openGauss AI特性创新实践课



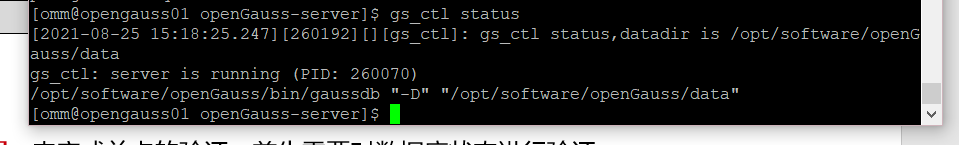
华为技术有限公司

# 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：

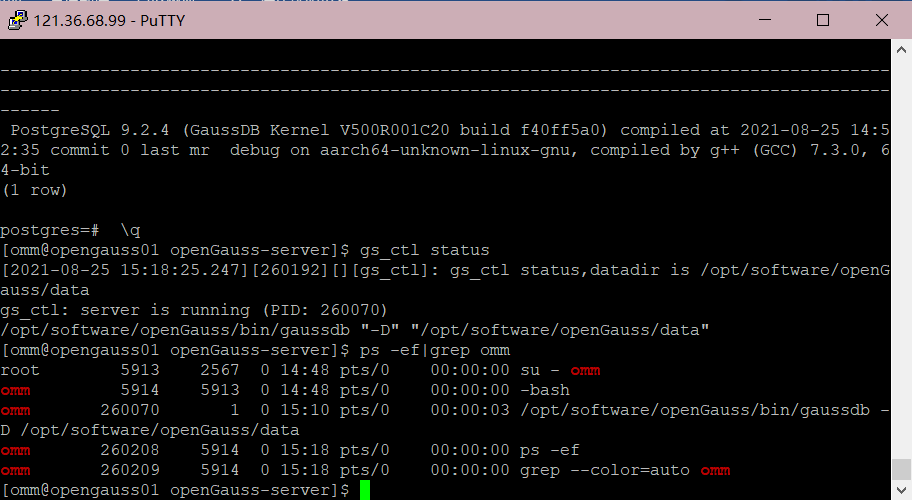
任务一：数据库状态验证

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）



简单描述关卡一中，安装数据库所需要的步骤。

前期准备：

用root用户名登录ECS，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。

下载第三方编译库和openGauss源码。

将[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)包，上传至服务器/usr/local/src/下进行解压。

使用yum安装依赖包。

将python3的链接设置为python的。

修改/opt/software路径的用户所属组及权限。

开始安装：

切换omm用户, 设定omm用户的环境变量，添加环境变量，使环境变量生效。进入openGauss源码下，生成配置文件。执行make命令进行编译，执行make install安装。

实验思考题：为什么需要通过源码编译，安装数据库？

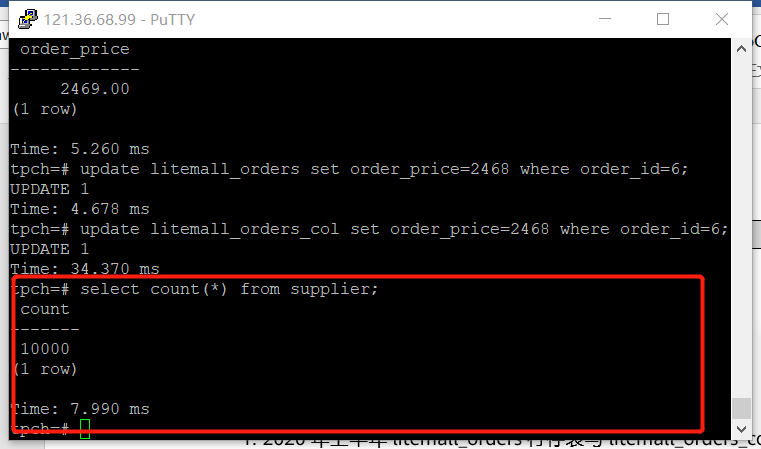
源码安装优点:可用性大,需要什么功能自己加什么参数；标准的二进制发行版可以在任何地方随时运行。版本众多，都是源代码自己构建起来的，各个依赖库的编译配置也都不一样，如果直接2进制版本安装很难保证能在所有版本下运行，所以得根据系统实际情况自己编译。

# 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：

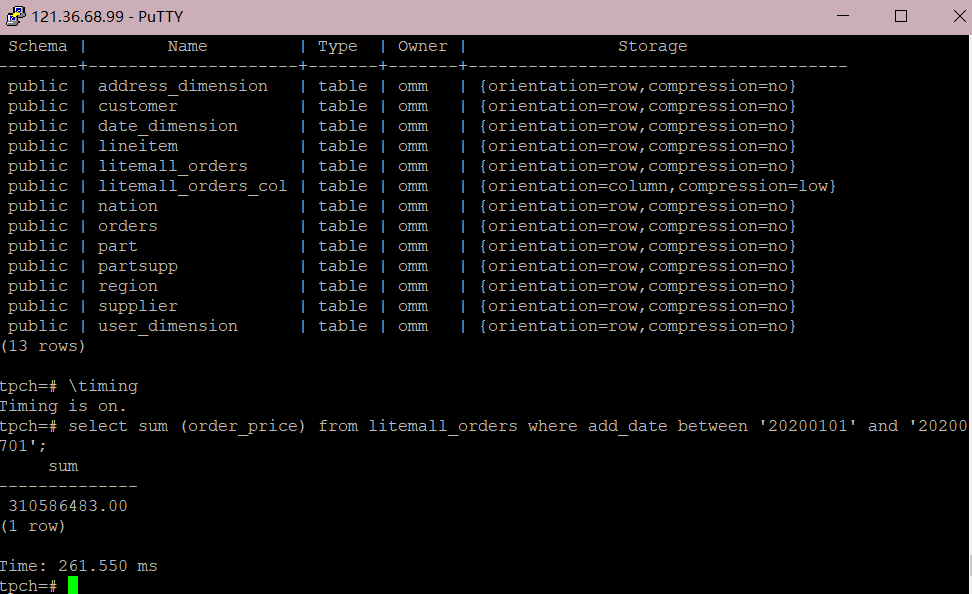
select count(\*) from supplier;;



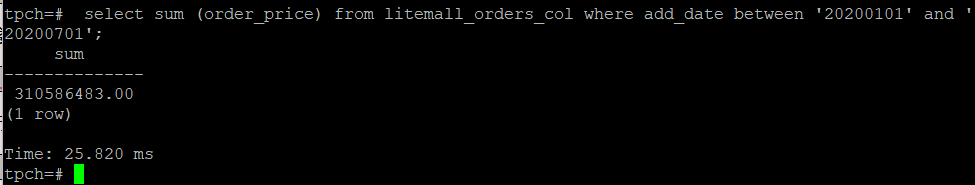
任务二：行存表与列存表执行效率对比

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';

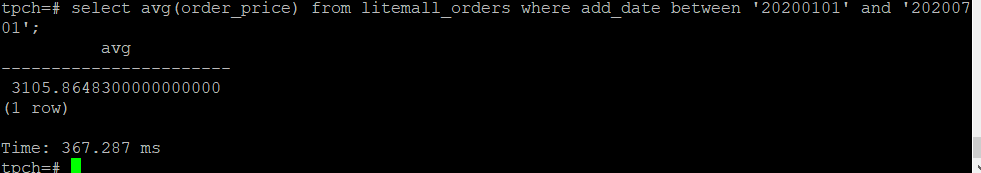


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

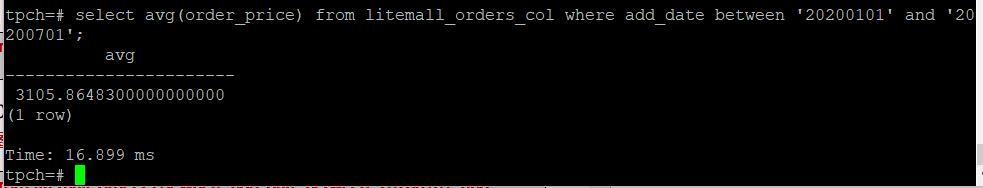


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

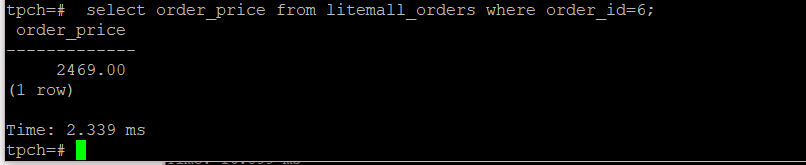


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

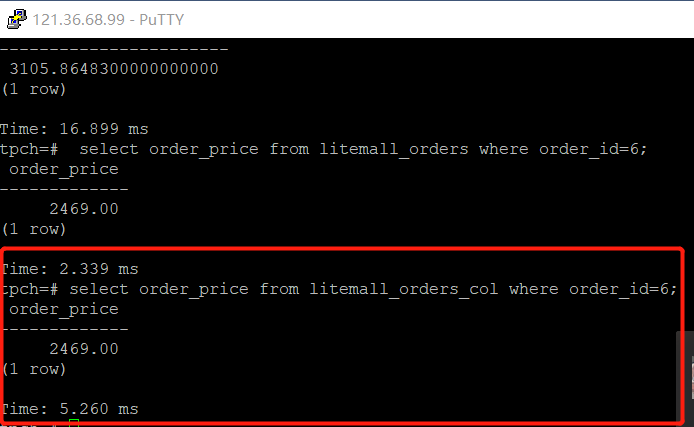


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

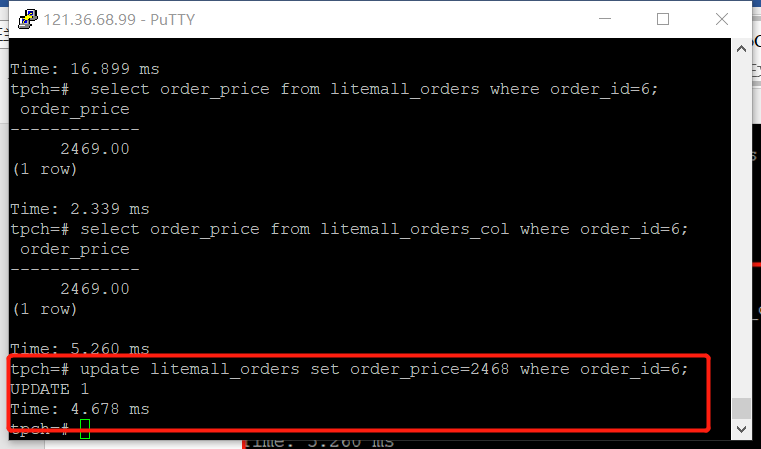


select order\_price from litemall\_orders\_col where order\_id=6;

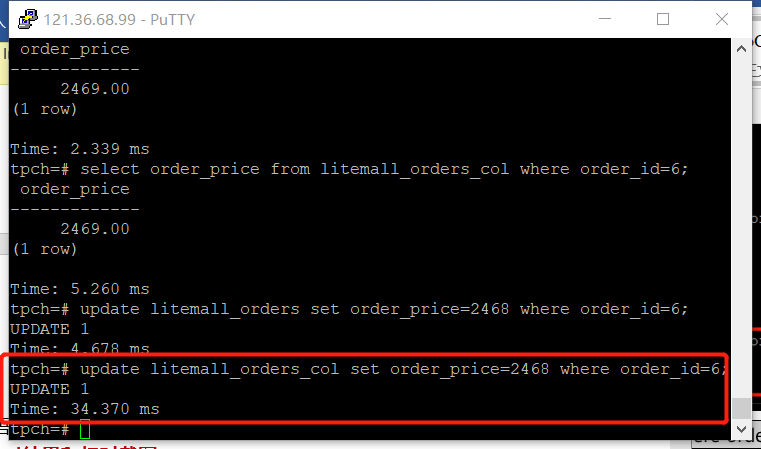


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



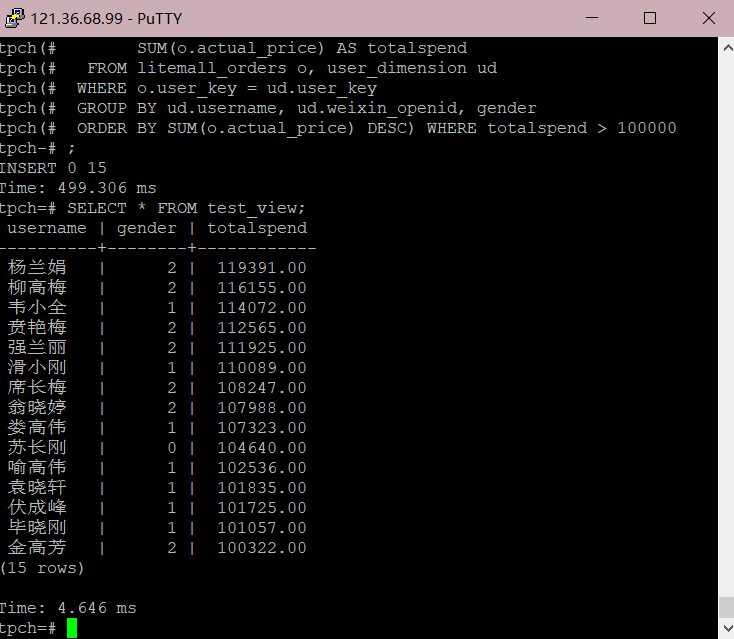
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

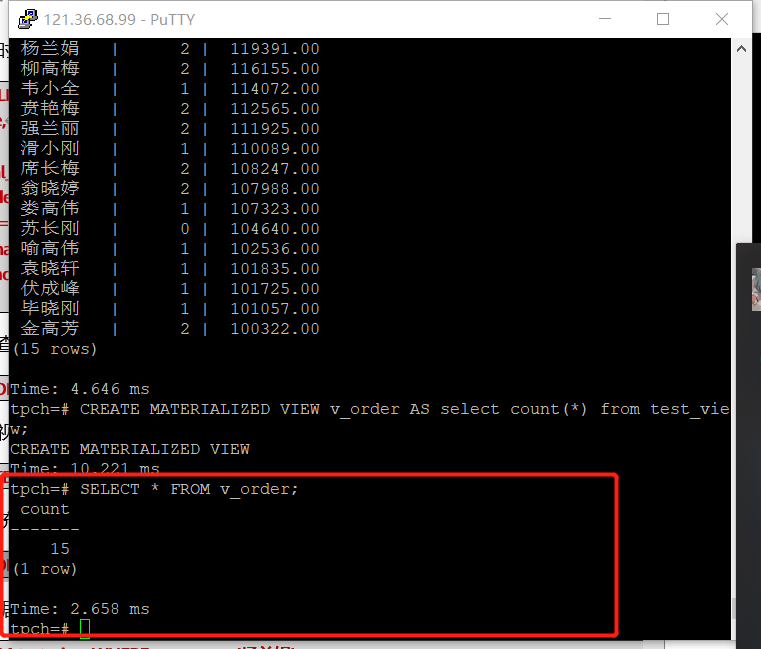
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



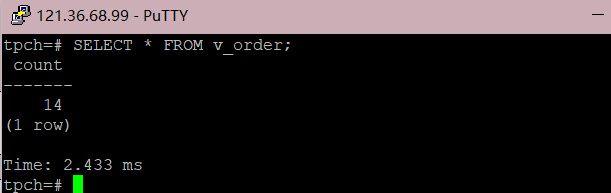
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



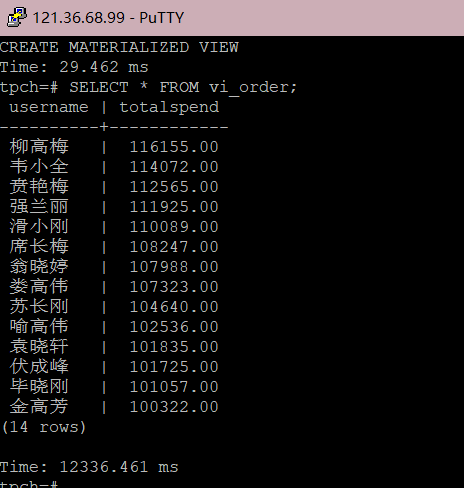
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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;



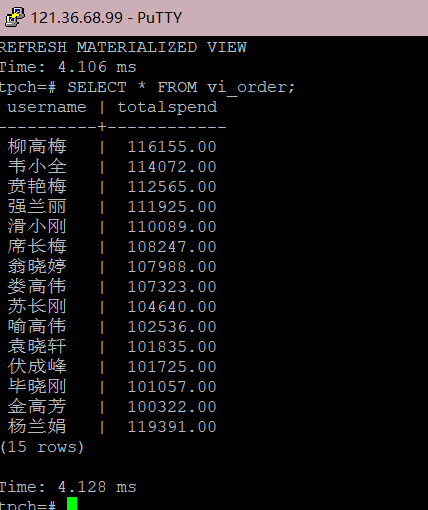
5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



6. 再次插入数据，刷新后查询物化视图结果，**将执行结果截图**。

tpch=# **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？

在基于行式存储的数据库中， 数据是按照行数据为基础逻辑存储单元进行存储的， 一行中的数据在存储介质中以连续存储形式存在；在基于列式存储的数据库中， 数据是按照列为基础的逻辑存储单元进行存储的，一列中的数据在存储介质中以连续存储形式存在。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。因此同样的语句查询两种表时，所需的时间不同。

本实验中，查找某属性为特定值的字段的另一属性的值，或者修改这一属性的值，行存表效率更高；总和、平均值查询，列存表效率更高。

如果所关注的内容是不需要通过任何聚集运算的，行式存储效率更高。关注某几列的内容，或者有频繁聚集需要的，通过聚集之后进行数据分析的表，列式存储效率高。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

物化视图是包括一个查询结果的数据库对象，它是远程数据的的本地副本，或者用来生成基于数据表求和的汇总表。

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。创建全量物化视图语法和CREATE TABLE AS语法一致，不支持对全量物化视图指定NodeGroup创建。

增量物化视图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。

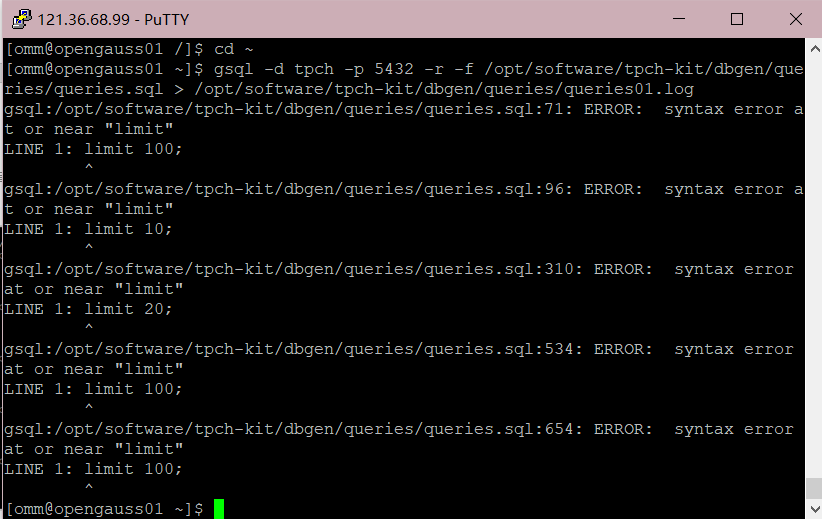
实验可以得知增量物化视图的查询较慢。

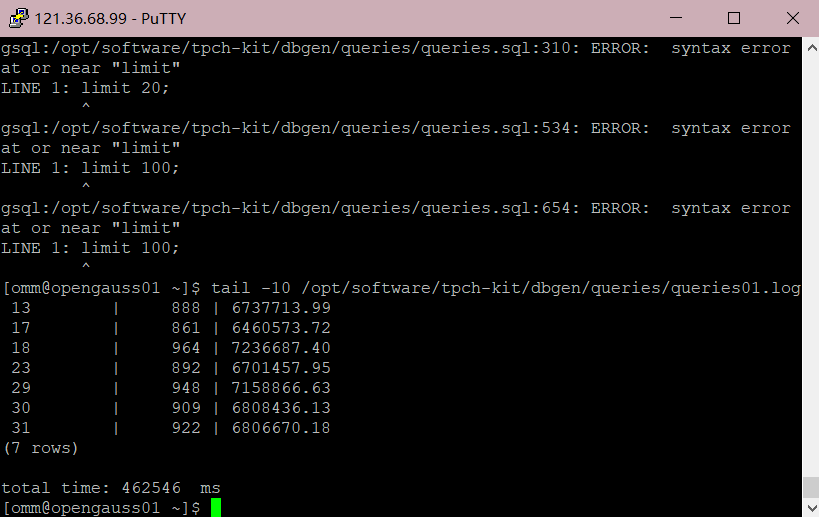
# 关卡三、openGauss的AI4DB特性应用

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：

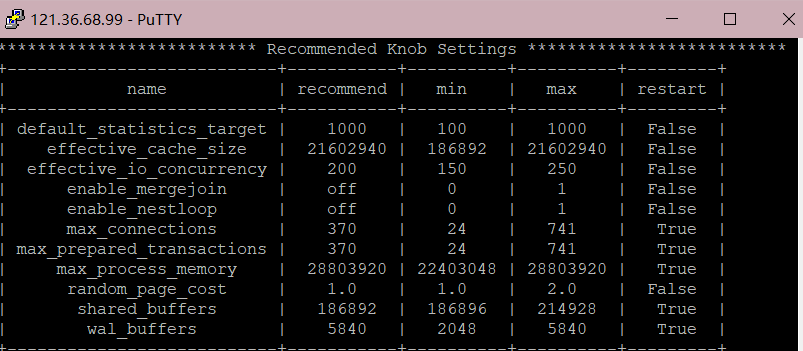
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log

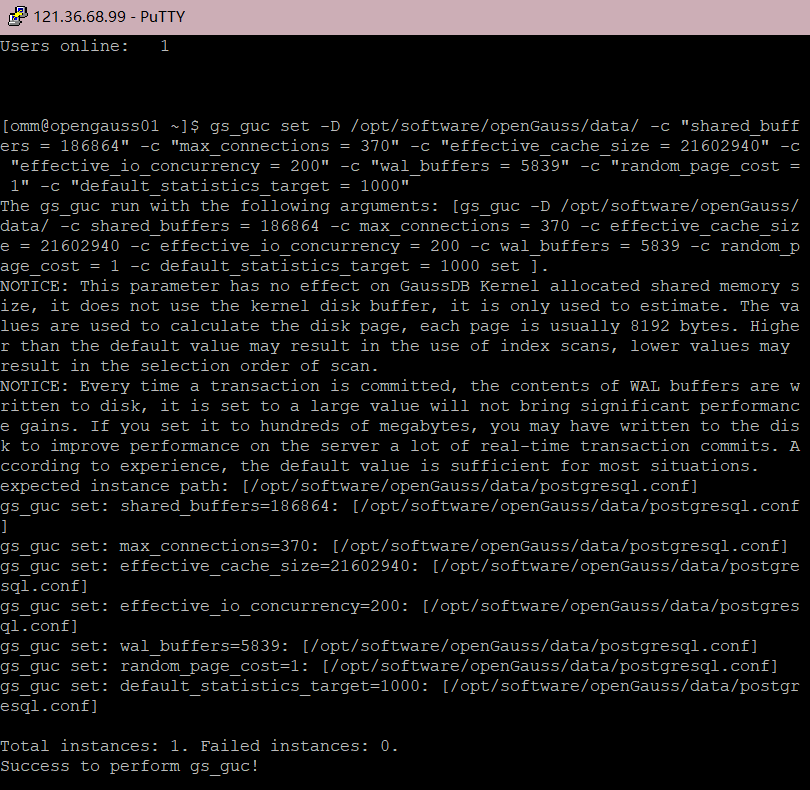




2. 使用root用户，执行X-Tuner进行参数建议优化，将执行结果截图

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

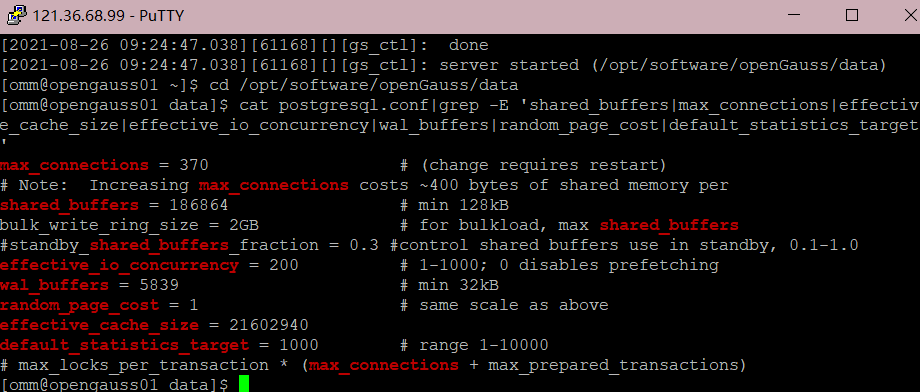




3.重启完成后，获取参数值：

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

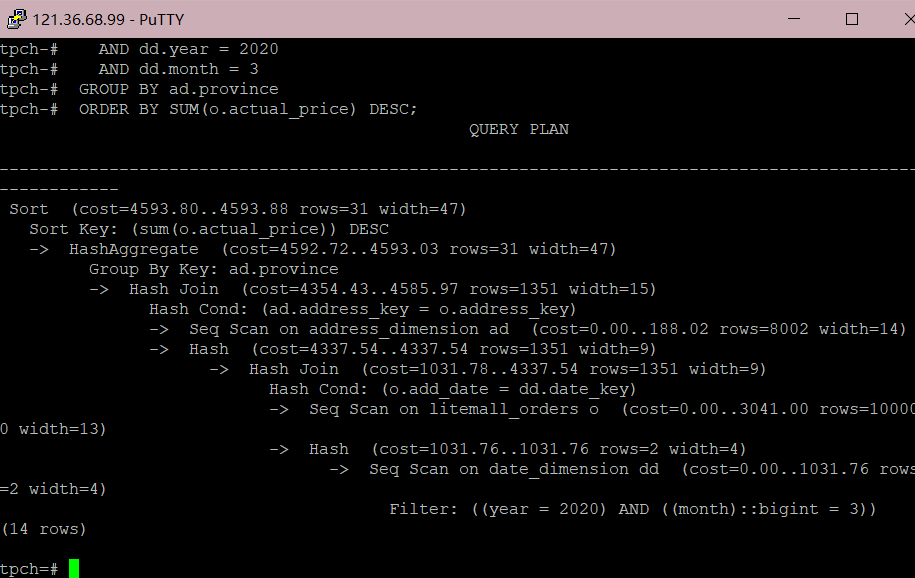
AND o.add\_date = dd.date\_key

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

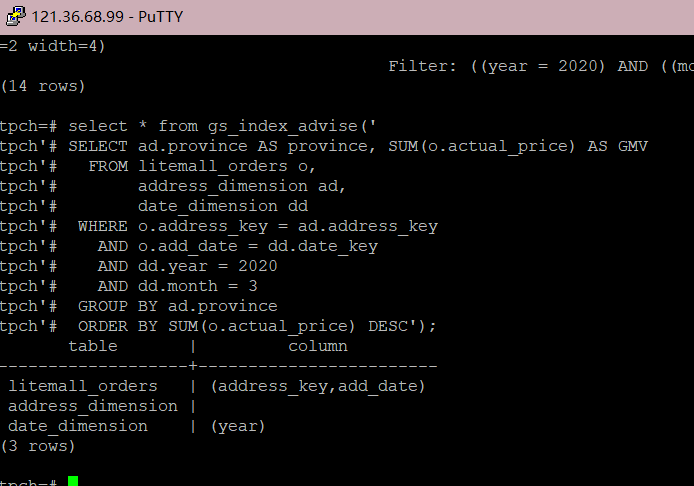
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

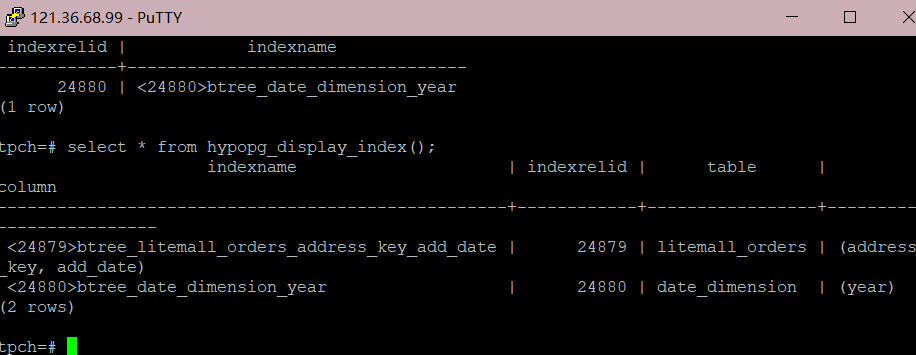
GROUP BY ad.province

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。

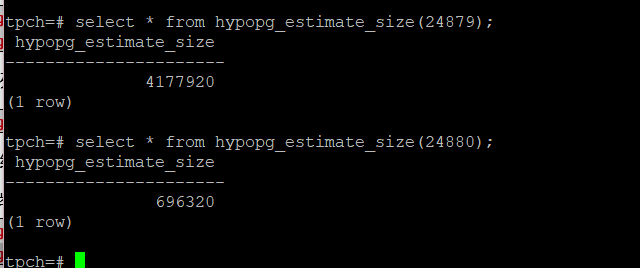
select \* from hypopg\_display\_index();



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

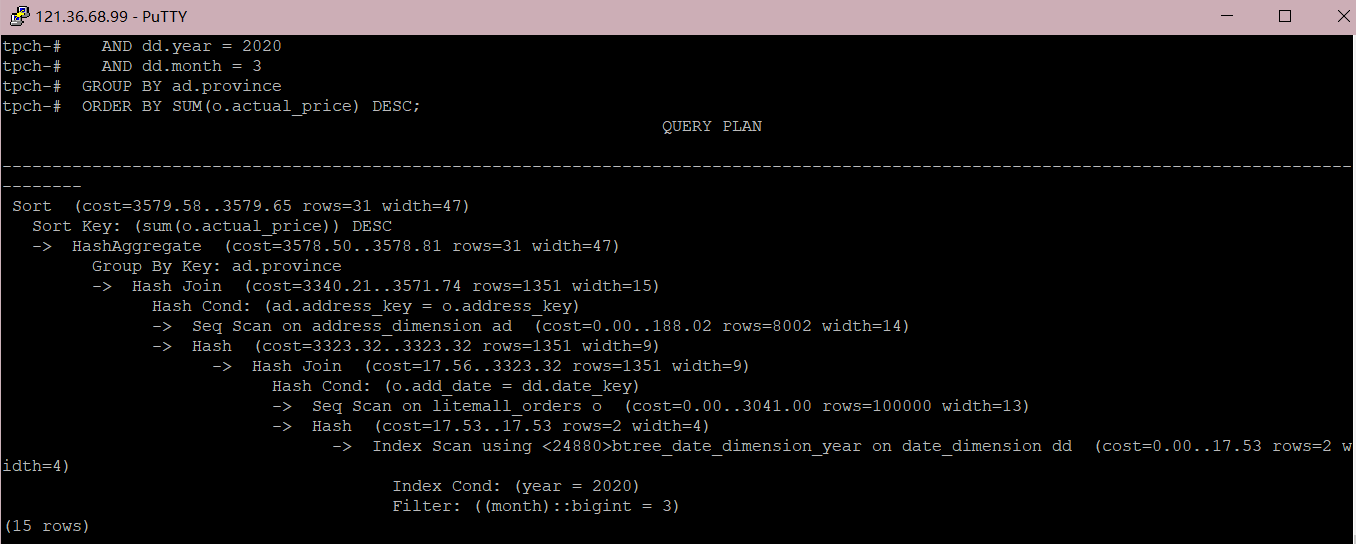
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

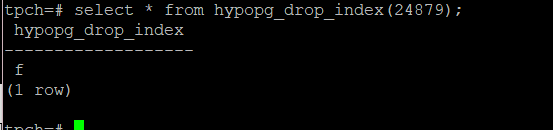
GROUP BY ad.province

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



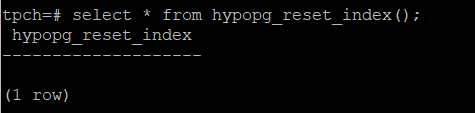
6. 删除某一个索引虚拟列，将执行结果截图。

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



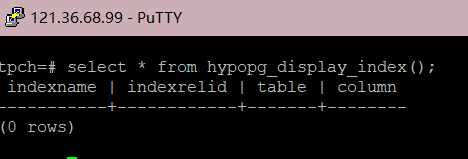
7. 删除所有索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

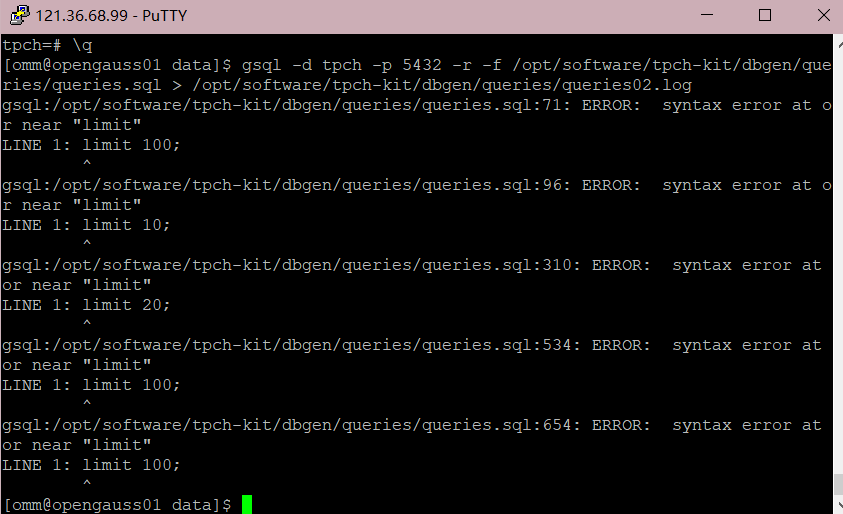
select \* from hypopg\_display\_index();

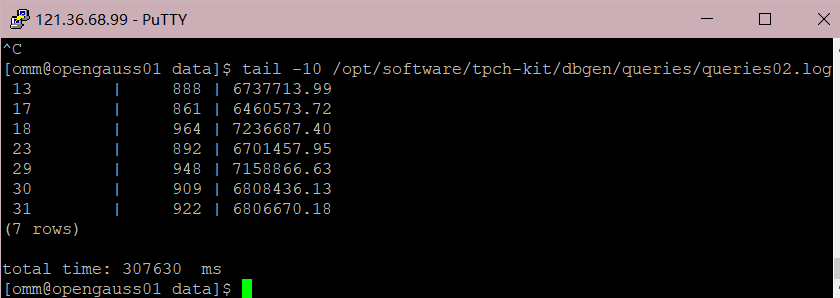


任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

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



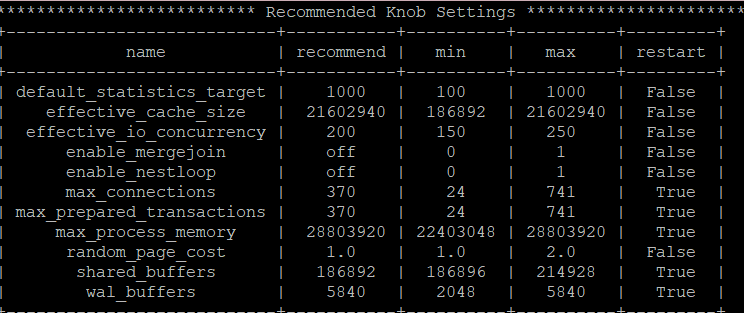


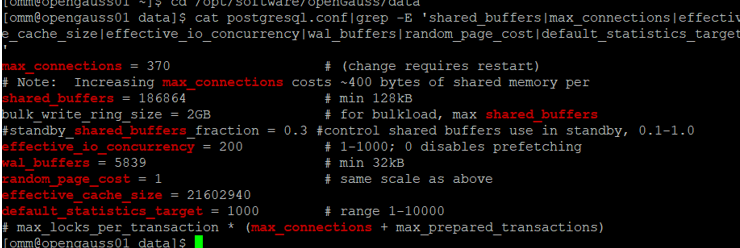
挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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？





优化参数：max\_connections, shared\_buffers, effective\_io\_concurrency, wal\_buffers, random\_page\_cost, effective\_cache\_size, default\_statistics\_target;

max\_connections：针对所有的账号所有的客户端并行连接到MYSQL服务的最大并行连接数。简单说是指MYSQL服务能够同时接受的最大并行连接数。 在测试的过程中，370个连接已经足够.

shared\_buffers：shared\_buffers是一个8KB的数组，postgres在从磁盘中查询数据前，会先查找shared\_buffers的页，如果命中，就直接返回，避免从磁盘查询。shared\_buffers存储表数据、索引、执行计划。在访问数据时，数据会先加载到os缓存，然后再加载到shared\_buffers。在IO压力很大的情况下，提高该值可以减少IO

effective\_io\_concurrency ：设置PostgreSQL可以同时被执行的并发磁盘 I/O 操作的数量;调高这个值，可以增加任何单个PostgreSQL会话试图并行发起的 I/O 操作的数目;允许的范围是 1 到 1000，或 0 表示禁用异步 I/O 请求。当前这个设置仅影响位图堆扫描。

wal\_buffers：可以降低IO，如果遇上比较多的并发短事务，应该和commit\_delay一起用。

random\_page\_cost ：设置从数据文件上随机读取一个数据块的执行成本。默认值是4.0。变量random\_page\_cost用于决定使用索引的代价是否值得，或者和seq\_page\_cost合作使用。

effective\_cache\_size：优化器假设一个查询可以用的最大内存，和shared\_buffers。设置稍大，优化器更倾向使用索引扫描而不是顺序扫描。

default\_statistics\_target：设置默认的收集优化器统计数据的目标值。它的值越大，ANALYZE操作的执行的时间越长，扫描的数据行的个数也就越多，得到的优化器统计数据就越准确。

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？

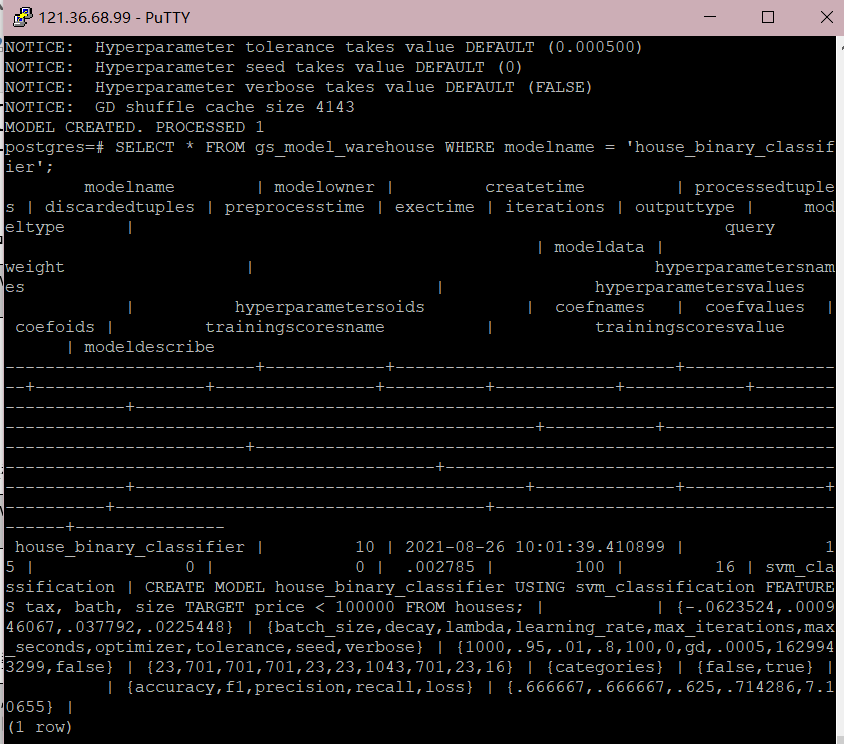
数据库索引是为了增加查询速度而对表字段附加的一种标识。DB在执行一条Sql语句的时候，默认的方式是根据搜索条件进行全表扫描，遇到匹配条件的就加入搜索结果集合。如果我们对某一字段增加索引，查询时就会先去索引列表中一次定位到特定值的行数，大大减少遍历匹配的行数，所以能明显增加查询的速度。

优化的其他方面：数据库表结构、系统配置、硬件、优化查询语句。

# 关卡四、openGauss的DB4AI特性应用

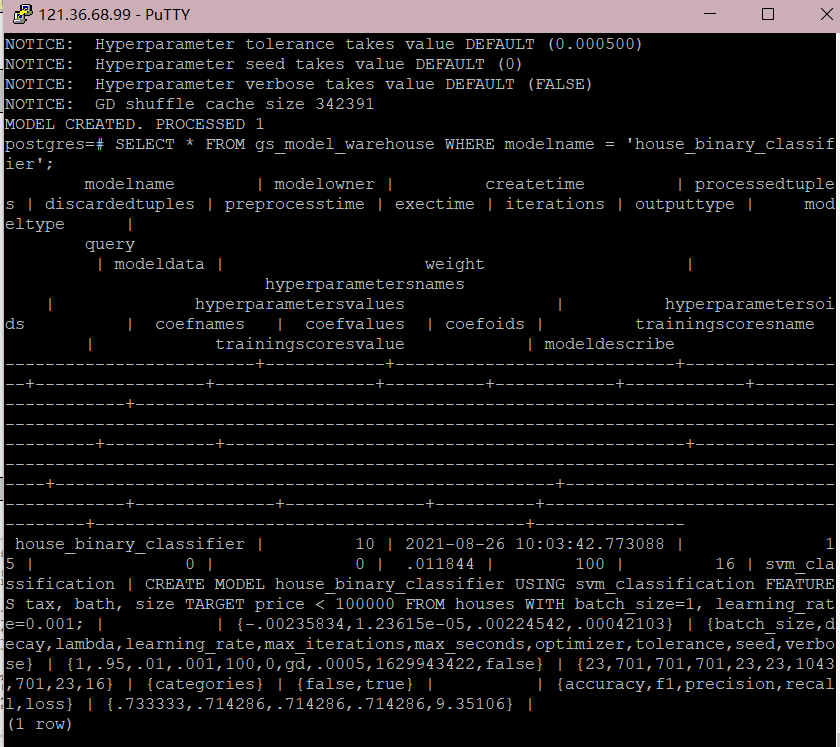
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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

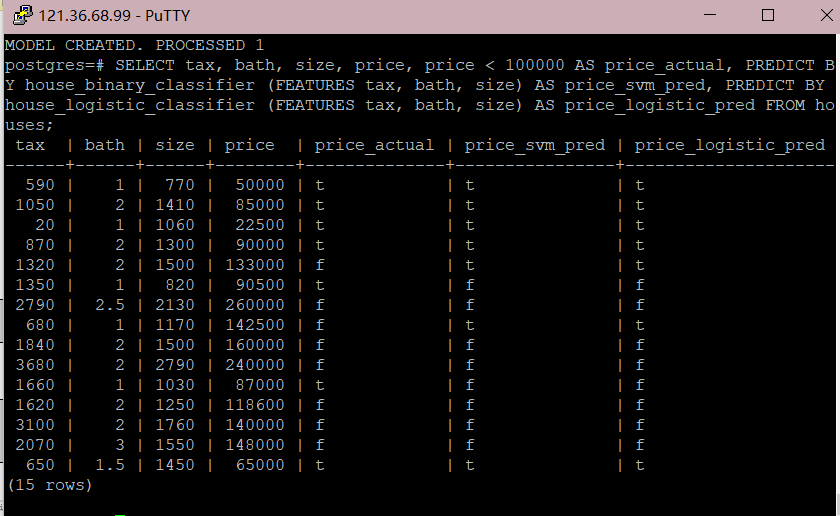


任务二：观察新模型的信息，将执行结果截图。

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;



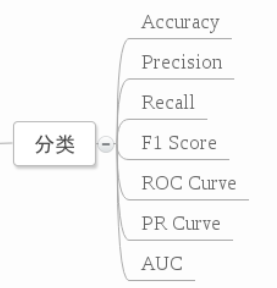
实践思考题1：分类模型与回归模型有何不同？

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。关键区别是目标属性是否连续。

实践思考题2：什么是SVM算法？

支持向量机(SVM)是90年代中期发展起来的基于统计学习理论的一种机器学习方法，通过寻求结构化风险最小来提高学习机泛化能力，实现经验风险和置信范围的最小化，从而达到在统计样本量较少的情况下，亦能获得良好统计规律的目的。该算法可以用于解决分类和回归问题,同时还使用了一种称之为kernel trick的技术进行数据的转换,然后再根据这些转换信息,在可能的输出之中找到一个最优的边界。

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？



准确率、精确率、召回率。

准确率：对于给定的测试数据集，分类器正确分类的样本数与总样本数之比。

精确率：所有"正确被检索的"占所有"实际被检索到的"的比例。

召回率：检索出的相关文档数和文档库中所有的相关文档数的比率，衡量的是检索系统的查全率。

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？



MAE:即平均绝对值误差，是一种用于回归模型的常用损失函数。MAE是目标值和预测值之差的绝对值之和。其只衡量了预测值误差的平均模长，而不考虑方向，取值范围也是从0到正无穷.

MSE:即均方误差，可以看做是一种L2损失，也是一种最常用的回归损失函数。MSE是求预测值与真实值之间距离的平方和。