openGauss AI特性创新实践课



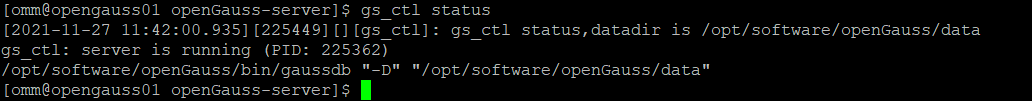
华为技术有限公司

# 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：

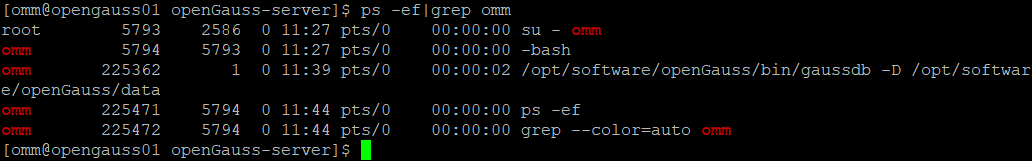
任务一：数据库状态验证

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？

答：

源码需要通过编译将高级语言转化为机器语言进而转为二进制代码使得计算机能够识别对应的操作命令，以最终实现对数据库的安装。同时，最主要通过源码编译安装数据库的原因便是通过源码可以定制安装，其在编译过程中可以进行参数的设定；并且安装的版本也可以自己选择，灵活性比较高。

安装数据库所需步骤：

主要分为编译数据库安装包前的服务器与依赖源码准备；和对数据库的源码进行对应的编译与安装到云服务器端上，并对其进行相关的测试操作：

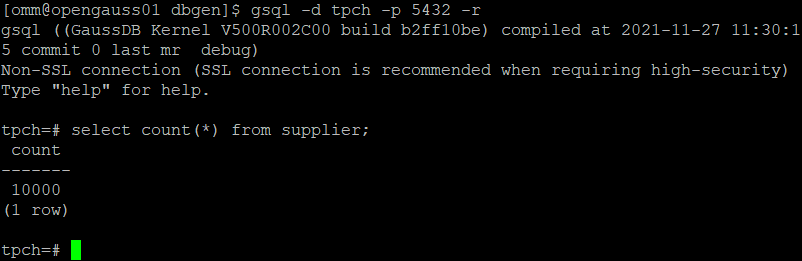
1. 编译前准备：包括登录ECS云服务器、创建用户omm与属组dbgrp、下载第三方编译库-解压、下载opengauss源码、传cmake包、用yum安装依赖包、替换python2->3、改目录的所有者为omm；
2. 数据库安装编译：切换用户为omm、设定环境变量-添加环境变量、在openGauss源码下生成配置文件并使用make与make install命令编译安装openGauss数据库、最终在初始化数据库后启动数据库。最终可以进行登录数据库并修改账号密码、查看数据库版本等操作。

# 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：

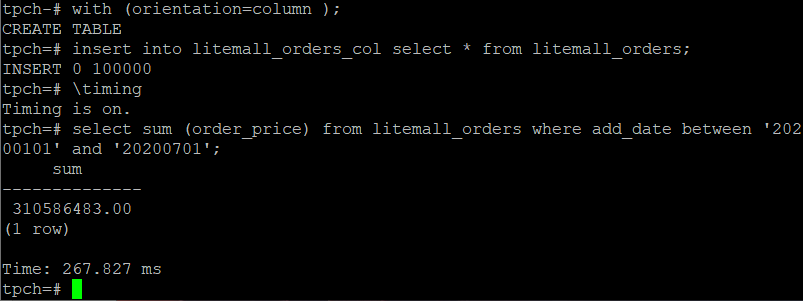
select count(\*) from supplier;



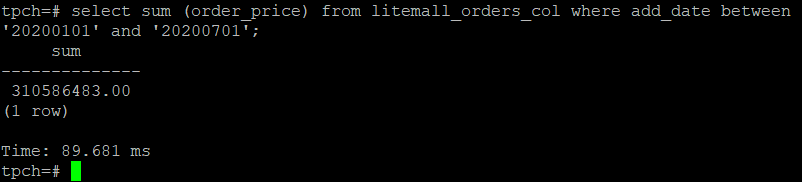
任务二：行存表与列存表执行效率对比

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';

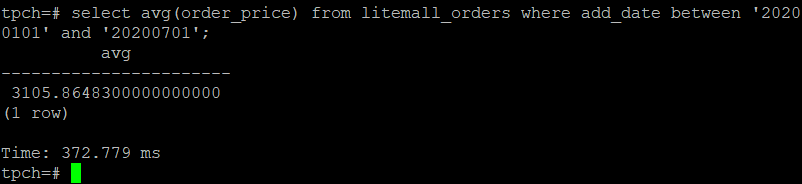


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

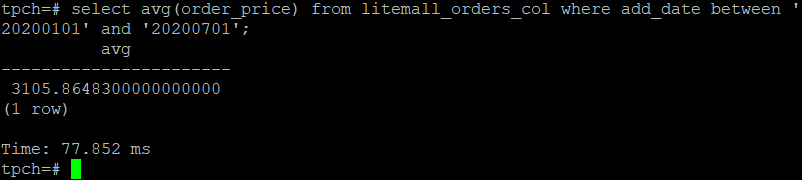


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

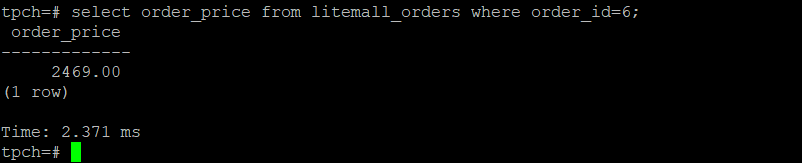


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

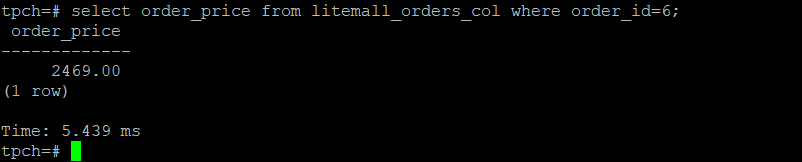


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

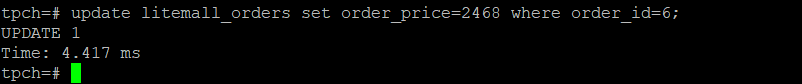


select order\_price from litemall\_orders\_col where order\_id=6;

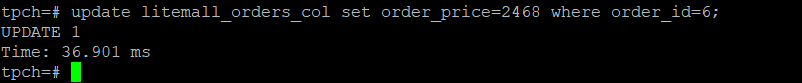


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



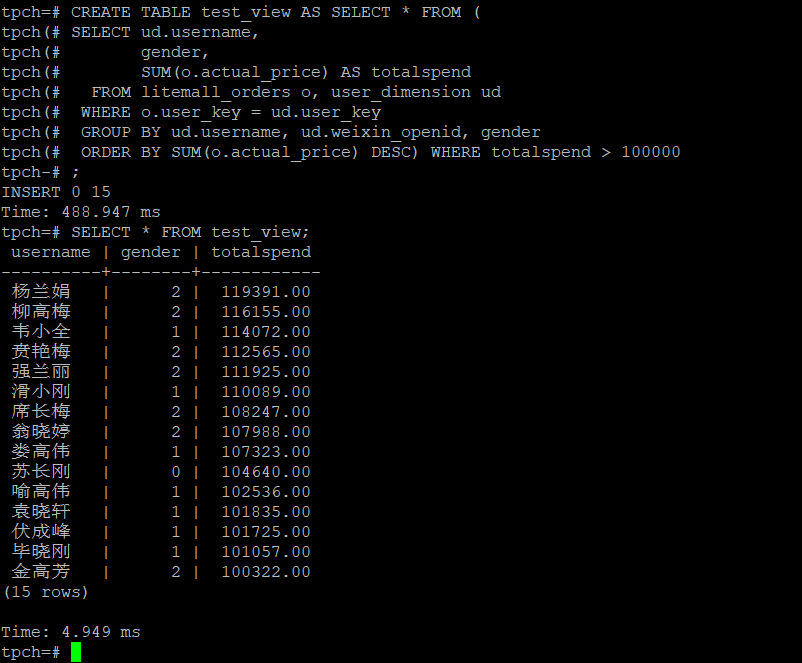
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

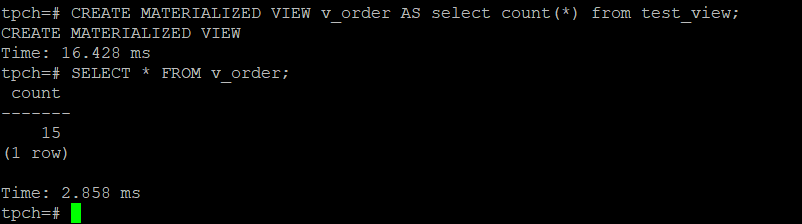
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



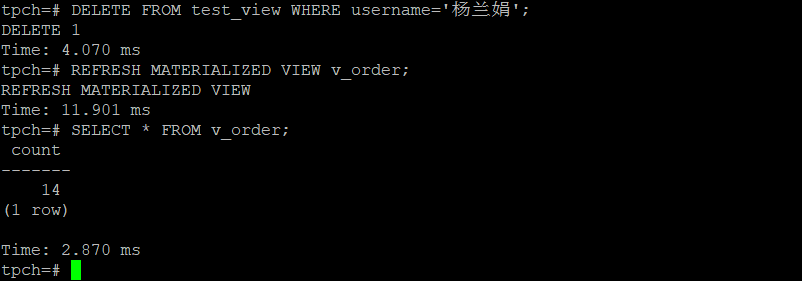
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



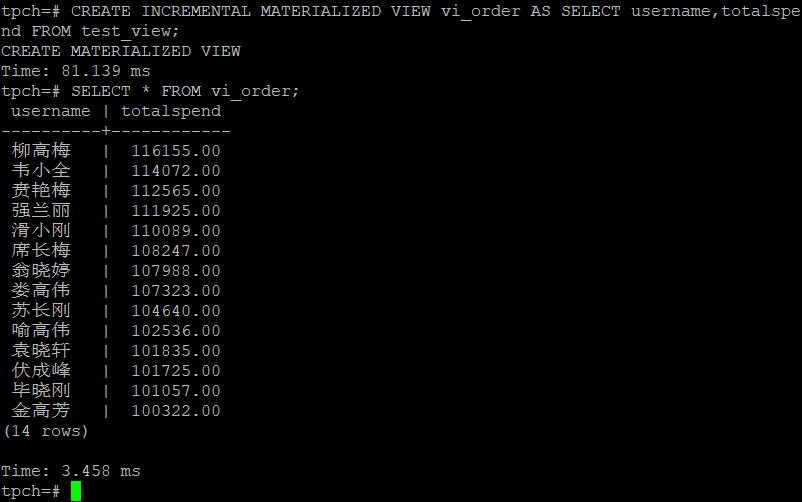
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



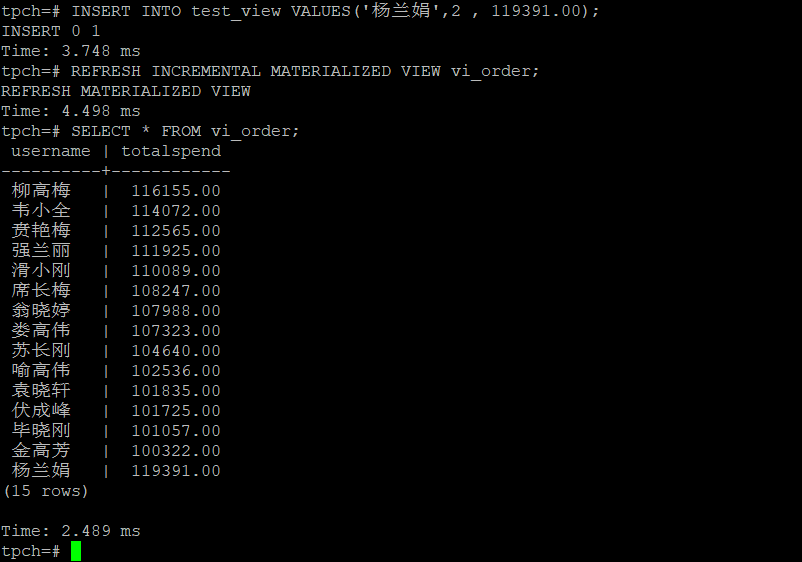
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

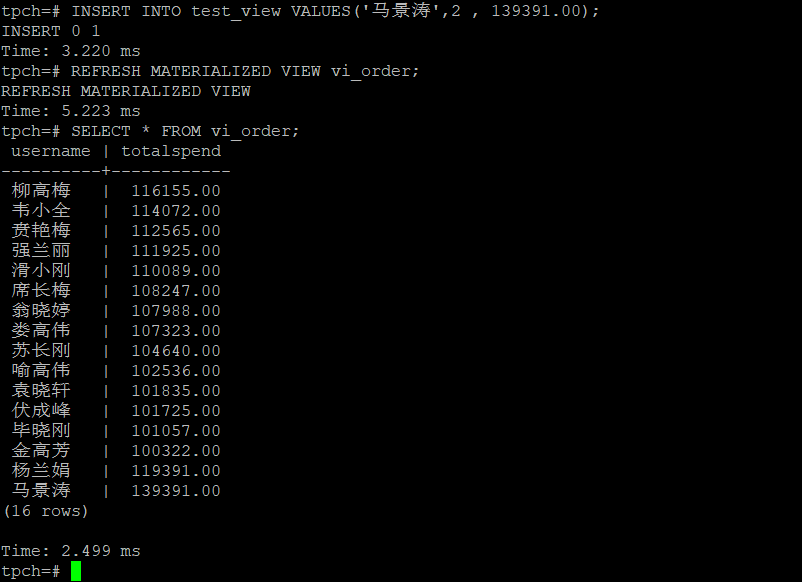
SELECT \* FROM vi\_order;



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？

答：

1、行存表在物理存储中的形式是以每行（元组）作为一组的存储引擎来作为组织逻辑进行存储；而列存表是在物理存储中以每列为一组的存储引擎作为组织逻辑进行存储。当我们要查询数据时，将要查询的相关数据在数据表的存储格式不同显然会使查询的次数不同，因此执行的时间也就自然不同了。

2、当需要对同一对象的不同属性进行查询时，显然行存表的效率更高。行存储更适合随机的增删改查操作、需要在行中选取所有属性的查询操作与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。更适合如商业领域与互联网领域。

3、当需要针对同一属性的不同对象进行查询时，显然列存表的效率更高。针对聚集函数而言列存储会比较快、对投影操作更高效。其任何列都能够作为索引，可以显著减少I/O消耗，并降低查询响应时间，非常适合数据仓库与分布式的应用，适合数据定长的大数据计算。更适合如互联网等领域的应用。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

答：

1、全量物化视图与增量物化视图均是需要在查询之前刷新数据，同步远程数据端与本地副本；

2、对大表来说，应优先考虑增量刷新，因为对大表进行全量刷新是很慢的，尤其是在有索引的情况下；

3、所以对于物化视图来说：全量物化与增大物化的主要差别在于刷新方式：全量物化与增量物化视图的查询时间均远小于直接对表进行查询，其作为对远程数据的本地副本，或用来生成基于数据表求和的汇总表，物化视图存储基于远程表的数据，也可以称为快照。均能够有效提高查询效率：

* 全量物化在刷新时是对全表数据进行刷新，一般时间比较均匀，无论对数据库进行何操作在进行刷新后查询的时间均是差不多的；
* 而增量物化在刷新时是对新增或修改的数据进行刷新，这样当像开始创建表这样的对大量表数据进行改变时，所花费的时间相较全量更新会较长；而之后在对某些数据项进行少量的更新或增加时，相对而言只对改变的数据进行刷新，本地与远端的交互消耗时间即从主表到本地表第一次复制的时间一般会比全量更新更小。增量物化视图支持增量表扫描与union all 语句。

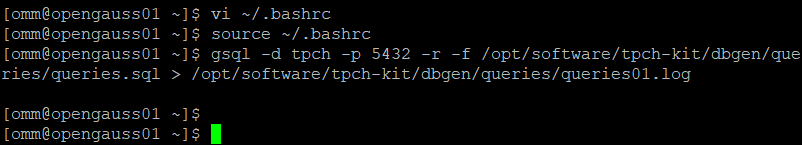
但平均而言，考虑到冗余与查询等因素，需要综合考虑表的大小与存储格式来判断选择那种物化视图。

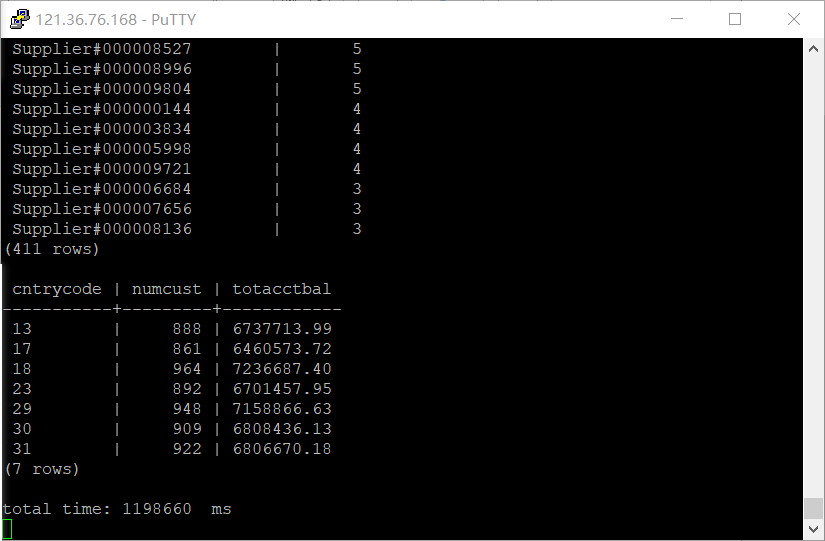
# 关卡三、openGauss的AI4DB特性应用

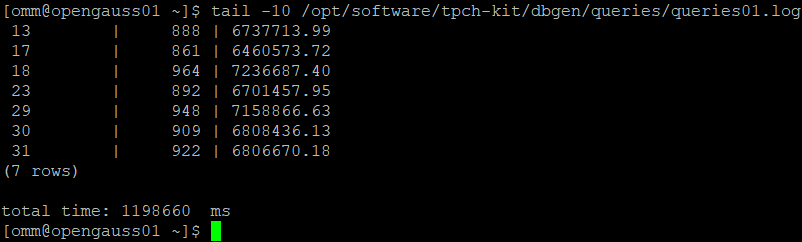
任务一：使用X-Tuner进行参数优化

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

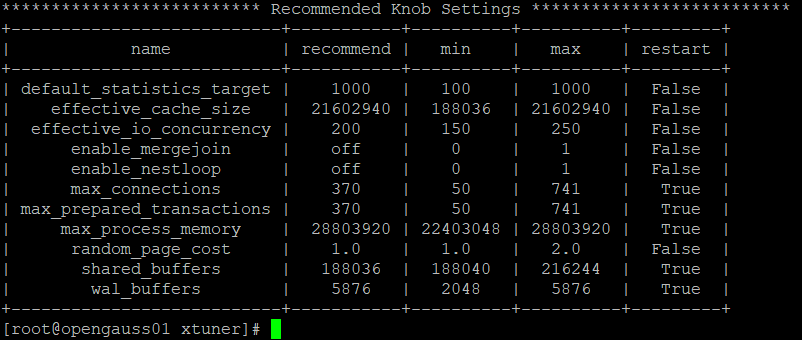






2. 使用root用户，执行X-Tuner进行参数建议优化，将执行结果截图

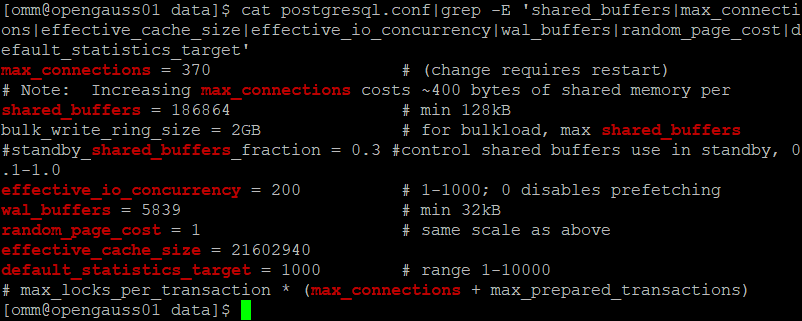
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



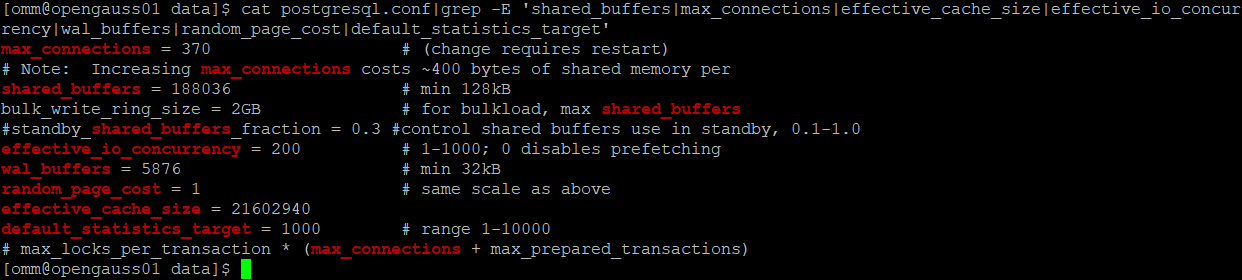
3.重启完成后，获取参数值：

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'



均修改为最优推荐值后：



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

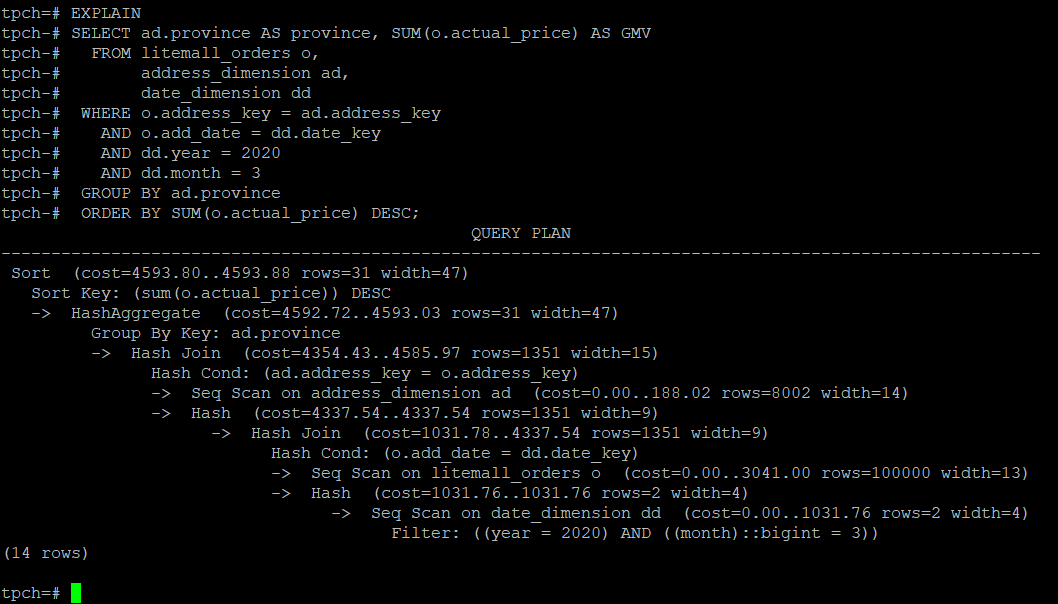
AND o.add\_date = dd.date\_key

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

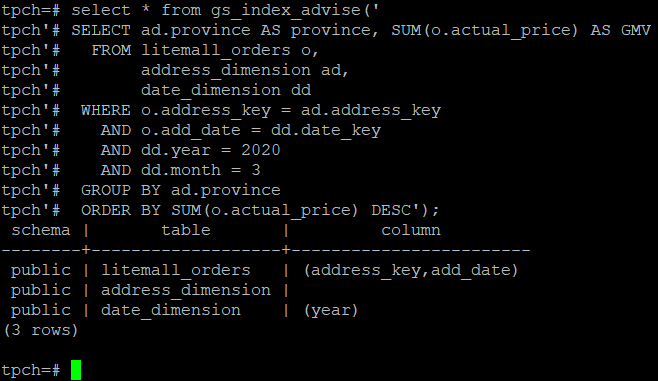
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

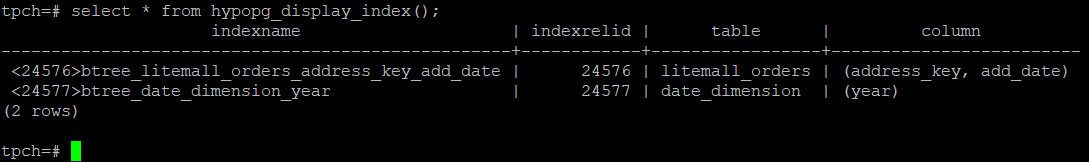
GROUP BY ad.province

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。

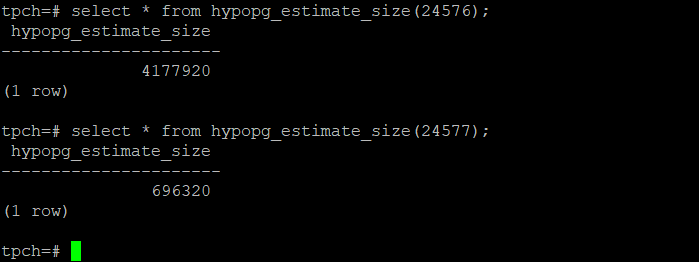
select \* from hypopg\_display\_index();



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

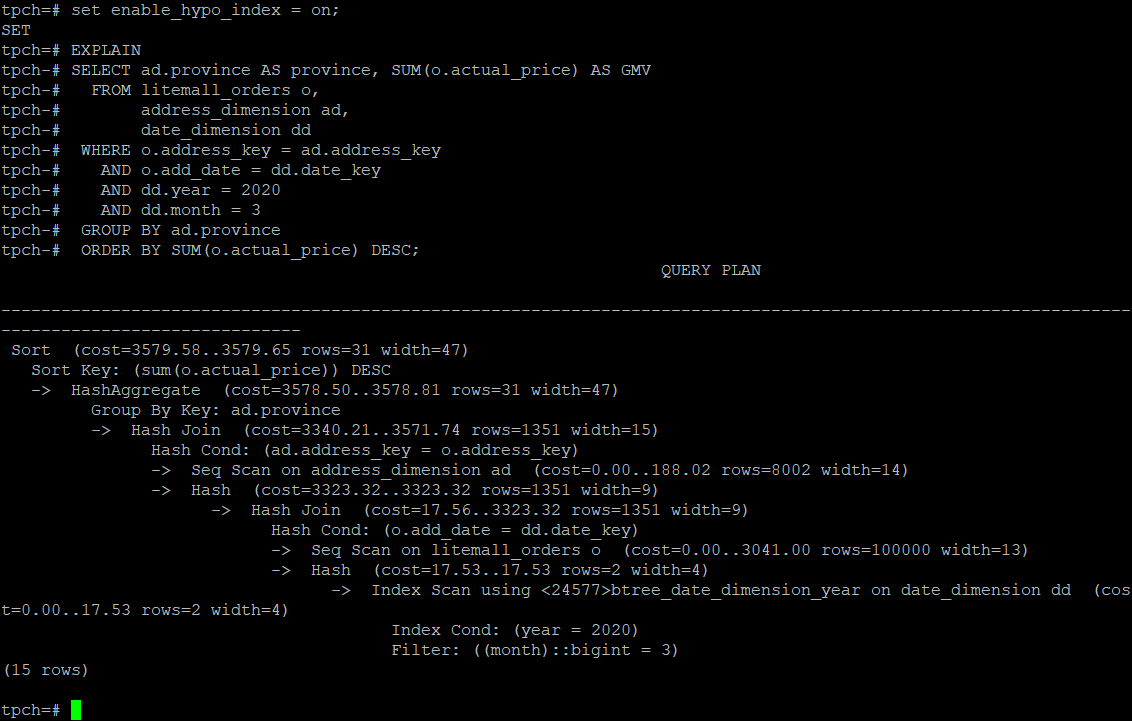
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

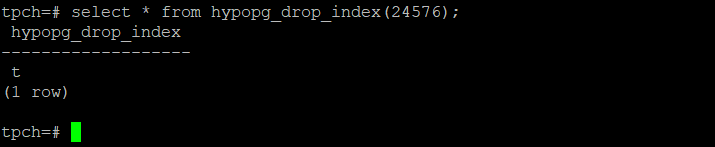
GROUP BY ad.province

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



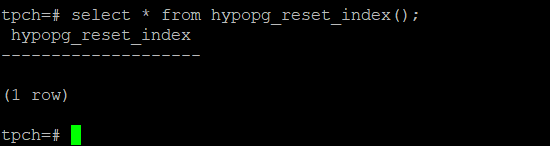
6. 删除某一个索引虚拟列，将执行结果截图。

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



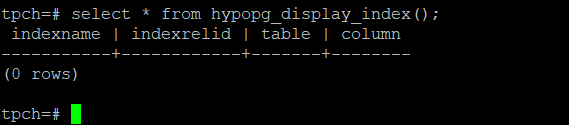
7. 删除某一个索引虚拟列，将执行结果截图。

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：

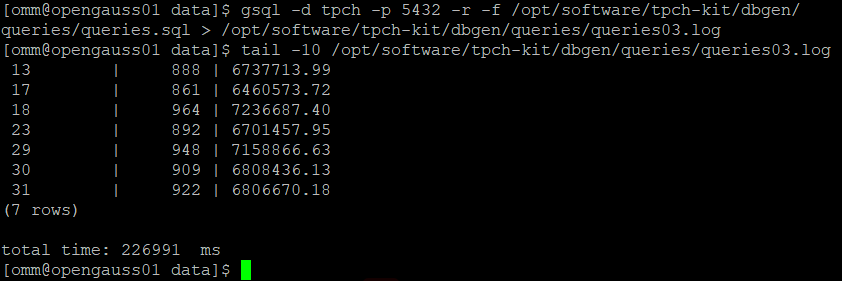
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

答：

1、根据X-Tuner给出的参数优化，主要是对于shared\_buffers、 max\_connections 、effective\_cache\_size、effective\_io\_concurrency、wal\_buffers、random\_page\_cost 、default\_statistics\_target等参数进行了优化。同时还可以对max\_prepared\_transactions、max\_process\_memory进行优化。

2、因为TPC-H是一个面向分析型业务（AP）的基准测试，其由一系列热点查询组成，他们都是高度复杂的，所以执行时间往往较长。而在实验中，当对数据库手动加载TPC-H数据时，其保存在tpch数据库中，默认的TPC-H数据库中表缺少索引，所以参数没有任何优化，执行的效率较差。所以基于openGauss的X-Tuner参数自调优功能，对于数据库的运行性能，在修改这些参数至推荐最优值后能够有效提升数据库的运行性能。

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？

答：

1、索引的使用对SQL来说，其好处包括：

* 能够有效减少查询的效率；
* 通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性；
* 可以大大加快数据的检索速度；
* 可以加快表之间的连接；
* 在使用分组和排序等子句进行数据检索时，可以显著减少在查询中分组和排序的时间；
* 通过使用索引，可以在查询的过程中使用优化隐藏器，以此提高系统的性能。

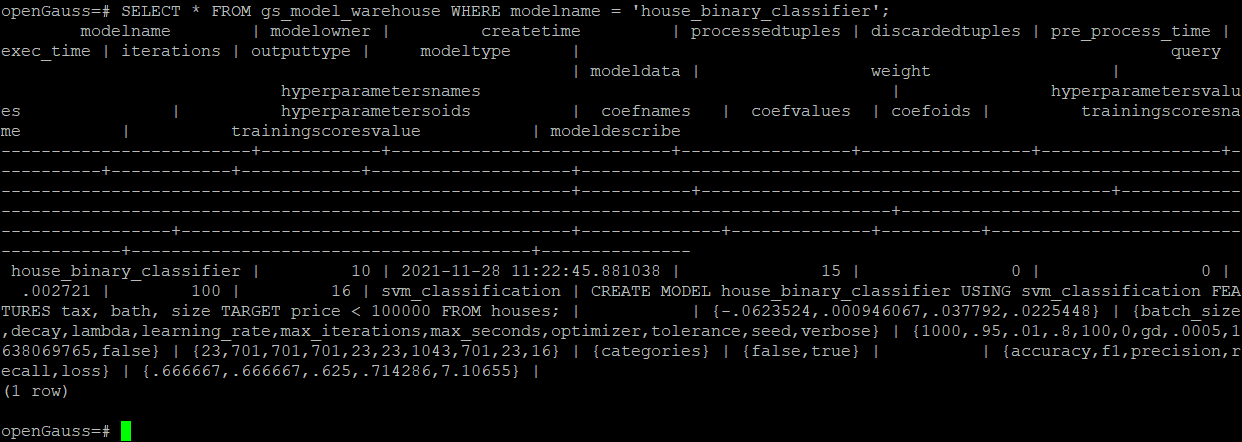
2、数据库优化方面：

* 可以优化SQL的查询语句，分析相关的慢查询等。对SQL的结构优化；
* 在设计数据表结构的时候严格根据数据的设计规范来设计数据库，方便更有效的查询；
* 同时使用缓存，把经常访问到而且不需要变化的数据放到缓存中；
* 在数据库的系统配置上优化，对文件达到一定数量时，因为无法打开就会频繁I/O，所以可以由此优化；
* 硬件优化，用更快的I/O、更多的内存去优化数据库；
* 优化锁的机制等。

# 关卡四、openGauss的DB4AI特性应用

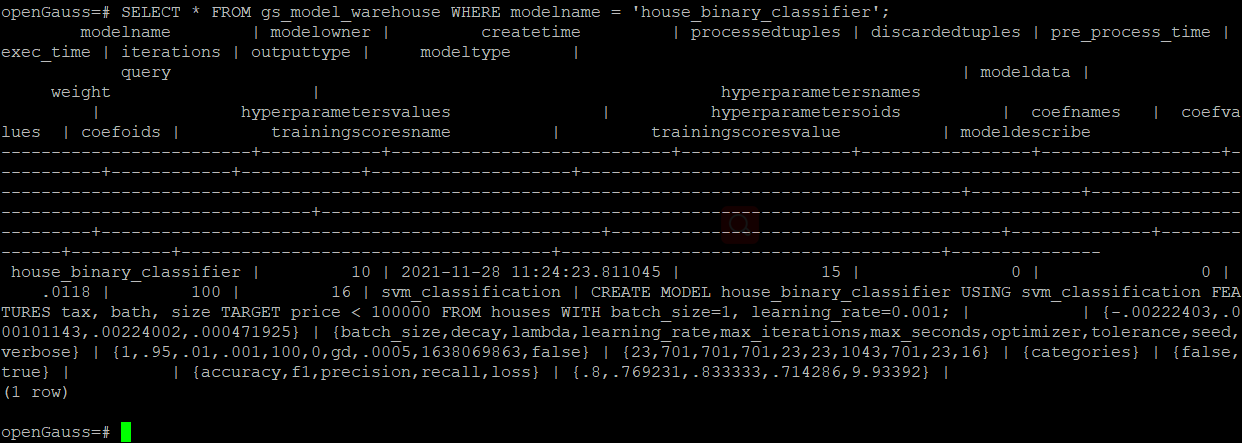
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



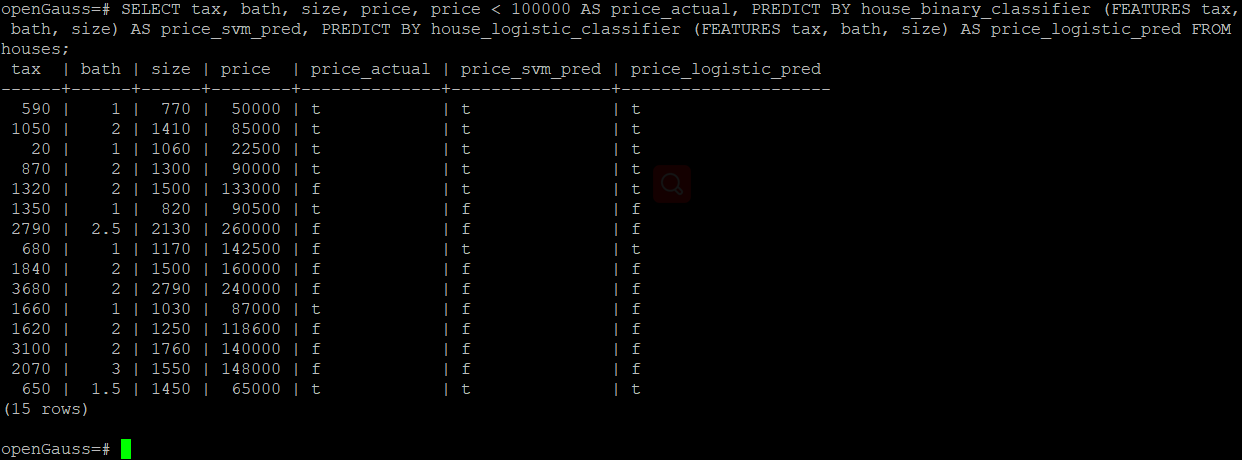
任务二：观察新模型的信息，将执行结果截图。

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？

答：

分类模型与回归模型的本质一致，其主要区别在于输出变量的类型不同；也在于其输出空间是否为一个度量空间：

1. 分类模型：

分类的输出为定性的。它是用于对离散的变量进行预测，即可以将其他的模型输出离散化。其常用于对事物打上一个标签，没有逼近的概念，对于最终的分类结果只有一个正确与否的判定。

1. 回归模型：

回归的输出为定量的，它是用于对连续的变量进行预测，即可以将其他的模型输出连续化。其是对真实值的一种逼近预测，常用来预测一个值，即会在一个度量空间中去衡量逼近真实值。回归模型会减少所谓的输出值与真实值之间的“误差大小”。

实践思考题2：什么是SVM算法？

答：

SVM即支持向量机，是在分类与回归分析中分析数据的监督式学习模型与相关的学习算法。每个训练实例都会被标记为属于两个类别中的一个或另一个，使其成为非概率的二元线性分类器。它的目标就是为了使训练样本分类正确，让分类函数划分的超平面尽可能地将训练样本分割开。而对于距离超平面最近的几个训练样本点可以称其为“支持向量”，两个异类支持向量到超平面的距离之和被称为“间隔”，通过优化使这个“间隔”最大化的算法便是SVM算法。

对于在原始空间中线性不可分的样本集合，我们可以将原有限的空间映射到维数更高的空间中，而一般来说样本在高维空间的分类更为容易。其中为了计算负荷合理，便采用SVM算法的不同核函数来定义其空间映射即“再生核希尔伯特空间”。对应核函数有线性核、多项式核、高斯核、拉普拉斯核和Sigmoid核等，核的总和可以用来衡量每个测试点相对于待分离的样本集合中数据点的相对接近度。

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？

答：

1. 准确率（Accuracy）：

即对于给定的数据，分类正确的样本数占总样本数的比例。

1. 精确率（Precision）：

假设我们将关注的类定义为正类，其他类定义为负类。通过分类器在数据集上的分类正确错误，我们定义以下4种情况：

* TP ：True Positive， 把正类预测为正类；
* FP ：False Positive， 把负类预测为正类；
* TN：True Negative， 把负类预测为负类；
* FN：False Negative，把正类预测为负类。

则精确率就是指在预测为正类的样本中真实类别为正类所占的比例，即

1. 查全率（召回率 Recall）

召回率是指在所有的正类中被预测为正类的比例，即：

1. F1-Score

即精确率和召回率的调和平均值，即：

1. ROC曲线和AUC

真阳率TPR和假阳率FPR分别为：

使用FPR为横坐标，TPR为纵坐标得到的ROC曲线中，其下的面积即为AUC。因为ROC曲线越靠近左上角，分类器效果越好，所以AUC越大越好。

1. Kappa系数

对总体分类精度和，即第i类真实样本数\*第i类预测出的样本数之和除以样本总数平方。K表示为：

K越大分类效果越好。

1. 海明距离

其衡量了预测标签与真实标签之间的距离，取值在0~1之间。距离为0说明预测的结果与真实结果完全相同，距离为1就说明模型的预测每次恰好为相反的类别。

1. Jaccard相似系数

其与海明距离的不同之处在于分母。当系数为1时，预测结果与实际情况完全相符时；当系数为0时，预测结果与实际情况完全不符。

1. Hinge loss

主要用以使“边缘最大化”，该损失取值在0-1之间。当取值为0时，表示多分类模型分类完全准确；当取值为1时，表示模型的分类结果没有任何意义。

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？

答：

1. 平均绝对误差（MAE）

即对绝对误差损失的平均值，反映预测值误差的实际情况：

1. 平均绝对百分比误差（MPAE）

即对相对误差损失的预期值。其中相对误差就是绝对误差和真实结果值的百分比：

1. 均方误差（MSE）

即观测值与真实值的偏差的平方和与观测次数的比值，回归过程中尽可能使之最小：

1. 均方根误差（RMSE)

即对平均误差的算术平方根的期望值，衡量观测值与真实值之间的偏差：

1. 均方误差对数（MSLE）

即对平方对数差的期望值：

1. 中位绝对误差（MedAE）

即取目标和预测之间的所有绝对差值的中值来计算损失：

1. R2 Score

也被成为R-Squared，为可决系数，也叫拟合优度，反映自变量x对因变量y的变动的解释程度。分子为训练得到模型的预测误差和，分母为随机取值下得到的误差和，一般为观测值的平均值。越接近于1,拟合越好。一般认为超过0.8的模型就拟合优度较高：