openGauss AI特性创新实践课



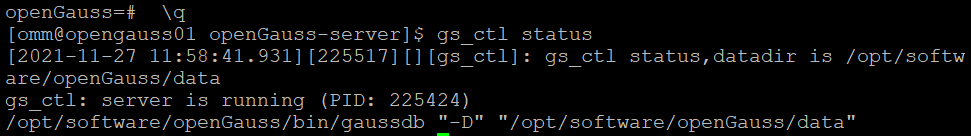
华为技术有限公司

# 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：

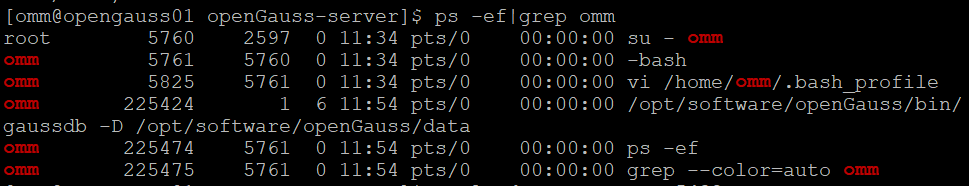
任务一：数据库状态验证

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？**

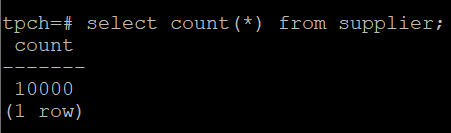
使用源码编译安装数据库，可定制。

# 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：

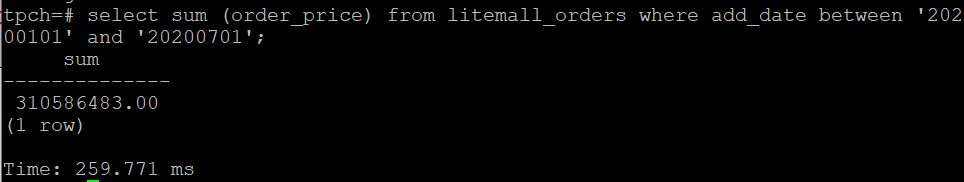
select count(\*) from supplier;;



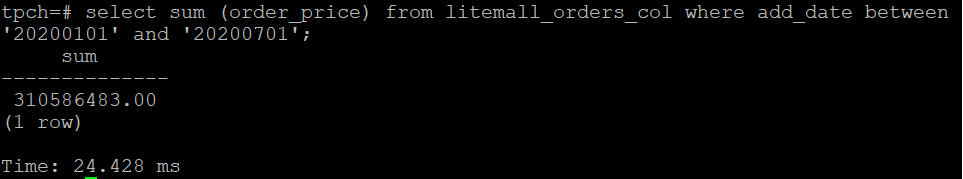
任务二：行存表与列存表执行效率对比

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';

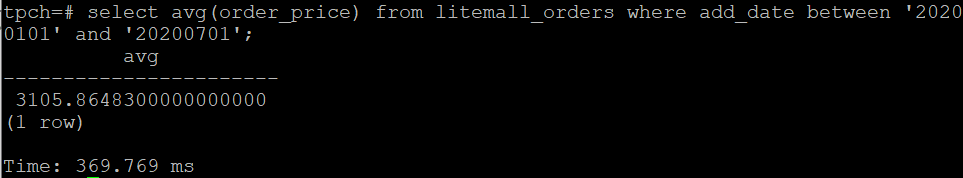


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

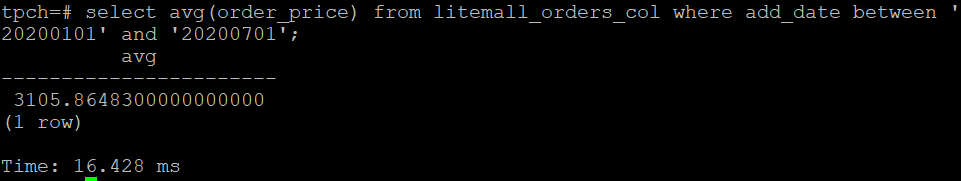


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

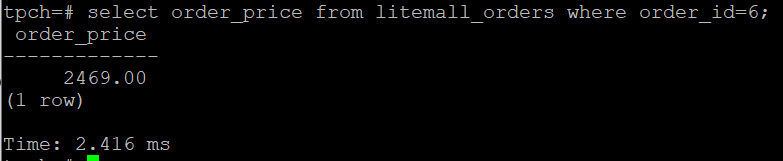


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

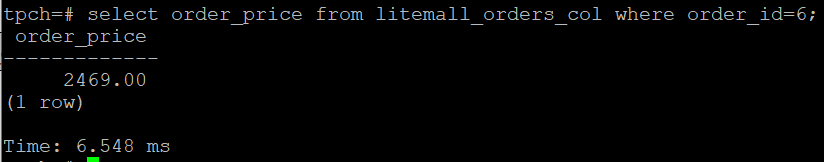


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

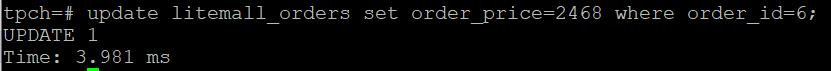


select order\_price from litemall\_orders\_col where order\_id=6;

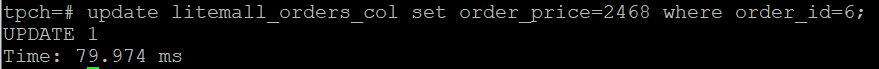


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



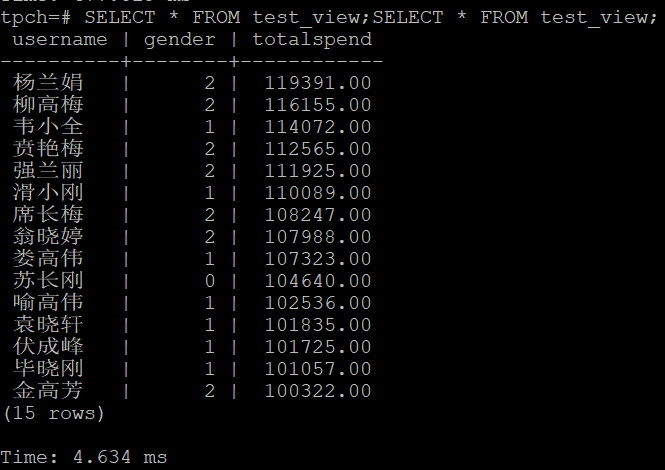
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

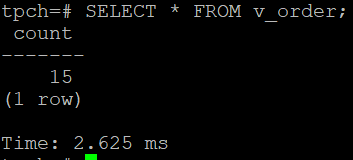
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



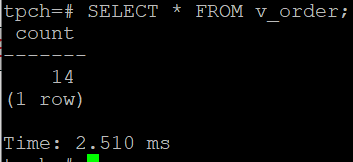
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



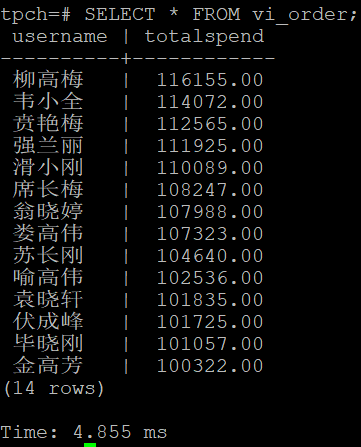
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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？**

|  |  |  |
| --- | --- | --- |
| 序号 | name | age |
| 1 | a | 1 |
| 2 | b | 2 |

行存：(1,a,1)(2,b,2)…，默认形式；列存：(1,2)(a,b),(1,2)，同一个括号里，数据类型相同，性能提升，适合于大数据

①行存表与列存表的数据存储结构不同，列存表元素的数据类型相同，因此在执行相同的SQL语句时，执行的时间不同。

②当需要关注整张表的内容，或者需要经常更新数据（因为行存表的写入是一次性完成的，消耗时间比列存表少，且能保证数据的完整性），需要经常读取整行数据而不需要聚集运算，又或在快速查询、数据表本身数据行并不多的情况下，行存表效率更高。

③当需要经常关注一张表的某几列而非整张表的数据，或者基于一列或比较少的列进行计算，又或在数据表拥有非常多的列，数据表有非常多行数据并且需要进行聚集计算，数据表列里有非常多的重复数据时（因为在读取过程中不会产生冗余数据），列存表效率更高，有利于高度压缩，适用于对数据完整性要求不高的大数据处理领域，例互联网。

**实践思考题2：全量物化视图与增量物化视图有哪些差别？**

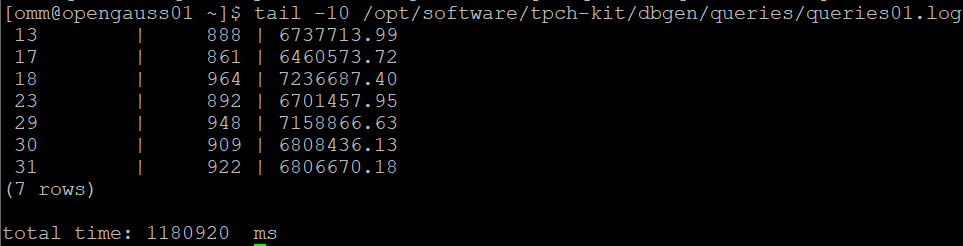
全量物化视图与增量物化视图的刷新方式不同。全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新，即所有参数都传给后台，全部入库。增量更新是在进行更新操作时，只更新需要改变的地方，不需要更新或者已经更新过的地方则不会重复更新，即传一部分参数，然后更新传递的参数，相对于全量物化视图更快捷，处理量少。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

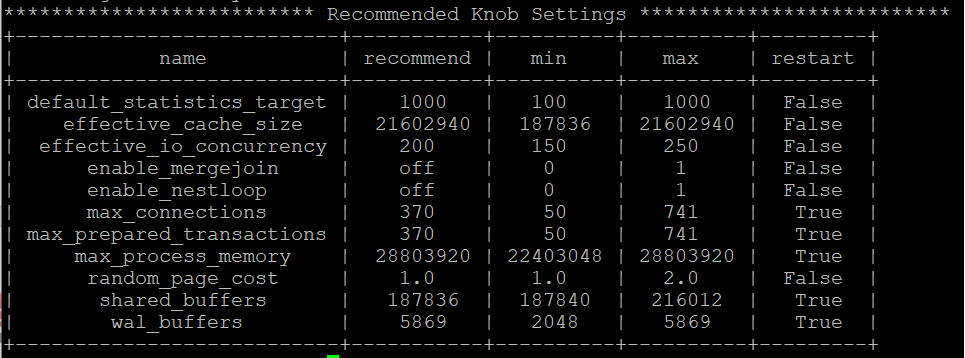
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

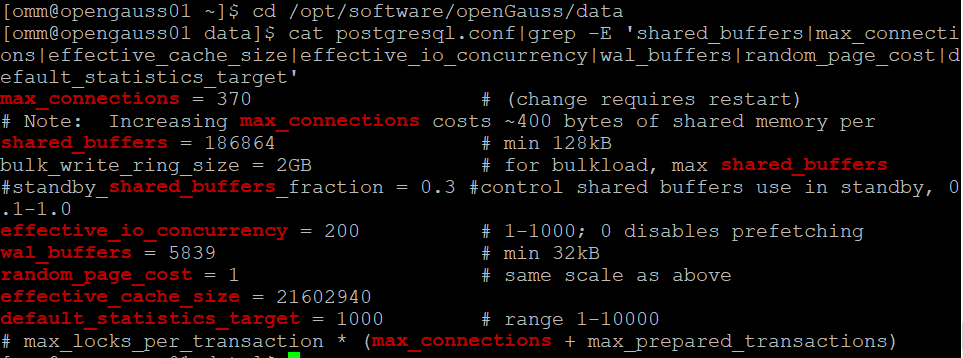
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

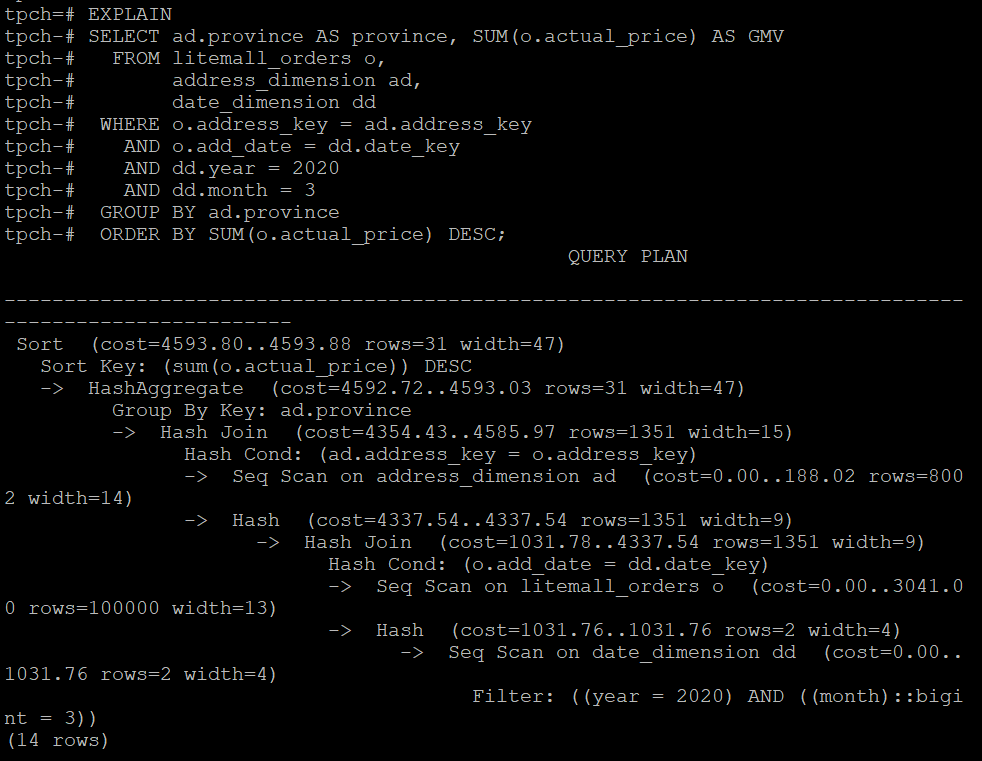
AND o.add\_date = dd.date\_key

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

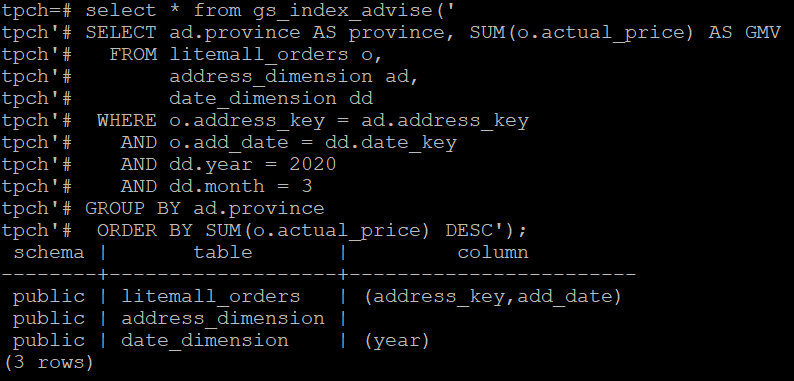
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

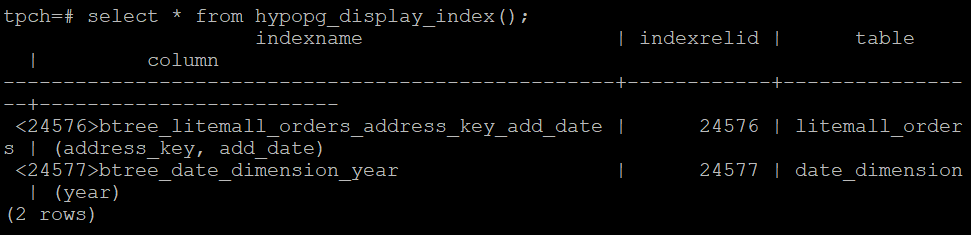
GROUP BY ad.province

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。

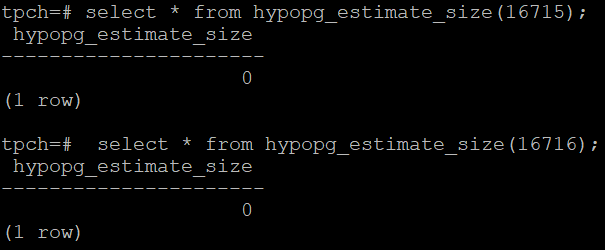
select \* from hypopg\_display\_index();



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

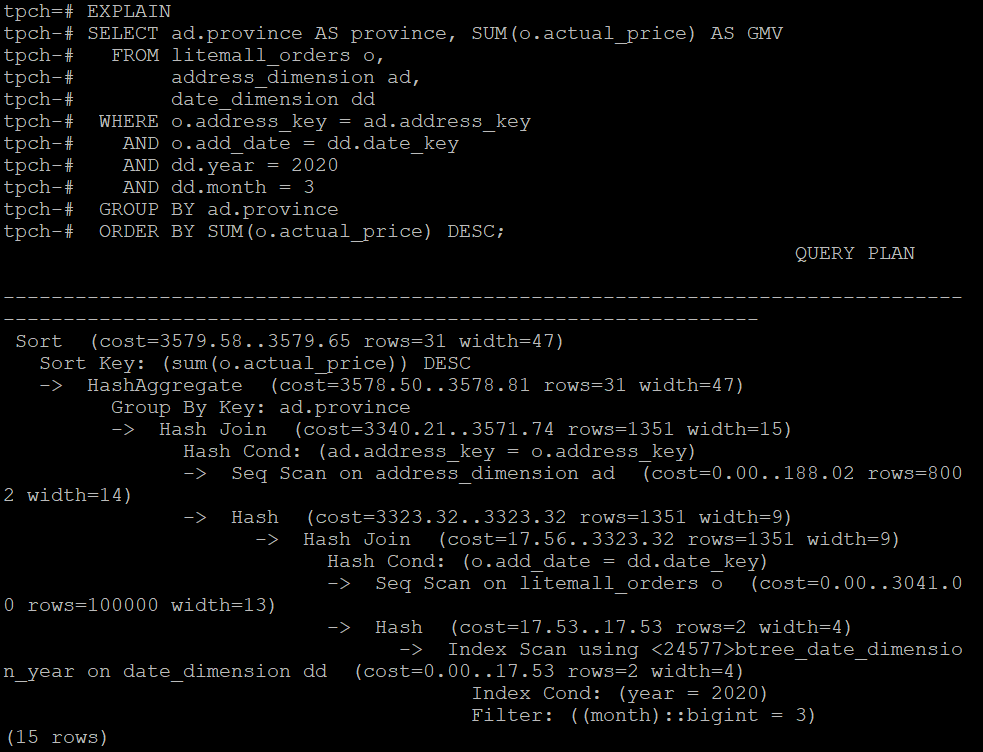
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

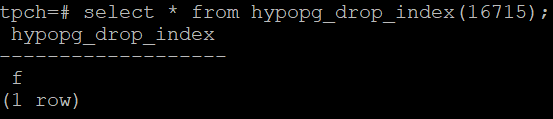
GROUP BY ad.province

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



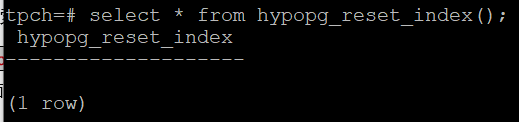
6. 删除某一个索引虚拟列，将执行结果截图。

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



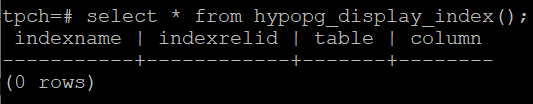
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

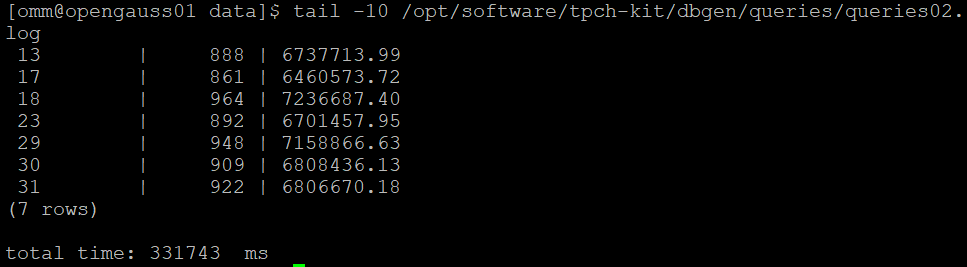
select \* from hypopg\_display\_index();



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？**

①X-Tuner对以下参数进行了优化：

shared\_buffers、max\_connections、effective\_cache\_size、effective\_io\_concurrency、wal\_buffers、random\_page\_cost、default\_statistics\_target

②对这些参数进行优化的原因：

这些参数中的很多都与数据库的表现密切相关，对这些参数进行优化，可以提高数据库在某些方面的执行效率，从而满足用户对性能的期望，保障数据库系统的稳定性。

**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？**

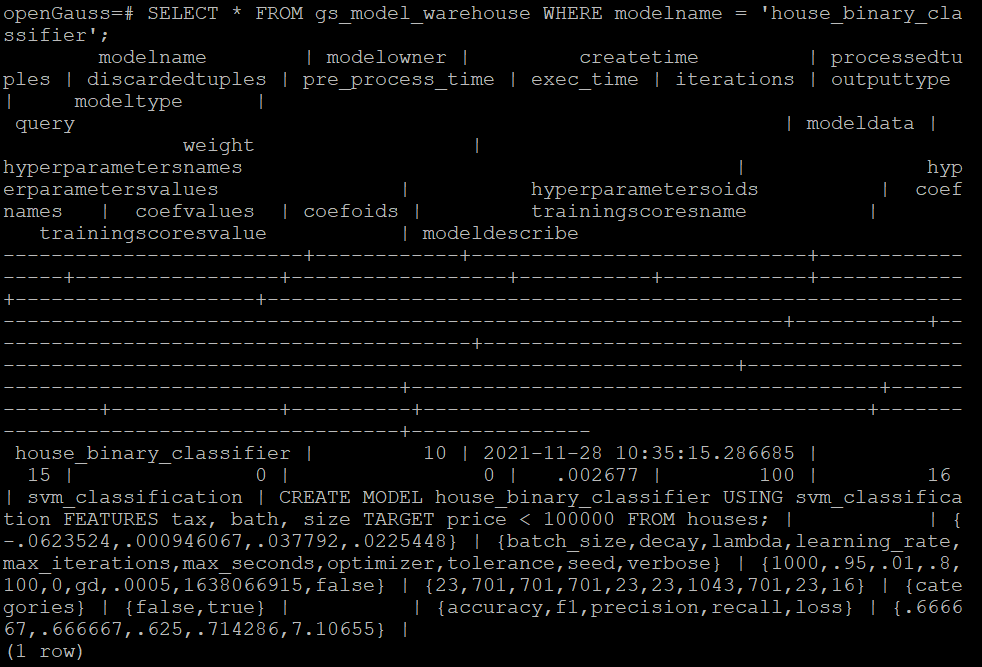
①由于索引可以加速表与表之间的连接，在实现数据的参考完整性方面上作用显著，大大加快数据的检索速度等，因此使用索引可以在一定程度上减少执行SQL的时间，提高执行效率，提高系统的性能。

②可以直接使用openGuass优化器对数据库进行优化。openGauss优化器是典型的基于代价的优化 (Cost-Based Optimization，简称CBO)。在这种优化器模型下，数据库根据表的元组数、字段宽度、NULL记录比率、distinct值、MCV值、HB值等表的特征值，以及一定的代价计算模型，计算出每一个执行步骤的不同执行方式的输出元组数和执行代价(cost)，进而选出整体执行代价最小/首元组返回代价最小的执行方式进行执行。CBO优化器能够在众多计划中依据代价选出最高效的执行计划，最大限度的满足客户业务要求。

# 关卡四、openGauss的DB4AI特性应用

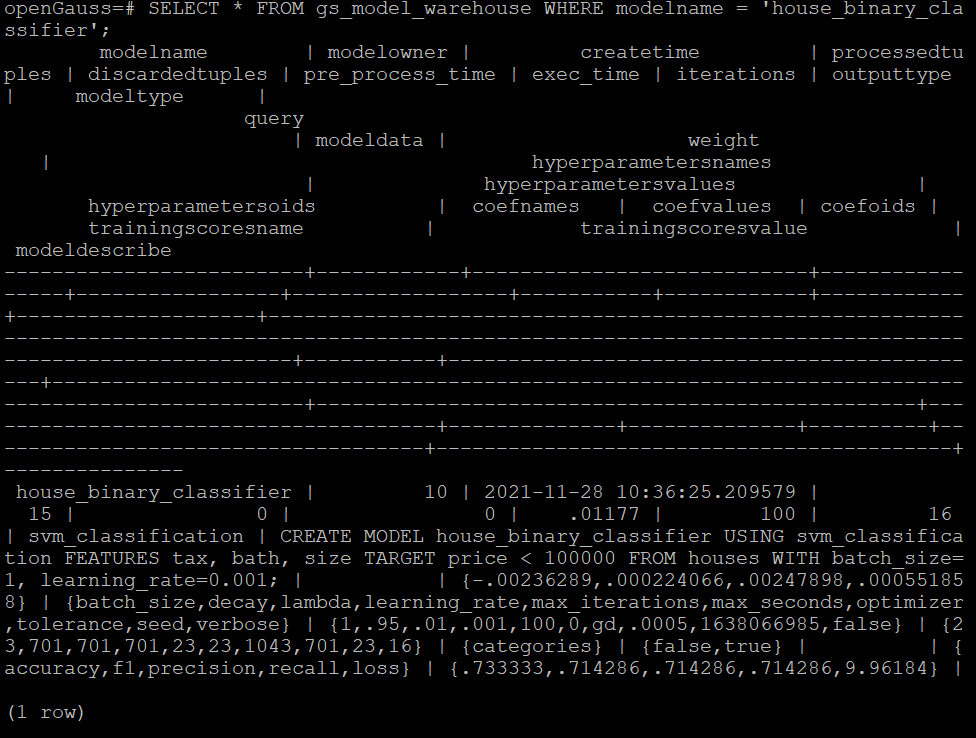
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



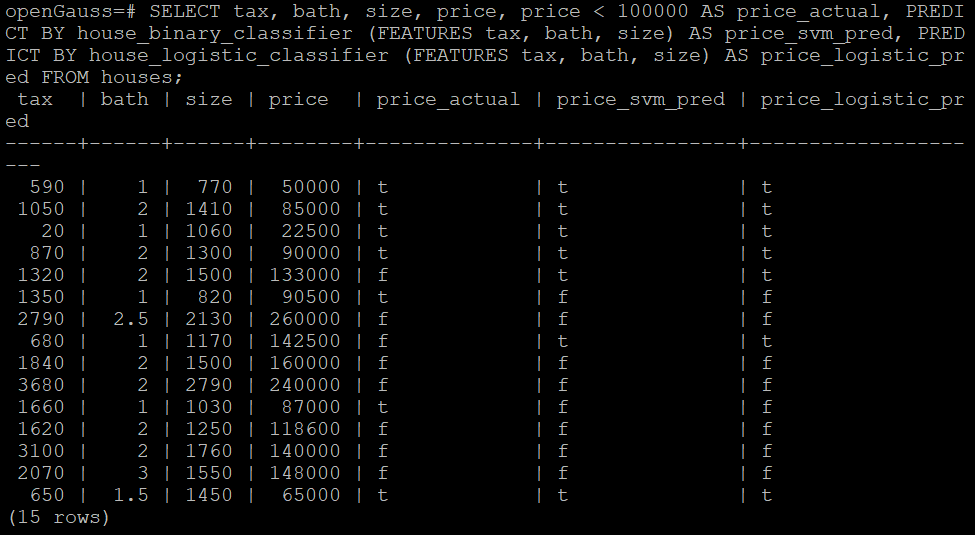
任务二：观察新模型的信息，将执行结果截图。

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？**

①分类模型是定性输出，离散变量预测，即分类模型可将回归模型的输出离散化。根据训练数据集和类标号属性，构建模型来分类现有的数据，并用来分类新数据。

②回归模型是定量输出，连续变量预测，即回归模型也可将分类模型的输出连续化，预测未来的情况。

**实践思考题2：什么是SVM算法？**

SVM（support Vector Mac）又称为支持向量机，是一种二分类的模型。当然，如果进行修改之后也是可以用于多类别问题的分类。支持向量机可以分为线性和非线性两大类。其主要思想为找到空间中的一个更够将所有数据样本划开的超平面，并且使得数据集中所有数据到这个超平面的距离最短，从而得到分界面，即求得在的优化下的超平面分界面。

**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：分类正确的样本数占总样本数的比例；

②精确率：预测正确的正样本数量占预测为正样本数量的比例；

③召回率：预测正确的正样本数量占实际为正样本数量的比例；

④ROC：以精确率作为横坐标，召回率作为纵坐标得到ROC曲线。ROC曲线越靠近左上角，该分类器的性能越好；

⑤AUC：ROC曲线下的面积。AUC越大，该分类器的分类效果越好；

**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？**

①RMSE（平方根误差）：

②MAE（平均绝对误差）：

③Coefficient of determination（决定系数）：决定系数是相关系数的平方。相关系数是用来描述两个变量之间的线性关系的，但决定系数的适用范围更广，可以用于描述非线性或者有两个及两个以上自变量的相关关系。