openGauss AI特性创新实践课



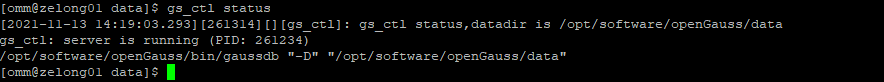
华为技术有限公司

# 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：

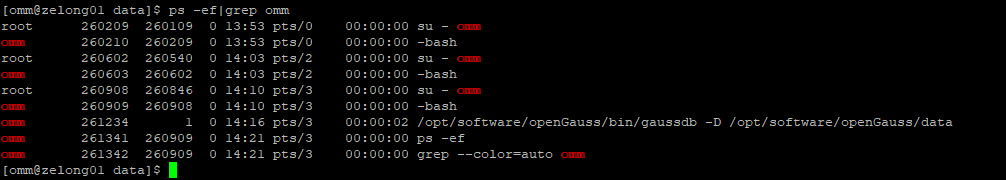
任务一：数据库状态验证

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？

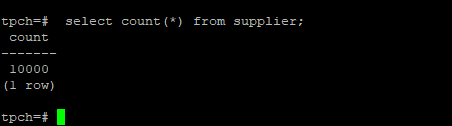
答：通过源码编译安装，可以设定参数，按照需求来进行安装，而且可以自主选择安装的版本，灵活度大； 在实际使用场景中，很多新功能的使用和bug的处理需要及时上线，通过源码编译可以处理这种情况； 通过源码编译，在对源码的理解高的情况下可以自制需要的模块来使用； 对于刚接触linux的人来说，这种安装过程更加透明，每一步出现问题可以找到问题所在，并尝试去解决，而使用安装包等方式出现错误会更加难以解决。

# 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：

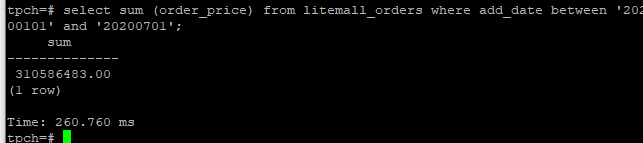
select count(\*) from supplier;;



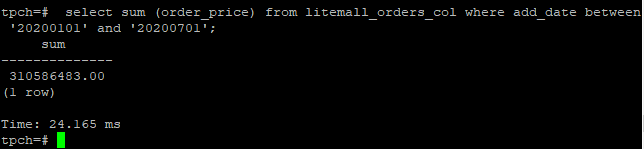
任务二：行存表与列存表执行效率对比

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';

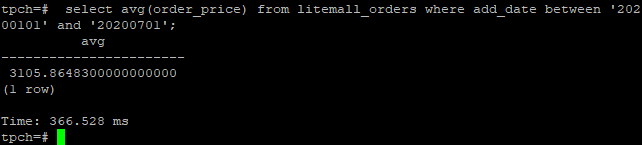


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

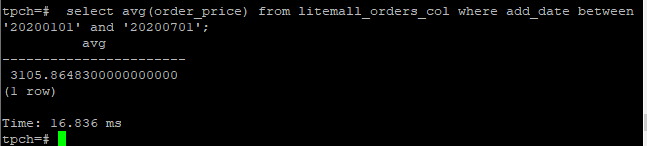


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

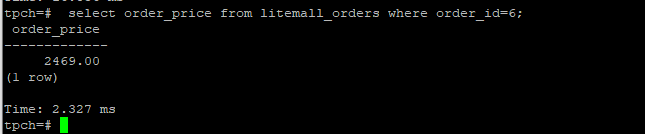


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

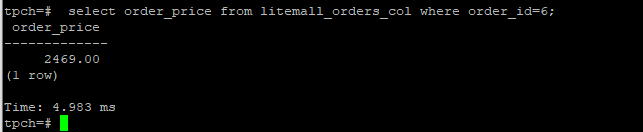


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

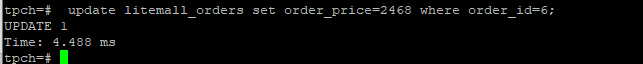


select order\_price from litemall\_orders\_col where order\_id=6;



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

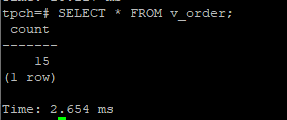
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

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;



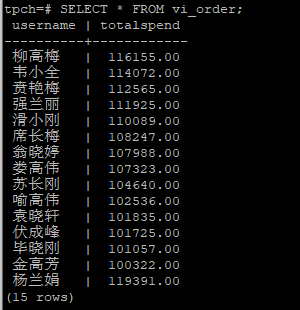
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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：因为行存表和列存表的存储模式不同，行存表以行为形式存储，全表要扫描更多的数据块，读取任意列的成本不同，越往后的列，成本越高；列存表以列为形式存储，读取任何一列的成本都相同，但是读取多列时，需要访问多个文件，成本会更高。因此在查询某一行的信息中行存表执行时间比列存表更快，总结单列相关信息时，由于不存在冗余且相同的数据类型更容易解析，列存表执行时间比行存表更快。

在执行一些随机的增删改查，与多个属性有关的查询操作，或者需要频繁插入或更新的sql语句时，使用行存表效率更高；在操作大多是围绕同一列属性的数据进行，且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的时候，使用列存表效率更高

实践思考题2：全量物化视图与增量物化视图有哪些差别？

答：首先二者都能同步数据，全量是在一定的周期中，把当前系统在周期时间内所有数据复制到目标表/系统，增量就是指抓取某个时刻（更新时间）或者检查点以后的数据来同步，两种视图的区别就在全量与增量上，增量的基础就是全量；对于数据量极大的表的更新，增量

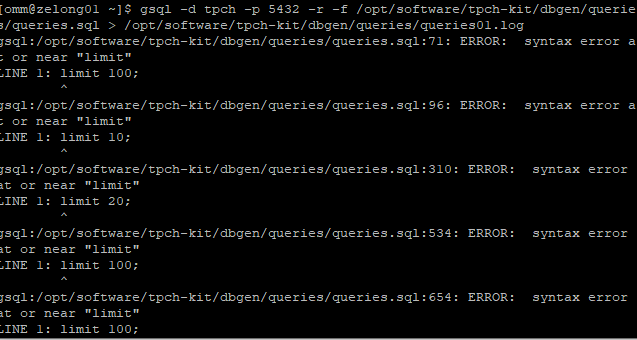
物化视图的计算量更小，在速度上会更快，但是增量物化视图的更新中，对于数据提供方和接收方的要求较高，对增量内容的处理要很严谨，且出现问题时难以定位，所以在数据量不是特别巨大的情况下，一般使用全量物化视图更新。

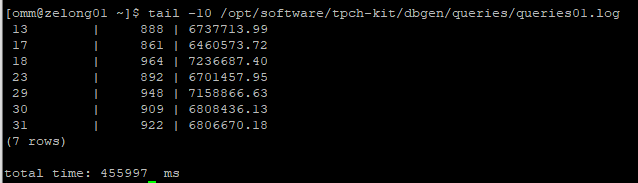
# 关卡三、openGauss的AI4DB特性应用

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





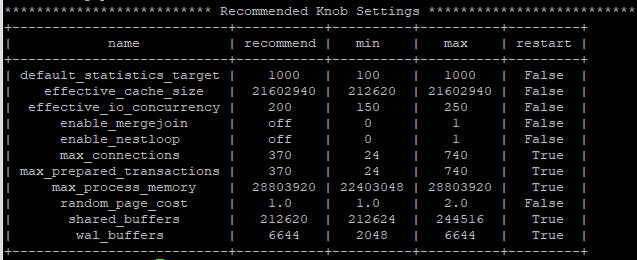
修改sql脚本错误后上传并将脚本属组改为omm





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

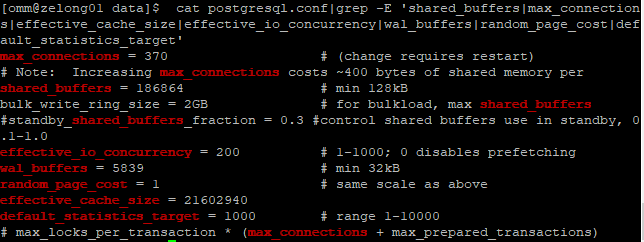
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

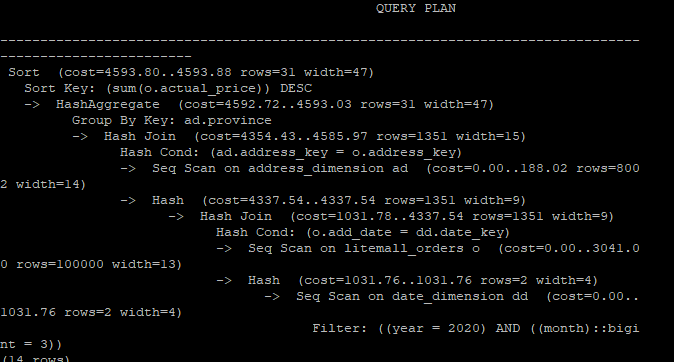
AND o.add\_date = dd.date\_key

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

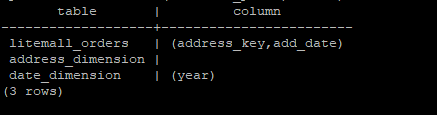
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

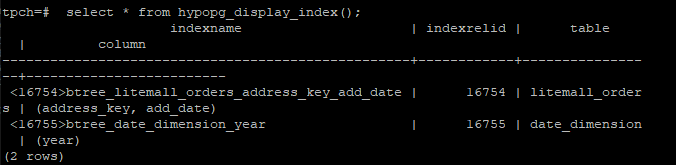
GROUP BY ad.province

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。

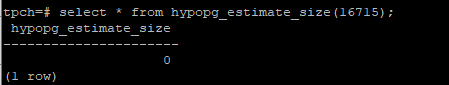
select \* from hypopg\_display\_index();

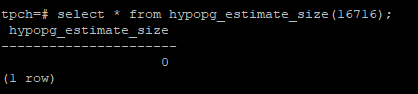


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

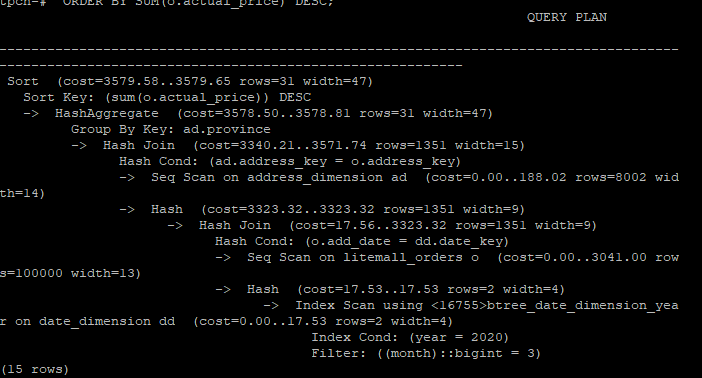
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

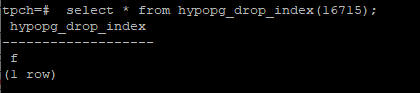
GROUP BY ad.province

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



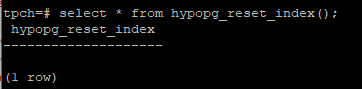
6. 删除某一个索引虚拟列，将执行结果截图。

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



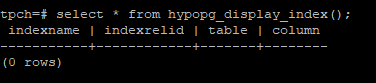
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

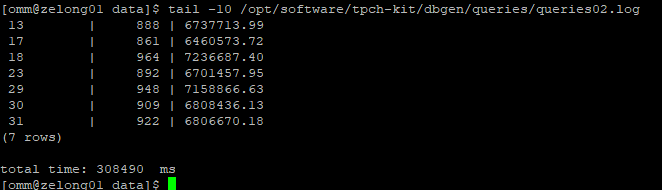
select \* from hypopg\_display\_index();



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：

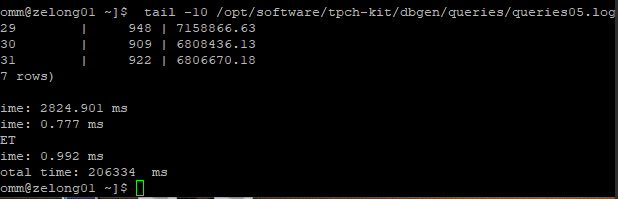
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

答：优化的参数有：shared\_buffers 数据页缓冲区 max\_connections 最大连接数

Effective\_cache\_size 有效缓存大小 effective\_io\_concurrency 有效的io并发

wal\_buffers 预写日志缓冲区 random\_page\_cost 随机页数 default\_statistics\_target 默认统计目标

X-Tuner 根据recommend 生成的报告，通过对相关参数的优化，让服务器的资源利用率尽可能的高。

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：创建索引可以大大提高系统的性能。

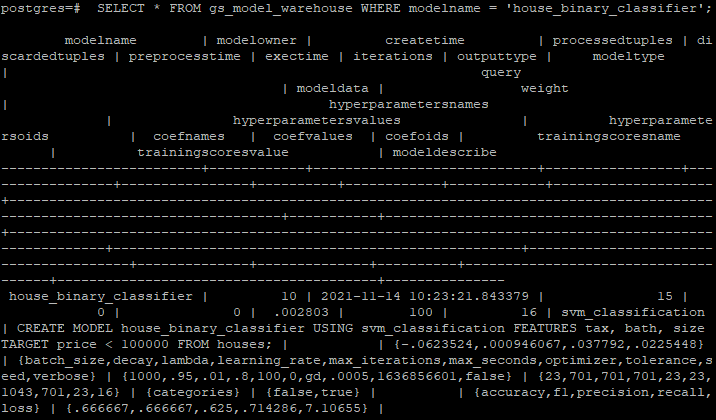
第一，通过创建唯一性索引，可以保证[数据库](http://lib.csdn.net/base/mysql)表中每一行数据的唯一性。  
第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。  
第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。  
第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。  
第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

优化：选用合适的字段，使用连接代替子查询，使用联合来代替手动创建的临时表，优化sql语句，使用事务和锁定表的方法。

# 关卡四、openGauss的DB4AI特性应用

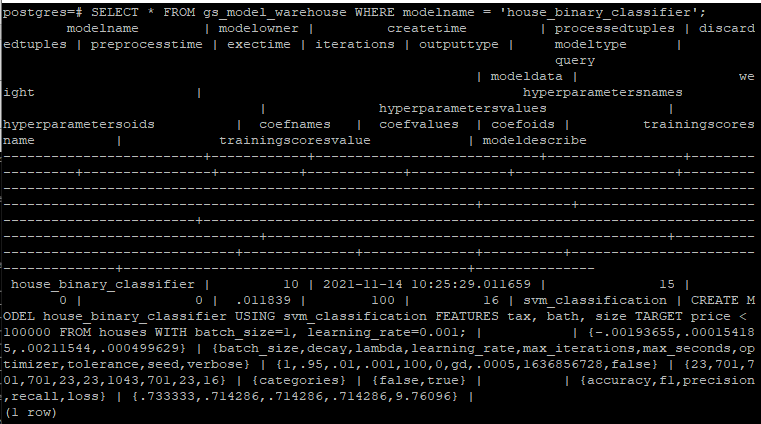
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



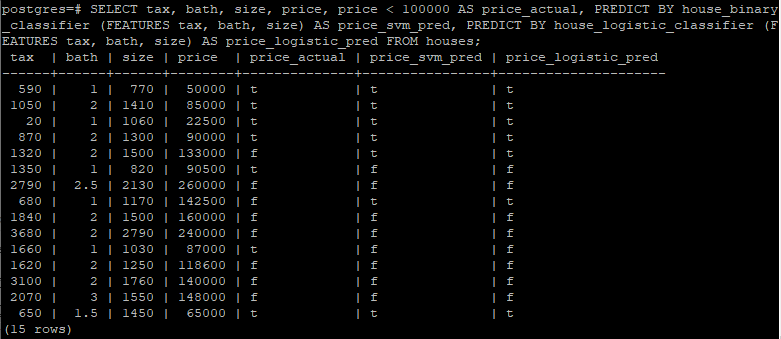
任务二：观察新模型的信息，将执行结果截图。

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：在输出的类型上，分类模型在预测后会给出一个离散的y值，即预测结果属于哪一类，

回归模型是进行连续性预测，通过训练预测出一个连续的y值，即结果是多少。

实践思考题2：什么是SVM算法？

答：SVM即支持向量机算法，属于分类模型上常用的机器学习算法。在二维空间上，两类点被一条线分隔开称为线性可分。在n维空间中，要分开两个线性可分的点集合，我们需要找到一个超平面，从二维空间扩展到多维空间时，分开左右两类点的直线 wx + b = 0就成为了一个超平面。为了使这个超平面更具有鲁棒性，我们需要寻找到一个最大间隔把两类点分开的超平面，称为最大间隔超平面。该超平面到两类点的最近点的距离最大，且两类点分布在超平面的两侧。其中，距离最大间隔超平面的点称为支持向量。

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：分类问题的评价指标有

准确率：对于给定的测试数据集，分类器正确分类的样本数与总样本数之比

精确率：正确分类的正例个数占分类为正例的实例个数的比例

召回率：正确分类的正例个数占实际正例个数的比例

TPR：真正率， TPR代表能将正例分对的概率

FPR：假正率， FPR代表将负例错分为正例的概率

AUC：表征了分类器把正样本排在负样本前边的能力

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：回归问题的评价指标有：

MAE：平均绝对误差

MSE：均方误差

RMSE：均方根差

SD：标准差