openGauss AI特性创新实践课



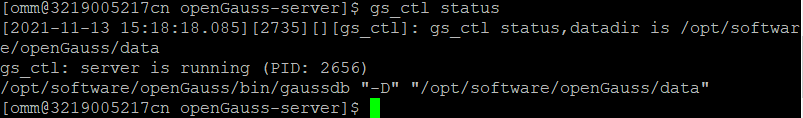
华为技术有限公司

# 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：

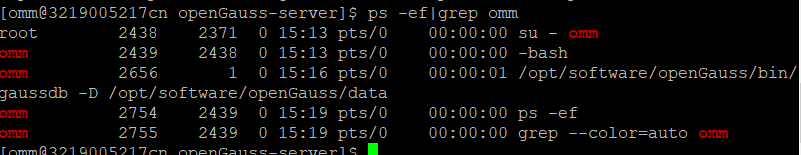
任务一：数据库状态验证

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？

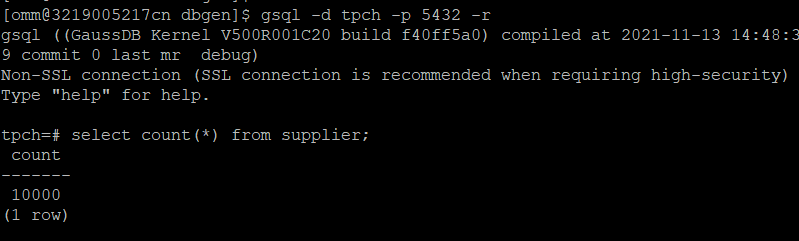
使用源码编译安装数据库，可以在编译安装过程，设定参数，按照需求，进行安装，并且安装的版本，可以自己选择，灵活性比较大。

# 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：

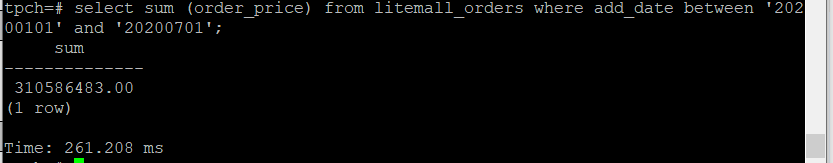
select count(\*) from supplier;;



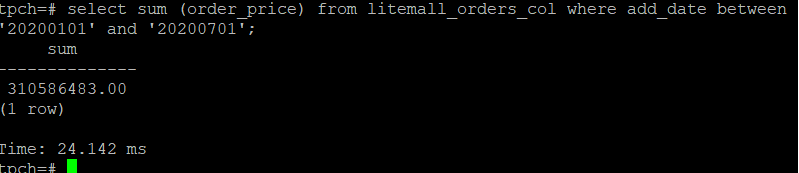
任务二：行存表与列存表执行效率对比

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';

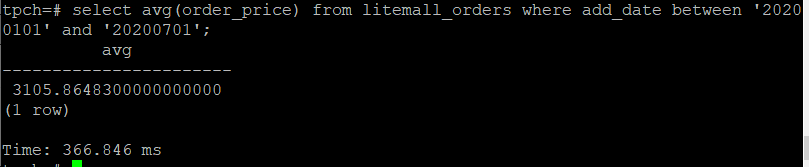


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

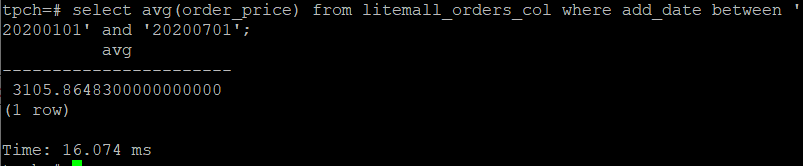


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

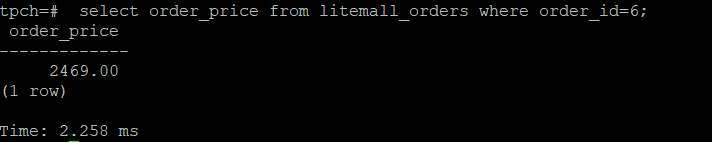


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

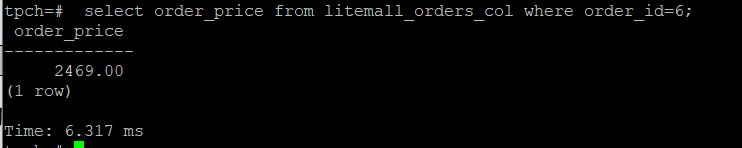


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

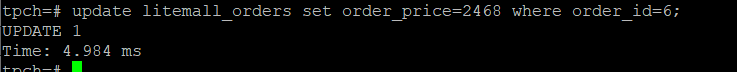


select order\_price from litemall\_orders\_col where order\_id=6;

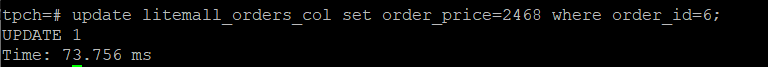


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

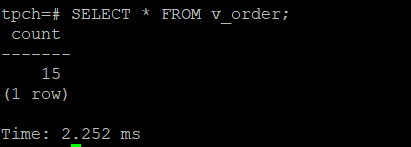
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



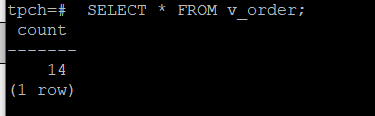
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

不同之处：

1. 在数据写入上：
   1. 行存表的写入是一次完成。
   2. 列存表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。
   3. 数据修改时，行存表是在指定位置写入一次，列存表是将磁盘定位到多个列上分别写入，这个过程仍是行存表的列数倍
2. 在数据读取上：
   1. 行存表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。
   2. 列存表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
3. 从存储的数据上：
   1. 列存表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。
   2. 行存表要复杂得多，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，列存表的解析过程更有利于分析大数据。

行存表效率更高的使用场景：

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。

列存表效率更高的使用场景：

1. 适用于OLAP大数据量查询

实践思考题2：全量物化视图与增量物化视图有哪些差别？

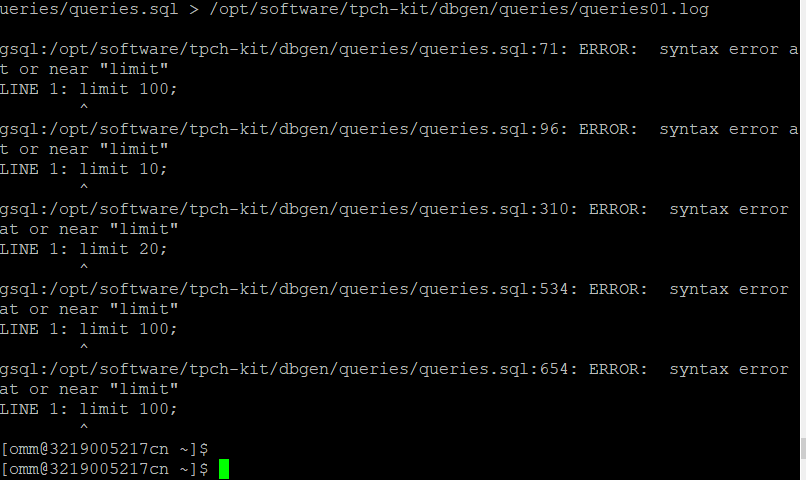
1. 全量物化视图：刷新机制是首先物化视图对应表中的数据采用delete全部删除，然后再从原表中使用insert把数据重新插入。
2. 增量物化视图：刷新是基于主表上的物化视图日志进行刷新的，主表上每插入或删除一条数据，对应物化视图日志中同样会插入一条数据（主表更新一条数据，对应物化视图中会插入两条记录数据）。

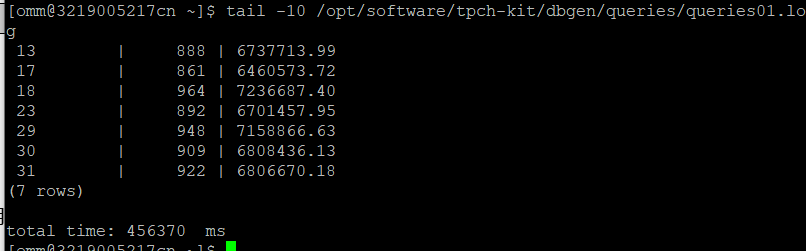
# 关卡三、openGauss的AI4DB特性应用

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：

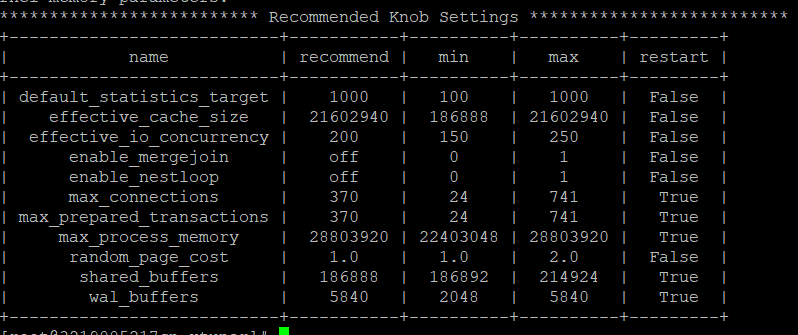
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log





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

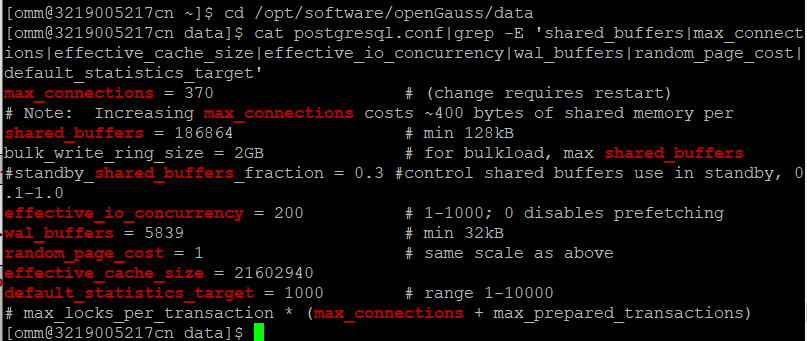
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

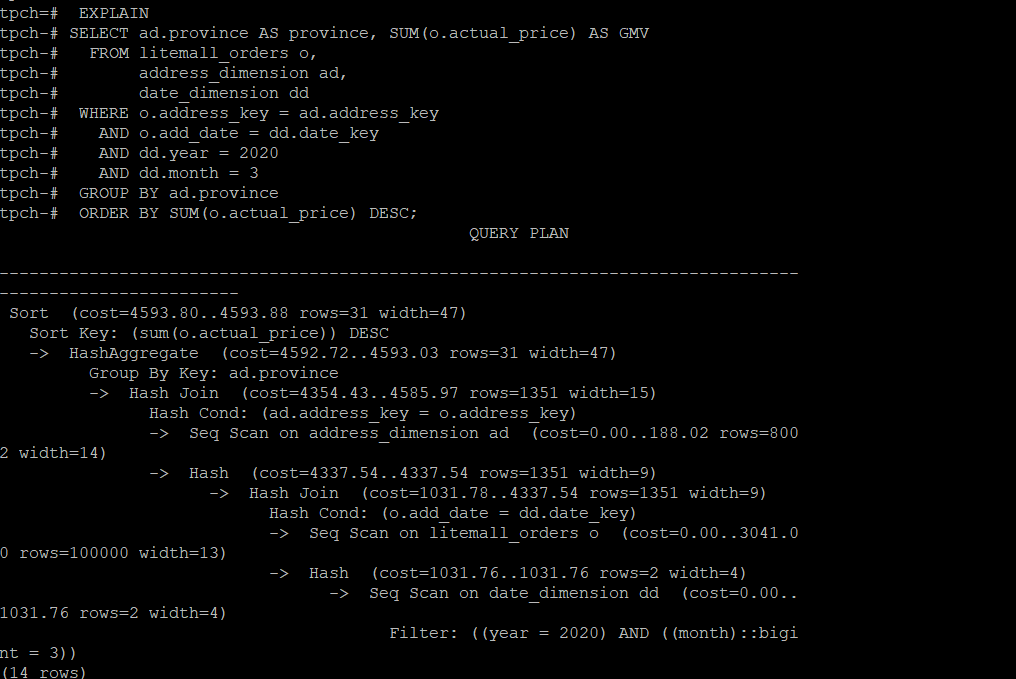
AND o.add\_date = dd.date\_key

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

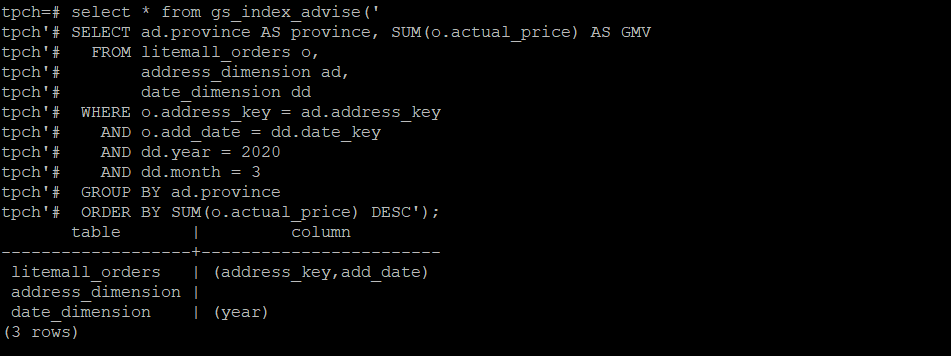
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

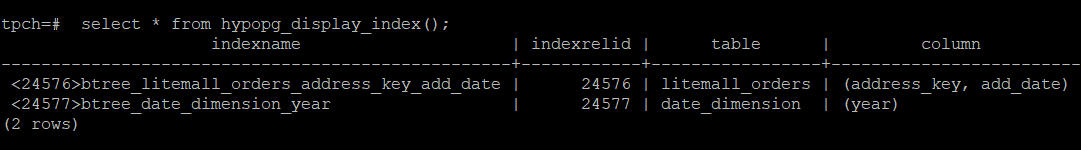
GROUP BY ad.province

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。

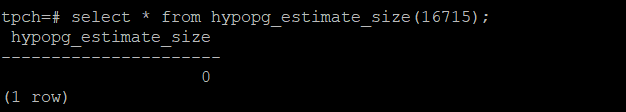
select \* from hypopg\_display\_index();

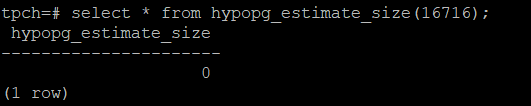


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

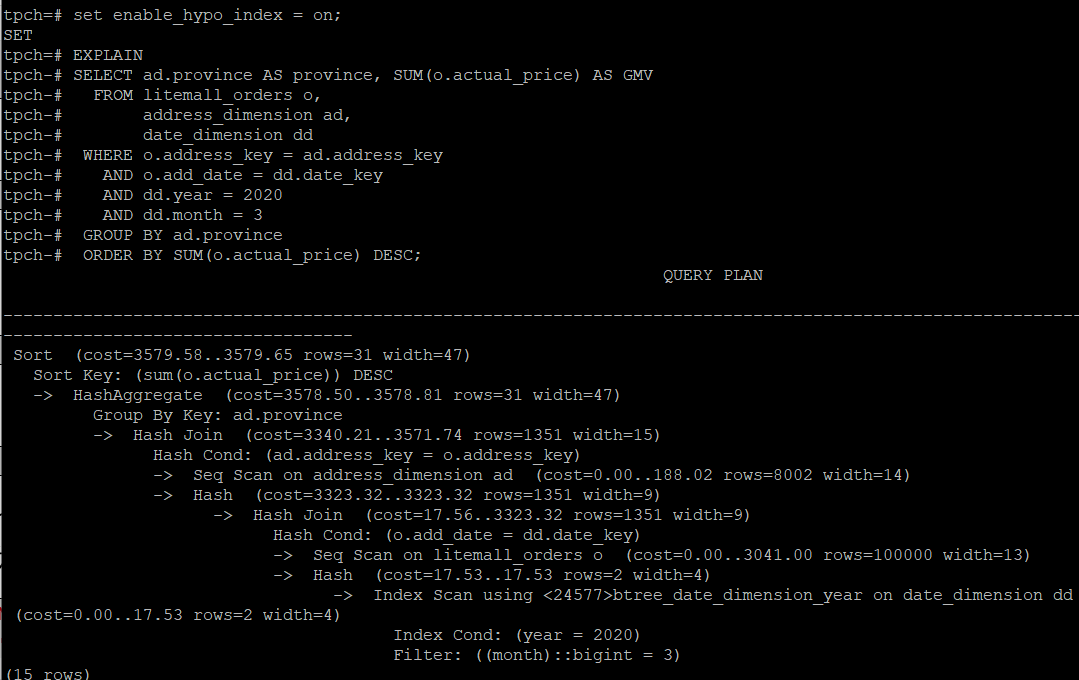
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

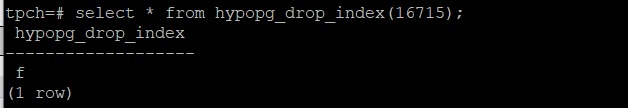
GROUP BY ad.province

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



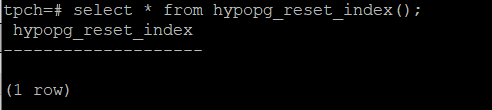
6. 删除某一个索引虚拟列，将执行结果截图。

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



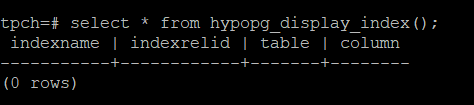
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

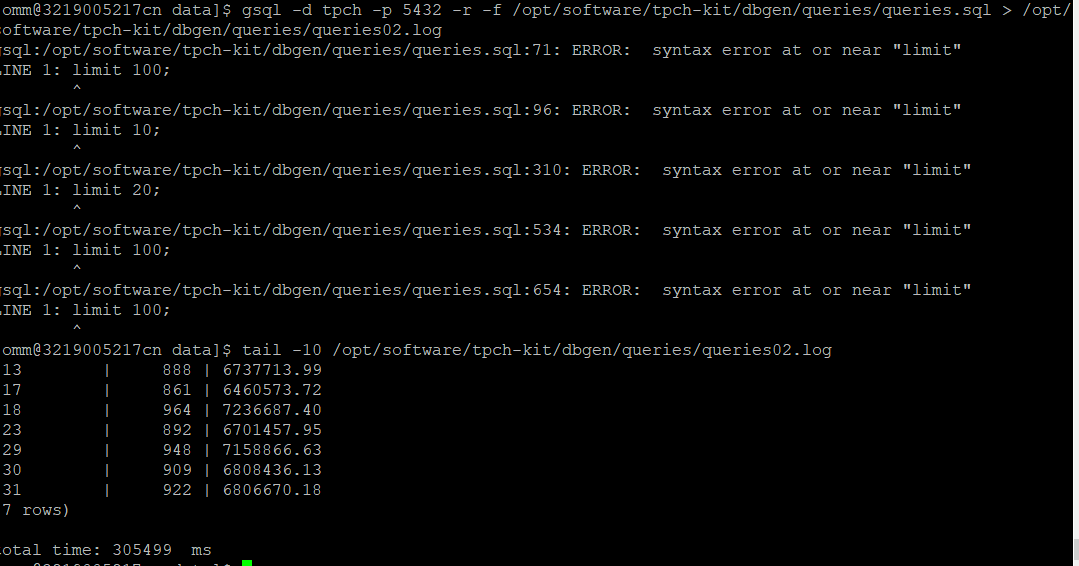
select \* from hypopg\_display\_index();



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？

优化的参数：default\_statistics\_target | effective\_cache\_seze | effective\_io\_concurrency | enable\_nestloop | max\_connections | max\_prepared\_trasactions | max\_process\_memory | random\_page\_cost | shared\_buffers | wal\_buffers

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？

索引使用的好处：

1. 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。
2. 可以大大加快数据的检索速度。
3. 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。
4. 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。
5. 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。

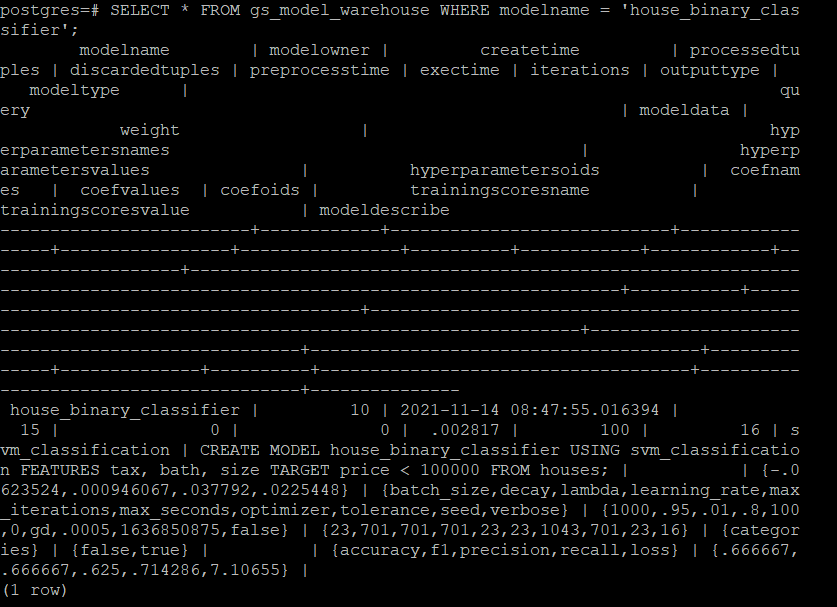
数据库优化方式：

1. 硬优化
   1. 配置多核心和频率高的cpu,多核心可以执行多个线程
   2. 配置大内存,提高内存,即可提高缓存区容量,因此能减少磁盘I/O时间,从而提高响应速度
   3. 配置高速磁盘或合理分布磁盘:高速磁盘提高I/O,分布磁盘能提高并行操作的能力
   4. 分库分表：把一个库拆分为多个库，部署在多个数据库服务上，这时作为主库承载写入请求。然后每个主库都挂载至少一个从库，由从库来承载读请求。
2. 软优化
   1. 子查询
   2. 分解表：对于字段较多的表,如果某些字段使用频率较低,此时应当,将其分离出来从而形成新的表
   3. 增加中间表：对于将大量连接查询的表可以创建中间表,从而减少在查询时造成的连接耗时
   4. 增加冗余字段：类似于创建中间表,增加冗余也是为了减少连接查询

# 关卡四、openGauss的DB4AI特性应用

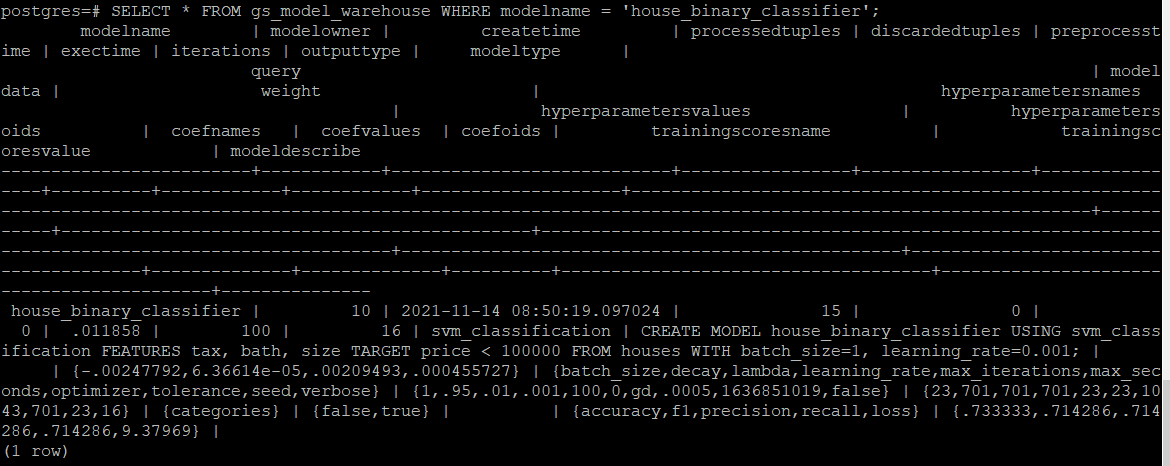
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



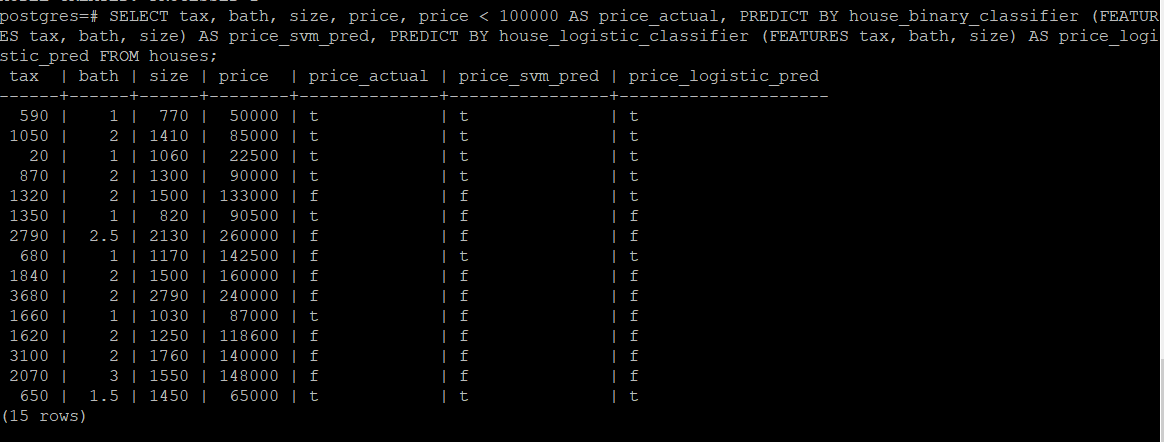
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型和回归模型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是一种二分类模型，它将实例的特征向量映射为空间中的一些点，SVM 的目的就是想要画出一条线，以 “最好地” 区分这两类点，以至如果以后有了新的点，这条线也能做出很好的分类。SVM 适合中小型数据样本、非线性、高维的分类问题。

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？

1. 准确率(Accuracy)：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。
2. 精确率(Precision)：检索出相关文档数与检索出的文档总数的比率（正确分类的正例个数占分类为正例的实例个数的比例），衡量的是检索系统的查准率。
3. 召回率(Recall)：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（正确分类的正例个数占实际正例个数的比例），衡量的是检索系统的查全率。
4. F1值：对Precision和Recall进行整体评价。
5. 综合评价指标F-Score：Precision和Recall的加权调和平均。
6. ROC曲线：接收者操作特征（receiver operating characteristic），ROC曲线上每个点反映着对同一信号刺激的感受性。
7. AUC曲线：ROC曲线下的面积，介于0.1和1之间。AUC作为数值可以直观的评价分类器的好坏，值越大越好。

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？

1. 均方误差（MSE）：用真实值-预测值然后平方之后求和平均，预测数据和原始数据对应点误差的平方和的均值，线性回归用MSE作为损失函数。
2. 均方根误差（RMSE）：也叫回归系统的拟合标准差，是MSE的平方根
3. R方（R-Squared）：衡量模型拟合度的一个量,是一个比例形式，被解释方差/总方差。