openGauss AI特性创新实践课



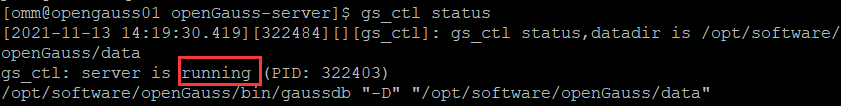
华为技术有限公司

# 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：

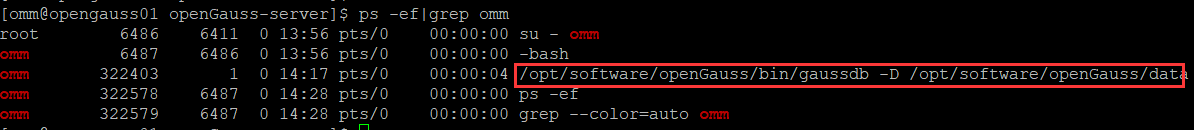
任务一：数据库状态验证

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？

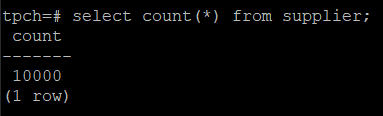
因为openGauss数据库是开源的，我们可以很方便的拿到openGauss源码，并且可以自己针对数据库进行一些小优化，对源码改动，这样编译安装的数据库可以说是我们自定义的openGauss数据库，而不像安装包那样已经将官方源码进行编译安装了。

# 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：

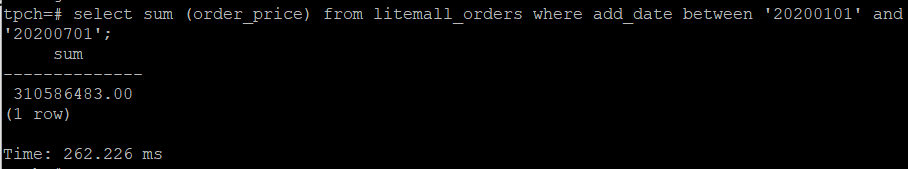
select count(\*) from supplier;;



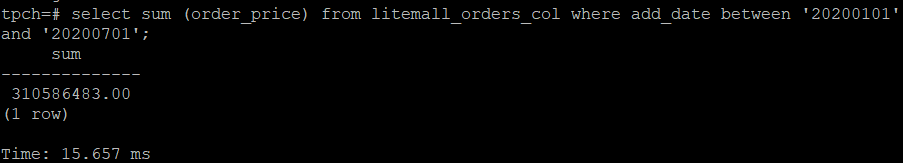
任务二：行存表与列存表执行效率对比

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';

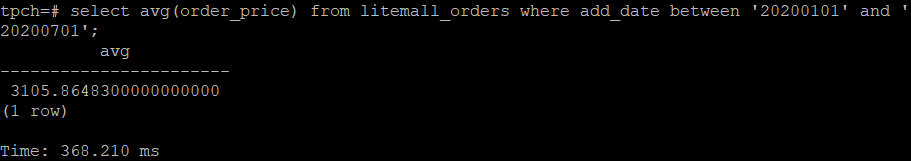


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

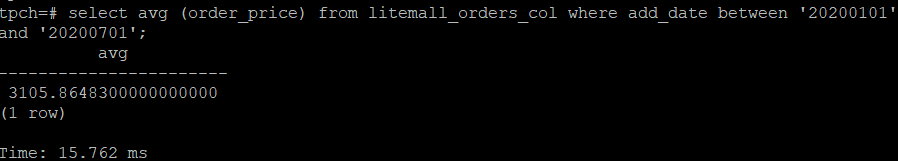


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

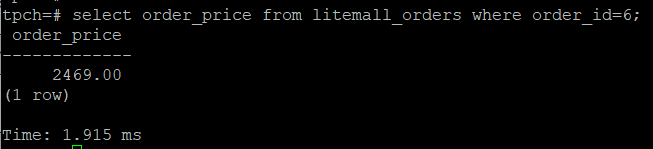


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

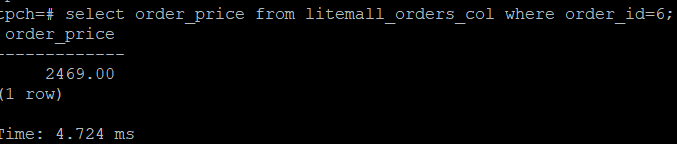


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

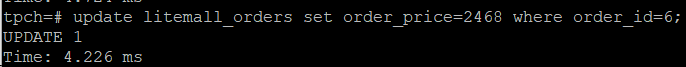


select order\_price from litemall\_orders\_col where order\_id=6;



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

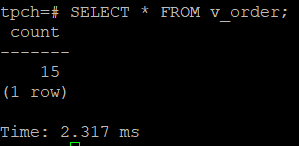
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



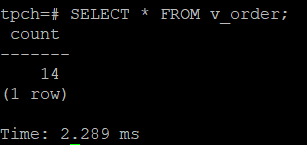
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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;

插入数据后增量刷新



插入数据后全量刷新



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）因为行存表和列存表存放数据的形式不同，行存表一条数据是以行的形式存储，列存表一条数据是以列的形式存储，故在执行相同的SQL语句时执行的时间不同。

（2）行存表执行以下SQL语句时效率更高：

　　① 适合随机的增删改查操作;

　　② 需要在行中选取所有属性的查询操作;

　　③ 需要频繁插入或更新的操作，其操作与索引和行的大小更为相关。

列存表执行大量数据的查询且只涉及少量列的时候效率更高，不过行存表可以通过加索引的方式提高这方面SQL的效率。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

刷新方式的差别：

全量物化视图采用全量刷新，具体是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。

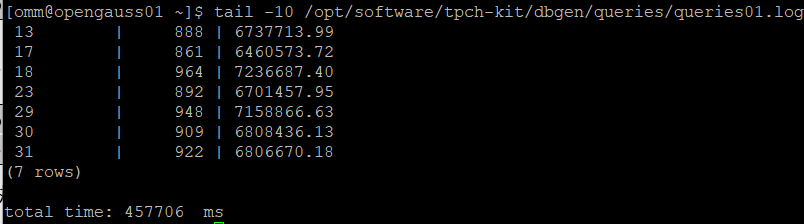
增量物化视图采用增量刷新，具体是基于主表上的物化视图日志进行刷新的，主表上每插入、更新或删除一条数据，对应物化视图日志中同样会记录这些操作，物化视图刷新后主表上物化视图日志记录信息会被清空，重新开始记录后面的对数据的操作。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

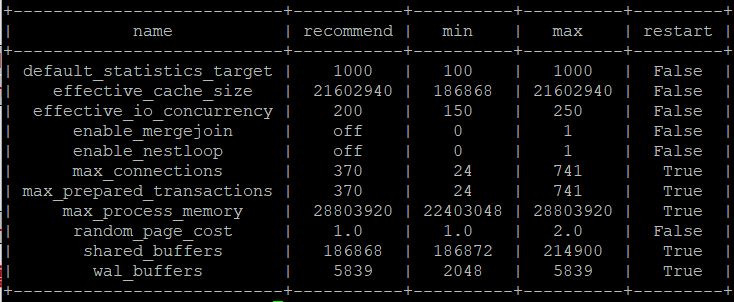
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

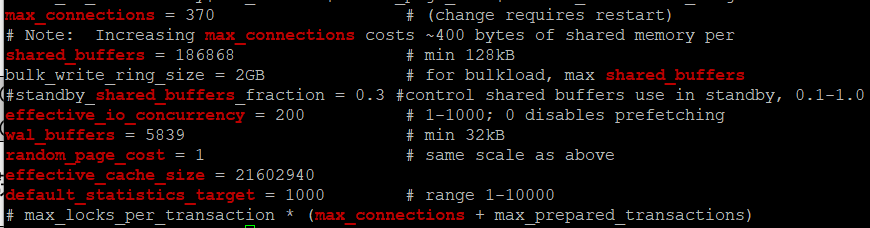
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

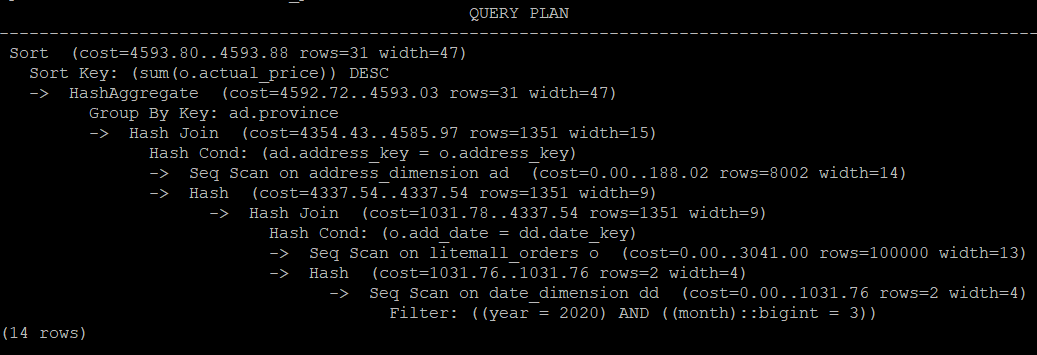
AND o.add\_date = dd.date\_key

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

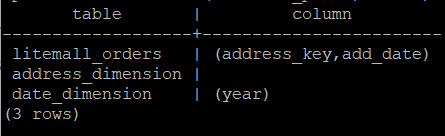
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

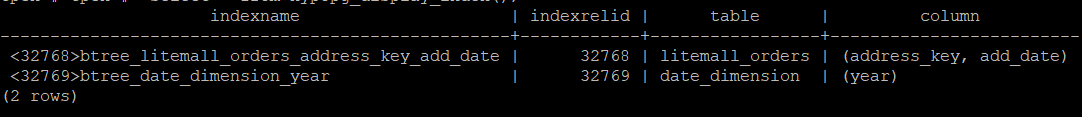
GROUP BY ad.province

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。

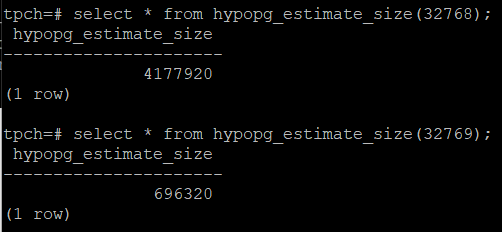
select \* from hypopg\_display\_index();



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

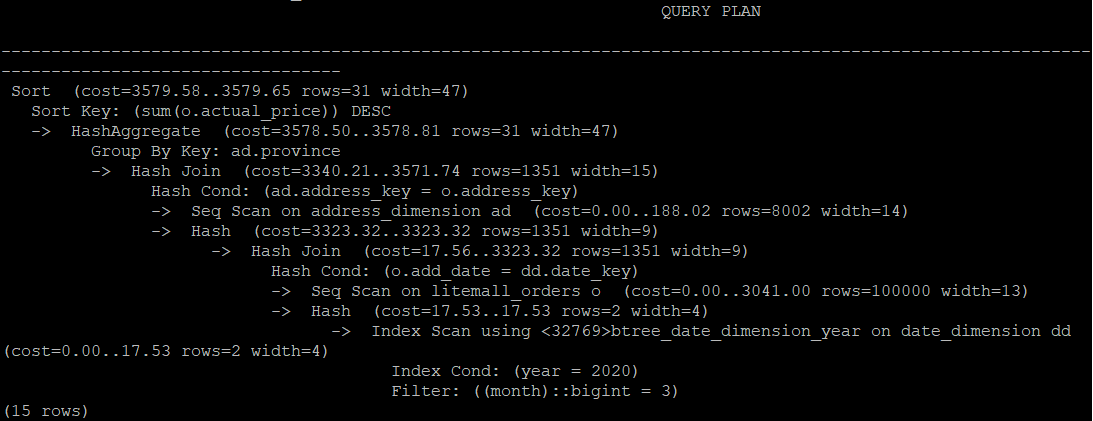
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

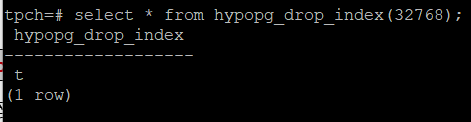
GROUP BY ad.province

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



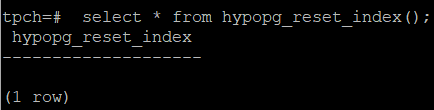
6. 删除某一个索引虚拟列，将执行结果截图。

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



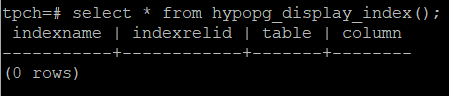
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

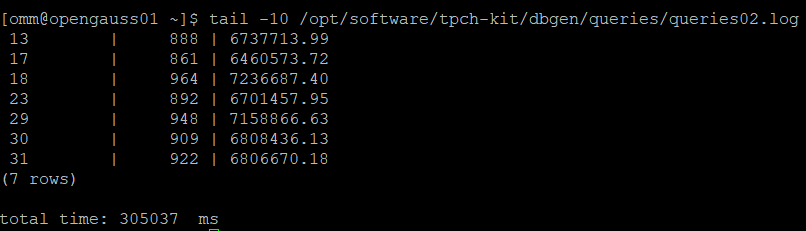
select \* from hypopg\_display\_index();



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



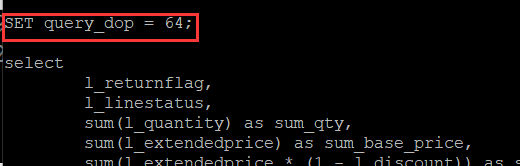
挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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

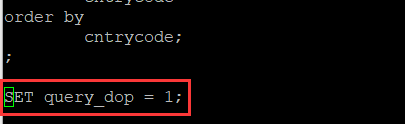
可以通过设置query\_dop（默认值为1）来最大限度利用空闲资源提高性能，缩短执行时间。

通过先设置query\_dop为100发现提示范围应在[-64, 64]，故进入queries.sql将其设置成64。

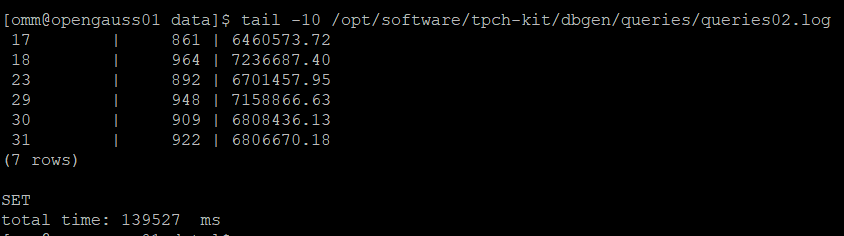


注意：还需在queries.sql末尾将query\_dop设置回默认值1，因为下次的sql不一定要这么高

的query\_dop。



重新执行queries.sql查询，得到结果



结论：发现执行大量sql的情况下提高并行度比设置索引更能加快sql执行的速度。

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？

优化了以下参数

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

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

通过缓存、连接数、事务量等参数进行数据库调优

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）使用索引对于执行SQL有以下好处

① 加快数据的检索速度

② 加快表之间的连接

③ 减少分组和排序时间

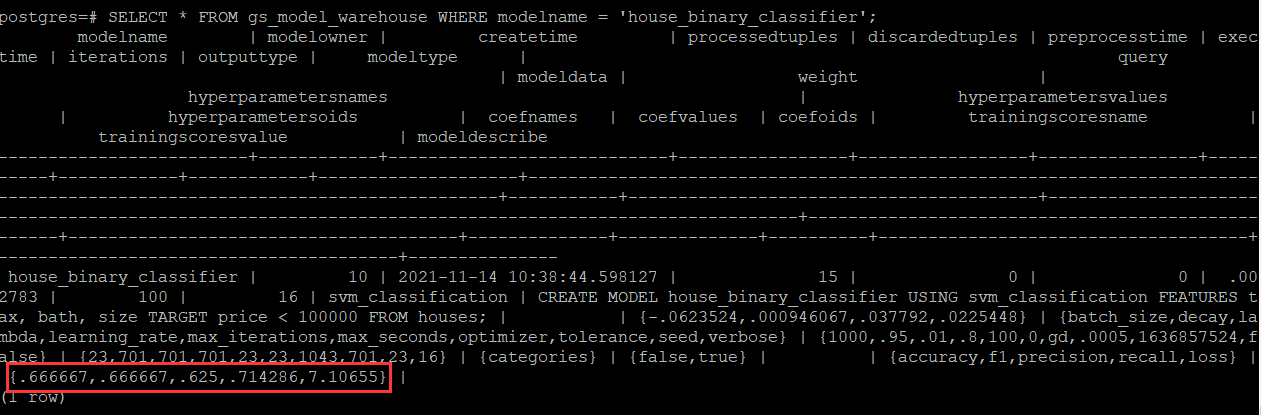
④ 使用优化隐藏器提高系统性能

（2）可以通过分区分库分表进行数据库的优化。

# 关卡四、openGauss的DB4AI特性应用

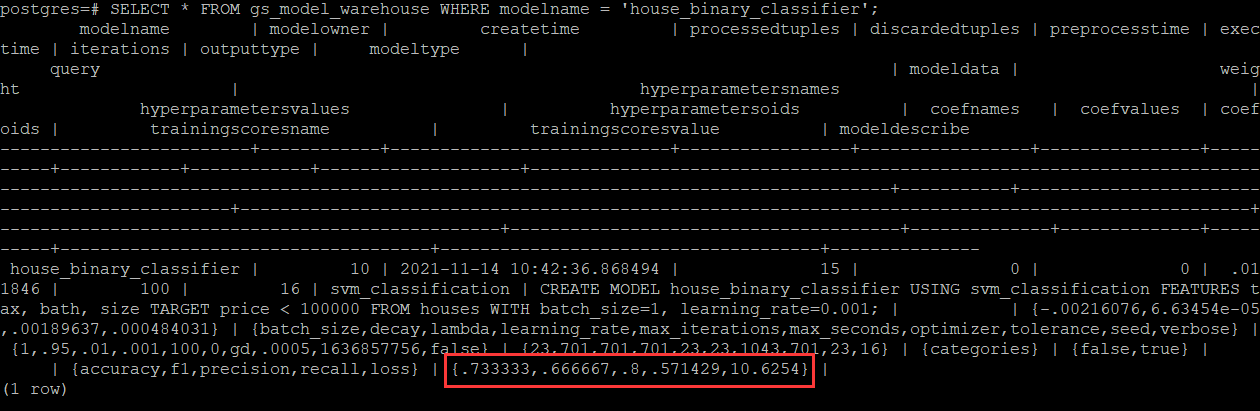
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



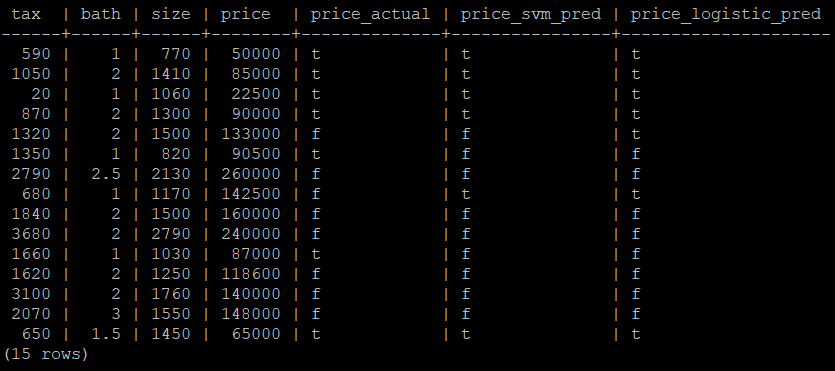
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型与回归模型不同点在于输出变量的类型。

回归模型是定量输出，也可以说是连续变量预测；

分类模型是定性输出，也可以说是离散变量预测。

实践思考题2：什么是SVM算法？

支持向量机（SVM）是一类按监督学习方式对数据进行二元分类的广义线性分类器，其决策边

界是对学习样本求解的最大边距超平面。

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？

① 准确率：(预测正确)/(预测对的和不对的所有结果)，即预测正确的比例；

② 召回率：真正为正例的样本中预测结果为正例的比例。正样本有多少被找出来了（召回了多少）；

③ 精确率：预测结果为正例样本（TP+FP）中真实值为正例（TP）的比例。

④ f1-score：精确率和召回率的调和平均数；

⑤ AUC：一个模型评价指标，表面上意思是ROC曲线下边的面积

ROC曲线（接收者操作特征曲线）

真正例率TPR = TP / (TP + FN)

预测为正例且实际为正例的样本占所有训练集中为正例样本的比例。

将正例预测对的占正样本的比例（预测对的比例），这个比例越大越好

伪反例率FPR = FP / (FP + TN)

预测为正例但实际为负例的样本占训练集中所有负例样本的比例

将负例预测错的占负样本的比例（预测错的比例），这个比例越小越好

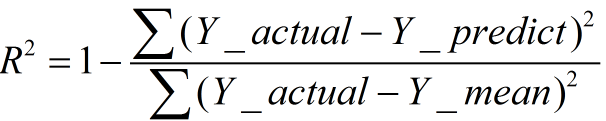
实践思考题4：回归问题有哪些评价指标，请分别说明他们的含义？

① SSE(误差平方和)



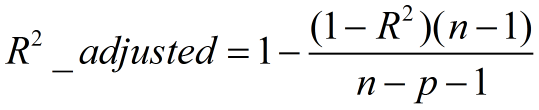
同样的数据集的情况下，SSE越小，误差越小，模型效果越好

② R-square(决定系数)



越接近0，表明模型拟合的越差

③ Adjusted R-Square (校正决定系数）



n为样本数量，p为特征数量

消除了样本数量和特征数量的影响