openGauss AI特性创新实践课



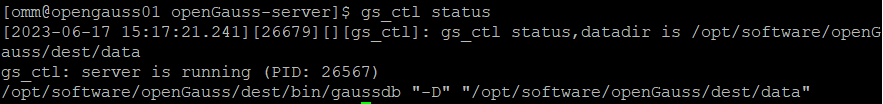
华为技术有限公司

# 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：

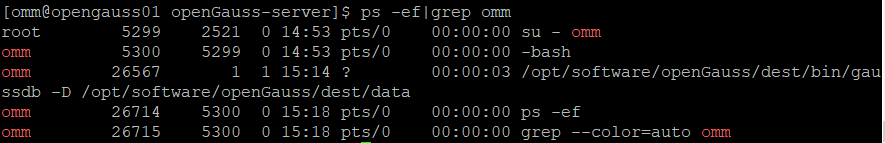
任务一：数据库状态验证

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？

通过源码编译和安装数据库主要有以下几个主要原因：

1、定制化配置：通过源码编译安装数据库，我们可以根据自己的需求进行定制化配置。我们可以选择要启用的功能和选项，以及优化数据库的性能和资源使用。这种灵活性使得我们能够根据特定的应用程序需求进行优化和调整。

2、最新版本：源码编译安装还可以确保我们安装的是最新版本的数据库。这对于希望获得最新功能、性能优化和安全修复的用户来说很重要。

3、平台适配性：源码编译使得我们能够针对特定的操作系统和硬件平台进行优化。数据库源代码通常会针对多种平台进行编写和维护，因此我们可以根据自己的需求和硬件架构选择合适的编译选项，以获得更好的性能和兼容性。

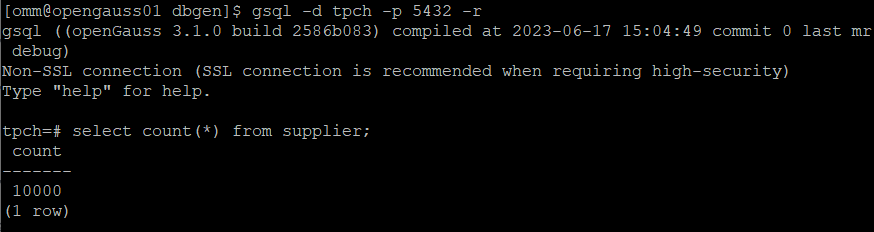
4、学习和定制开发：通过源码编译和安装数据库，我们可以深入了解数据库的内部工作原理和实现细节。这对于学习和理解数据库系统以及进行自定义开发和调试非常有帮助。以及我们可以通过查看数据库的源代码，研究其算法和数据结构，并根据需要进行修改和扩展。

# 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：

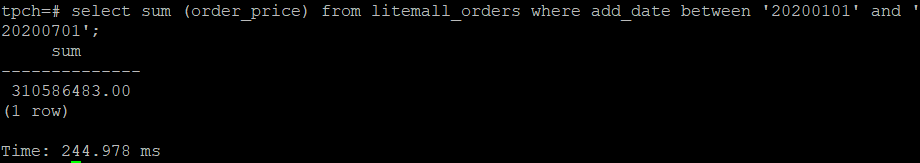
select count(\*) from supplier;;



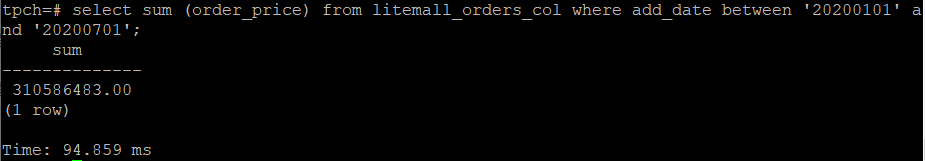
任务二：行存表与列存表执行效率对比

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';

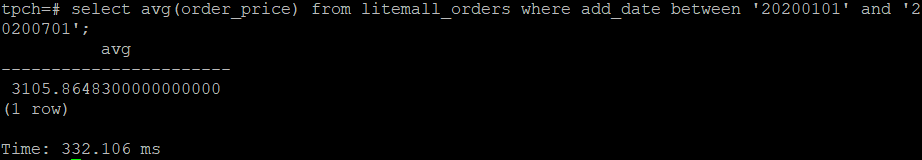


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

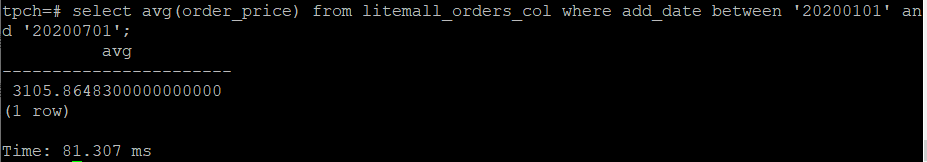


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

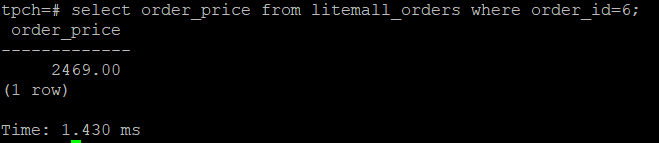


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

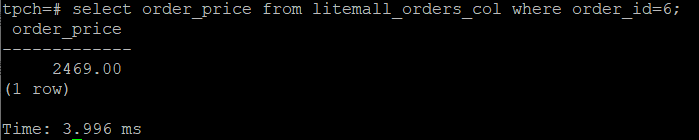


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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



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



1. 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

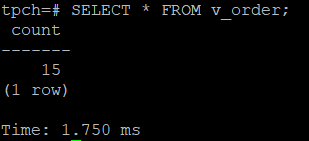
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



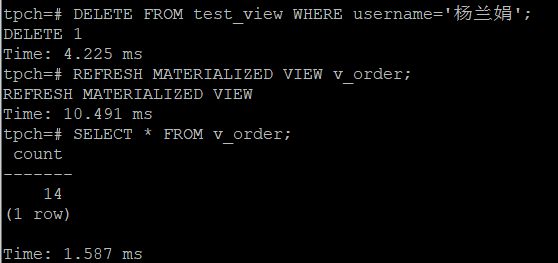
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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;



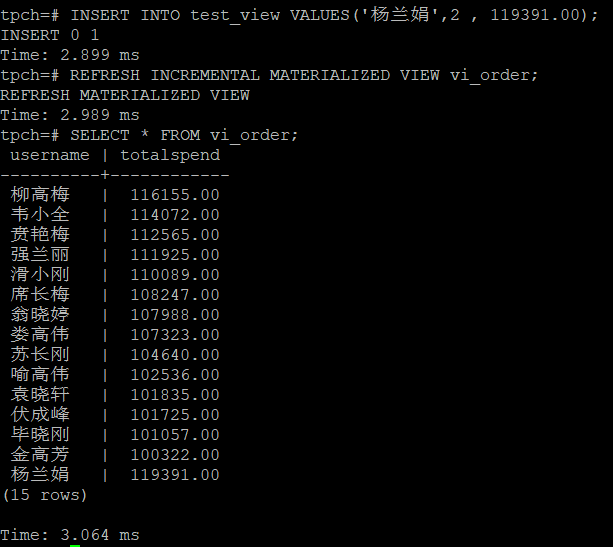
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;

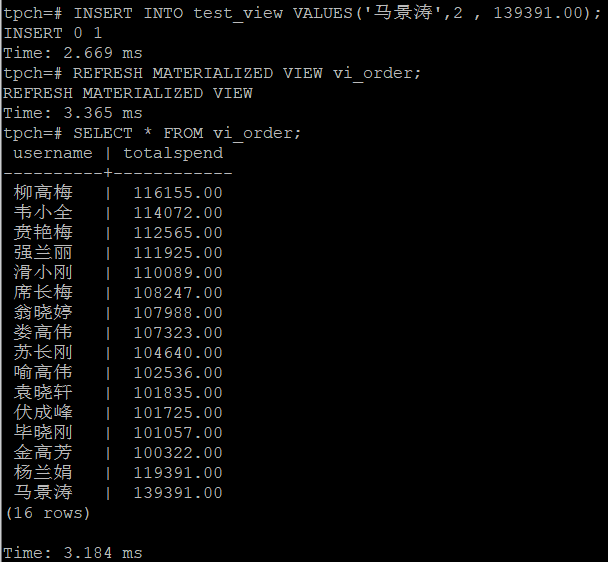


5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



再次插入数据、全量刷新物化视图、查询物化视图结果，奖执行结果截图



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？

行存表和列存表是在数据库中存储数据的不同方式，它们的存储结构和数据读取方式不同，因此执行相同的SQL语句时可能会导致不同的执行时间。

具体而言，在行存表中，数据按行存储在磁盘上，每一行包含一条完整的记录，包括所有列的数据。当执行SQL查询时，数据库需要读取整行数据，即使查询只需要其中一部分列的数据。这可能导致不必要的IO操作和数据传输，增加了查询的时间和资源开销。

而在列存表中，数据按列存储在磁盘上。每个列被存储为连续的数据块，类似于列的数据类型和值。当执行SQL查询时，数据库只需要读取相关列的数据块，而不需要读取整行数据。这样可以减少IO操作和数据传输量，提高查询性能。尤其在需要聚合函数、分析或筛选特定列的数据时，列存表通常表现更好。综上所述，执行相同的SQL语句时，会导致不同的执行时间。

对于需要读取整行数据或多个列的查询，行存表可能更快，因为它可以一次读取整行数据。而对于只需要读取特定列数据或进行聚合、分析操作的查询，列存表可能更快，因为它可以避免读取不必要的数据。

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在数据库中用于提高查询性能和数据分析的技术。它们之间的主要差别在于数据的刷新方式和更新频率。

全量物化视图是通过执行完整的数据抽取和计算过程来刷新的。当底层数据发生变化时，全量物化视图需要重新执行完整的抽取、计算和存储过程，以便保持与底层数据的一致性。这意味着全量物化视图的刷新过程可能需要较长的时间和较大的资源开销，特别是当底层数据量较大或计算复杂度较高时。

增量物化视图是通过仅计算和更新变化的部分数据来刷新的。当底层数据发生变化时，增量物化视图只需要计算和更新受到影响的数据部分，而无需重新执行完整的抽取和计算过程。这样可以大大减少刷新时间和资源消耗。增量物化视图通常使用增量变更日志或类似的技术来跟踪和捕捉底层数据的变化。

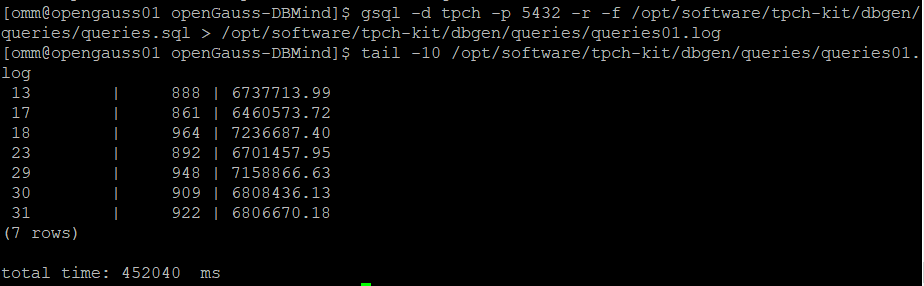
综上所述，全量物化视图和增量物化视图的主要差别在于刷新的方式和更新频率。全量物化视图适用于底层数据变化频率较低或计算复杂度较低的情况，而增量物化视图适用于底层数据变化频繁或计算复杂度较高的情况。增量物化视图可以更快地反映底层数据的变化，但可能需要更多的管理和维护成本，例如监控和处理增量变更日志。选择使用哪种类型的物化视图应该根据具体的业务需求、数据特征和性能要求来进行评估和决策。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

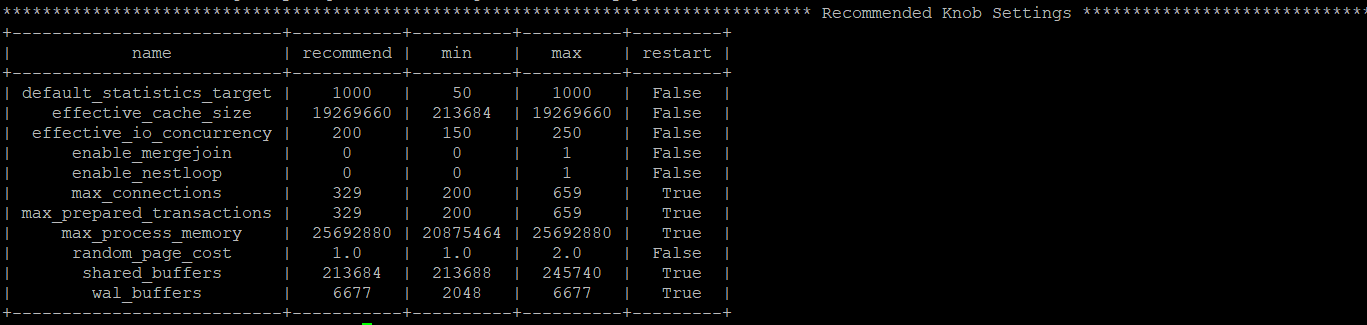
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

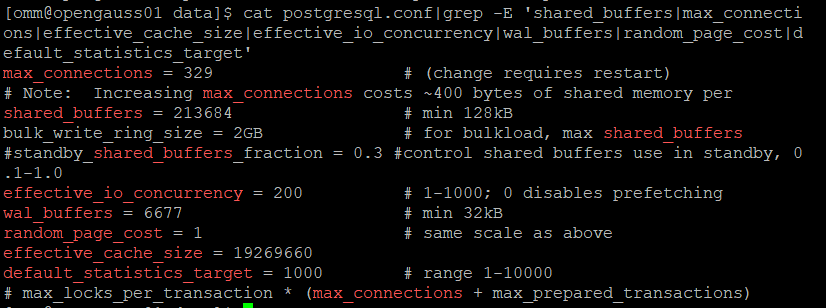
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

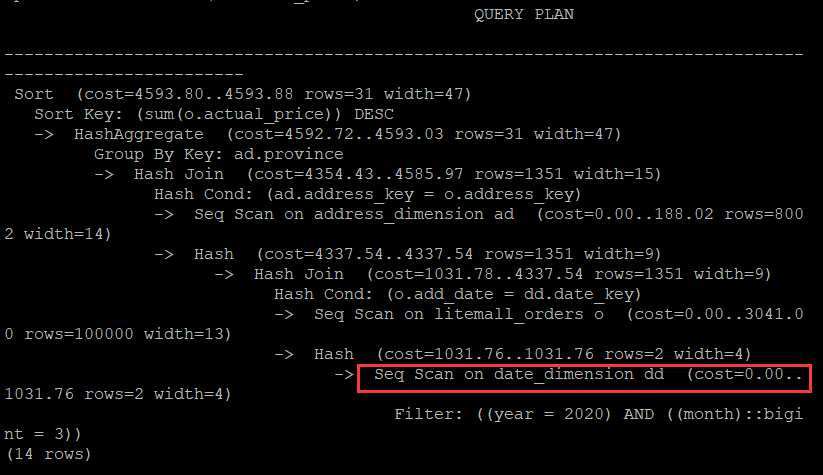
AND o.add\_date = dd.date\_key

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

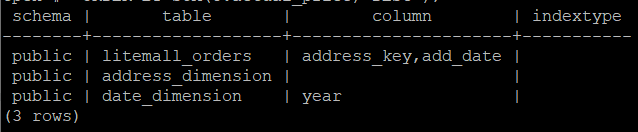
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

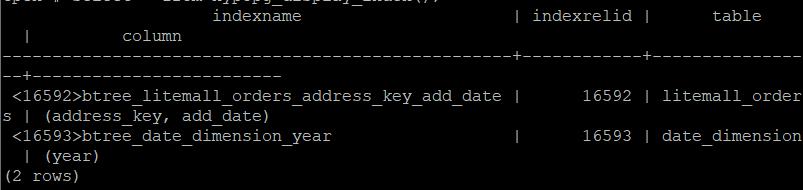
GROUP BY ad.province

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。

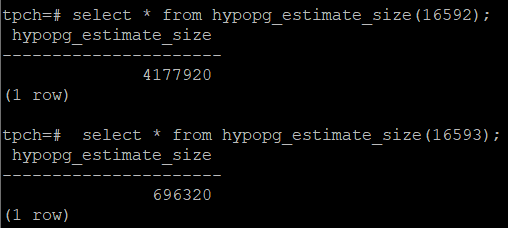
select \* from hypopg\_display\_index();



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(16592);

select \* from hypopg\_estimate\_size(16593);



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

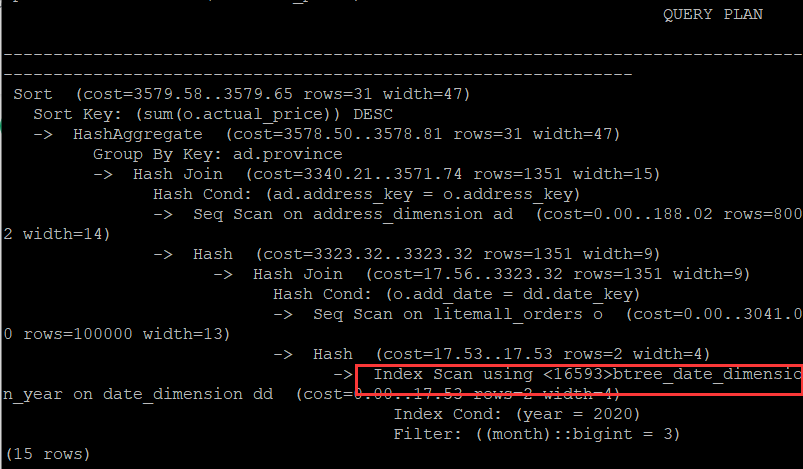
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

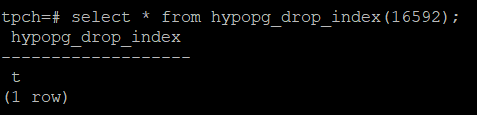
GROUP BY ad.province

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



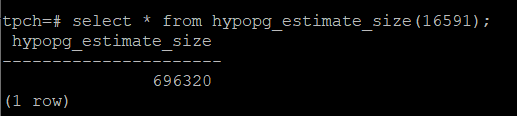
6. 删除某一个索引虚拟列，将执行结果截图。

select \* from hypopg\_drop\_index(16592);



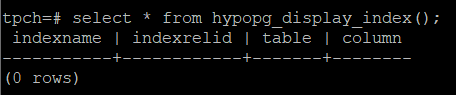
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

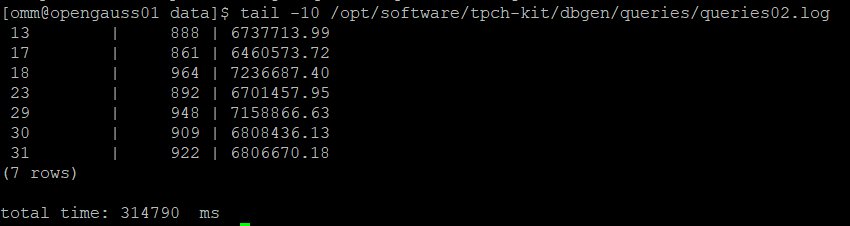
select \* from hypopg\_display\_index();



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：

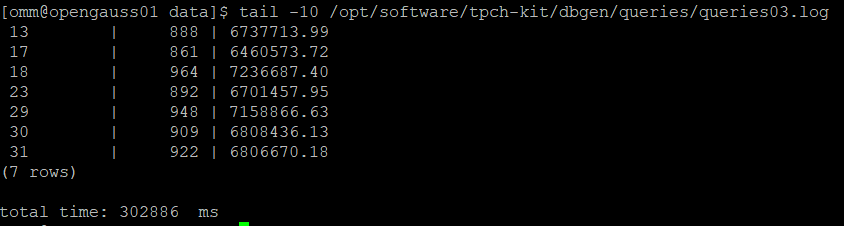
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

根据上述X-Tuner给出的参数优化可知，对shared\_buffers、max\_connections、effective\_cache\_size、effective\_io\_concurrency、wal\_buffers、random\_page\_cost、default\_statistics\_target等参数进行了优化，通过对这些参数进行优化，可以提高数据库系统的性能、响应时间和吞吐量，优化查询计划的选择，减少磁盘IO操作，提高数据访问的效率，以满足应用的需求和性能要求。

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？

索引在执行SQL查询时具有以下好处：

1. 提高查询性能：索引可以加快SQL查询的速度。通过在列上创建索引，数据库可以更快地定位和检索所需的数据，而无需扫描整个表。这可以减少磁盘IO操作和数据传输量，从而提高查询性能。
2. 减少数据排序和聚合的开销：对于需要排序或聚合操作的查询，索引可以减少排序和聚合的开销。数据库可以利用索引的有序性来避免对整个数据集进行排序或聚合操作，从而提高查询的效率。
3. 优化连接操作：当执行连接操作（JOIN）时，索引可以帮助减少数据集之间的比较和匹配操作的开销。通过在连接列上创建索引，数据库可以更快地定位和匹配连接条件，提高连接操作的性能。

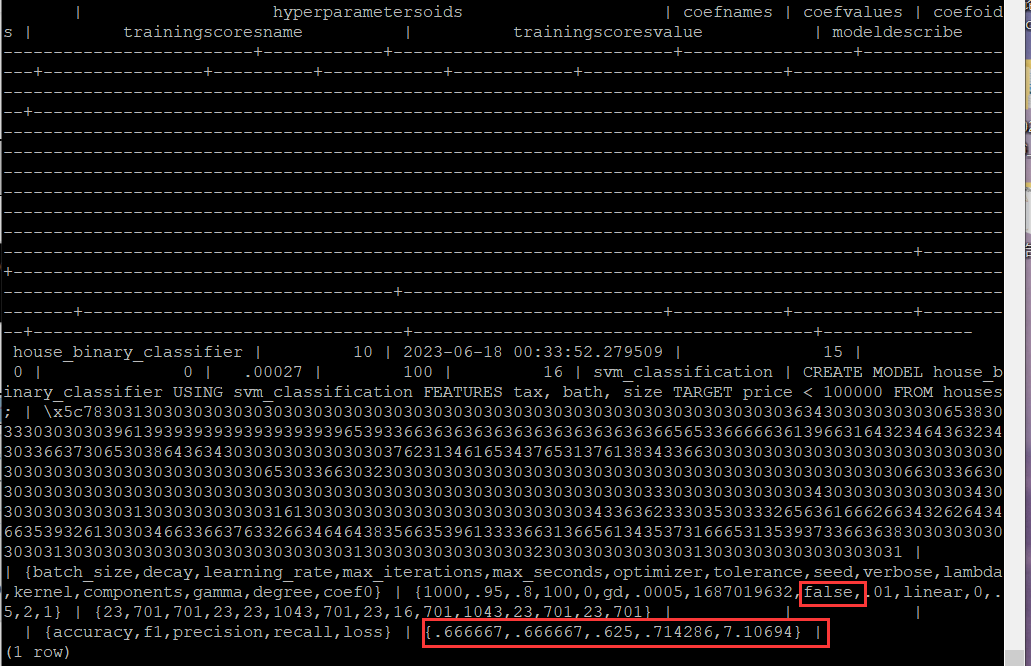
除了使用索引和参数，还可以通过以下几个方面对数据库进行优化：

1. 数据库设计：良好的数据库设计可以提高数据库的性能和可维护性。包括选择适当的数据类型、表之间的关系设计、范式化和去范式化的权衡等。
2. 查询优化：编写高效的查询语句可以减少数据库的负载和提高查询性能。包括选择合适的JOIN类型、使用合理的过滤条件、避免使用SELECT \* 等。
3. 分区和分片：对大型表进行分区或分片可以提高查询和维护的效率。这可以将数据划分为更小的逻辑或物理单元，从而减少查询范围和IO操作。
4. 缓存和缓存策略：使用缓存技术和合理的缓存策略可以减少对数据库的访问次数，提高数据的读取速度和响应性能。

# 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';

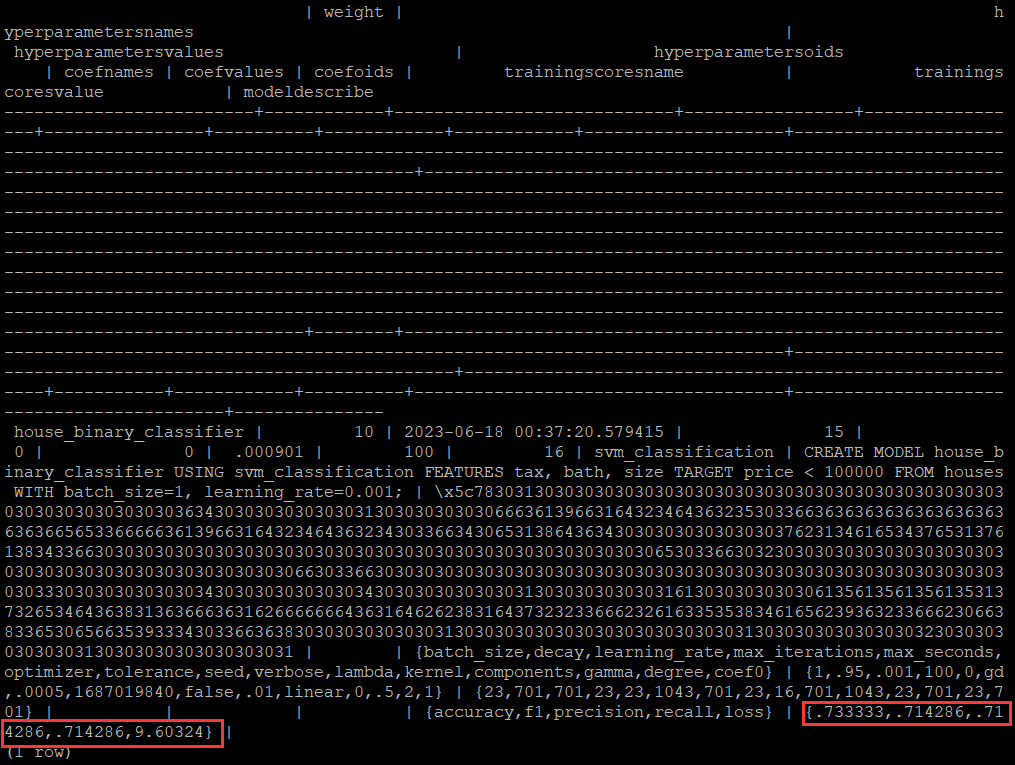


通过系统表的返回信息可以看到训练过程的预测准确率为 0.666667。

这对于二分类来说准确率并不高，我们后面将尝试通过超参数来提升模型表现。

任务二：观察新模型的信息，将执行结果截图。

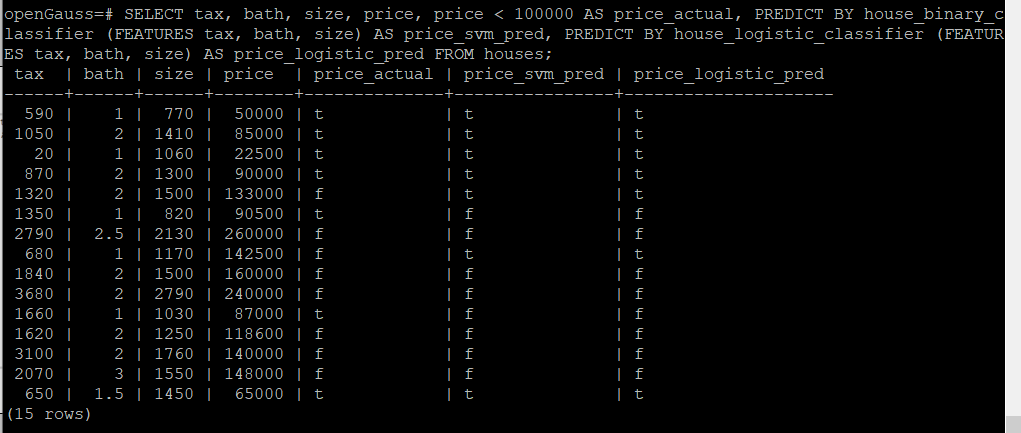
postgres=# SELECT \* FROM gs\_model\_warehouse WHERE modelname = 'house\_binary\_classifier';



整体准确率要相对默认超参数有所提升

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？

分类模型用于处理离散型的目标变量，通过将输入数据映射到预定义的离散类别中进行分类预测，输出为离散的类别标签。而回归模型用于处理连续型的目标变量，建立函数或模型来对输入数据进行预测，输出为连续的数值。因此，分类模型和回归模型在目标变量类型、输出类型、评估指标和模型选择等方面存在差异。

实践思考题2：什么是SVM算法？

SVM（Support Vector Machines，支持向量机）算法是一种监督学习算法，用于解决分类和回归问题。其基本思想是通过在特征空间中找到一个最优的超平面，将不同类别的样本点尽可能地分开。SVM算法通过最大化间隔来选择超平面，使得两个最靠近超平面的类别样本点之间的距离最大化。支持向量是离超平面最近的样本点，它们决定了超平面的位置和方向。SVM算法可以通过线性核函数进行线性分类，也可以通过非线性核函数进行非线性分类。在分类问题中，SVM算法的目标是找到一个最优的超平面来分隔不同类别的样本点，并对新样本进行分类预测。在回归问题中，SVM算法通过寻找一个最优的超平面来建立回归模型，预测连续型的目标变量。SVM算法具有处理高维特征空间、处理线性和非线性问题的能力，并且在具有少量支持向量的情况下仍能取得较好的性能。

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？

分类问题中常用的评价指标有准确率、召回率、精确率和F1分数。

1. 准确率（Accuracy）：准确率是指分类模型正确预测的样本数与总样本数之比。它衡量了分类模型整体的预测准确程度。准确率的计算公式为：准确率 = (预测正确的样本数) / (总样本数)。然而，准确率并不能反映出不同类别之间的不平衡性，当样本分布不均衡时，准确率可能会给出误导性的结果。
2. 召回率（Recall）：召回率是指分类模型正确预测为正类别的样本数与实际正类别样本数之比。它衡量了分类模型对正类别样本的识别能力。召回率的计算公式为：召回率 = (预测为正类别且实际为正类别的样本数) / (实际正类别样本数)。较高的召回率意味着模型能够更好地捕捉到正类别样本，但可能会导致误判负类别样本为正类别。
3. 精确率（Precision）：精确率是指分类模型正确预测为正类别的样本数与预测为正类别的样本数之比。它衡量了分类模型预测为正类别的准确程度。精确率的计算公式为：精确率 = (预测为正类别且实际为正类别的样本数) / (预测为正类别的样本数)。较高的精确率意味着模型预测为正类别的样本中有较少的错误。
4. F1分数（F1 Score）：F1分数是精确率和召回率的调和均值，综合了两者的评估指标。它能够平衡分类模型的准确性和召回性能。F1分数的计算公式为：F1分数 = 2 \* (精确率 \* 召回率) / (精确率 + 召回率)。F1分数的取值范围在0和1之间，值越接近1表示模型的综合性能越好。

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？

回归问题中常用的评价指标有均方误差、均方根误差、平均绝对误差和决定系数。

1. 均方误差（MSE）：均方误差是预测值与真实值之间差异的平方的平均值。它衡量了模型预测结果与实际值之间的平均误差的平方大小。MSE的计算公式为：，其中 是第个样本的真实值， 是预测值，是样本数量。MSE的值越小表示模型的预测结果与实际值越接近。
2. 均方根误差（RMSE）：均方根误差是均方误差的平方根。它衡量了模型预测结果与实际值之间的平均误差的平方根大小。RMSE的计算公式为：。RMSE与MSE具有相同的优势和缺点，但RMSE的单位与原始目标变量的单位相同，更易于解释和比较。
3. 平均绝对误差（MAE）：平均绝对误差是预测值与真实值之间差异的绝对值的平均值。它衡量了模型预测结果与实际值之间的平均绝对误差的大小。MAE的计算公式为：。MAE的值越小表示模型的预测结果与实际值越接近。
4. 决定系数（R²）：决定系数是用来衡量回归模型对目标变量的解释能力。它表示模型预测结果能够解释目标变量变异程度的比例。决定系数的取值范围在0和1之间，越接近1表示模型的预测能力越好。决定系数的计算公式为：，其中是第个样本的真实值，是预测值，是目标变量的均值。