openGauss AI特性创新实践课



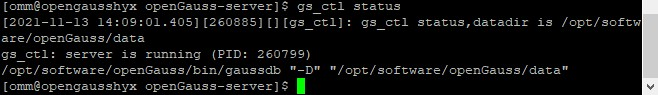
华为技术有限公司

# 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：

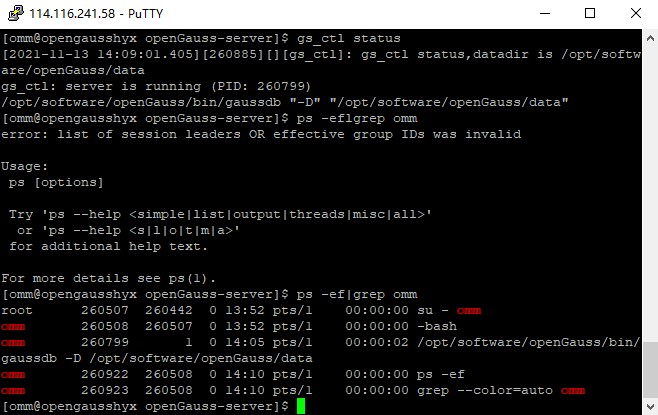
任务一：数据库状态验证

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？

因为openGauss是免费、开源数据库，采用木兰宽松许可证v2发行。

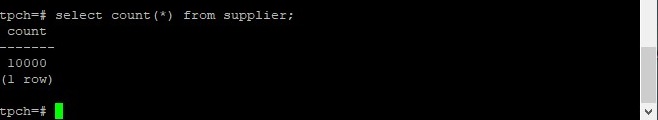
通过源码编译来安装数据库，使用者可以根据自己的需求，修改openGauss源码，定制自己所需功能；同时，openGauss鼓励社区贡献、合作，使用者若发现openGauss的bug或者缺少的功能，可以在源码上找到bug相关代码并修改，以及添加api功能代码，经测试无误后可以提交至openGauss的远端库，为openGauss社区贡献。

# 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：

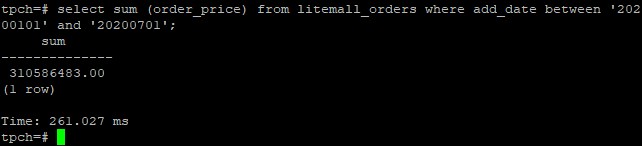
select count(\*) from supplier;;



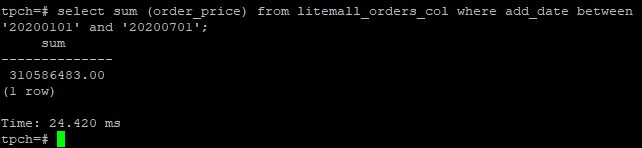
任务二：行存表与列存表执行效率对比

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';

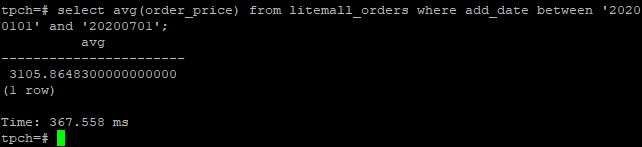


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

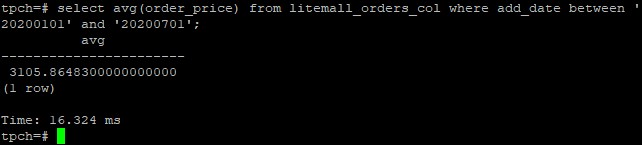


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

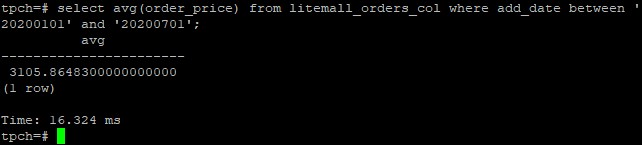


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

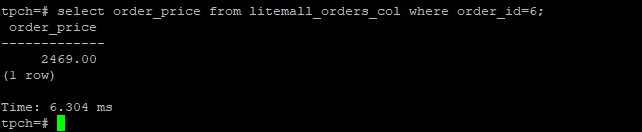


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

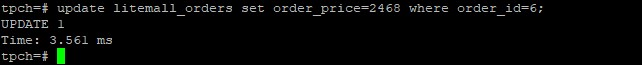


select order\_price from litemall\_orders\_col where order\_id=6;

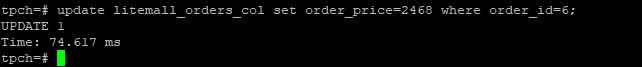


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



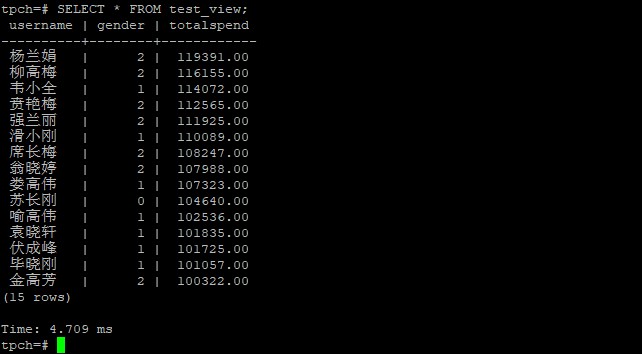
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

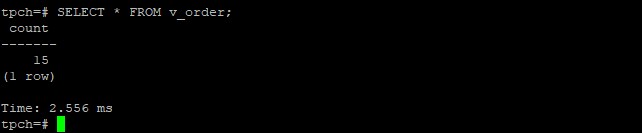
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



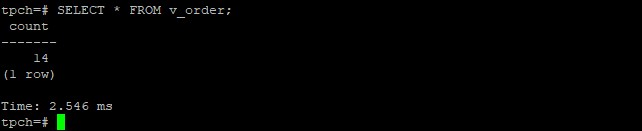
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



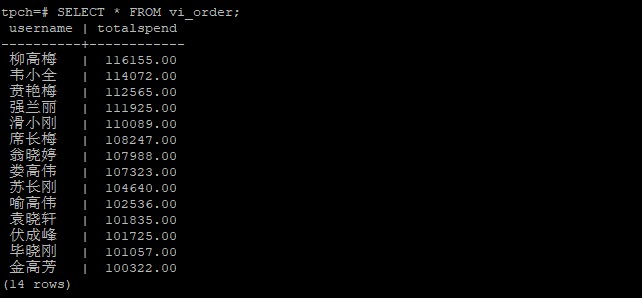
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



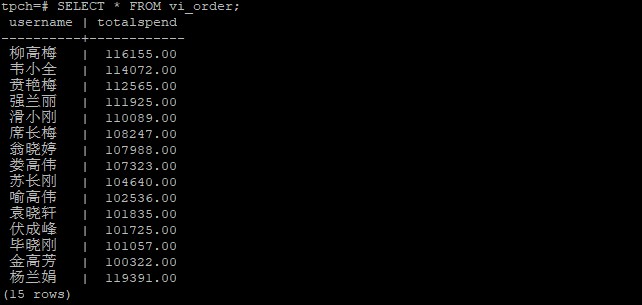
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

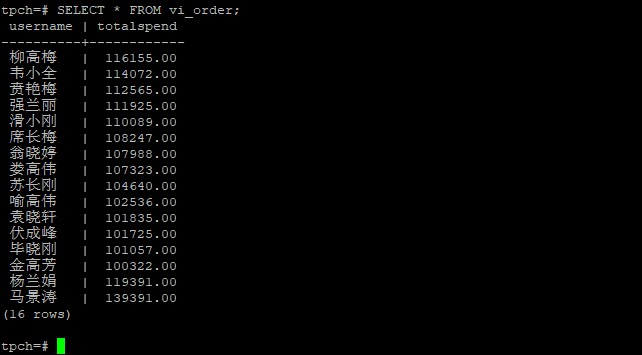
SELECT \* FROM vi\_order;



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？

因为行存表和列存表的存储模式不同，行存表数据按行进行存储，读取任意列的成本不同，读取的列越靠后，付出的成本越高；列存表按列进行存储，读取任意列的成本相同，不同列数据存储在不同文件内，因此读取多个列的情况付出的成本会增高

行存表在表的字段个数比较少，查询大部分字段、数据更新频繁、频繁少量插入数据的情况下效率更高；

列存表在数据批量插入、更新较少、以查询为主、列数很多的情景下效率更高

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量刷新机制是首先物化视图对应表中的数据采用delete全部删除，然后再从原表中使用insert把数据重新插入。

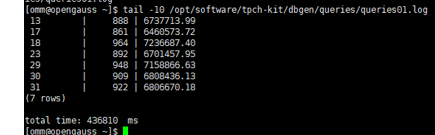
增量刷新：增量刷新是基于主表上的物化视图日志进行刷新的，主表上每插入或删除一条数据，对应物化视图日志中同样会插入一条数据（主表更新一条数据，对应物化视图中会插入两条记录数据），物化视图刷新后主表上物化视图日志记录信息会被清空，重新开始记录后面的更新

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

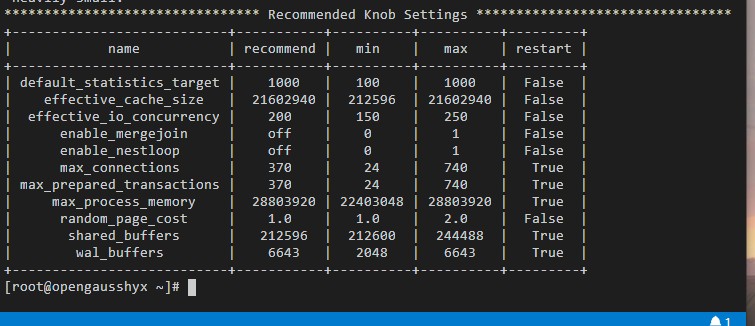
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

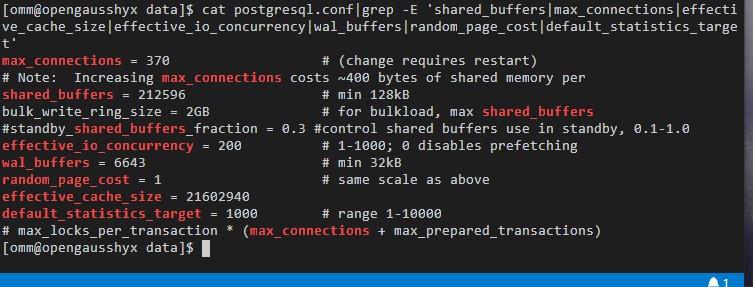
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

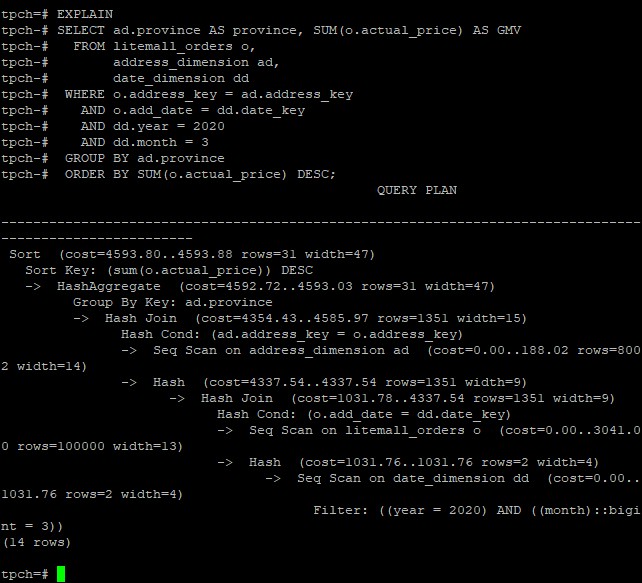
AND o.add\_date = dd.date\_key

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

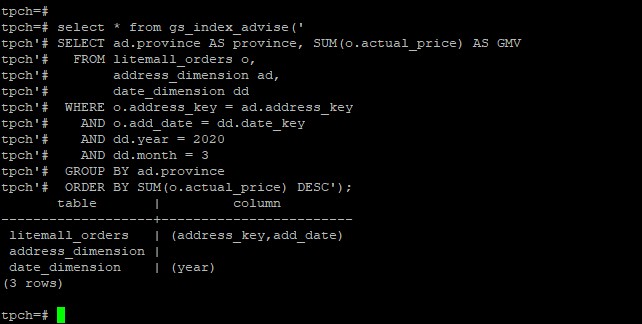
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

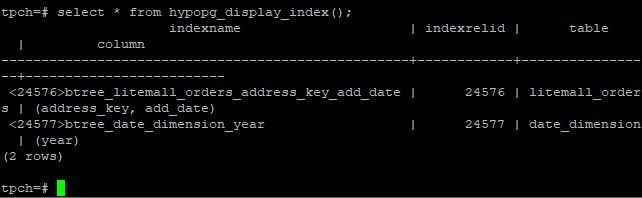
GROUP BY ad.province

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。

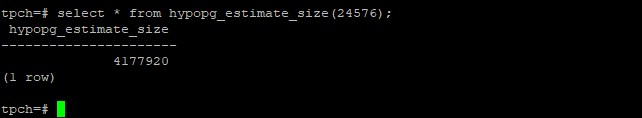
select \* from hypopg\_display\_index();

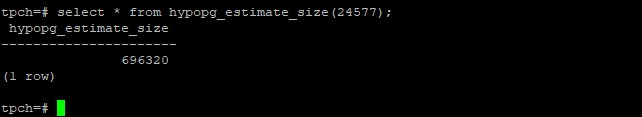


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

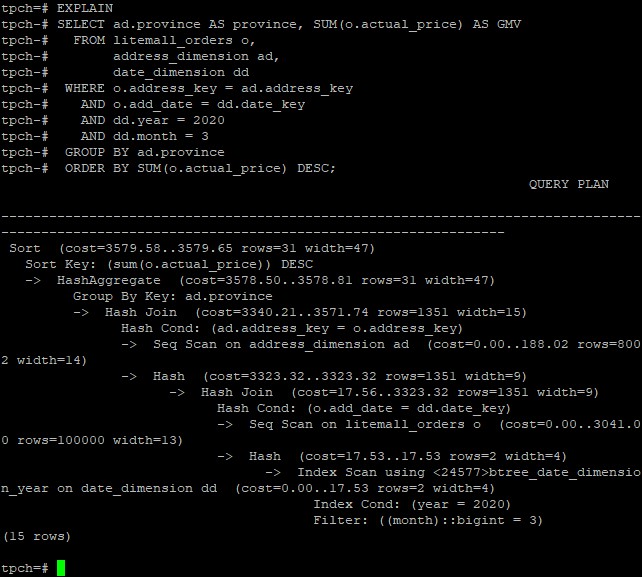
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

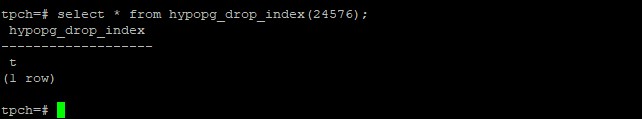
GROUP BY ad.province

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



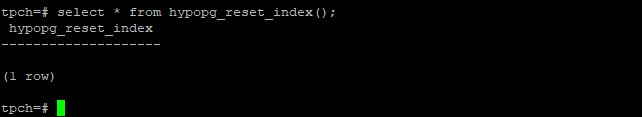
6. 删除某一个索引虚拟列，将执行结果截图。

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



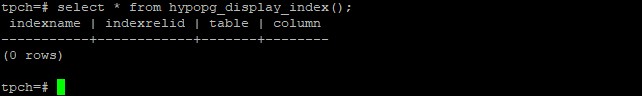
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

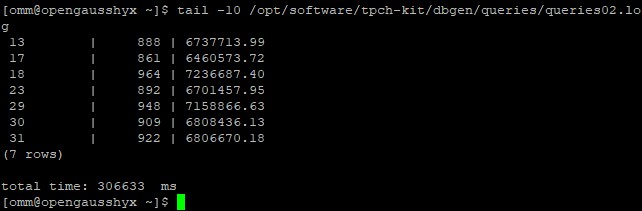
select \* from hypopg\_display\_index();



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？

在shared\_buffers、effective\_cache\_size、wal\_buffers参数上进行了优化，通过AI计算得到这些参数的更合适的值，使用这些值可以使数据库效率更好

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？

好处：

a）可以保证数据库表中每一行的数据的唯一性  
b）加速表与表之间的连接   
c）在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间  
d）通过使用索引，可以在时间查询的过程中，使用优化隐藏器，提高系统的性能

优化：

a）利用数据库特性，例如openGauss支持列存表，对于适合使用列存表的表，使用列存表存储后，数据库查询、插入等效率会变高

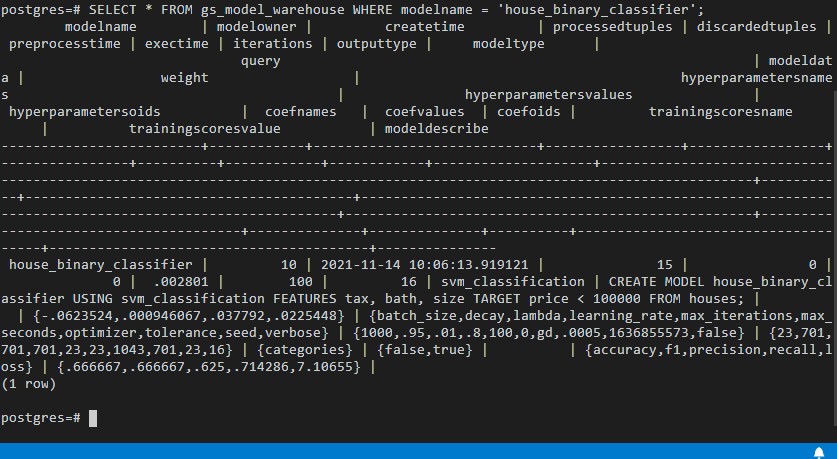
b）使用连接代替子查询，使用连接操作，不需要在内存中创建临时表来完成，且sql语句的可读性会变高

c）优化查询语句，减少冗余

# 关卡四、openGauss的DB4AI特性应用

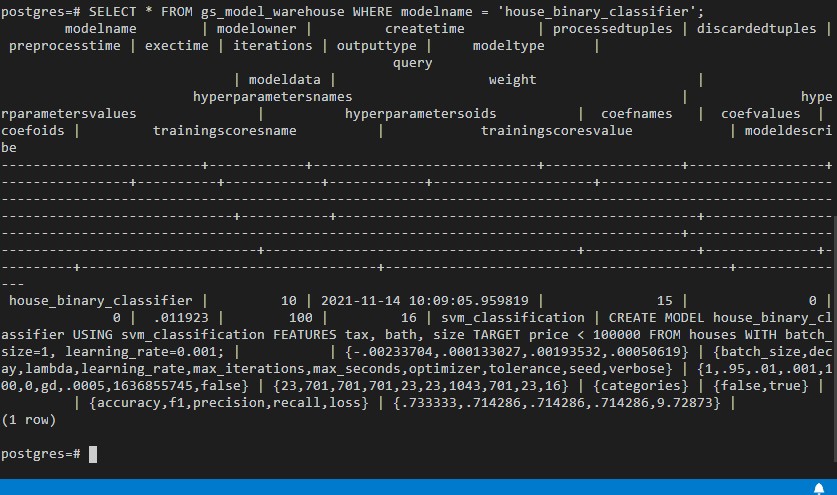
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



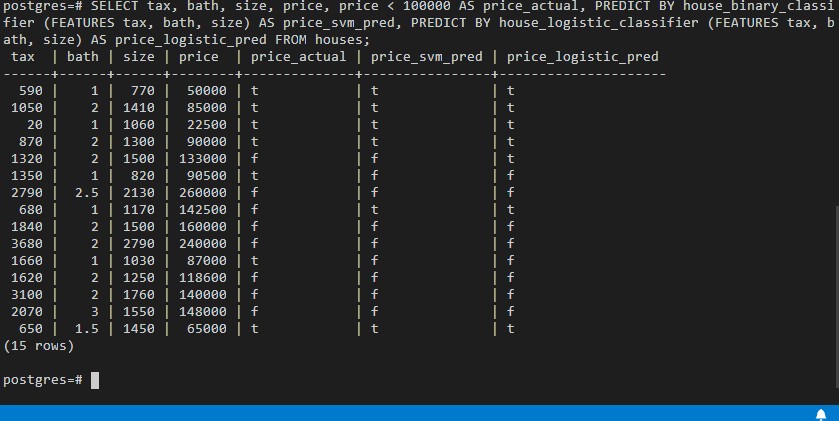
任务二：观察新模型的信息，将执行结果截图。

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:

概念：对于分类问题，监督学习从数据中学习一个分类模型或者分类决策函数，称为分类器。分类器对新的输入预测其属于哪一类别，称为分类。  
优化过程：找到最优决策面  
输出：离散值，如0/1，yes/no

回归:

概念：回归用于预测输入变量和输出变量之间的关系，特别是当输入变量的值发生变化时，输出变量的值也会跟着变化。回归模型正是表示输入变量到输出变量之间的映射函数，回归问题的学习等价于函数拟合。  
优化过程：使得学习到的函数曲线跟真实曲线拟合的最好  
输出：连续值，如房子的售价，天气温度

实践思考题2：什么是SVM算法？

支持向量机（support vector machines）是一个二分类的分类模型（或者叫做分类器）。

它分类的思想是，给定给一个包含正例和反例的样本集合，svm的目的是寻找一个超平面来对样本根据正例和反例进行分割。

各种资料对它评价甚高，说“ 它在解决小样本、非线性及高维模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中”

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？

a）准确率：预测正确的样本数量占总量的百分比

b）精准率：又称为查准率，是针对预测结果而言的一个评价指标。在模型预测为正样本的结果中，真正是正样本所占的百分比

c）召回率：又称为查全率，是针对原始样本而言的一个评价指标。在实际为正样本中，被预测为正样本所占的百分比

d）F1 Score：针对精准率和召回率都有其自己的缺点；如果阈值较高，那么精准率会高，但是会漏掉很多数据；如果阈值较低，召回率高，但是预测的会很不准确

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？

a）MSE：均方误差

MSE是真实值与预测值的差值的平方然后求和平均。通过平方的形式便于求导，所以常被用作线性回归的损失函数。

b）RMSE：均方根误差

衡量观测值与真实值之间的偏差。常用来作为机器学习模型预测结果衡量的标准。

c）MAE：平均绝对误差

是绝对误差的平均值。可以更好地反映预测值误差的实际情况。

d）SD：标准差

方差的算术平均根。用于衡量一组数值的离散程度。