openGauss AI特性创新实践课



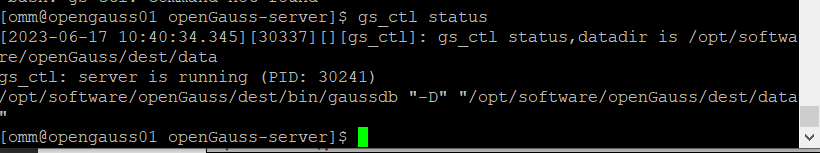
华为技术有限公司

# 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：

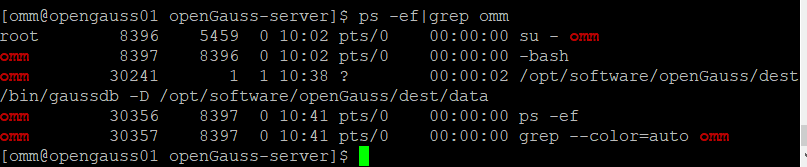
任务一：数据库状态验证

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？

①满足不同的运行平台，我们[Linux](https://so.csdn.net/so/search?q=Linux&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_37392757/article/details/_blank)发型版本众多，但是每个版本采用的软件或者内核版本都不一样，而我们的二进制包所依赖的环境不一定能够正常运行，所以大部分软件直接提供源码

②方便定制，满足不同的需求

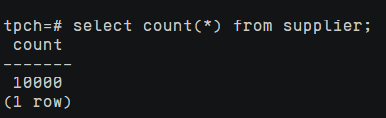
③方便运维、开发人员维护，源码是可以打包二进制的，但是对于这个软件的打包都会有一份代价不小的额外工作，包括维护

# 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：

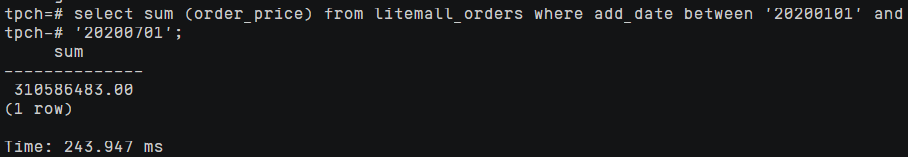
select count(\*) from supplier;;



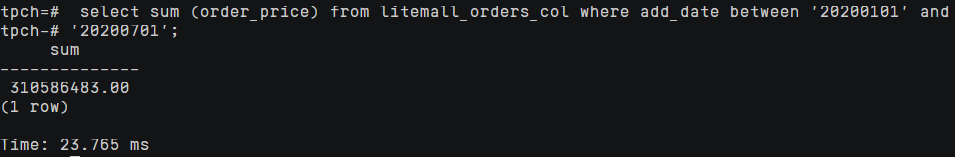
任务二：行存表与列存表执行效率对比

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';

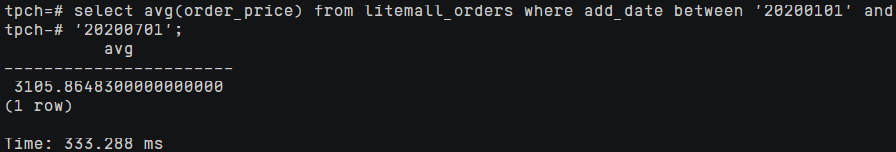


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

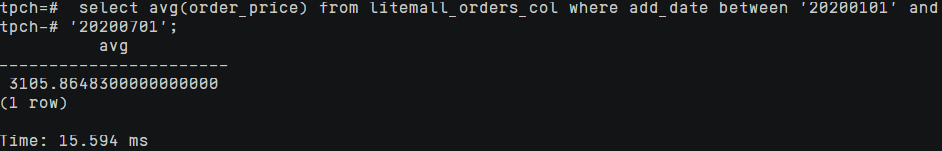


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

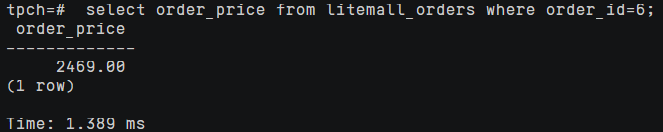


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

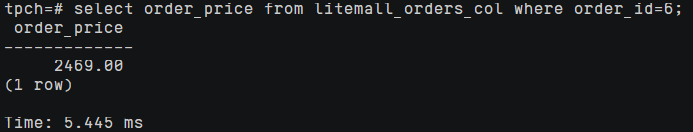


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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;

2.8

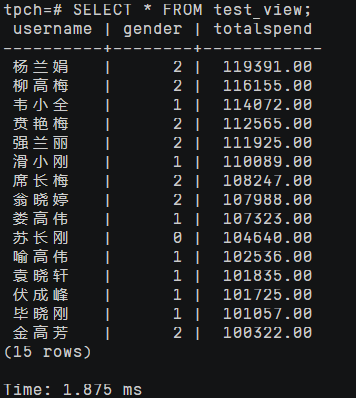
update litemall\_orders\_col set order\_price=2468 where order\_id=6;

2.9

任务三：物化视图的使用

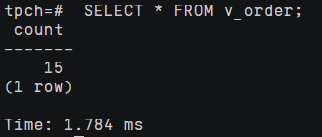
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



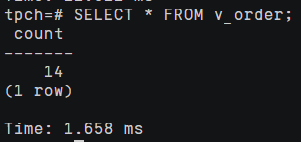
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



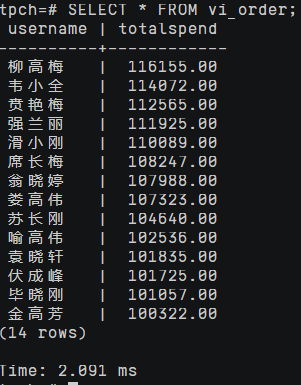
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



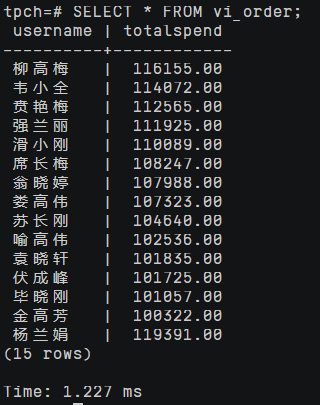
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

①行存储的写入是一次完成，列存储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，实际时间消耗会更大。

②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;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。

③一个OLAP类型的查询可能需要访问几百万甚至几十亿个数据行，且该查询往往只关心少数几个数据列，这个时候就需要列存表。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

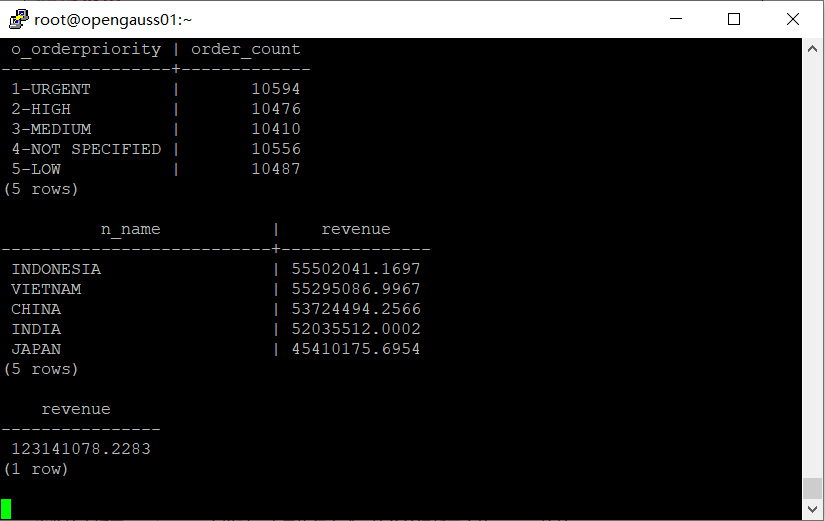
全量物化视图仅支持对已创建的物化视图进行全量更新，而不支持进行增量更新，增量物化视图可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据刷新。与全量创建物化视图的不同在于目前增量物化视图所支持场景较小。

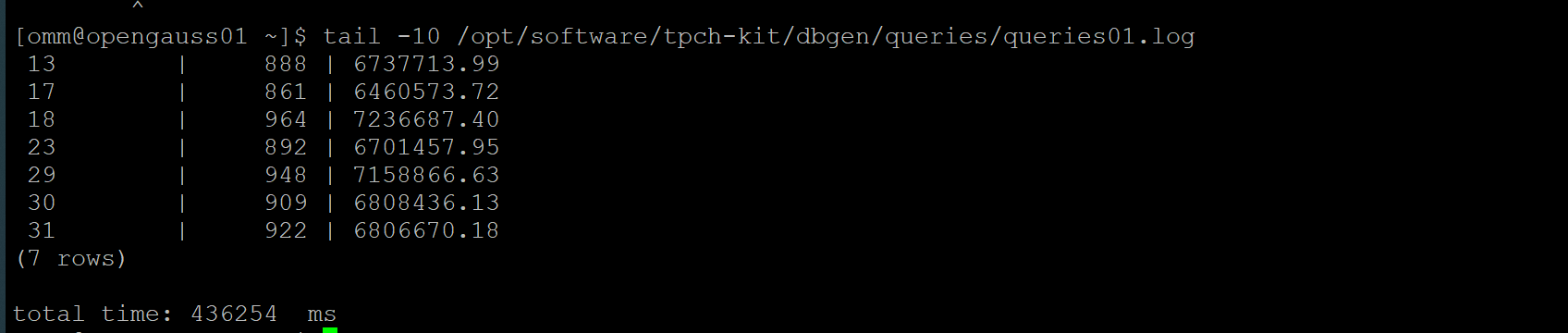
# 关卡三、openGauss的AI4DB特性应用

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：

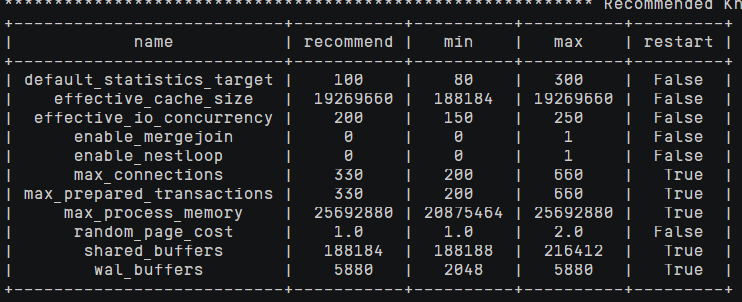
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log





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

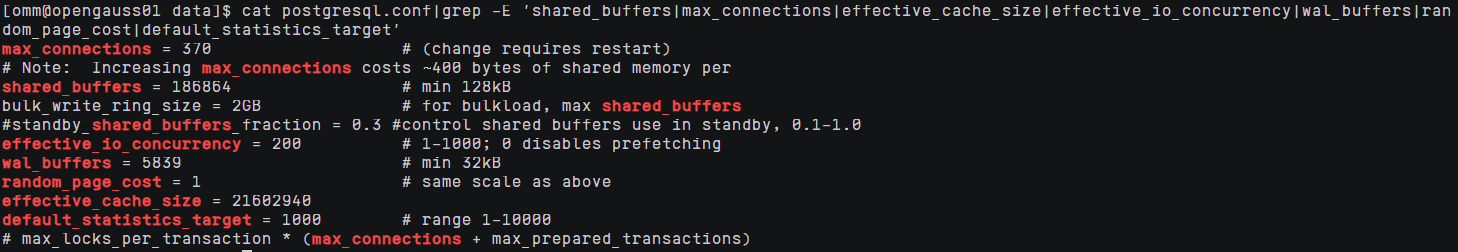
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

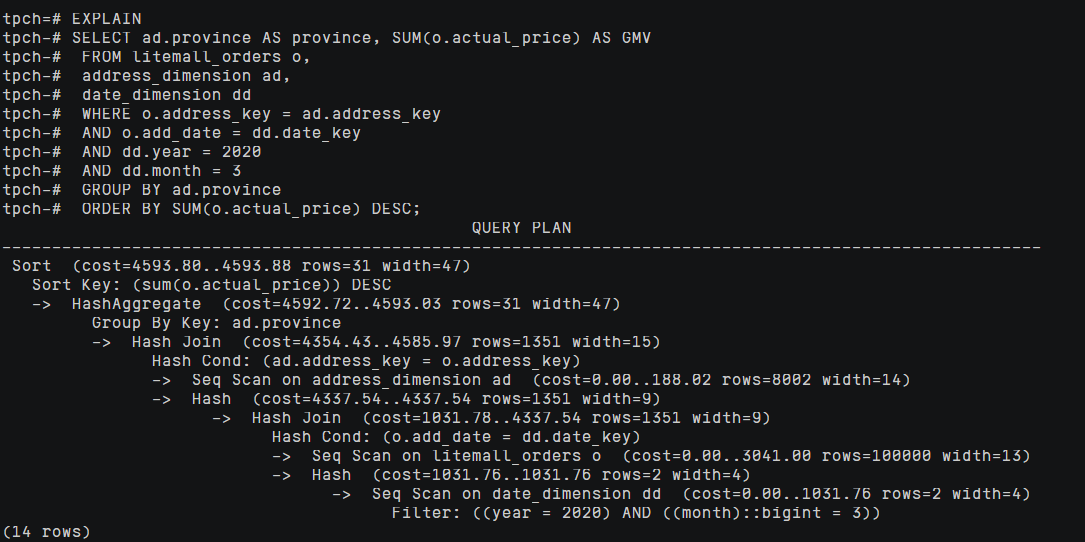
AND o.add\_date = dd.date\_key

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

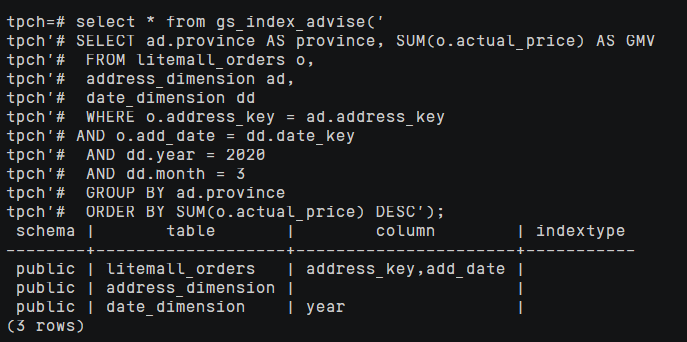
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

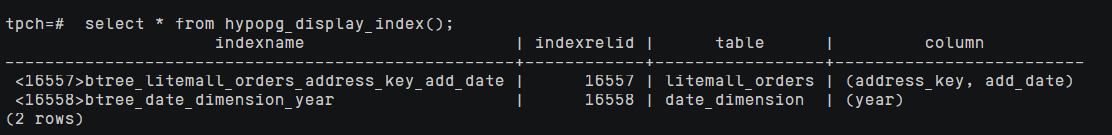
GROUP BY ad.province

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。

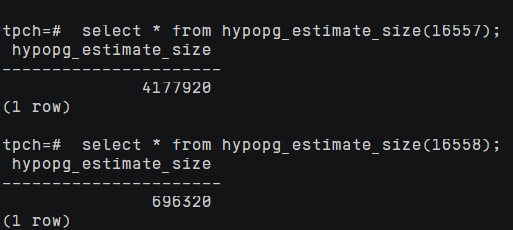
select \* from hypopg\_display\_index();



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

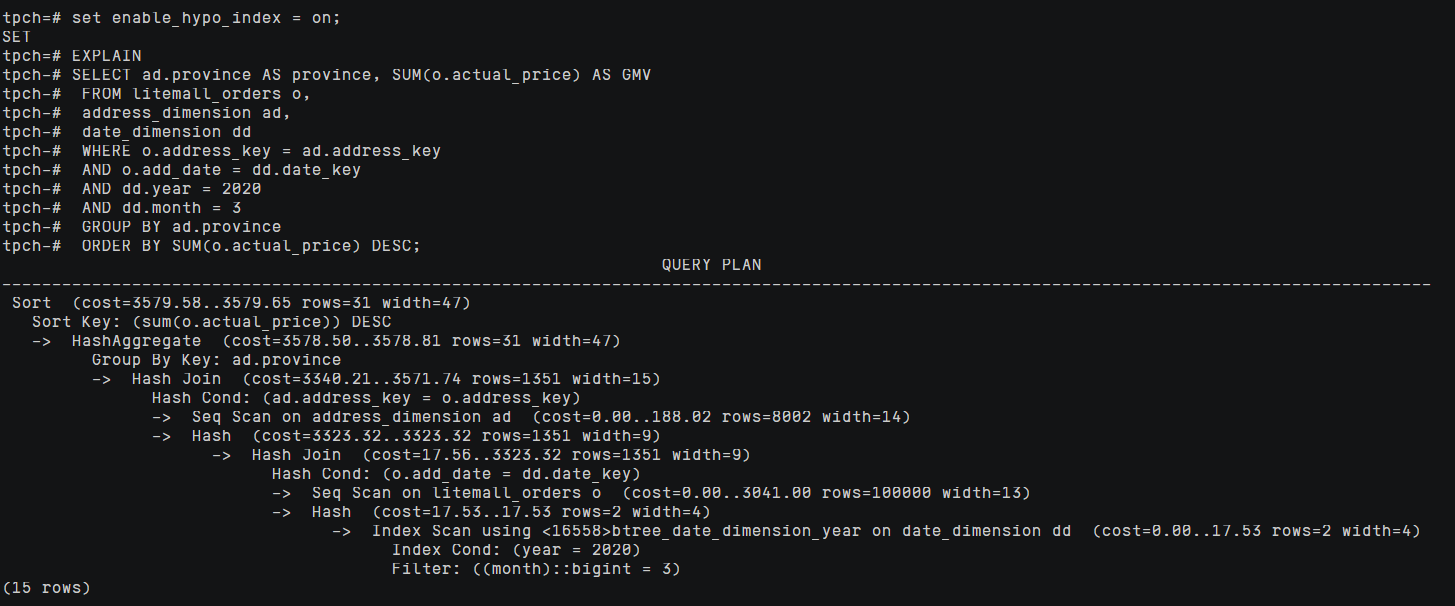
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

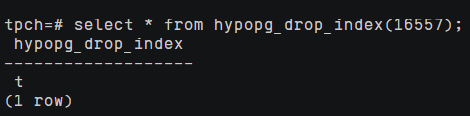
GROUP BY ad.province

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



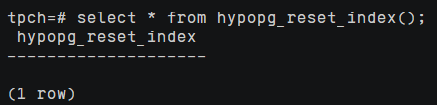
6. 删除某一个索引虚拟列，将执行结果截图。

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



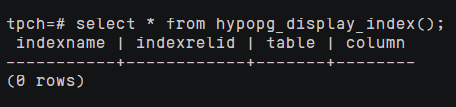
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

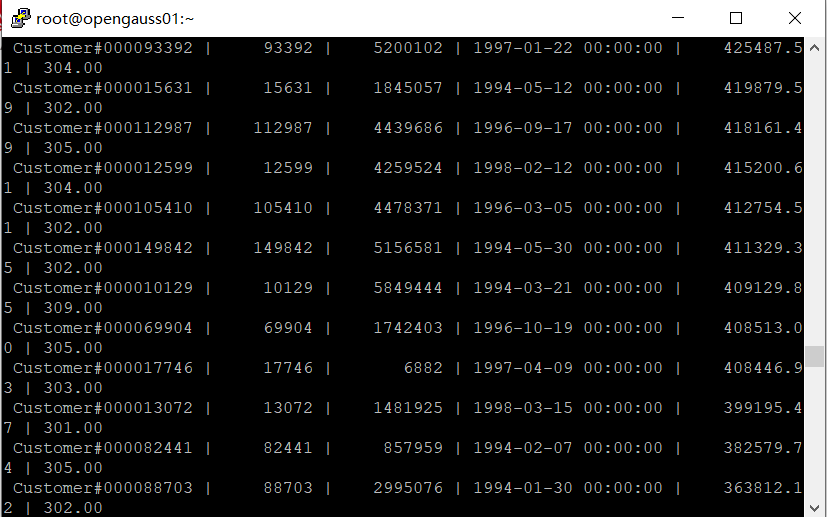
select \* from hypopg\_display\_index();

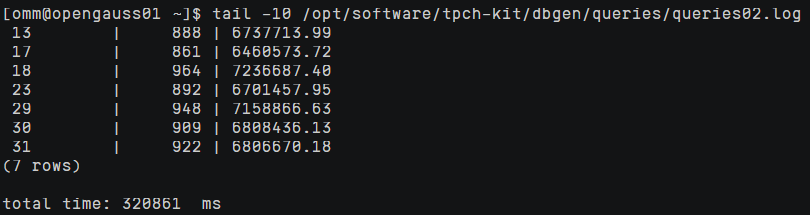


任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

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

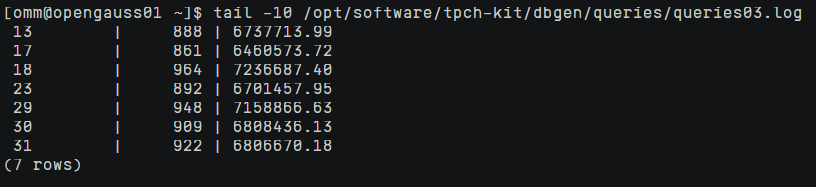




挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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？

优化的参数有：shared\_buffers、max\_connections、effective\_cache\_size、effective\_io\_concurrency、wal\_buffers、random\_page\_cost、default\_statistics\_target

因为这些参数在sql执行过程中多次被调用，对这些参数优化后，通过虚拟索引执行sql语句可以再度提高执行速度和执行效率。

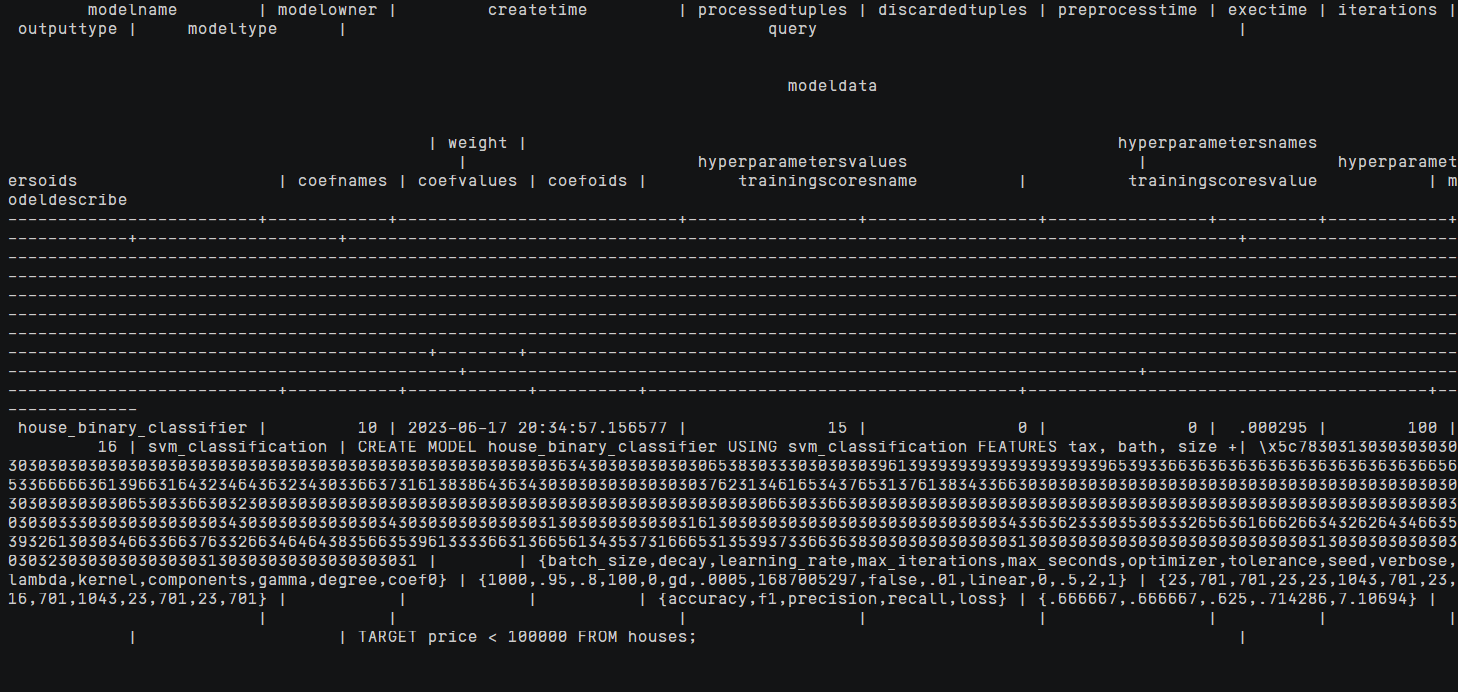
实践思考题2：索引的使用，对于执行SQL有什么好处？除了使用索引和参数外，还有哪些方面可以对数据库进行优化？

使用索引的好处主要在于可以提高SQL的查询执行速度；除了使用索引和参数外，还可以通过优化SQL语句或者优化数据库表结构来对数据库进行优化。

# 关卡四、openGauss的DB4AI特性应用

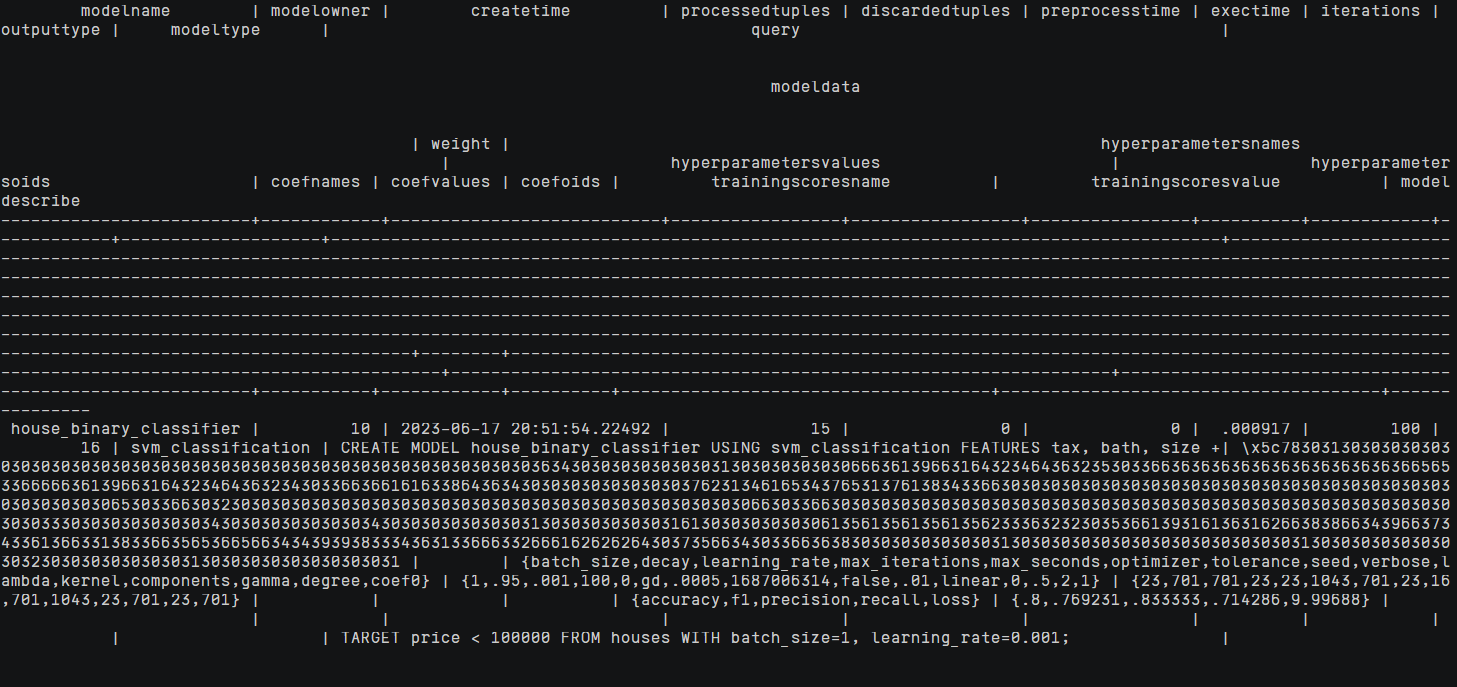
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



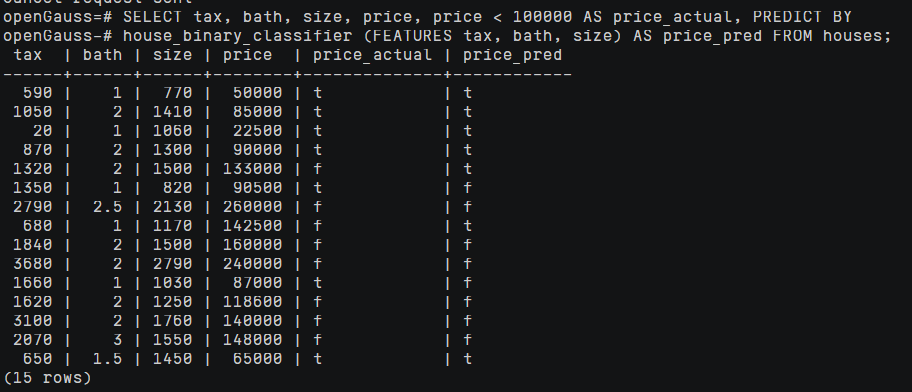
任务二：观察新模型的信息，将执行结果截图。

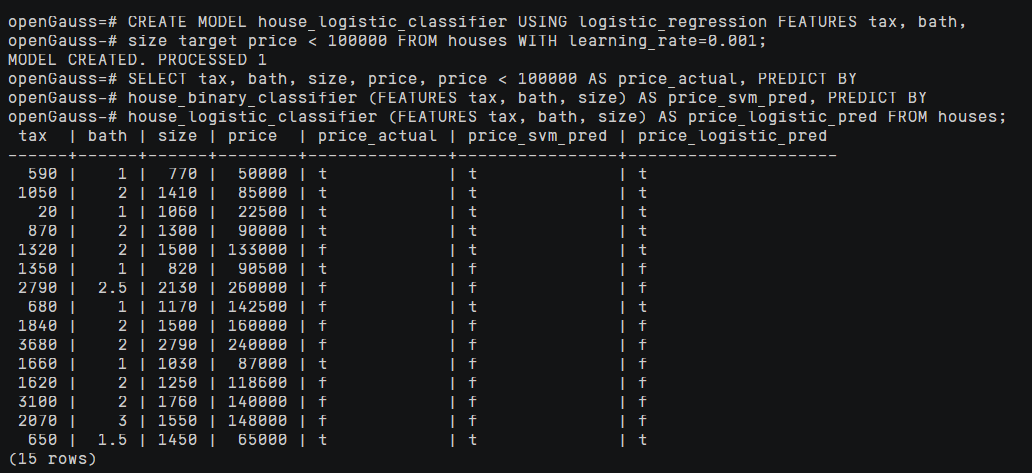
postgres=# SELECT \* FROM gs\_model\_warehouse WHERE modelname = 'house\_binary\_classifier';



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？

定性输出称为分类，或者说是离散变量预测。

定量输出称为回归，或者说是连续变量预测；

实践思考题2：什么是SVM算法？

SVM算法是一种用于分类和回归的监督学习算法。目的是在特征空间中找到间隔最大的超平面。

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）:对于给定的数据，分类正确的样本数占总样本数的比例;

2. 精确率（Precision）: 预测为正类的样本中真正类所占的比例;

3. 查全率（召回率 Recall）: 在所有的正类中被预测为正类的比例;

4. F1-Score: 精确率和召回率的调和平均

5. ROC曲线和AUC:

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。

AUC(Area Under Curve)：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。AUC的物理意义为，任取一对（正、负）样本，正样本的Score大于负样本的Score的概率。

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. MAE：平均绝对误差
2. MSE：均方误差
3. RMSE：均方根误差