openGauss AI特性创新实践课



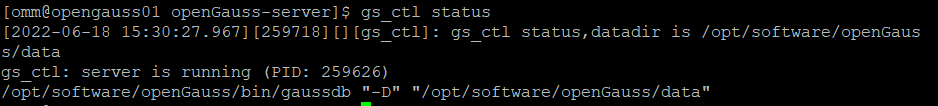
华为技术有限公司

# 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：

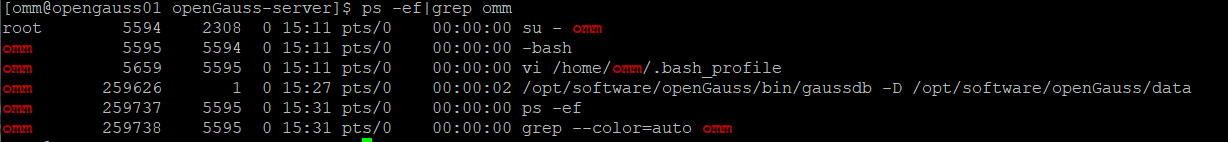
任务一：数据库状态验证

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？

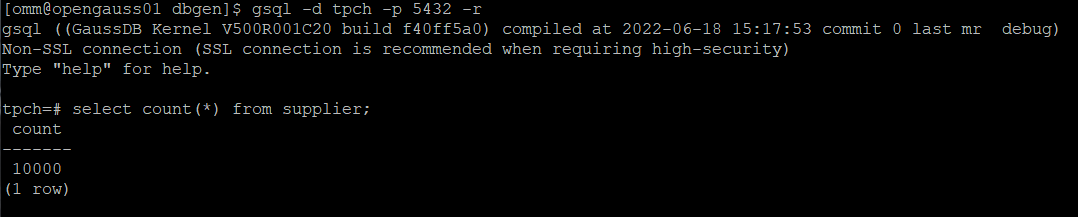
编译安装过程，可以设定参数，按照需求，进行安装，并且安装的版本，可以自己选择，灵活性比较大，可定制性比较强。基于源码编译方式可以自定义功能和模块，而且可以自定义部署的路径，统一的管理，减轻后期的维护工作量。

# 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：

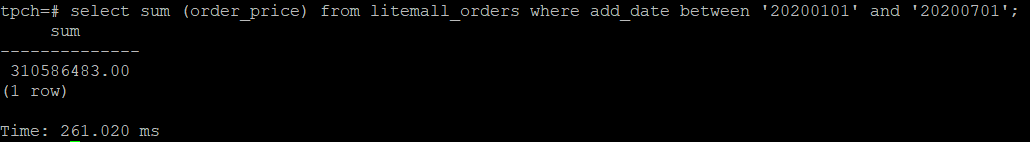
select count(\*) from supplier;;



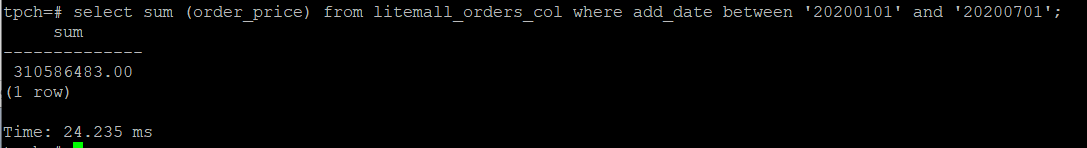
任务二：行存表与列存表执行效率对比

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';

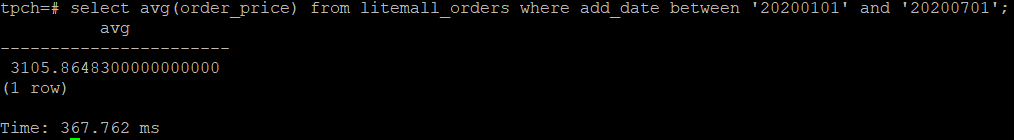


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

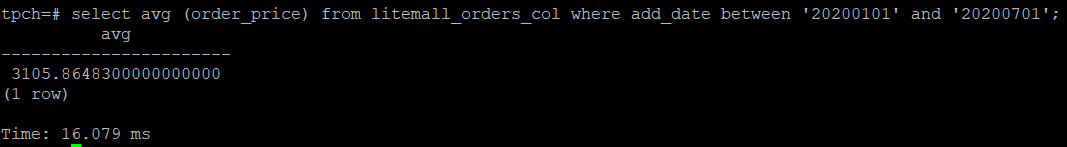


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

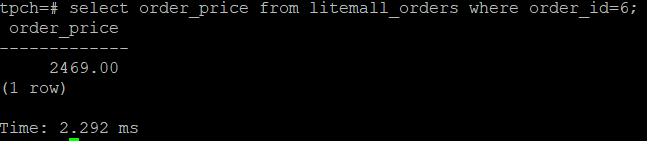


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

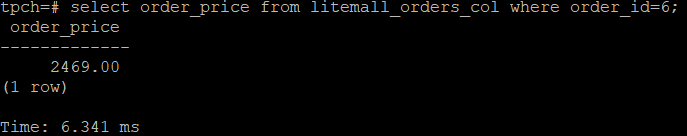


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

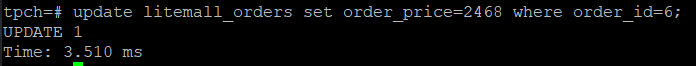


select order\_price from litemall\_orders\_col where order\_id=6;



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

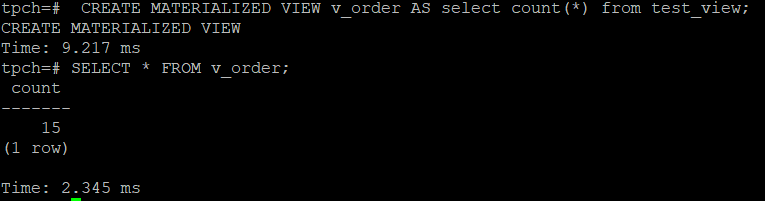
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



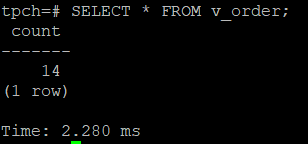
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

行式数据库，数据是按行存储的，当我们读某个列的时候，不能只读列这部分数据，必须把整个block读取到[内存](https://so.csdn.net/so/search?q=%E5%86%85%E5%AD%98&spm=1001.2101.3001.7020)中，然后再把这些列的数据取出来，换句话说，为了读表中某些列的数据，必须把整个列的行读完，才可以读到这些列；列式数据库，数据是按列存储的，每一列单独存放。只访问查询涉及的列,大量降低系统io，每一列有一个线索来处理，支持查询的[高并发](https://so.csdn.net/so/search?q=%E9%AB%98%E5%B9%B6%E5%8F%91&spm=1001.2101.3001.7020)。当需要读某个列的时候，只需要把相关的文件或块读到内存中去，整个列就会被读出来，这样I/O会少很多。

行式存储的适用场景包括：

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。

列式存储的适用场景包括：

1、查询过程中，可针对各列的运算并发执行(SMP)，在内存中聚合完整记录集***，***可降低查询响应时间;

2、可在数据列中高效查找数据，无需维护索引(任何列都能作为索引)，查询过程中能够尽量减少无关IO，避免全表扫描;

3、因为各列独立存储，且数据类型已知，可以针对该列的数据类型、数据量大小等因素动态选择压缩算法，以提高物理存储利用率;如果某一行的某一列没有数据，那在列存储时，就可以不存储该列的值，这将比行式存储更节省空间

实践思考题2：全量物化视图与增量物化视图有哪些差别？

物化视图实际上就是一种特殊的物理表，物化视图是相对普通视图而言的。普通视图是虚拟表，应用的局限性较大，任何对视图的查询实际上都是转换为对SQL语句的查询，性能并没有实际上提高。而物化视图实际上就是存储SQL所执行语句的结果，起到缓存的效果。

全量：简单来说，就是在一定的周期中，把当前系统在周期时间内**所有**数据复制到目标表/系统这样的同步方式就叫做—>全量。

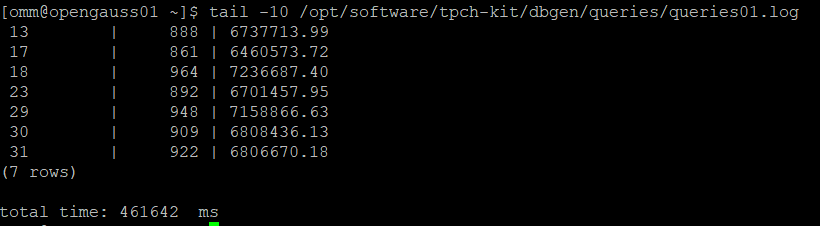
增量：增量同步的前提是全量，然后再更具规则增量同步；增量的基础是全量，就是你要使用某种方式先把全量数据拷贝过来，然后再采用增量方式同步更新。增量的话，就是指抓取某个时刻（更新时间）或者检查点（checkpoint）以后的数据来同步，不是无规律的全量同步。这里引入一个关键性的前提：副本一端要记录或者知道（通过查询更新日志或者订阅更新）哪些更新了

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

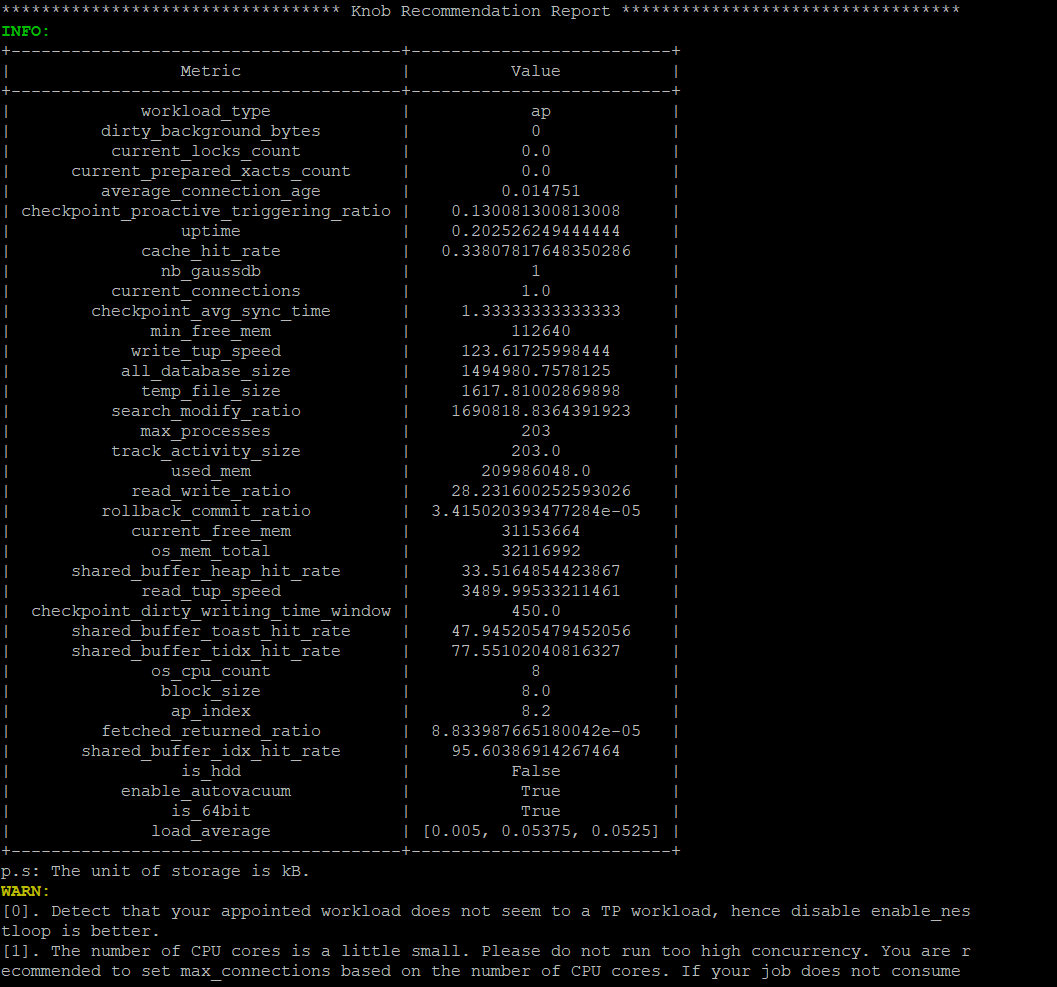
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

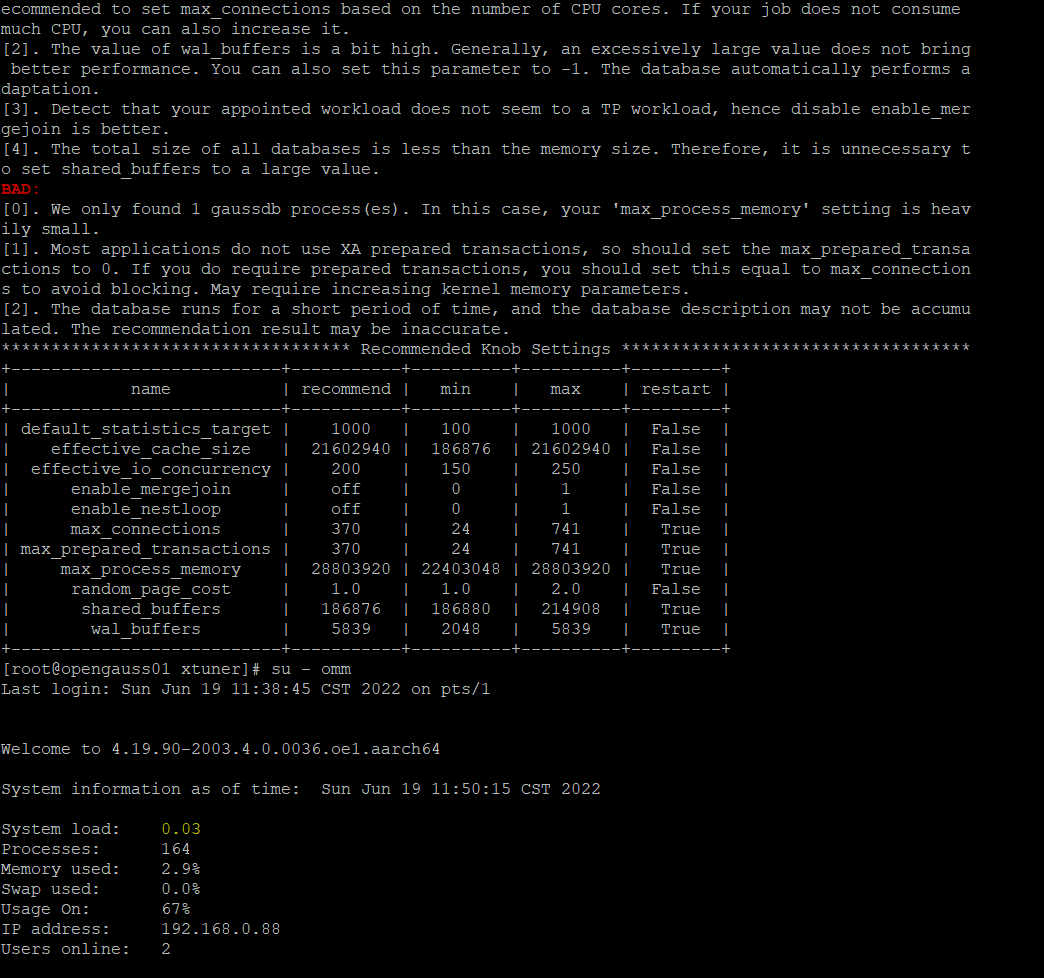
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log



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

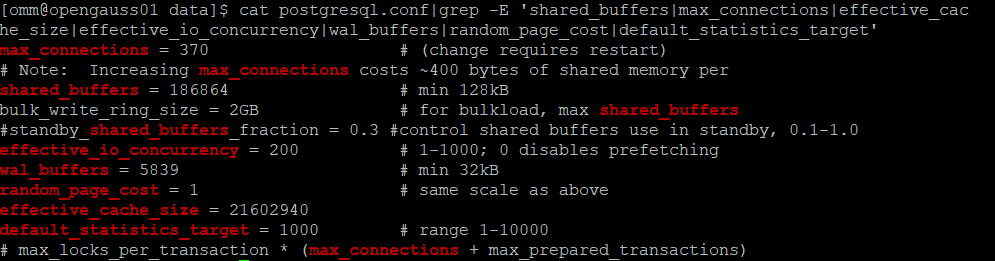




3.重启完成后，获取参数值：

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

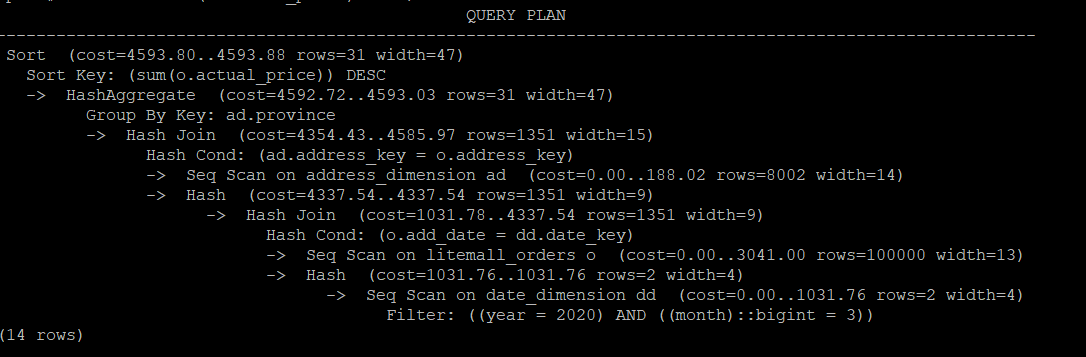
AND o.add\_date = dd.date\_key

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

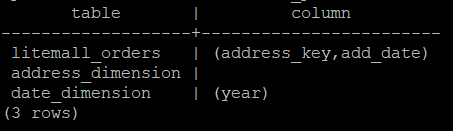
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

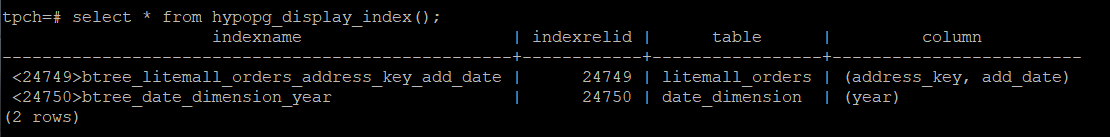
GROUP BY ad.province

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。

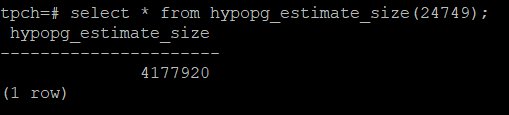
select \* from hypopg\_display\_index();

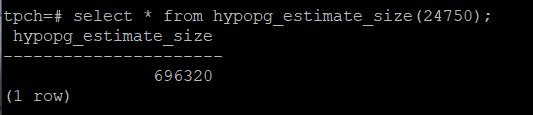


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

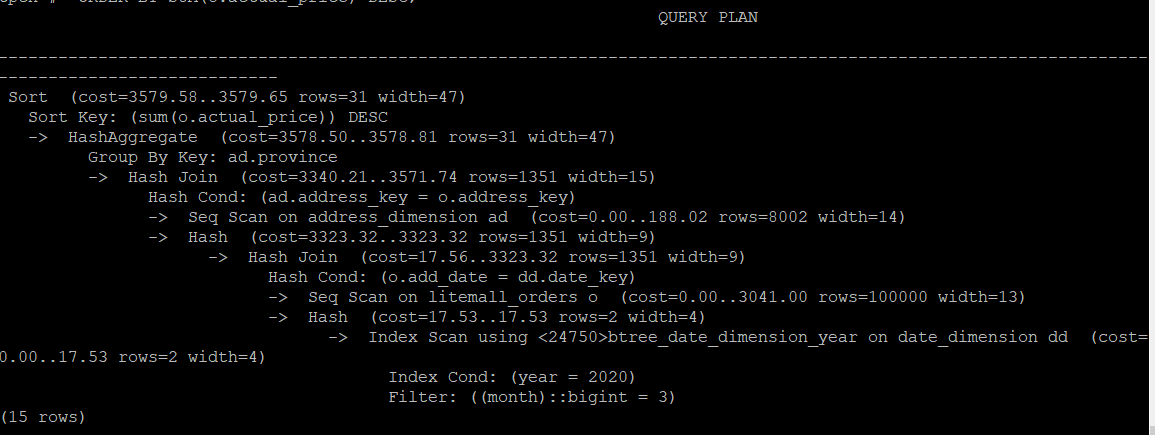
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

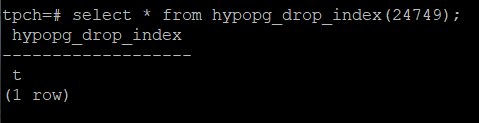
GROUP BY ad.province

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



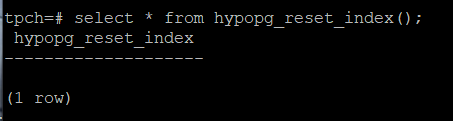
6. 删除某一个索引虚拟列，将执行结果截图。

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



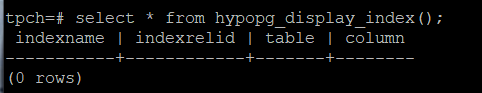
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

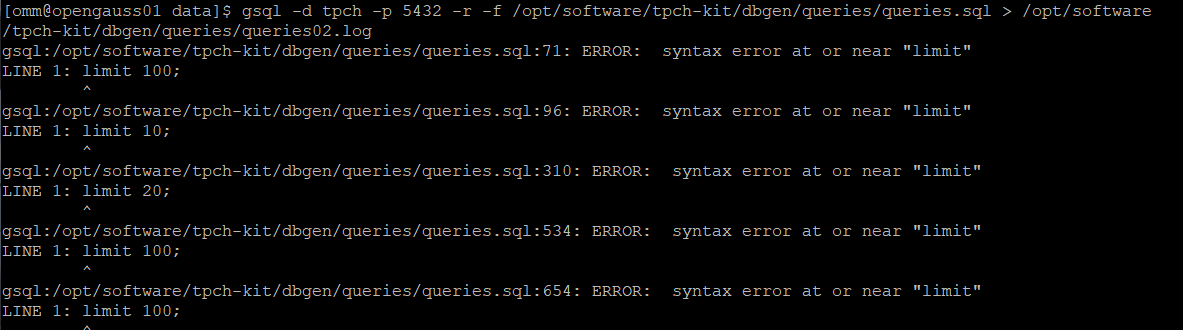
select \* from hypopg\_display\_index();

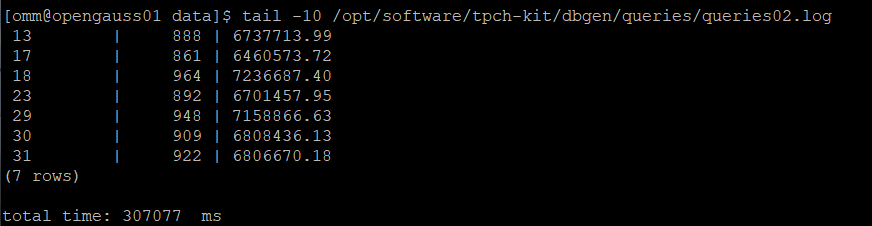


任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

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

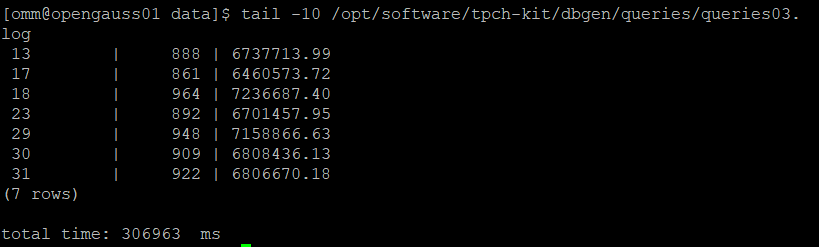




挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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？

default statistics\_target

effective\_cache\_size

effective\_io\_concurrency

enable\_mergejoin

enable\_nestloop

max\_connections

max\_prepared\_transactions

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？

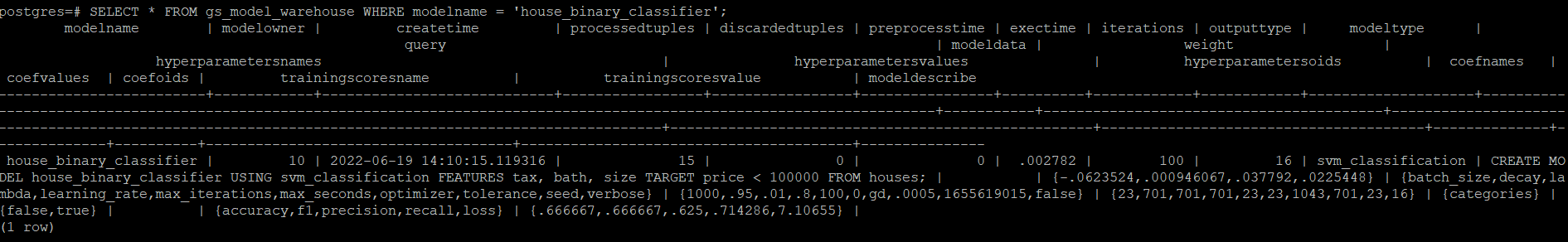
通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。可以大大加快数据的检索速度，这也是创建索引的最主要的原因。可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。在使用分组和排序 子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

优化数据库表结构的设计，减少数据库连接等。

# 关卡四、openGauss的DB4AI特性应用

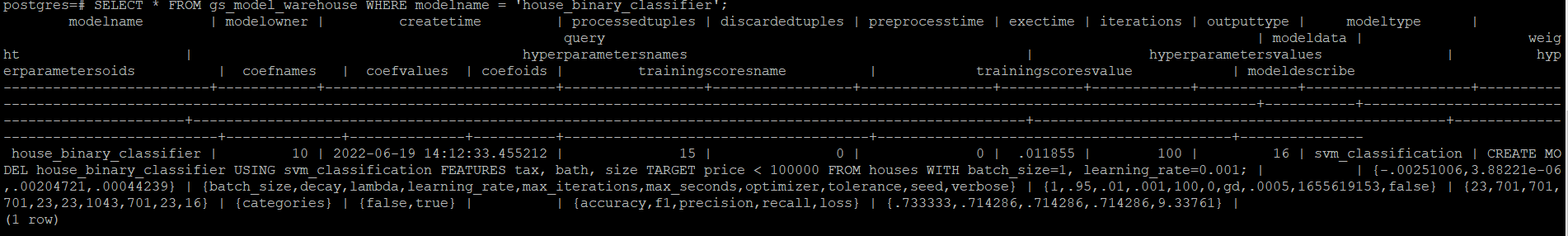
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



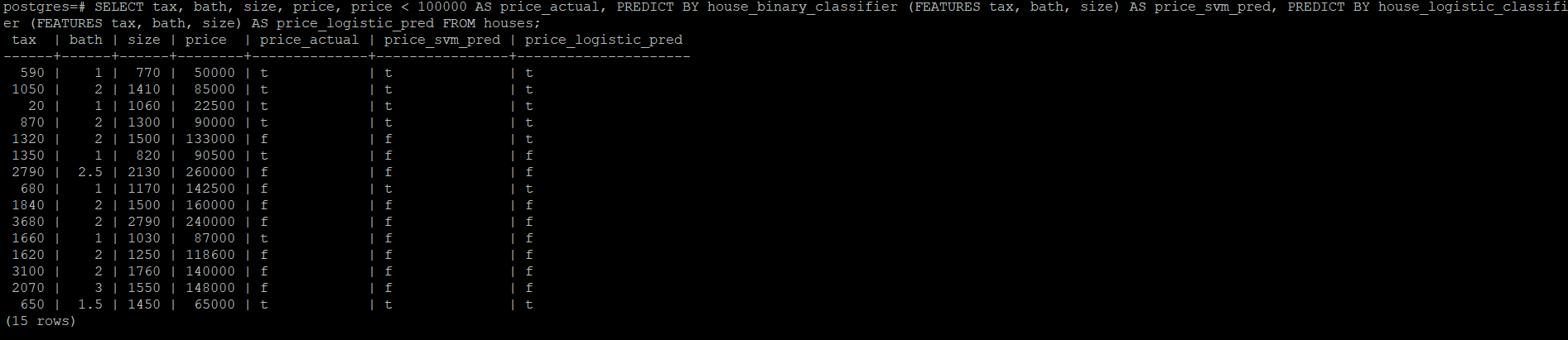
任务二：观察新模型的信息，将执行结果截图。

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 machines，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。

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：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：检索出相关文档数与检索出的文档总数的比率

召回率：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

ROC曲线、AUC曲线

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？

**SSE(误差平方和)**

**R-square(决定系数)**

**R-square(决定系数)**