openGauss AI特性创新实践课

姓名：和任强

学号：1120190159

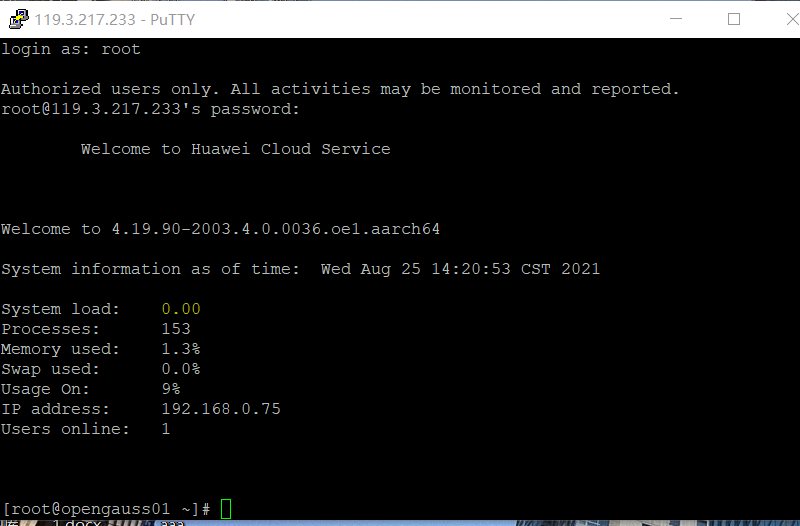


华为技术有限公司

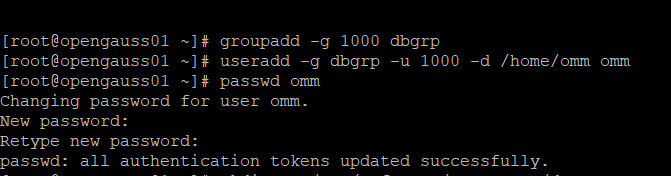
# 关卡一、openGauss数据安装及基本操作

openGauss数据安装及基本操作, 作业提交任务如下：

登录成功：

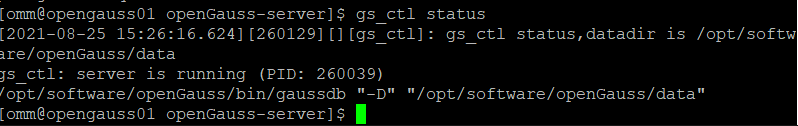


用户创建成功：



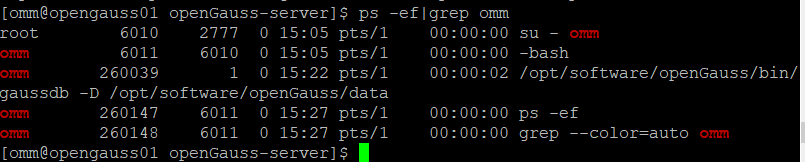
任务一：数据库状态验证

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：步骤 3 简单描述关卡一中，安装数据库所需要的步骤。完成以上信息，表明关卡一完成。

1. 使用putty和购买服务器时所用的账号密码远程登录服务器，并在putty进行命令行操作；  
2. 创建用数据的安装用户omm以及附属组，并修改用户密码  
3. 使用mkdir命令创建gauss数据库源码的存放文件夹  
4. 使用wget下载并解压第三方编译库，并重命名为binarylibs  
5. 下载openGauss源码  
6. 下载cmake并解压  
7. 使用yum下载python依赖包，并将自带python换成新下载的python3  
8. 切换至用户omm，配置数据库安装所需的环境变量并应有  
9. 进入数据库源码目录下，生成配置文件  
10. 使用make命令对源码进行编译以及安装  
11. 配置数据库hostname以及密码  
12. 启动数据库，数据库安装完成

实验思考题：为什么需要通过源码编译，安装数据库？

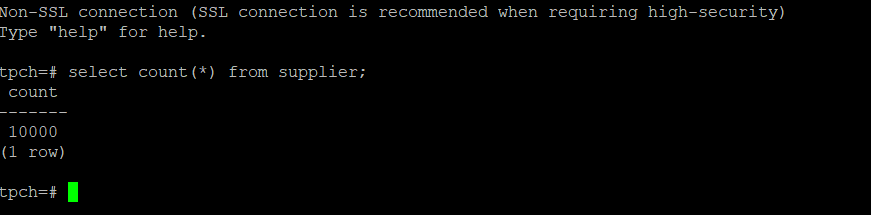
因为不同的操作系统的环境是不一样的，那么数据库需要的依赖包也可能在不同的主机上处于不同的位置，在进行编译的时候需要进行配置环境，这样就可以方便定位到依赖包所处在的具体位置。

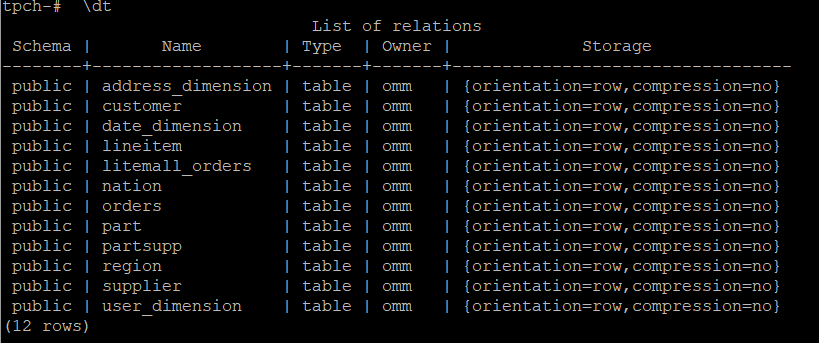
# 关卡二、openGauss数据导入及基本操作

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;;

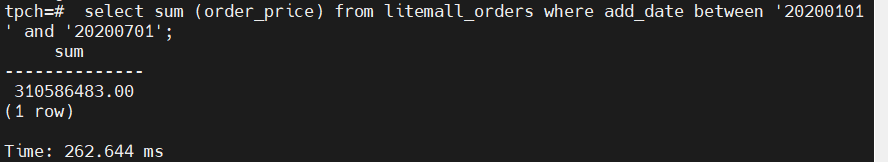




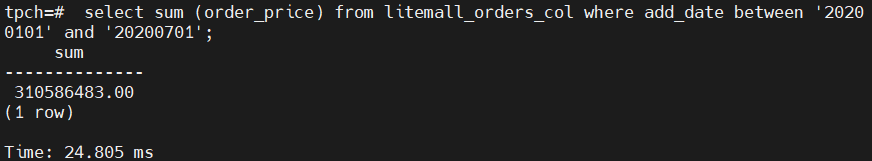
任务二：行存表与列存表执行效率对比

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';

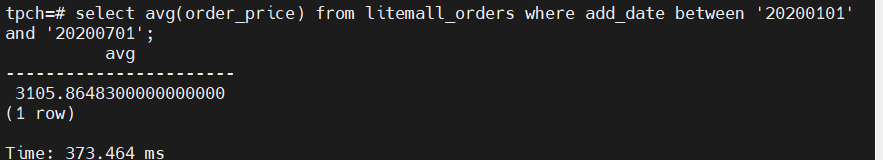


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

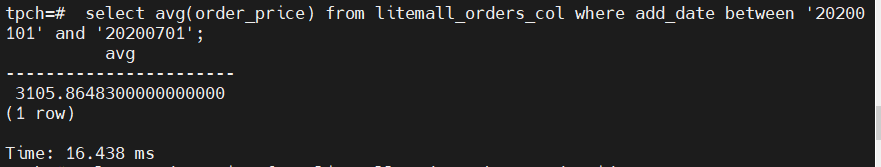


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

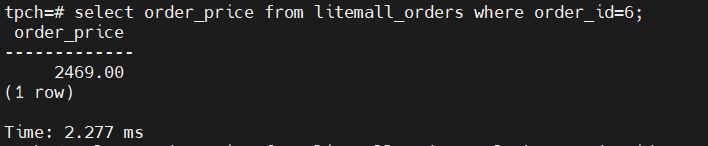


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

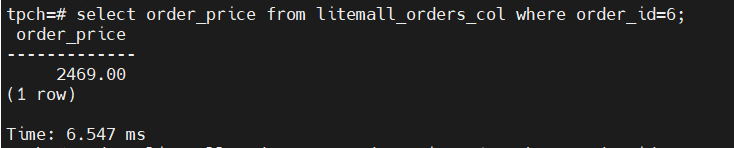


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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



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

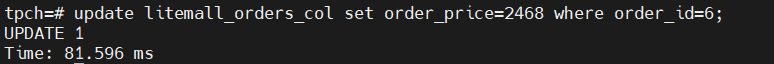


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

QQ截图20210825155559

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



任务三：物化视图的使用

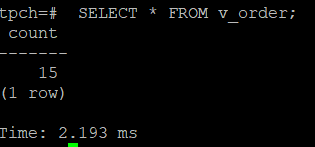
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



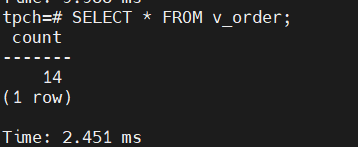
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



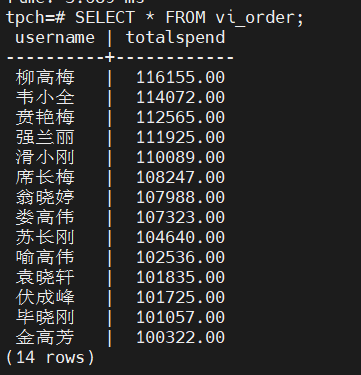
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



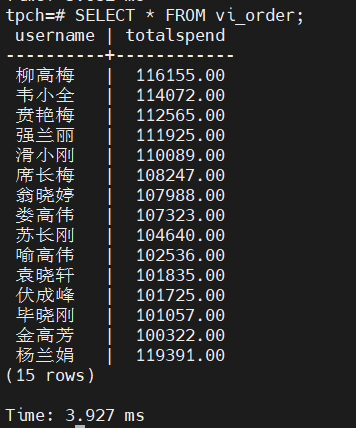
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;

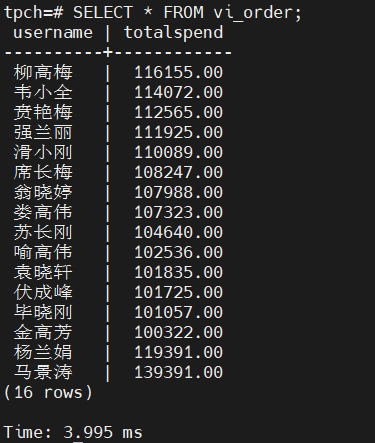


5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



6. 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。



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？

行存储是按照一个一个存储单元进行存储，一个存储单元就是又多个属性列构成；而列存储是按照一类一类属性列进行存储的，在进行查询，更改，添加，删除时，其执行的循序都不一样，比如在进行添加一个新数据，行存储可以直接添加在之前存储单元的后面，而列存储需要检查前面的列，在每个列的后面进行添加。

在执行查询类型语句的时候，列存表效率更高；在执行更新类型语句的时候，行存表效率更高。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

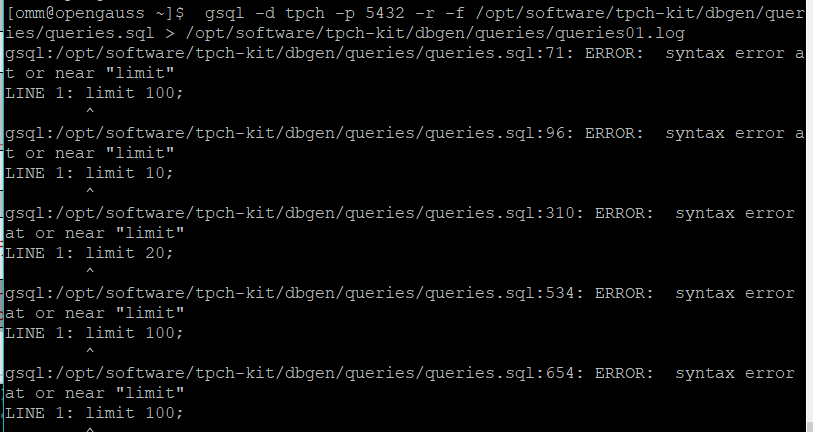
全量物化视图在进行刷新的时候将原先的物化视图数据全部删除掉，在重新从数据表中插入进来；而增量物化视图是根据物化视图的日志进行更新，在原本物化视图的基础上进行刷新，根据日志来进行增删改。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

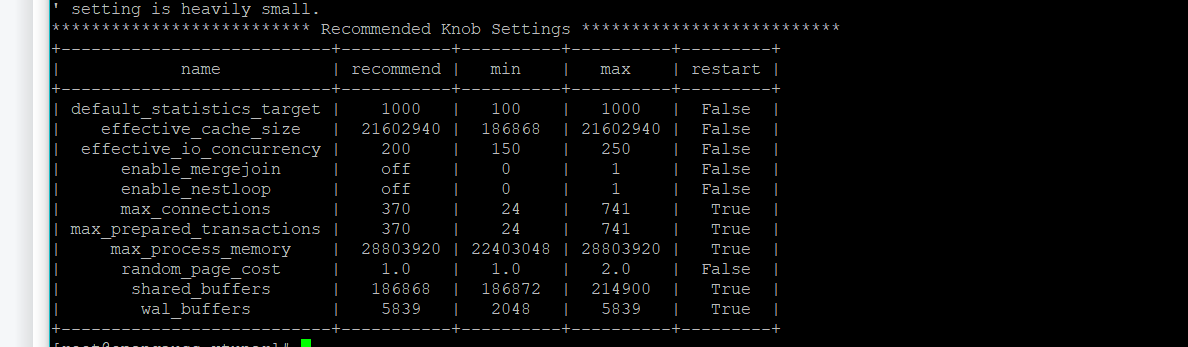
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

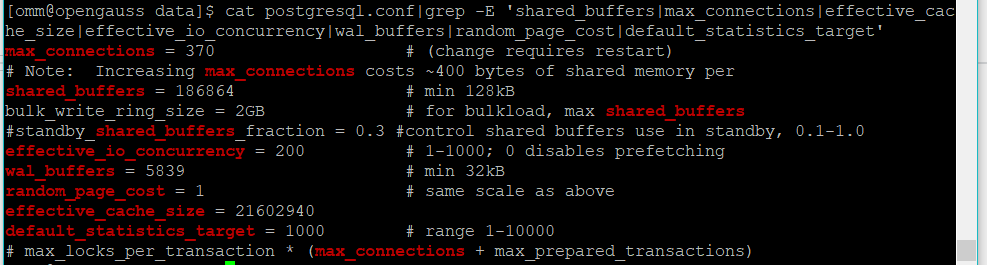
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

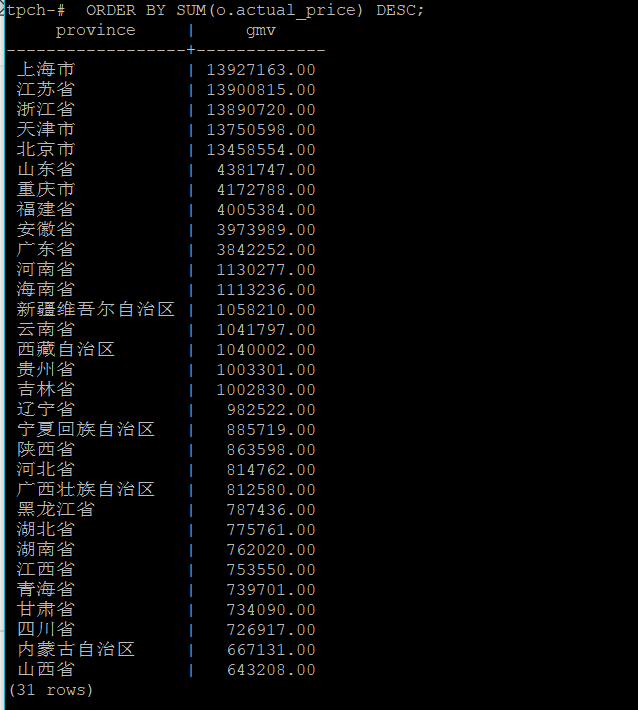
AND o.add\_date = dd.date\_key

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

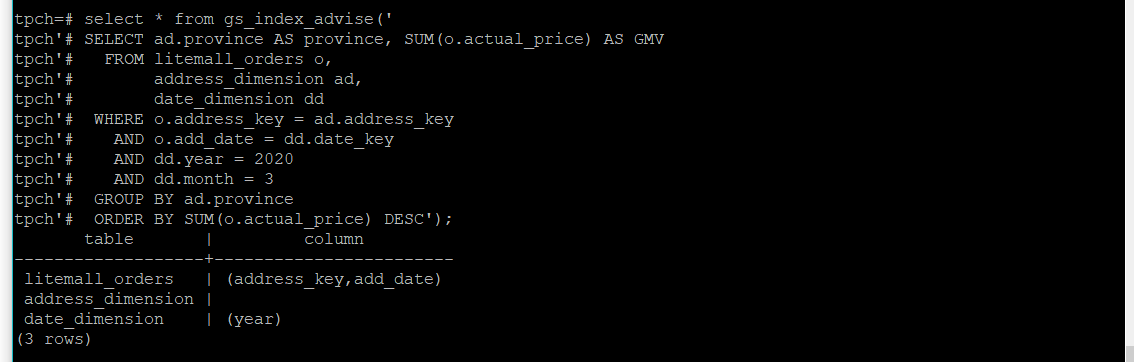
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

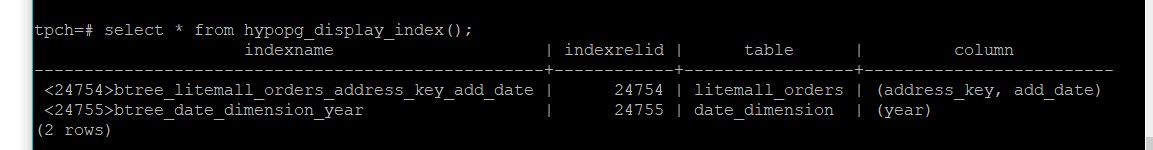
GROUP BY ad.province

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。

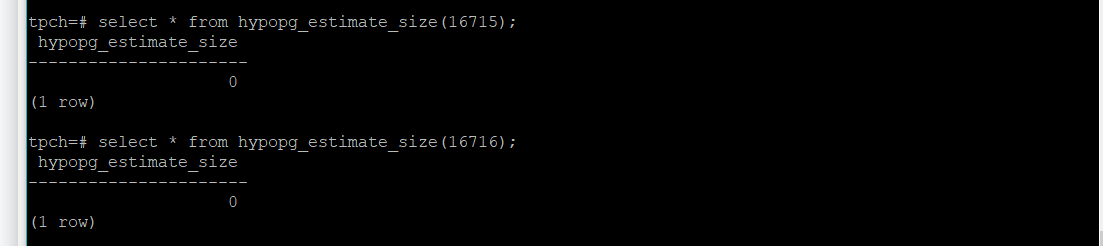
select \* from hypopg\_display\_index();



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

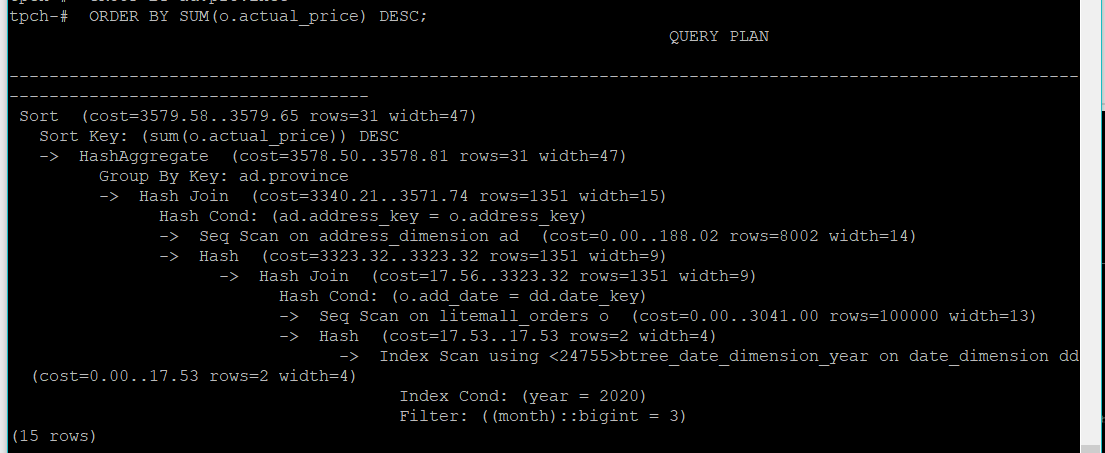
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

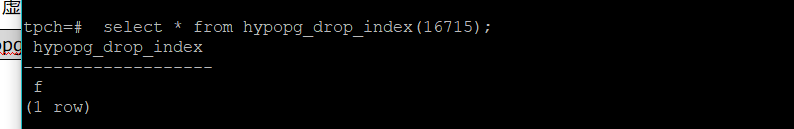
GROUP BY ad.province

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



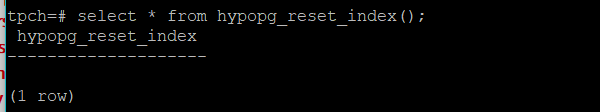
6. 删除某一个索引虚拟列，将执行结果截图。

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



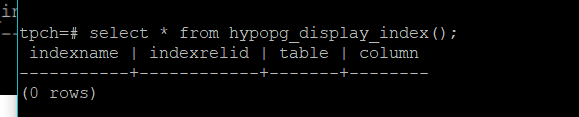
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

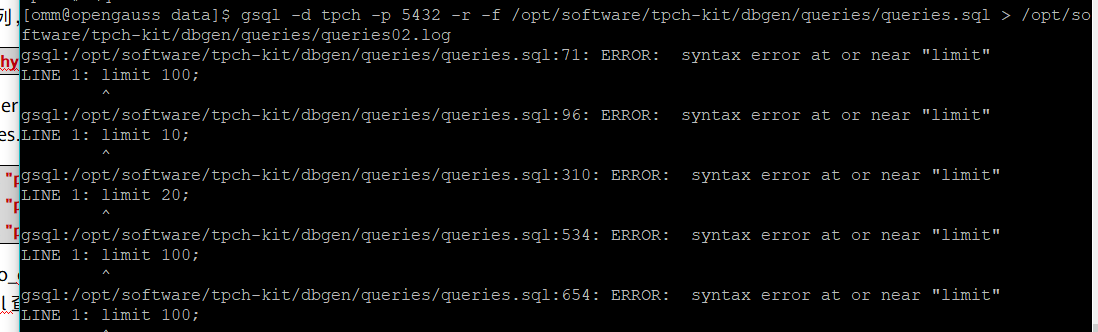
select \* from hypopg\_display\_index();

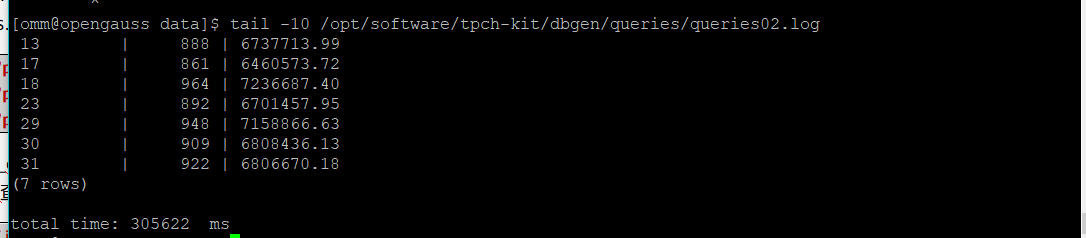


任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

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

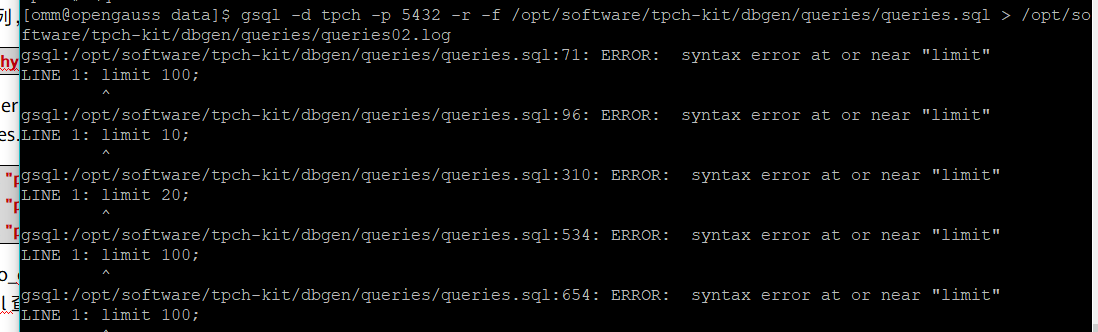




挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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？

对wal\_buffers,shared\_buffers这两个参数上进行了优化。对shared\_buffers进行调整，因为原先的数据页缓冲的默认值是偏低的，将该值调高，可以提高系统的整体效率。对wal\_buffers进行调整，调整预写日志缓冲区可以调整共享磁盘的大小，提高性能。

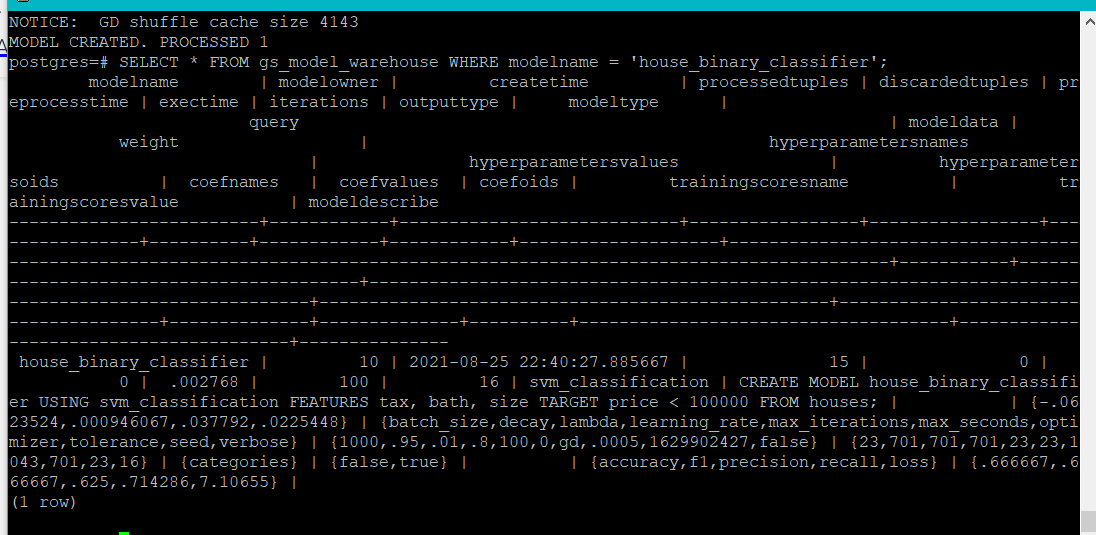
实践思考题2：索引的使用，对于执行SQL有什么好处？除了使用索引和参数外，还有哪些方面可以对数据库进行优化？

索引可以加快SQL的查询速度，原先查表的时候属于全表都进行遍历一边，在添加索引后，在进行查询时会优先对添加的索引列进行查询，在检测到该列对应的值时就可以查询到该行，减少了全局遍历的次数以及数量。还可以通过优化查询的SQL语句以及在设计数据表的时候通过范式来对数据表进行设计。

# 关卡四、openGauss的DB4AI特性应用

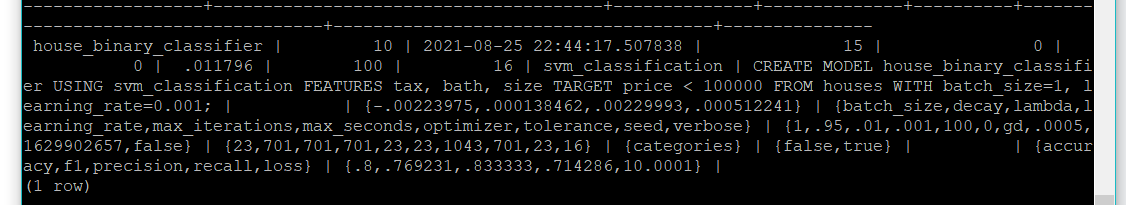
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



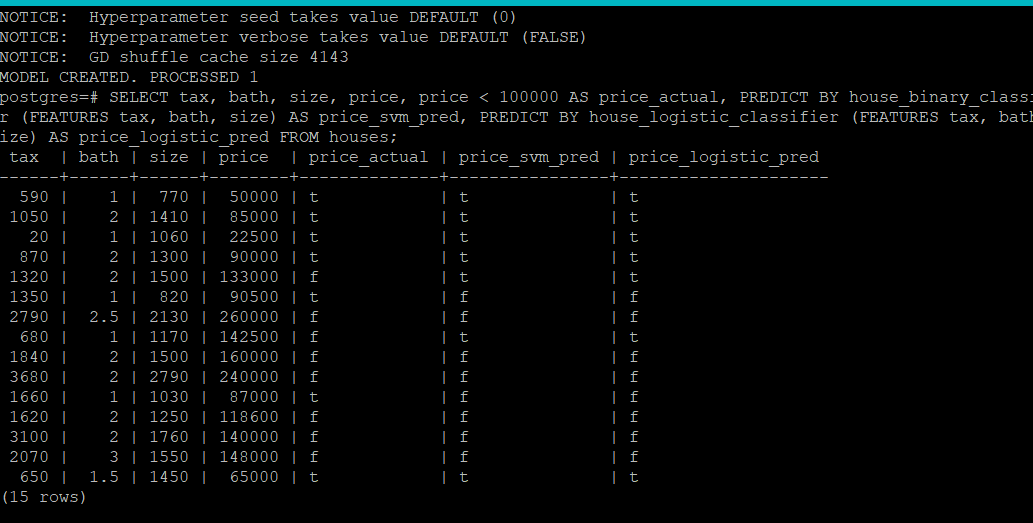
任务二：观察新模型的信息，将执行结果截图。

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？

* + - 1. 分类模型是用于离散化的数据；
      2. 回归模型是用于连续性的数据。

实践思考题2：什么是SVM算法？

SVM是支持向量机，通过寻找数据集空间中的一个分割线或者分割面来将数据集进行划分数据集。

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？

分类问题评价指标有：

准确率：就是能够正确分类的样本占总样本的比例。

召回率：就是能够正确分类的样本占原先实际样本的比例。

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？

回归问题评价指标有：

均方误差：真实值与预测值之间的差求和平均，表示预测值与实际值相差了多少。

决定系数：用来描述两个变量之间的相关性。