openGauss AI特性创新实践课



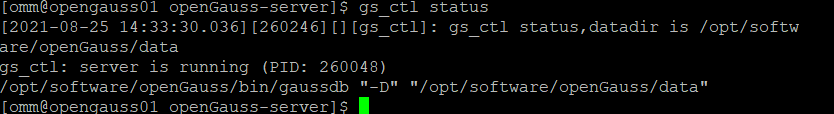
华为技术有限公司

# 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：

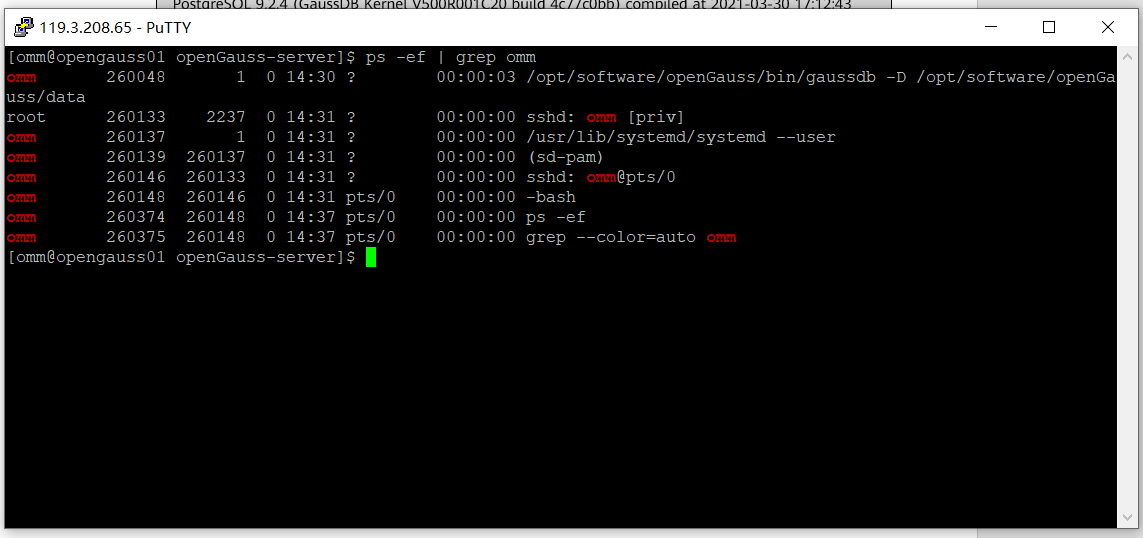
任务一：数据库状态验证

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？

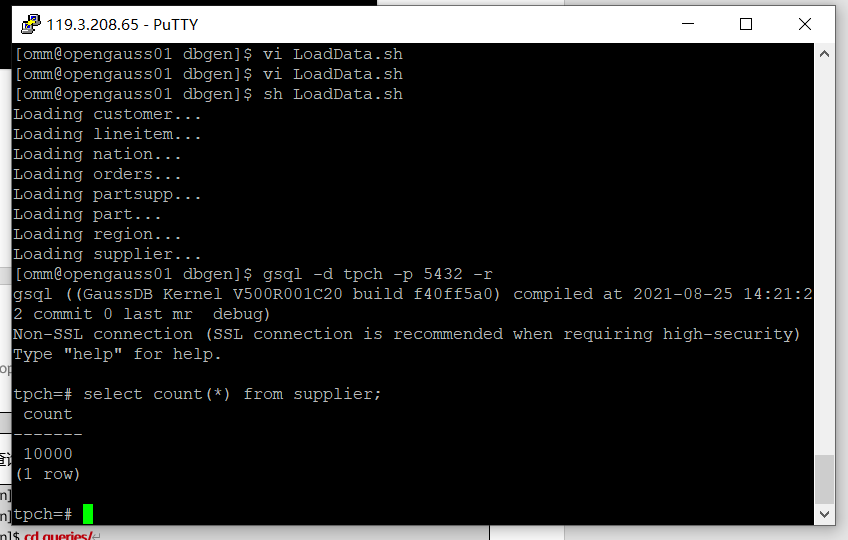
可以根据需求自定义安装配置以及调整优化设置，例如通过修改build.sh，可以选择安装[debug | release | memcheck]版软件，以及在鲲鹏、x86等不同硬件平台上的优化。

# 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：

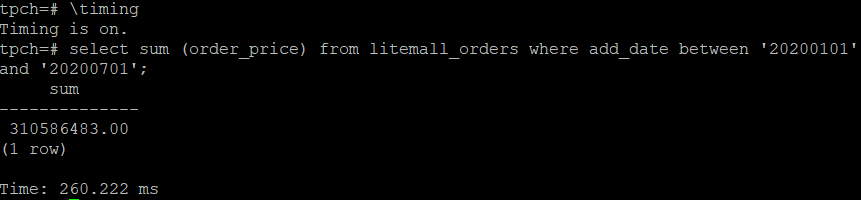
select count(\*) from supplier;;



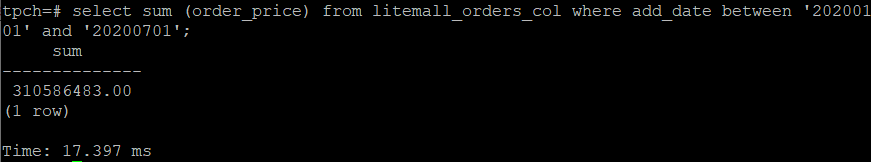
任务二：行存表与列存表执行效率对比

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';

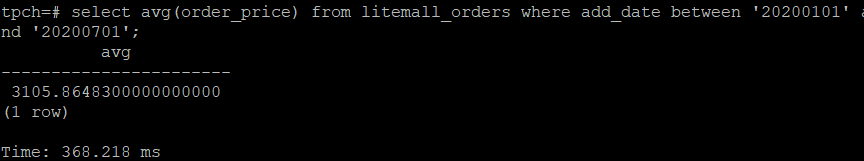


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

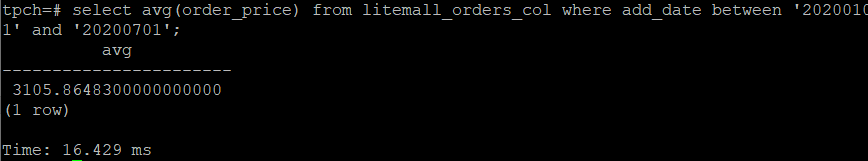


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

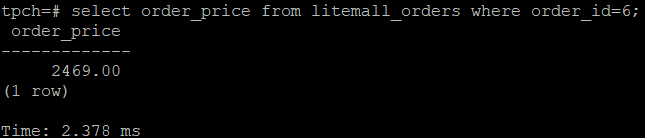


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

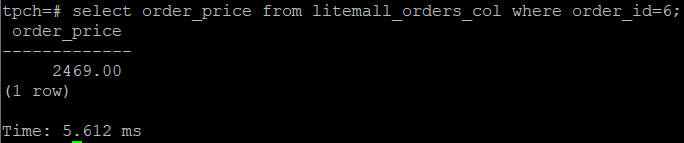


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

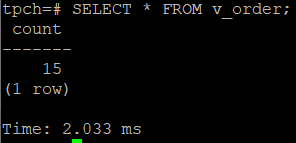
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



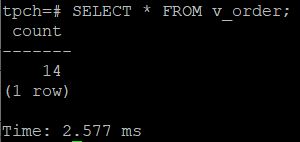
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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。

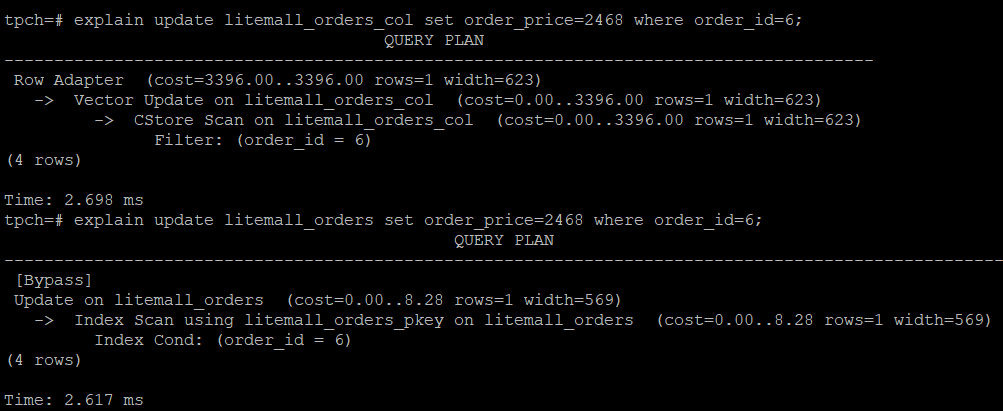
SELECT \* FROM vi\_order;



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？

行存表和列存表，顾名思义在内部存储结构上不同。行存表以行为单位聚集成数据块，进行存储，在查找时一次获取到一行的全部数据，而列存表以列为单位，将不同行的相同列聚合存储，因此一次只能查找到某单元格的数据（例如HBase的(row, colfam, column)三元键设计），但查询中可以并行执行。二者相比，当一次查找需要对一行数据的多个列的数据进行读写时，行存表效率更高；而当表的列数巨大，一次只需要读取一行的少数几列，或需要读取大量不同行的相同列时，列存表更高效。

本实验中使用相同的10000行x 36列行存表和列存表进行测试，结果可以看到，步骤1求一个列的sum值，列存比行存效率高10倍以上，步骤2求avg值也类似；步骤3单行单列的读取，二者效率都很高；步骤4单行单列的写入，通过EXPLAIN SQL语句可以看出：



列存操作是在对应列表中进行了一次扫描，并更新对应数据，而行存由于在order\_id使用了索引，可以快速定位到order\_id=6的行，并对行进行更新，因此此操作中行存效率更高。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

物化视图是从原始表中将数据拷贝出来，形成有物理存储的一组新数据，其主要问题是维护 (maintenance)，即原始表中数据更新时如何保持已经物化的视图与原表保持一致。一般需要使用REFRESH MATERIALIZED VIEW命令来手动刷新视图，此刷新会忽视当前物化视图中已有的数据，全量地重新从原表中获取数据。Postgre文档(https://wiki.postgresql.org/wiki/Incremental\_View\_Maintenance)中提出了增量(Incremental)更新的的物化视图，其在刷新时只从原表中抓取增量数据，提高刷新效率。本实验中，默认的全量物化视图执行REFRESH命令时间长于增量物化视图，估计当视图中数据量增大时该差距将更加明显：





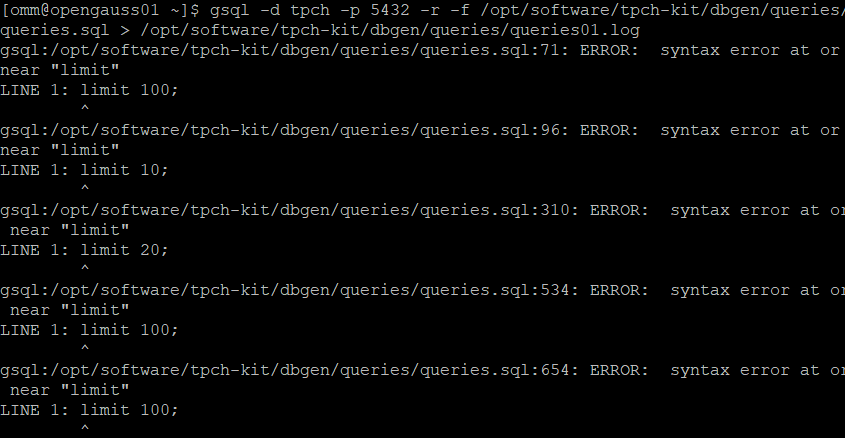
查资料发现也有提出原表数据更新时自动刷新物化视图的方案，而测试发现openGauss未采用。个人理解其原因可能是这样会增加一般增删改操作的开销，而这种开销可能难以被发现，或者得不偿失。

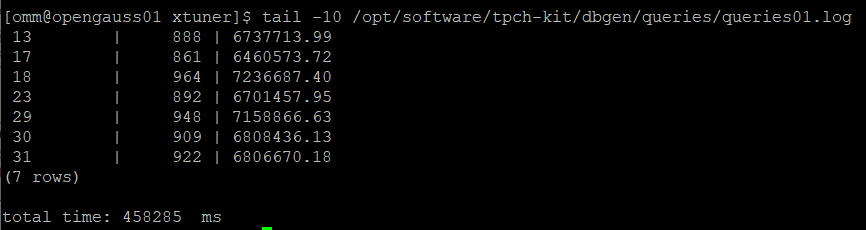
# 关卡三、openGauss的AI4DB特性应用

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：

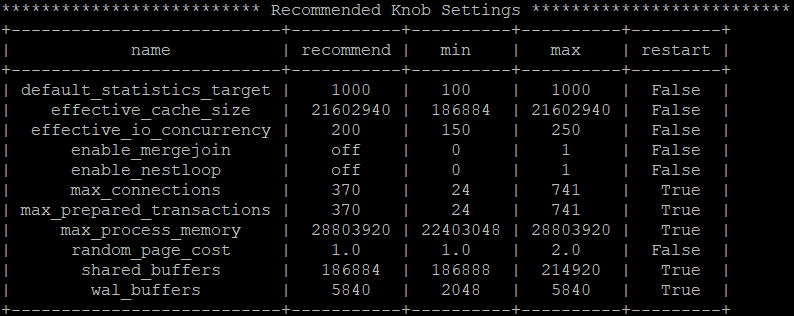
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log





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

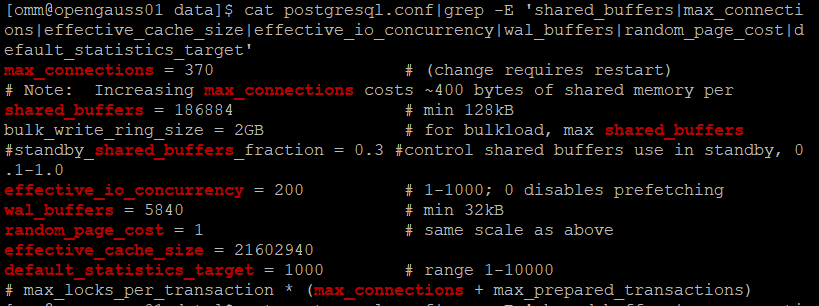
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

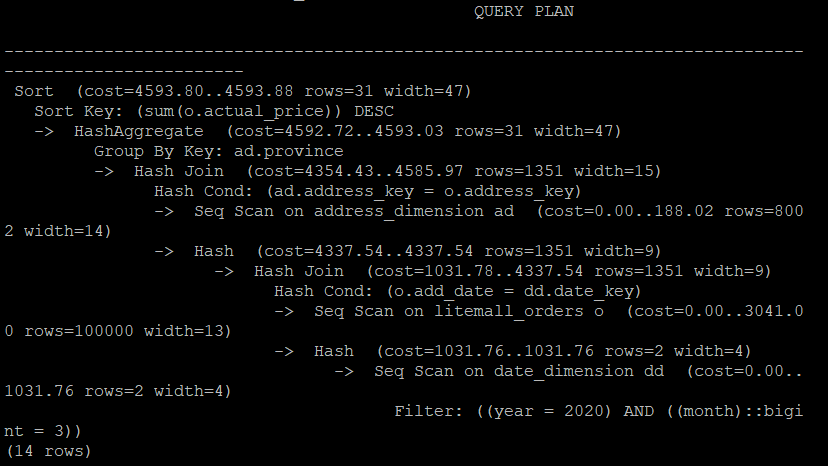
AND o.add\_date = dd.date\_key

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

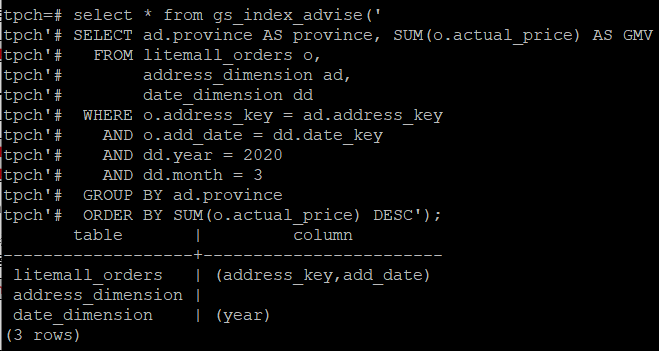
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

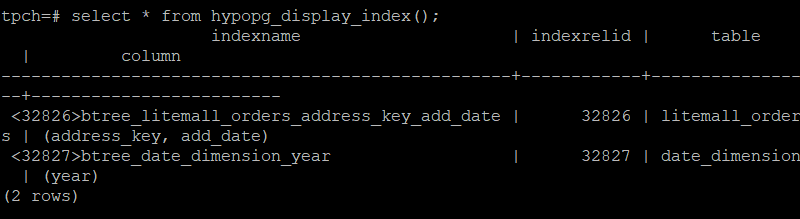
GROUP BY ad.province

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。

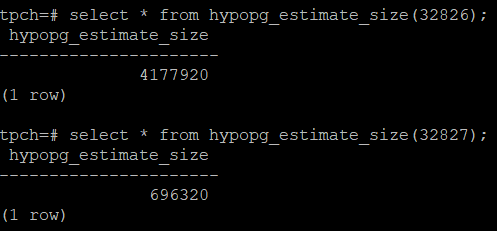
select \* from hypopg\_display\_index();



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

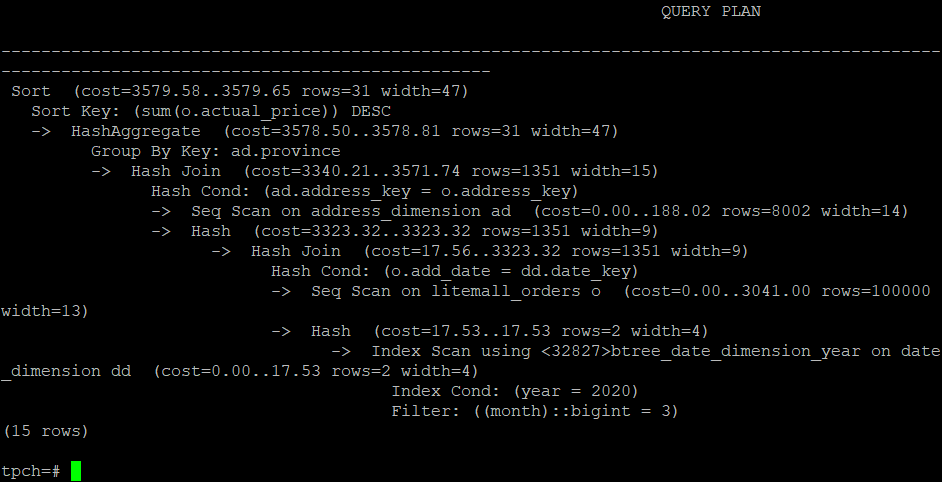
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

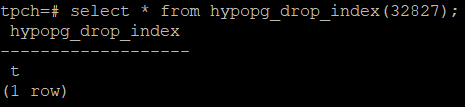
GROUP BY ad.province

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



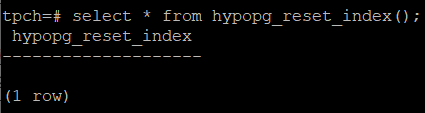
6. 删除某一个索引虚拟列，将执行结果截图。

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



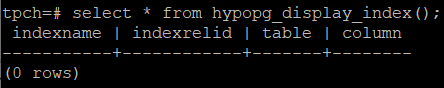
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

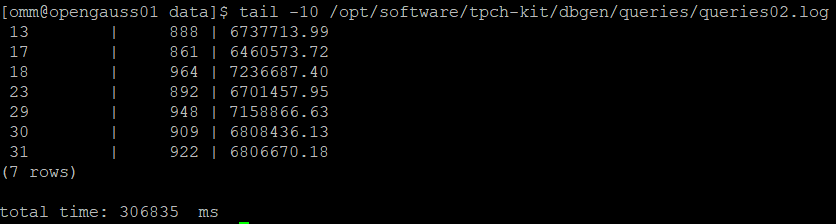
select \* from hypopg\_display\_index();



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

这步实验失败了。我尝试了一下用官方文档中提到的workload级别索引推荐(<https://opengauss.org/zh/docs/2.0.1/docs/Developerguide/workload%E7%BA%A7%E5%88%AB%E7%B4%A2%E5%BC%95%E6%8E%A8%E8%8D%90.html>)，但提示必须有一个--schema参数，网上没有相关资料，如果给一个空文档运行出来结果也是空的，不知道是文档还没有记录还是什么原因。

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？

首先需要说明，本实验中数据库刚刚创建，还没有足够多的事务记录，X-Tuner给出的参数优化很不稳定，目前主要推荐优化的参数为：

max\_connections：数据库最大并发连接数；

max\_prepared\_transactions：支持的最大PREPARE TRANSACTION数，官方文档建议至少设置为与max\_connections相等，这里X-Tuner给的建议也的确是这样的（二者均为370，即最大值的一半）；

shared\_buffers：使用的共享内存大小；

wal\_buffers：WAL(Write Ahead Logging)数据的共享内存空间。

数据库引擎的这些运行参数会直接影响其性能，例如共享连接数过低导致不能服务足够多的客户端，共享内存太小导致响应速度下降甚至崩溃等。

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？

使用索引可以加速从表中查找到某行的速度，从而提升整个SQL语句的执行速度。

除索引和参数，其他优化方式：

1. 选择合适的数据库运行环境和部署方式；

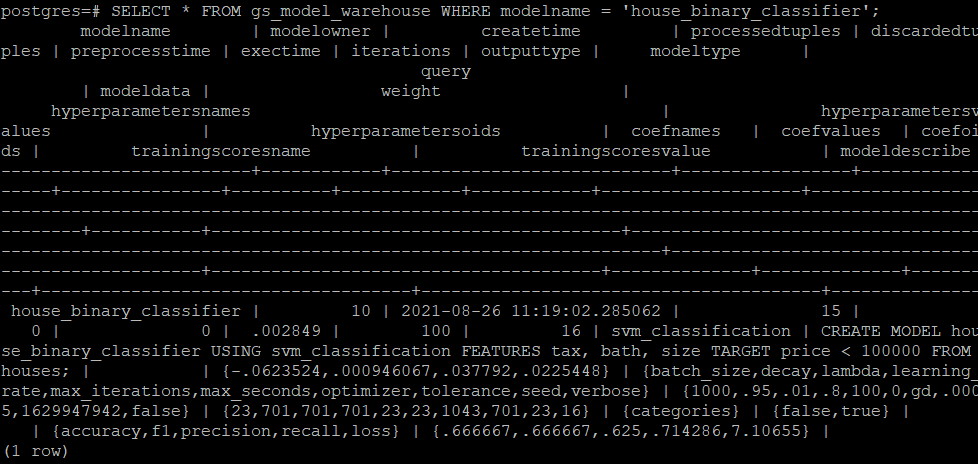
2. 合理建库建表，根据实际场景选择行/列存储，主键外键，约束，触发器等；

3. 合理编写SQL语句，例如只SELECT需要的数据，衡量实现同一效果的不同语句（如WHERE和HAVING）的效率等；

# 关卡四、openGauss的DB4AI特性应用

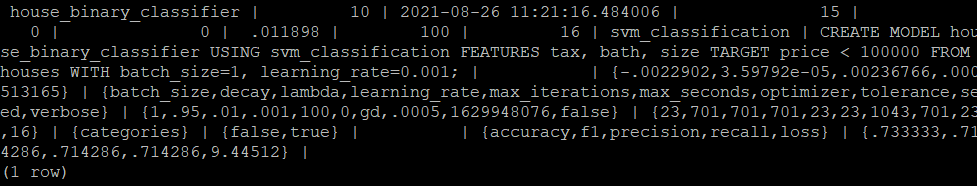
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



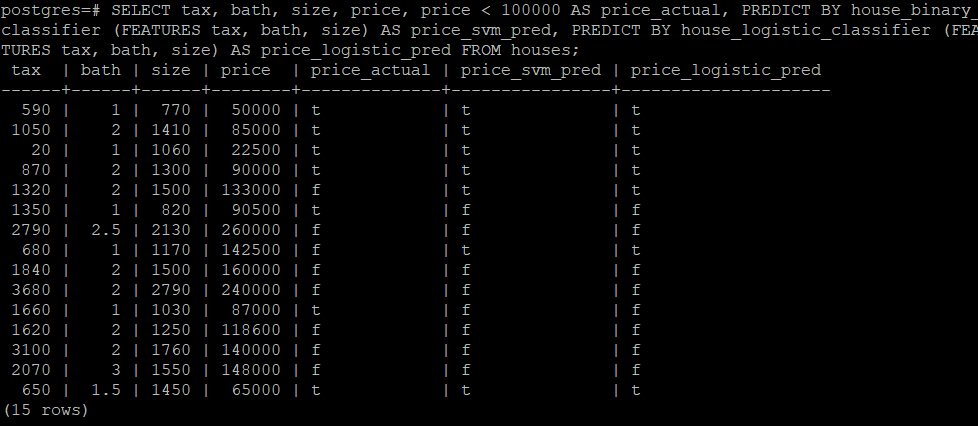
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型与回归模型都用于解决预测问题，区别在于分类问题的输出变量为有限个离散变量（即分类/标签），而回归问题的输入变量与输出变量均为连续变量。

实践思考题2：什么是SVM算法？

（这里确实没学过，参考维基了orz菜.jpg）支持向量机（support vector machine，SVM）是监督式学习模型与相关的学习算法。给定一组训练实例，每个训练实例被标记为属于两个类别中的一个或另一个，SVM训练算法创建一个将新的实例分配给两个类别之一的模型，使其成为非概率二元线性分类器。SVM模型是将实例表示为空间中的点，这样映射就使得单独类别的实例被尽可能宽的明显的间隔分开。然后，将新的实例映射到同一空间，并基于它们落在间隔的哪一侧来预测所属类别。

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)：分类正确的样本数占总样本数的比例。

精确率(Precision)：预测为正类的样本中真正类所占的比例。

召回率(Recall)：所有的正类中被预测为正类的比例。

F1-Score：精确率和召回率倒数之和的倒数，是二者的的综合平均。

ROC曲线和AUC：ROC曲线即横轴为TPR，纵轴的FPR的曲线，AUC为ROC曲线与横轴围成的面积，数值越大分类效果越好。

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？

均方误差(MSE)和均方根误差(RMSE)：观测值与真值偏差的平方的平均值，以及其开根。

平均绝对误差(MAE)：观测值与真值偏差绝对值的平均值。

R Square：