统安全性：可以方便用户进行对源码的审查，在一定程度上减少或者禁止软件后门的安插。提升用户系统的安全。源码编译为用户提供了更大的控制权，便于用户拥有更大的操作空间。

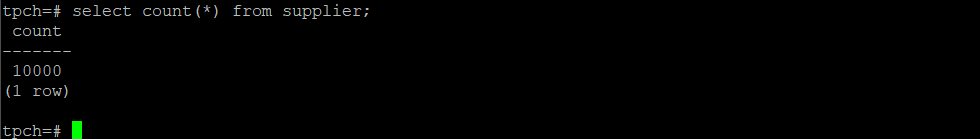
3.可以便于从业者进行学习：可以在一定程度上了解数据库的核心原理（例如数据库对数据处理的手段或者OpenGuass中对于数据和ai的关系）。加深对数据库内部工作原理和开发技术的理解，为学生提供了教育资源。

# 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;

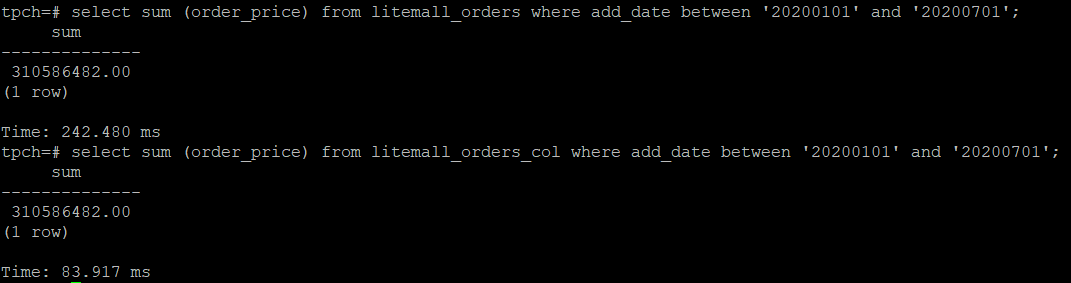


任务二：行存表与列存表执行效率对比

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';

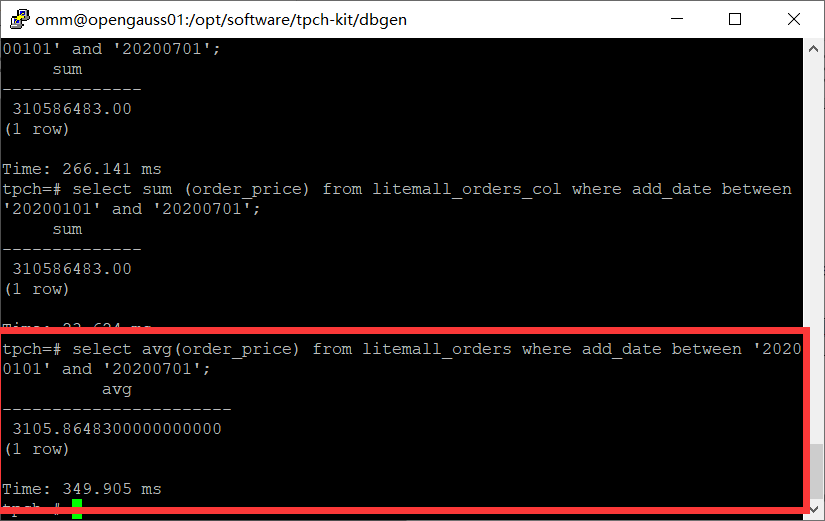
select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



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';

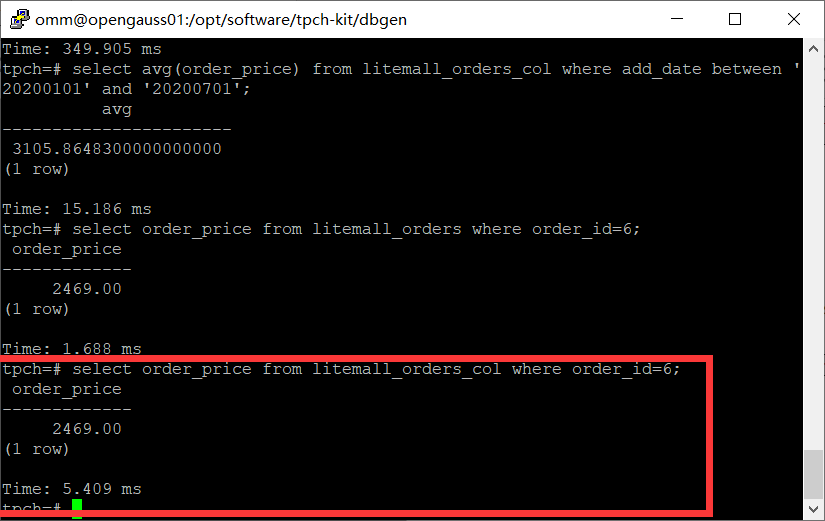
select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



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;

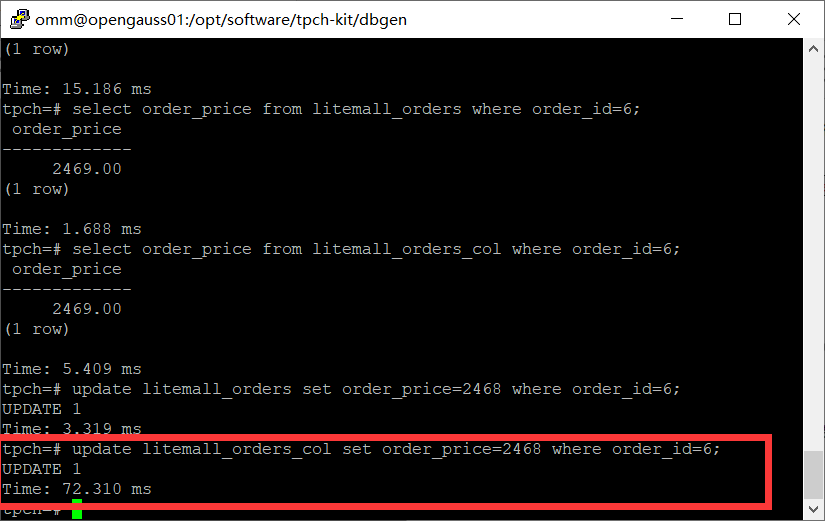
select order\_price from litemall\_orders\_col where order\_id=6;



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;

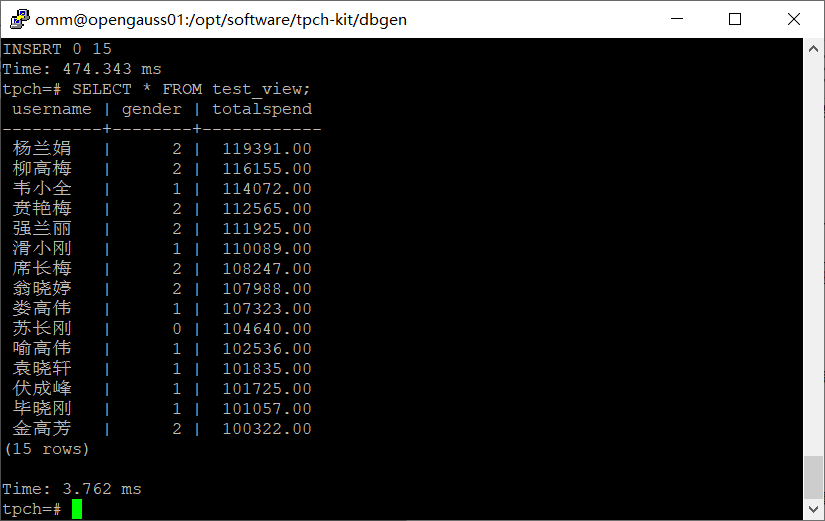
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

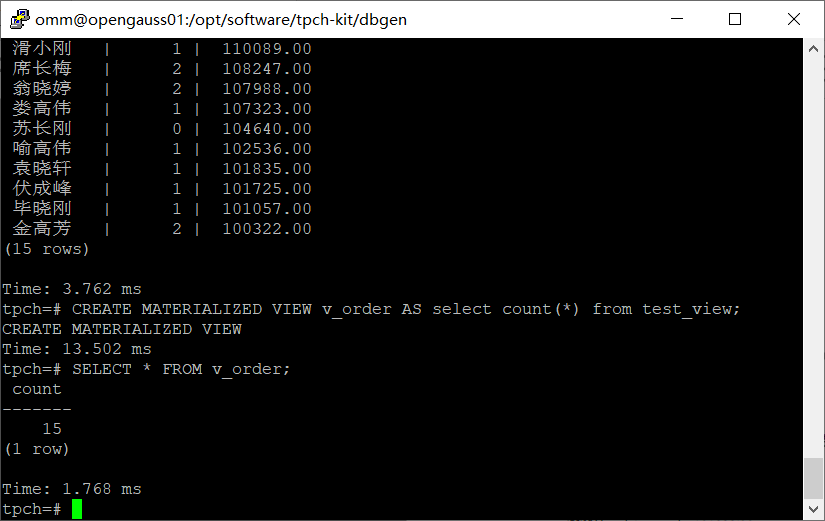
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



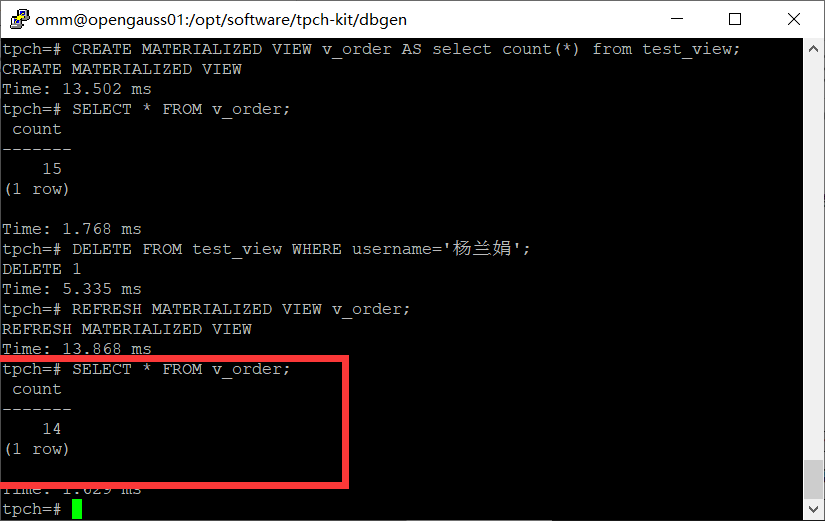
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



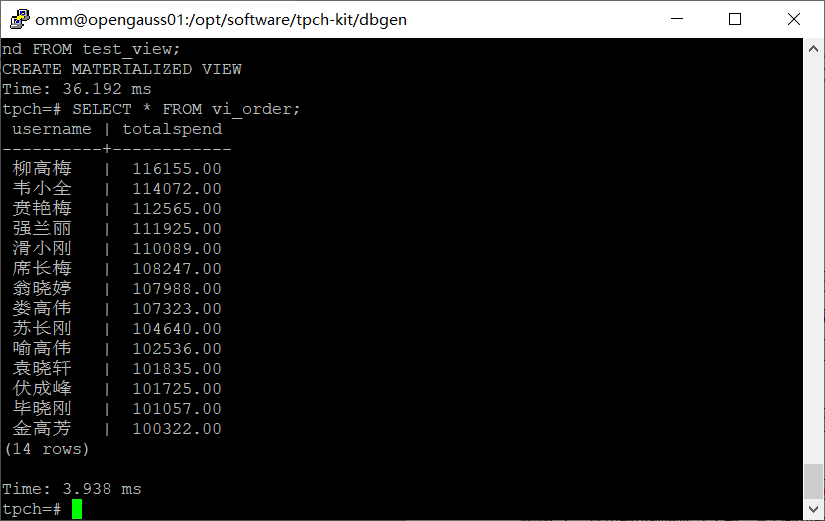
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



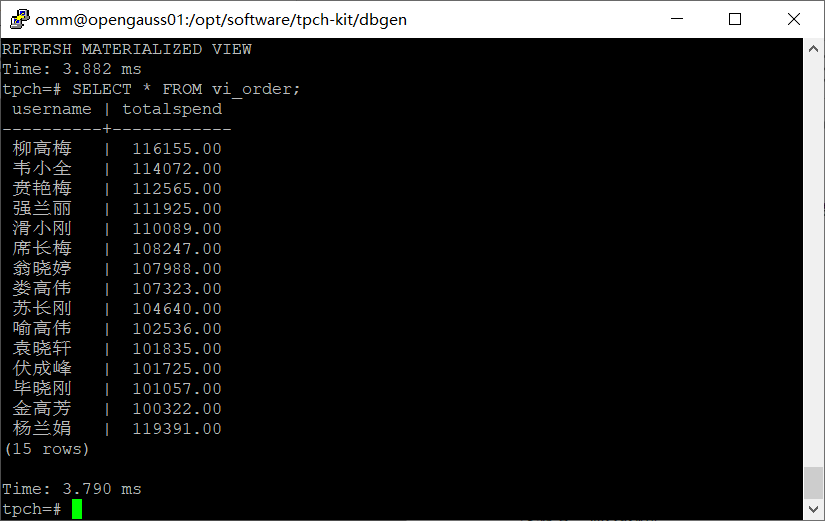
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

为何执行的时间不同：因为行存表与列存表在数据储存和访问方式上存在差异。行存表中，数据按行储存在磁盘上，所以每一行包含了此行的所有列，无需通过磁盘进行更多操作。行存表更适合条件过滤或者返回多个列的操作。而列存表中，数据按列储存在磁盘上。当操作涉及大量行或者少量列的查询操作时，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？

1.数据更新方式：

全量物化视图：全量物化视图是通过完全重新计算和刷新视图中的数据来进行更新。这意味着在每次刷新时，所有相关的基本表都被扫描和处理，以确保视图中的数据与基本表保持一致。

增量物化视图：增量物化视图是通过跟踪和记录基本表的变化来进行更新。只有与变化相关的数据才会被更新到物化视图中，这样可以减少刷新的开销和时间。通常使用日志或触发器等机制来捕获和应用变化。

2.刷新频率：

全量物化视图：全量物化视图通常需要在较长时间间隔内进行刷新，因为它需要对所有数据进行重新计算和加载。刷新时间可能会比较长，尤其是当基本表数据量较大时。

增量物化视图：增量物化视图可以更频繁地进行刷新，因为只有变化的数据需要更新。这使得增量物化视图更适合于需要更实时数据的场景。

3.查询性能

全量物化视图：由于全量物化视图需要在每次刷新时重新计算和加载数据，所以查询性能通常在刷新期间较低，而在刷新完成后才能获得较好的性能。

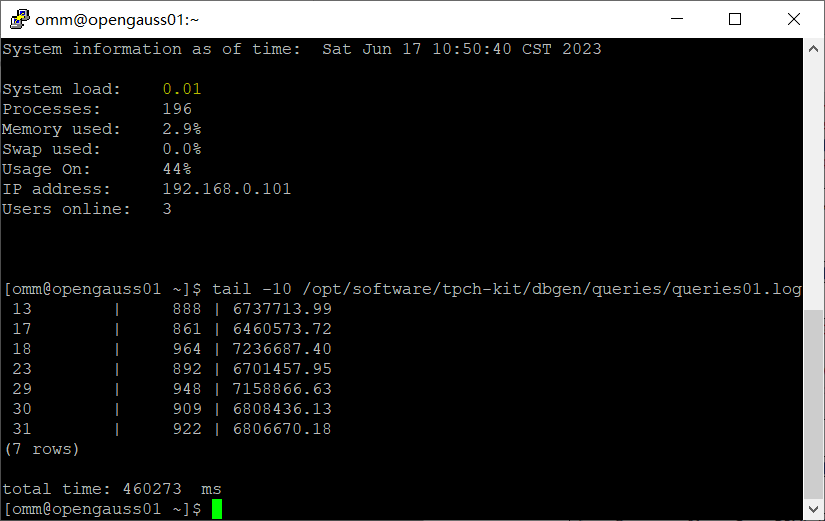
增量物化视图：增量物化视图由于只更新变化的数据，所以在刷新期间的查询性能相对较好。它可以提供更快的查询响应时间，并在基本表的变化较大时尤其有效。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

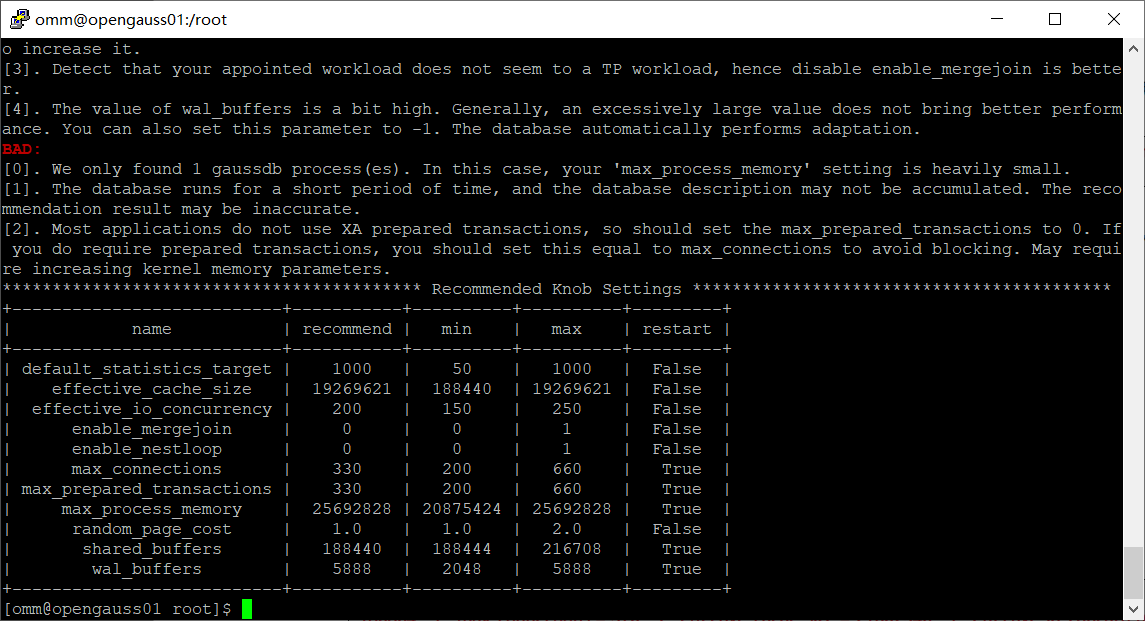
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

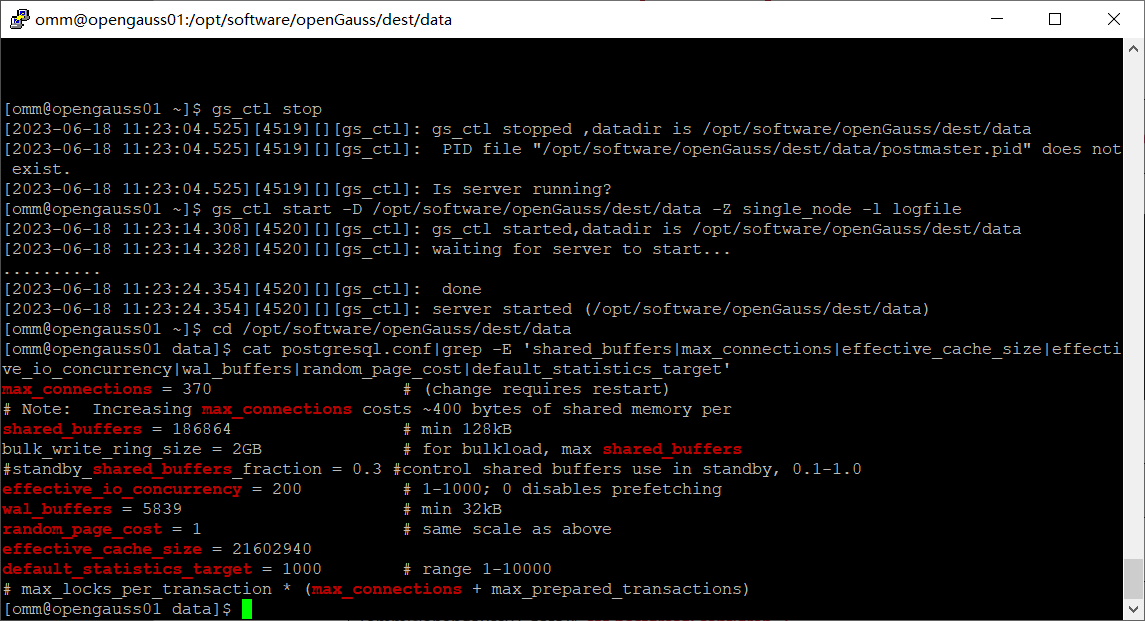
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

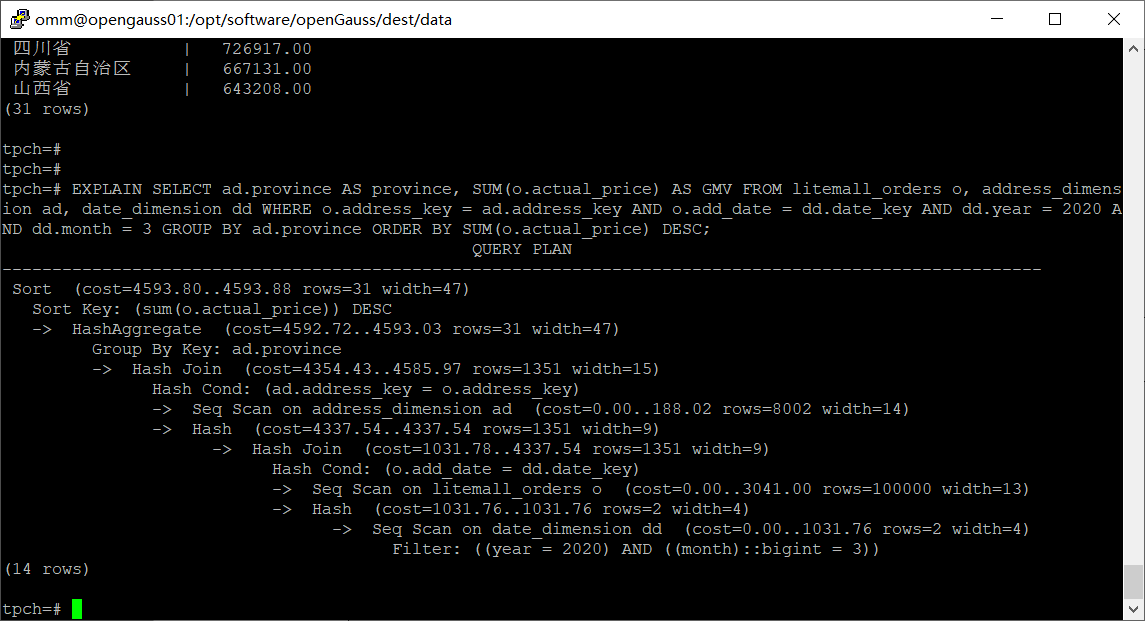
AND o.add\_date = dd.date\_key

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

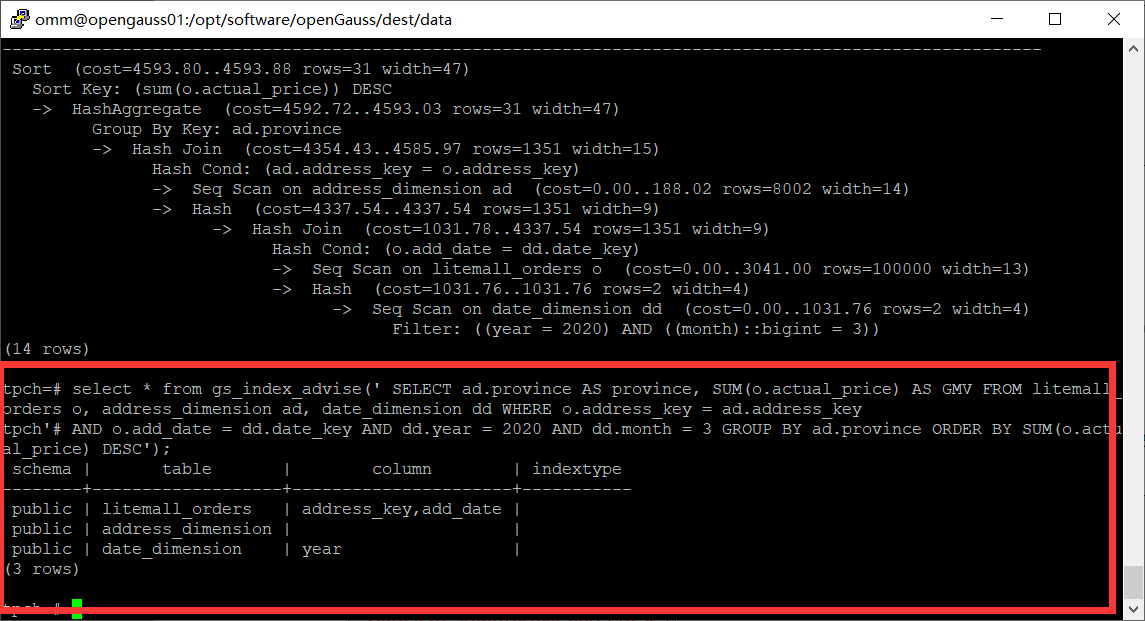
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

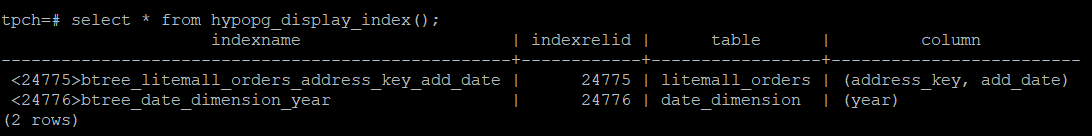
GROUP BY ad.province

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。

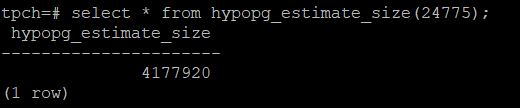
select \* from hypopg\_display\_index();

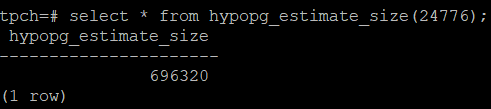


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

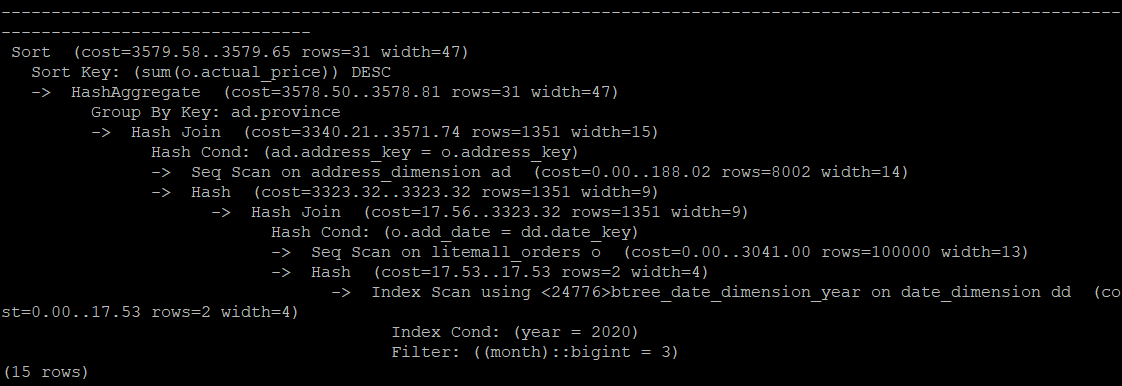
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

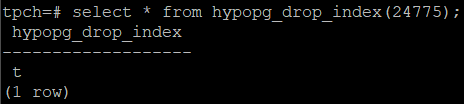
GROUP BY ad.province

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



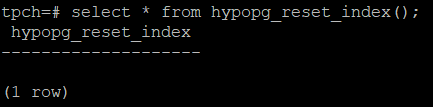
6. 删除某一个索引虚拟列，将执行结果截图。

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



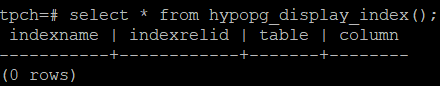
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

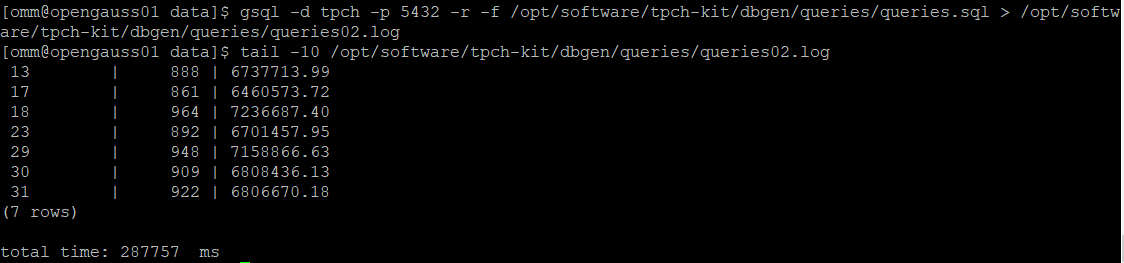
select \* from hypopg\_display\_index();



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：

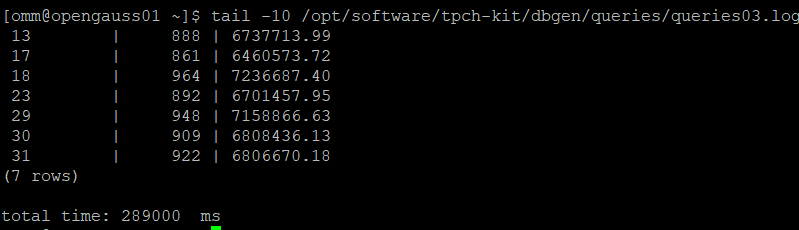
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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



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？

|  |
| --- |
| gs\_guc set -D /opt/software/openGauss/dest/data/ -c "shared\_buffers = 186864" -c "max\_connections = 370" -c "effective\_cache\_size = 21602940" -c "effective\_io\_concurrency = 200" -c "wal\_buffers = 5839" -c "random\_page\_cost = 1" -c "default\_statistics\_target = 1000" |

优化参数：shared\_buffers = 186864，max\_connections = 370，effective\_cache\_size = 21602940，effective\_io\_concurrency = 200，wal\_buffers = 5839，random\_page\_cost = 1，default\_statistics\_target = 1000

1. shared\_buffers = 186864增加了共享缓存区的大小，可以减少对磁盘的io操作，从而提升查询性能和响应速度

2. max\_connections = 370 配置了数据库系统允许的最大连接数，可以更好地估计数据在内存中的可用性，优化查询计划的生成和执行。

3. effective\_io\_concurrency = 200 增加IO操作的并发度可以提高数据库系统在进行磁盘读取和写入时的效率，加快数据访问速度。

4. wal\_buffers = 5839 增加WAL缓冲区的大小可以提高事务日志的写入性能和持久性。

5. random\_page\_cost = 1 通过设置较低的随机IO成本，数据库系统更倾向于选择使用随机IO操作，从而提高查询的执行效率。

6. effective\_cache\_size = 21602940 通过设置适当的有效缓存大小，数据库系统可以更好地估计数据在内存中的可用性，优化查询计划的生成和执行。

7. default\_statistics\_target = 1000 增加统计信息的目标数量可以提高查询优化器生成更准确的查询计划，提高查询性能。

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？

1.提高查询速度：索引可以加快查询的速度，因为数据库系统可以使用索引快速定位到满足查询条件的数据，而不需要逐行扫描整个表。

2.减少IO操作：索引可以减少磁盘IO操作的次数，因为数据库可以通过索引直接访问所需的数据页，而不是从磁盘读取整个表。

3.改善数据的排序和分组：索引可以用于排序和分组操作，提高排序和分组操作的效率。

还可以进行以下操作：

1. 硬件优化：选择适当的硬件设备，包括CPU、内存、磁盘等，以满足数据库的需求，并优化存储设备的配置和性能。

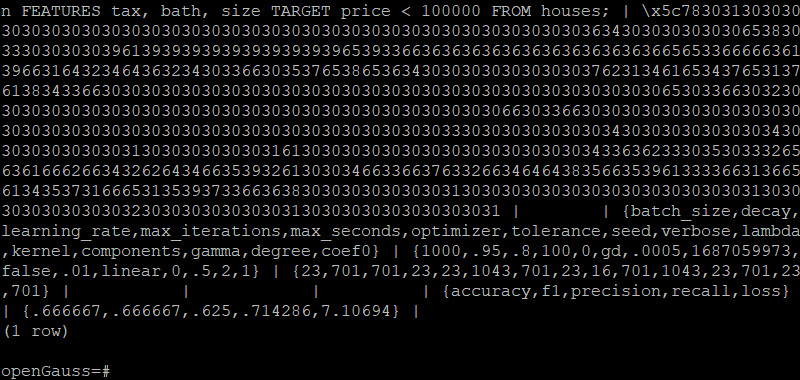
2. 数据库设计：良好的数据库设计可以提高数据库的性能。这包括合理的表结构设计、关系建模、范式化等。

3. 查询优化：通过编写高效的查询语句、避免不必要的查询、合理使用连接和子查询等方法，可以提高查询性能。

# 关卡四、openGauss的DB4AI特性应用

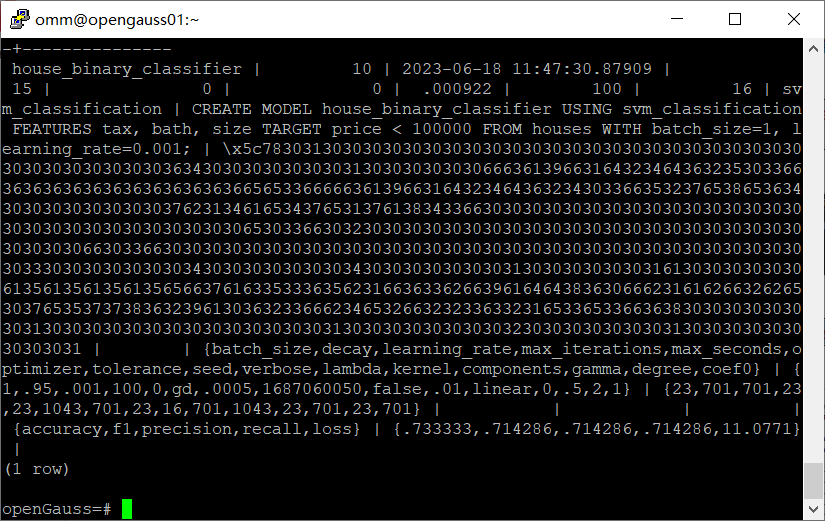
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



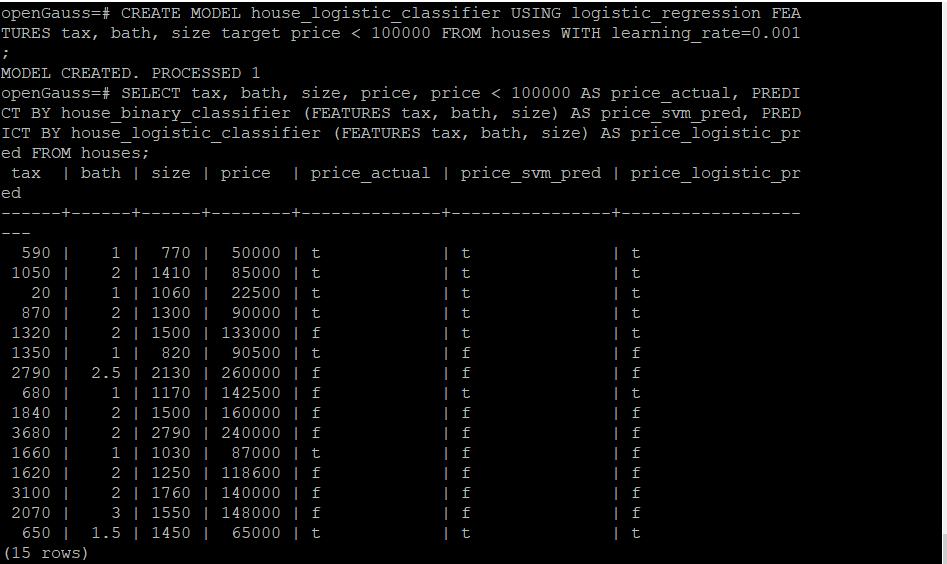
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型用于解决分类问题，即将输入样本分为不同的离散类别。回归模型用于解决回归问题，即预测一个连续的数值输出。

实践思考题2：什么是SVM算法？

SVM即支持向量机，基于统计学习理论中的结构风险最小化原则，通过在特征空间中找到一个最优的超平面来进行分类或回归。即超平面可以将不同类别的样本分开，并且使得距离最近的样本点（支持向量）到该超平面的距离最大化，这样可以提高分类的准确性和鲁棒性。

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？

1. 准确率（Accuracy）：准确率是最简单直观的评价指标，表示分类器正确分类的样本数与总样本数之比。

2. 精确率（Precision）：精确率衡量了分类器在预测为正例的样本中有多少是真正的正例，即预测为正例且正确分类的样本数与预测为正例的样本总数之比。

3. 召回率（Recall）：召回率衡量了分类器在所有真正的正例中有多少被正确预测出来，即预测为正例且正确分类的样本数与真正例的样本总数之比。

4. F1 值（F1-Score）：F1 值是精确率和召回率的综合评价指标，是二者的调和均值，用于综合考虑分类器的准确性和召回率。

5. 特异度（Specificity）：特异度衡量了分类器在预测为反例的样本中有多少是真正的反例，即预测为反例且正确分类的样本数与预测为反例的样本总数之比。

6. ROC 曲线与AUC：ROC（Receiver Operating Characteristic）曲线是反映分类器性能的一种综合指标，它以真正例率（TPR，召回率）为纵轴，以假正例率（FPR，1-特异度）为横轴，展示了不同阈值下分类器的性能变化。

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. 均方误差MSE：均方误差是最常用的回归评价指标，它衡量了模型预测值与真实值之间的平均差异程度。

2. 均方根误差RMSE：均方根误差是均方误差的平方根，它与原始数据的单位相一致，更直观地反映了模型预测值与真实值之间的平均差异程度。

3. 平均绝对误差MAE：平均绝对误差衡量了模型预测值与真实值之间的平均绝对差异程度，它对异常值不敏感。

4. 相关系数：相关系数衡量了预测值与真实值之间的线性相关程度。