openGauss AI特性创新实践课



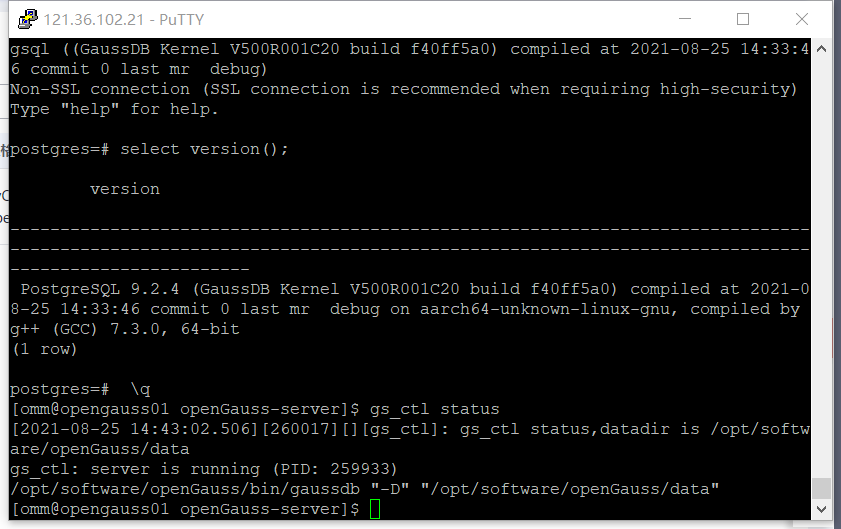
华为技术有限公司

# 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：

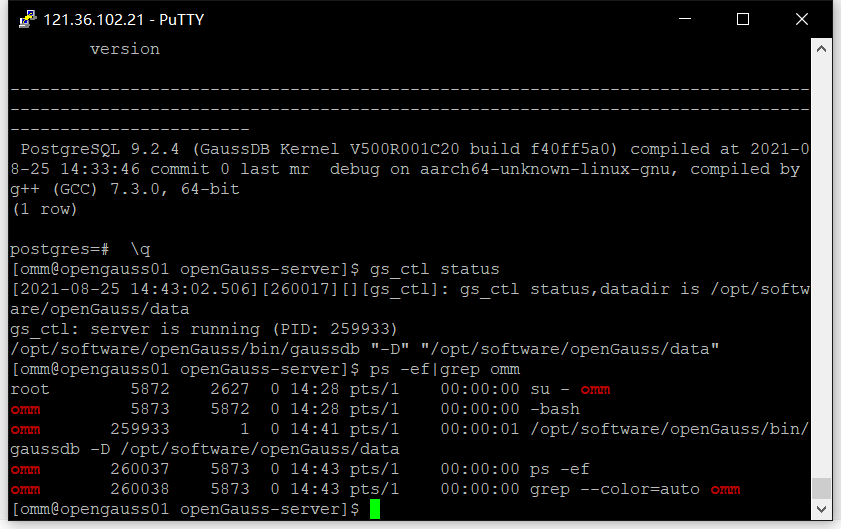
任务一：数据库状态验证

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？

通过源码编译安装的数据库版本更新，功能更多，没有依赖关系更适合自己的机器设置。

# 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：

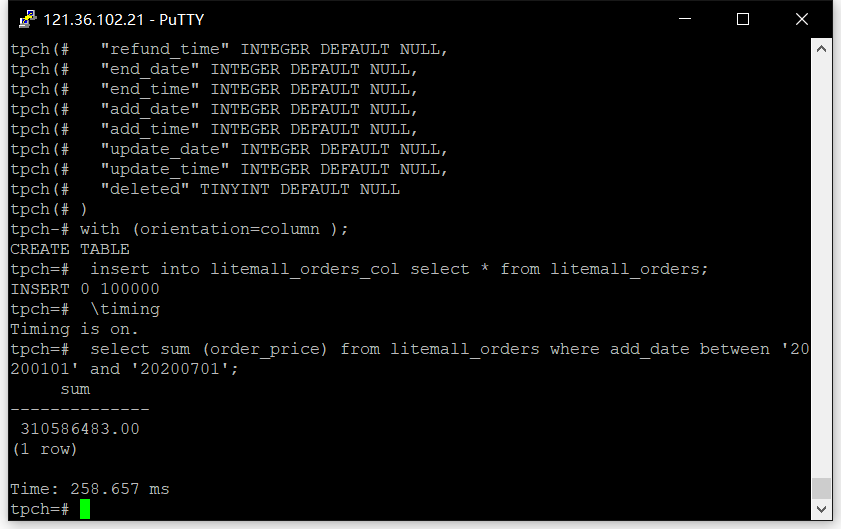
select count(\*) from supplier;;



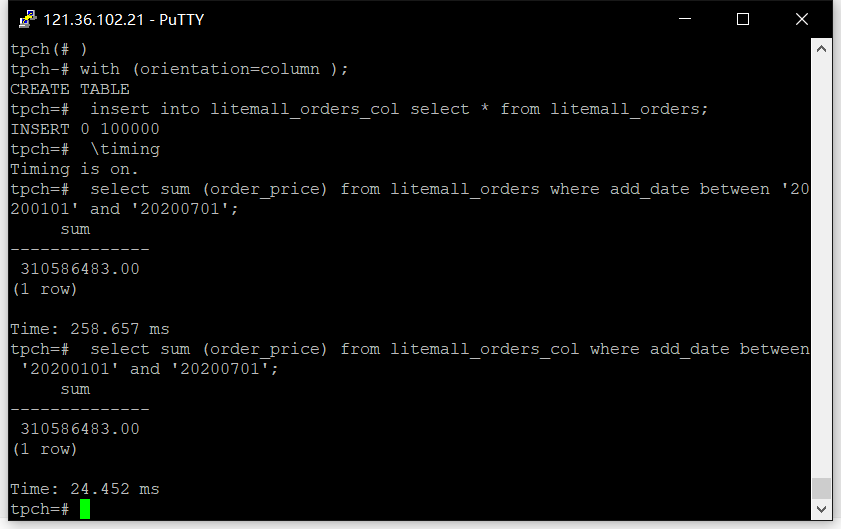
任务二：行存表与列存表执行效率对比

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';

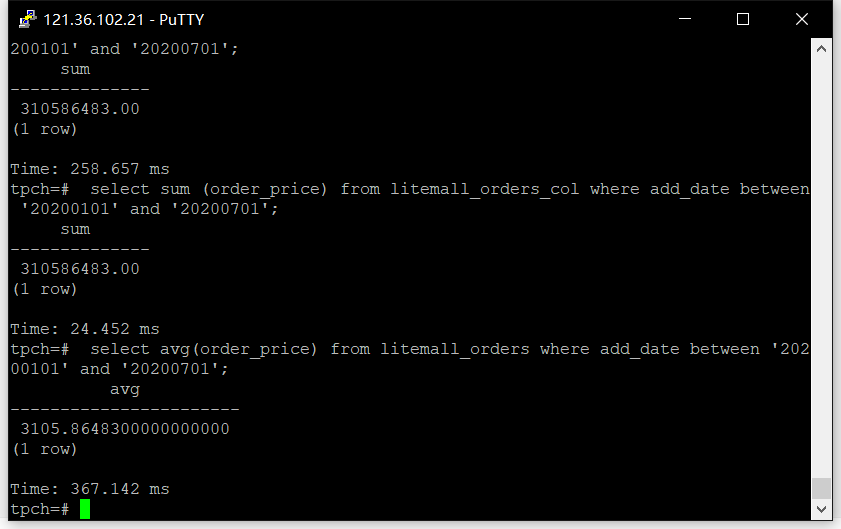


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

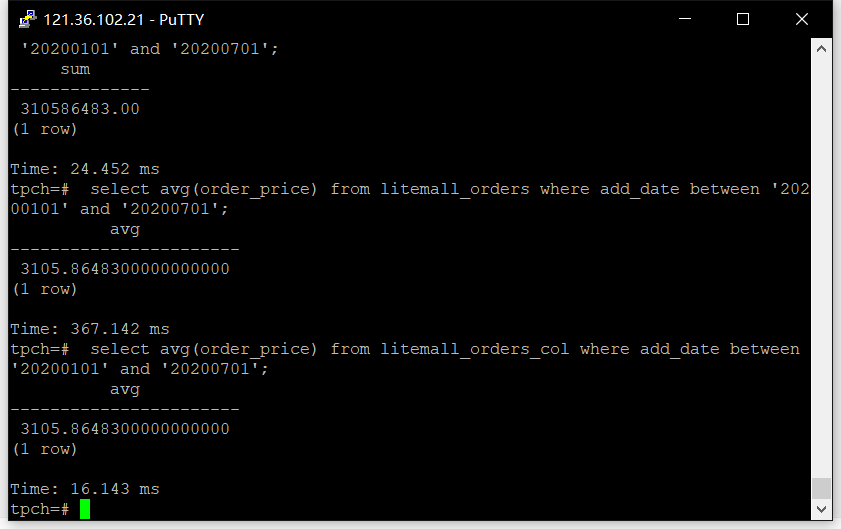


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

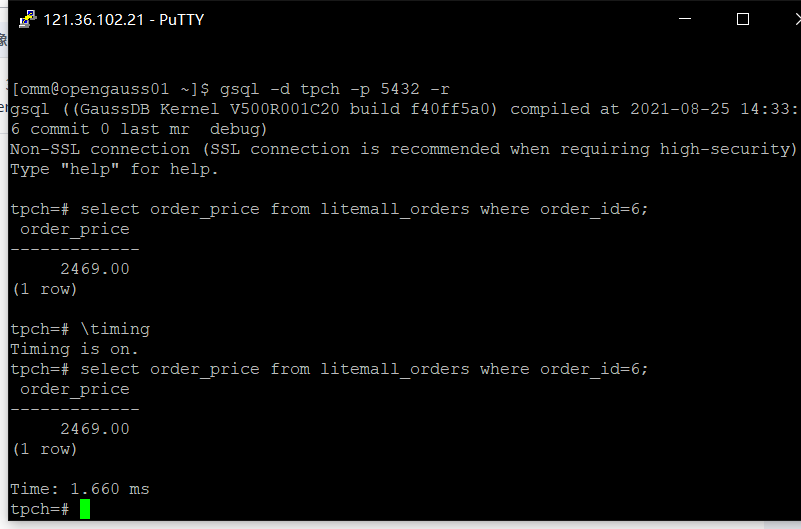


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

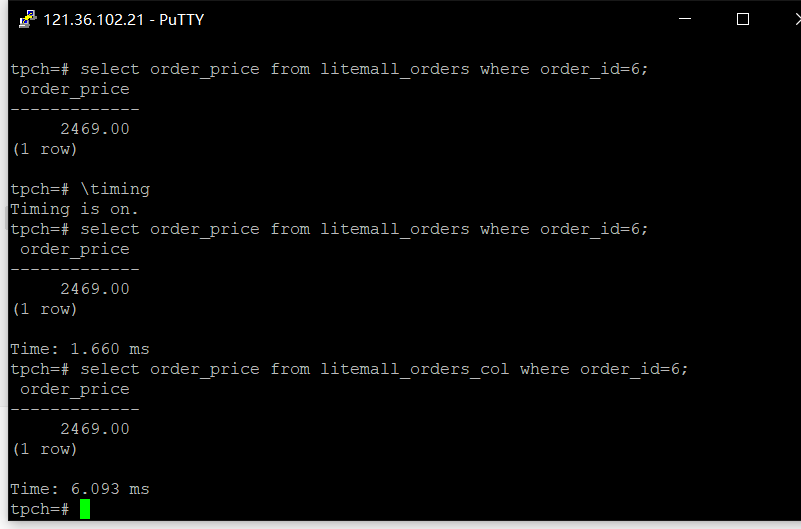


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

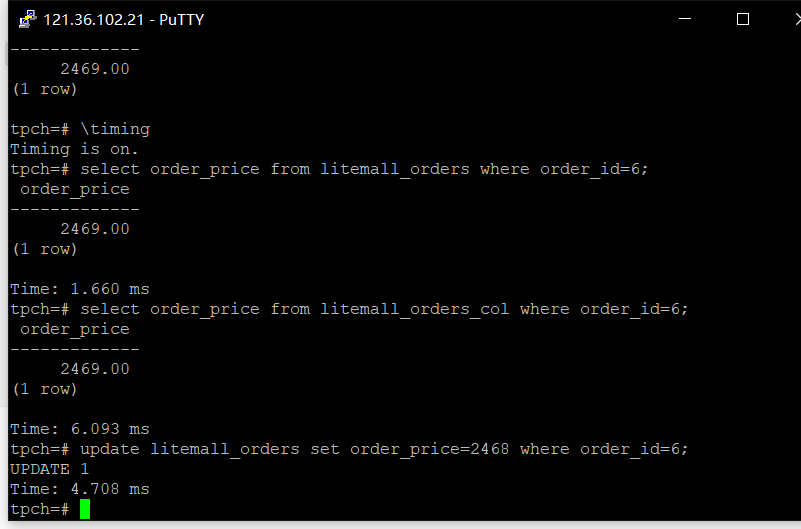


select order\_price from litemall\_orders\_col where order\_id=6;

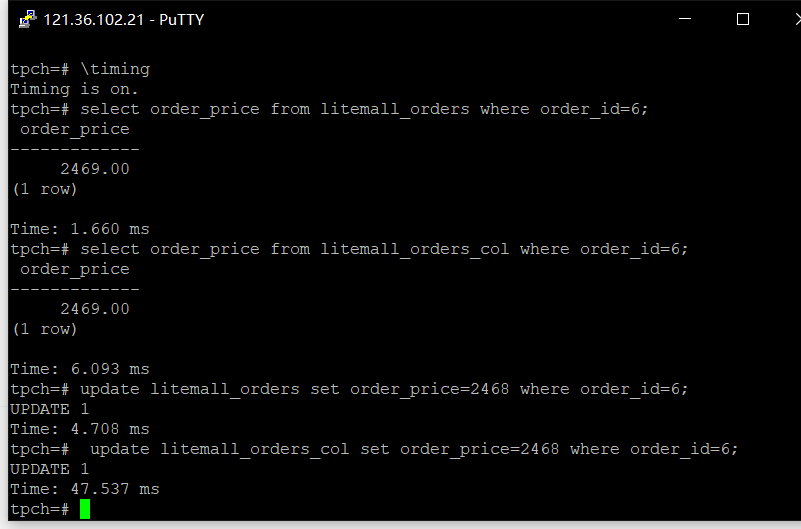


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



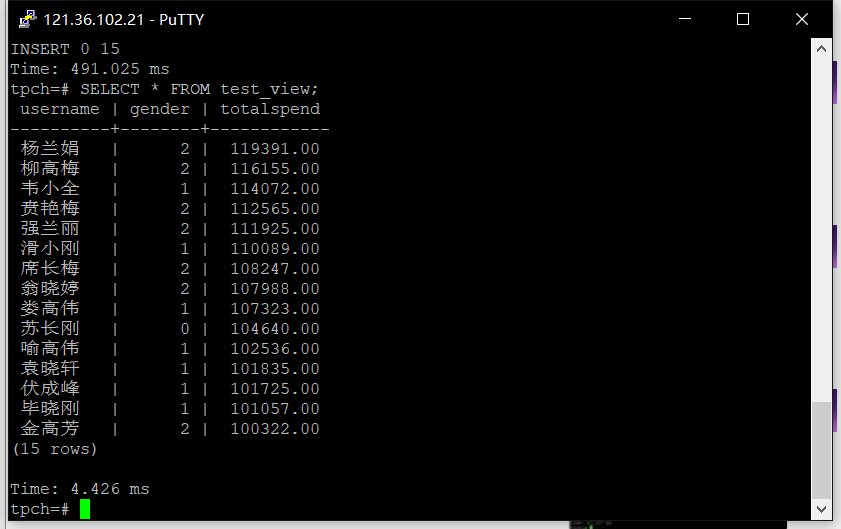
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

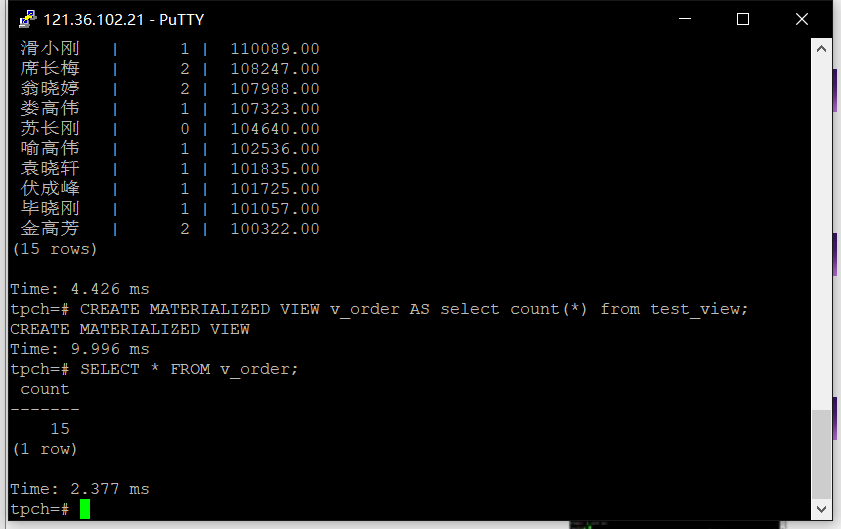
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



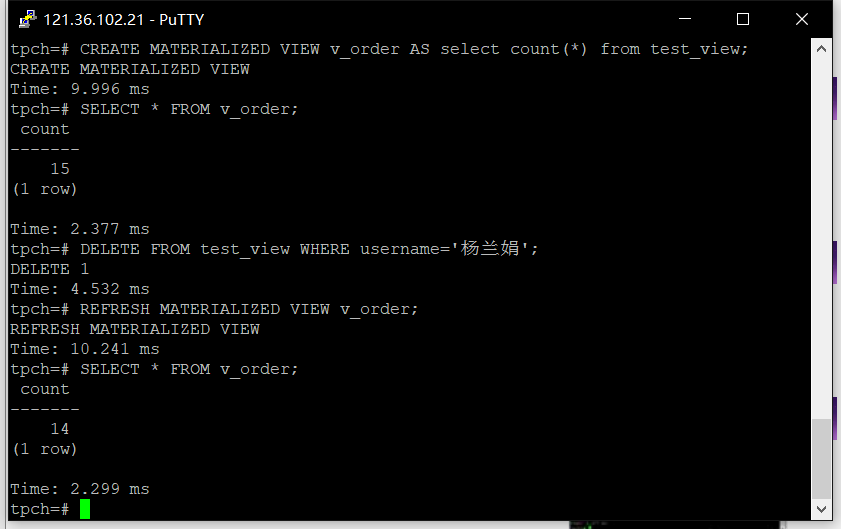
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



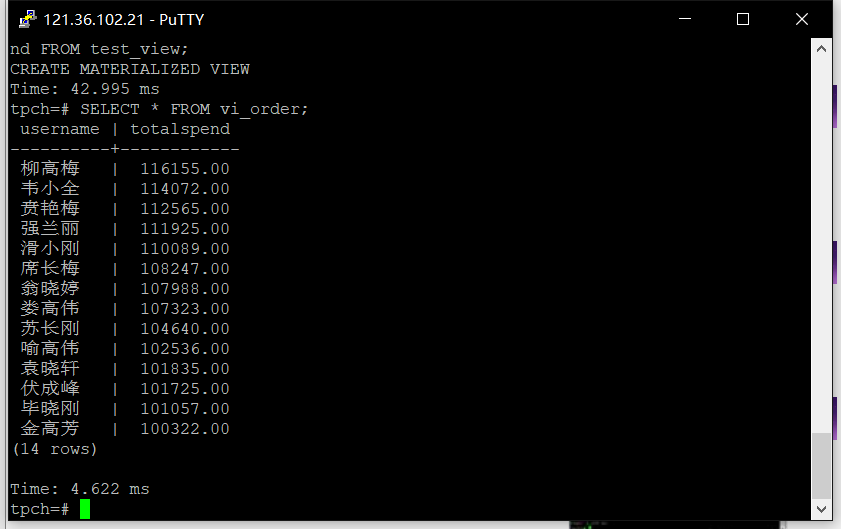
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



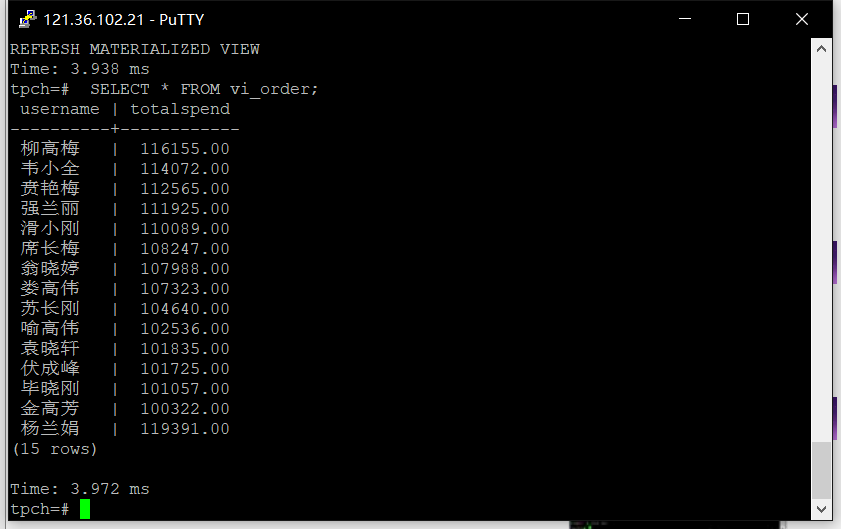
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

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，出于缩短处理时间的考量，消除冗余列的过程通常是在内存中进行的。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。两种存储的数据分布。由于列存储的每一列数据类型是同质的，不存在二义性问题。比如说某列数据类型为整型(int)，那么它的数据集合一定是整型数据。这种情况使数据解析变得十分容易。相比之下，行存储则要复杂得多，因为在一行记录中保存了多种类型的数据，数据解析需要在多种数据类型之间频繁转换，这个操作很消耗CPU，增加了解析的时间。所以，列存储的解析过程更有利于分析大数据。

行式存储的适用场景：

　　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。

列式存储的适用场景：

一般来说，一个OLAP类型的查询可能需要访问几百万甚至几十亿个数据行，且该查询往往只关心少数几个数据列。行式数据库需要读取所有的数据列。因此，列式数据库大大地提高了OLAP大数据量查询的效率。

对于数据仓库和分布式数据库来说，大部分情况下它会从各个数据源汇总数据，然后进行分析和反馈，其操作大多是围绕同一列属性的数据进行的，而当查询某属性的数据记录时，列式数据库只需返回与列属性相关的值，在大数据量查询场景中，列式数据库可在内存中高效组装各列的值，最终形成关系记录集，因此可以显著减少IO消耗，并降低查询响应时间，非常适合数据仓库和分布式的应用。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

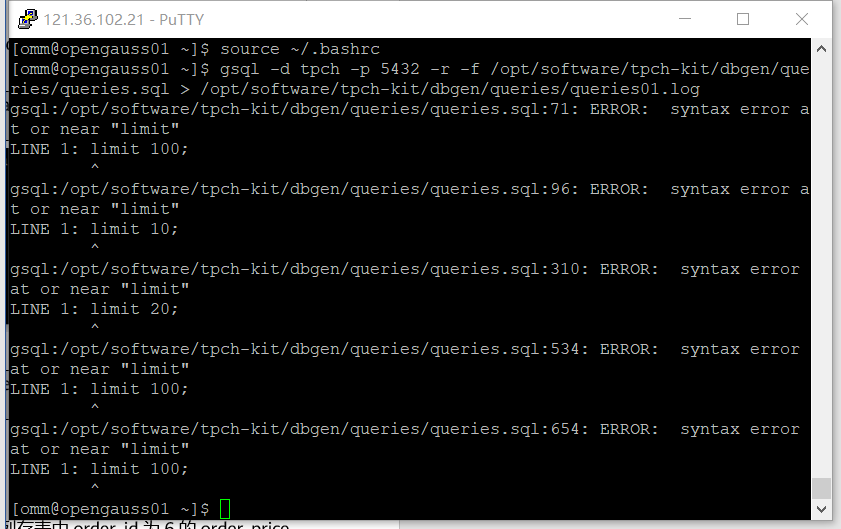
全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新。创建全量物化视图语法和CREATE TABLE AS语法一致，不支持对全量物化视图指定NodeGroup创建。增量物化视图顾名思义就是可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。与全量创建物化视图不同在于目前增量物化视图所支持场景较小。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

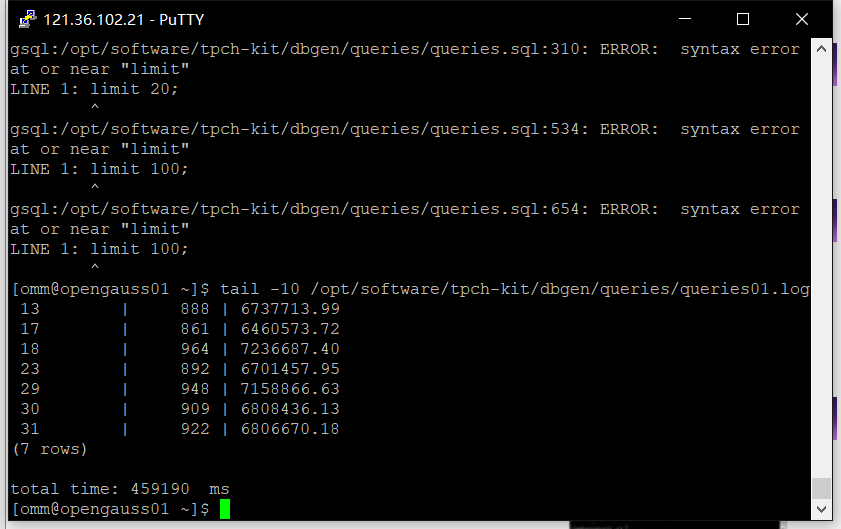
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

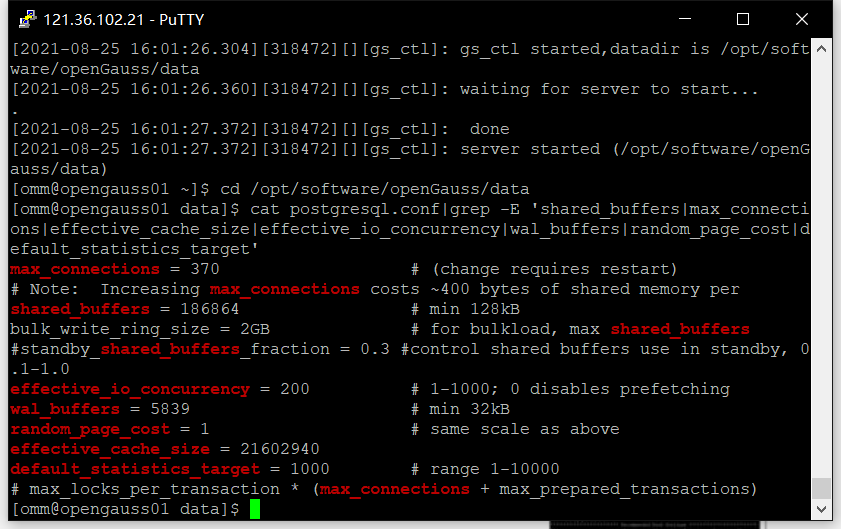
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

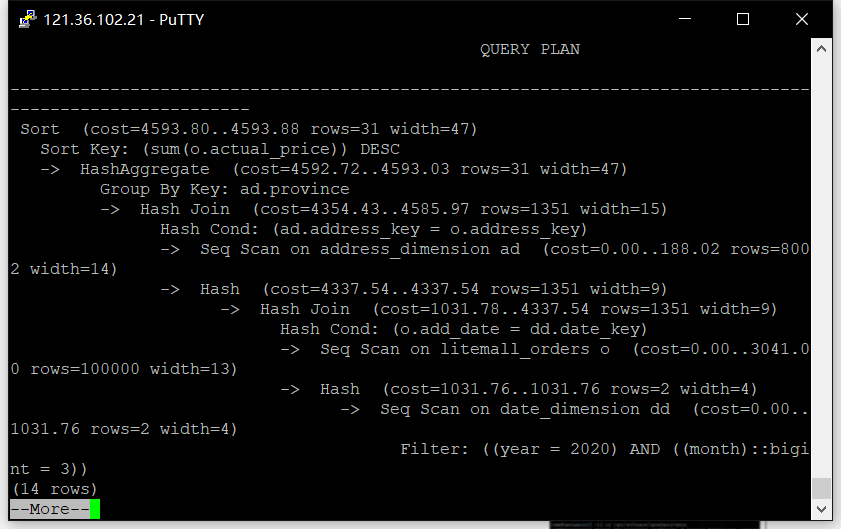
AND o.add\_date = dd.date\_key

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

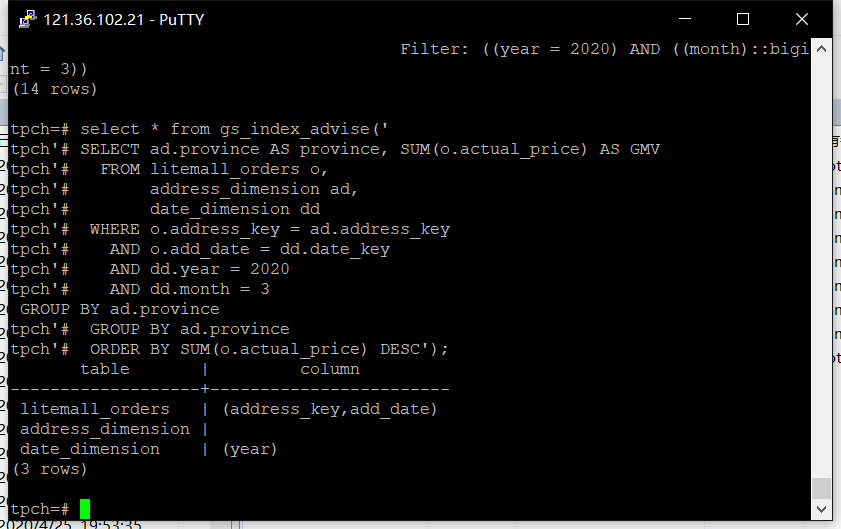
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

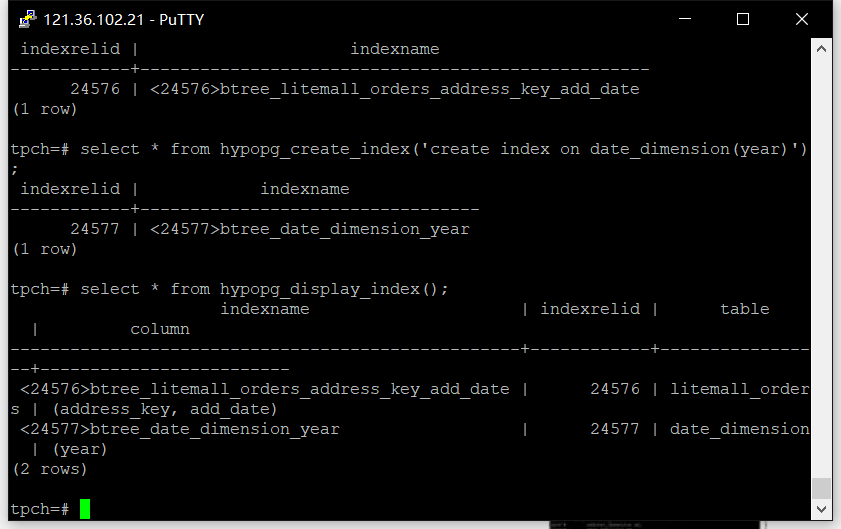
GROUP BY ad.province

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。

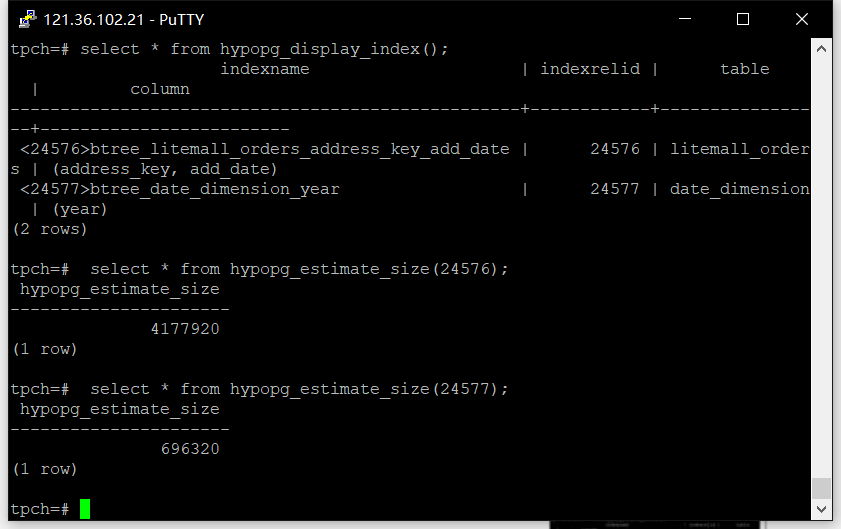
select \* from hypopg\_display\_index();



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

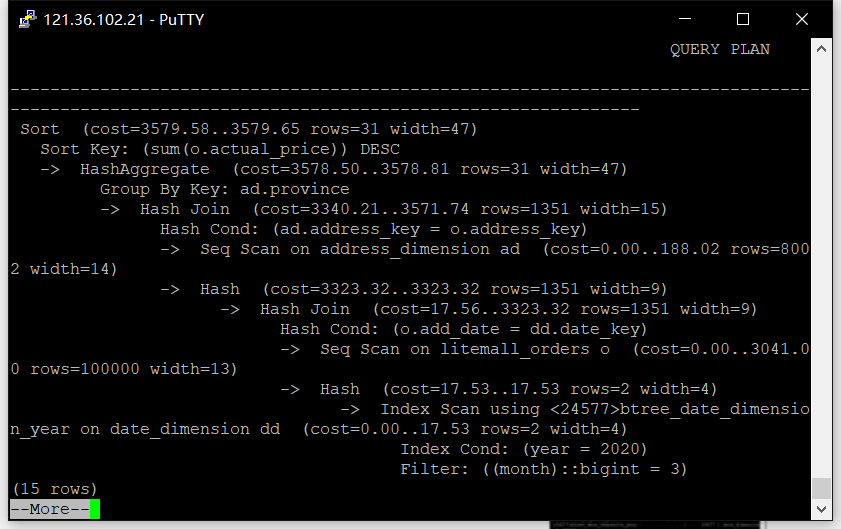
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

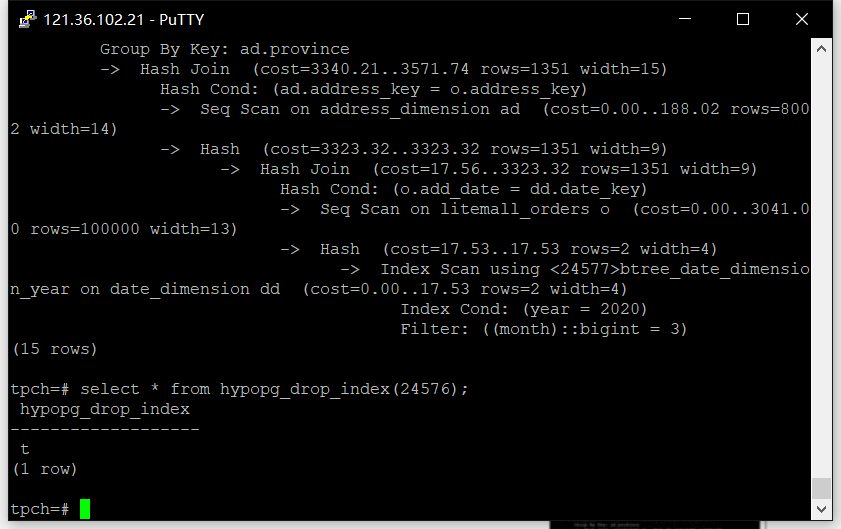
GROUP BY ad.province

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



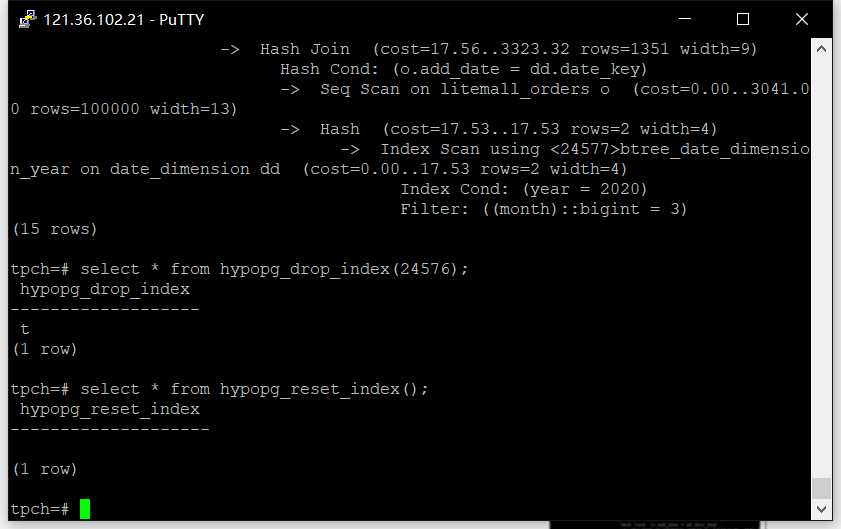
6. 删除某一个索引虚拟列，将执行结果截图。

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



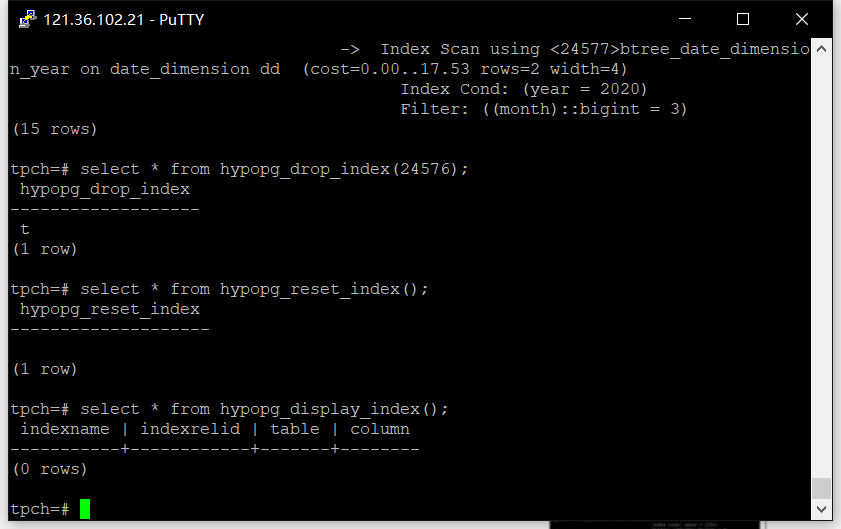
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

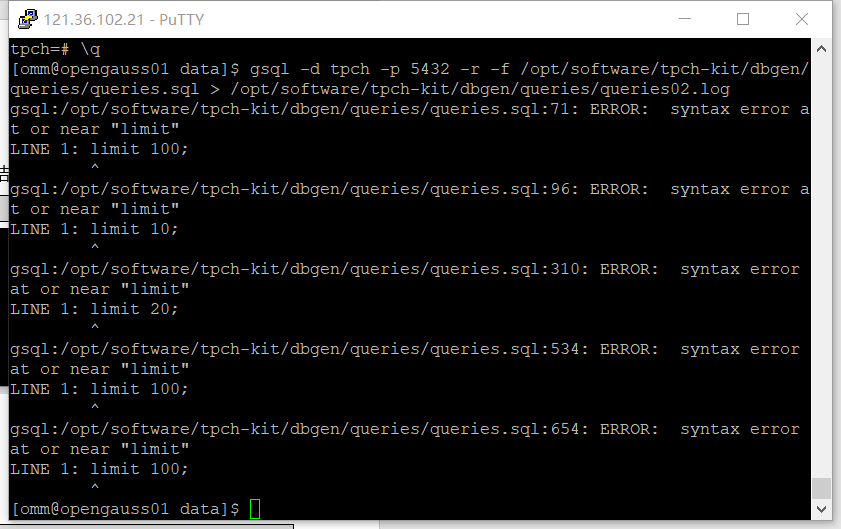
select \* from hypopg\_display\_index();

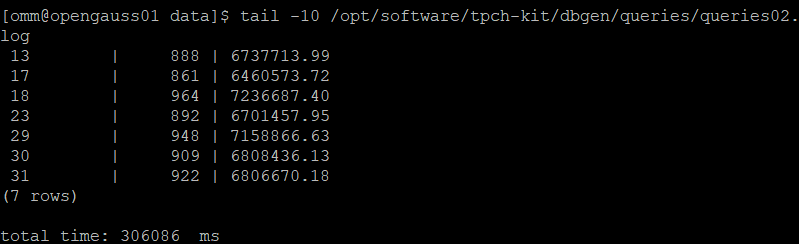


任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

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

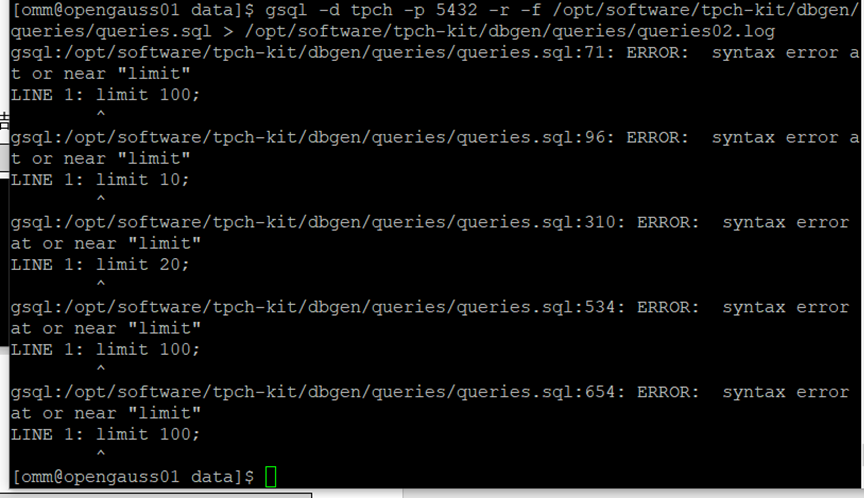


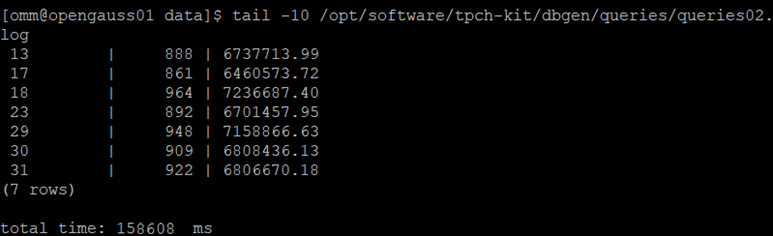


挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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





提高了约50%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？

Shared\_buffer

shared\_buffer是数据页缓冲区。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，OpenGuass将数据缓存在RAM中，来提高性能。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。OpenGuass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，更新shared\_buffers是提高整体性能最有效的设置之一。

Wal\_buffer

wal buffer是预写日志(wal)缓冲区，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。

effective\_cache\_size

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。它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。如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effective\_cache\_size的保守值是系统上可用内存总量的1/2。最常见的情况是，该值被设置为专用DB服务器上系统总内存的75%，但根据特定服务器工作负载上的特定离散需求，该值可以有所不同。​

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？

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。

优化方法：

1、选取最适用的字段属性；

2、使用连接（JOIN）来代替子查询(Sub-Queries)；

3、使用联合(UNION)来代替手动创建的临时表；

4、事务；

5、锁定表；

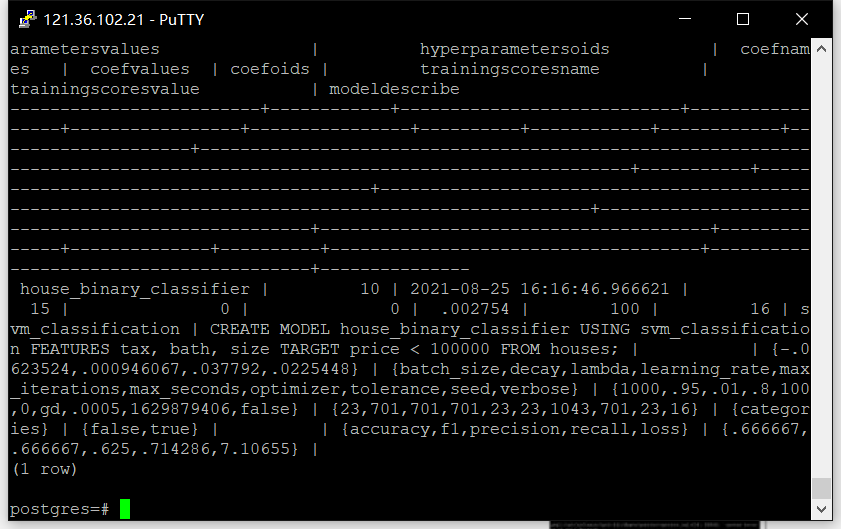
6、使用外键；

7、优化的查询语句.

# 关卡四、openGauss的DB4AI特性应用

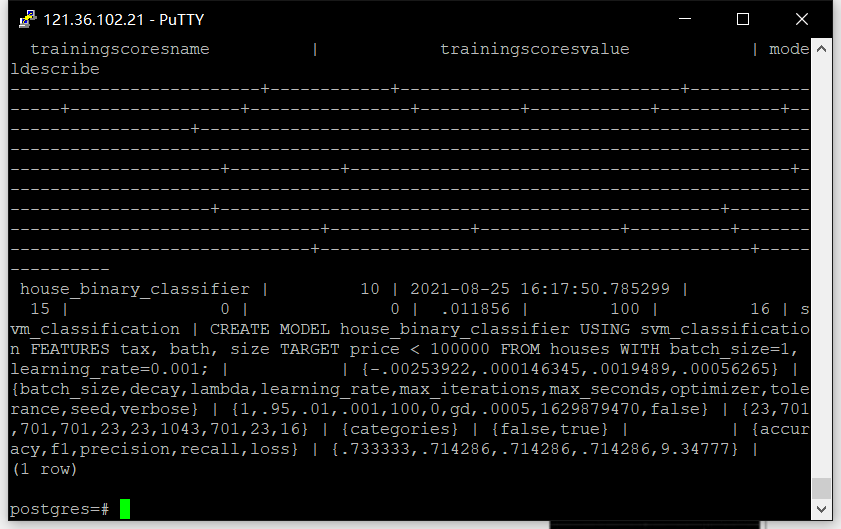
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



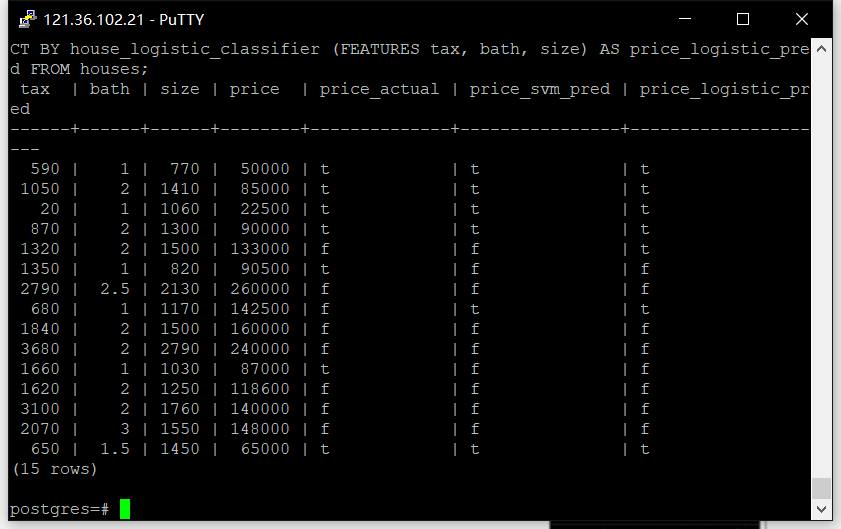
任务二：观察新模型的信息，将执行结果截图。

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;

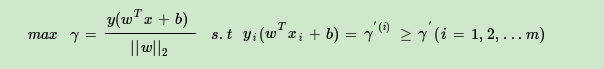


实践思考题1：分类模型与回归模型有何不同？

分类和回归的区别在于输出变量的类型。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。

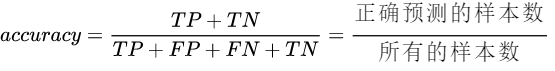
实践思考题2：什么是SVM算法？

SVM是一个二元分类算法，线性分类和非线性分类都支持。经过演进，现在也可以支持多元分类，同时经过扩展，也能应用于回归问题。SVM的模型是让所有点到超平面的距离大于一定的距离，也就是所有的分类点要在各自类别的支持向量两边。用数学式子表示为：



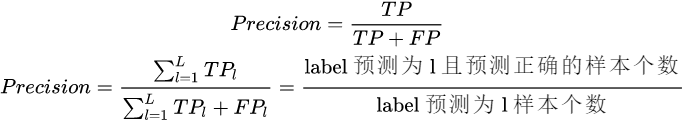
实践思考题3：分类问题有哪些评价指标，请分别说明他们的含义？

Accuracy：

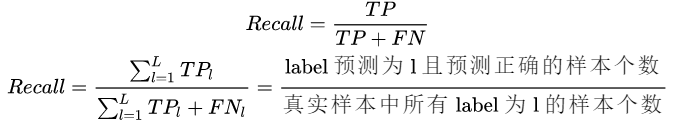


Accuracy 能够清晰的判断我们模型的表现，但有一个严重的缺陷： 在正负样本不均衡的情况下，占比大的类别往往会成为影响 Accuracy 的最主要因素，此时的 Accuracy 并不能很好的反映模型的整体情况。

Precision：

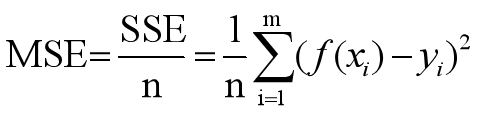


Recall：

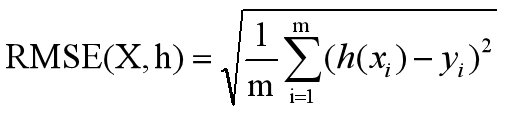


实践思考题4：回归问题有哪些评价指标，请分别说明他们的含义？

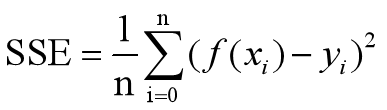
均方误差：是反映估计值与被估计量之间差异程度的一种度量。



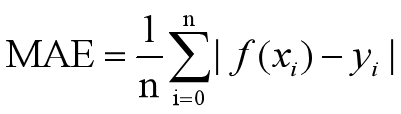
RMSE均方根误差：观测值与真值偏差的平方和与观测次数m比值的平方根，用来衡量观测值同真值之间的偏差。



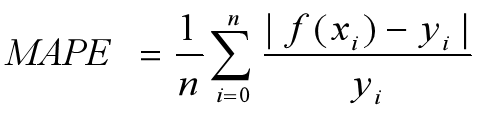
SSE和方误差



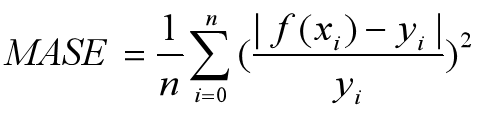
MAE：直接计算模型输出与真实值之间的平均绝对误差



MAPE：不仅考虑预测值与真实值误差，还考虑了误差与真实值之间的比例。



平均平方百分比误差



决定系数

