openGauss AI特性创新实践课



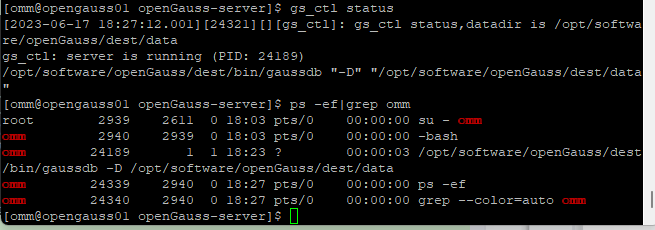
华为技术有限公司

# 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：

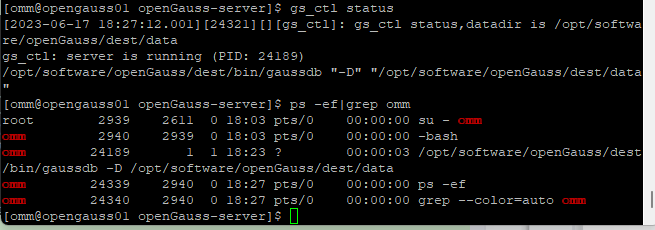
任务一：数据库状态验证

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？

定制化需求：使用源码编译可以根据实际需求进行自定义配置和优化。源码提供了灵活性，可以选择所需的功能、插件或模块，并进行编译安装。可以选择需要的功能模块，减少不必要的安装，同时也可以加入第三方扩展和插件。

最新版本：通过源码编译可以获取最新版本的数据库软件。官方发布的软件包可能会有一定的滞后，并且可能没有包含最新的修复程序、功能增强或安全更新。通过从源代码构建和安装数据库，可以获得最新的特性和修复。

兼容性和平台支持：源码编译可确保数据库软件与您的操作系统、硬件和其他软件组件之间的兼容性。通过自定义编译选项，可以更好地适配和优化数据库以适应特定的环境。

性能优化：通过自定义编译参数和优化选项，可以针对特定的硬件和工作负载优化数据库性能。例如，可以启用特定的优化指令集、调整内存分配策略或使用并行编译等方法来提高数据库的性能。同时，也可以避免一些由于软件包依赖错误而导致的问题。

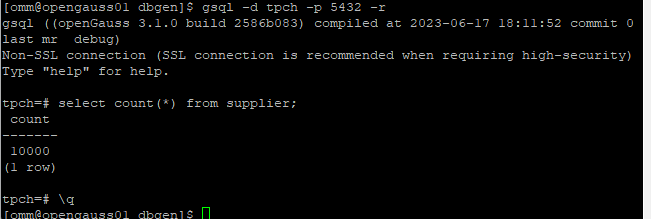
调试和故障排查：如果在使用官方发行版时遇到问题，通过从源代码编译和安装数据库，可以更容易地进行调试和故障排查。在编译过程中，可以启用调试符号和日志记录等功能来帮助定位和解决问题。

# 关卡二、openGauss数据导入及基本操作

任务一：数据初始化验证

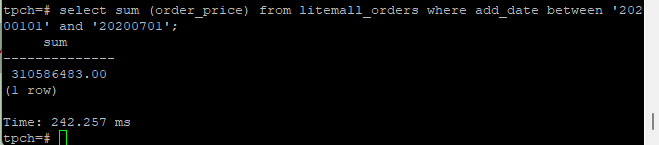
1. 查询supplier表的行数，并将结果进行图：

select count(\*) from supplier;

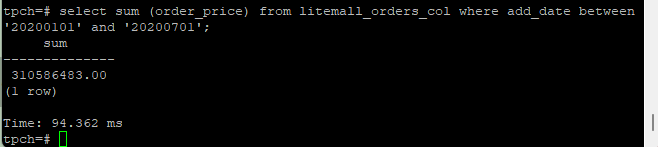


任务二：行存表与列存表执行效率对比

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

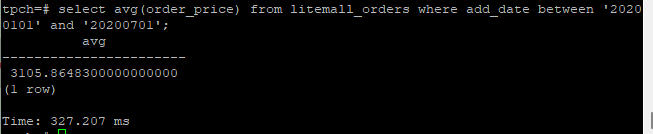
select sum (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701'; 

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

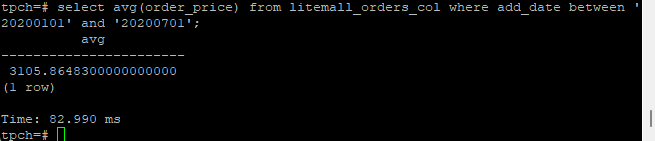


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

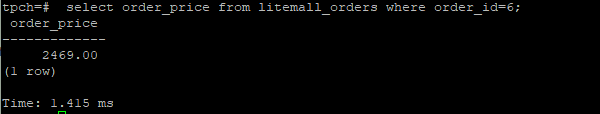


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

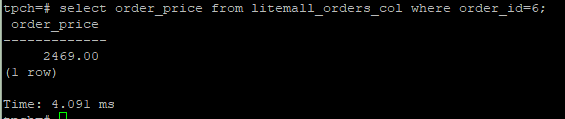


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

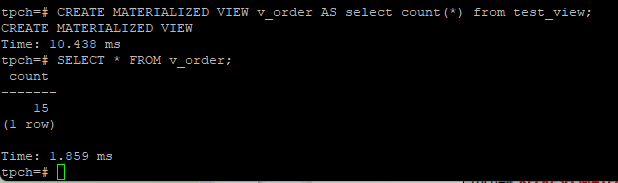
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



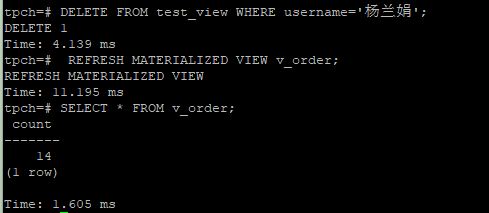
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



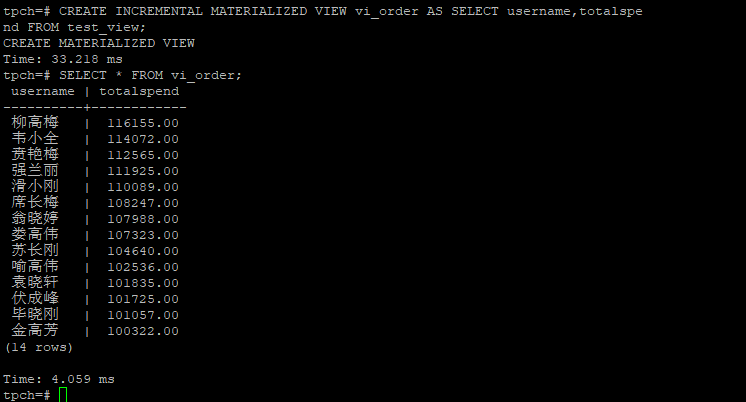
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



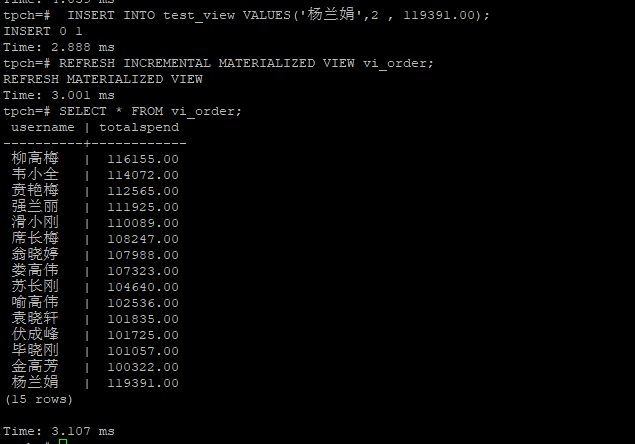
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

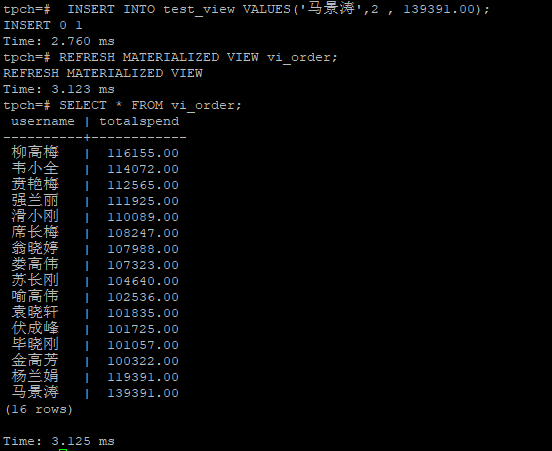
SELECT \* FROM vi\_order;



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主要是由于它们的内部组织结构和数据存储方式不同。

在执行哪些类型的SQL时，行存表效率更高：

当需要查询特定行的所有列数据时，行存表效率更高。因为行存表将整行数据作为一个单位存储，查询时可以直接获取到所需的数据。

当需要进行频繁的更新或插入操作时，行存表相对效率更高。因为更新或插入数据时，只需要修改或添加相应行中的数据即可，无需关注其他列。

在执行哪些类型的SQL时，列存表效率更高：

当需要查询特定列的数据时，列存表效率更高。因为列存表将同一列的数据连续存储，查询时只需读取所需的列数据，避免了读取不需要的列数据，节省了IO开销。

当涉及聚合查询、统计和分析类操作时，列存表效率更高。因为这些操作通常只需要针对特定列进行运算，而列存表的存储方式可以更有效地处理大量列数据的计算和聚合。

行存表：行存表是将每一行数据作为一个整体进行存储，因此在查询时需要扫描整个表，包括未使用的字段和行，这会导致查询效率较慢。行存表适用于以下类型的SQL：大量的INSERT、UPDATE和DELETE操作；需要频繁更新的OLTP（Online Transaction Processing）应用。

列存表：列存表是将每一列数据分别存储，因此在查询时只需要扫描需要的列，而不需要扫描整个表，这会大大提高查询效率。列存表适用于以下类型的SQL：大量的SELECT操作；需要聚合、计算和分析的OLAP（Online Analytical Processing）应用。

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两种不同的物化视图类型，它们在刷新机制和数据存储方式上有所区别。

全量物化视图：

刷新机制：全量物化视图在刷新时会重新计算和检索所有数据，将结果存储在物化视图中。这意味着每次刷新时都需要扫描整个源表或使用相应查询来获取最新数据。

数据存储方式：全量物化视图中保存的是完整的数据集，与源表的数据一一对应。每当物化视图刷新时，将源表的数据复制到物化视图中，并覆盖原有数据。

增量物化视图：

刷新机制：增量物化视图仅计算并存储源表中发生变化的部分数据，而不是重新计算所有数据。它会跟踪源表的变化，并根据变化情况更新物化视图。

数据存储方式：增量物化视图中保存的是增量数据，即源表中发生变化的部分，或者是在特定时间段内新增、修改或删除的数据。

主要差别：

刷新方式：全量物化视图需要重新计算和检索所有数据，而增量物化视图只需要处理源表中发生变化的部分数据，减少了计算量和IO开销。

数据存储与更新：全量物化视图保存完整的数据集，并覆盖原有数据；增量物化视图只存储变化的部分数据，并根据源表的变化进行更新。

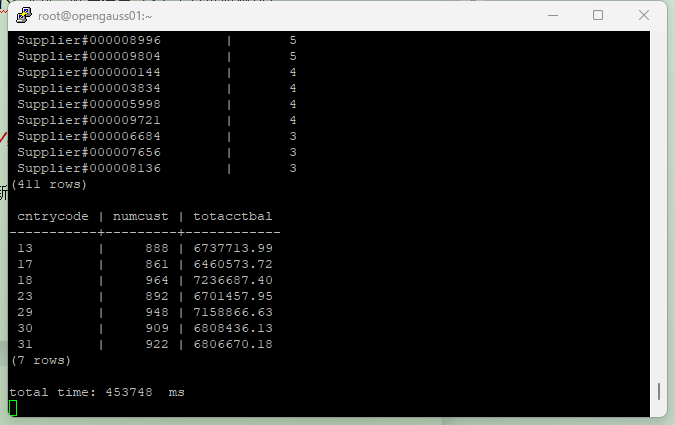
# 关卡三、openGauss的AI4DB特性应用

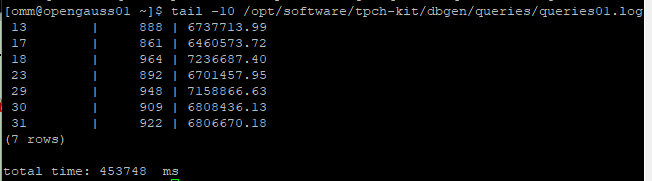
任务一：使用X-Tuner进行参数优化

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

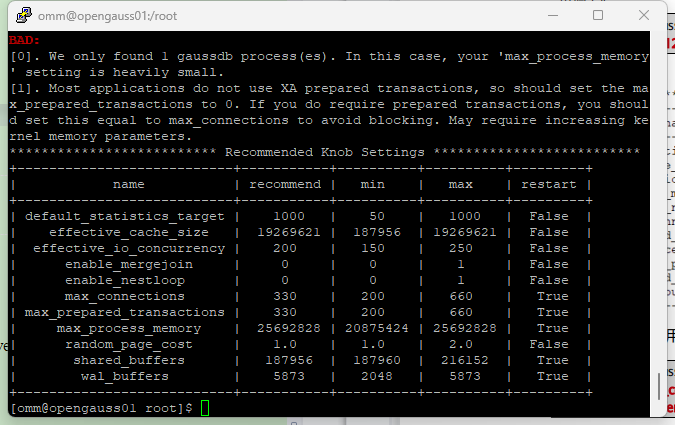






2. 使用root用户，执行X-Tuner进行参数建议优化，将执行结果截图

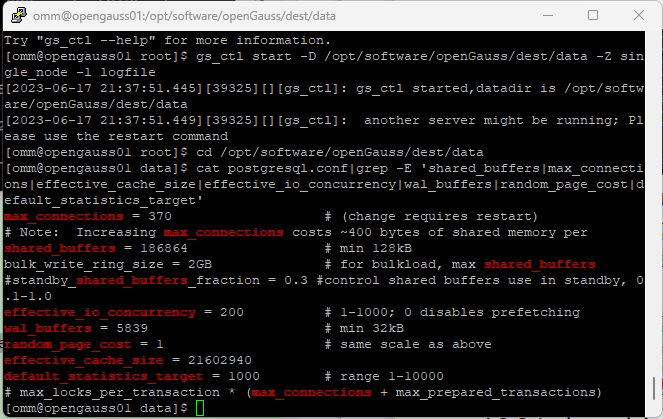
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

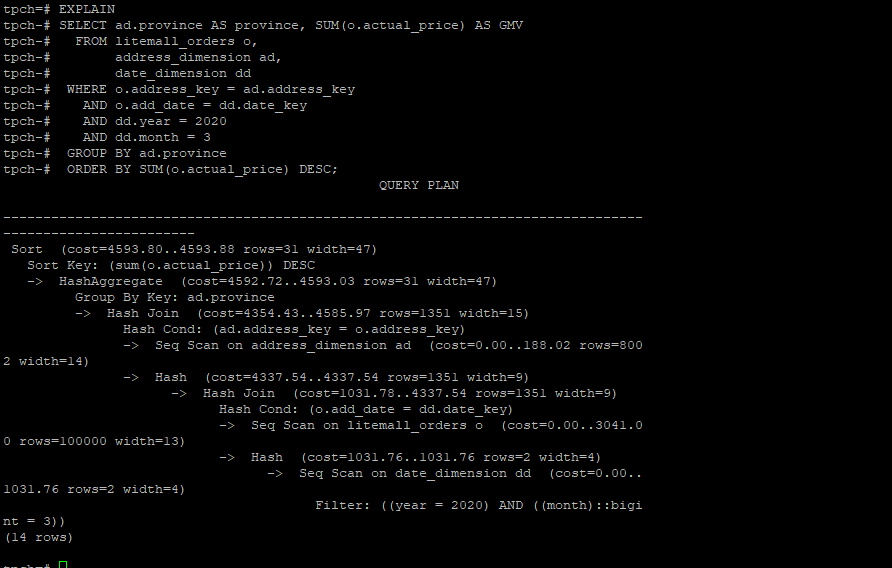
AND o.add\_date = dd.date\_key

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

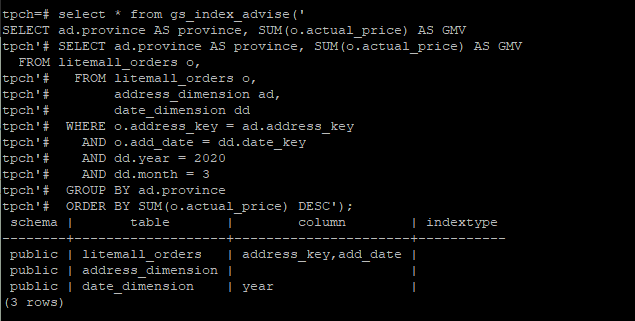
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

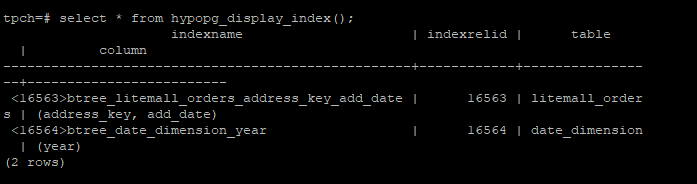
GROUP BY ad.province

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。

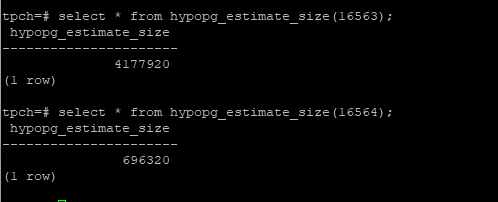
select \* from hypopg\_display\_index();



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(16563);

select \* from hypopg\_estimate\_size(16564);



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

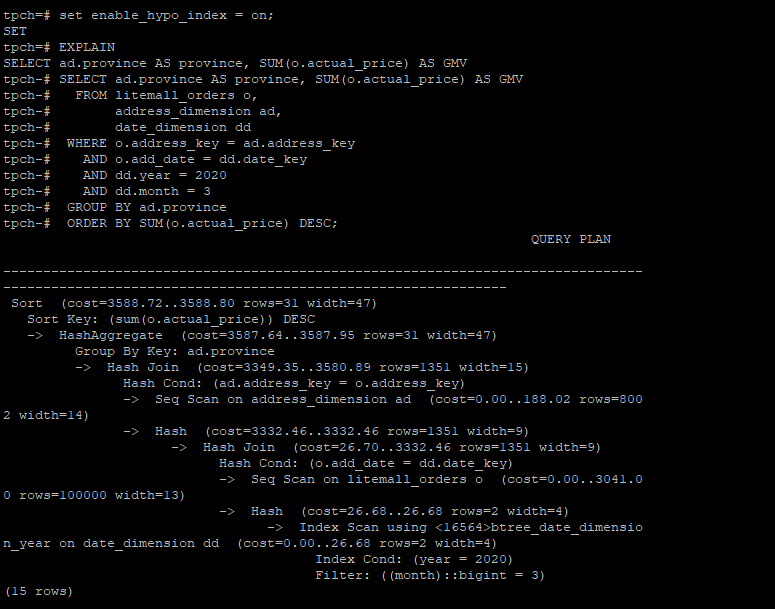
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

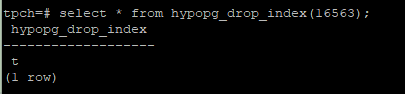
GROUP BY ad.province

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



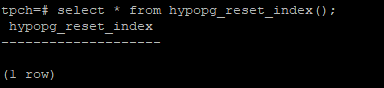
6. 删除某一个索引虚拟列，将执行结果截图。

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



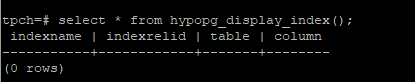
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

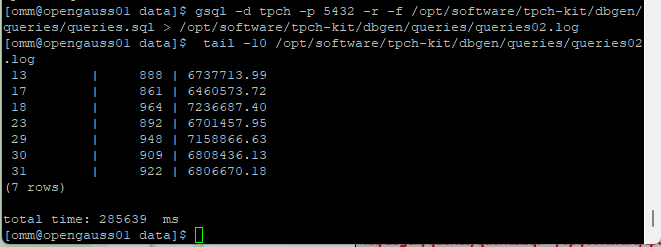
select \* from hypopg\_display\_index();



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

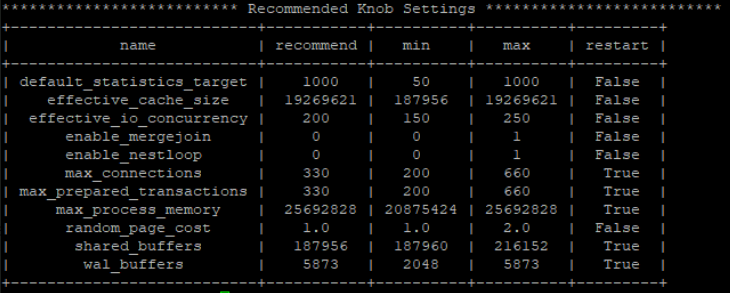


挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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？



"shared\_buffers = 186864"

"max\_connections = 370"

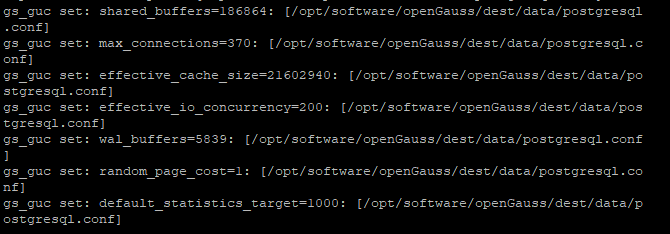
"effective\_cache\_size = 21602940"

"effective\_io\_concurrency = 200"

"wal\_buffers = 5839"

"random\_page\_cost = 1"

"default\_statistics\_target = 1000"



这些参数的优化可以提高数据库的性能和稳定性。对于opengauss数据库，进行参数优化的目的是为了提高数据库的性能和效率。下面逐个解释每个参数的作用及优化原因：

shared\_buffers：该参数指定了系统中用于缓存数据页的内存大小。增加shared\_buffers可以提高内存缓存的命中率，减少磁盘IO操作，从而提升查询性能。

max\_connections：该参数指定了数据库可以接受的最大并发连接数。适当调整max\_connections可以根据实际需求控制数据库连接数量，避免过多的连接导致资源竞争和性能下降。

effective\_cache\_size：该参数指定了查询规划器估计的系统可用内存大小。增大effective\_cache\_size可以使查询规划器更加倾向于使用索引扫描而不是磁盘IO操作，提高查询性能。

effective\_io\_concurrency：该参数指定了系统对IO请求的并发处理能力。增大effective\_io\_concurrency可以提高文件系统或存储设备并行处理IO请求的能力，从而加快IO操作的速度。

wal\_buffers：该参数指定了写入日志缓冲区的大小。增加wal\_buffers可以提高写入日志的性能，并减少写日志时的磁盘IO操作。

random\_page\_cost：该参数指定了随机访问磁盘页的代价。适当调整random\_page\_cost可以影响查询规划器对索引扫描和顺序扫描的选择，提高查询性能。

default\_statistics\_target：该参数指定了自动生成统计信息时的样本大小。增加default\_statistics\_target可以更准确地估计表中数据的分布情况，优化查询规划和执行。

通过调整这些关键参数，可以根据系统配置和实际业务需求来优化opengauss数据库的性能，提升查询速度和系统的整体吞吐量。

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？

使用索引可以对执行SQL语句带来多方面的好处：

提高查询性能：索引可以加快数据检索速度，减少数据扫描的开销。通过在关键列上创建索引，数据库可以直接定位到符合条件的数据，避免全表扫描，从而提升查询效率。

优化排序和分组操作：如果SQL语句包含ORDER BY或GROUP BY等操作，索引可以提供有序的数据访问，降低排序和分组的开销。

加速连接操作：对于JOIN操作，通过在连接列上创建索引，可以加快表之间的数据匹配速度，提升连接查询的性能。

减少IO访问：索引可以减少磁盘IO的次数，从而减轻系统负载，并缩短查询执行时间。

除了使用索引和调整参数外，还可以采取以下措施来优化数据库性能：

合理设计数据模型：优化数据表结构、字段类型和关系，减少重复数据和冗余字段，使数据库模型更精简和高效。

针对具体业务场景进行数据库规划：根据应用需求和数据量估算，选择合适的硬件配置、存储方案和数据分区策略。

数据库连接管理：适当控制并发连接数量，避免连接池过大造成资源浪费或过小导致请求堆积。

查询优化：编写高效的SQL语句，避免不必要的数据检索、排序和聚合操作。分析查询执行计划，使用数据库提供的性能分析工具进行调优。

缓存管理：合理使用数据库缓存技术，如查询缓存、结果集缓存等，减少磁盘IO操作。

监控和调优：定期监控数据库性能和资源利用情况，识别潜在瓶颈和性能问题，并及时采取优化措施。

# 关卡四、openGauss的DB4AI特性应用

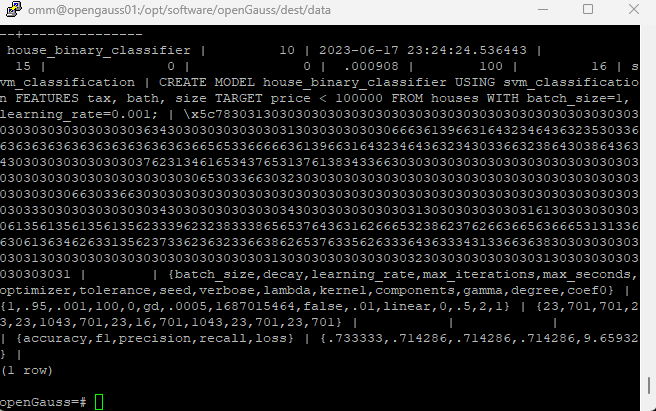
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



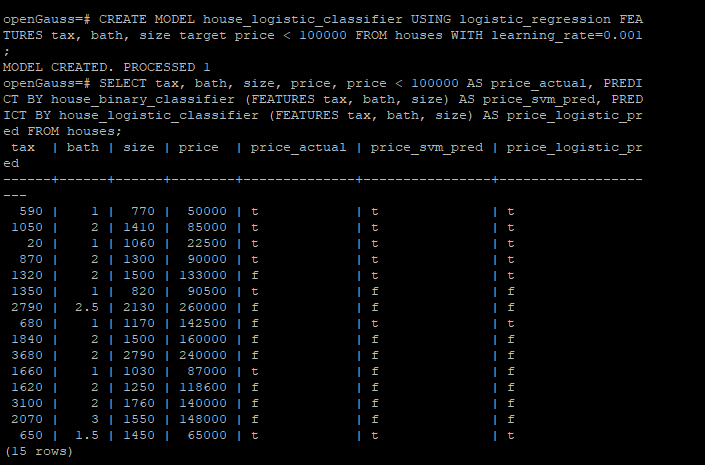
任务二：观察新模型的信息，将执行结果截图。

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是机器学习中两种常见的预测模型，它们在应用和特点上存在一些不同之处：

目标变量类型：

分类模型：适用于离散型目标变量，即将实例分到不同的类别或标签中。例如，预测一个电子邮件是否为垃圾邮件。

回归模型：适用于连续型目标变量，即对实例进行数值预测。例如，预测房屋价格。

输出结果：

分类模型：输出预测结果为离散的类别或标签，如二元分类（正/负）或多类分类（A/B/C）。

回归模型：输出预测结果为连续的数值，可以是任意范围的实数。

损失函数：

分类模型：常见的损失函数包括交叉熵损失函数（Cross Entropy Loss）、对数损失函数等，用于衡量分类结果的错误程度。

回归模型：常见的损失函数包括均方误差（Mean Squared Error）、均绝对误差（Mean Absolute Error）等，用于衡量回归结果与真实值之间的差异。

模型选择：

分类模型：通常使用诸如逻辑回归（Logistic Regression）、决策树（Decision Tree）、支持向量机（Support Vector Machine）和神经网络（Neural Network）等算法来构建分类模型。

回归模型：常用的回归算法包括线性回归（Linear Regression）、岭回归（Ridge Regression）、Lasso回归（Lasso Regression）以及决策树回归（Decision Tree Regression）等。

需要根据实际问题的特点和目标变量的类型选择合适的模型。如果目标是进行类别预测，则使用分类模型；如果目标是进行数值预测，则选择回归模型。

实践思考题2：什么是SVM算法？

SVM算法是一种常用的监督学习算法，主要用于分类和回归任务。它在数据集中找到一个最优的超平面或者子空间，将不同类别的数据样本尽可能地分开。

SVM算法的基本思想是将数据映射到高维空间中，使得数据在新的空间中能够线性可分。然后在这个高维空间中寻找一个最优的超平面，也就是能够将不同类别的样本分隔开的平面。该超平面被选择为能够最大化训练数据与超平面之间的最小间隔（即支持向量），使得模型具有更好的泛化能力。

SVM算法的关键思想包括以下几点：

1. 核函数：通过 kernel trick（核技巧），可以避免在高维空间中进行实际的计算，通过定义合适的核函数，将样本直接映射到高维特征空间中。
2. 支持向量：在训练过程中，SVM选择支持向量作为决策边界的关键点，这些点是离超平面最近的样本点。
3. 软间隔和惩罚项：为了处理存在噪声或异常点的数据集，SVM允许一定程度的错分类，引入惩罚项来平衡间隔最大化和错误分类之间的权衡。

SVM算法在实际应用中具有广泛的适用性，它可以处理线性可分或近似线性可分的问题，并且通过选取不同的核函数，可扩展到非线性问题。此外，SVM还具有对维度灾难的抵抗能力和较好的泛化性能。

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？

分类问题的评价指标有以下几种：

准确率（Accuracy）：准确率是最常用的评价指标之一，表示模型预测正确的样本数量与总样本数之间的比例。准确率越高，说明模型的预测结果与实际情况越接近。

精确率（Precision）：精确率衡量了模型在预测为正例的样本中，真正正例的比例。它关注模型的假阳性（将负例预测为正例）误判情况。精确率越高，模型在预测为正例的样本中，真正是正例的比例就越高。

召回率（Recall）：召回率度量了模型在所有真正正例样本中，成功预测为正例的比例。它关注模型的漏报（将正例预测为负例）误判情况。召回率越高，模型对于真正正例的识别能力就越强。

F1值（F1-Score）：F1值是精确率和召回率的调和均值，综合考虑了模型的准确性和完整性。F1值越高，说明模型相对于精确率和召回率都有较好的表现。

ROC曲线与AUC值（Receiver Operating Characteristic Curve and Area Under the Curve）：ROC曲线是以真阳性率（TPR）为纵坐标，假阳性率（FPR）为横坐标绘制的曲线。它可以描述模型在不同阈值下的分类能力。AUC值表示ROC曲线下的面积，范围在0.5到1之间，AUC值越大，说明模型分类能力越好。

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？

回归问题的评价指标主要有以下几种：

均方误差（Mean Squared Error，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真实值的整体偏离程度，数值越小表示模型拟合效果越好。

均方根误差（Root Mean Squared Error，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同。RMSE一般用于对误差进行解释和比较，数值越小表示模型预测的准确性越高。

平均绝对误差（Mean Absolute Error，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真实值的平均偏离程度，数值越小表示模型的预测准确性越高。

决定系数（Coefficient of Determination，R²）：决定系数是通过计算预测值与真实值之间的变异比例来衡量模型的拟合优度。它的取值范围为0到1，越接近1表示模型对目标变量的解释能力越强。