openGauss AI特性创新实践课



华为技术有限公司

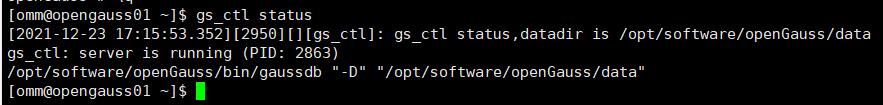
# 关卡一、openGauss数据安装及基本操作

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

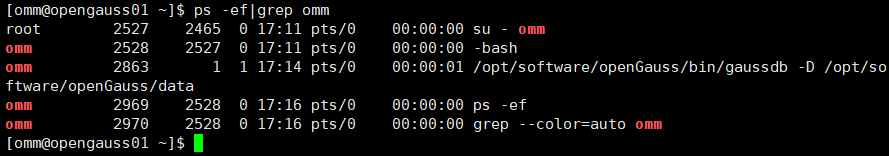
首先需要对数据库状态进行验证: gs\_ctl status



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）

其次，对数据库进程进行截图验证: ps -ef|grep omm



简单描述安装数据库所需要的步骤:

用root权限分组，在omm用户分组下存放openGauss源码及相关文件，通过外部链接导入资源库，依赖包等，配置omm用户下的环境变量，生成配置文件，使用make命令编译安装文件，初始化数据库并预设数据库密码。

思考题：为什么需要通过源码编译，完成数据库的安装？

1.提供一个较好的人机交互过程。习惯于源码编译的用户不习惯只用二进制文件安装数据库。

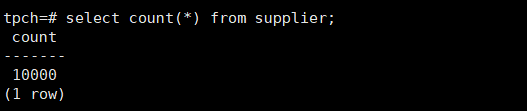
2.可定制。二进制安装只能按照设定好的常规模式选项编译，无法满足特殊的需要

# 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：

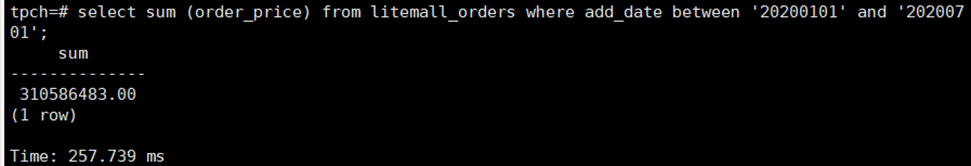
select count(\*) from supplier;;



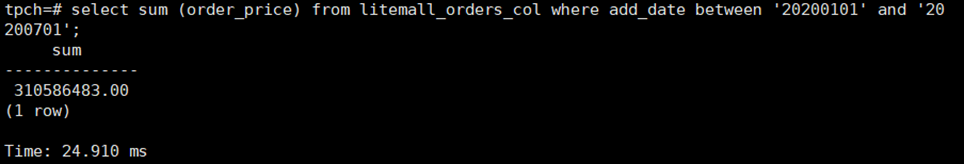
任务二：行存表与列存表执行效率对比

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';

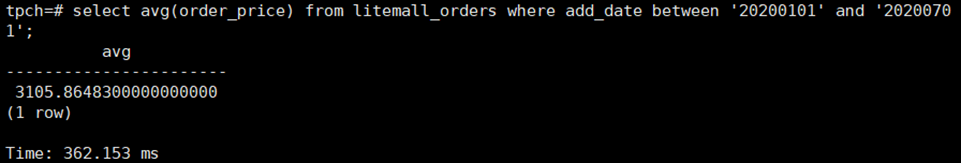


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

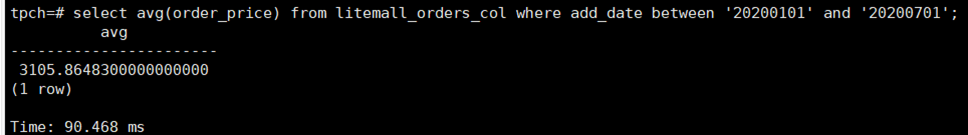


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

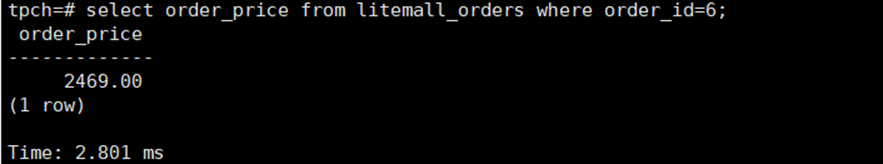


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

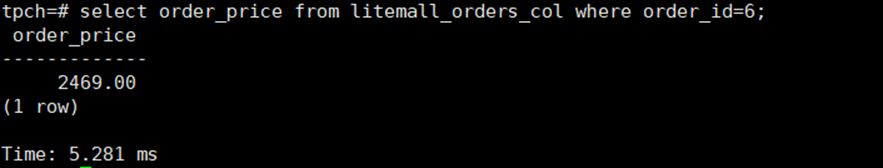


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

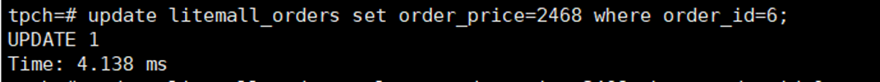


select order\_price from litemall\_orders\_col where order\_id=6;

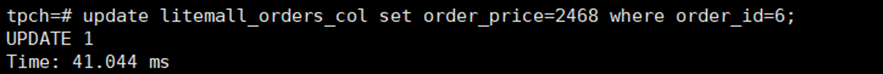


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

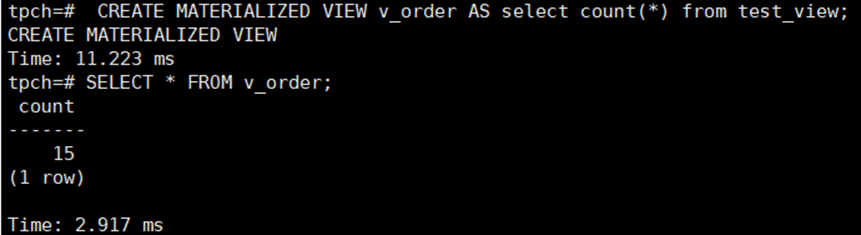
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

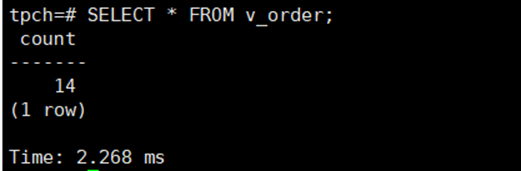
SELECT \* FROM test\_view;



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。

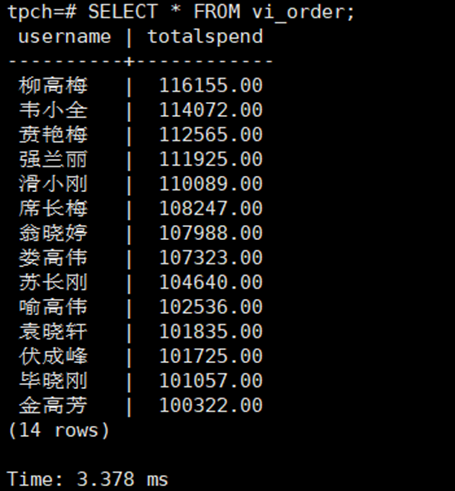
SELECT \* FROM v\_order;





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？

答： 两种表对应两种存储方式，行存储和列存储两种数据存储方式有差异，查询时因为存储结构的不同，执行时间也会不同。

行存储是在指定位置写入一次，就能够保证数据的完整性；列存储需要把一行记录拆分成多个单列进行保存，写入次数比行存储多。行存储通常将一行数据完全读出，列存储每次读取的数据是集合中的一段或者全部。因此读写数据时行存表效率更高。

因为列存表的的数据是同质的，它让数据解析变得容易；而行存表在一行记录中保存了多种类型的数据，数据解析需要在多种数据类型之间频繁地进行转换，因此进行统计查询和数据解析时列存表效率更高。

执行单个值修改和查询时，行存表效率更高；执行总和与平均值等统计查询时，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？

答：全量物化视图：只支持全量刷新物化视图

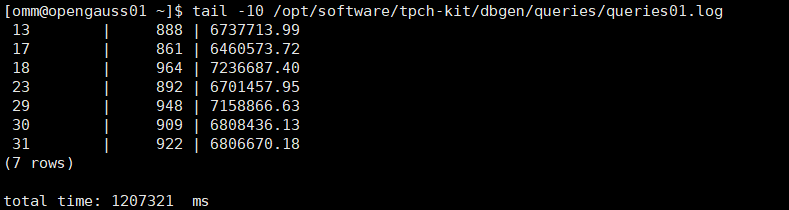
增量物化视图：支持全量刷新和增量刷新物化视图

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

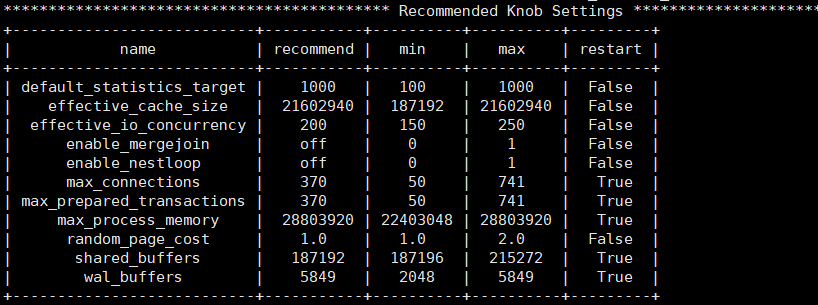
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

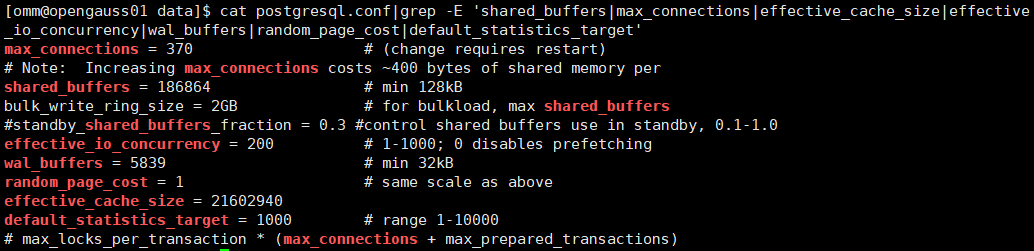
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

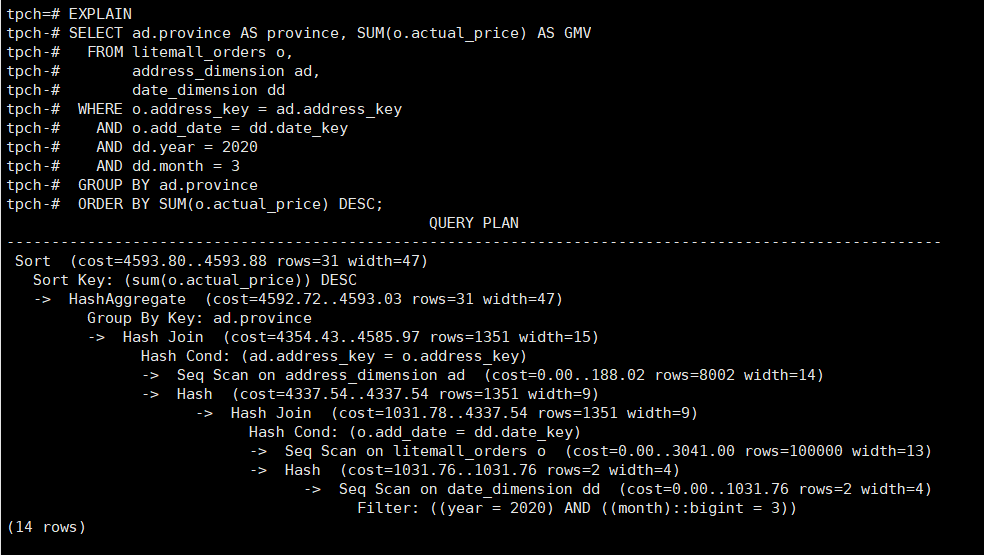
AND o.add\_date = dd.date\_key

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

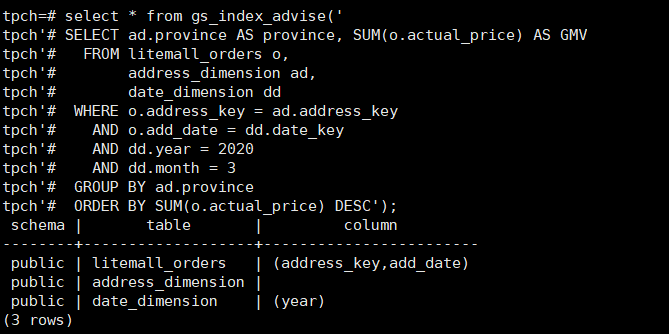
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

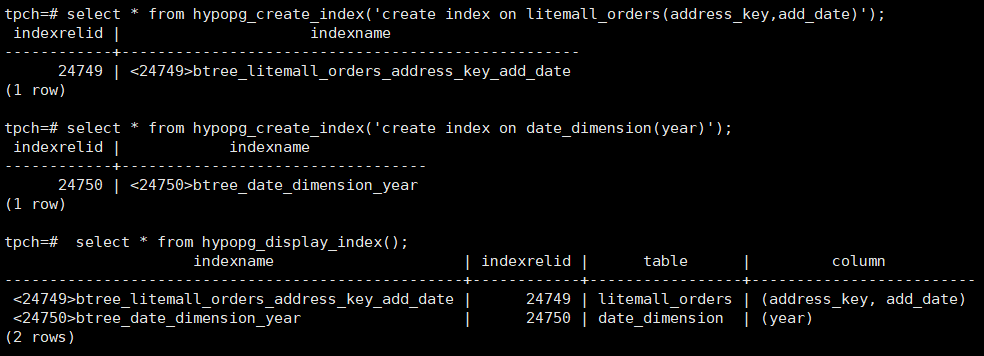
GROUP BY ad.province

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。

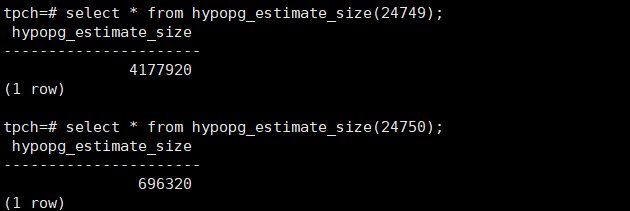
select \* from hypopg\_display\_index();



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

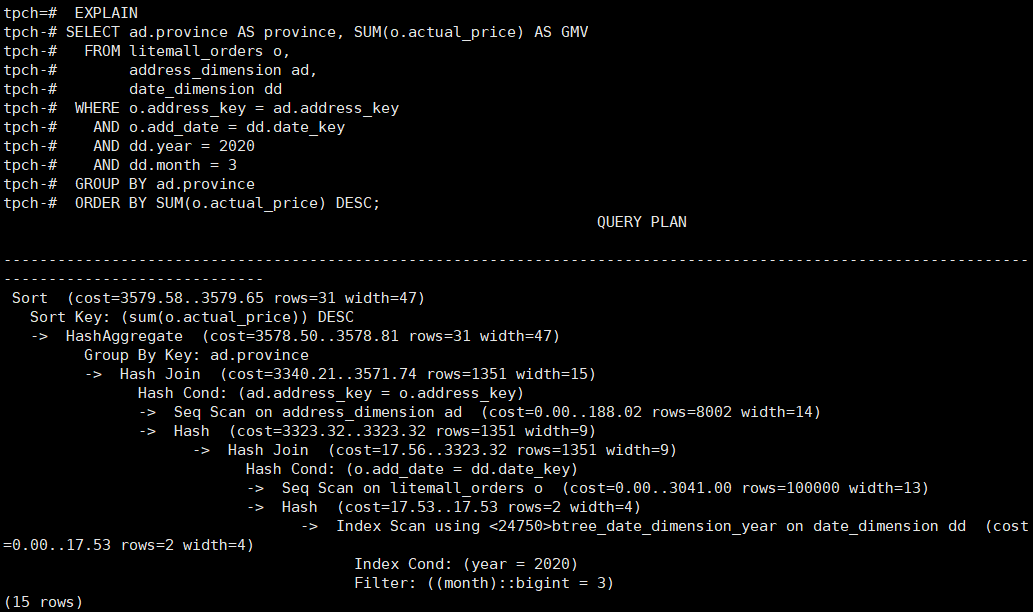
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

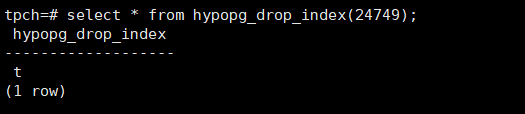
GROUP BY ad.province

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



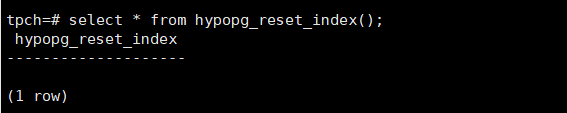
6. 删除某一个索引虚拟列，将执行结果截图。

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



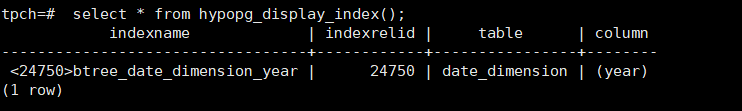
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

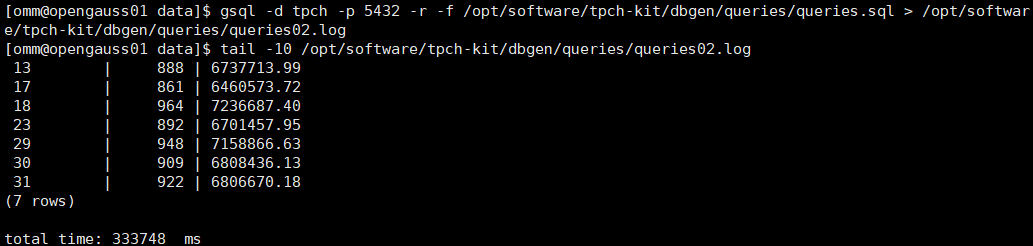
select \* from hypopg\_display\_index();



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？

答：优化的参数有：shared\_buffers ；max\_connections ；effective\_cache\_size ；effective\_io\_concurrency；wal\_buffers ；random\_page\_cost；default\_statistics\_target

原因：

1. 所有数据库的总大小小于内存大小。因此，没有必要将共享缓冲区设置为大值。

2. 检测您指定的工作负载似乎不是TP工作负载，因此禁用启用\u mergejoin更好。

3. wal\_缓冲区的值有点高。通常，过大的值不会带来更好的性能。也可以将此参数设置为-1。数据库自动执行自适应。

4. CPU内核的数量有点少。不要运行太高的并发性。建议您根据CPU资源的数量设置最大连接数。如果您的作业/工作负载不消耗太多CPU，也可以增加它。

5. 检测指定的工作负载似乎不是TP工作负载，因此禁用enable\_nestloop更好。

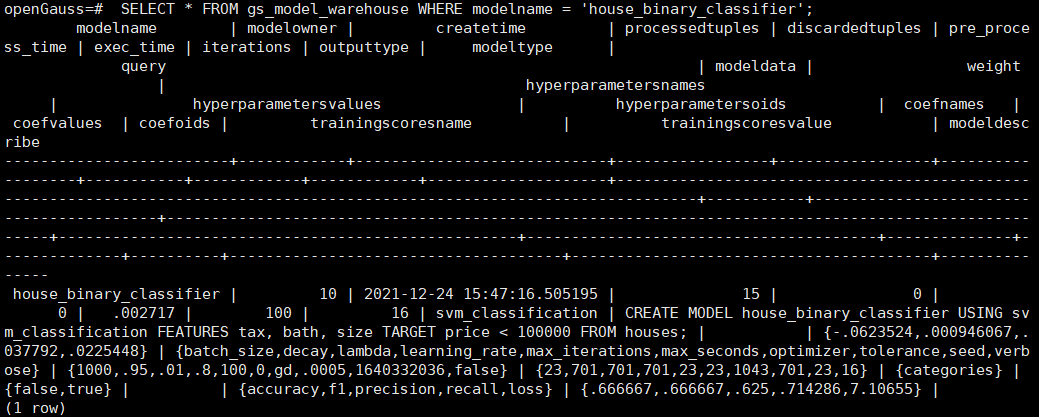
实践思考题2：索引的使用，对于执行SQL有什么好处？除了使用索引和参数外，还有哪些方面可以对数据库进行优化？

答：索引提供指向存储在表的指定列中的数据值的指针，然后根据您指定的排序顺序对这些指针排序。数据库使用索引以找到特定值，然后顺指针找到包含该值的行。这样可以使对应于表的SQL语句执行得更快，可快速访问数据库表中的特定信息。还有分区表可以对数据库进行优化。

# 关卡四、openGauss的DB4AI特性应用

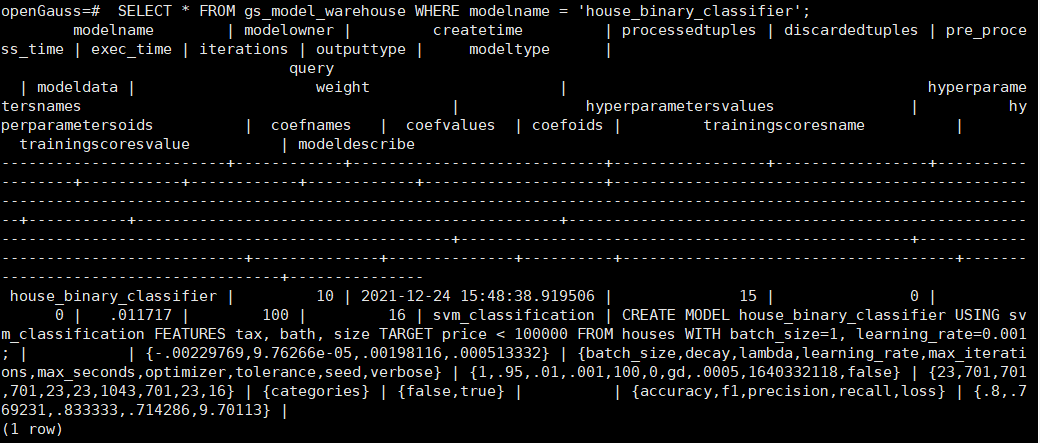
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



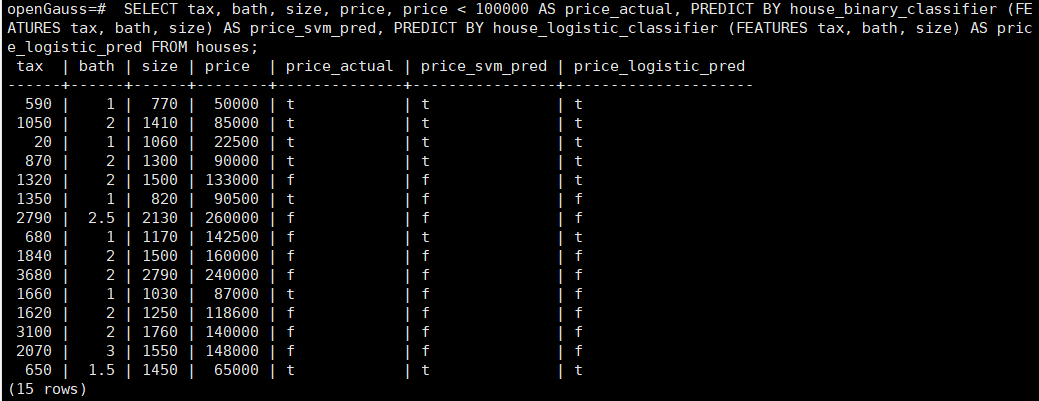
任务二：观察新模型的信息，将执行结果截图。

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？

答：分类：输出为离散值，也就是预测该数据的类别

回归：输出为连续值，也就是预测一个数值

实践思考题2：什么是SVM算法？

答：支持向量机是一种二分类模型，是定义在特征空间上的间隔最大的线性分类器。支持向量机的核心思想是间隔最大化。使用核函数可以实现非线性分类。

对于输入空间中的非线性分类问题，可以通过非线性变换将它转化为某个维特征空间中的线性分类问题，在高维特征空间中学习线性支持向量机。由于在线性支持向量机学习的对偶问题里，目标函数和分类决策函数都只涉及实例和实例之间的内积，所以不需要显式地指定非线性变换，而是用核函数替换当中的内积。

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？

答：Accuracy准确率 含义 ：(TP+TN)/(TP+FN+FP+TN)

Precision精确率 含义 ：TP/(TP+FP)

Recall召回率 含义 ：TP/(TP+FN)

P-R曲线：描述精确率/召回率变化的曲线

F1 分数：精确率和召回率的调和平均数

混淆矩阵：以矩阵形式将数据集中的记录按照真实的类别与分类模型作出的分类判断两个标准进行汇总。

ROC：受试者工作特征曲线

AUC：ROC曲线下的面积

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？

答：MSE均方误差：描述了样本真实值与预测值差方求和的平均值

RMSE ：对MSE开平方根

MAE平均绝对误差：真实值与预测值差的绝对值和求平均

MAPE：所有样本真实值与预测值差绝对值与真实值的比例的和求平均

MSPE：所有样本的真实值与预测值的差与真实值的比例求平方的和求平均