openGauss AI特性创新实践课



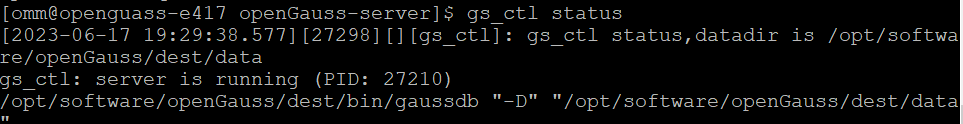
华为技术有限公司

# 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：

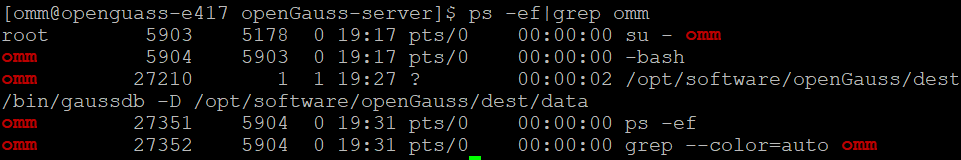
任务一：数据库状态验证

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？

源码编译和安装数据库有以下几个原因：

1. 最新版本：从源码编译并安装数据库可以获得最新版本的数据库软件，这有助于解决已知的漏洞和缺陷，并提供新的功能和性能优化。

2. 自定义选项：从源码编译并安装数据库也可以让用户选择他们所需的选项，这包括编译时启用或禁用某些功能、选择适当的配置选项等，满足自己的需求。

3. 操作系统兼容性：在某些情况下，用户的操作系统和硬件可能会影响数据库的性能和稳定性。通过从源代码编译并安装数据库，可以确保数据库与操作系统和硬件进行了充分的集成和优化。

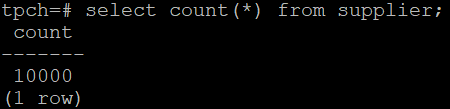
4. 技术支持：源码编译和安装数据库也可以帮助用户更好地理解数据库如何工作，并具备更好的技术支持和故障排除能力。

# 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：

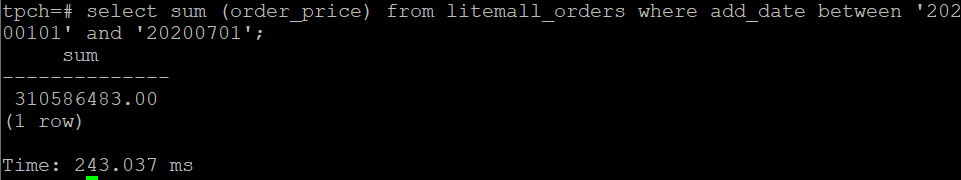
select count(\*) from supplier;;



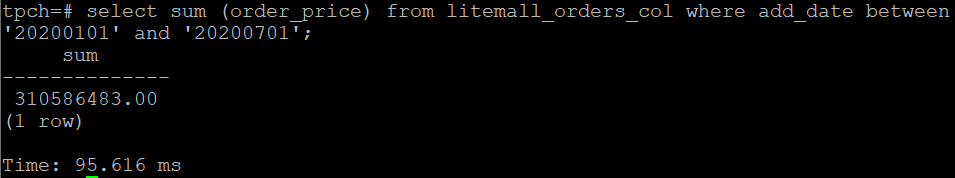
任务二：行存表与列存表执行效率对比

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';

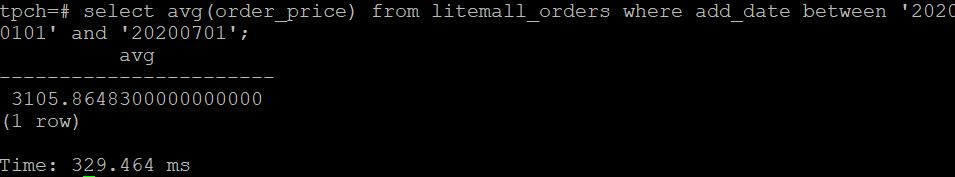


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

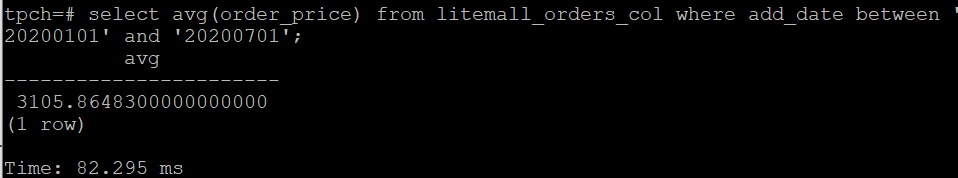


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

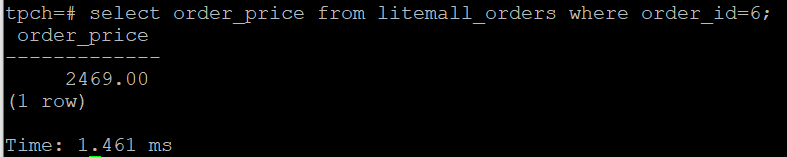


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

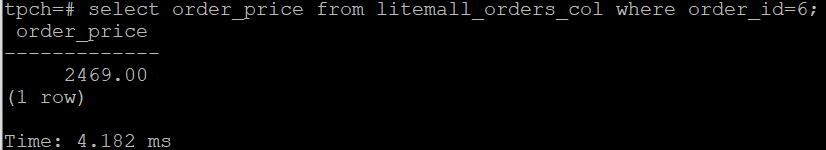


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

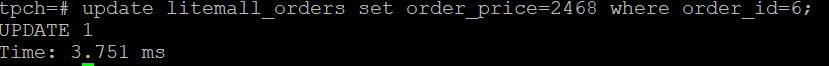


select order\_price from litemall\_orders\_col where order\_id=6;

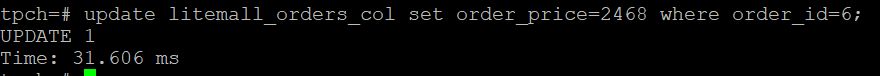


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

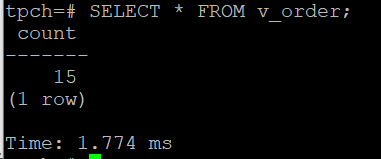
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



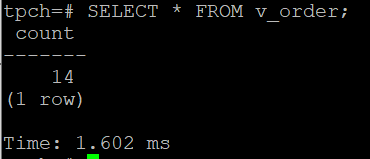
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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;



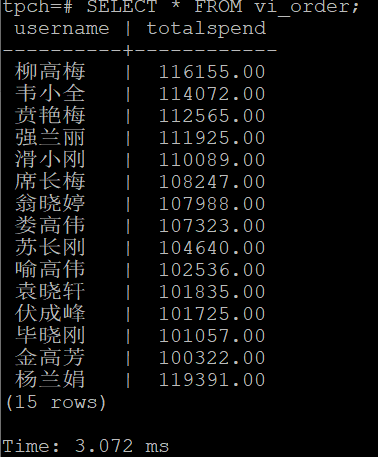
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

行存表和列存表在执行相同的SQL语句时，执行时间不同的原因是因为它们的数据组织方式和查询方式不同。

行存表使用的是每行一个记录的方式进行存储，查询时需要检索整行数据，无论查询需要的数据有多少。相比之下，列存表使用的是按列存储的方式，只检索需要的数据列，而不需要检索整行。这使得列存表在处理大量数据时更加高效。

对于具体的 SQL 查询类型，行存表效率更高的情况包括：

1. 需要频繁执行插入、更新、删除操作的场景。因为行存表只需要处理一个行级别更新操作，而列存表需要更新每个列。

2. 需要频繁查询单行记录的场景。行存表按照行存储，检索单行记录比列存表更高效。

而列存表效率更高的情况包括：

1. 需要分析大量数据、查询多个列、进行聚合操作的场景。列存表只需要检索需要的列，而不需要检索整行，提高了查询效率。

2. 资源限制比较严重，需要在有限的内存和磁盘空间下，尽量减少 I/O 操作的场景。列存表可以只检索需要的列，尽量减少 I/O 操作，使得效率更高。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量物化视图和增量物化视图是物化视图的两种类型，它们的主要区别在于数据的更新方式。

全量物化视图是一种静态的物化视图，它通过将查询结果缓存到磁盘中来减少查询时间。它在创建时将整个查询结果存储在磁盘上，当底层表的数据发生变化时，全量物化视图的数据也需要更新。这个更新过程是一个长时间的操作，需要重新计算整个查询结果，因此更新操作的成本比较高。

增量物化视图，顾名思义，是一种增量更新的物化视图。增量物化视图会为每个底层表上的数据变化创建增量更新，这样可以更快地更新物化视图，因为只有变化的部分需要被更新。增量物化视图通过增量更新减少了更新成本，并提高了更新速度。

因此，根据更新数据的方式，全量物化视图和增量物化视图的不同主要表现在：

1. 更新成本：全量物化视图需要重新计算和更新整个查询结果，代价较高，而增量物化视图只需要更新变化的部分，成本较低。

2. 更新速度：增量物化视图具有更快的更新速度，因为只更新变化的部分，不需要重新检索整个底层表。

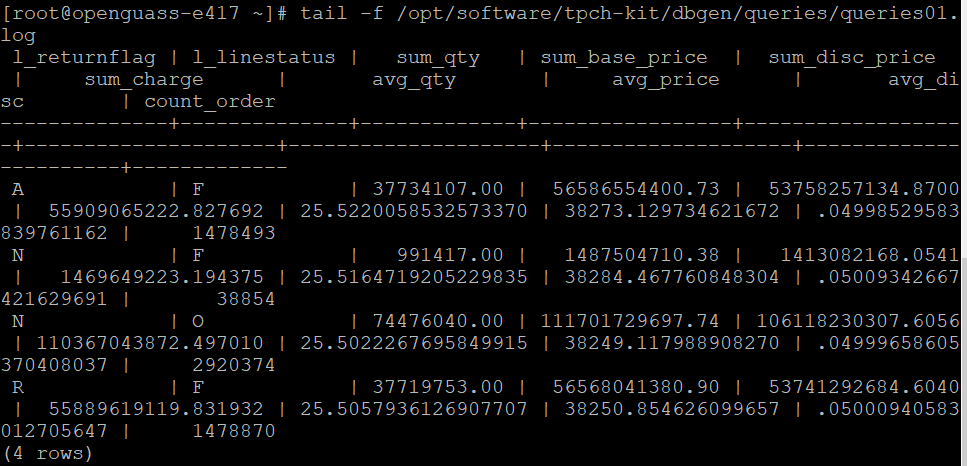
3. 适用场景：全量物化视图适用于查询结果不频繁变化且查询结果集比较小的情况，而增量物化视图适用于查询结果频繁变化且查询结果集比较大的情况。

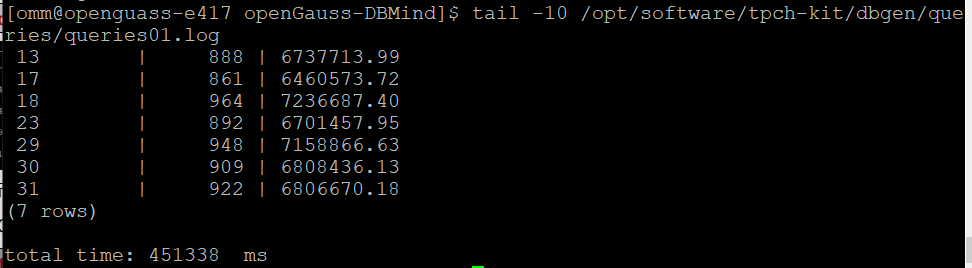
# 关卡三、openGauss的AI4DB特性应用

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：

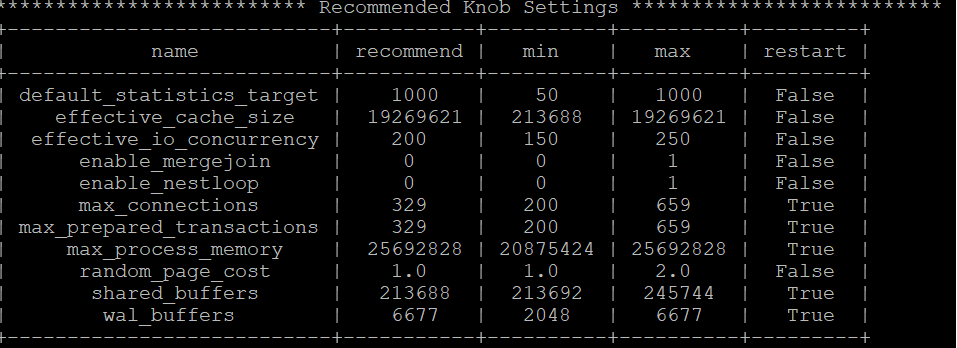
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log

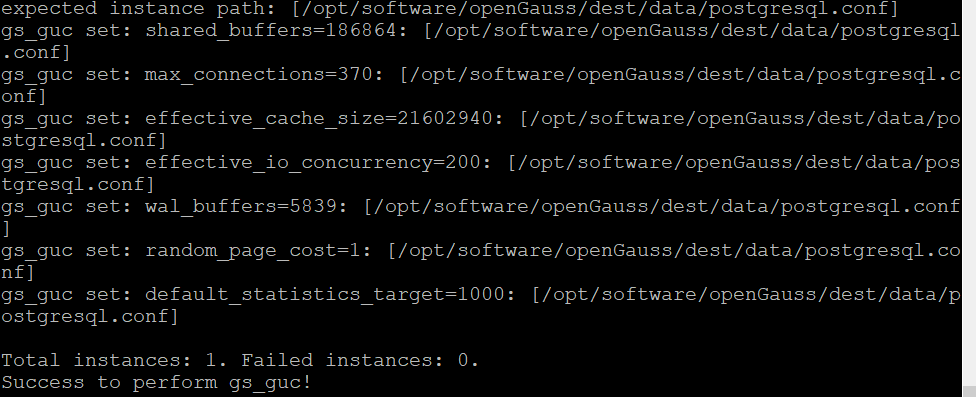




2. 使用omm用户，执行X-Tuner进行参数建议优化，将执行结果截图

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

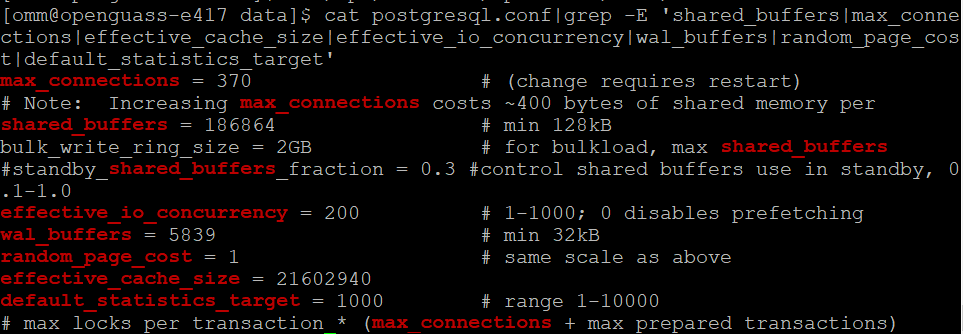




3.重启完成后，获取参数值：

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

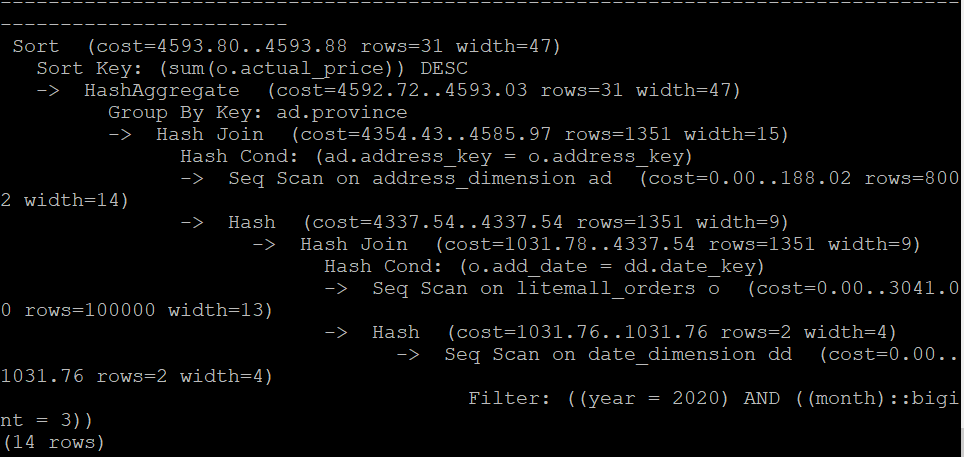
AND o.add\_date = dd.date\_key

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

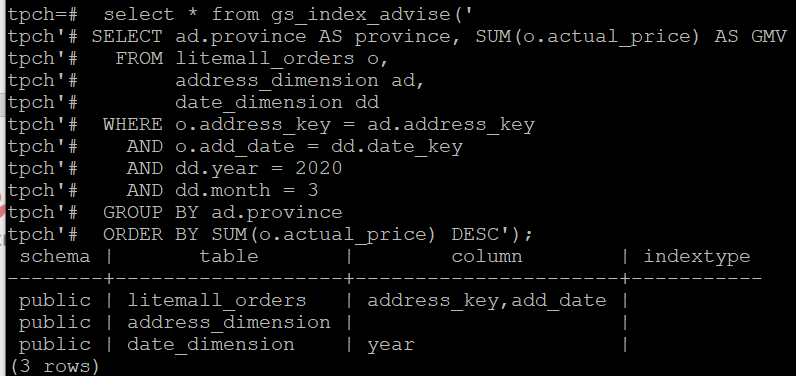
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

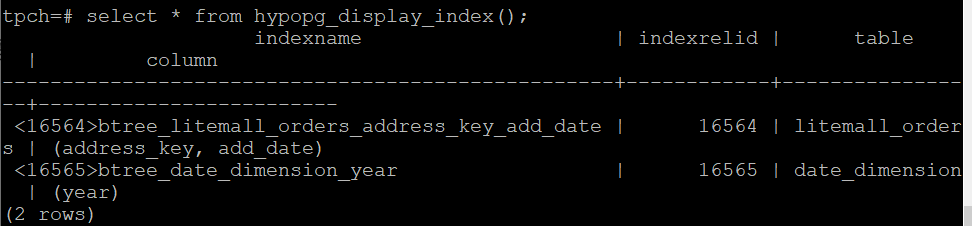
GROUP BY ad.province

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。

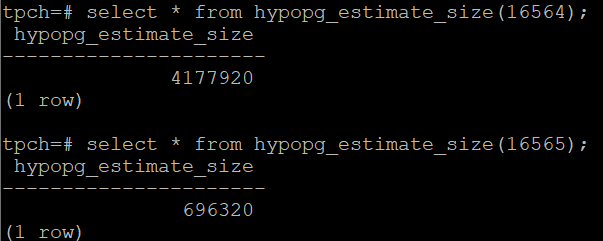
select \* from hypopg\_display\_index();



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

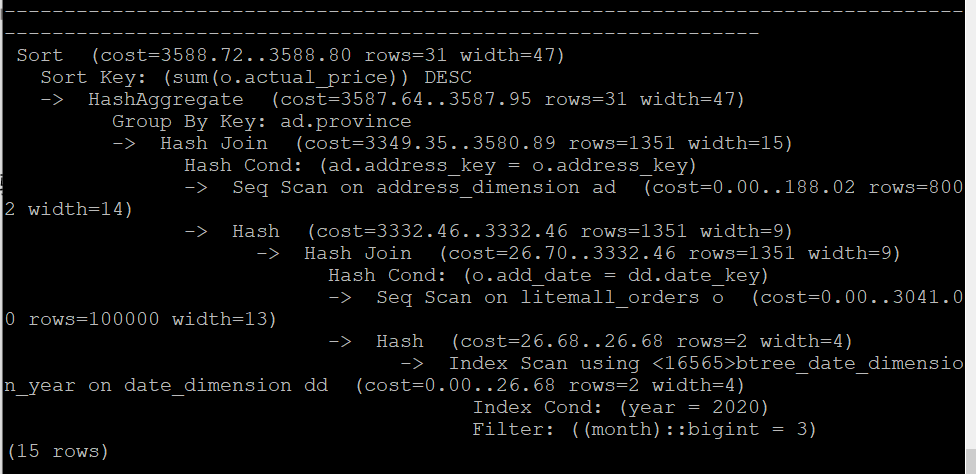
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

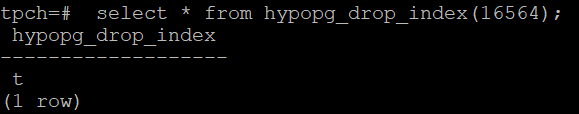
GROUP BY ad.province

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



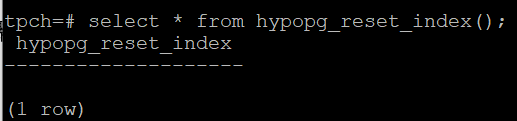
6. 删除某一个索引虚拟列，将执行结果截图。

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



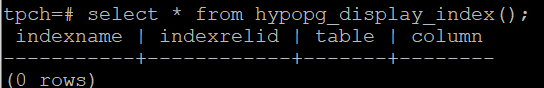
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

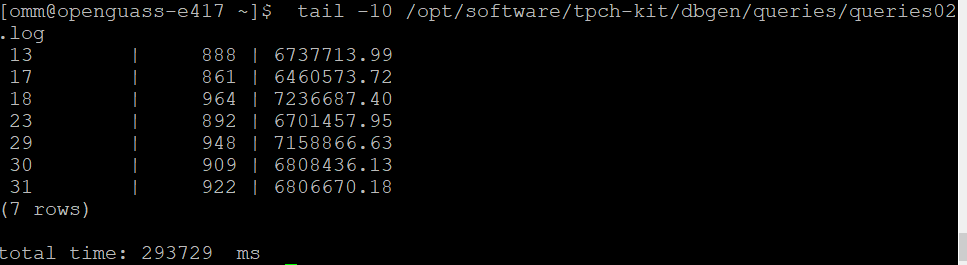
select \* from hypopg\_display\_index();



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：

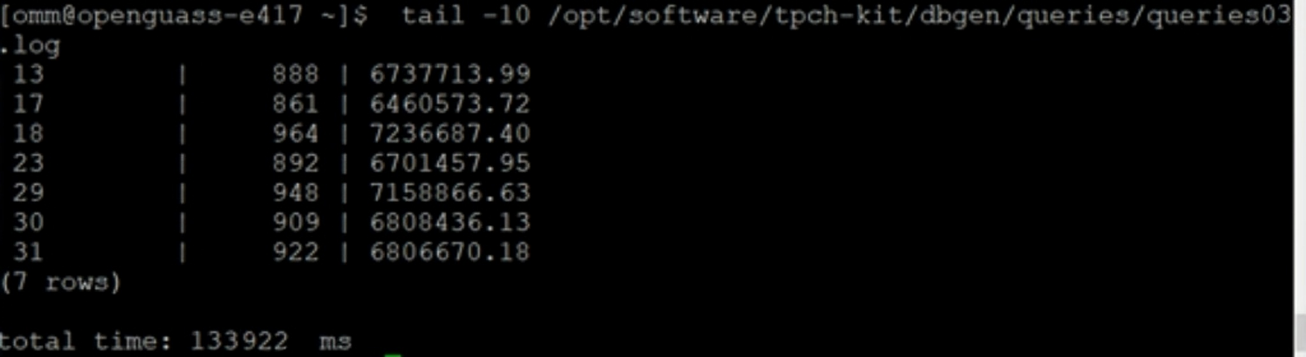
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

X-Tuner优化的参数包括：

1. 缓冲池相关参数：主要包括缓冲池大小、最小缓冲池大小、缓冲池预取等参数，它们会影响数据库的读取速度和内存消耗。

2. 日志相关参数：主要包括日志缓冲池大小、日志文件大小等参数，这些参数可以提高日志的写入性能。

3. 查询优化相关参数：主要包括SHARED\_POOL\_SIZE、PGA\_AGGREGATE\_TARGET等，可以提高 SQL 查询的性能。

4. 并发控制相关参数：主要包括MAX\_CONNECTIONS、DB\_BLOCK\_MAX\_DIRTY\_TARGET等，可以提高数据库多用户并发性能。

通过对这些参数的优化，可以提高数据库的性能和可靠性。例如，提高缓冲池等参数可以减少硬盘 I/O 操作，加快查询速度；提高日志相关参数可以提高日志的写入速度；优化查询优化相关参数可以提高 SQL 查询的性能；提高并发控制相关参数可以提高数据库的并发性。

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？

索引是一种提高数据库性能的重要手段，主要好处包括：

1. 提高查询效率。索引能够提高查询的速度，因为索引可以使查询语句快速定位目标数据。

2. 提高排序和分组操作的性能。索引可以为排序和分组操作提供帮助，缩短操作时间。

3. 减少磁盘 I/O。在没有索引的情况下，查询语句需要遍历整个表，而有了索引时，可以避免执行全表扫描，从而减少磁盘 I/O 操作，提高查询效率。

除了使用索引和参数外，还有以下几个方面可以对数据库进行优化：

1. 应用程序设计和优化。合理设计应用程序，包括数据表和查询语句的设计，在应用层加入缓存和分页功能等，减少对数据库的请求次数和数据传输量等，从而提高数据库性能。

2. 数据库结构设计和优化。合理设计数据库表结构，包括使用正确的数据类型、避免冗余和重复数据等，以达到最优的性能。

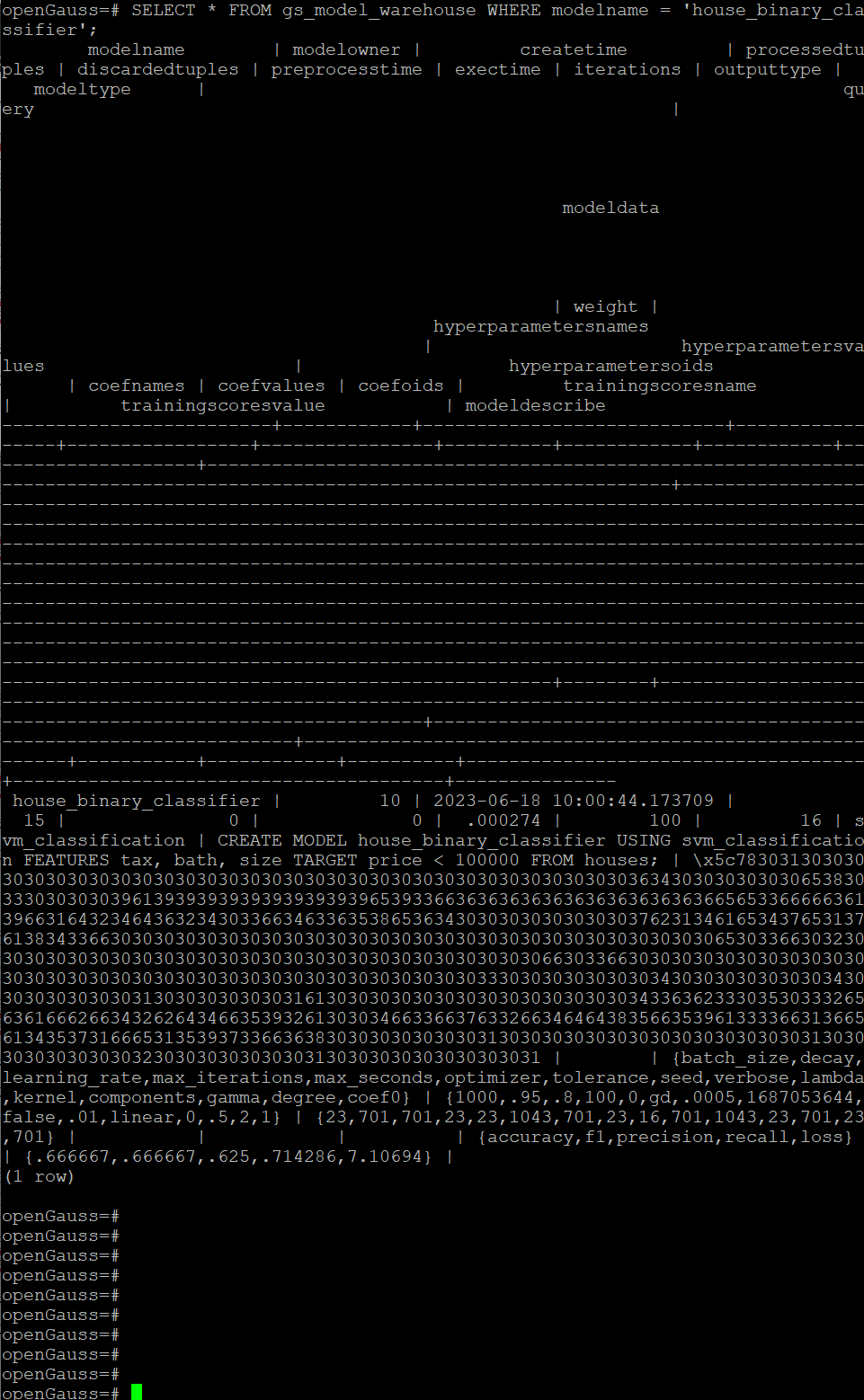
3. 硬件优化。优化硬件配置包括增加内存、使用更快的硬盘、使用更快的 CPU 和网络等，以提高数据库访问速度。

4. 压缩和分区。数据压缩可以减少存储和网络带宽，提高查询效率；数据分区可以提高查询效率，分散负载，实现更高的可靠性和更好的数据管理。

# 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';



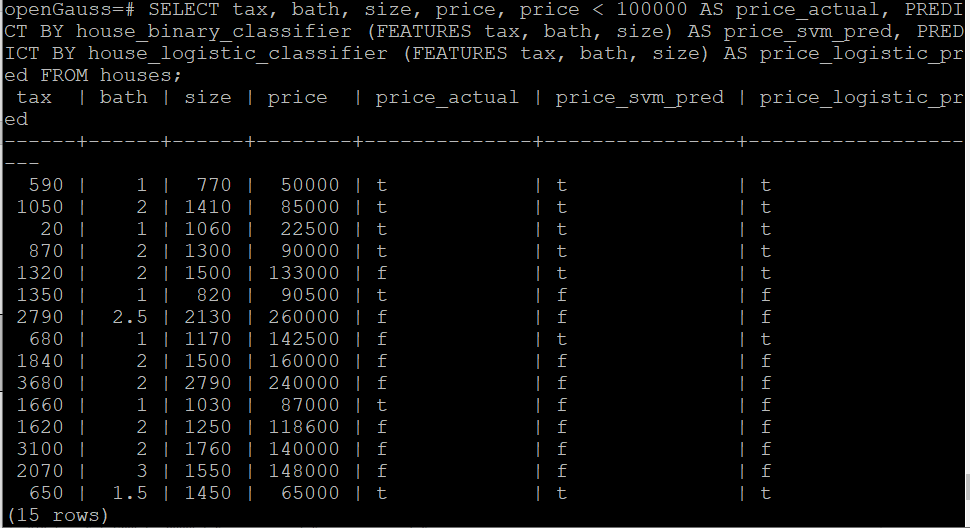
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型和回归模型都是机器学习中常用的模型，它们之间有以下不同：

1. 预测结果类型不同：分类模型的预测结果为类别标签或者类别概率值等离散型变量，用于对数据进行分类。而回归模型的预测结果为数值型变量，用于对数据进行取值预测。

2. 模型的评价标准不同：分类模型的评价标准包括准确率、召回率、精确率等，主要是衡量分类的准确度。而回归模型的评价标准包括均方误差、平均绝对误差等，主要是衡量预测结果的精准度。

3. 适用场景不同：分类模型适用于处理标称型或有序型数据，例如对图像进行分类、垃圾邮件识别等。而回归模型适用于处理连续型数据，例如对股票价格、气温预测等。

4. 模型的算法不同：分类模型的算法包括决策树、朴素贝叶斯、支持向量机等。而回归模型的算法包括线性回归、岭回归、Lasso 回归等。

实践思考题2：什么是SVM算法？

支持向量机（Support Vector Machine，SVM）算法是一种经典的监督学习算法，用于进行二分类和多分类问题的分类和回归分析，其优化目标是找到一个能够将类别分开的最优超平面。

SVM算法的基本思想是通过在数据中寻找一个最优的超平面，将不同类别的数据分开。这个最优超平面通常被定义为能够最大化离超平面最近的数据点（支持向量）到超平面的距离，这个距离被称为间隔。由于SVM算法只使用到了距离最近的支持向量，所以其计算开销比其他机器学习算法更小。

在实现过程中，SVM算法通过对数据点的映射，将原有的特征空间转化为高维空间，进而使得线性不可分的问题变为可分的问题。这是通过设计不同的核函数，将原空间的非线性数据映射到高维空间进行线性分类来实现的。

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？

分类问题中，评价指标是用来衡量机器学习模型对测试集中样本的分类准确度和性能的一种标准，常用的分类评价指标有以下四种：

1. 准确率（Accuracy）：是指分类器正确分类的样本数占总样本数的比例。其计算公式为：Accuracy = (TP + TN) / (TP + TN + FP + FN)，其中 TP 代表真正例数量，TN 代表真负例数量，FP 代表假正例数量，FN 代表假负例数量。准确率值越高，则模型的分类准确度越高。

2. 精确率（Precision）：是指所有被分类器预测为正类别的样本中，确实属于正类别的样本所占的比例。其计算公式为：Precision = TP / (TP + FP)。精确率高表示分类器对正类别的预测准确性高。

3. 召回率（Recall）：是指所有正类别的样本中，有多少被分类器正确识别为正类别。其计算公式为：Recall = TP / (TP + FN)。召回率高表示分类器能够正确识别正类别的数量比较多。

4. F1-score（F1）：是综合考虑精确率和召回率的评价指标。F1-score 是精确率和召回率的调和平均数，其计算公式为：F1 = 2 \* (Precision \* Recall) / (Precision + Recall)。F1-score 值越高，则模型的分类准确率和分类器识别正类别的准确性都越高。

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？

在回归问题中，评价指标用来衡量模型预测的结果与实际结果之间的差异和性能表现。下面介绍回归问题中常用的四个评价指标：

1. 均方误差（Mean Squared Error，MSE）：是回归模型中最常用的评价指标之一。MSE 是预测值与实际值之差的平方和的均值。其计算公式为：MSE = 1/n \* ∑(y\_i - ŷ\_i)^2，其中 y\_i 为实际值，ŷ\_i 为预测值，n 为样本数量。MSE 越小表示模型拟合效果越好。

2. 平均绝对误差（Mean Absolute Error，MAE）：MAE 表示预测值与真实值之差的绝对值的平均数，计算公式为：MAE = 1/n \* ∑|y\_i - ŷ\_i|，其中 y\_i 为真实值，ŷ\_i 为预测值， n 为样本数量。MAE 越小表示模型的拟合效果越好。

3. 均方根误差（Root Mean Squared Error，RMSE）：RMSE 是 MSE 的平方根。其计算公式为：RMSE = √MSE，其中 MSE 为均方误差。因为 RMSE 的值更加接近预测误差的实际大小，所以在有些情况下选择 RMSE 作为指标比 MSE 更好。

4. R2 得分（R-Squared，R2）：该评价指标用于判断模型对样本预测的合理性，其计算公式为：R² = 1 - SSres / SStot，其中 SSres 为残差平方和，SStot 为总离差平方和。R2 得分在 [0, 1] 区间内，R2 等于 1 表示模型的预测能力很好，R2 等于 0 则表示模型的预测能力很差。