openGauss AI特性创新实践课



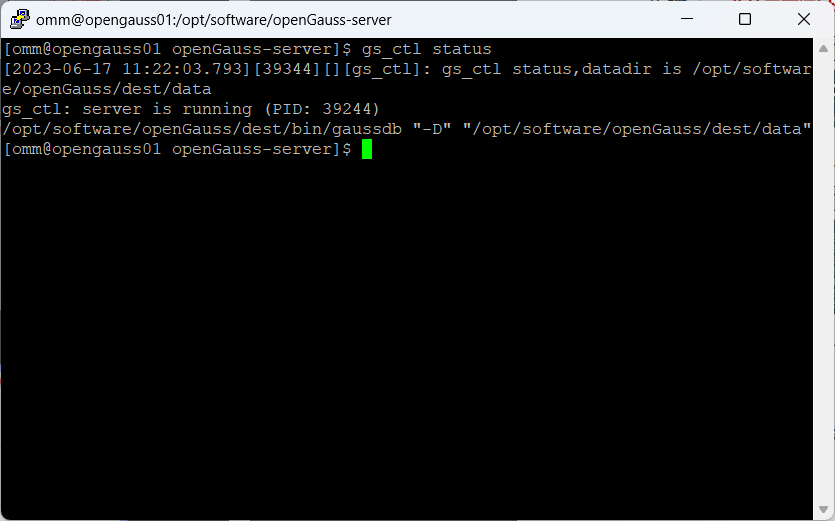
华为技术有限公司

# 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：

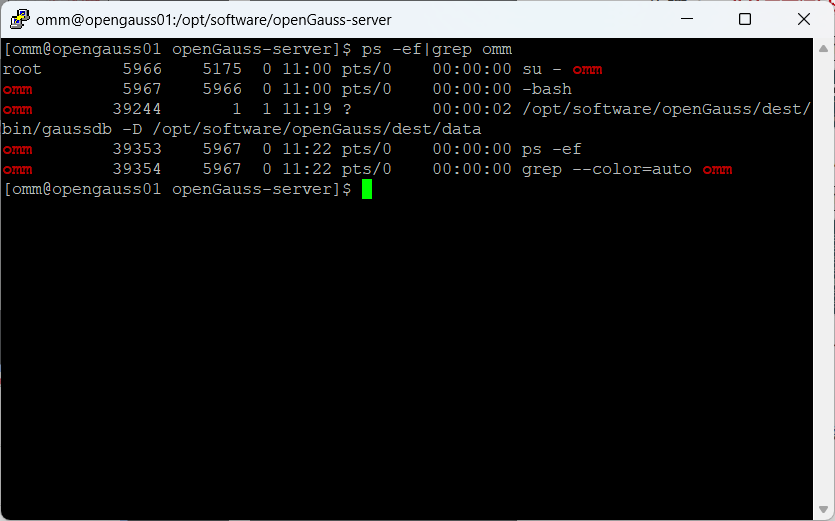
任务一：数据库状态验证

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？

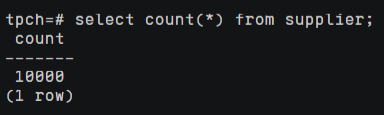
可以精确控制编译参数，修改代码，获得最新的版本

# 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：

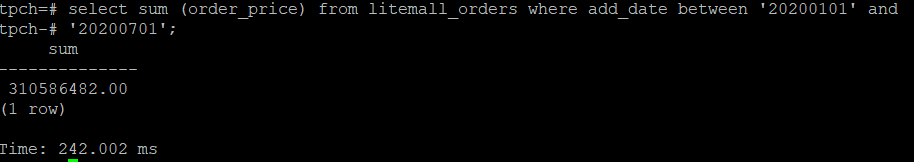
select count(\*) from supplier;;



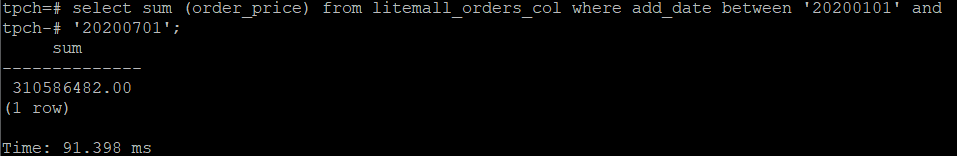
任务二：行存表与列存表执行效率对比

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';

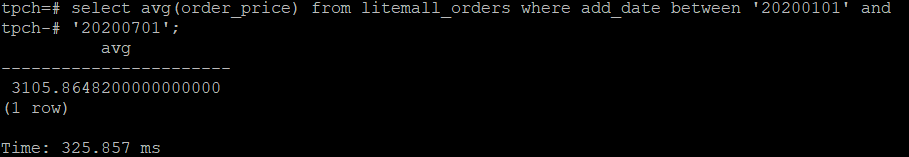


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

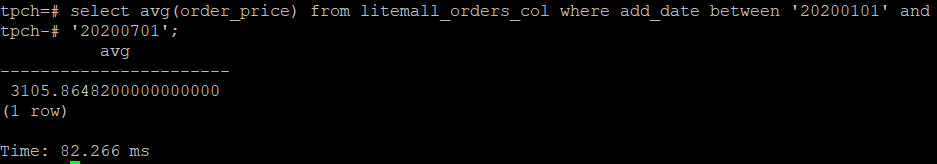


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

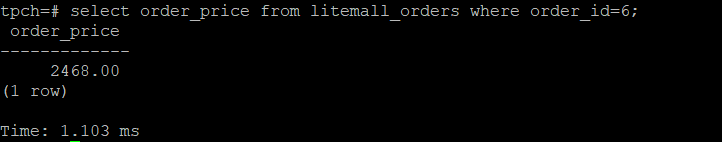


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

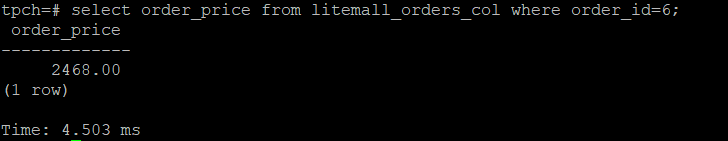


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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



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

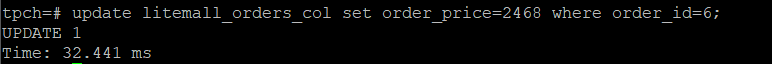


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

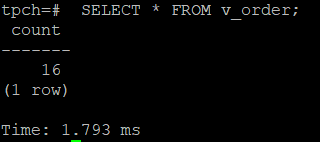
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



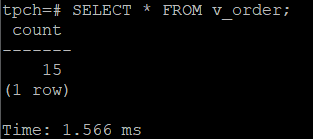
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

因为行存表执行操作时是一行一行寻找数据，列存表执行操作时是一列一列寻找数据，所以执行的时间会有不同。

当查询表中的某一字段并进行计算，如求和、平均等，对列存表进行操作比对行存表进行操作要快。当查询表中某一存在的值，或者对表进行修改时，对行存表进行操作比对列存表进行操作要快。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新。增量物化视图可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

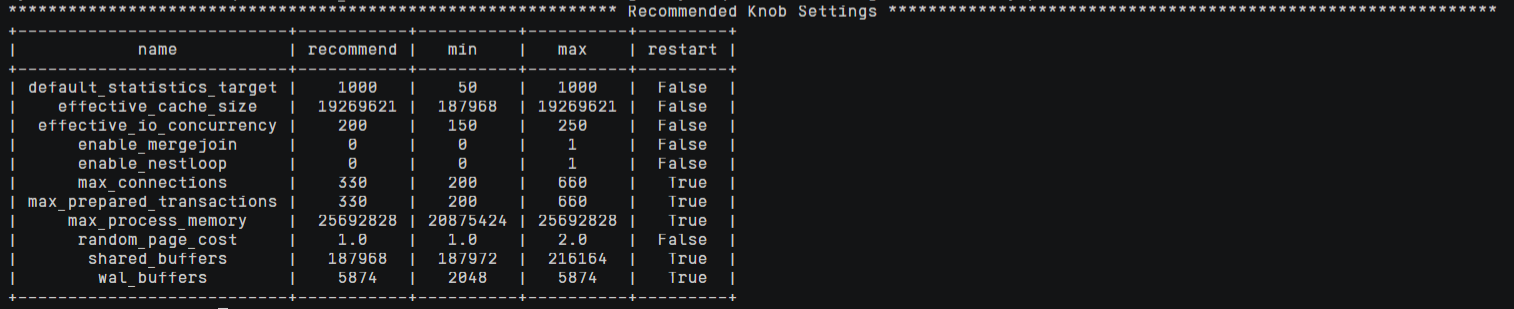
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

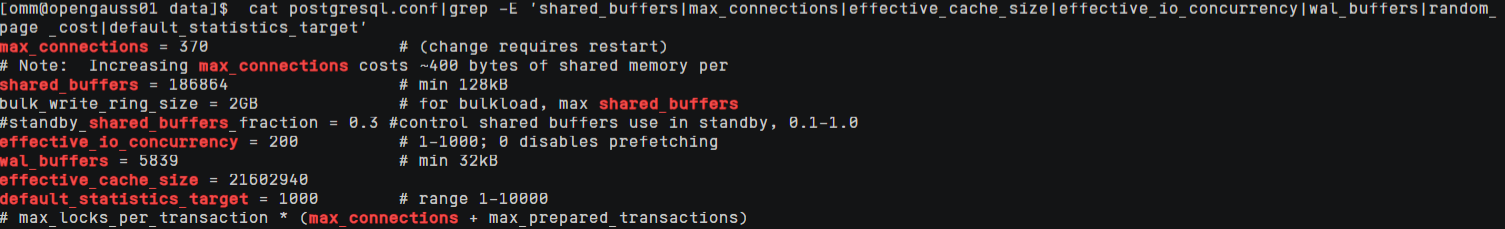
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

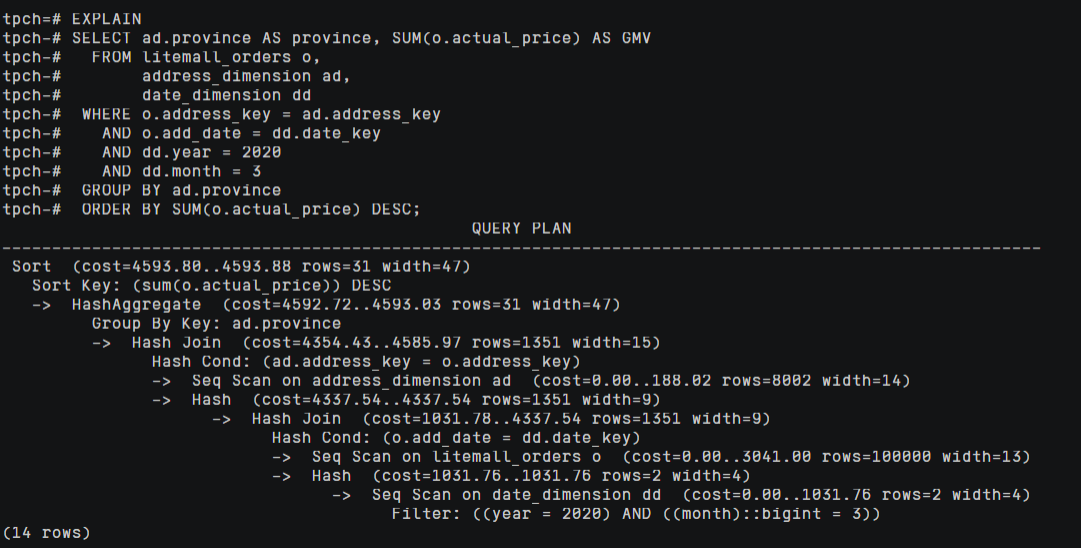
AND o.add\_date = dd.date\_key

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

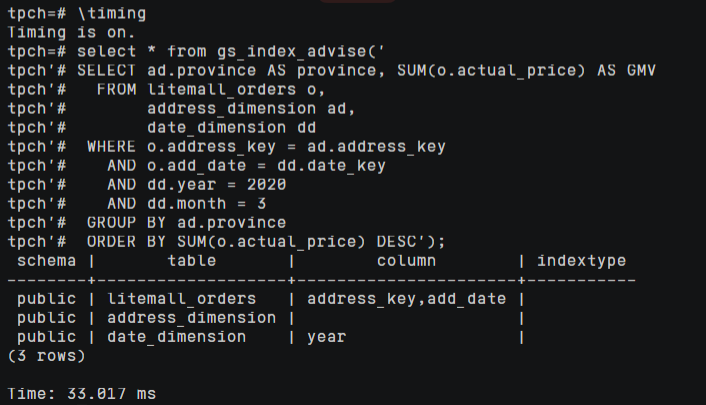
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

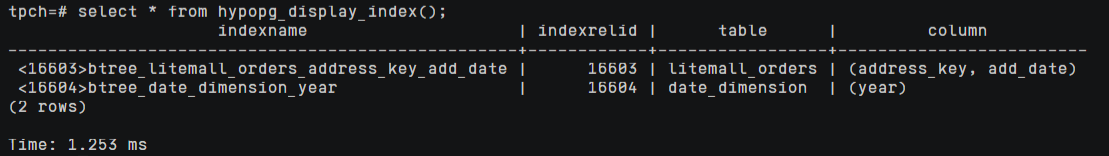
GROUP BY ad.province

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。

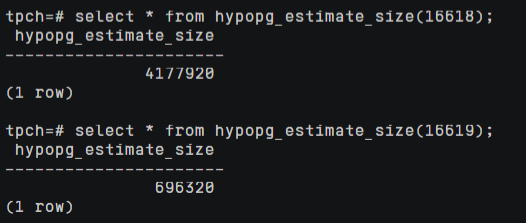
select \* from hypopg\_display\_index();



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(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

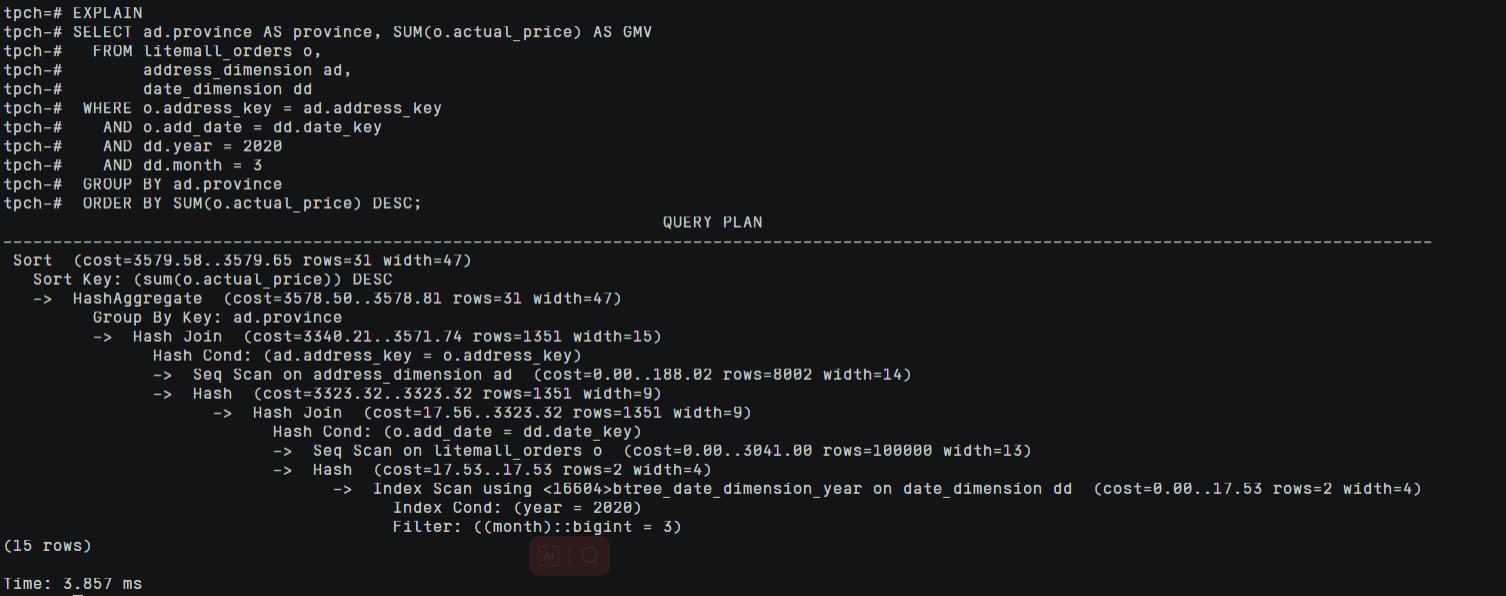
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

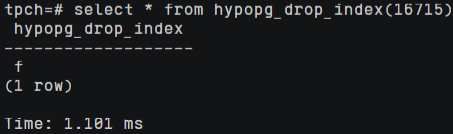
GROUP BY ad.province

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



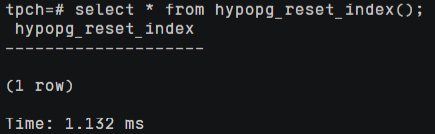
6. 删除某一个索引虚拟列，将执行结果截图。

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



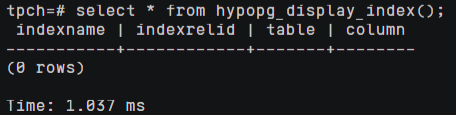
7. 删除某一个索引虚拟列，将执行结果截图。

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？

对max\_connections，shared\_buffers，effective\_io\_concurrency，wal\_buffers，effective\_cache\_size，default\_statistic\_target等参数进行了优化。对这些参数优化能够在有限的资源中更大限度地发挥资源能力，并能够有效减少不必要的损耗。

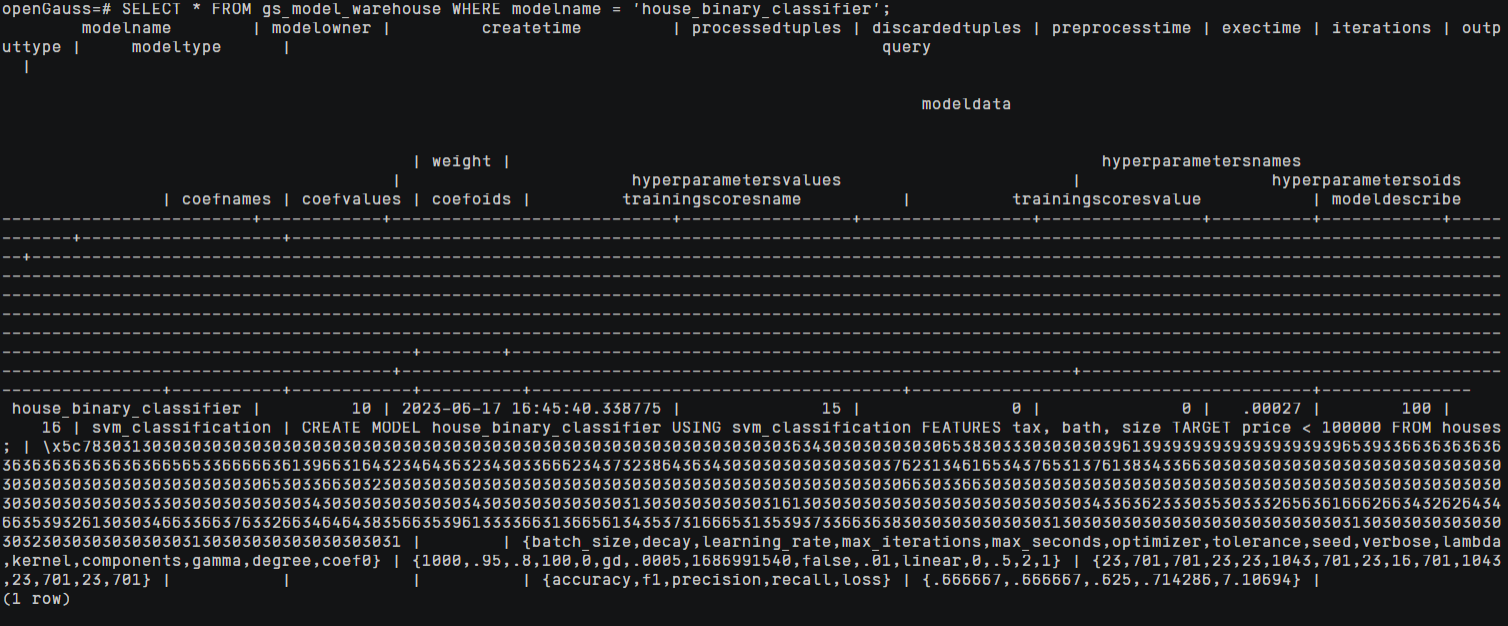
实践思考题2：索引的使用，对于执行SQL有什么好处？除了使用索引和参数外，还有哪些方面可以对数据库进行优化？

能够减少执行SQL的时间。

# 关卡四、openGauss的DB4AI特性应用

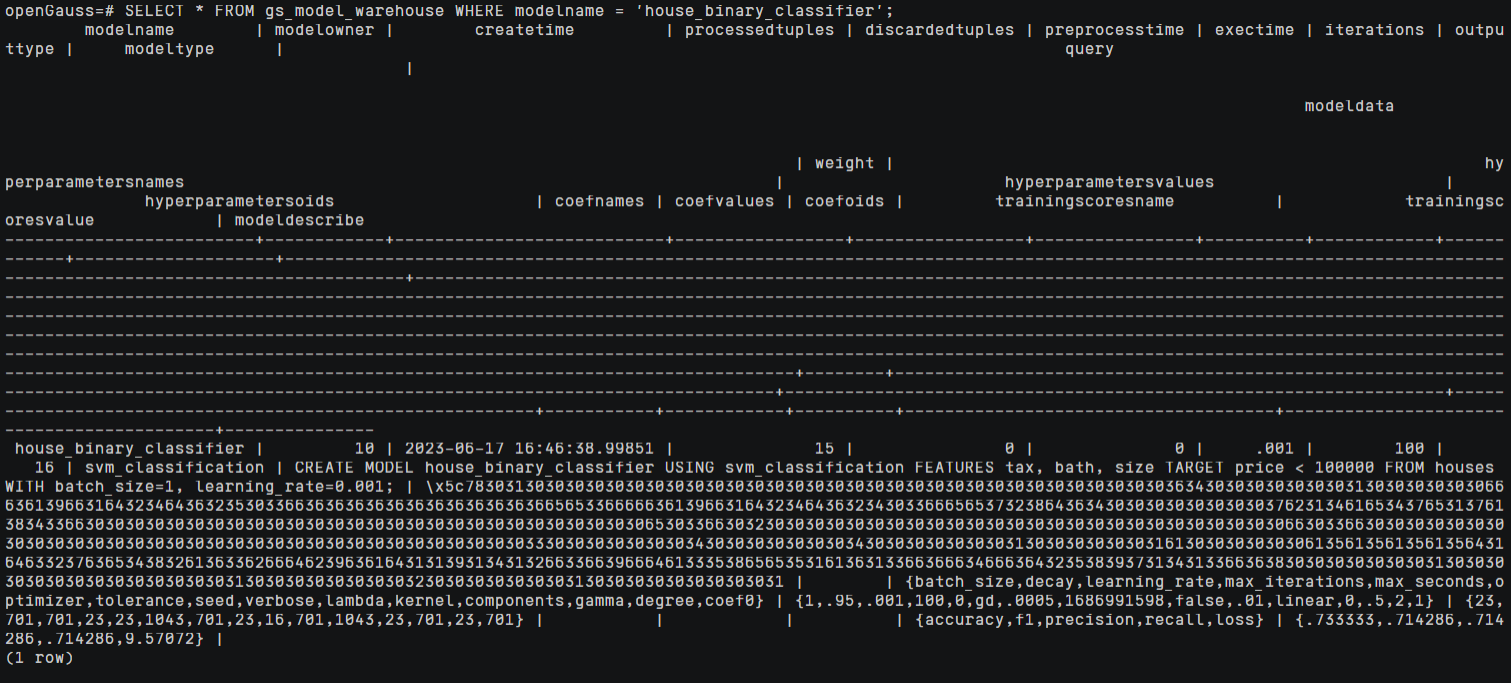
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



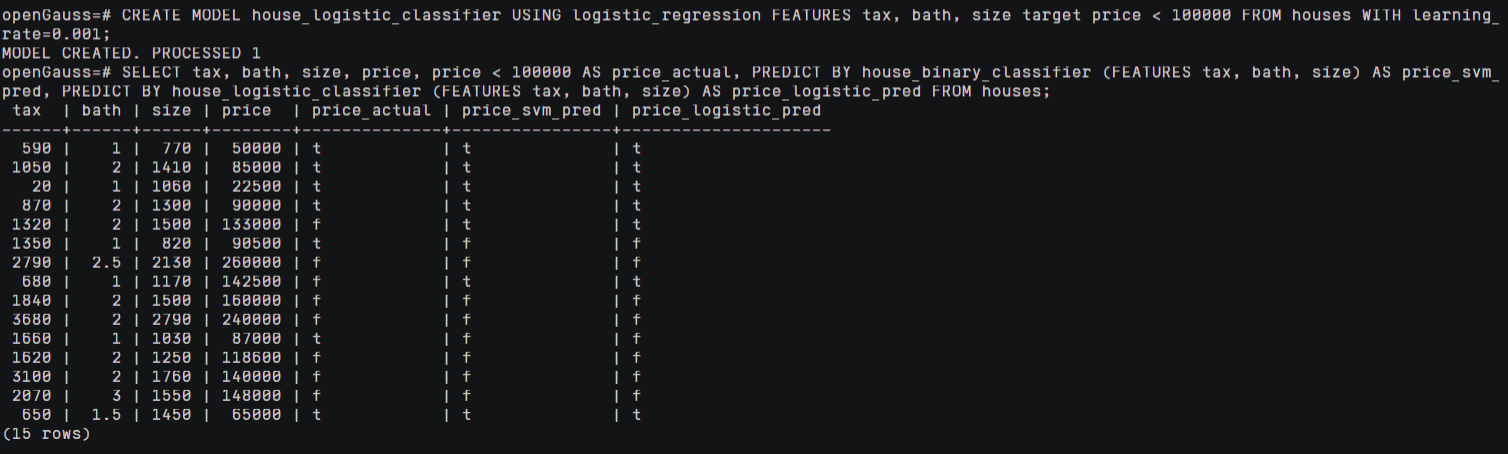
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型用于解决分类问题，其中目标是将输入实例分为不同的离散类别或标签。分类模型的输出是一组预定义的类别之一。例如，根据一些特征，将电子邮件分类为垃圾邮件或非垃圾邮件，或将患者的病情分类为良性或恶性肿瘤。

回归模型用于解决回归问题，其中目标是预测一个连续的数值输出。回归模型的输出是基于输入特征的数值。例如，根据房屋的特征（如面积、位置等），预测房屋的价格，或者预测销售量随广告投入的增加而增加的程度。

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。它的主要思想是通过在特征空间中找到一个最优的超平面，将不同类别的样本实例分开，同时最大化与该超平面最近的训练样本之间的间隔，从而达到更好的分类性能。

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？

准确率（Accuracy）：准确率是最简单和直观的评价指标，表示分类正确的样本数量与总样本数量之比。计算公式为：准确率 = (真阳性 + 真阴性) / (真阳性 + 真阴性 + 假阳性 + 假阴性)。准确率高表示分类器的整体性能较好，但它无法区分不同类别的分类错误。

精确率（Precision）：精确率是在所有被分类为正类的样本中，被正确分类为正类的样本数量与被分类为正类的样本总数之比。计算公式为：精确率 = 真阳性 / (真阳性 + 假阳性)。精确率衡量了分类器将负类样本错误分类为正类的风险，较高的精确率表示分类器的误判率较低。

召回率（Recall）：召回率是在所有真实正类样本中，被正确分类为正类的样本数量与真实正类样本总数之比。计算公式为：召回率 = 真阳性 / (真阳性 + 假阴性)。召回率衡量了分类器对正类样本的识别能力，较高的召回率表示分类器能够较好地捕捉到正类样本。

F1 分数（F1-Score）：F1 分数综合考虑了精确率和召回率，是精确率和召回率的调和平均值。计算公式为：F1 分数 = 2 \* (精确率 \* 召回率) / (精确率 + 召回率)。F1 分数综合了精确率和召回率的信息，对于不平衡数据集和分类器性能的综合评估很有用。

混淆矩阵（Confusion Matrix）：混淆矩阵是一个表格，用于展示分类器的预测结果与真实标签之间的对应关系。它包括四个元素：真阳性（True Positive，TP）、真阴性（True Negative，TN）、假阳性（False Positive，FP）和假阴性（False Negative，FN）。混淆矩阵可以用来计算准确率、精确率和召回率，并提供更详细的分类器性能信息。

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（MSE）：均方误差是预测值与真实值之间差异的平方的平均值。计算公式为：MSE = (1/n) \* Σ(yᵢ - ŷᵢ)²，其中 n 是样本数量，yᵢ 是真实值，ŷᵢ 是预测值。MSE越小表示模型的预测能力越好。

均方根误差（RMSE）：均方根误差是均方误差的平方根，它对误差的量纲进行了恢复，与原始目标变量的单位一致。计算公式为：RMSE = √(MSE)。与MSE类似，RMSE越小表示模型的预测能力越好。

平均绝对误差（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。计算公式为：MAE = (1/n) \* Σ|yᵢ - ŷᵢ|。MAE衡量了模型预测的平均误差大小，与原始目标变量的单位一致。

决定系数（R-squared）：决定系数是用于评估回归模型拟合程度的指标，表示因变量的方差能够由自变量解释的比例。决定系数的取值范围在0和1之间，越接近1表示模型对目标变量的解释能力越强，越接近0表示模型拟合较差。计算公式为：R² = 1 - (Σ(yᵢ - ŷᵢ)² / Σ(yᵢ - ȳ)²)，其中 yᵢ 是真实值，ŷᵢ 是预测值，ȳ 是真实值的平均值。