openGauss AI特性创新实践课

195202

胡婧涵



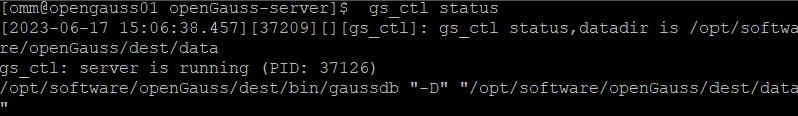
华为技术有限公司

# 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：

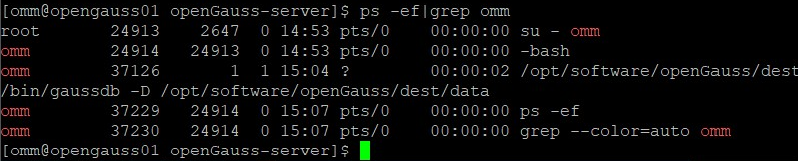
任务一：数据库状态验证

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. 定制化安装：源码编译允许用户根据自己的需求进行定制化安装。用户可以选择编译特定的功能模块、配置参数，以及优化编译选项，以满足其特定的使用场景和需求。  
与平台无关，可按需定制编译，最灵活；

2.性能较好；最新版本和功能：源码编译通常可以确保使用最新的数据库版本，包括最新的功能和安全修复。一些数据库的官方发行版或二进制包可能会有一些延迟，而通过源码编译可以更及时地获取到最新的更新。

3.  平台适配性：不同的操作系统和硬件平台可能需要特定的编译选项和优化参数。通过源码编译，用户可以针对自己的操作系统和硬件环境进行定制，以获得更好的性能和稳定性。可以安装到自己想要安装的目录、并且支持单台服务器安装多个版本的数据库；

4.调试和优化：源码编译允许用户在需要时进行调试和优化。如果在使用过程中遇到问题或性能瓶颈，可以通过编译选项和调试工具进行诊断和改进。可以自定义功能和模块，而且可以自定义部署的路径，统一的管理，减轻后期的维护工作量；

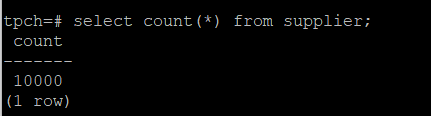
5. 教学和学习目的：通过编译和安装数据库源码，用户可以深入了解数据库的内部工作原理和架构。这对于数据库管理员、开发人员和研究人员来说是非常有价值的学习和教学资源。

# 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：

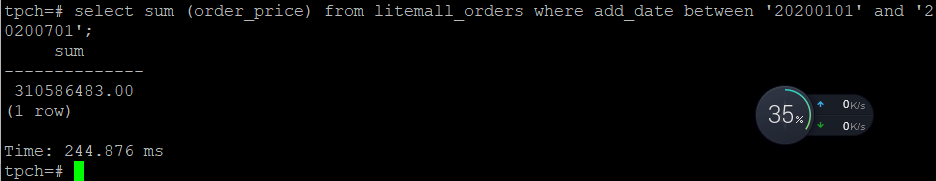
select count(\*) from supplier;;



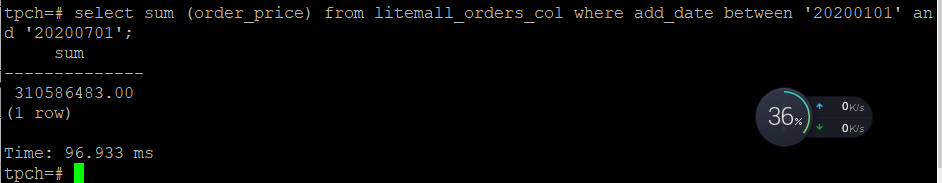
任务二：行存表与列存表执行效率对比

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';

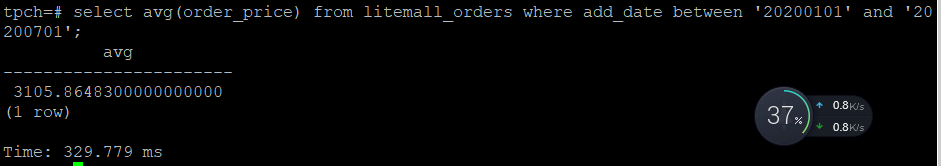


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

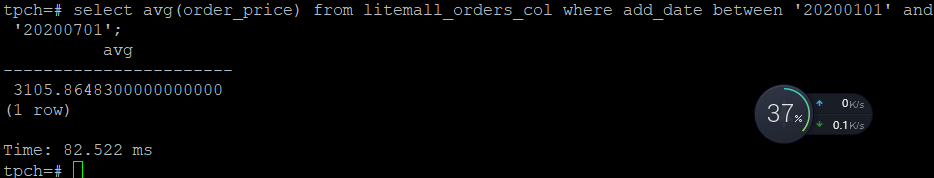


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

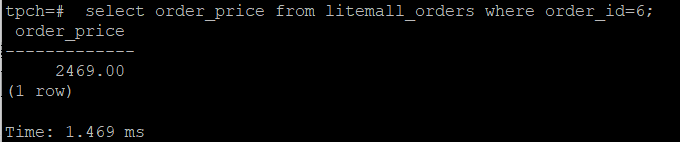


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

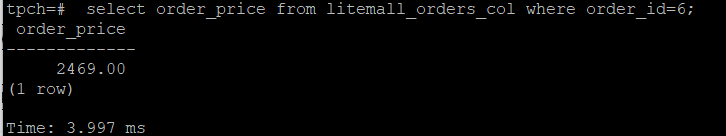


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

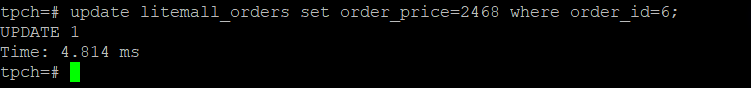


select order\_price from litemall\_orders\_col where order\_id=6;

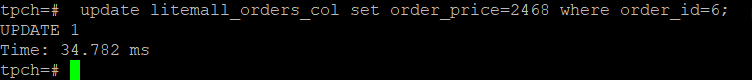


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



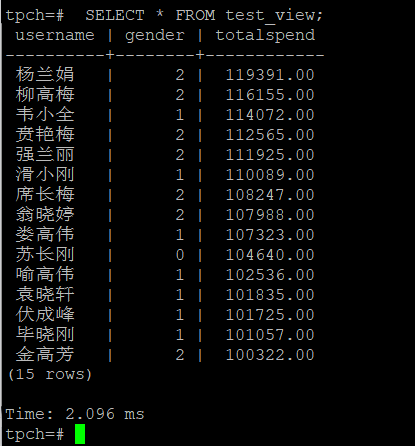
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

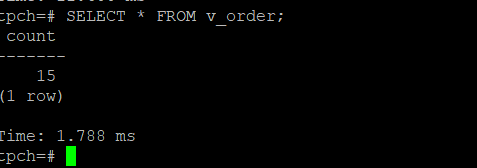
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



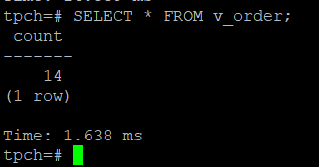
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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;



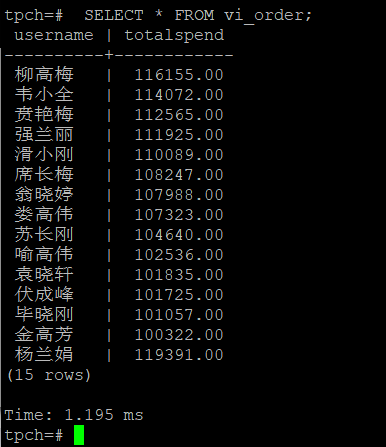
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

1.行存表是将一行数据作为一个整体来存储，适合于经常读取整行数据或者需要经常更新数据的场景，但是如果只需要查询某几列的数据，就会存在冗余列，增加了读取的时间和空间；

2.列存表是将一张表的每一列单独存放，适合于经常关注一张表某几列而非整表数据的场景，或者基于一列或比较少的列进行计算的场景，可以减少读取的时间和空间，但是如果需要查询整行数据或者频繁更新数据，就会增加了重组或写入的时间和空间；

3.行存表和列存表在数据压缩方面也有不同，列存表由于每一列的数据类型相同，更容易进行字典编码等压缩方式，减少了磁盘占用和IO开销，提高了查询效率；行存表则不容易进行压缩，而且压缩后还会影响查询效率；

4.行存表和列存表在索引方面也有不同，行存表通常需要建立B树等索引结构来加速查询，但是索引本身也会占用额外的空间和维护成本；列存表则可以利用每一列本身就是一个索引的特点，进行位图索引等方式来加速查询。

效率不同情况：

如果SQL语句涉及到对整行数据的读取或更新操作，或者需要快速插入数据，那么行存表效率更高；

如果SQL语句涉及到对某几列数据的读取或计算操作，或者需要进行聚合分析或压缩处理，那么列存表效率更高。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量物化视图是每次更新时把物化视图清空，再查询并把结果全部插入到物化视图中，适合于基表数据变化不频繁或者物化视图包含复杂的查询逻辑的场景；

增量物化视图是每次更新时只获取对基表的增量修改，再计算并应用到物化视图上，适合于基表数据变化频繁或者物化视图包含简单的查询逻辑的场景；

全量物化视图不需要记录基表的修改日志，而增量物化视图需要通过物化视图日志来监测基表的变动，并进行增量刷新；

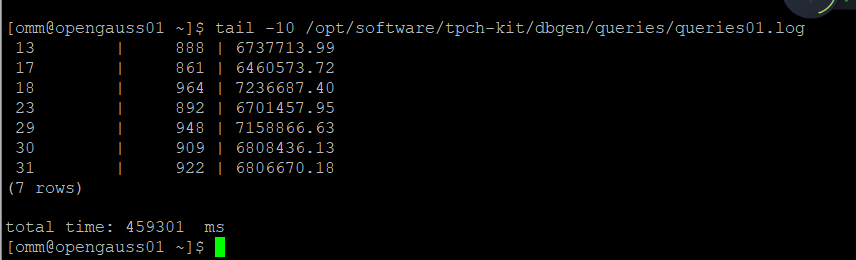
全量物化视图和增量物化视图所支持的查询语句有所不同，目前增量物化视图仅支持基表扫描语句或者UNION ALL语句，而全量物化视图则支持更多的查询语句。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

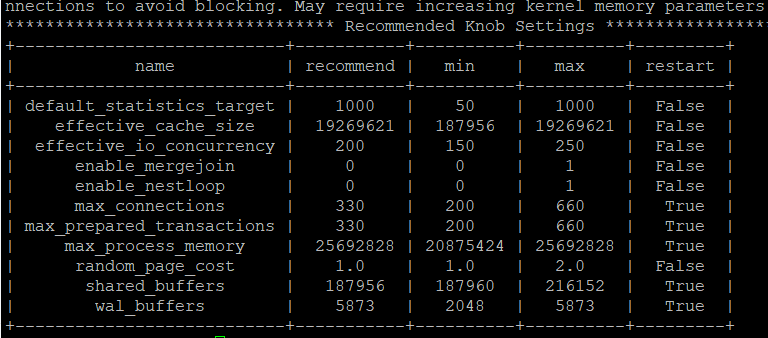
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

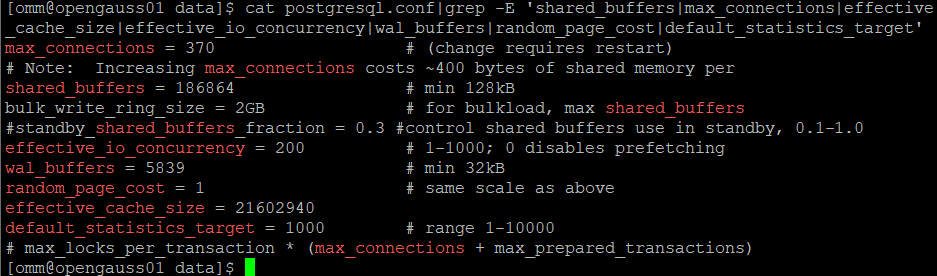
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

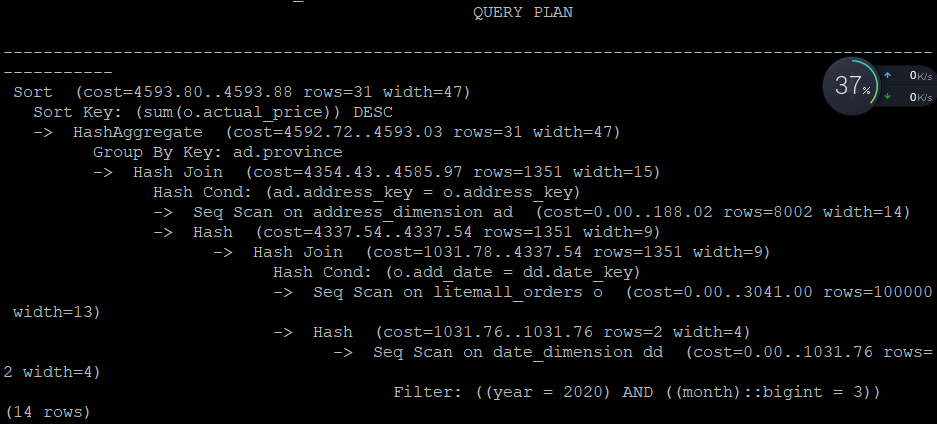
AND dd.year = 2020

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

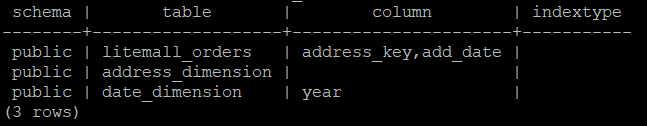
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

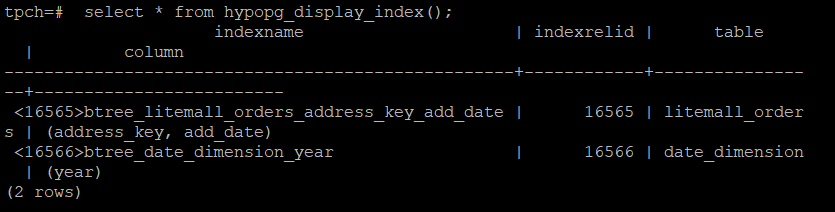
GROUP BY ad.province

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。

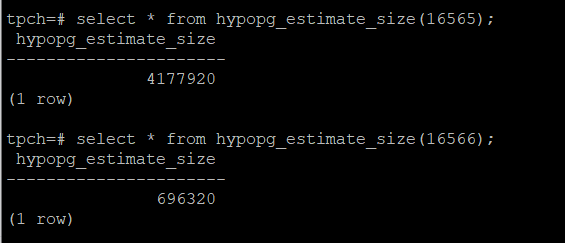
select \* from hypopg\_display\_index();



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

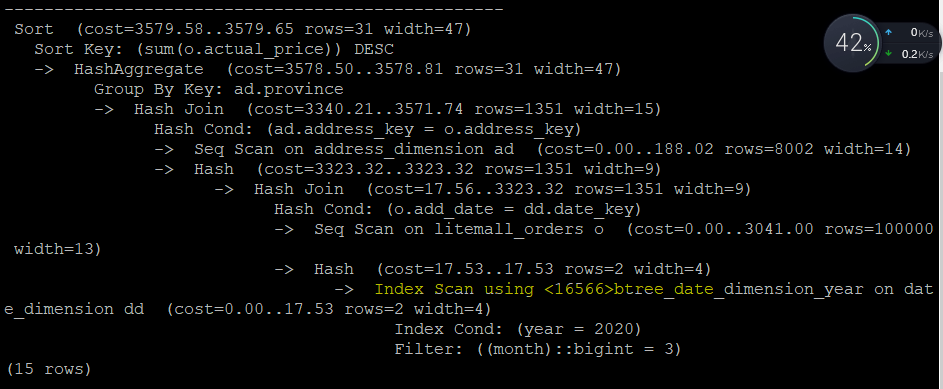
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

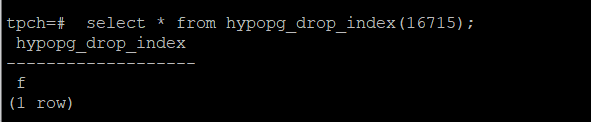
GROUP BY ad.province

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



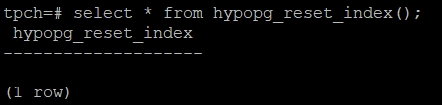
6. 删除某一个索引虚拟列，将执行结果截图。

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



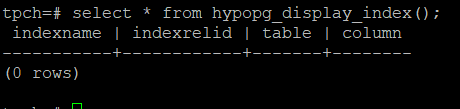
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

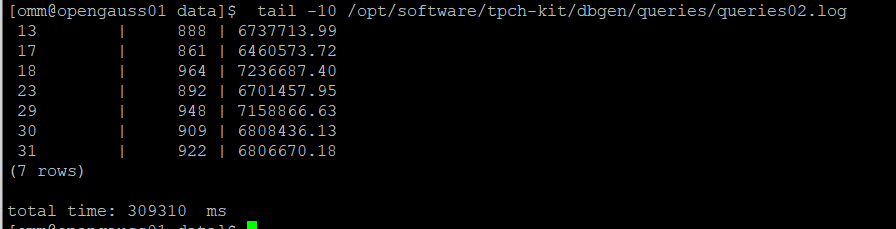
select \* from hypopg\_display\_index();



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：

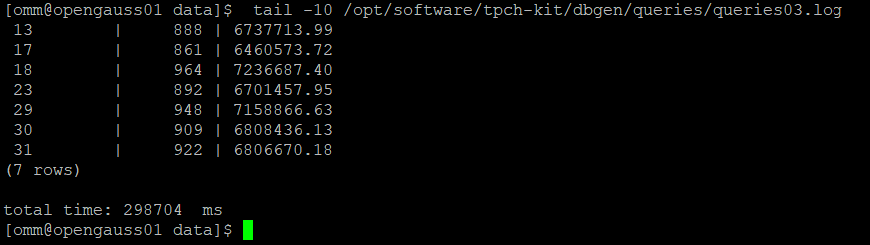
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

参数优化：

shared\_buffers：设置节点上用于共享缓冲区的内存大小，影响节点的缓存命中率和IO开销；

max\_connections：设置节点上允许的最大并发连接数，影响节点的并发能力和资源消耗；

effective\_cache\_size：设置节点优化器在一次单一的查询中可用的磁盘缓冲区的有效大小，影响节点的查询计划选择；

effective\_io\_concurrency：设置节点上每个表空间可以同时执行的最大异步IO请求数，影响节点的IO并发能力；

wal\_buffers：设置用于写WAL日志的缓冲区大小，影响节点的WAL写入性能和恢复速度；

random\_page\_cost：设置随机访问一个磁盘页的成本相对于顺序访问一个磁盘页的成本，影响节点的查询计划选择；

default\_statistics\_target：设置收集统计信息时默认采样率，影响节点的统计信息精度和查询计划选择。

原因：

因为这些参数都会直接或间接地影响数据库的性能，例如缓存命中率、IO开销、查询计划选择等。根据不同的负载和环境特征，这些参数可能需要调整为不同的值，以达到最佳的性能效果。X-Tuner根据数据库运行时收集到的信息，来推荐最合适的参数值。

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？

好处：

索引可以提高查询速度，减少全表扫描的开销，提高数据库的性能；

索引可以降低排序和分组操作的成本，避免额外的排序操作；

索引可以提高连接操作的效率，减少连接表的数据量；

索引可以实现数据的唯一性和完整性，保证数据的质量。

[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](https://zhuanlan.zhihu.com/p/61687047)：

优化SQL语句，避免使用不必要的子查询、连接、排序、聚合等操作，尽量使用简单的查询语句；

优化表结构，避免使用过长的字段类型，合理设计主键和外键，规范化或反规范化表结构；

优化存储引擎，根据不同的业务需求选择合适的存储引擎，如InnoDB、MyISAM等；

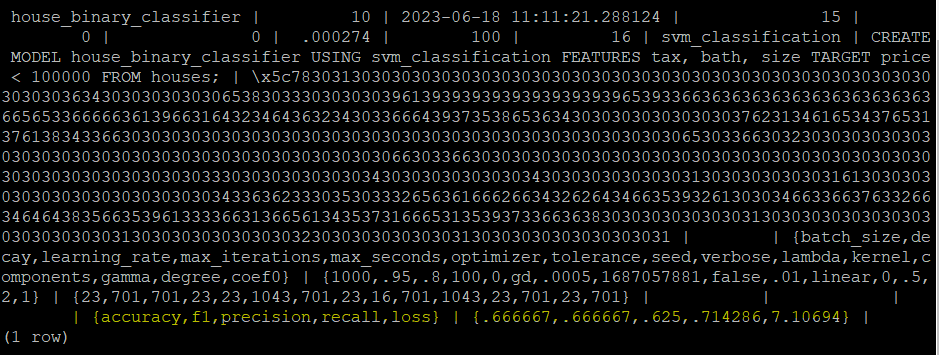
优化缓存策略，合理配置缓存大小和缓存过期时间，利用缓存技术减少数据库访问次数；

优化分区策略，根据数据的访问频率和特征进行分区，提高数据的局部性和可管理性。

# 关卡四、openGauss的DB4AI特性应用

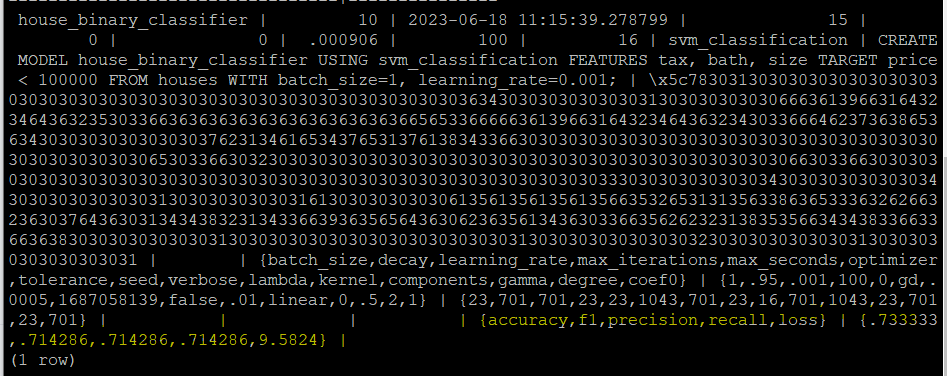
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



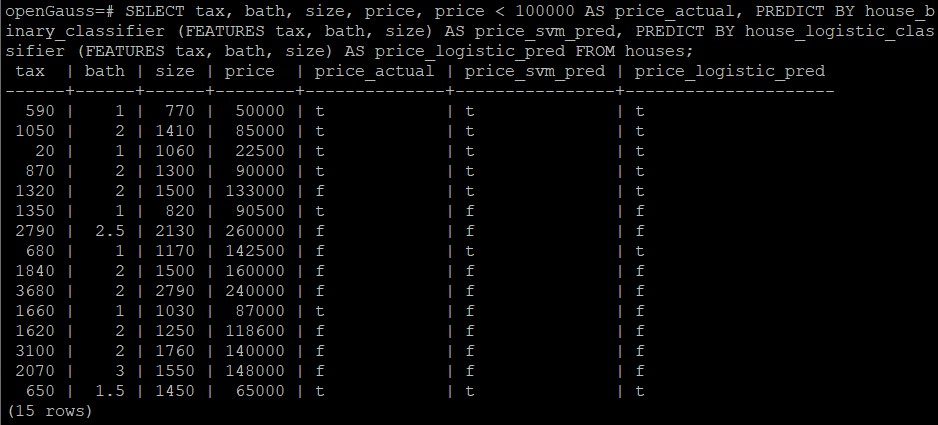
任务二：观察新模型的信息，将执行结果截图。

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：  
- 用途：分类模型用于预测离散的类别或标签。它将输入数据映射到预定义的类别中的一个，并进行分类判定。  
- 输出变量：输出变量是有限个离散的类别标签，例如二元分类（两个类别）或多元分类（多个类别）。

回归模型：  
- 用途：回归模型用于预测连续的数值输出。它通过学习输入特征与输出变量之间的关系，对连续数值进行预测。  
- 输出变量：输出变量是连续的数值或实数，可以是任意范围内的实数值。

主要区别：  
1. 输出变量类型：分类模型的输出是离散的类别标签，而回归模型的输出是连续的数值。  
2. 预测目标：分类模型旨在将数据分为不同的类别，回归模型旨在预测数值。  
3. 模型类型：分类模型通常使用分类算法，如逻辑回归、决策树、支持向量机等。回归模型通常使用回归算法，如线性回归、决策树回归、神经网络回归等。  
  
4.需要根据具体的问题和数据类型选择适当的模型类型，分类模型用于预测类别，而回归模型用于预测连续数值。

实践思考题2：什么是SVM算法？

SVM算法是一种分类算法，其基本思想是将数据映射到高维空间中，并在该空间中找到一个超平面，使得各类数据点到该超平面的距离最大。

SVM算法可以处理线性可分和非线性可分的数据集，通过使用核函数将非线性可分的数据转化为线性可分的数据。

SVM算法可以通过拉格朗日乘子法和对偶问题将原始的优化问题转化为凸二次规划问题，并通过最优化算法求解。

SVM算法可以用于二分类和多分类问题，也可以用于回归问题。

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）：  
   - 含义：准确率是分类正确的样本数与总样本数之比。它衡量模型在整个数据集上分类正确的能力。  
   - 计算方法：准确率 = 分类正确的样本数 / 总样本数。

2. 精确率（Precision）：  
   - 含义：精确率衡量模型在预测为正类别的样本中，真正为正类别的比例。它关注的是模型的预测结果中真正为正类别的准确性。  
   - 计算方法：精确率 = 真正为正类别的样本数 / 预测为正类别的样本数。

3. 召回率（Recall）：  
   - 含义：召回率衡量模型在所有真实为正类别的样本中，正确预测为正类别的比例。它关注的是模型对正类别的覆盖程度。  
   - 计算方法：召回率 = 真正为正类别的样本数 / 所有真实为正类别的样本数。

4. F1 值（F1 Score）：  
   - 含义：F1 值是精确率和召回率的综合指标，用于平衡两者之间的关系。它是精确率和召回率的调和平均值。  
   - 计算方法：F1 值 = 2 \* (精确率 \* 召回率) / (精确率 + 召回率)。

5. ROC曲线：ROC（Receiver Operating Characteristic）曲线，又称接受者操作特征曲线。曲线对应的纵坐标是真正类率（True Positive Rate, TPR），横坐标是假正类率（False Positive Rate, FPR）。ROC 曲线描述了分类器在不同阈值下的性能变化情况。ROC 曲线越靠拢左上角（0,1）点，即越偏离 45 度对角线越好。

6.AUC值：AUC（Area Under Curve）值是处于 ROC 曲线下方的那部分面积的大小。AUC 值越大，代表模型的性能越好。AUC 值可以通过计算 ROC 曲线下方各个小矩形的面积之和得到。

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（Mean Squared Error，MSE）：  
   - 含义：均方误差衡量模型预测值与真实值之间的平均差异程度。它计算预测值与真实值之间差的平方，并求其平均值。  
   - 计算方法：MSE = (1/n) \* Σ(yi - ŷi)²，其中 yi 是真实值，ŷi 是模型的预测值，n 是样本数量。

2. 均方根误差（Root Mean Squared Error，RMSE）：  
   - 含义：均方根误差是均方误差的平方根，它是对均方误差的标准化指标。RMSE 衡量模型预测值与真实值之间的平均差异程度。  
   - 计算方法：RMSE = √MSE

3. 平均绝对误差（Mean Absolute Error，MAE）：  
   - 含义：平均绝对误差衡量模型预测值与真实值之间的平均绝对差异程度。它计算预测值与真实值之间的绝对差，并求其平均值。  
   - 计算方法：MAE = (1/n) \* Σ|yi - ŷi|，其中 yi 是真实值，ŷi 是模型的预测值，n 是样本数量。

4. 决定系数（Coefficient of Determination，R²）：  
   - 含义：决定系数衡量模型对因变量的解释能力，即模型对观测数据的方差的解释程度。它表示模型预测值与真实值之间的方差比例。  
   - 计算方法：R² = 1 - (Σ(yi - ŷi)² / Σ(yi - ȳ)²)，其中 yi 是真实值，ŷi 是模型的预测值，ȳ 是真实值的平均值。  
这些评价指标在回归问题中用于度量模型的预测精度和误差程度。MSE、RMSE、MAE 衡量预测值与真实值之间的差异程度，越小表示模型越准确。R² 衡量模型解释观测数据方差的能力，越接近1表示模型拟合较好。根据具体问题的需求，选择适合的指标进行评估和比较。