openGauss AI特性创新实践课



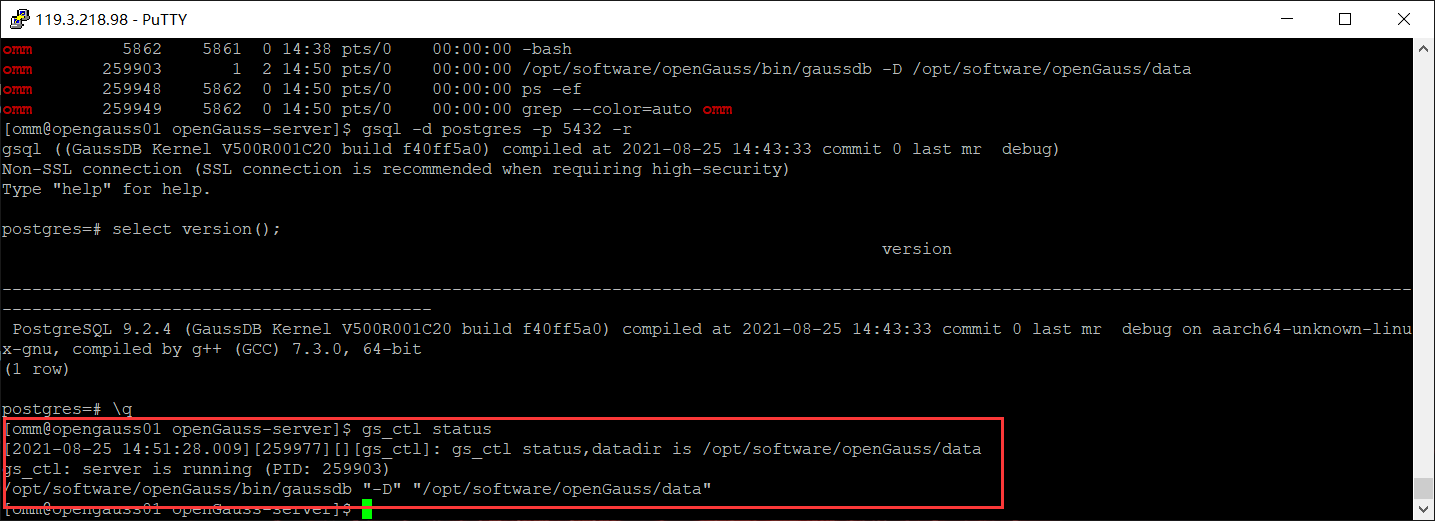
华为技术有限公司

# 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：

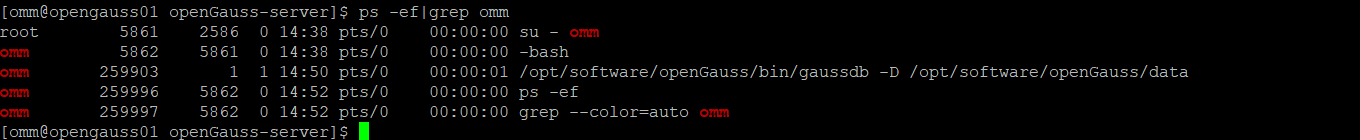
任务一：数据库状态验证

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？

1. 源代码可以允许用户选择软件编译配置，不同用户的linux内核版本可能不同，计算机的硬件也不同，编译器和依赖库的版本也会不同，用源码包可以在configure中自定义。
2. 对程序员来说，直接的产出就是源代码，直接把源代码扔给你是『最省事』的方式。而且鉴于一些开源协议的关系，很多 Linux 程序是必须发布源代码才合法的。所以，发布源代码是必须的。

简单描述安装数据库所需要的步骤：

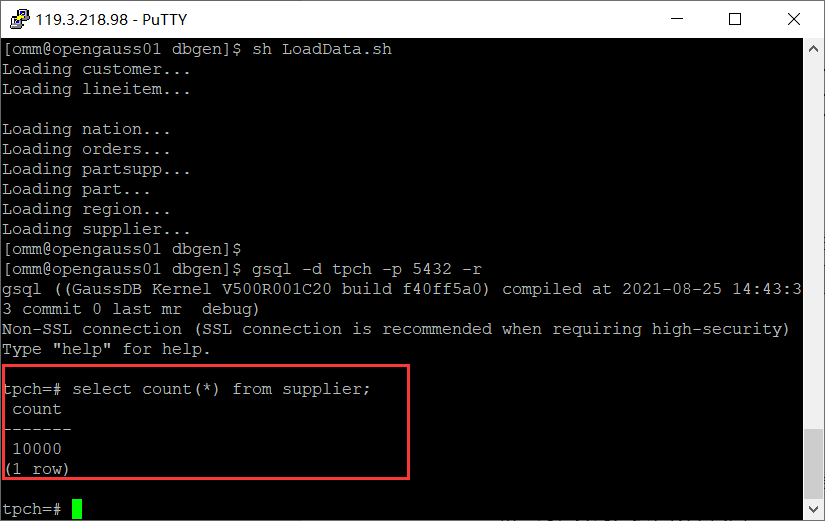
1. 安装好第三方编译库binarylibs和编译管理器cmake
2. 下载openGauss数据库的源码
3. 安装对应依赖包
4. 设置用户的环境变量且使生效
5. 进入Openguass源码下生成配置文件
6. 使用make+cmake编译且安装数据库

# 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：

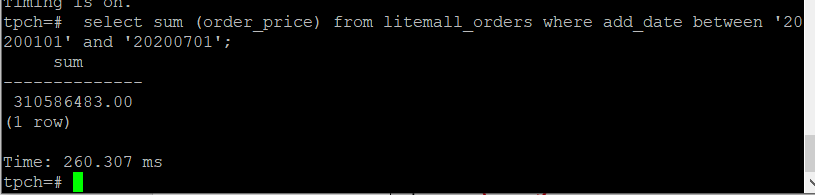
select count(\*) from supplier;;



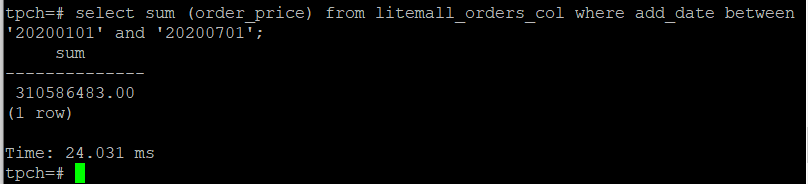
任务二：行存表与列存表执行效率对比

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';

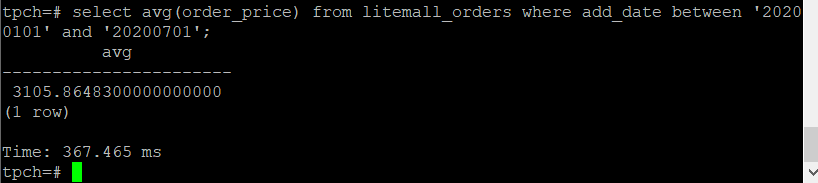


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

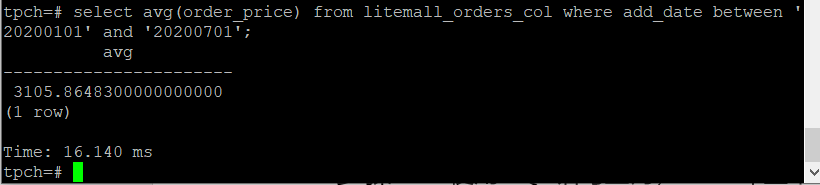


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

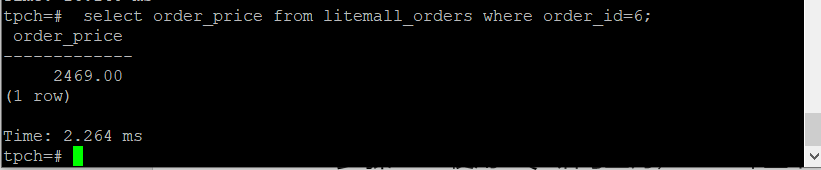


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

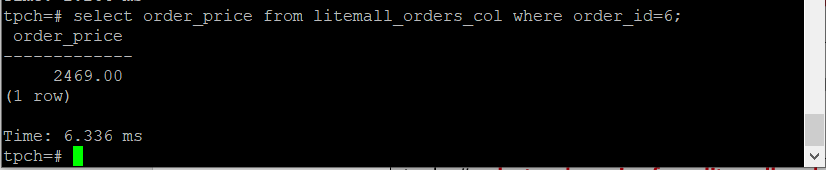


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

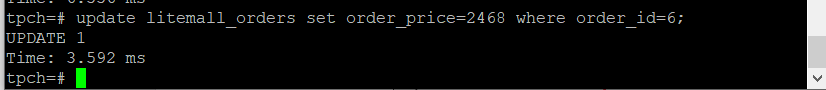


select order\_price from litemall\_orders\_col where order\_id=6;

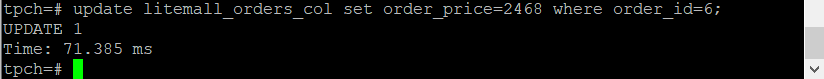


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



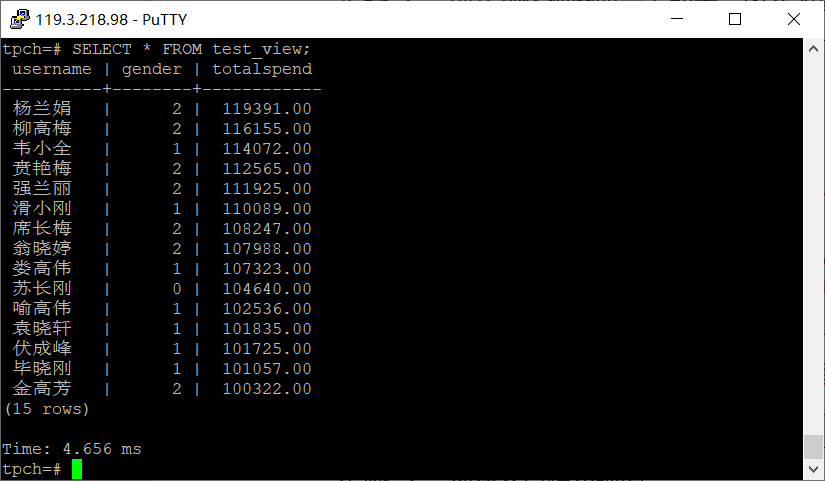
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

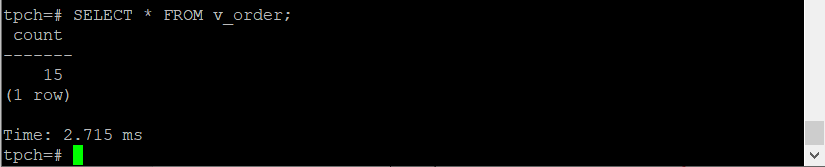
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



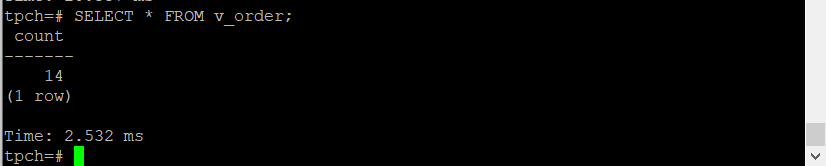
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



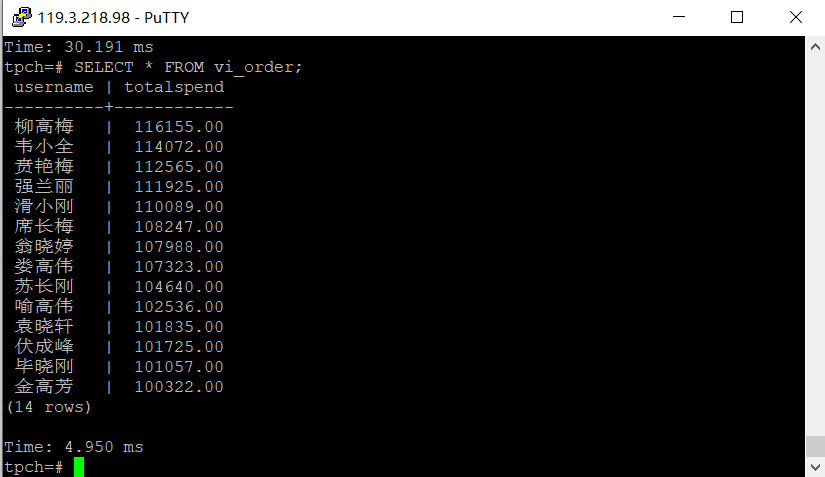
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



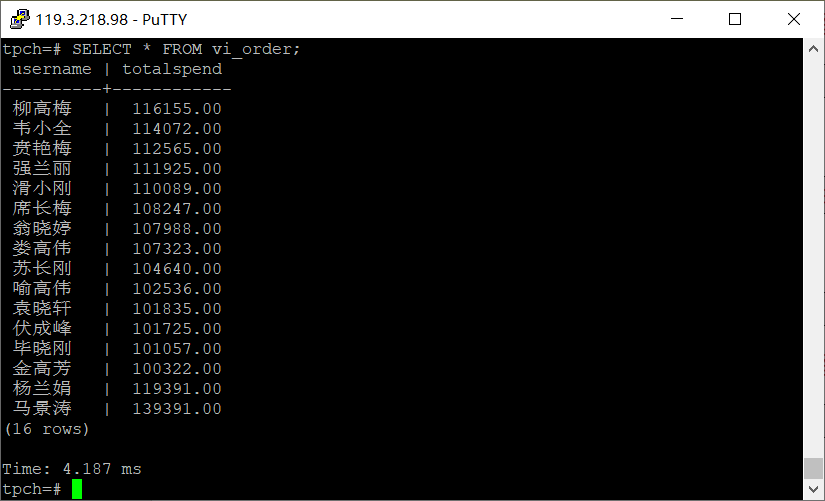
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

1-2.列比行快约11倍

3-4.列比行快约23倍

4-5.行比列快约3倍

7-8.行比列快约20倍

数据读取时，行存储通常将一行数据完全读出，而列存储每次读取的数据是集合的一段或者全部。

在1-4图中，存在年份date数据的判定和价钱numeric数据的相加或平均值，行存储需要读完所有数据，存在冗余列。而列存储只需要读取两列数据，不存在冗余性问题。并且列存储的每一列数据类型是同质的，不存在二义性问题。比如说价格数据类型为浮点数(numeric)，那么它的数据集合一定是浮点数数据。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，此时列存储的解析过程更有利于分析数据。

在5-8图中

行存储只需要读取每行的order\_id，相当于读一列完成；

列存储需要读取order\_id列，再读取order\_price列，在读取上花时就比行存储用时多，再加上列存储的修改和添加数据意味着磁头调度次数多，而磁头调度是需要时间的，一般在1ms~10ms，再加上磁头需要在盘片上移动和定位花费的时间，实际时间消耗会更大。

总而言之，如果关注整张表的内容，而不是单独某几列，并且所关注的内容是不需要通过任何聚集运算的，那么推荐使用行式存储，效率更高。反之，关注的都是某几列的内容，或者有频繁聚集需要的，通过聚集之后进行数据分析的表那么用列存储效率更高。

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。

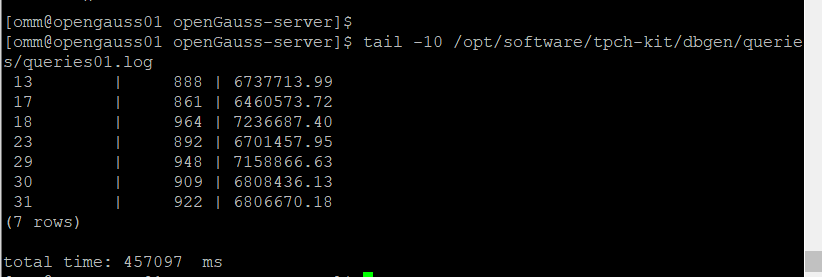
增量物化视图顾名思义就是可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

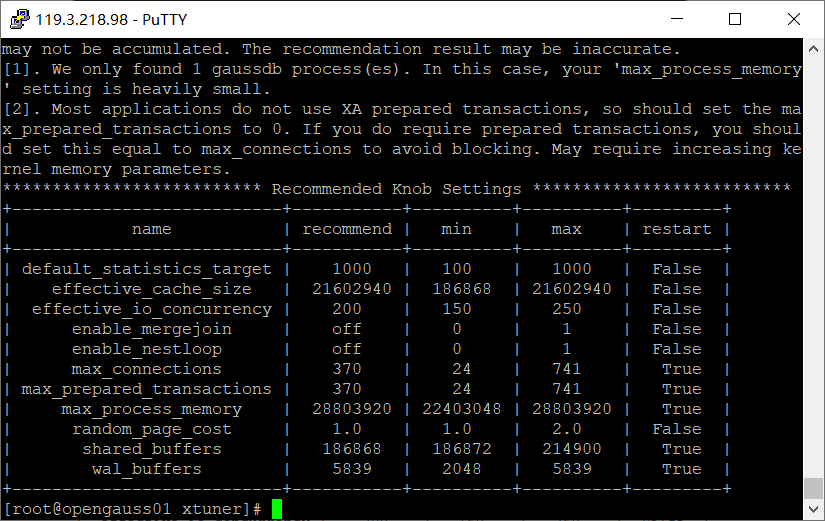
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

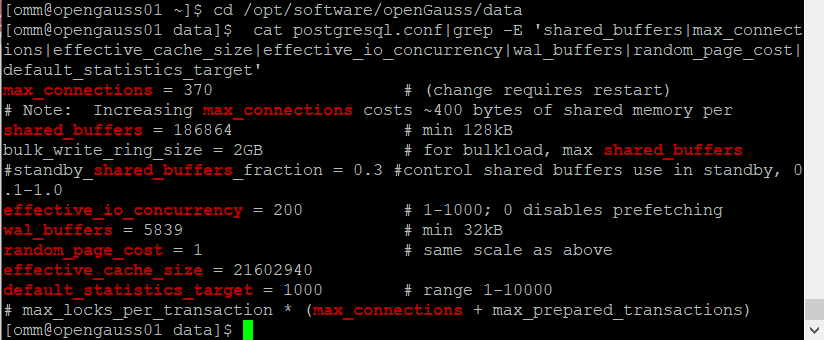
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

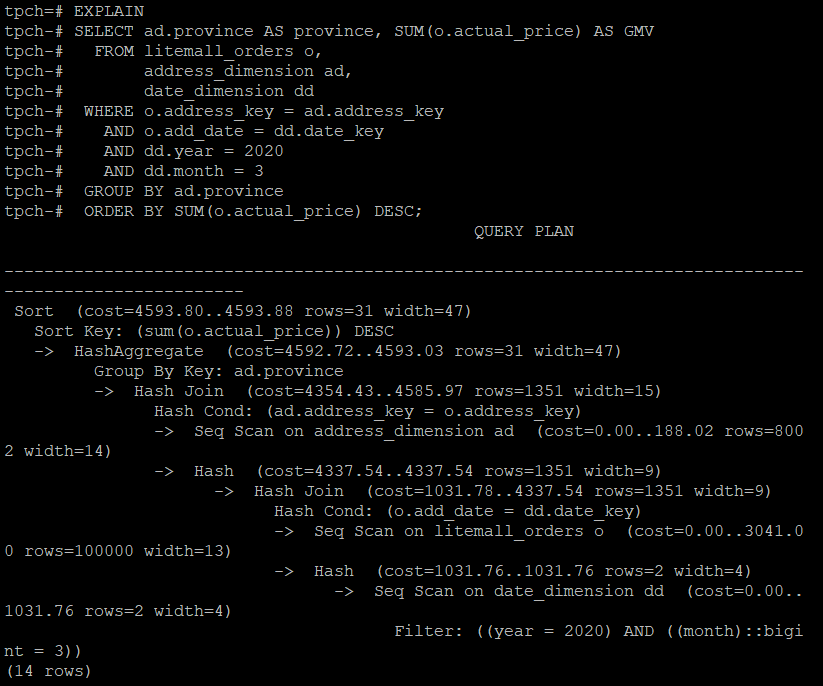
AND o.add\_date = dd.date\_key

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

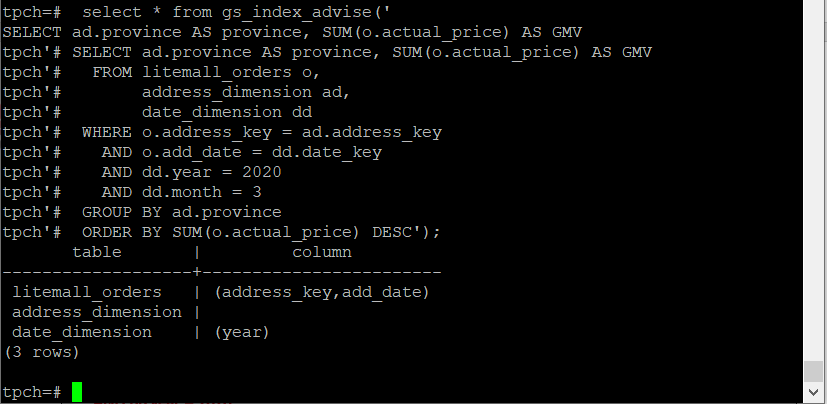
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

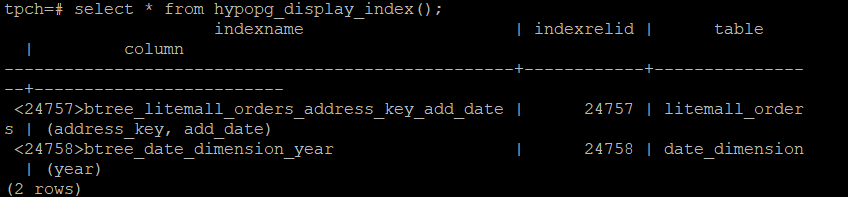
GROUP BY ad.province

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。

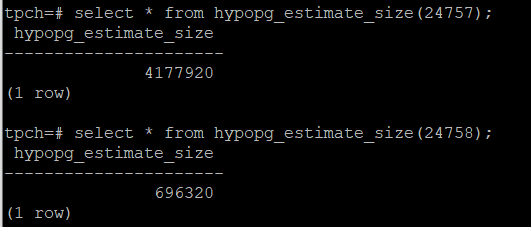
select \* from hypopg\_display\_index();



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(24757);

select \* from hypopg\_estimate\_size(24758);



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

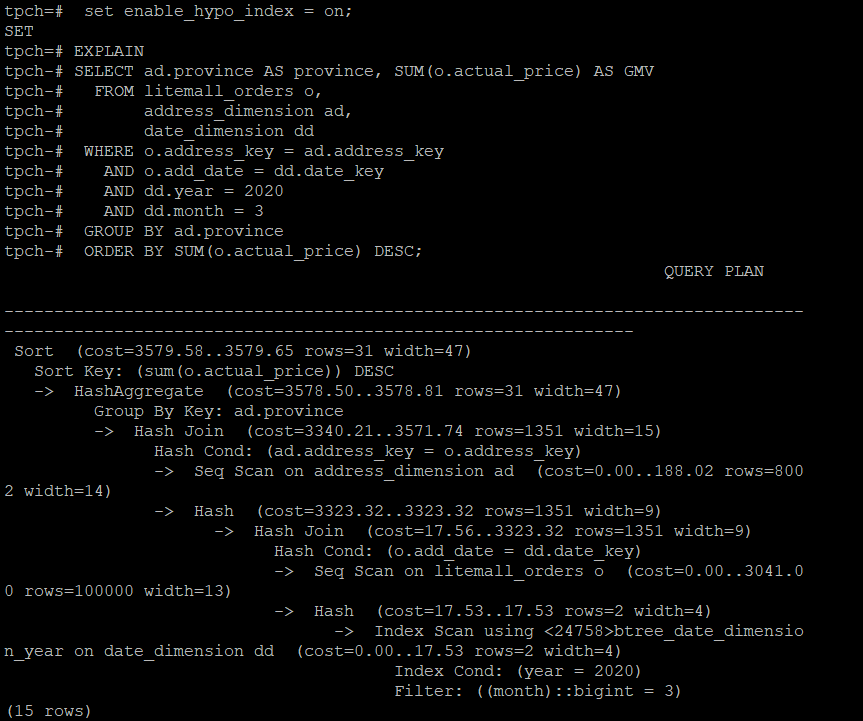
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

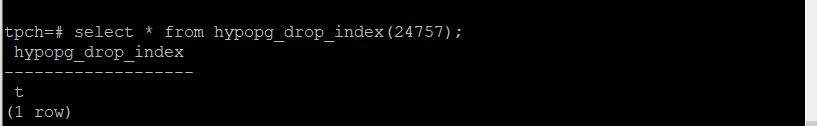
GROUP BY ad.province

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



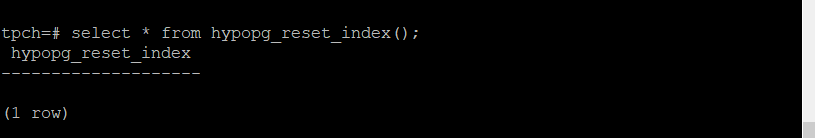
6. 删除某一个索引虚拟列，将执行结果截图。

select \* from hypopg\_drop\_index(24757);



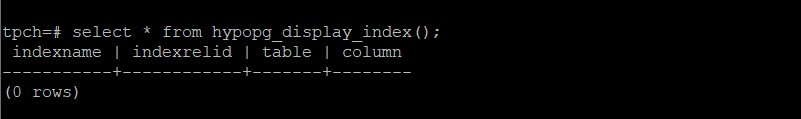
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

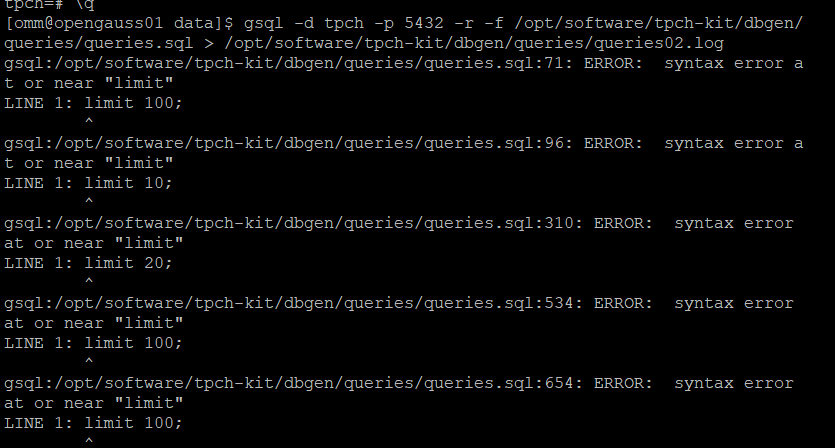
select \* from hypopg\_display\_index();

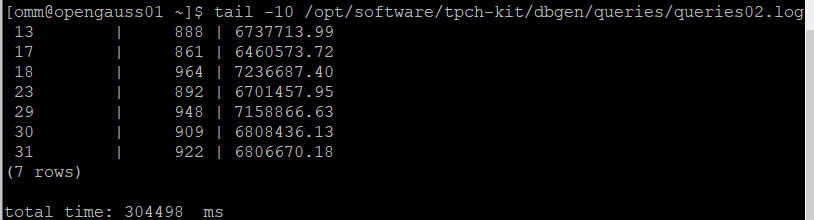


任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

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？

2个参数，max\_connections，shared\_buffers

max\_connections

max\_connections 决定了客户端的最大并发连接数，默认值通常为 100。如果出现连接数过多，无法连接数据库的错误时，可能需要考虑增加最大连接数。不过，修改该参数还需要考虑对其他参数的影响（尤其是 work\_mem）；因为它们是基于每个连接设置的值，增加连接数也会导致这些内存使用量的增加。

shared\_buffers

除了操作系统的 I/O 缓存之外，PostgreSQL 还会使用自己的内部缓存。PostgreSQL 共享内存缓冲区由参数 shared\_buffers 设置，它决定了 PostgreSQL 能够使用的专用缓存大小。

为确保在所有机器和操作系统上的兼容性，PostgreSQL 默认将该值设置得很小，通常是 128 MB。因此，增加 shared\_buffers 的值是提高性能最有效的设置之一。

修改 shared\_buffers 之后必须重启服务器才能生效：

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。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。

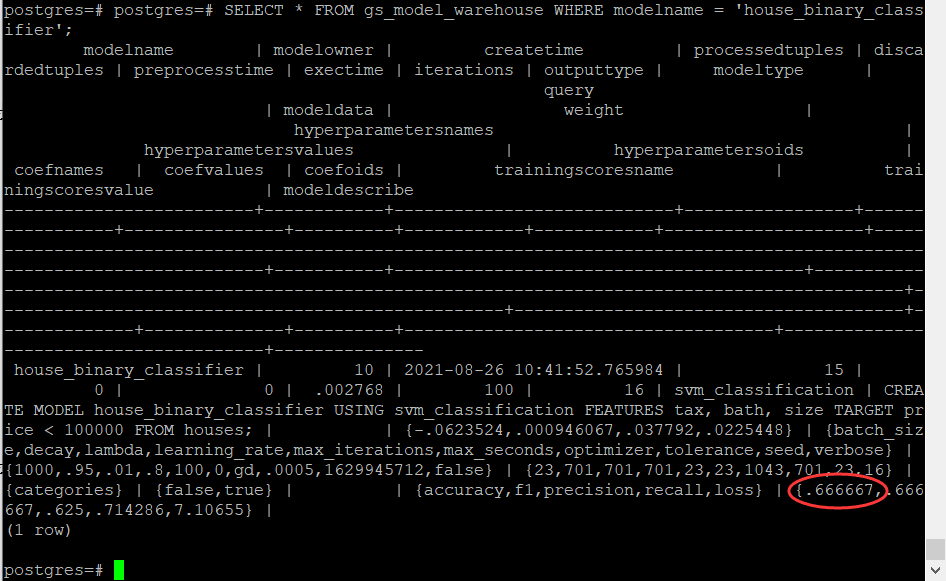
其余优化方法：

1. 设计表时选好字段类型，对数据量较大的表，为减少与其他表的关联，增加一些冗余字段提高查询性能。
2. 服务器不能太差，磁盘的读写性能影响io
3. 对访问频繁的数据，充分利用数据库cache和应用的缓存
4. 数据库实体的碎片的整理

# 关卡四、openGauss的DB4AI特性应用

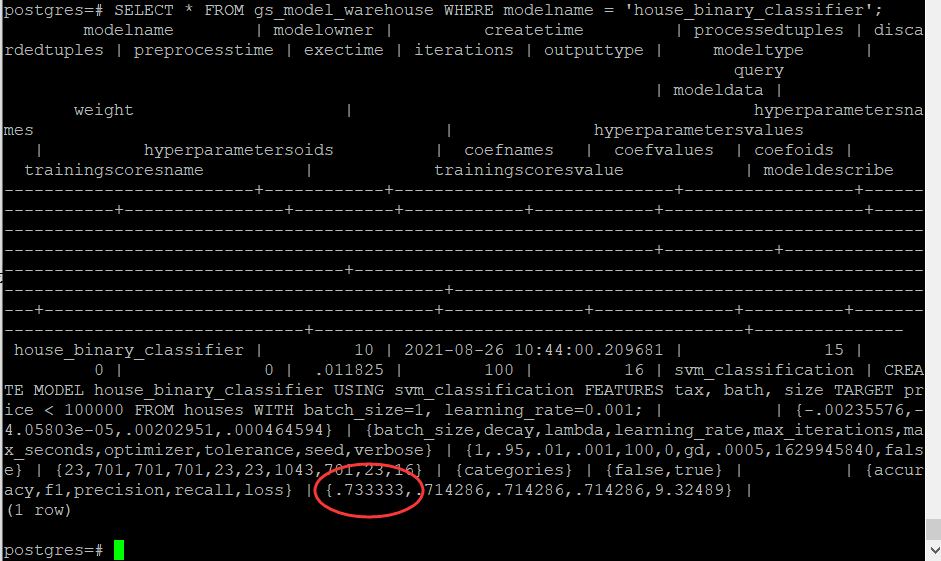
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



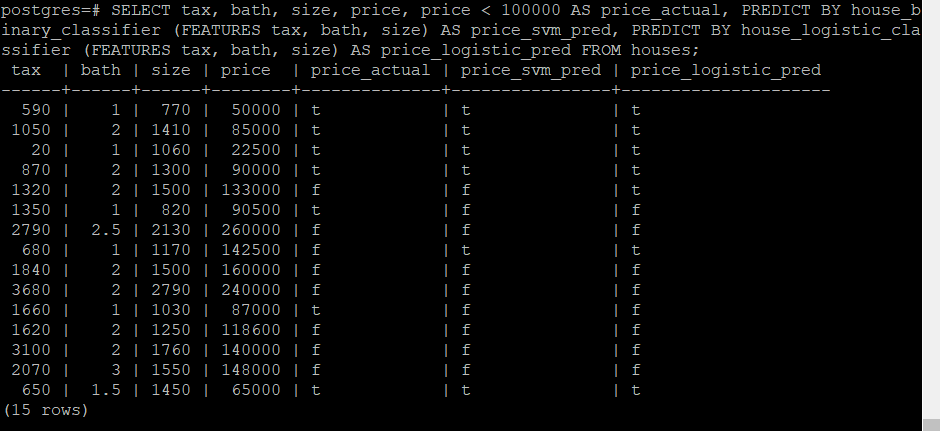
任务二：观察新模型的信息，将执行结果截图。

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的区别主要在于输出变量类型是否连续，预测连续值的为回归问题、预测离散值的为分类问题。

实践思考题2：什么是SVM算法？

支持向量机(support vector machine)是一种分类算法，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。通俗来讲，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，即支持向量机的学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。

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：Accuracy-所有正确预测文档数与总数的比率TP+TN/TP+FP+FN+TN

精确率：Precision-检索出相关文档数与检索出的文档总数的比率TP/TP+FP

召回率：Recall-所有准确的条目的比例TP/TP+FN

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：MSE-真实值与预测值的差值的平方然后求和平均

均方根误差：RMSE-衡量观测值与真实值之间的偏差

平均绝对误差：MAE-绝对误差的平均值