openGauss AI特性创新实践课



华为技术有限公司

# 目录

**[关卡一、openGauss数据安装及基本操作 1](#_Toc26859)**

**[任务一：数据库状态验证 1](#_Toc20369)**

**[任务二：数据库服务进程验证 1](#_Toc16108)**

**[任务三：简单描述关卡一中，安装数据库所需要的步骤 2](#_Toc31461)**

[实验思考题：为什么需要通过源码编译，安装数据库？ 2](#_Toc3337)

**[关卡二、openGauss数据导入及基本操作 3](#_Toc6114)**

**[任务一：数据初始化验证 3](#_Toc14647)**

**[任务二：行存表与列存表执行效率对比 3](#_Toc16010)**

**[任务三：物化视图的使用 7](#_Toc9672)**

[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？ 10](#_Toc12508)

[实践思考题2：全量物化视图与增量物化视图有哪些差别？ 11](#_Toc6636)

**[关卡三、openGauss的AI4DB特性应用 13](#_Toc6593)**

[任务一：使用X-Tuner进行参数优化 13](#_Toc7006)

[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。 15](#_Toc6122)

[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执行时间。 21](#_Toc30904)

[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。 22](#_Toc15885)

[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？ 22](#_Toc14016)

[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？ 23](#_Toc12992)

**[关卡四、openGauss的DB4AI特性应用 25](#_Toc3045)**

[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： 25](#_Toc23223)

[任务二：观察新模型的信息，将执行结果截图。 25](#_Toc23147)

[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。 25](#_Toc26113)

[实践思考题1：分类模型与回归模型有何不同？ 26](#_Toc30105)

[实践思考题2：什么是SVM算法？ 26](#_Toc30834)

[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？ 26](#_Toc16630)

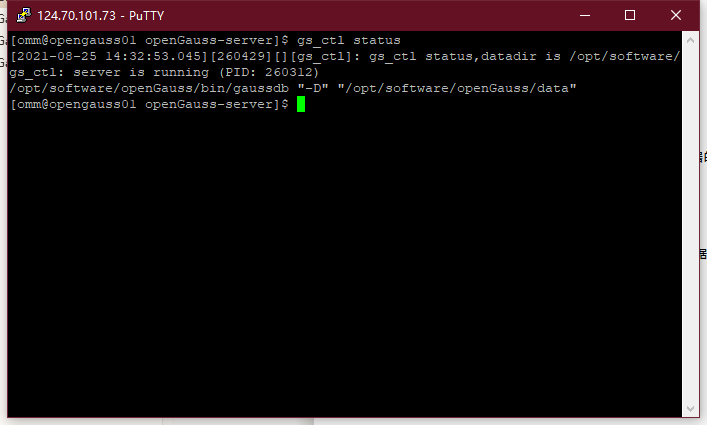
[实践思考题4：回归问题有哪些评价指标，请分别说明他们的含义？ 27](#_Toc5504)

# 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：

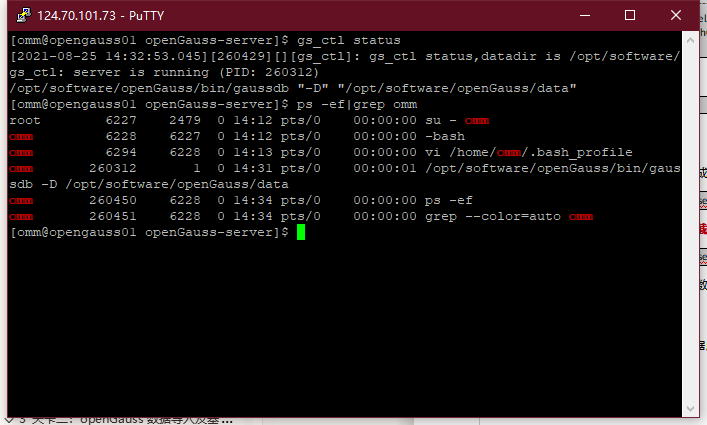
**任务一：**数据库状态验证

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）



**任务三：**简单描述关卡一中，安装数据库所需要的步骤

编译前准备：远程登录->创建openGauss数据库的安装用户omm及其属组dbgrp->创建openGauss源码存放及openGauss安装路径->下载第三方编译库->解压下载好的第三方编译库，并重命名为binarylibs->下载openGauss源码->上传并解压[cmake-3.20.0-linux-aarch64.tar.gz](https://github.com/Kitware/CMake/releases/download/v3.20.0/cmake-3.20.0-linux-aarch64.tar.gz)->使用yum安装依赖包->修改/opt/software路径的用户所属组及权限

开始编译：切换omm用户->设定omm用户的环境变量并生效->进入openGauss源码下，生成配置文件->编译并安装->初始化数据库，设置数据库密码->启动，登录，退出

**实验思考题：**为什么需要通过源码编译，安装数据库？

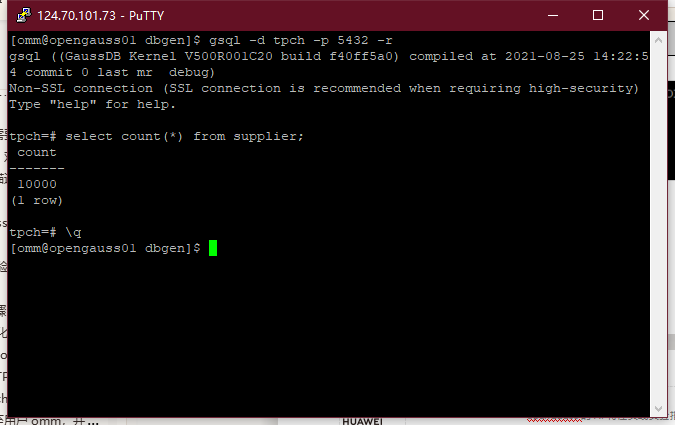
MySQL 5.5之后，所有的编译操作都通过cmake进行，使用cmake最大的好处是其独立于源码(out-of-source)的编译功能，即编译工作可以在另一个指定的目录中而非源码目录中进行，这可以保证源码目录不受任何一次编译的影响，因此在同一个源码树上可以进行多次不同的编译，如针对于不同平台编译。

# 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：

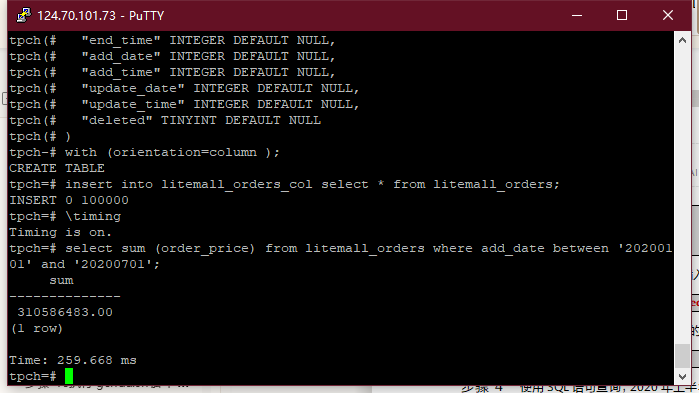
select count(\*) from supplier;;



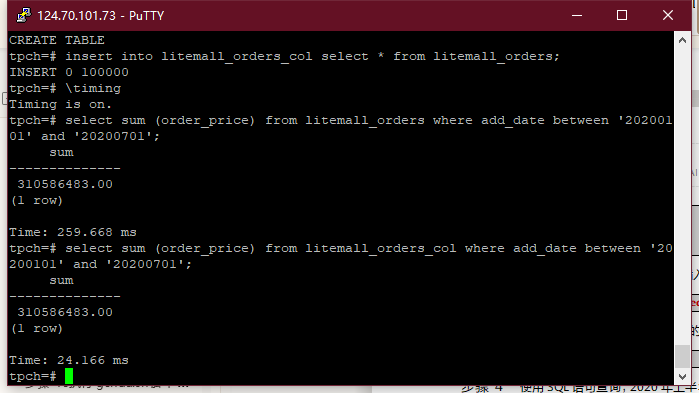
**任务二：**行存表与列存表执行效率对比

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';

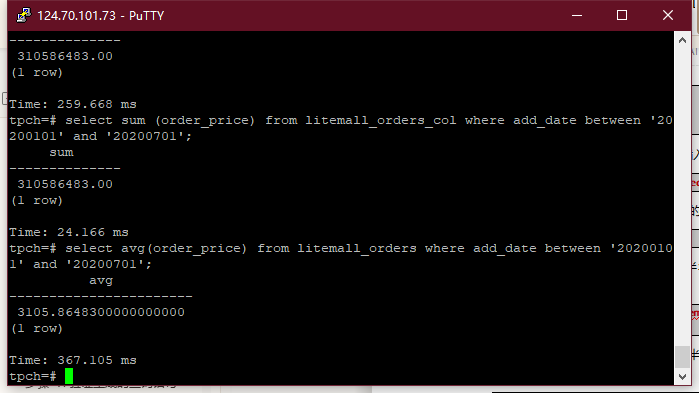


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

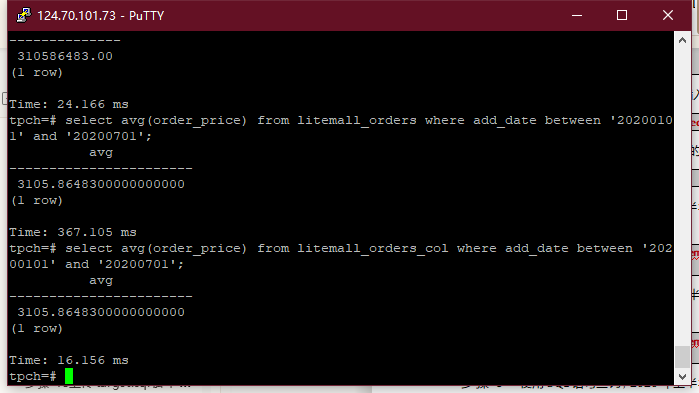


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

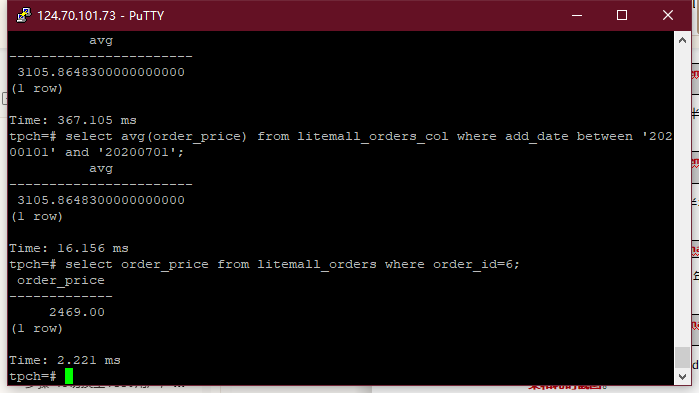


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

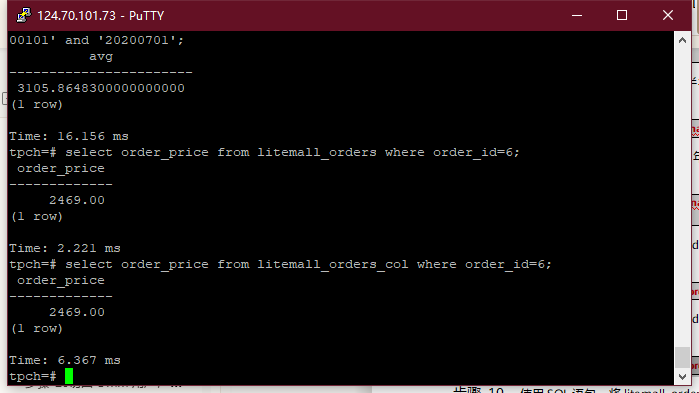


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

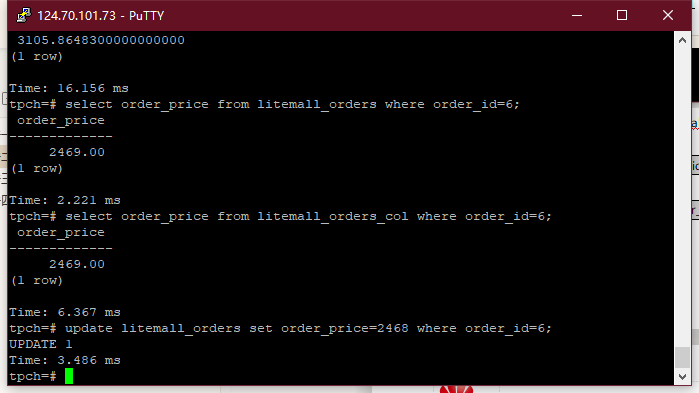


select order\_price from litemall\_orders\_col where order\_id=6;

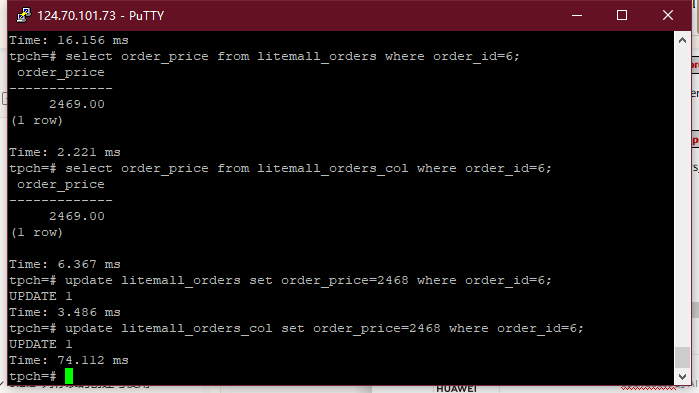


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



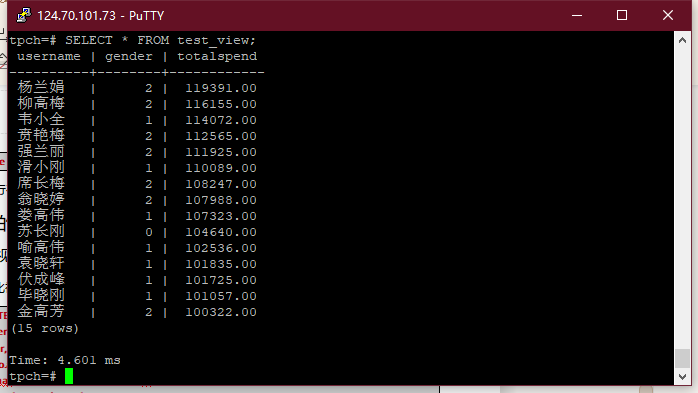
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



**任务三：**物化视图的使用

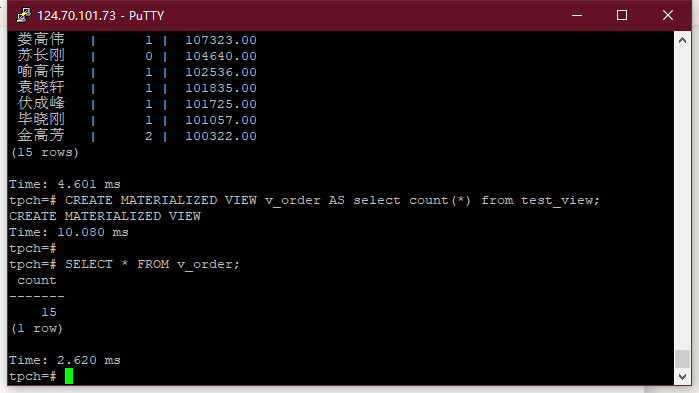
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



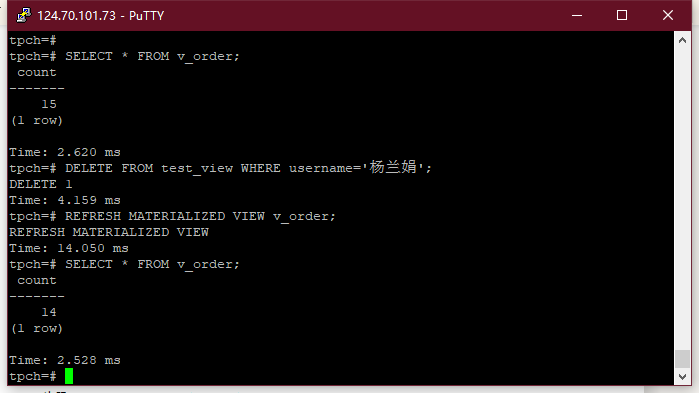
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



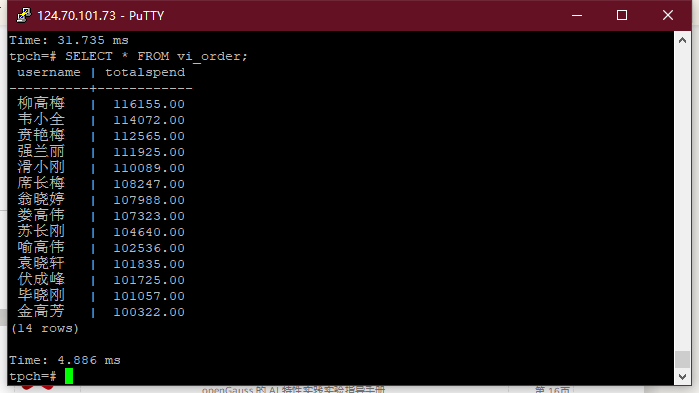
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



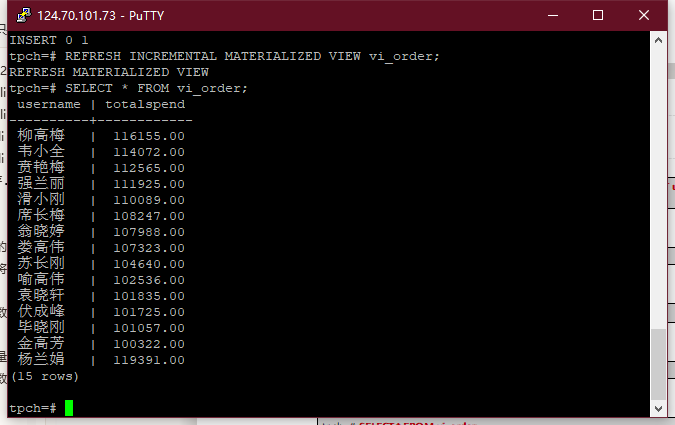
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

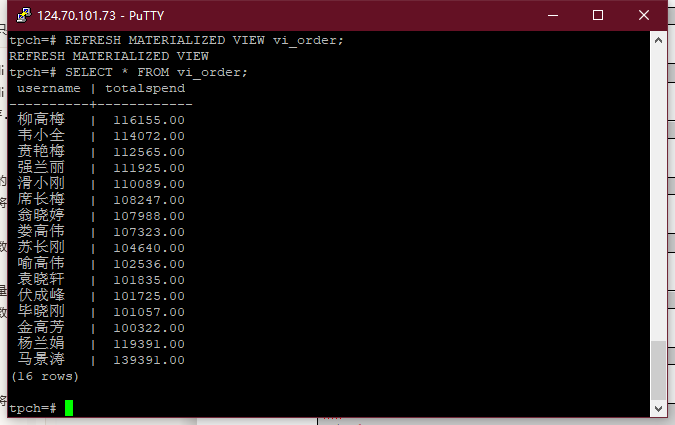
SELECT \* FROM vi\_order;



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;



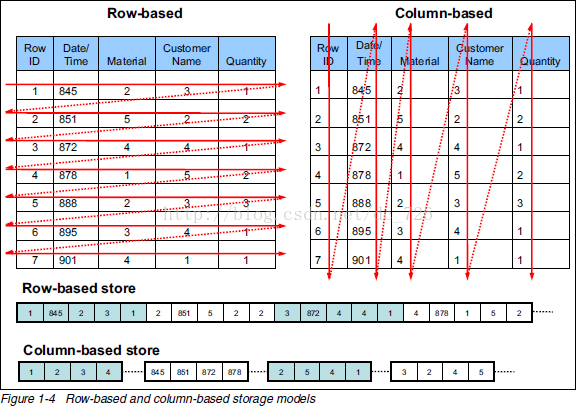


**实践思考题1：**行存表与列存表在执行相同的SQL语句时，为何执行的时间不同？在执行哪些类型SQL时，行存表效率更高？在执行哪些类型SQL时，列存表效率更高？

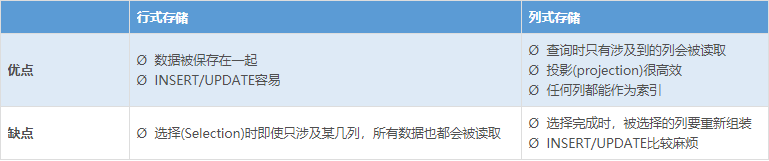
列式存储(Columnar or column-based)是相对于传统关系型数据库的行式存储(Row-basedstorage)来说的。简单来说两者的区别就是如何组织表(翻译不好，直接抄原文了)：

Ø Row-based storage stores atable in a sequence of rows.

Ø Column-based storage storesa table in a sequence of columns.



从上图可以很清楚地看到，行式存储下一张表的数据都是放在一起的，但列式存储下都被分开保存了。所以它们就有了如下这些优缺点：



只指定column，不指定某行，列存表效率高（求一整列的平均值）；

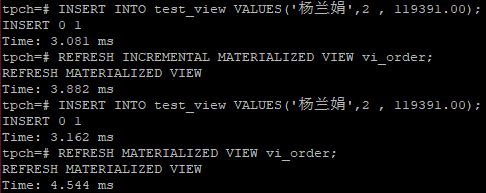
指定column中的某行后，行存表效率高（修改某一行的值）。

**实践思考题2：**全量物化视图与增量物化视图有哪些差别？

类似全更新与增量更新的区别：

FAST刷新采用增量刷新，只刷新自上次刷新以后进行的修改。

COMPLETE刷新对整个物化视图进行完全的刷新。



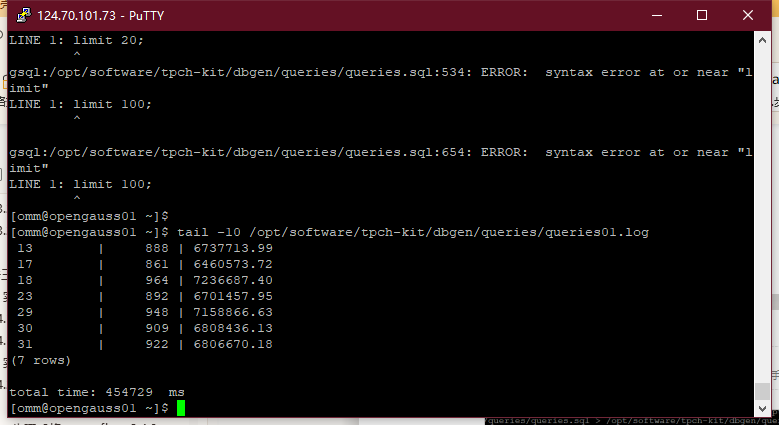
很明显，插入相同数据时，增量物化视图的刷新速度更快，当数据基量增大、修改量减少时，这种差距会愈发显著。不过更严谨的对照实验应是在中间删除掉刚刚新增的数据。

# 关卡三、openGauss的AI4DB特性应用

**任务一：**使用X-Tuner进行参数优化

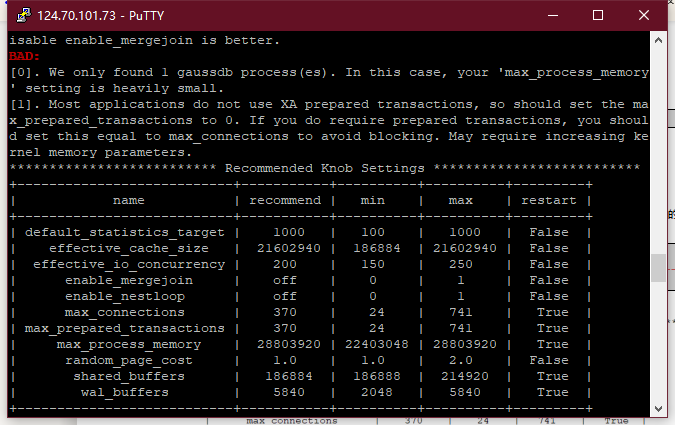
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

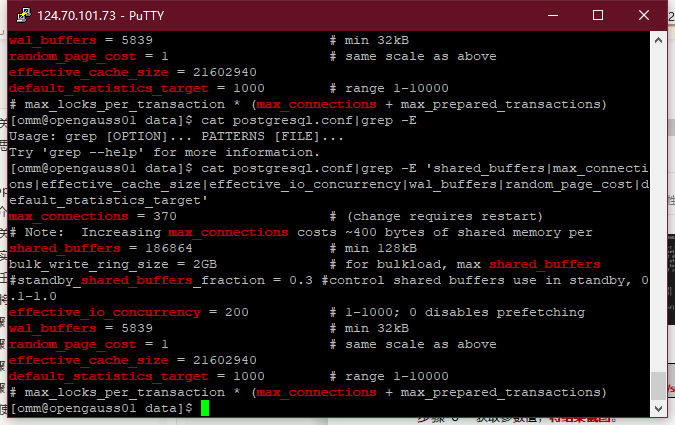
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

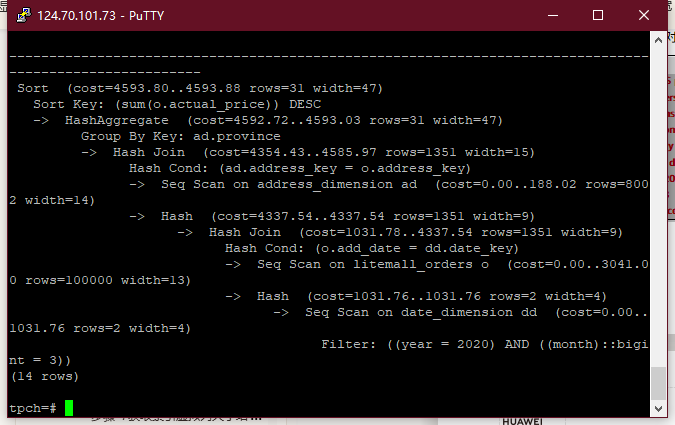
AND o.add\_date = dd.date\_key

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

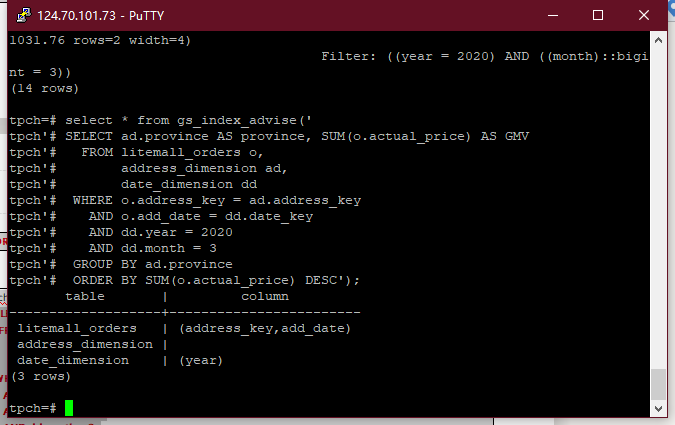
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

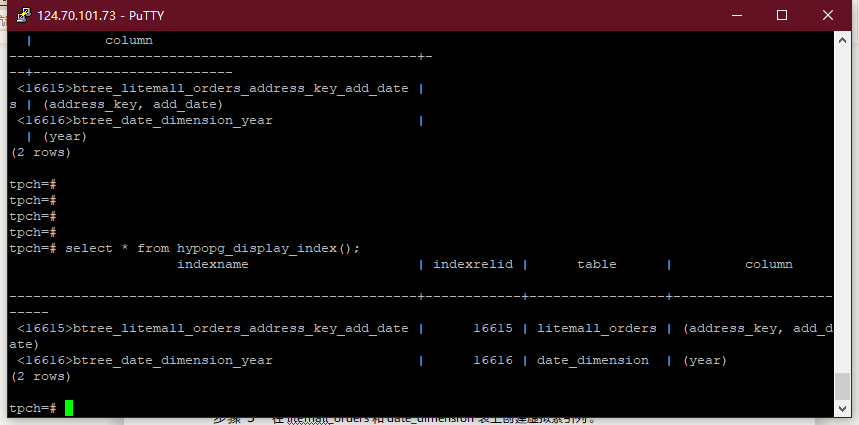
GROUP BY ad.province

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。

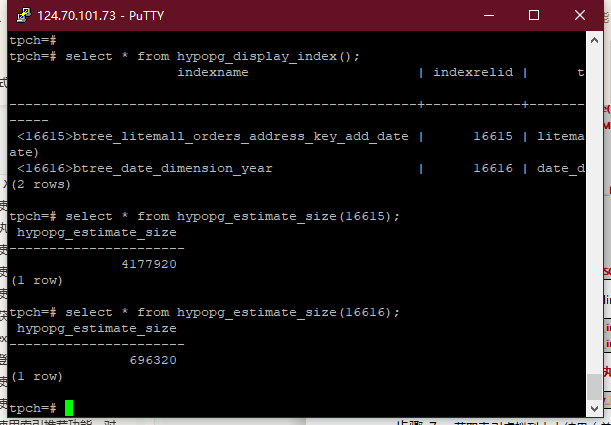
select \* from hypopg\_display\_index();



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

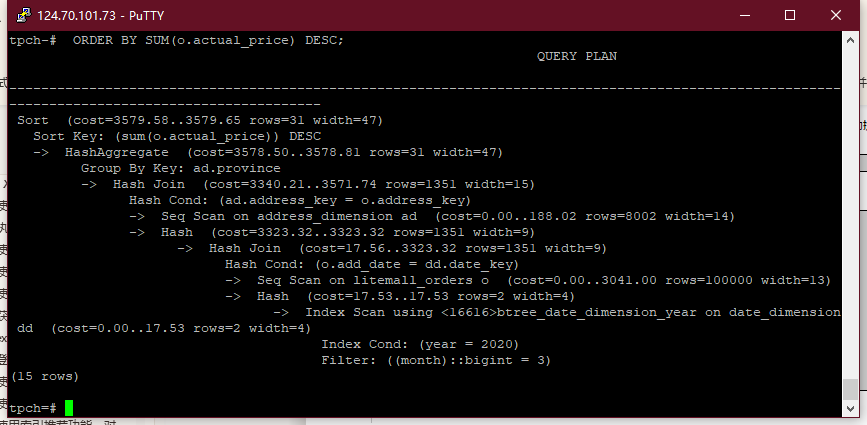
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

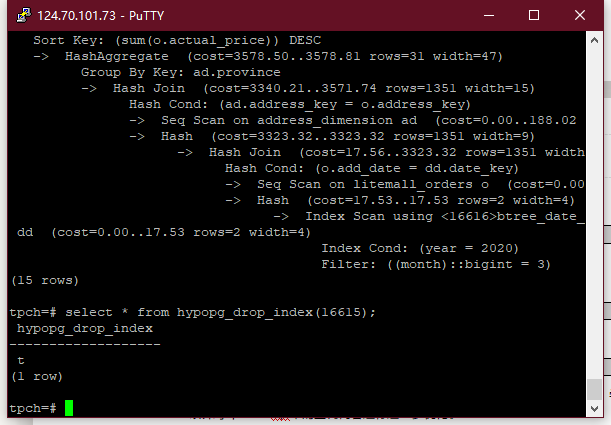
GROUP BY ad.province

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



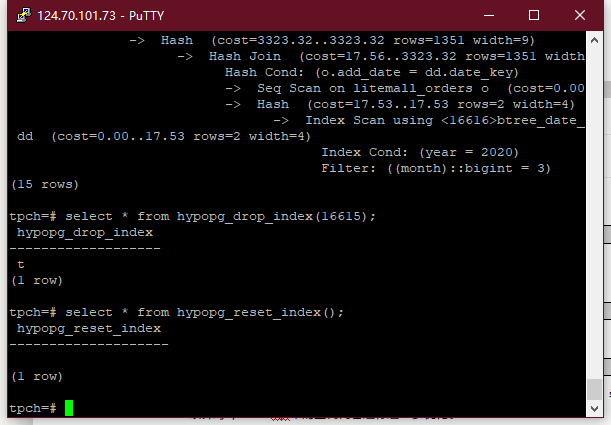
6. 删除某一个索引虚拟列，将执行结果截图。

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



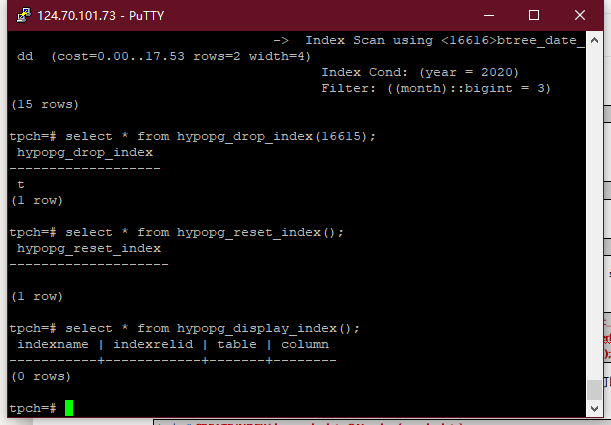
7. 删除所有索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

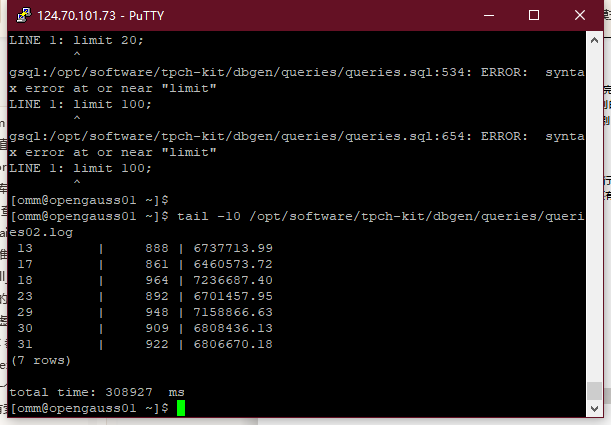
select \* from hypopg\_display\_index();



**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

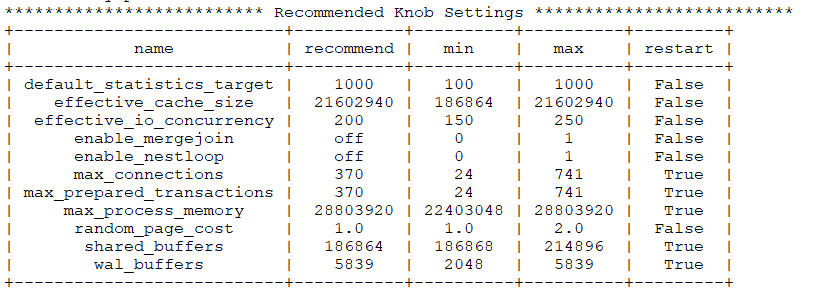


**挑战一：**进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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

**实事求是，本题我不会做，对不住。**

**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？

shared\_buffers = 数据页缓冲区

max\_connections = 最大连接数

effective\_cache\_size = 有效缓存大小

effective\_io\_concurrency = 有效IO并发

wal\_buffers = 预写日志缓冲区

random\_page\_cost = 随机页面开销

default\_statistics\_target = 默认统计目标

1、为什么要引入shared\_buffer?

在数据库系统中，我们主要关注磁盘IO, 大部分oltp工作负载都是随机IO,因此从磁盘获取非常慢。为了解决这个问题，数据库将数据缓存在RAM中，来提高性能。数据库在查询前，会先查找shared\_buffer的页，如果命中，就直接返回，避免从磁盘中查询。

2、为什么要修改shared\_buffer参数的值？

由于数据库的设计原则是确保在所有受支持的机器和操作系统上兼容，因此默认情况下这个值被保守地设置为较低。因此，在大多数现代操作系统上，更新shared\_buffers是提高整体性能最有效的设置之一。

3、wal\_buffer应该设置为多大？

缓冲区的默认大小由wal\_buffers设置，设置最初为16MB。如果要调优的系统有大量并发连接，那么wal\_buffers的值越高，性能越好。

4、什么是effective\_cache\_size？

首先要弄清楚，操作系统上的内存使用是怎么回事。

假设我们服务器一共有100gb内存，2 GB可能被操作系统占用，同时我们给了25GB给shared\_buffer，同时数据库可能还需要一些内存对数据进行排序，保持数据库连接等。假设之这些一共需要30GB。剩下差不多70GB，其中一些可能是空闲的，但是大部分将最终作为文件系统缓存。

每当Linux执行I/O时，如果文件系统缓存周围有足够的空闲内存，就会开始缓存数据，尽可能避免磁盘I/O。文件系统是至关重要的，可以根据需要动态地改变大小。如果数据库需要更多的RAM来排序数据，它会分配内存，这反过来会使操作系统根据需要收缩文件系统缓存，以确保效率。

5、effective\_cache\_size的作用是什么？

这个值仅由数据库的查询规划器使用，以确定它所考虑的计划是否应该适合RAM。

在数据库中，优化器负责确保查询以最有效的方式执行。然而，要做到这一点，有必要知道实际有多少内存。系统知道它自己的内存(= shared\_buffers)的大小，而effective\_cache\_size有助于查询规划器确定实际缓存的数量，并有助于调整I/O缓存。

具体来说就是：如果effecve\_cache\_size的值太低，那么查询规划器可能会决定不使用某些索引，即使它们可以极大地提高查询速度。所以换句话说，这个值是数据库用来估计索引的成本用的。

**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？

数据库索引是为了增加查询速度而对表字段附加的一种标识。

首先明白为什么索引会增加速度：DB在执行一条Sql语句的时候，默认的方式是根据搜索条件进行全表扫描，遇到匹配条件的就加入搜索结果集合。如果我们对某一字段增加索引，查询时就会先去索引列表中一次定位到特定值的行数，大大减少遍历匹配的行数，所以能明显增加查询的速度。

那么在任何时候都应该加索引么？这里有几个反例：1、如果每次都需要取到所有表记录，无论如何都必须进行全表扫描了，那么是否加索引也没有意义了。2、对非唯一的字段，例如“性别”这种大量重复值的字段，增加索引也没有什么意义。3、对于记录比较少的表，增加索引不会带来速度的优化反而浪费了存储空间，因为索引是需要存储空间的，而且有个致命缺点是对于update/insert/delete的每次执行，字段的索引都必须重新计算更新。

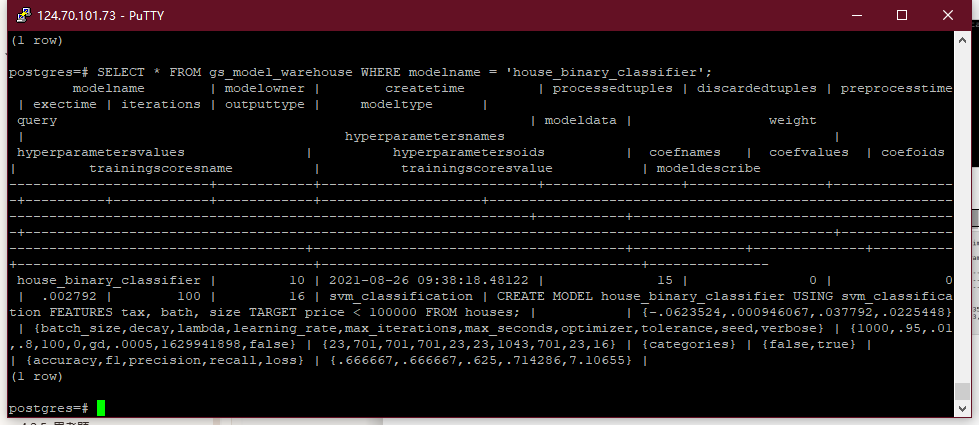
那么在什么时候适合加上索引呢？如：SELECT c.companyID, c.companyName FROM Companies c, User u WHERE c.companyID = u.fk\_companyID AND c.numEmployees >= 0 AND c.companyName LIKE '%i%' AND u.groupID IN (SELECT g.groupID FROM Groups g WHERE g.groupLabel = 'Executive') 这条语句涉及3个表的联接，并且包括了许多搜索条件比如大小比较，Like匹配等。在没有索引的情况下Mysql需要执行的扫描行数是77721876行。而我们通过在companyID和groupLabel两个字段上加上索引之后，扫描的行数只需要134行。通过Explain Select来查看扫描次数，可以看出来在这种联表和复杂搜索条件的情况下，索引带来的性能提升远比它所占据的磁盘空间要重要得多。

常见的数据库优化方法有：1、选取最适用的字段属性；2、使用连接来代替子查询；3、使用联合来代替手动创建的临时表；4、事务；5、锁定表；6、使用外键。

# 关卡四、openGauss的DB4AI特性应用

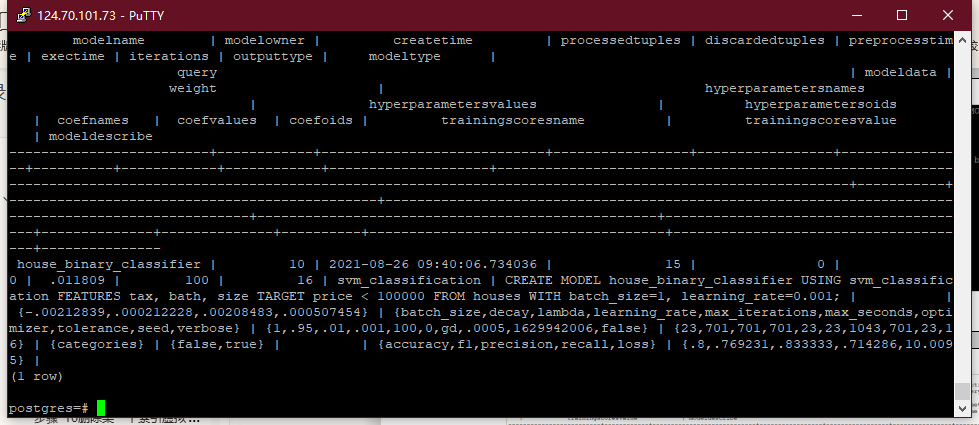
**任务一：**在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



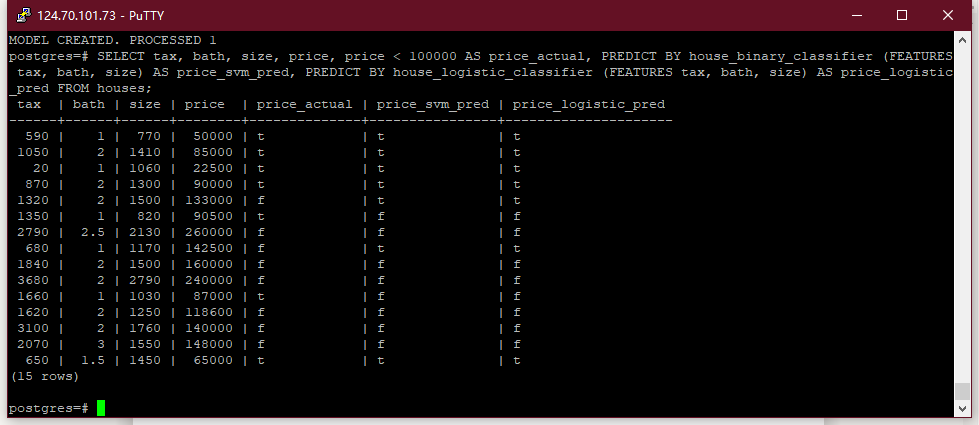
**任务二：**观察新模型的信息，将执行结果截图。

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。

**实践思考题2：**什么是SVM算法？

SVM（support Vector Mac）又称为支持向量机，是一种二分类的模型，可以分为线性核非线性两大类。其主要思想为找到空间中的一个更够将所有数据样本划开的超平面，并且使得本本集中所有数据到这个超平面的距离最短。

**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？

一，准确率

准确率是我们常见的评价指标之一，一般定义是，分类正确的样本数占总样本的比例数。

但是准确率在数据不均衡的数据集上会缺少一定的说服力，比如一个数据集，有990正样本，10个负样本，现在模型把样本全部都预测成正样本，那么我们说模型的准确率是99%，这个从计算上看是没问题，但很明显当样本不均衡时，光使用准确率来评价一个模型的好坏是不够的。

二，精准率和找回率

二分类问题的预测结果可以根据情况分成以下四类：

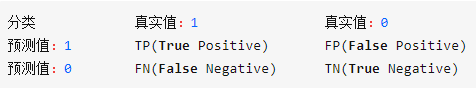
真正例（True Positive）：预测值为1，真实值为1

假正例（False Positive）：预测值为1，真实值为0

真反例（True Negative）：预测值为0，真实值为0

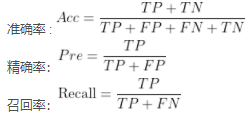
假反例（False Negative）：预测值为0，真实值为1

构造成结果的混淆矩阵：



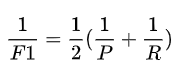
精准率又称查准率：预测为正例的样本中，有多少是真的正例

召回率又称查全率，真实的正例的样本中，有多少被预测出来



P和R指标有时候会出现的矛盾的情况，这样就需要综合考虑他们，最常见的方法就是F-Measure（又称为F-Score）。

F1：精准率和召回率的调和平均



三，AUC

AUC是另一种评价二分类算法的指标，被定义为 ROC 曲线（受试者工作特征曲线 （receiver operating characteristic curve，简称ROC曲线），又称为感受性曲线（sensitivity curve）。）下的面积。

四，混淆矩阵

混淆矩阵（confusion matrix），又称为可能性表格或是错误矩阵。它是一种特定的矩阵用来呈现算法性能的可视化效果，通常是监督学习（非监督学习，通常用匹配矩阵：matching matrix）。其每一列代表预测值，每一行代表的是实际的类别。这个名字来源于它可以非常容易的表明多个类别是否有混淆（也就是一个class被预测成另一个class）。

混淆矩阵的作用：

１）用于观察模型在各个类别上的表现，可以计算模型对应各个类别的准确率，召回率；

２）通过混淆矩阵可以观察到类别直接哪些不容易区分，比如Ａ类别中有多少被分到了Ｂ类别，这样可以有针对性的设计特征等，使得类别更有区分性；

**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？

点对点误差：

一，MSE均方误差

MSE(Mean Square Error)：



均方误差描述了样本真实值与预测值差方求和的平均值。利用均方误差最小化优化模型的方法称为“最小二乘法”；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。

二，RMSE

RMSE（Root Mean Square Error）即对MSE开平方根：



衡量观测值与真实值之间的偏差。

三，MAE平均绝对误差

MAE（Mean Absolute Erroe）即真实值与预测值差的绝对值和求平均：

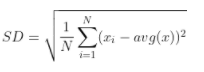


可以更好地反映预测值误差的实际情况。

四，SD

SD（Standard Deviation）标准差

方差的算术平均根。



用于衡量一组数值的离散程度。

带归一化的误差求解方法：

五，MAPE

MAPE(Mean Absolute Percentage Error)：



即求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。

六，MSPE

MASE（Mean Scaled Percentage Error）平均平方百分比误差：



即求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。