openGauss AI特性创新实践课



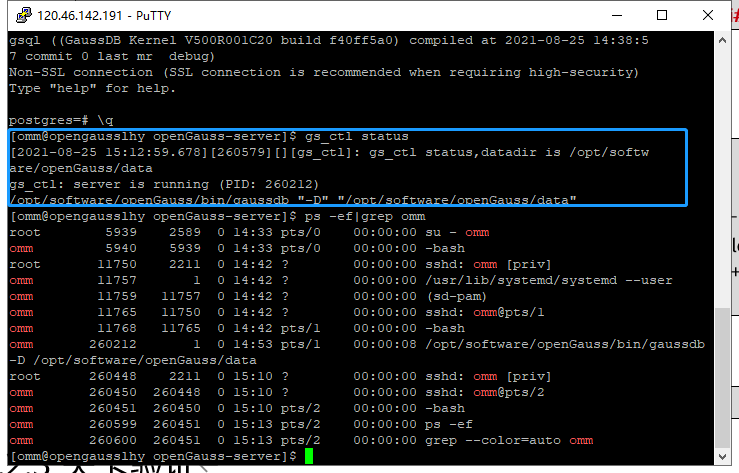
华为技术有限公司

# 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：

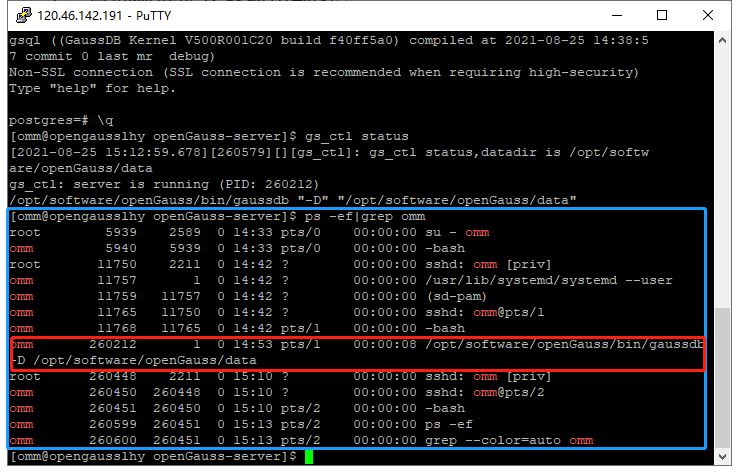
任务一：数据库状态验证

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？

（1）为了满足不同的运行平台，Linux发型版本众多，但是每个版本采用的软件或者内核版本或系统中依赖库的版本都不一样。因此直接提供编译好的二进制包，如果系统更新的话，可能导致Gauss数据库的二进制包所依赖的环境不一定能够正常运行，所以在确定了我们的Linux系统为OpenEuler(arm版)之后，直接提供对应OpenEuler系统的源码编译，可以保证在规定的版本的LINUX系统上正常运行。

（2）方便我们根据自己的需求选择安装的包，直接打包安装可能有很多我们不需要的包。

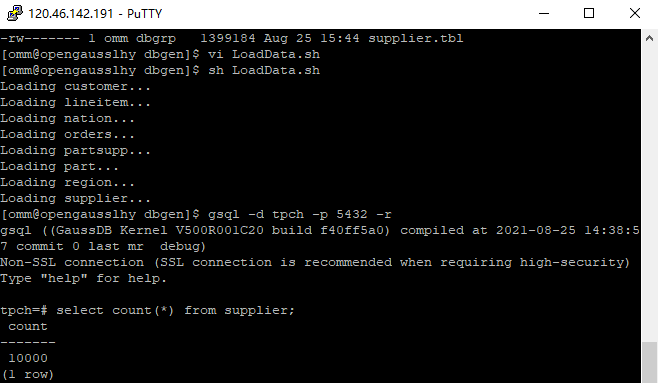
（3）减少工作量，方便运维、开发人员维护，我们的源码是可以打包二进制的，但是对于这个软件的打包都会有一份代价不小的额外工作，包括维护，所以如果是开源源码的话，软件产商会直接维护，二进制文件的打包问题则交给Linux厂商处理。

# 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：

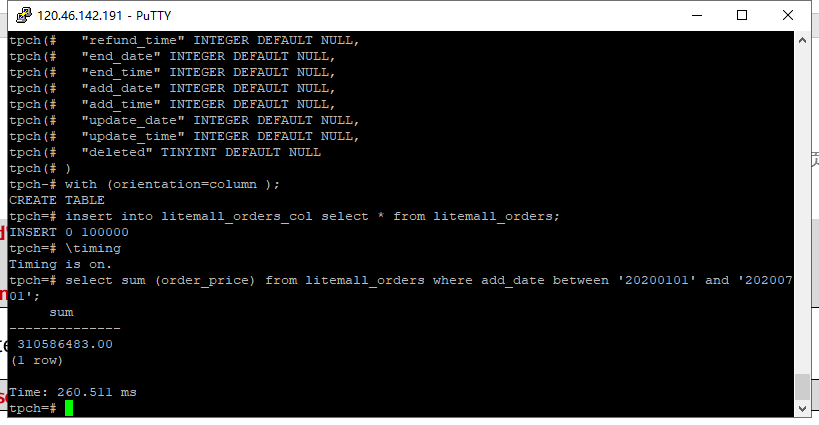
select count(\*) from supplier;;



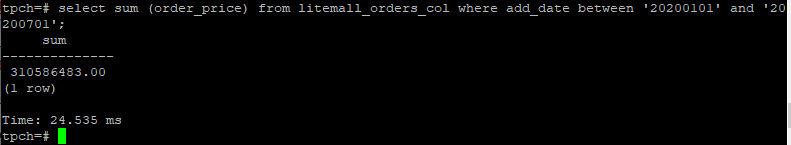
任务二：行存表与列存表执行效率对比

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';

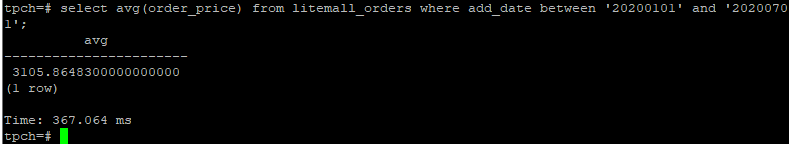


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

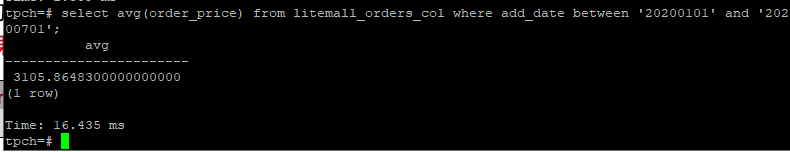


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

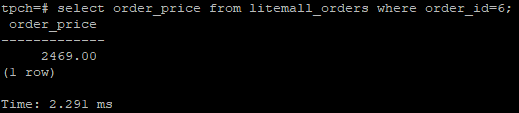


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

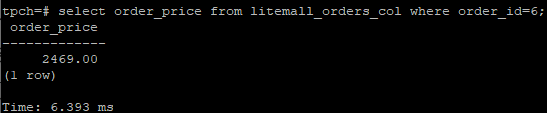


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

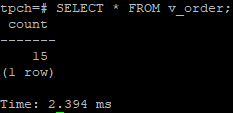
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



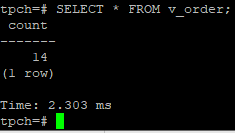
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

（1）数据读取时，行存储通常将一行数据完全读出，如果只需要其中几列数据的情况，就会存在冗余列；列存储每次读取的数据是集合的一段或者全部，不存在冗余性问题。行存储的写入是一次完成；列存储由于需要把一行记录拆分成单列保存，写入次数明显比行存储多。

（2）

①在执行数据修改操作，如数据的增、删、改时，行存储效率更高。

因为这实际也是一次写入过程。数据修改是对磁盘上的记录做删除标记。行存储是在指定位置写入一次，列存储是将磁盘定位到多个列上分别写入，这个过程仍是行存储的列数倍。所以，数据修改也是以行存储占优。

②在执行数据读取（查）和对数据完整性要求不高的大数据处理领域（OLAP）列存储占优。

列存储优势在于在读取过程，不会产生冗余数据，查询过程中，可针对各列的运算并发执行(SMP)，;可在数据列中高效查找数据，无需维护索引(任何列都能作为索引)，查询过程中能够尽量减少无关IO，避免全表扫描;因为各列独立存储，且数据类型已知，可以针对该列的数据类型、数据量大小等因素动态选择压缩算法，以提高物理存储利用率;如果某一行的某一列没有数据，那在列存储时，就可以不存储该列的值，这将比行式存储更节省空间。

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。

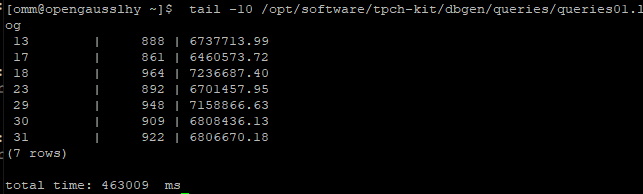
增量物化视图顾名思义就是可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。与全量创建物化视图不同在于目前增量物化视图所支持场景较小，目前物化视图创建语句仅支持基表扫描语句或者UNION ALL语句。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

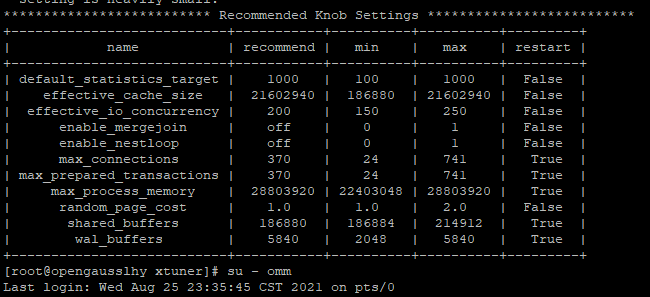
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

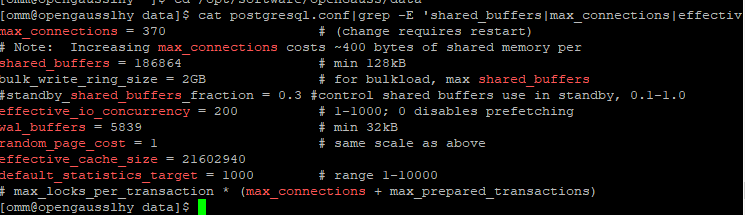
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

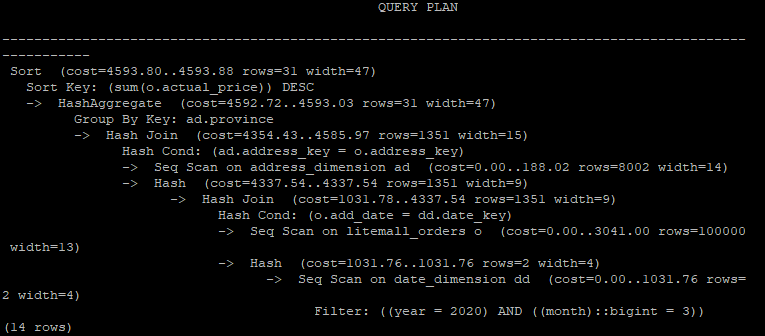
AND o.add\_date = dd.date\_key

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

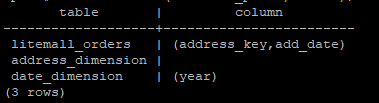
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

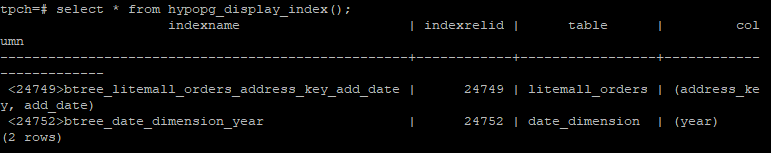
GROUP BY ad.province

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。

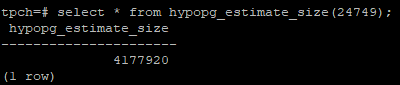
select \* from hypopg\_display\_index();

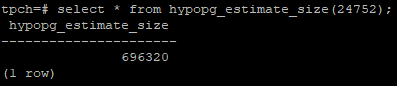


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

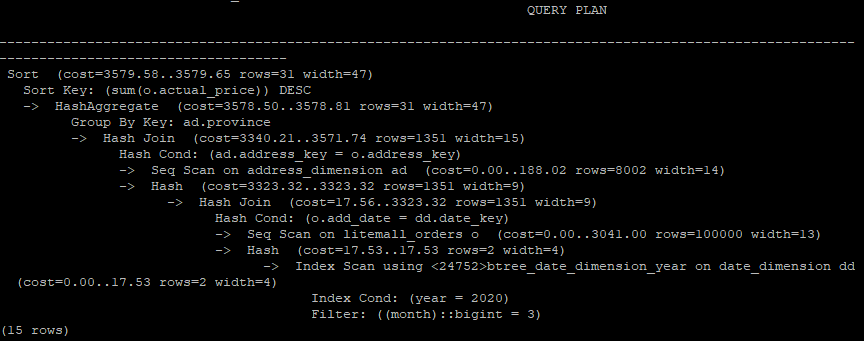
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

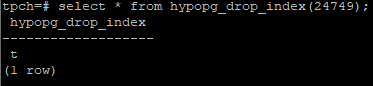
GROUP BY ad.province

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



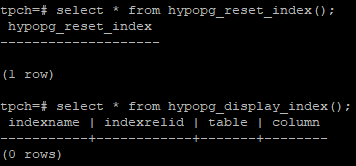
6. 删除某一个索引虚拟列，将执行结果截图。

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



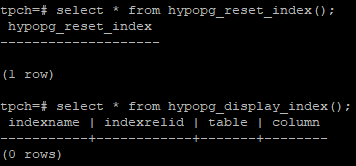
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

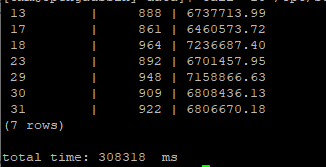
select \* from hypopg\_display\_index();



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。

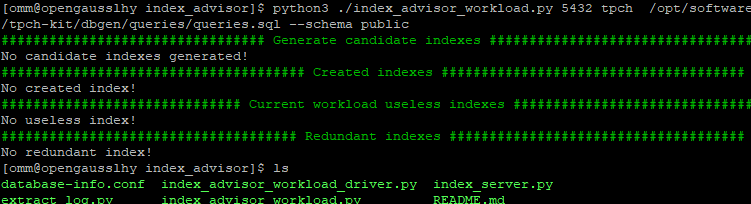
1. 重新执行queries.sql查询，将执行结果截图：

gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries03.log

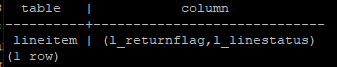
对queries.sql使用workload级别的优化

python3 ./index\_advisor\_workload.py 5432 tpch /opt/software/tpch-kit/dbgen/queries/queries.sql --schema public

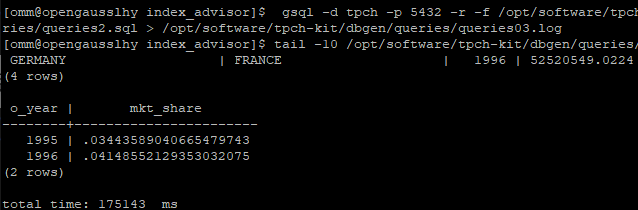
系统推荐不需要索引，因为数据量不大，不需要索引也能很好的完成任务



所以针对queris.sql中的部分复杂语句进行优化，推荐了如下索引：



设置索引后，重新执行，前后执行时间出现1倍左右的的提升



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？

1. shared\_buffers; max\_connections; effective\_cache\_size; effective\_io\_concurrency;

wal\_buffers; random\_page\_cost; default\_statistics\_target

1. 对这些参数优化，是为了提升数据库的性能表现。

如当connections过多时，如果max\_connections过小，就会导致出现too many connections等错误；当connections过少，而max\_connections过大，则会导致申请的文件扫描符数量过多，产生冗余。所以对数据库的各项参数进行合理的优化，是很有必要的，通过X-Tuner对数据库的参数监控，可以发现当前数据库执行sql语句时，各项参数的最大，最小值等，并根据参数的变化，推荐一个最合适的参数调优值，使得数据库的性能处于最佳状态。

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？

1. 可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

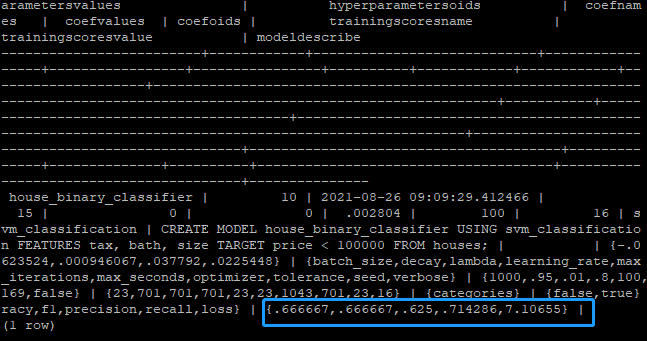
此外，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性；也可以加速表和表之间的连接；在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间；可以在查询的过程中，使用优化隐藏器，提高系统的性能。

1. 除了使用索引和参数外，还可以：从硬件层面优化，cpu和网络等进行优化；优化查询语句本身，通过Explain和Describe关键字分析select查询语句，从而使开发人员知道查询效率低的原因；也可以优化数据库本身的结构，如优化字段类型、字符编码、适当拆分、优化数据库的表等

# 关卡四、openGauss的DB4AI特性应用

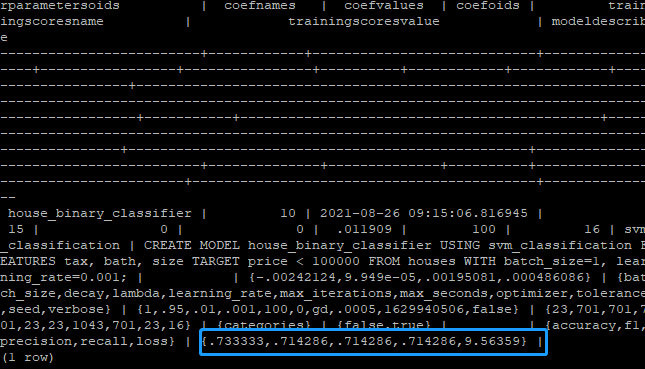
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



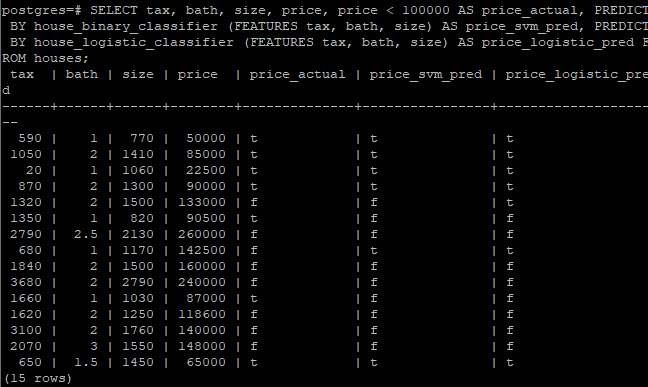
任务二：观察新模型的信息，将执行结果截图。

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;



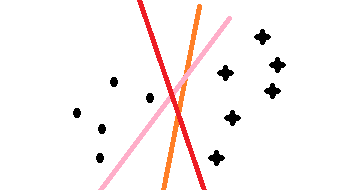
实践思考题1：分类模型与回归模型有何不同？

定量输出称为回归，或者说是连续变量预测；定性输出称为分类，或者说是离散变量预测。分类模型返回的只是一个结果，相当于你提问题，它只会告诉你是或者不是，比如他只会告诉你明天是温暖还是寒冷而不会预测出明天具体有多少摄氏度；而回归模型，可以告诉你具体问题的答案，比如预测的明天的温度是28.5摄氏度。

实践思考题2：什么是SVM算法？

支持向量机(SVM) 是一个监督学习算法；在 SVM 算法中，我们将数据绘制在 n 维空间中（n 代表数据的特征数），每个特征数的值是特定坐标的值。然后我们通过查找可以将数据分成两类的超平面。

SVM算法通俗的理解在二维上，就是找一分割线把两类分开，问题是如下图三条颜色都可以把点和星划开，但哪条线是最优的呢，这就是我们要考虑的问题；



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？

**写在前面：**

1、若一个实例是正类并且被预测为正类，即为真正类(True Postive TP)

2、若一个实例是正类，但是被预测成为负类，即为假负类(False Negative FN)

3、若一个实例是负类，但是被预测成为正类，即为假正类(False Postive FP)

4、若一个实例是负类，但是被预测成为负类，即为真负类(True Negative TN)

**评价指标：**

1. 精确率（precision）

precision=TP/(TP+FP)，表示被分为正例的示例中实际为正例的比例。

1. 召回率（recall）

recall=TP/(TP+FN)=TP/P=sensitve，表示的是覆盖面的度量，度量有多个正例被分为正例。

1. 准确率（accuracy）

准确率：被分对的样本数除以所有的样本数，通常来说，正确率越高，分类器越好。  
AUC=(TP+TN)/(TP+TN+FP+FN)

1. 错误率（error）

错误率：error rate = (FP+FN)/(TP+TN+FP+FN).错误率则与准确率相反，描述被分类器错分的比例.对某一个实例来说，分对与分错是互斥事件，所以accuracy =1 - error rate

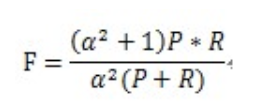
1. 灵敏度(sensitive)

sensitive = TP/P，表示的是所有正例中被分对的比例，衡量了分类器对正例的识别能力。

1. 特效度(specificity)

specificity = TN/N，表示的是所有负例中被分对的比例，衡量了分类器对负例的识别能力。

1. F函数

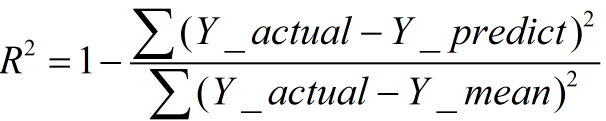
F-Measure是Precision和Recall加权调和平均，可知F1综合了P和R的结果，当F1较高时则能说明试验方法比较有效。

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？  
1）SSE(误差平方和)：The sum of squares due to error

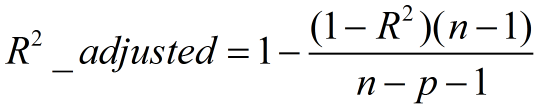


同样的数据集的情况下，SSE越小，误差越小，模型效果越好  
缺点：  
SSE数值大小本身没有意义，随着样本增加，SSE必然增加，也就是说，不同的数据集的情况下，SSE比较没有意义

2) R-square(决定系数)：Coefficient of determination



数学理解： 分母理解为原始数据的离散程度，分子为预测数据和原始数据的误差，二者相除可以消除原始数据离散程度的影响  
其实“决定系数”是通过数据的变化来表征一个拟合的好坏。  
理论上取值范围（-∞，1], 正常取值范围为[0 1] ------实际操作中通常会选择拟合较好的曲线计算R²，因此很少出现-∞  
越接近1，表明方程的变量对y的解释能力越强，这个模型对数据拟合的也较好  
越接近0，表明模型拟合的越差  
3) Adjusted R-square(矫正决定系数)：Degree-of-freedom adjusted coefficient of determination



在R平方中，它描述的是输入变量对输出变量的解释程度，在单变量线性回归中，R平方越大拟合程度越好。可是一旦引入更多的变量，不论增加的变量是否和输出变量存在关系，R平方都会增加，所以这时候就需要校正R平方了。它主要做了一件事，对那些增加的且不会改善模型效果的变量增加了一个惩罚项，这样，如果加入的无关变量越多，校正R平方就会下降，而如果加入了显著相关的变量，就会提高。因此，一般来说，对单变量线性回归会采取R平方，对多变量的情况则使用校正R平方。