openGauss AI特性创新实践课



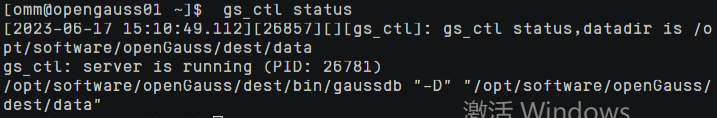
华为技术有限公司

# 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：

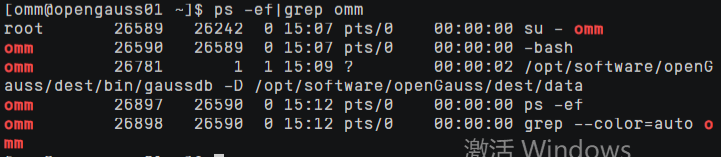
任务一：数据库状态验证

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？

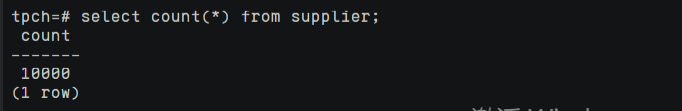
1. 自定义配置：通过源码编译和安装，可以根据特定的需求和环境进行自定义配置。可以选择适合的硬件和操作系统的编译选项、优化参数和扩展功能，以获得更好的性能和适应性。
2. 平台适配性：源码编译可以确保数据库系统能够适配您所使用的操作系统和硬件平台。通过源码编译，可以针对特定的操作系统版本和硬件架构进行优化，提高数据库在特定平台上的性能和稳定性。
3. 安全性和可验证性：源码编译可以提高安全性和可验证性。可以审查和验证编译过程中所使用的源代码，以确保代码的完整性和安全性。此外，通过自行编译和安装，您可以选择特定的安全选项和加密算法，以满足您对数据安全的需求。
4. 最新功能和修复：通过源码编译和安装，可以获得最新的功能和修复。可以选择最新的稳定版本或开发版本，以获得新功能、性能改进和错误修复，同时也可以参与到社区的开发和贡献中。

# 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：

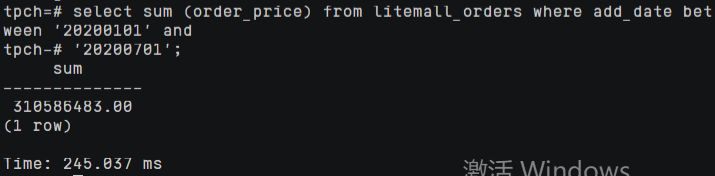
select count(\*) from supplier;;



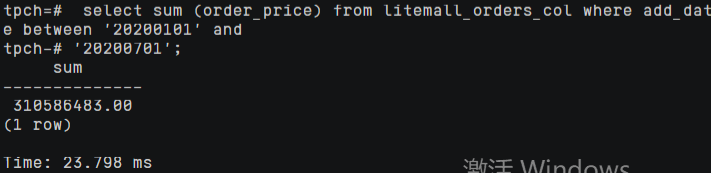
任务二：行存表与列存表执行效率对比

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';

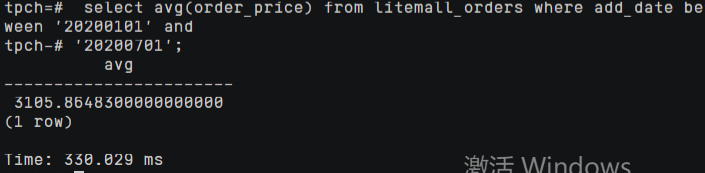


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

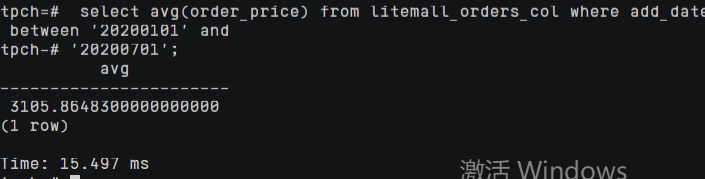


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

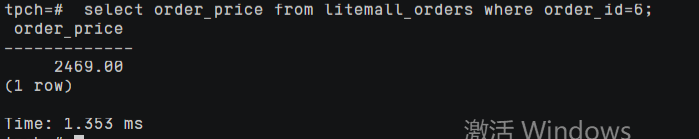


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

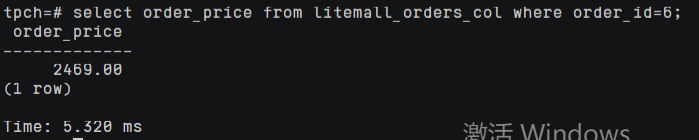


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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



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

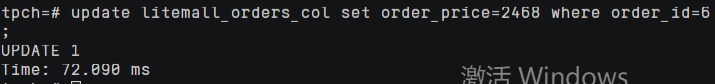


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



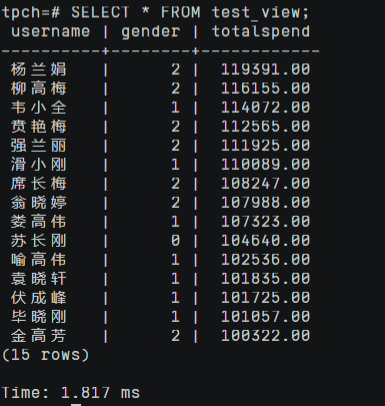
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

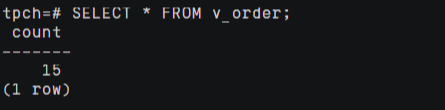
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



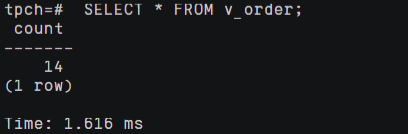
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



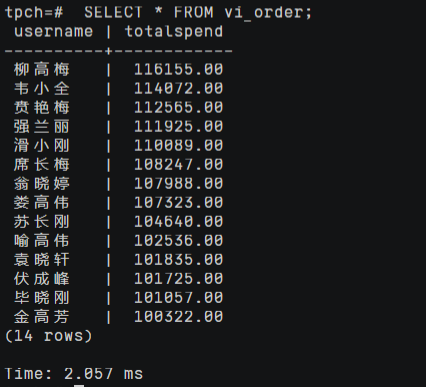
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



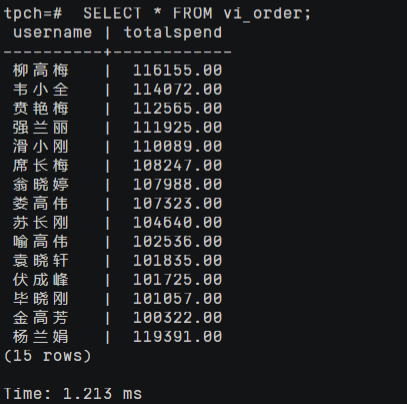
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

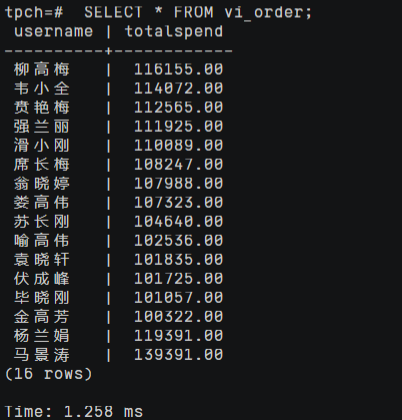
SELECT \* FROM vi\_order;



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数据存储方式不同以及对不同类型的查询操作有不同的优化效果。

行存表是按行存储数据的，每一行的数据存储在一起，包含了所有的列。当执行基于特定行的查询时，行存表的效率较高，因为可以直接读取所需的行数据。适用于如下类型的 SQL 查询：

1.单条记录查询：当只需要获取单个或少量记录时，行存表可以更快地找到并返回这些记录。

2.随机访问：如果查询需要在表中随机访问不同的行，行存表可以更有效地满足这种需求。

3.更新操作：行存表在执行更新操作时较为高效，因为只需要修改相应行的数据。

列存表是按列存储数据的，将每一列的数据存储在一起。当执行聚合、统计和分析类的查询时，列存表的效率较高，因为可以只读取需要的列数据。适用于如下类型的 SQL 查询：

1.聚合查询：当需要对某些列进行聚合计算（例如 SUM、COUNT、AVG 等）时，列存表可以仅读取相关列的数据，提高计算效率。

2.大规模数据分析：当处理大规模数据集进行复杂分析时，列存表的优势更加明显，因为只需要加载需要的列数据，减少了 I/O 开销。

3.压缩和存储效率：列存表通常使用压缩算法来减少存储空间，特别适合处理大量冗余数据的查询。

**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类型的物化视图，它们在数据更新和查询更新的方式上存在差异。

全量物化视图（Materialized View）：

1.全量物化视图是基于源表的全部数据创建的静态快照。

2.当源表数据发生变化时，需要重新刷新全量物化视图，即重新执行视图的完整计算过程。

3.刷新全量物化视图可能需要耗费较多的时间和资源，特别是在源表数据量较大时。

4.查询全量物化视图时，直接从物化视图中获取数据，不需要访问源表，可以提供较快的查询性能。

5.适用于源表更新较少的情况，查询操作频繁且对实时性要求不高的场景。

增量物化视图（Incremental Materialized View）：

增量物化视图只存储源表数据的部分更新信息，通常是存储源表数据的变化量或增量。

1.当源表数据发生变化时，只需要更新增量物化视图中的相关数据，而不需要重新计算整个视图。

2.更新增量物化视图的过程相对较快，只需处理变化量或增量数据。

3.查询增量物化视图时，可能需要结合源表数据进行计算，以获取最新的结果。

4.适用于源表频繁更新的情况，能够提供较好的实时性和性能。

需要注意的是，增量物化视图的实现通常较为复杂，需要维护变化量或增量数据的一致性，并且对源表的更新操作有一定的限制。选择全量物化视图还是增量物化视图取决于具体的业务需求、数据更新频率和查询性能要求。

# 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

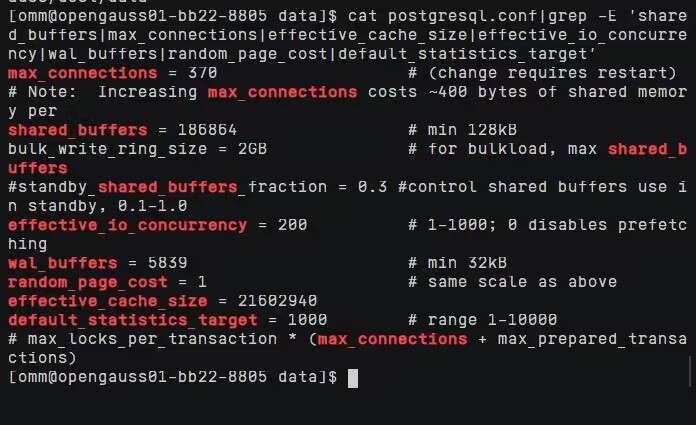
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

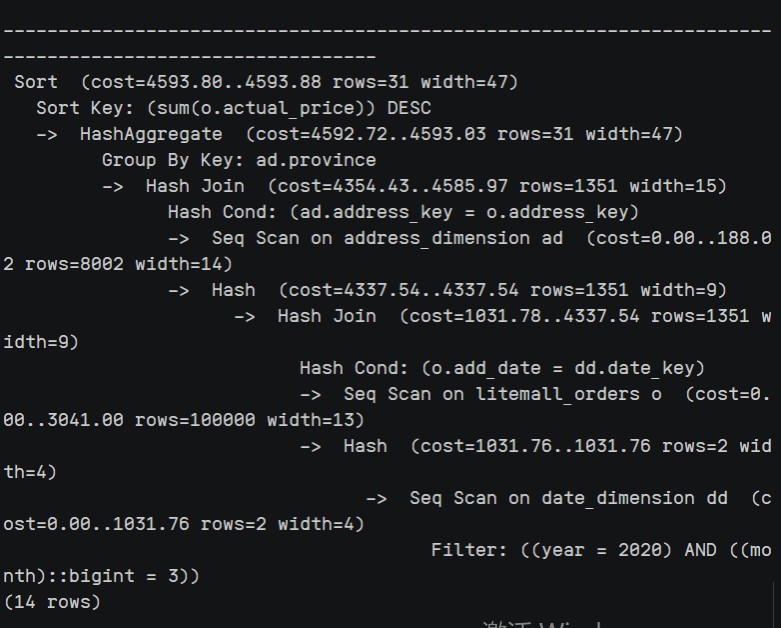
AND o.add\_date = dd.date\_key

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

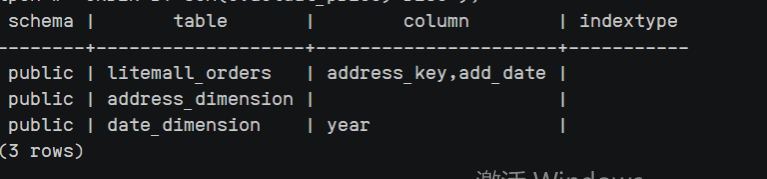
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

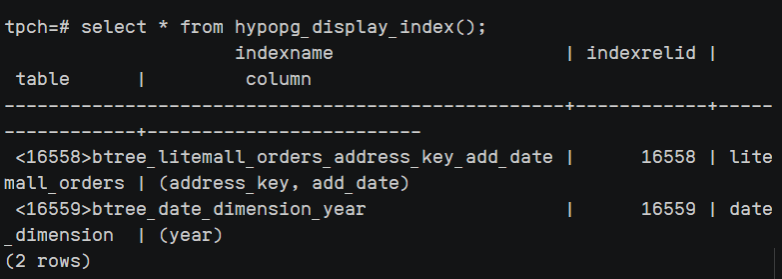
GROUP BY ad.province

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。

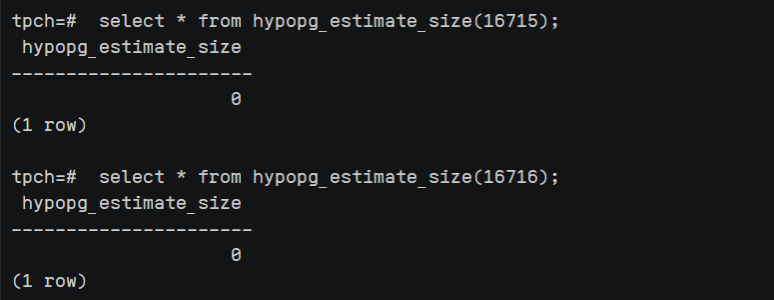
select \* from hypopg\_display\_index();



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

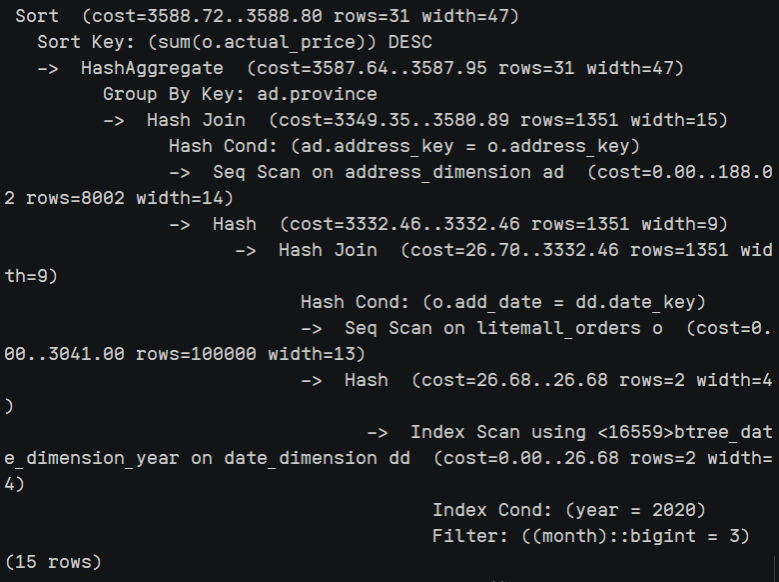
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

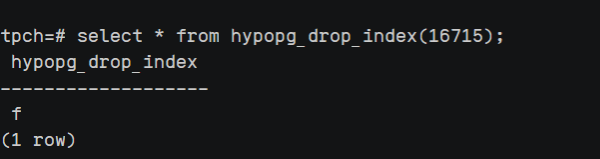
GROUP BY ad.province

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



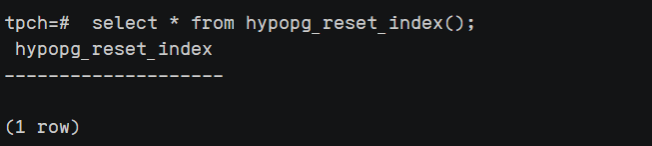
6. 删除某一个索引虚拟列，将执行结果截图。

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



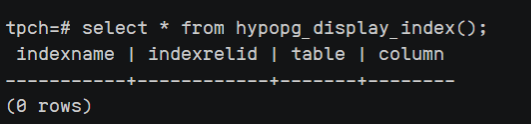
7. 删除某一个索引虚拟列，将执行结果截图。

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？

1.内存相关参数：包括shared\_buffers、wal\_buffers、effective\_cache\_size等。这些参数控制着数据库在内存中的使用情况，通过合理调整这些参数可以提高查询的性能，减少磁盘IO访问。

2.并发相关参数：包括max\_connections、effective\_io\_concurrency等。这些参数控制着数据库并发连接和并行执行的能力，通过优化这些参数可以提高数据库的并发性能和吞吐量。

3.查询优化参数：包括random\_page\_cost等。这些参数影响查询优化器的成本估算和查询计划的选择，通过调整这些参数可以改善查询性能和执行计划的质量。

为何要对这些参数进行优化的原因如下：

1.提高性能：通过合理调整参数，可以提高数据库的性能，包括查询的响应时间、并发处理能力和系统吞吐量等。

2.最佳适应环境：不同的数据库应用场景和硬件环境对于参数的要求不同，通过优化参数可以使数据库在特定的环境下达到最佳的性能表现。

3.资源利用优化：合理的参数设置可以优化数据库对于内存、磁盘和CPU等硬件资源的利用，提高资源利用效率，降低系统负载。

4.解决特定问题：某些参数的调整可以解决特定的性能问题，如内存相关参数的优化可以减少磁盘IO，查询优化参数的调整可以改善查询性能。

综上所述，通过对数据库的参数进行优化，可以提高数据库的性能和吞吐量，适应不同的应用场景和硬件环境，并解决特定的性能问题，从而实现更高效的数据库运行。

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？

使用索引可以带来以下好处对于执行SQL的好处：

1.提高查询性能：索引可以加快数据检索速度，通过使用索引，数据库可以快速定位到符合查询条件的数据，避免全表扫描。

2.减少IO访问：索引可以减少磁盘IO访问，因为数据库引擎可以通过索引直接访问存储在磁盘上的数据页，而不是扫描整个表。

3.提高排序和聚合操作的性能：对于排序和聚合操作，索引可以提供已经排序或预计算的结果，减少排序和聚合的计算开销。

除了使用索引和参数外，还可以通过以下方面对数据库进行优化：

1.数据库设计优化：良好的数据库设计可以提高数据库的性能。包括合理的表结构设计、范式化设计、适当的数据类型选择等。

2.查询语句优化：优化查询语句可以改善数据库的性能。包括选择合适的查询方式、避免全表扫描、合理使用JOIN操作、避免不必要的子查询等。

3.硬件和系统优化：通过升级硬件设备、优化操作系统和数据库配置等手段，提高数据库服务器的性能和稳定性。

4.内存管理优化：合理配置数据库的内存参数，增加内存缓存的使用，提高数据的访问速度。

5.数据库分区和分表：对于大型数据库，可以考虑使用分区和分表技术，将数据分散存储在多个物理位置，提高查询和维护的效率。

6.定期维护和统计信息更新：定期执行数据库的维护任务，包括索引重建、统计信息更新、碎片整理等，保持数据库的高效运行。

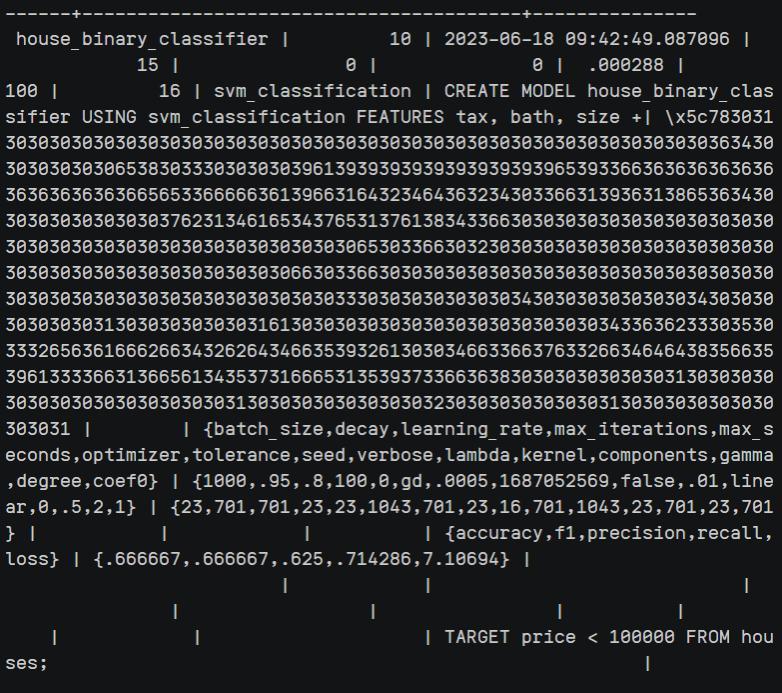
7.数据库连接池管理：合理管理数据库连接池，避免连接资源的浪费和性能下降。

8.数据库备份和恢复策略：建立合理的数据库备份和恢复策略，确保数据的安全性和可恢复性。

# 关卡四、openGauss的DB4AI特性应用

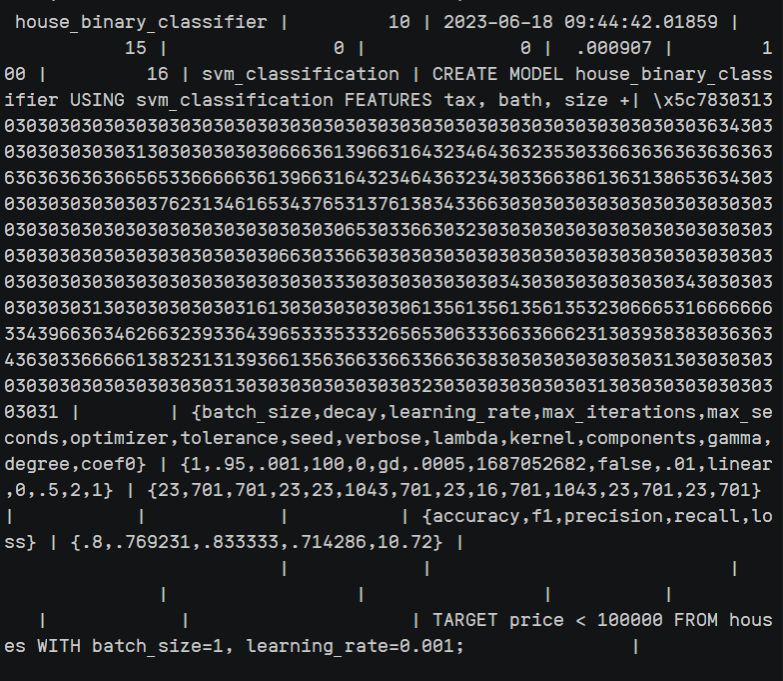
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



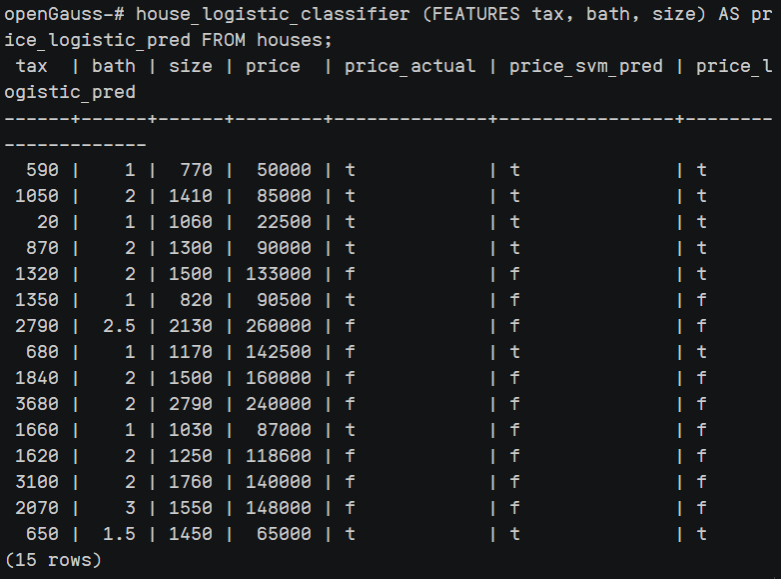
任务二：观察新模型的信息，将执行结果截图。

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，它们在目标和输出的类型上有所不同。

分类模型的目标是将输入数据分为不同的类别或标签。它试图预测离散的输出变量，通常表示为类别或标签。分类模型可以处理二分类问题（将数据分为两个类别）或多分类问题（将数据分为多个类别）。一些常见的分类模型包括逻辑回归、支持向量机（SVM）、决策树和随机森林。

回归模型的目标是预测连续的输出变量。它试图建立输入和输出之间的关系，并预测连续数值的输出。回归模型通常用于预测和估计数值，如房价、销售量等。一些常见的回归模型包括线性回归、多项式回归、岭回归和神经网络。

因此，分类模型和回归模型之间的主要区别在于它们处理的输出类型。分类模型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。

**实践思考题2：什么是SVM算法？**

支持向量机（Support Vector Machine，SVM）是一种用于分类和回归的监督学习算法。它的主要目标是找到一个最佳的决策边界（或称为超平面），将数据集分为不同的类别。

SVM的工作原理是将输入数据映射到高维特征空间，并在该空间中找到一个最优的超平面，使得不同类别的数据点尽可能地分开，并且使得离超平面最近的数据点（称为支持向量）到该超平面的距离最大化。这种最大化间隔的方法可以帮助提高模型的泛化能力，使其对未见过的数据具有较好的预测能力。

在SVM中，数据点被视为特征空间中的向量，其中每个特征是一个维度。算法通过优化一个目标函数来寻找最优超平面。常用的目标函数有两种形式：硬间隔（Hard Margin）和软间隔（Soft Margin）。硬间隔SVM旨在找到一个完全分隔不同类别的超平面，假设数据是线性可分的。软间隔SVM考虑到数据可能是线性不可分的情况，允许一些数据点位于超平面错误的一侧，通过引入松弛变量来容忍这些错误。

SVM还可以通过使用核函数将数据映射到高维空间来处理非线性问题。核函数可以在低维特征空间中计算两个样本之间的相似性，从而将非线性问题转化为高维空间中的线性问题。

SVM算法具有很好的泛化能力，并且在处理小样本数据、高维数据和非线性数据时表现出色。它在许多领域，如文本分类、图像分类和生物信息学中都得到广泛应用。

**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？**

在分类问题中，常用的评价指标包括准确率（Accuracy）、精确率（Precision）、召回率（Recall）、F1值（F1-Score）。

准确率（Accuracy）是最简单直观的评价指标，表示分类正确的样本数占总样本数的比例。准确率可以计算为：准确率 = (TP + TN) / (TP + TN + FP + FN)，其中TP（True Positive）是真正例的数量，TN（True Negative）是真反例的数量，FP（False Positive）是假正例的数量，FN（False Negative）是假反例的数量。准确率高表示模型的整体分类效果好，但它可能不适用于样本不平衡的情况下。

精确率（Precision）衡量的是分类为正例的样本中真正例的比例，也称为查准率。精确率可以计算为：精确率 = TP / (TP + FP)。精确率高表示分类为正例的样本中有较少的假正例。

召回率（Recall）衡量的是真正例中被正确分类的比例，也称为查全率或灵敏度。召回率可以计算为：召回率 = TP / (TP + FN)。召回率高表示模型能够正确识别出较多的真正例。

F1值（F1-Score）是精确率和召回率的综合指标，结合了两者的性能。F1值可以计算为：F1值 = 2 \* (精确率 \* 召回率) / (精确率 + 召回率)。F1值的取值范围在0和1之间，越接近1表示模型的性能越好

**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？**

在回归问题中，常用的评价指标包括均方误差（Mean Squared Error，MSE）、均方根误差（Root Mean Squared Error，RMSE）、平均绝对误差（Mean Absolute Error，MAE）、决定系数（Coefficient of Determination，R-squared）和相关系数（Correlation Coefficient）。

均方误差（MSE）是回归模型中最常用的评价指标之一，它衡量预测值与真实值之间的平均差异的平方。均方误差可以计算为：MSE = (1/n) \* Σ(y\_i - ŷ\_i)^2，其中y\_i是真实值，ŷ\_i是预测值，n是样本数量。MSE的值越小表示模型的预测越准确。

均方根误差（RMSE）是均方误差的平方根，它与原始数据的单位相同，因此更易于解释。RMSE可以计算为：RMSE = sqrt(MSE)。与MSE一样，RMSE的值越小表示模型的预测越准确。

平均绝对误差（MAE）衡量预测值与真实值之间的平均差异的绝对值。平均绝对误差可以计算为：MAE = (1/n) \* Σ|y\_i - ŷ\_i|。MAE的值越小表示模型的预测越准确。与MSE相比，MAE对异常值更加鲁棒，因为它不考虑差异的平方。