openGauss AI特性创新实践课



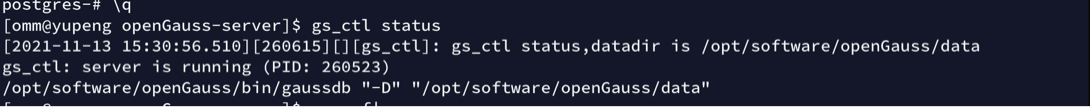
华为技术有限公司

# 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：

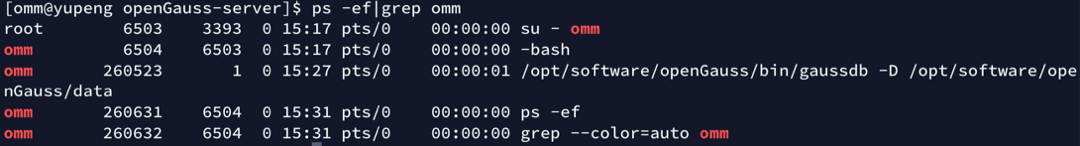
任务一：数据库状态验证

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？

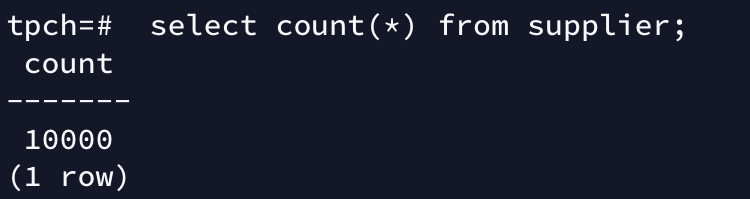
在学习方面，通过使用源码，可以增强我们对Linux语言的认识与掌握，提升自己的能力。在数据库方面，通过源码编译安装数据库，在编译安装的过程中，灵活性很大，可以设置和调整参数，可以按照需求实现调整。

# 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：

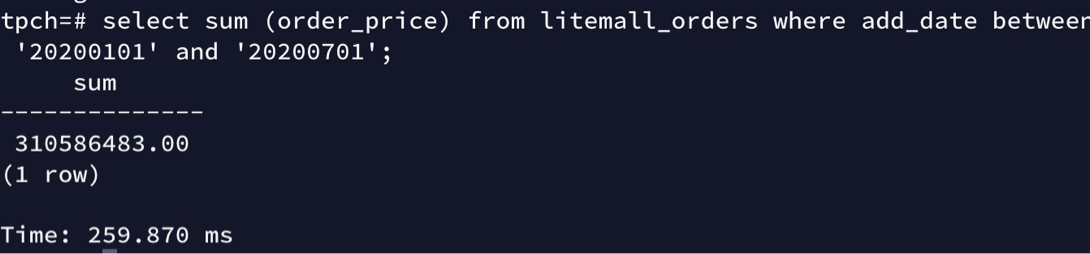
select count(\*) from supplier;;



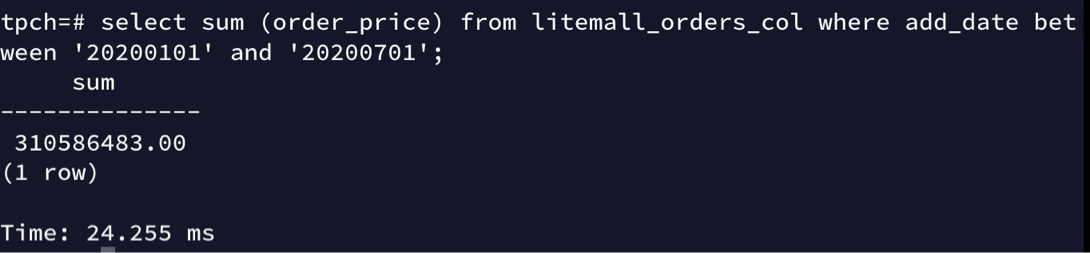
任务二：行存表与列存表执行效率对比

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';

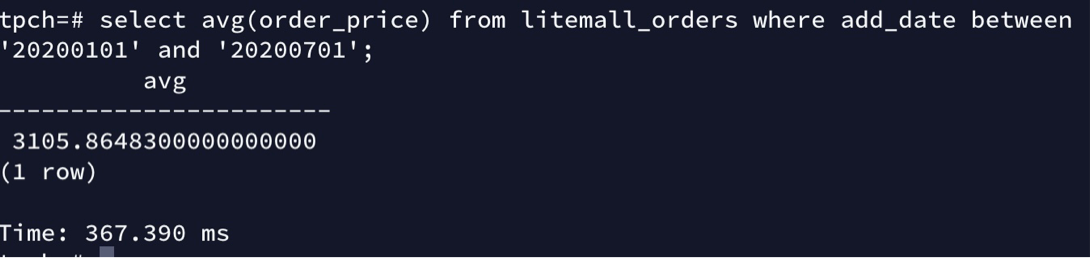


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

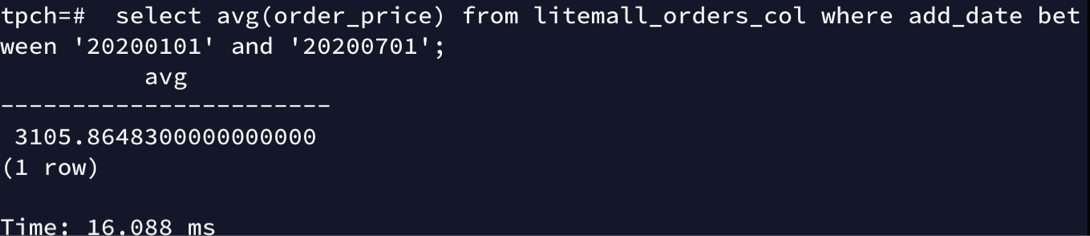


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

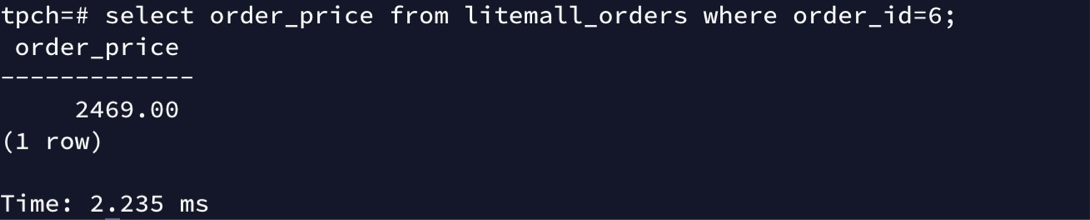


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

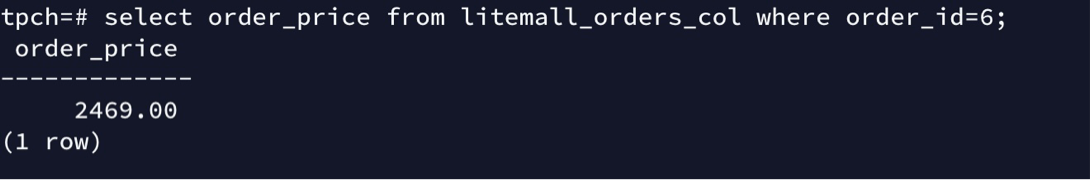


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

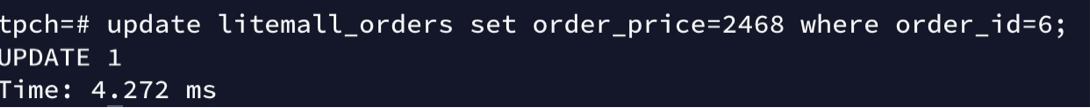


select order\_price from litemall\_orders\_col where order\_id=6;

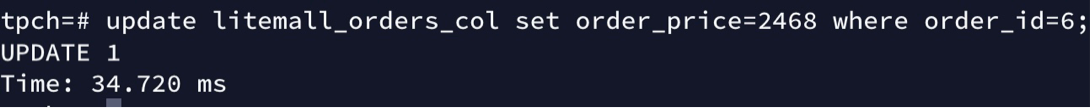


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

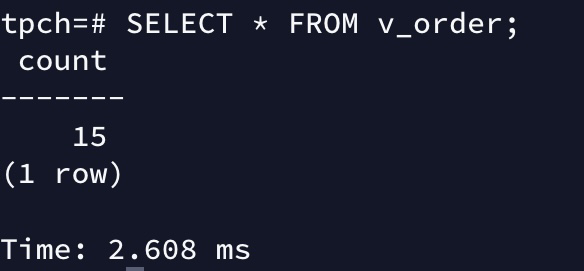
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



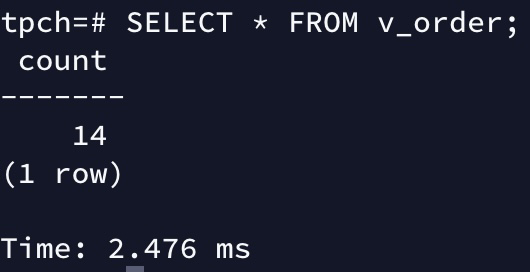
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



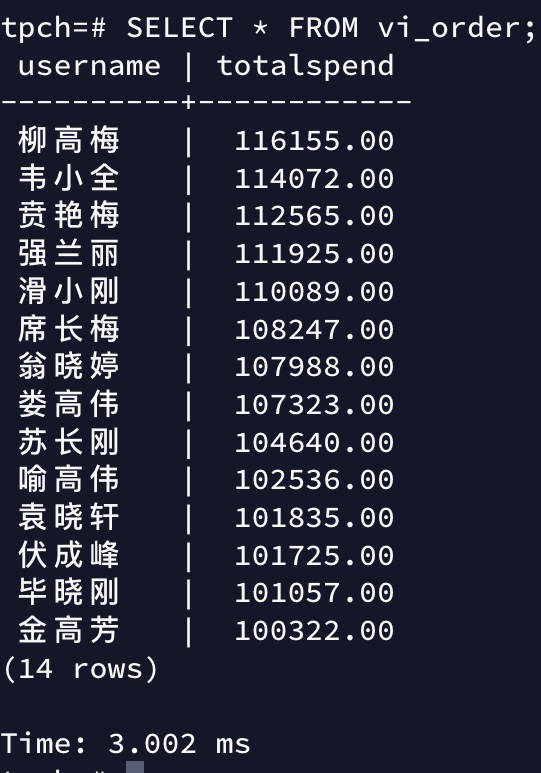
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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？

行储存与列储存：

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。

执行数据写入时：

列存储由于需要把一行记录拆分成单列保存，写入次数明显比行存储多列存储由于需要把一行记录拆分成单列保存，写入次数明显比行存储多。

数据修改时，行存储是在指定位置写入一次，列存储是将磁盘定位到多个列上分别写入，这个过程仍是行存储的列数倍。所以，数据修改也是以行存储占优。

执行数据读写时：

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。而且，列存储的解析过程更有利于分析大数据。

结论：

执行写与修改操作时，行存表效率高；执行读与分析操作时，列存表效率高。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

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。

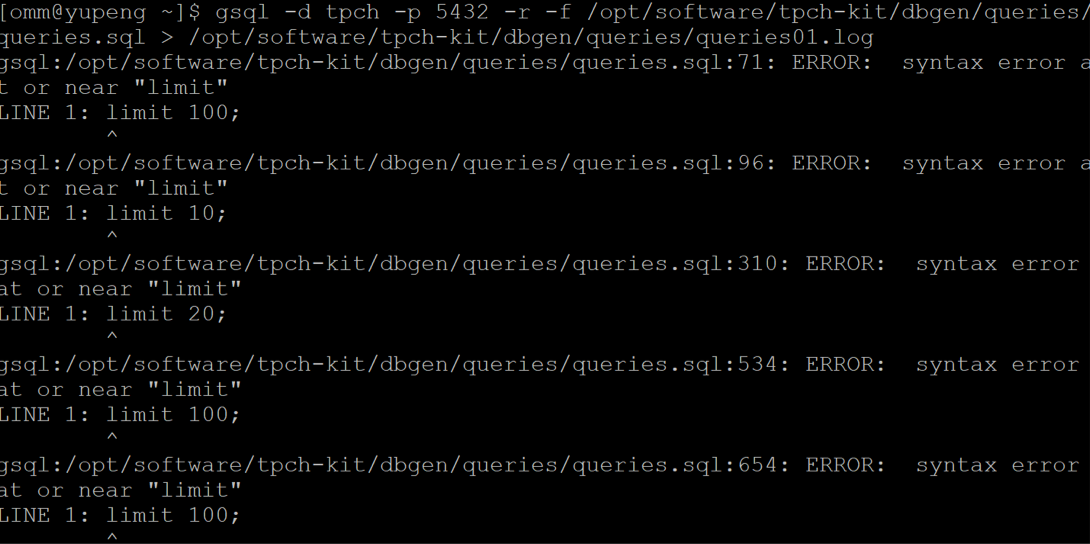
增量物化视图是可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。与全量创建物化视图不同在于目前增量物化视图所支持场景较小，目前物化视图创建语句仅支持基表扫描语句或者UNION ALL语句。

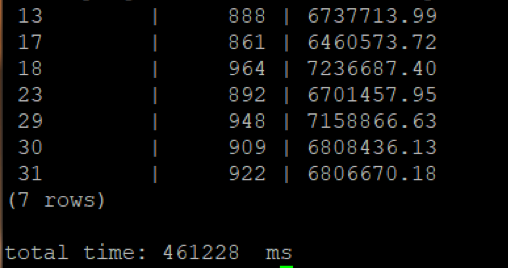
# 关卡三、openGauss的AI4DB特性应用

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：

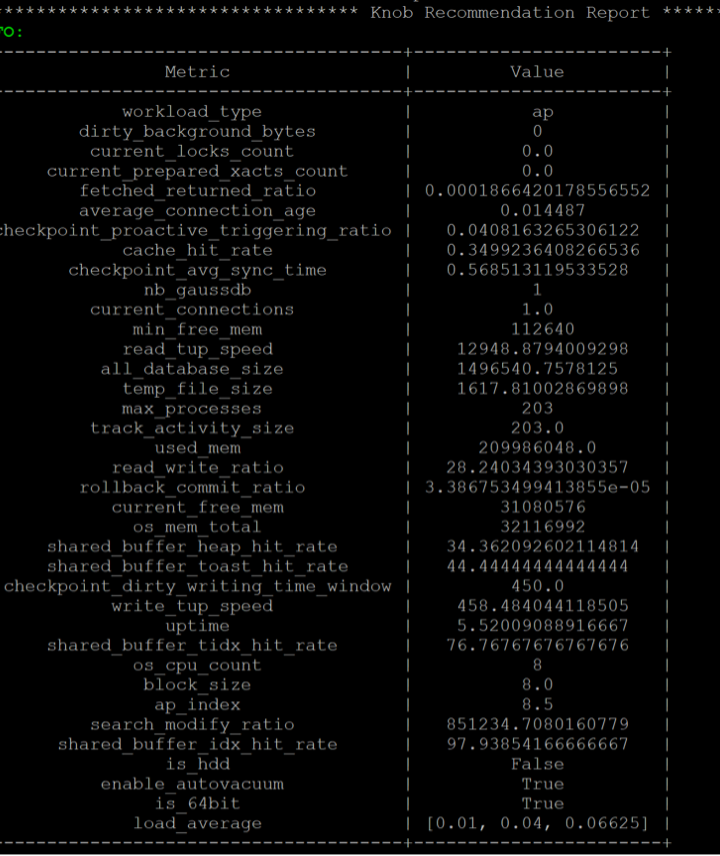
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log

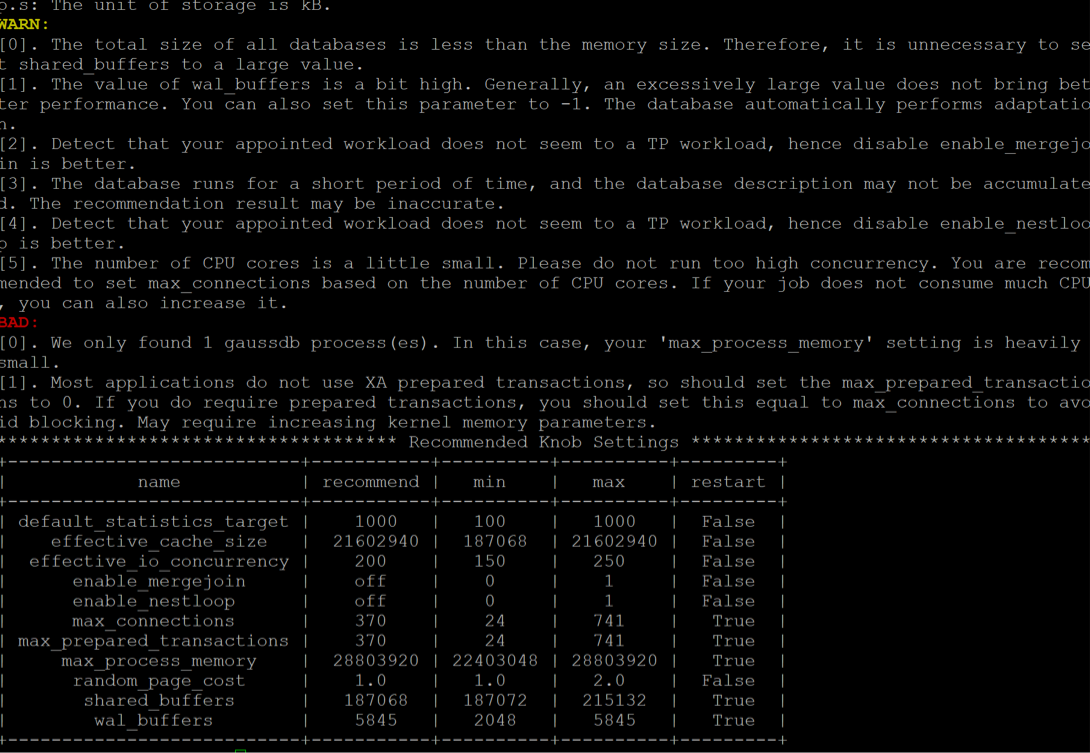




2. 使用root用户，执行X-Tuner进行参数建议优化，将执行结果截图

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

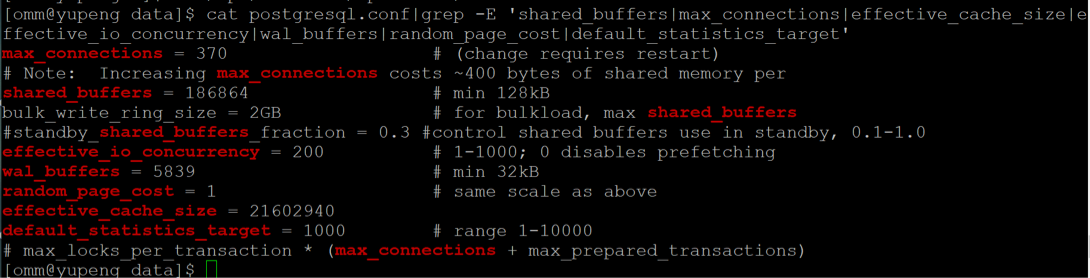




3.重启完成后，获取参数值：

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

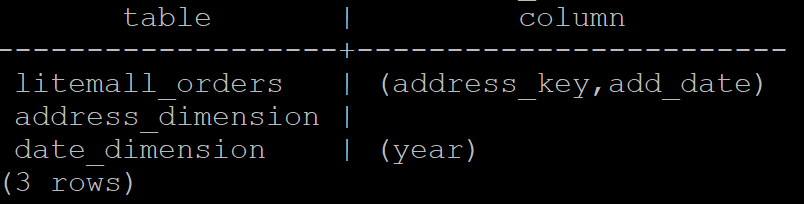
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

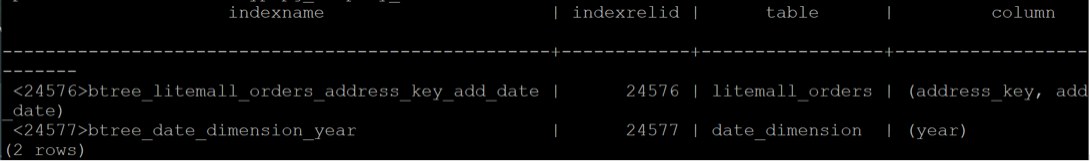
GROUP BY ad.province

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。

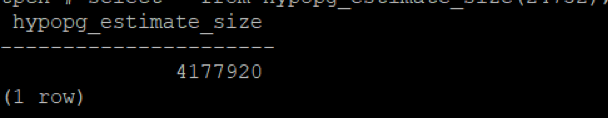
select \* from hypopg\_display\_index();

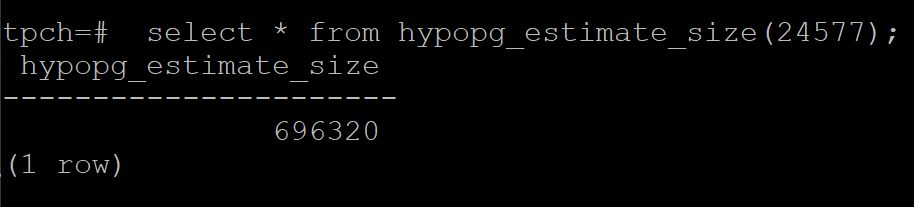


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

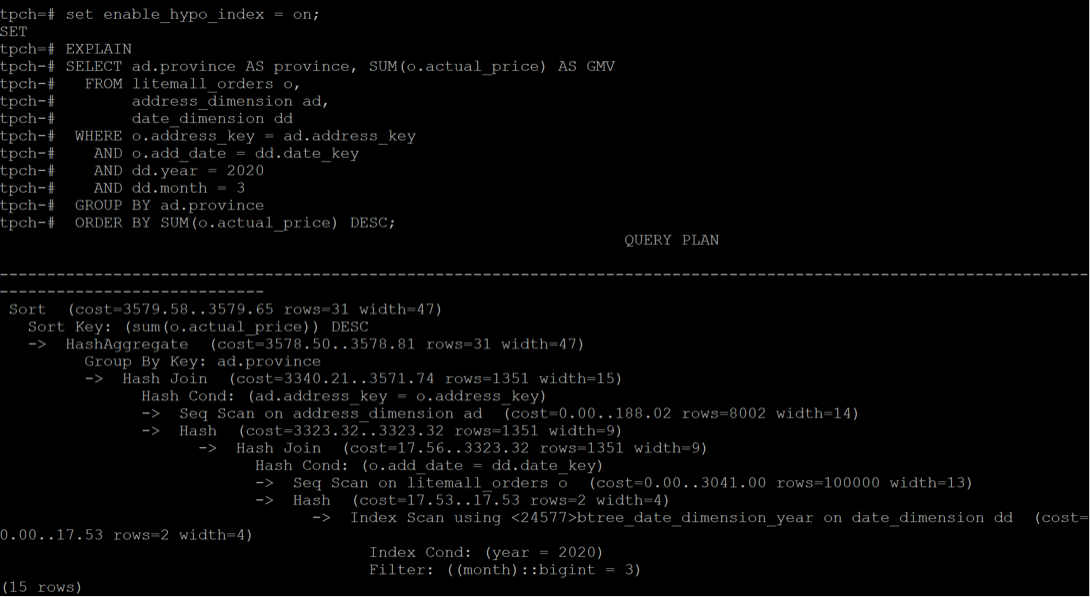
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

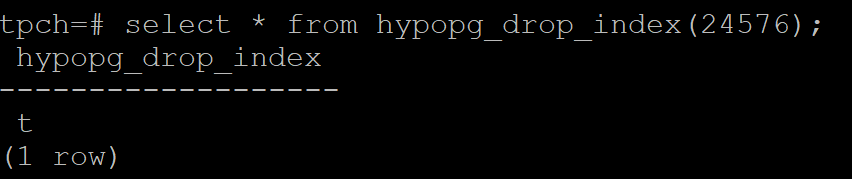
GROUP BY ad.province

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



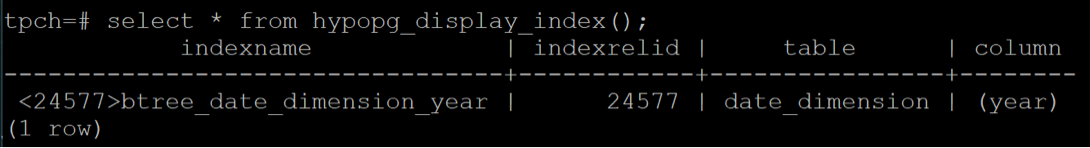
6. 删除某一个索引虚拟列，将执行结果截图。

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



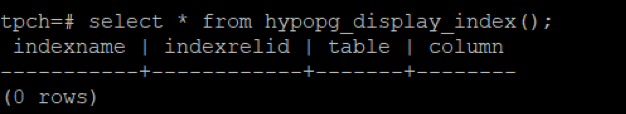
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

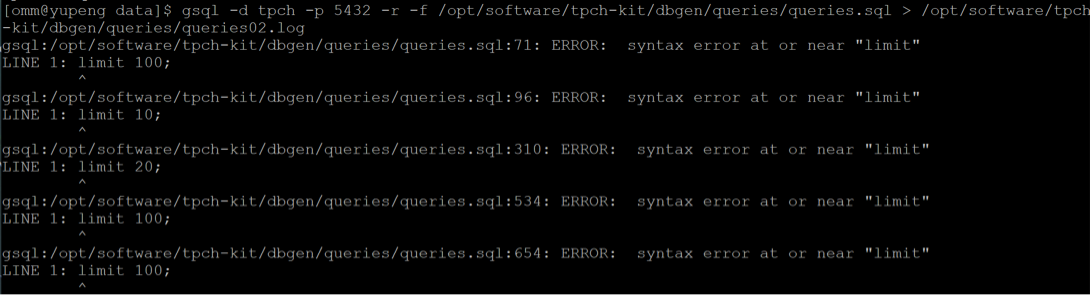
select \* from hypopg\_display\_index();

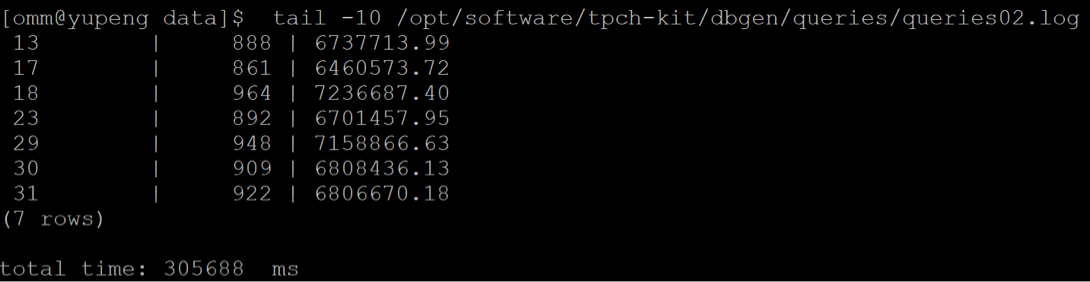


任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

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

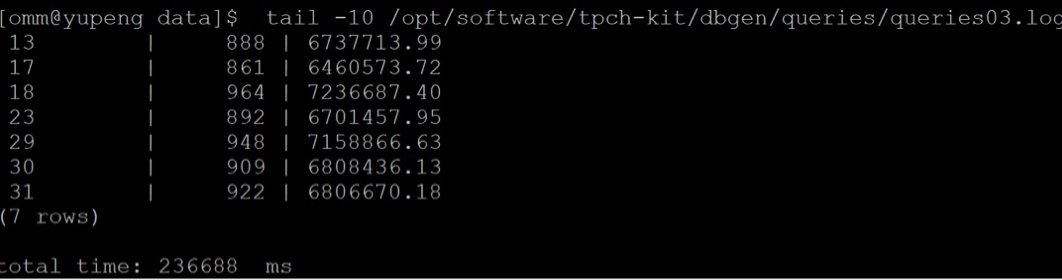




挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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？

对参数max\_connections实现了优化：

最大连接数通过该参数设置，如果服务器资源足够，我们应该尽量大的设置该参数，以此来满足更多客户端的链接要求。

对参数buffer实现了优化：

如果buffer过小，可能会使得nginx使用临时文件存储response，这样将导致磁盘读写IO，因此其大小是优化中重要的参数。

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？

索引的好处：

创建索引可以大大提高系统的性能。  
第一，   通过创建唯一性索引，可以保证[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)表中每一行数据的唯一性。  
第二，   可以大大加快数据的检索速度，这也是创建索引的最主要的原因。  
第三，   可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。  
第四，   在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。  
第五，   通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

优化的方法：

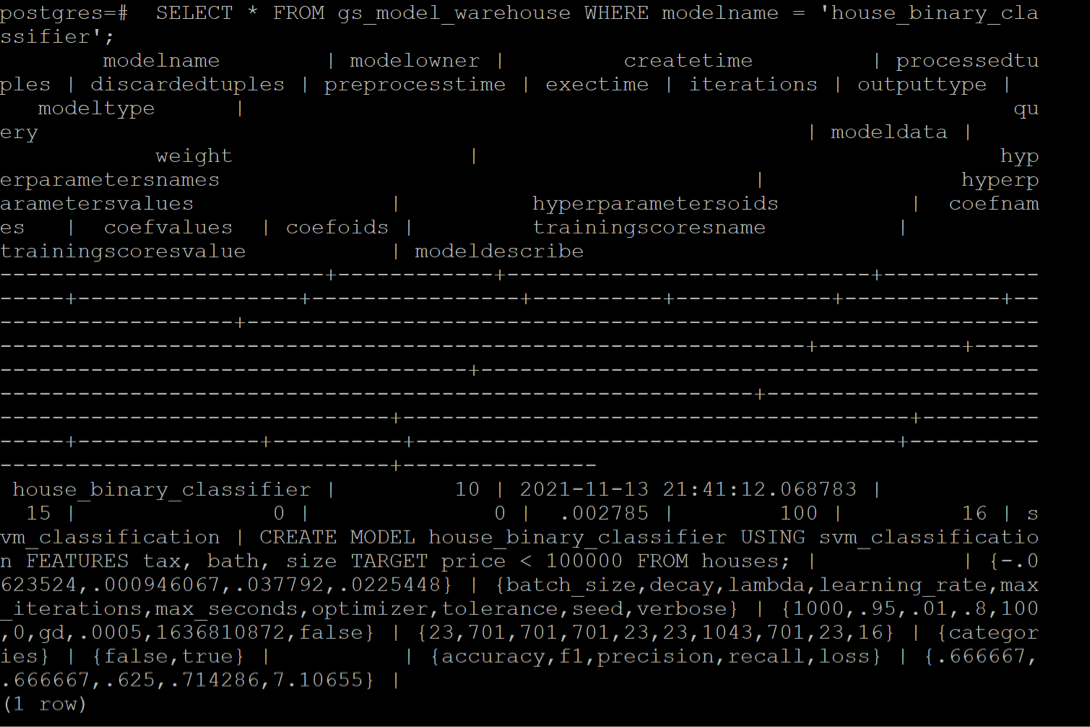
1、减少应用和数据库的交互次数、同一个sql语句的执行次数。  
2、数据库实体的碎片的整理（特别是对某些表经常进行insert和delete动作，尤其注意，索引字段为系列字段、自增长字段、时间字段，对于业务比较频繁的系统，最好一个月重建一次）。

3、减少表之间的关联，特别对于批量数据处理，尽量单表查询数据，统一在内存中进行逻辑处理，减少数据库压力（java处理批量数据不可取，尽量用c或者c++ 进行处理，效率大大提升）。   
4、对访问频繁的数据，充分利用数据库cache和应用的缓存。  
5、数据量比较大的，在设计过程中，为了减少其他表的关联，增加一些冗余字段，提高查询性能。

# 关卡四、openGauss的DB4AI特性应用

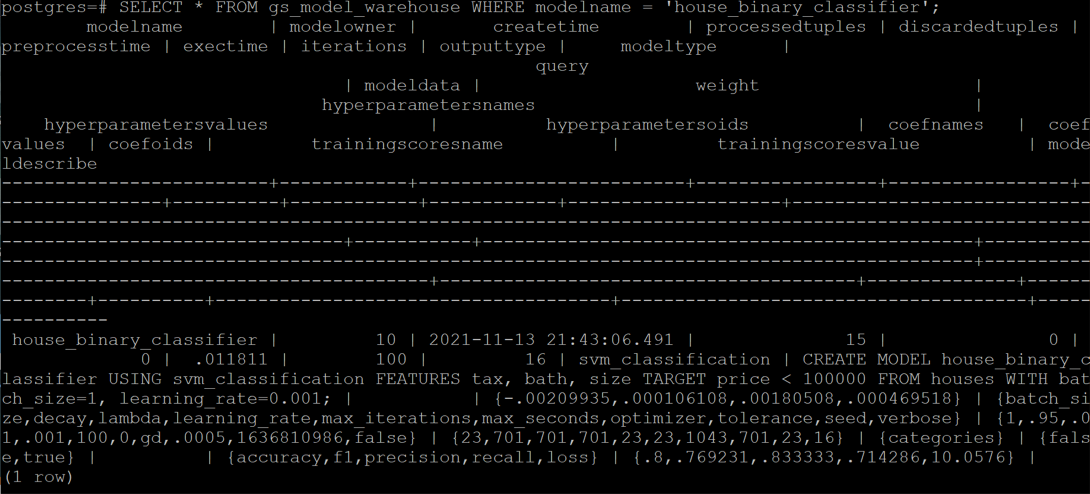
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



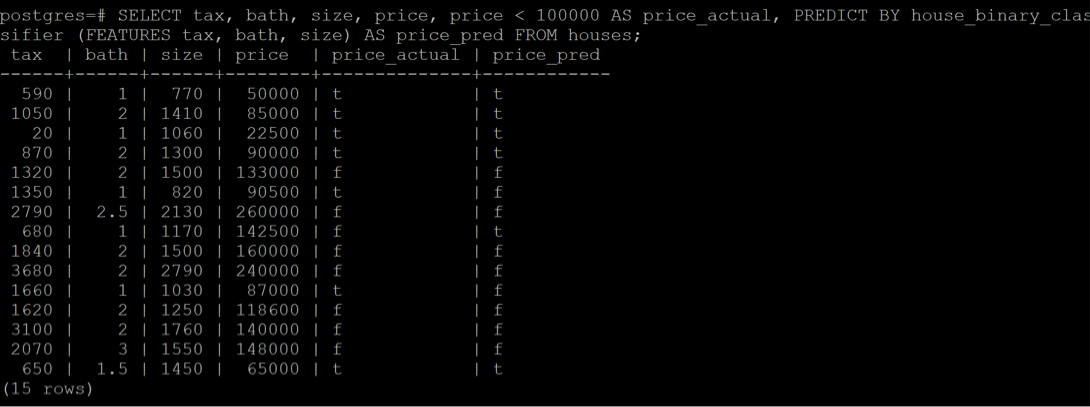
任务二：观察新模型的信息，将执行结果截图。

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？

通常来说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？

支持向量机(Support Vector Machine)是Cortes和Vapnik于1995年首先提出的，它在解决小样本、非线性及高维模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中。  
支持向量机方法是建立在统计学习理论的VC 维理论和结构风险最小原理基础上的，根据有限的样本信息在模型的复杂性（即对特定训练样本的学习精度，Accuracy）和学习能力（即无错误地识别任意样本的能力）之间寻求最佳折衷，以期获得最好的推广能力（或称泛化能力）。

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：

检索出相关文档数与检索出的文档总数的比率，衡量的是检索系统的查准率。

precision：

检索条目的准确率。

recall：

召回率是指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？

MSE：

均误差方（MSE）是所有样本的样本误差的平方的均值

MAE：

平均绝对误差（MAE）是所有样本的样本误差的绝对值的均值

MAPE：

平均绝对比例误差（MAPE）是所有样本的样本误差的绝对值占实际值的比例，mape越接近0，模型越准确。

R方：

R方是因变量的方差能被自变量解释的程度，R方越接近1，则代表自变量对因变量的解释度越高。