openGauss AI特性创新实践课



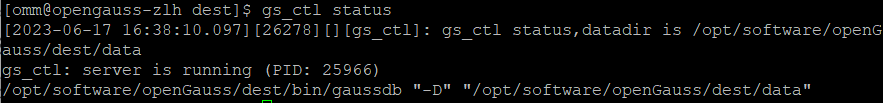
华为技术有限公司

# 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：

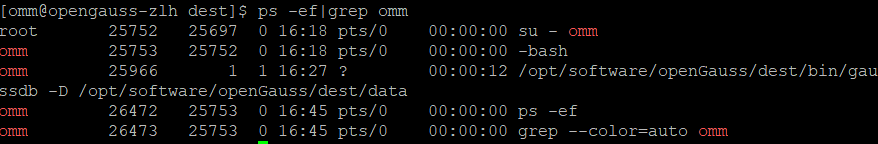
任务一：数据库状态验证

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，安装数据库的主要原因有以下几点：

①定制化需求：源码提供了更大的灵活性，可以根据特定的需求进行自定义配置和修改。编译源码可以允许我们选择需要的功能和参数，并针对特定硬件或操作系统进行优化设置。

②最新版本和修复补丁：源码编译可以确保使用最新版本的数据库软件，并及时获取到修复补丁和安全更新。这样可以提高数据库的稳定性和安全性。

③性能优化：编译源码使得我们可以根据自己的需求和环境进行性能调优，以获得更好的数据库性能。我们可以根据硬件平台的特点，选择合适的编译选项来优化数据库的运行效率。

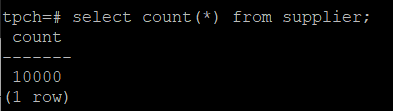
④系统集成：源码编译提供了与其他软件的集成和定制化的能力。这对于企业或特定的应用场景来说非常重要，可以保证数据库与其他组件之间的良好兼容性和无缝集成。

# 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：

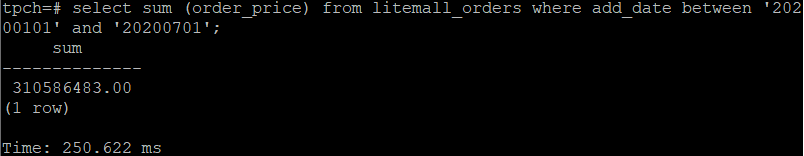
select count(\*) from supplier;;



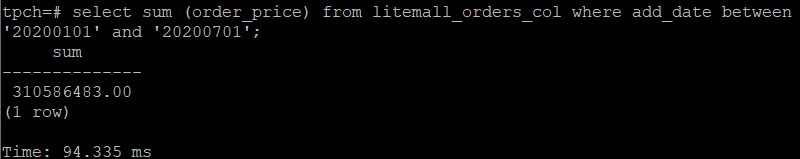
任务二：行存表与列存表执行效率对比

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';

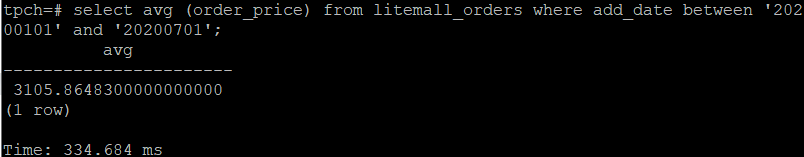


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

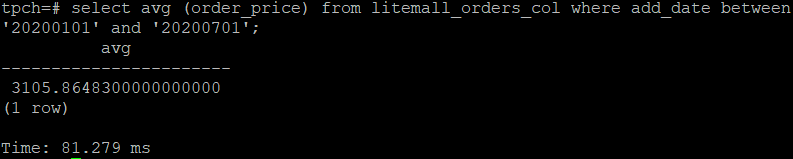


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

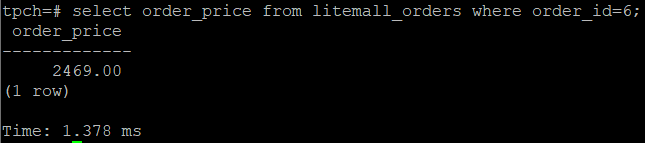


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

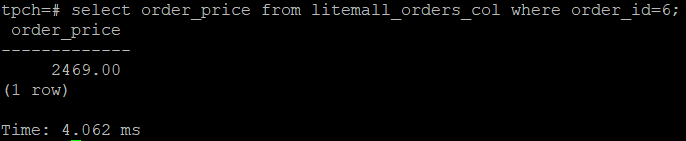


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

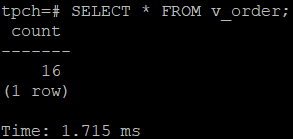
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



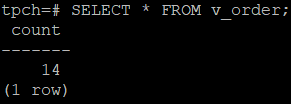
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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，执行时间不同主要有以下几个方面的原因：

①数据组织方式不同：行存表按照行的方式组织数据，每一行包含多个列的数据；而列存表按照列的方式组织数据，将同一列的值存储在一起。这样，在执行SQL语句时，行存表需要读取整行数据，而列存表只需要读取相关的列数据，这会导致列存表在某些查询场景下具有更快的执行速度。

②I/O 访问模式不同：行存表由于需要读取完整的行数据，可能会引发冗余的I/O操作，即使查询所需的列只占很小一部分。而列存表在查询时只需要读取所需的列数据，减少了无关的I/O访问，提高了查询效率。

③数据压缩和编码方式不同：列存表通常采用特定的压缩和编码技术，可以有效地减少存储空间，并提高数据的读取速度。而行存表通常不使用专门的压缩和编码方式。这意味着在同样的存储空间下，列存表可以存储更多的数据，并且在读取数据时可以减少磁盘I/O，从而加快查询速度。

在执行带有聚合函数的sql语句时，列存表效率更高；相反，在执行不带有聚合函数的sql语句时，行存表效率更高。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量物化视图和增量物化视图是两种不同的物化视图类型，它们的主要差别在于数据的刷新方式和更新策略。

①刷新方式：

全量物化视图：全量物化视图在每次刷新时都会完全重新计算和加载数据，将源表或其他物化视图的数据进行聚合、计算等操作，并替换现有的物化视图数据。这种刷新方式适用于数据规模较小、刷新频率较低的情况。

增量物化视图：增量物化视图只计算和加载源表或其他物化视图中发生变化的数据部分，这种刷新方式可以节省计算和加载的成本，提高刷新效率。增量物化视图通常通过记录变更日志或使用类似增量刷新技术来实现。

②更新策略：

全量物化视图：全量物化视图允许进行修改、插入和删除等操作，并将这些变更应用到源表。当刷新全量物化视图时，源表相应的数据也会被更新。这种更新策略可以保持物化视图和源表之间的一致性，但可能导致较大的刷新成本和性能开销。

增量物化视图：增量物化视图通常是只读的，不允许对其进行修改、插入和删除等操作。它主要用于查询和报表等读取操作，并通过增量刷新来保持数据的最新。这种更新策略适用于需要频繁查询但不需要实时一致性的场景，可以提高查询性能。

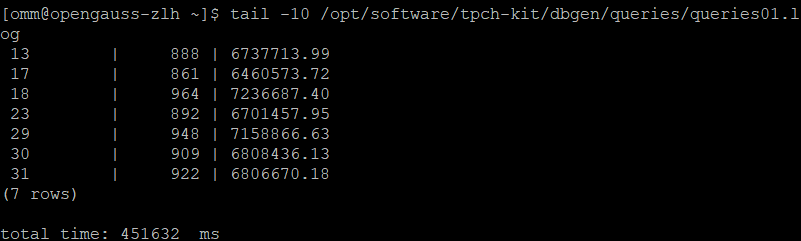
需要根据具体的业务需求和数据特点选择合适的物化视图类型。全量物化视图适合于静态或周期性更新的数据，而增量物化视图适合于频繁变更的数据和对实时性要求不高的场景。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

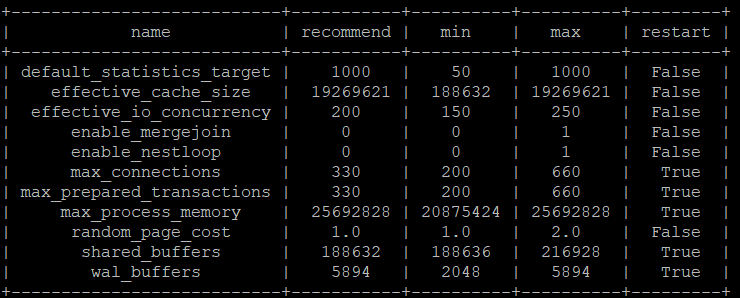
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

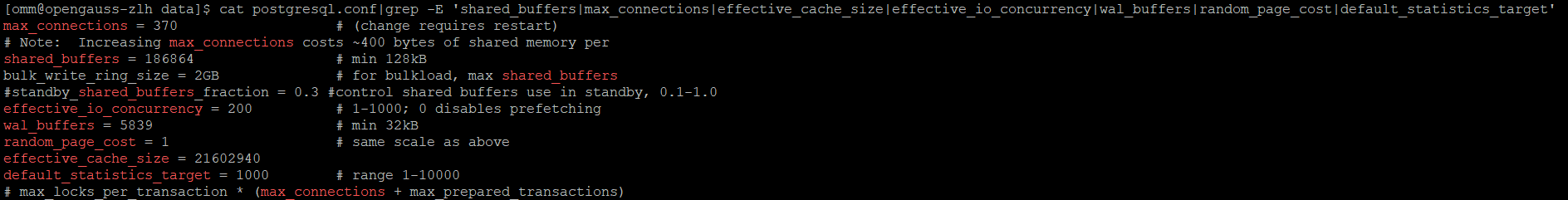
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

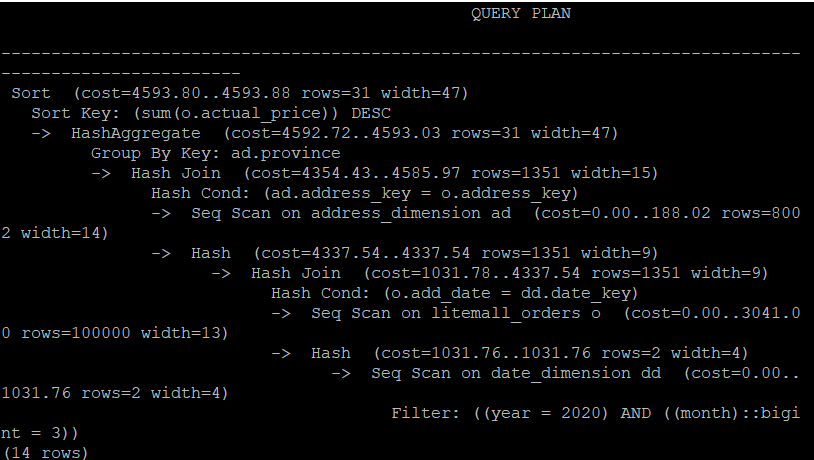
AND o.add\_date = dd.date\_key

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

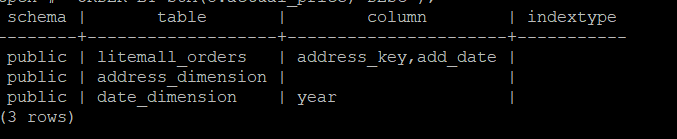
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

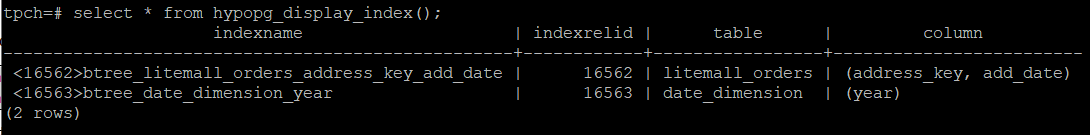
GROUP BY ad.province

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。

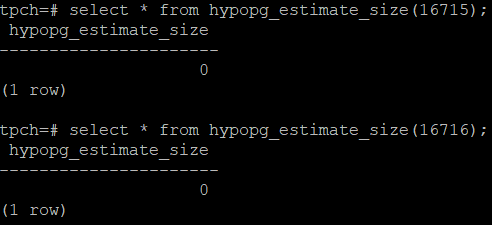
select \* from hypopg\_display\_index();



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

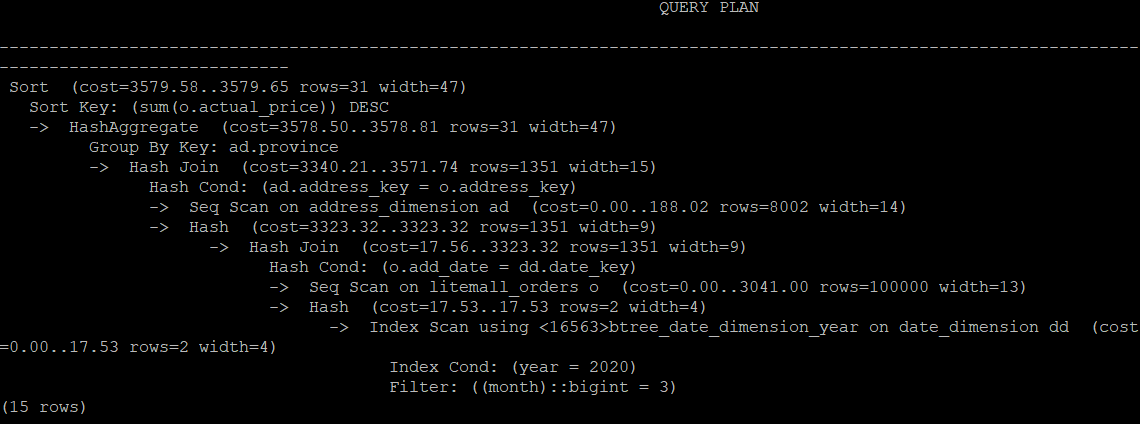
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

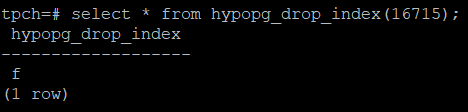
GROUP BY ad.province

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



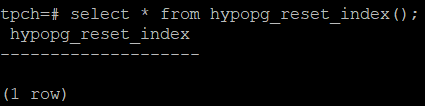
6. 删除某一个索引虚拟列，将执行结果截图。

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



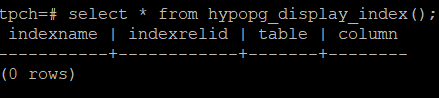
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

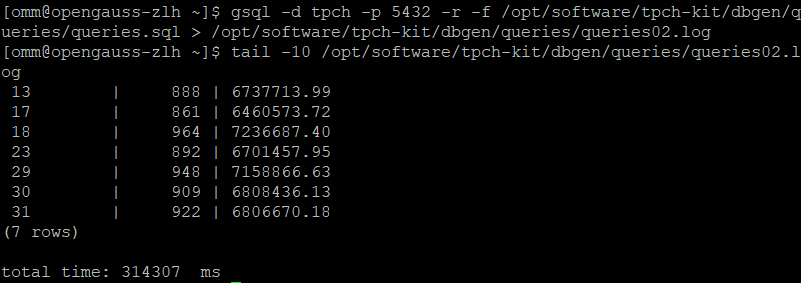
select \* from hypopg\_display\_index();



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：

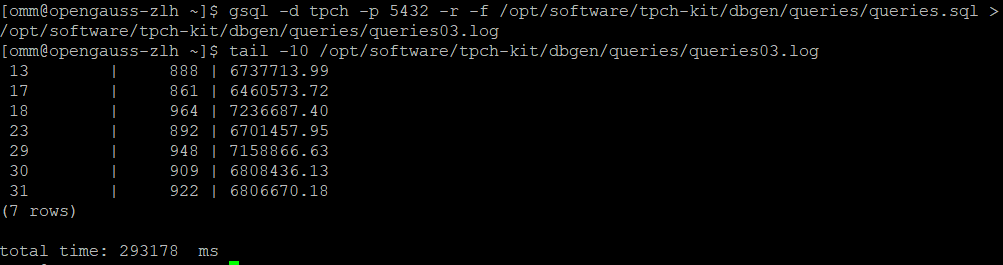
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

在shared\_buffers，max\_connections，effective\_cache\_size，effective\_io\_concurrency，wal\_buffers，random\_page\_cost，default\_statistics\_target上进行了优化，因为根据X-Tuner给出的意见，这些参数具有优化的空间。

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几个好处对于执行SQL：

①提高查询性能：索引可以加快数据库的查询速度，通过在索引列上建立索引，可以快速定位符合查询条件的数据，减少全表扫描的开销。

②减少磁盘I/O操作：索引可以使数据库系统减少磁盘I/O操作的次数，因为索引数据通常存储在内存中，通过索引可以直接获取数据块，而无需每次都读取整个数据块。

③优化排序和分组操作：对于包含排序、分组或聚合操作的SQL语句，索引可以提供有序的数据，减少排序和分组的时间开销。通过使用覆盖索引，可以避免在查询过程中访问主表，进一步提高性能。

除了使用索引和参数外，还有以下方面可以对数据库进行优化：

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靠性，包括选择适当的数据类型和字段长度、遵循范式规范、正确建立表之间的关系等。

②正确的索引策略：不仅要考虑哪些列需要创建索引，还要考虑索引的类型（B-Tree索引、哈希索引等）和顺序。

③优化SQL语句：对于频繁执行的SQL语句，可以通过重写、优化查询计划、使用合适的连接方式等手段来提高性能。

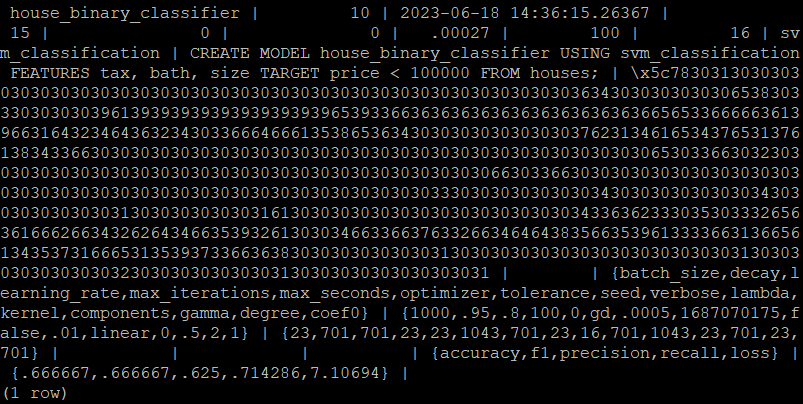
④缓存机制：使用数据库查询缓存，减少重复查询的开销。同时使用应用程序级别的缓存，如Redis等，可以减轻数据库的负载。

⑤分区和分表：针对大型数据表，可以考虑进行分区或分表，将数据划分到不同的存储位置上，以提高查询性能和管理效率。

# 关卡四、openGauss的DB4AI特性应用

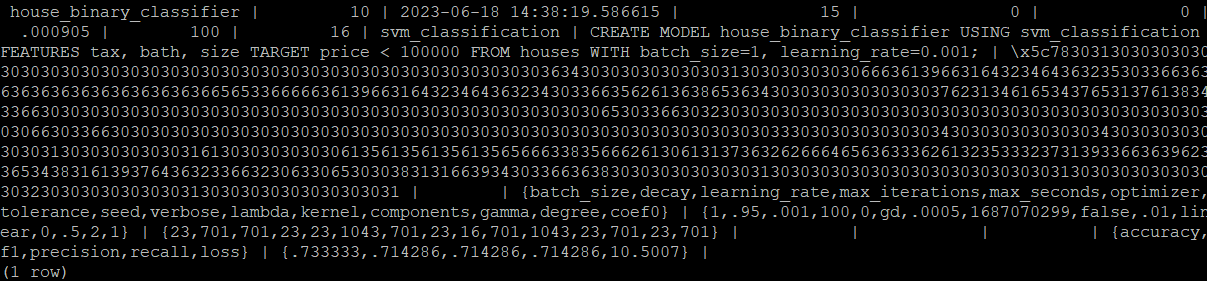
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



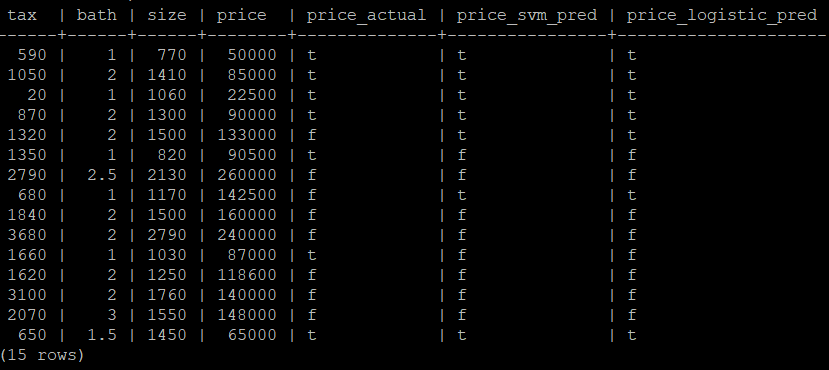
任务二：观察新模型的信息，将执行结果截图。

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常用的两种预测模型，它们在应用场景、输出类型等方面有所不同：

①应用场景：分类模型用于对样本进行分类或标签预测，将输入数据分为不同的类别。常见的分类问题包括垃圾邮件识别、图像分类、疾病诊断等。而回归模型则用于对连续性变量进行预测，通过建立输入特征与目标变量之间的数学关系，进行数值预测。典型的回归问题包括房价预测、销售额预测等。

②输出类型：分类模型的输出是离散的类别标签或概率。通常使用分类算法（如逻辑回归、决策树、支持向量机等）来训练模型，得到新样本所属的类别。回归模型的输出是连续的数值，可以是实数或整数。常用的回归算法包括线性回归、岭回归、随机森林回归等。

③模型评估指标：分类模型的评估指标通常包括准确率、精确率、召回率、F1值等，用于度量模型的分类性能。回归模型的评估指标通常包括均方误差（MSE）、平均绝对误差（MAE）、决定系数（R-squared）等，用于度量模型的预测准确程度和拟合优度。

④模型选择与建模方法：在选择分类模型时，可以根据数据类型、特征空间的分布情况等因素选择适合的算法，并通过特征工程、模型调参等方法进行模型训练与优化。而回归模型的选择则更多考虑目标变量是否为连续值，根据实际需求选择合适的回归算法，并进行数据预处理、特征选取等步骤来构建回归模型。

实践思考题2：什么是SVM算法？

SVM（Support Vector Machines）算法是一种常用的机器学习算法，主要用于分类和回归任务。它通过寻找一个最优的超平面来将样本空间划分为不同的类别或预测目标变量的值。

SVM的基本思想是找到一个能够将不同类别样本分隔开的超平面，并使得离该超平面最近的样本点到该超平面的距离最大化。这些离超平面最近的支持向量点决定了该超平面的位置和方向，因此称为支持向量机。

SVM算法的特点如下：

①非线性映射：SVM算法可以使用核函数将样本映射到高维特征空间中，从而在非线性问题上实现更好的分类效果。

②最大间隔分类器：SVM通过最大化支持向量到超平面的间隔，提高了模型的鲁棒性和泛化能力。

③非凸优化问题：SVM模型的训练过程可以转化为一个凸优化问题，可通过现有的优化算法高效求解。

④适用性广泛：SVM算法既可以用于二分类问题，也可以扩展到多分类和回归问题。

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？

如实践思考题1中所回答的，分类问题的评价指标包括准确率、精确率、召回率和F1值。下面分别说明它们的含义：

①准确率（Accuracy）：准确率是指分类器在所有样本中正确分类的比例，即正确预测的样本数除以总样本数。准确率可以给出一个整体的分类效果，但并不能完全反映分类器的性能，特别是在不均衡数据集中时。

②精确率（Precision）：精确率是指分类器在预测为正类的样本中，真正为正类的样本所占的比例。精确率衡量了分类器预测为正类的准确程度，其计算公式为：精确率 = TP / (TP + FP)，其中 TP 表示真正类（正样本被正确分类），FP 表示假正类（负样本被错误分类为正样本）。

③召回率（Recall）：召回率是指分类器在所有真实为正类的样本中，正确预测为正类的样本所占的比例。召回率衡量了分类器对正类样本的识别能力，其计算公式为：召回率 = TP / (TP + FN)，其中 TP 表示真正类，FN 表示假负类（正样本被错误分类为负样本）。

④F1值（F1-Score）：F1值是精确率和召回率的调和平均，综合了分类器的准确性和召回性能。F1值的计算公式为：F1 = 2 \* (Precision \* Recall) / (Precision + Recall)。F1值越高，表示分类器在精确率和召回率上取得了更好的平衡。

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？

如实践思考题1中所回答的，回归问题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）和决定系数（Coefficient of Determination，R-squared）。下面分别说明它们的含义：

①均方误差（MSE）：均方误差衡量了预测值与真实值之间的平均差异程度。它计算了预测值与真实值之差的平方的平均值，因此更重视较大误差。MSE的计算公式为：MSE = (1/n) \* Σ(y - y\_pred)^2，其中 n 表示样本数量，y 表示真实值，y\_pred 表示预测值。

②均方根误差（RMSE）：均方根误差是均方误差的平方根，用于衡量预测值与真实值之间的平均差异程度。RMSE和原始目标变量具有相同的量纲，因此更易于解释。RMSE的计算公式为：RMSE = sqrt(MSE)。

③平均绝对误差（MAE）：平均绝对误差衡量了预测值与真实值之间的平均绝对差异程度，忽略了误差的方向。它计算了预测值与真实值之差的绝对值的平均值，因此不受异常值的影响。MAE的计算公式为：MAE = (1/n) \* Σ|y - y\_pred|。

④决定系数（R-squared）：决定系数衡量了回归模型对观察值变异性的解释程度，即拟合优度。它的取值范围在0到1之间，越接近1表示回归模型的拟合程度越好，可以解释更多的观察值变异性；而越接近0表示模型对观察值的解释能力较弱。