openGauss AI特性创新实践课



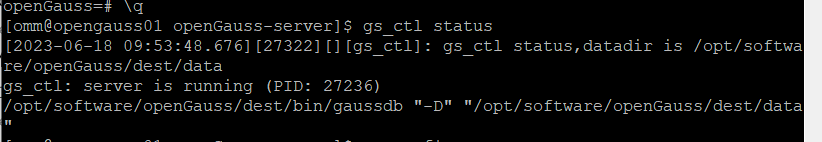
华为技术有限公司

# 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：

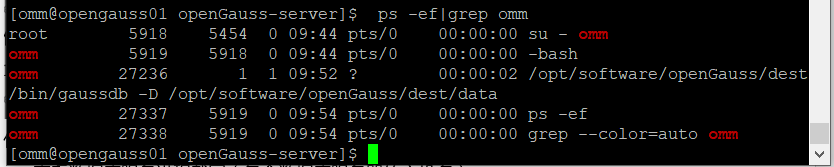
任务一：数据库状态验证

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？

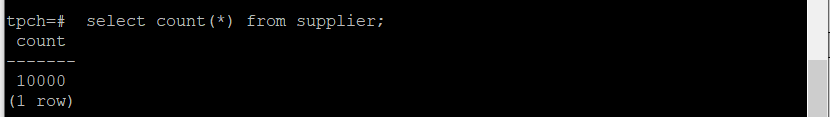
各种软件、库的小版本号和编译选项之间存在不可忽视的差异（即不兼容性），因此发行版预编译的二进制包派不上什么用场

# 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：

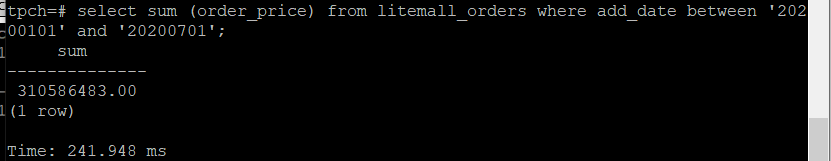
select count(\*) from supplier;;



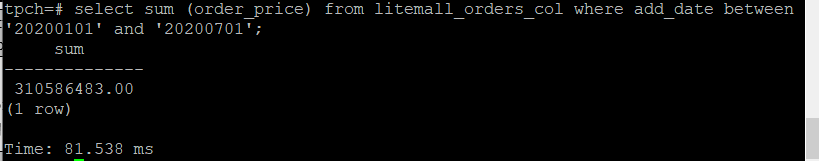
任务二：行存表与列存表执行效率对比

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';

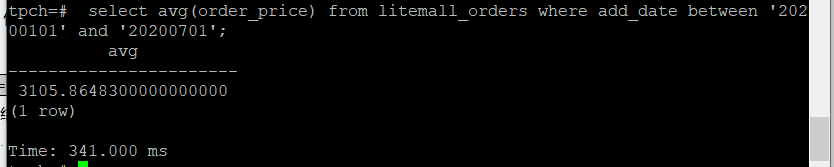


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

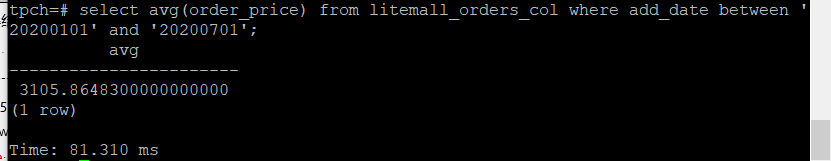


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

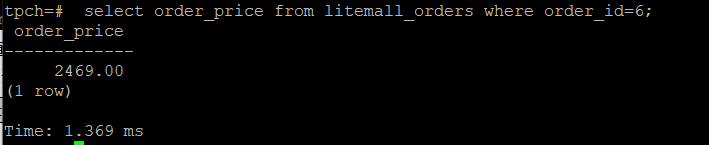


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

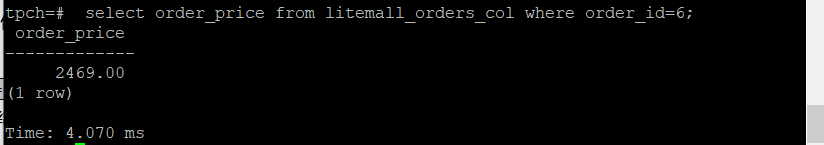


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

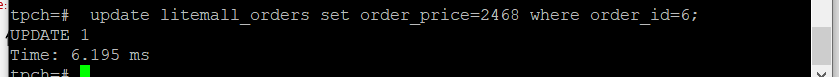


select order\_price from litemall\_orders\_col where order\_id=6;

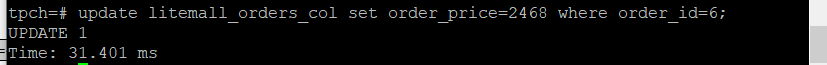


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

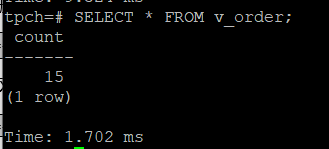
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



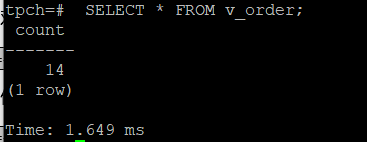
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

行存表将数据按行存储在磁盘上，每一行包含了一条记录的所有字段信息。而列存表则将数据按列存储，即将每个字段的值按照列存储在磁盘上，相同列的数据在物理上相邻。

当执行以下类型的 SQL 时，行存表效率更高：

需要读取整行数据的查询：如果 SQL 查询需要获取一整行的数据（多个字段），则行存表可以更快地提供所需的数据，因为整行数据在行存表中是连续存储的。

需要频繁进行插入和更新操作的场景：行存表在插入和更新操作上的性能通常较高，因为它们只需修改或添加单个行，而不涉及多个列的数据移动。

当执行以下类型的 SQL 时，列存表效率更高：

需要进行聚合操作的查询：列存表在进行聚合操作（如 SUM、AVG、COUNT 等）时效率较高，因为它们可以仅读取特定列的数据，而不需要读取整行。

需要查询特定列的查询：如果 SQL 查询只需要特定列的数据，列存表可以更快地提供所需的数据，因为它们的数据存储方式与查询方式更匹配。

需要进行大规模数据分析的场景：列存表适用于大规模数据分析场景，因为它们支持高效的列操作，例如列压缩、向量化计算等，可以在处理大量数据时提供更好的性能。

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：

全量物化视图是基于源表数据的一个快照，它存储了源表数据的预计算结果。

当源表数据发生变化时，全量物化视图需要完全刷新（refresh）以反映最新的数据。

刷新过程涉及删除旧数据并重新计算和插入新数据，可能需要消耗较长的时间和资源。

全量物化视图适用于源表数据更新频率较低的情况，或者对于实时性要求不高的报表查询等场景。

增量物化视图：

增量物化视图是基于源表数据的一个增量变化记录，它存储了源表数据的变化信息，而不是整个数据集。

当源表数据发生变化时，增量物化视图只需要更新变化的部分，而不需要重新计算和刷新整个视图。

更新过程通常比全量物化视图的刷新过程更快，因为它只处理变化的数据。

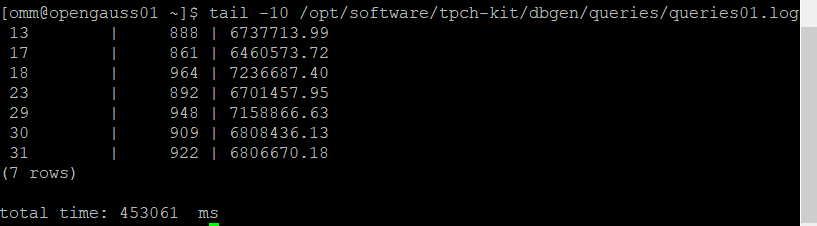
增量物化视图适用于源表数据更新频率较高的情况，可以更及时地更新视图，支持实时性要求较高的查询和分析。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

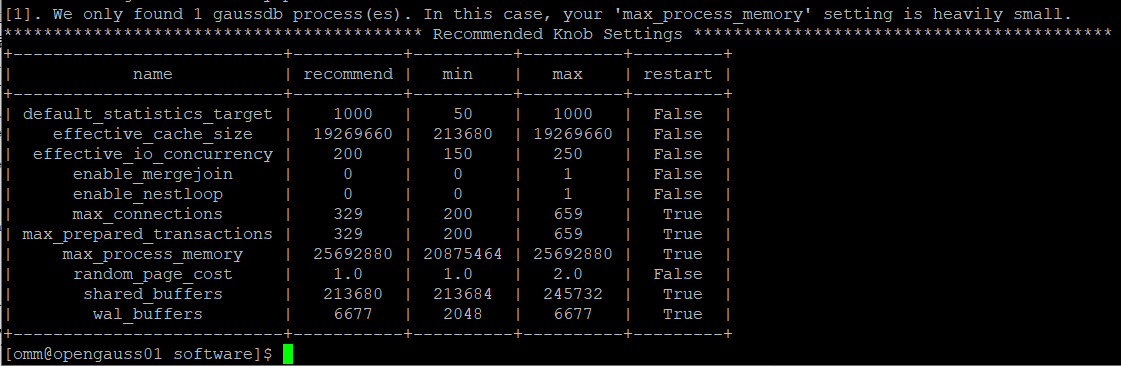
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

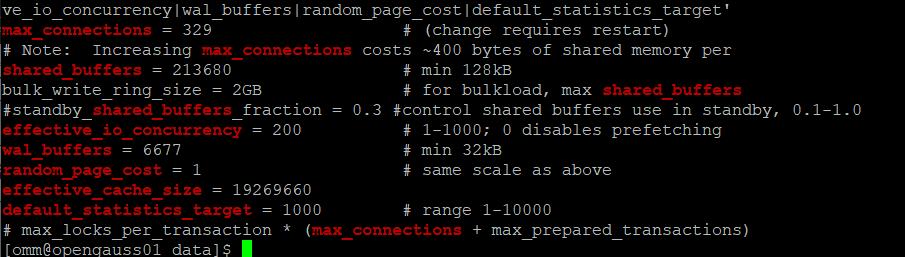
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

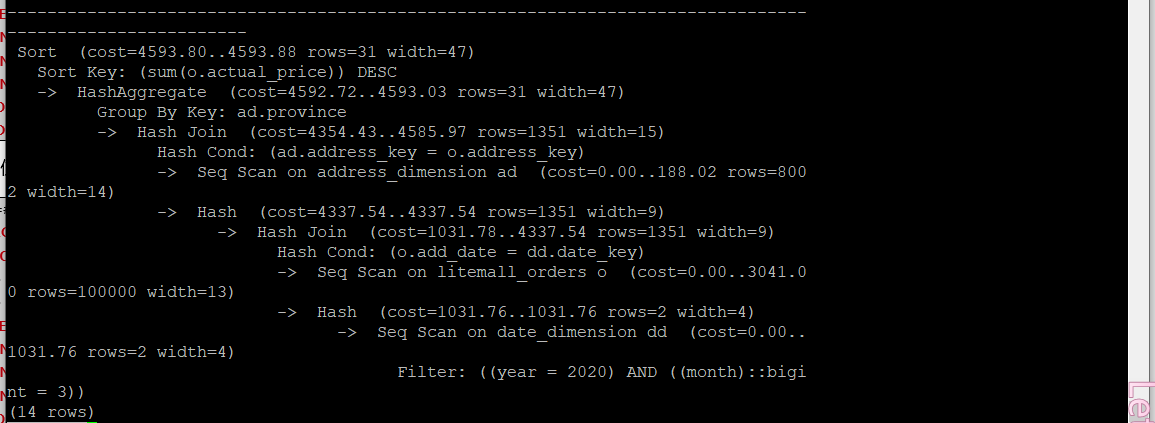
AND o.add\_date = dd.date\_key

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

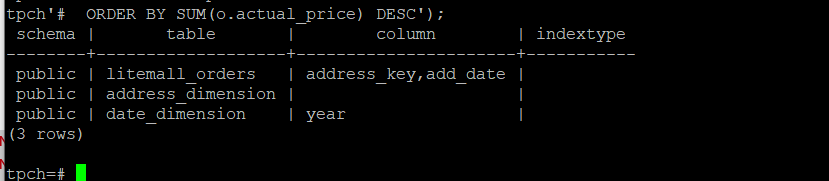
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

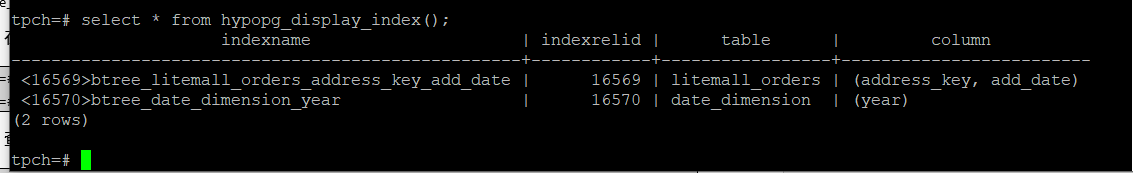
GROUP BY ad.province

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。

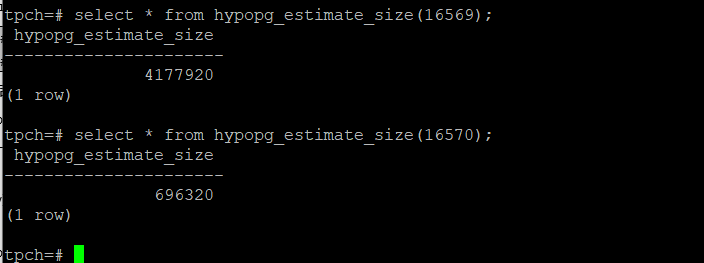
select \* from hypopg\_display\_index();



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

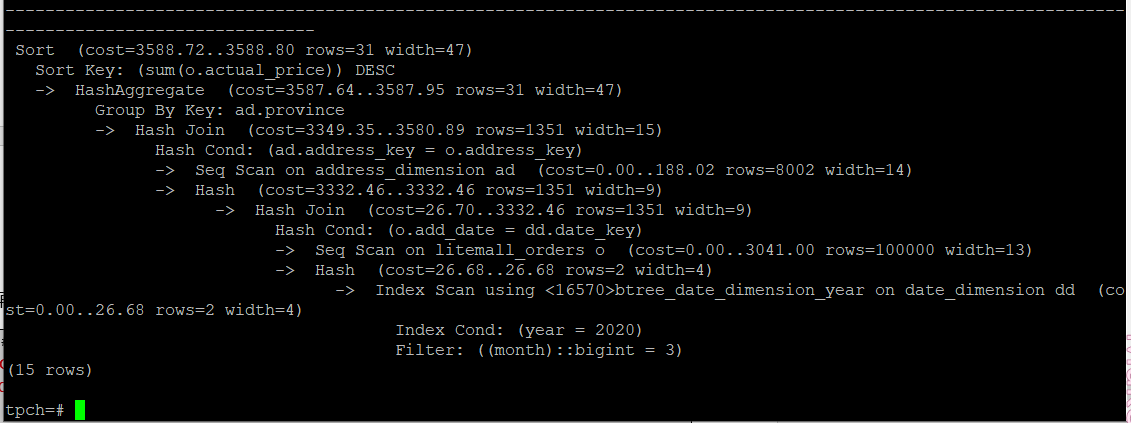
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

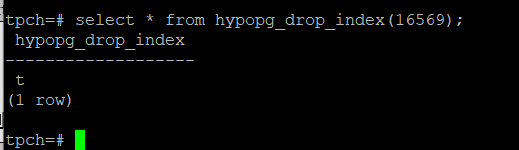
GROUP BY ad.province

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



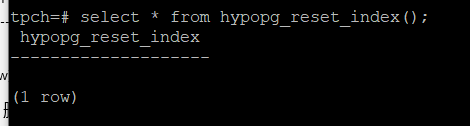
6. 删除某一个索引虚拟列，将执行结果截图。

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



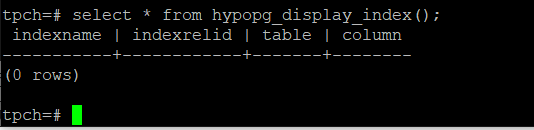
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

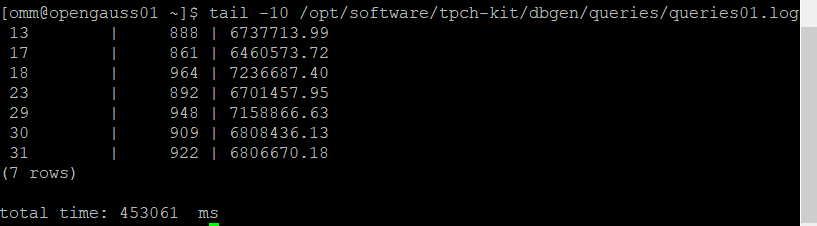
select \* from hypopg\_display\_index();



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：

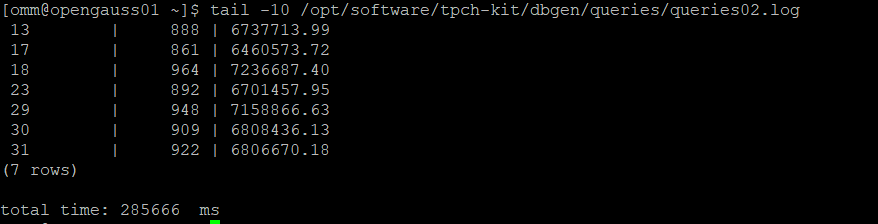
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

shared\_buffers

优化原因：该参数控制着数据库用于缓存数据页的内存大小。适当增大 shared\_buffers 可以提高数据的内存缓存效果，减少磁盘访问，从而提高查询性能。

max\_connections：

优化原因：该参数限制了数据库可以同时接受的最大客户端连接数。适当调整 max\_connections 可以根据系统的硬件资源和负载需求来合理分配连接数，避免过多的连接导致性能下降。

effective\_cache\_size

优化原因：该参数用于指定系统级缓存的大小，即操作系统文件系统缓存和数据库自身的共享缓存。适当设置 effective\_cache\_size 可以使查询使用缓存的概率更高，从而提高查询性能。

effective\_io\_concurrency

优化原因：该参数用于指定并行执行的磁盘 IO 操作的并发数。适当调整 effective\_io\_concurrency 可以充分利用磁盘的吞吐能力，提高查询的 IO 性能。

wal\_buffers

优化原因：该参数用于控制写入 WAL（Write-Ahead Log）的缓冲区大小。适当增大 wal\_buffers 可以提高 WAL 的写入性能，减少写入磁盘的次数，从而提高事务提交的性能。

random\_page\_cost：

优化原因：该参数用于指定随机访问磁盘页和顺序访问磁盘页的成本差异。设置 random\_page\_cost 为较低的值可以使查询优化器更倾向于选择使用索引等方法来减少随机磁盘访问，从而提高查询性能。

default\_statistics\_target

优化原因：该参数用于指定自动生成的统计信息的目标数量。适当增加 default\_statistics\_target 可以提高查询优化器生成准确统计信息的能力，从而提高查询性能和优化执行计划的准确性。

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？

索引在执行 SQL 查询时具有以下好处：

提高查询性能：索引可以加快查询的速度，通过快速定位到包含所需数据的位置，减少了扫描整个表的开销。索引可以使查询更高效，减少响应时间，提升用户体验。

减少磁盘 I/O：索引允许数据库直接定位到磁盘上存储数据的位置，而无需全表扫描。这减少了磁盘 I/O 操作的数量，降低了磁盘访问开销，提高了查询的效率。

支持排序和聚合：索引使得排序和聚合操作更高效，可以避免对整个表进行排序或聚合操作。通过利用索引的有序性，可以快速获取排序或聚合结果，减少排序和聚合的时间和资源消耗。

支持快速数据检索：索引可以根据查询条件快速定位到匹配的数据，从而加速数据检索过程。特别是在具有大量数据的表中，索引的使用可以极大地提升查询效率。

除了使用索引和优化参数外，还有其他一些方面可以对数据库进行优化，包括但不限于以下几个方面：

查询优化：通过优化 SQL 查询语句的编写方式、使用合适的查询语句和连接方式，以及创建适当的索引来提高查询性能。

表设计优化：设计合理的表结构，避免过度范式化或冗余设计。通过正确选择数据类型、定义主键和外键等，提高查询和操作的效率。

内存优化：增加内存资源，合理配置数据库的缓存区和内存参数，提高数据的内存缓存效果，减少磁盘 I/O，提升查询性能。

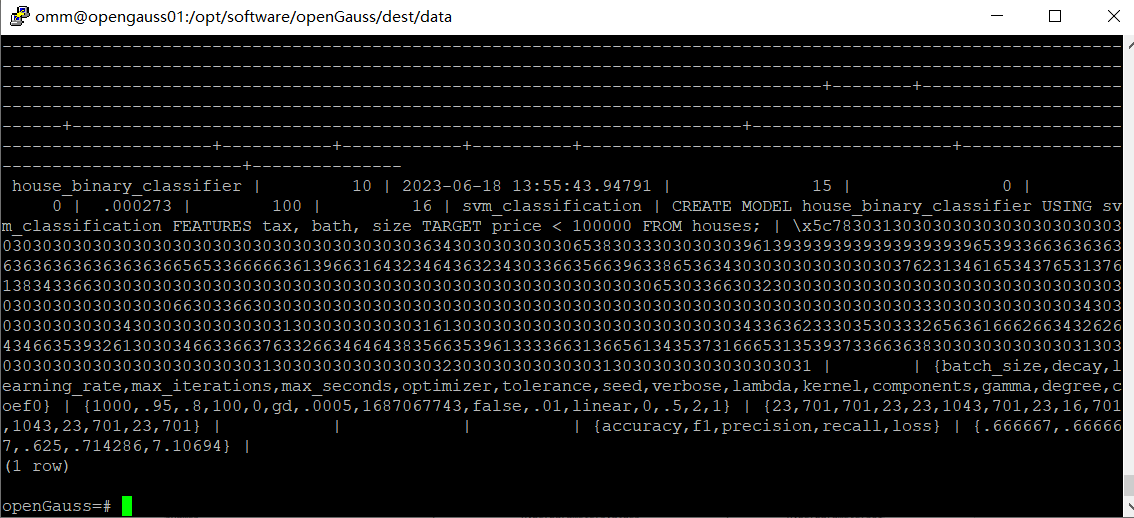
并发控制优化：通过合理的并发控制机制，如锁、事务隔离级别等，提高并发操作的效率和并发事务的处理能力。

定期维护和统计信息更新：定期进行数据库的维护工作，如优化索引、重新统计表的统计信息等，保持数据库的良好性能。

# 关卡四、openGauss的DB4AI特性应用

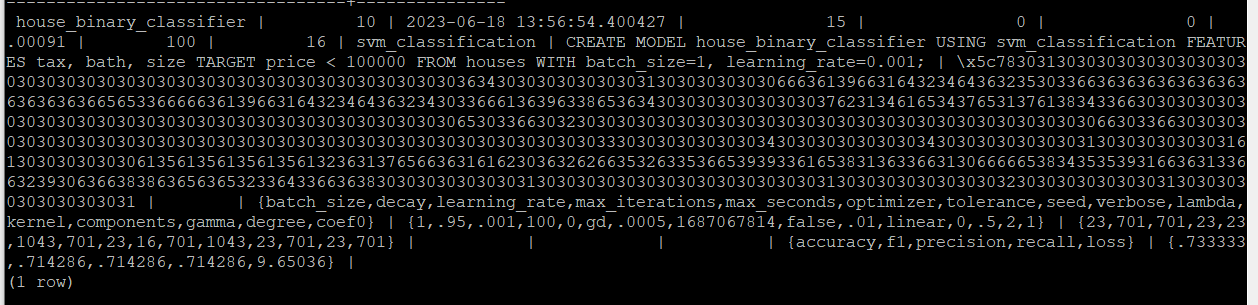
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



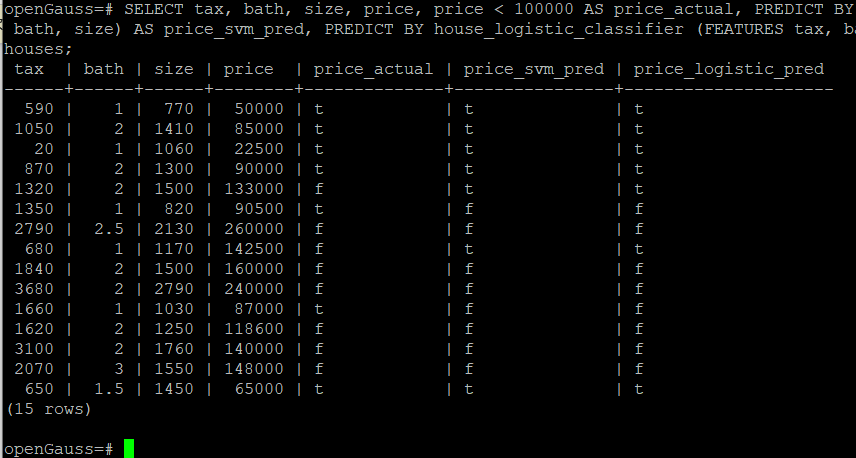
任务二：观察新模型的信息，将执行结果截图。

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不同类型的预测模型，它们在目标变量类型和建模方法上有所区别：

目标变量类型：

分类模型：分类模型用于预测离散的类别或标签。它的目标变量是有限的、预定义的类别。例如，预测邮件是垃圾邮件还是正常邮件、预测图像中的物体类别等。

回归模型：回归模型用于预测连续的数值型变量。它的目标变量可以是实数，例如，预测房价、预测销售额等。

建模方法：

分类模型：分类模型使用分类算法，通过从已有数据中学习特征和类别之间的关系，构建分类规则或决策边界。常见的分类算法包括逻辑回归、决策树、支持向量机、朴素贝叶斯等。

回归模型：回归模型使用回归算法，通过从已有数据中学习特征和目标变量之间的关系，构建数学函数模型，用于预测目标变量的连续值。常见的回归算法包括线性回归、岭回归、决策树回归、随机森林回归等。

模型输出：

分类模型：分类模型的输出是预测样本属于每个类别的概率或类别标签。例如，输出为垃圾邮件的概率为0.8，或直接输出类别标签为垃圾邮件。

回归模型：回归模型的输出是预测的连续数值。例如，预测房价为 200,000 美元。

总结来说，分类模型用于预测离散的类别或标签，使用分类算法，输出为类别概率或标签；而回归模型用于预测连续的数值型变量，使用回归算法，输出为预测的数值。

实践思考题2：什么是SVM算法？

SVM（支持向量机，Support Vector Machine）是一种常用的监督学习算法，用于解决分类和回归问题。它的主要目标是通过在特征空间中构建一个最优的超平面或决策边界，将不同类别的样本点分开。

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？

准确率（Accuracy）：准确率是最常用的评价指标之一，它表示分类器正确分类的样本数占总样本数的比例。准确率可以简单地衡量模型的整体分类准确性，但在不平衡类别分布的情况下，准确率可能不是一个全面的评估指标。

精确率（Precision）：精确率衡量的是分类器预测为正类的样本中有多少是真正的正类。它是真正例（True Positive）与预测为正例（True Positive + False Positive）的比值。精确率用于衡量分类器对正类的判断准确性。

召回率（Recall）：召回率是指所有真实正类中被分类器正确预测为正类的比例。它是真正例与实际正例（True Positive + False Negative）的比值。召回率用于衡量分类器对正类样本的查全率。

F1-score：F1-score是精确率和召回率的调和平均值，它综合了精确率和召回率的指标，用于综合评价分类器的性能。F1-score可以较好地平衡精确率和召回率的权衡，特别适用于不平衡类别分布的情况。

ROC 曲线与 AUC：ROC（Receiver Operating Characteristic）曲线是分类器性能评估中常用的工具，它展示了分类器在不同阈值下的真阳性率（True Positive Rate）与假阳性率（False Positive Rate）之间的关系。AUC（Area Under the Curve）是ROC曲线下的面积，用于量化分类器的整体性能。AUC越大，分类器的性能越好。

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？

均方误差（Mean Squared Error，MSE）：均方误差是最常用的回归评价指标之一，它衡量了预测值与真实值之间的平均差异的平方。MSE 越小表示预测结果越接近真实值。

均方根误差（Root Mean Squared Error，RMSE）：均方根误差是均方误差的平方根，它与均方误差具有相同的度量单位，但更易于解释。RMSE 也是衡量预测值与真实值之间的平均差异的指标，与 MSE 相比，RMSE 对异常值更加敏感。

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的绝对差值的平均值。MAE 衡量了预测值与真实值之间的平均绝对偏差，它对异常值不敏感y

决定系数（Coefficient of Determination，R²）：决定系数是衡量回归模型拟合优度的指标，表示模型能够解释目标变量方差的百分比。R² 的取值范围在0到1之间，越接近1表示模型对目标变量的解释能力越好。

相关系数（Correlation Coefficient）：相关系数是衡量预测值与真实值之间线性关系强度的指标。常用的相关系数是皮尔逊相关系数（Pearson Correlation Coefficient），它的取值范围在-1到1之间，正值表示正相关，负值表示负相关，接近0表示无相关性。