openGauss AI特性创新实践课



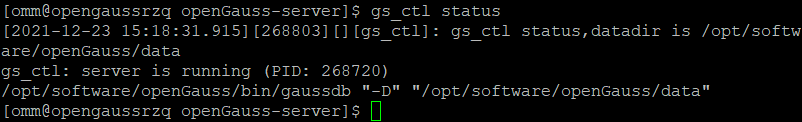
华为技术有限公司

# 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：

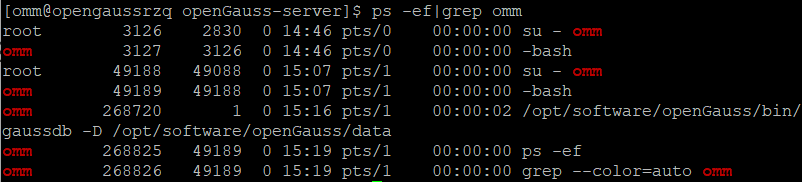
任务一：数据库状态验证

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：能够满足不同的运行平台。因为Linux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，而二进制包所依赖的环境不一定能够正常运行，所以直接提供源码编译以保证不同的运行平台都可以运行。

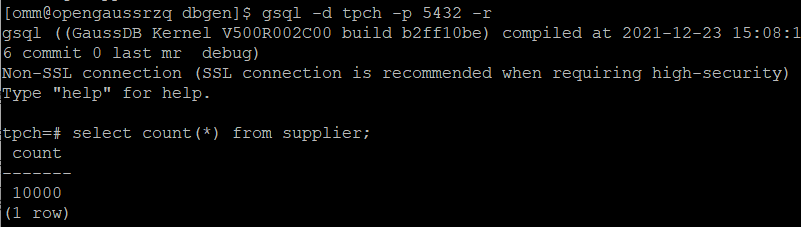
2：方便定制，满足不同的需求。很多时候我们所需要安装的软件都是可以定制的，需要什么就安装什么，而大多数二进制包都是一键装全，自由度并不高，所以源码编译可以满足我们自定义安装的需求。

# 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：

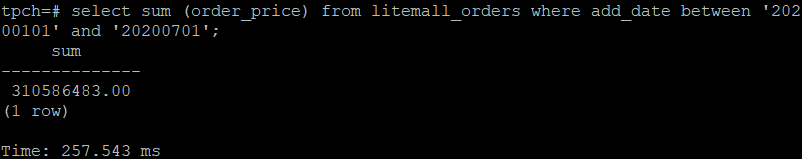
select count(\*) from supplier;;



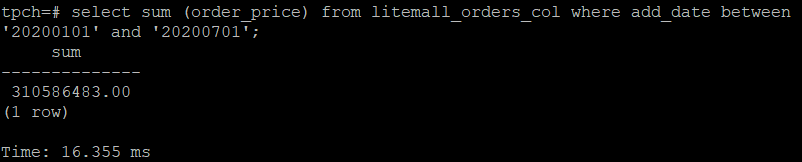
任务二：行存表与列存表执行效率对比

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';

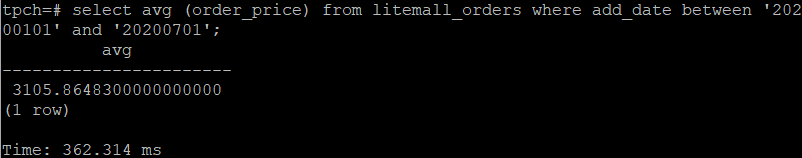


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

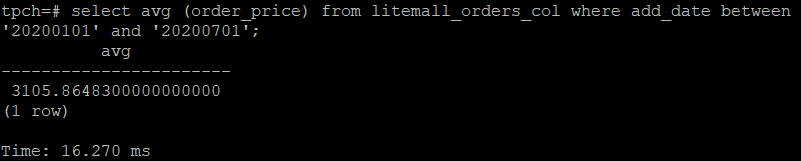


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

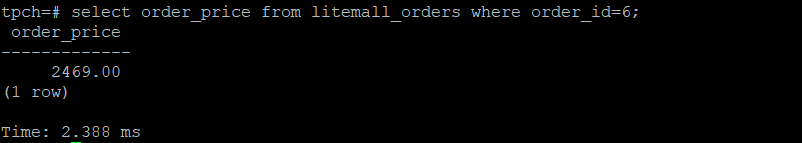


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

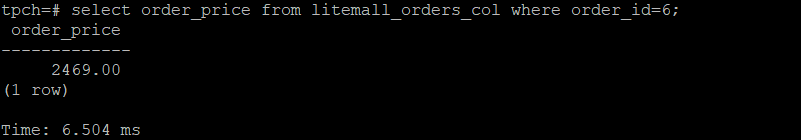


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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



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

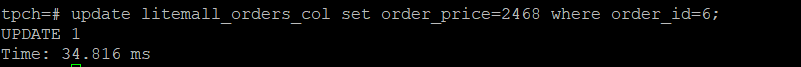


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

C:\Users\Administrator\AppData\Roaming\Tencent\Users\894968063\QQ\WinTemp\RichOle\0XB{WMPMV)DO$(_BS_$RJ2A.png

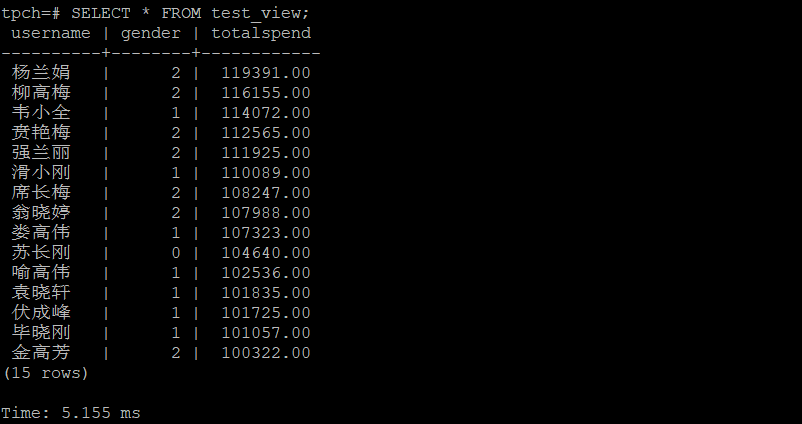
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

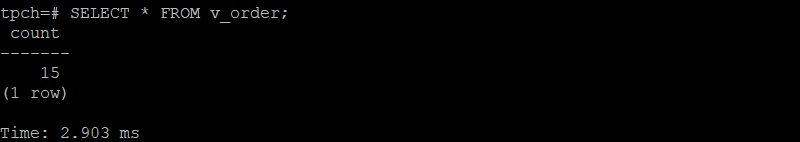
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



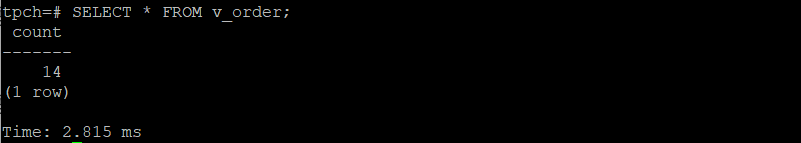
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



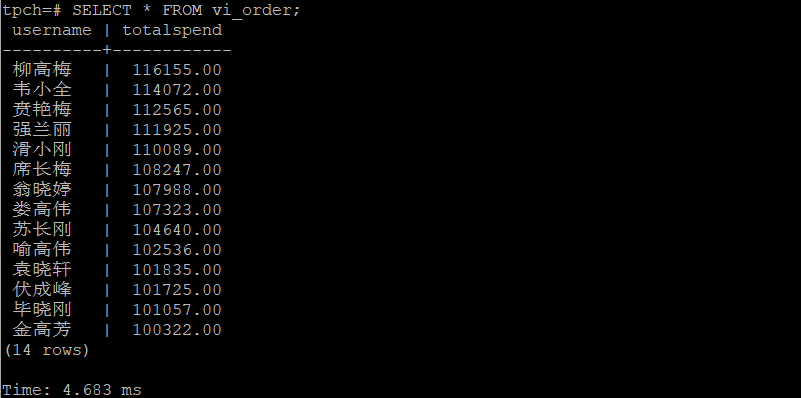
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



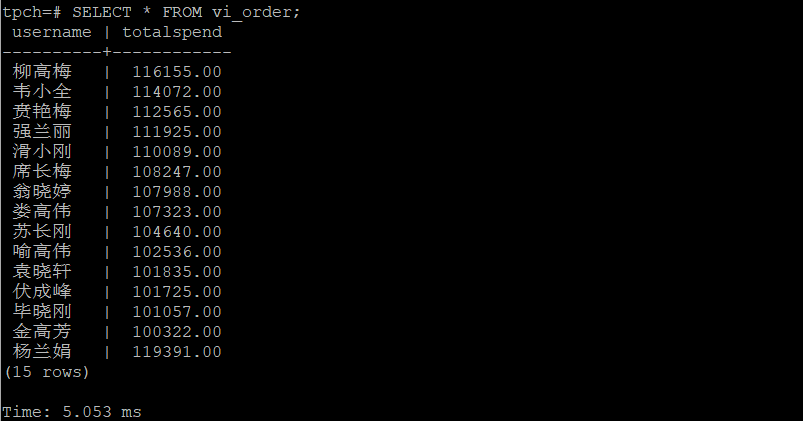
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

(1)行存表和列存表的存储数据方式不同导致了在执行相同的SQL语句时执行的时间不同。行存表是按照行为基础逻辑存储单元进行存储的（把一行中的数据值串在一起存储起来），而列存表是按照列为基础逻辑存储单元进行存储的，一列中的数据在存储介质中以连续存储形式存在（把一列中的数据值串在一起存储起来）。

(2)行存表在执行更新SQL语句和查询某个范围内的行数据的时候，行存表效率更高。

(3)列存表在执行查询对列的多行数据运算后的结果的SQL语句时，列存表效率更高

实践思考题2：全量物化视图与增量物化视图有哪些差别？

增量物化视图是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进行刷新。目前增量物化视图所支持场景较小，目前物化视图创建语句仅支持基表扫描语句或者UNION ALL语句

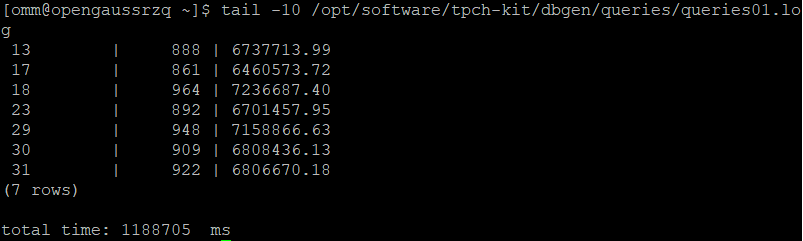
全量物化视图仅支持对创建好的物化视图做全量更新。创建全量物化视图语法和CREATE TABLE AS语法一致，不支持对全量物化视图指定NodeGroup创建。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

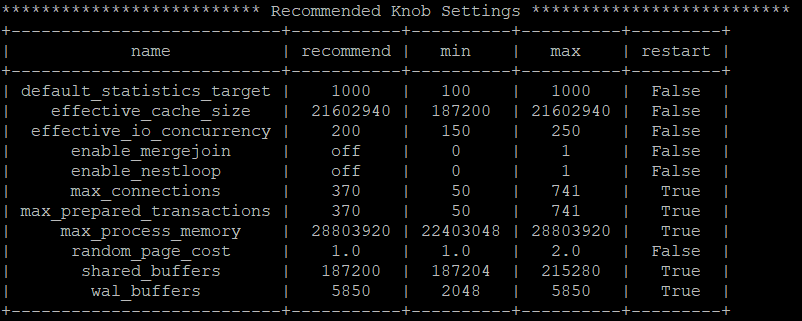
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

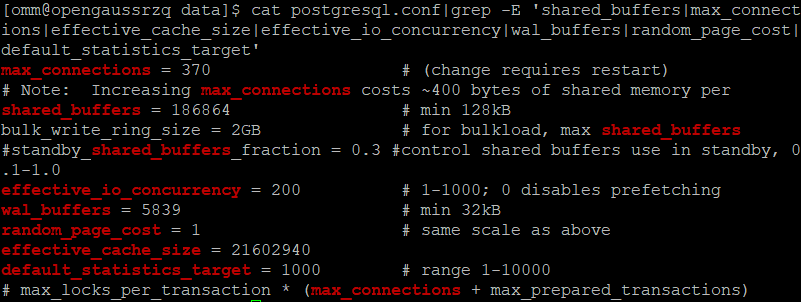
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

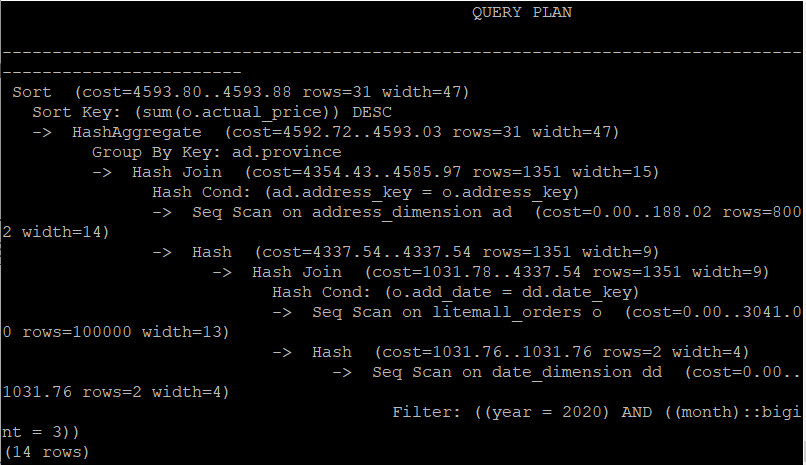
AND o.add\_date = dd.date\_key

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

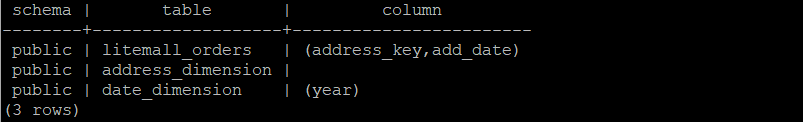
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

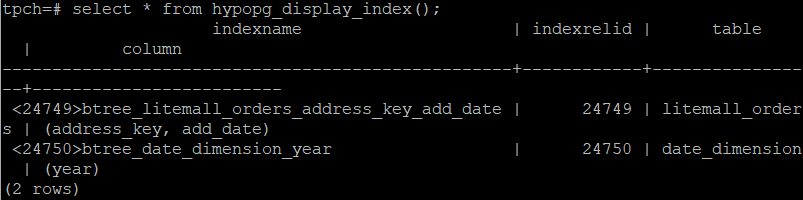
GROUP BY ad.province

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。

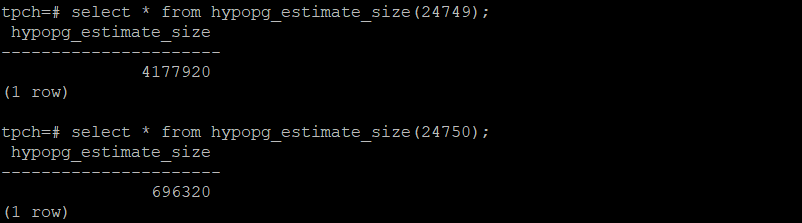
select \* from hypopg\_display\_index();



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

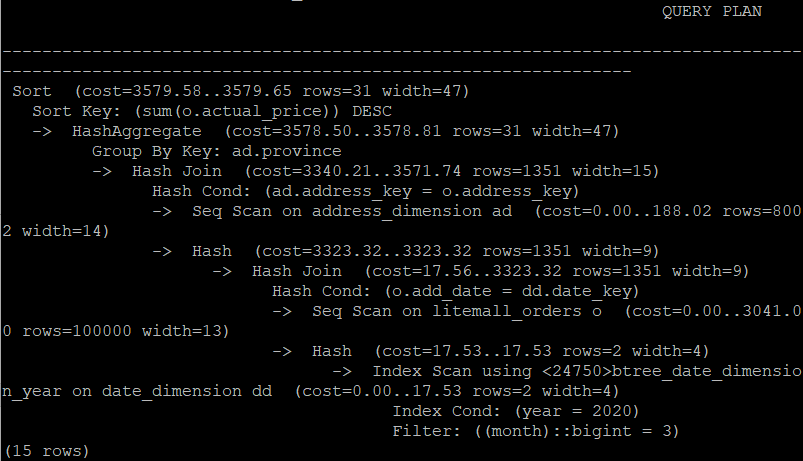
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

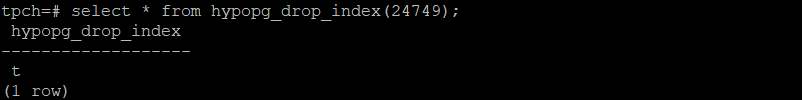
GROUP BY ad.province

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



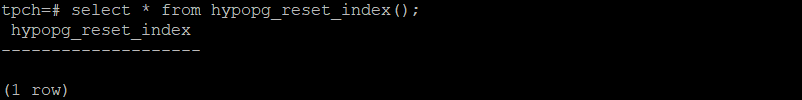
6. 删除某一个索引虚拟列，将执行结果截图。

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



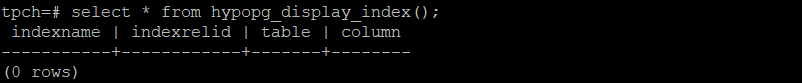
7. 删除所有索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

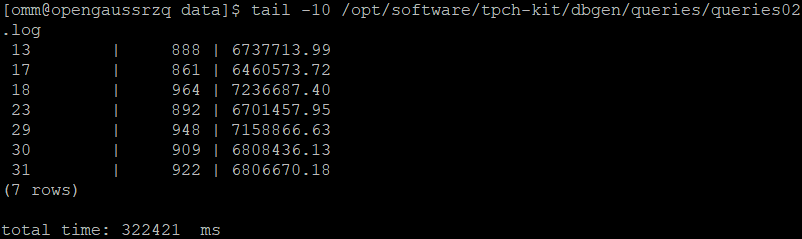
select \* from hypopg\_display\_index();



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？

我们在shared\_buffers、max\_connections、effective\_cache\_size、effective\_io\_concurrency、wal\_buffers、random\_page\_cost和default\_statistics\_target参数上进行了优化。优化这些参数的目的是为了获取最佳数据库参数配置。

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？

使用索引的好处是索引提高了SQL Server系统的性能，能够加快执行sql语句数据的查询速度与减少系统的响应时间。

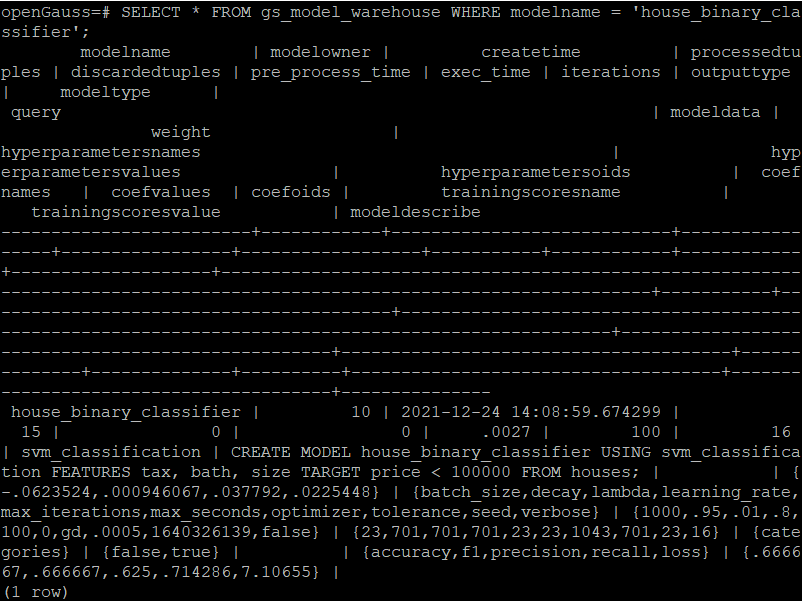
除了使用索引和参数以外，还可以通过选取最合适的的字段属性以及优化sql语句方面对数据库进行优化。

对sql语句的优化可以分为以下几个方面：避免使用子查询，可以使用JOIN链接查询替代；UNION All能满足业务需求不要使用UNION；WHERE子句尽量避免使用!=或<>操作符；WHERE子句使用OR的优化；WHERE子句使用IN或NOT IN优化；WHERE子句中使用IS NULL或IS NOT NULL优化；LIKE语句优化；LIKE语句优化；定义SQL语句字段列表替换"\*"；LIMIT分页优化；EXISTS代替IN等。

# 关卡四、openGauss的DB4AI特性应用

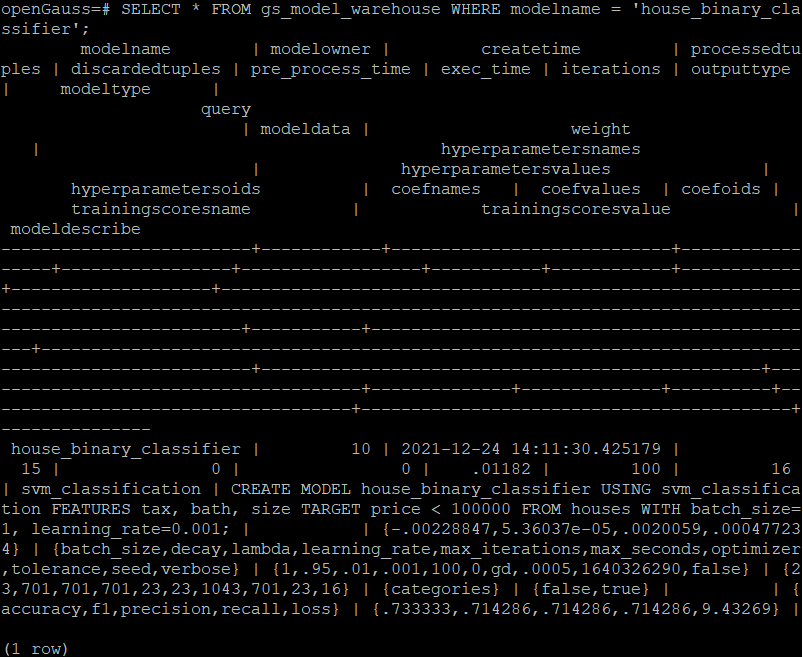
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



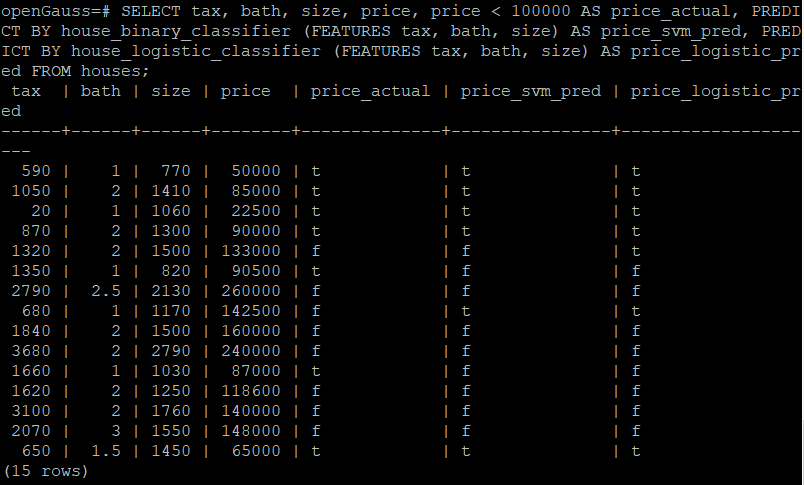
任务二：观察新模型的信息，将执行结果截图。

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。回归模型是定量输出，或者说是连续变量预测；分类模型是定性输出，或者说是离散变量预测。

实践思考题2：什么是SVM算法？

SVM是支持向量机(Support Vecor Machine)，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)混淆矩阵: 假设是二分类问题，计为正例（positive） 和 负例（negative） 分别是：

TP：真正，实际为正例且被分类器划分为正例的实例数（划分正确）。

FP：假正，实际为负例但被分类器划分为正例的实例数（划分错误）。

FN：假负，实际为正例但被分类器划分为负例的实例数（划分错误）。

TN：真负，实际为负例且被分类器划分为负例的实例数（划分正确）。

(2) 准确率：分类器分类正确的样本数量与样本总数的比；

(3) 错误率：与准确率的含义相反，分类器分类错误的样本数量与样本总数的比；

(4) 精准率：指即被分类器判定为正类样本中真正的正类样本所占的比重；

(5) 召回率：指被分类器正确判定的正类样本占总的正类样本的比重；

(6) F-score：精确率和召回率的调和均值；

(7) 宏平均：指所有类别的每一个统计指标值的算数平均值，也就是宏精确率；

(8) 微平均：对数据集中的每一个示例不分类别进行统计建立全局混淆矩阵，然后计算相应的指标；

(9) PR曲线：Precision Recall Curve的简称，描述的是precision和recall之间的关系，以recall为横坐标，precision为纵坐标绘制的曲线；

(10) ROC曲线：受试者工作特征（Receiver Operating Characteristic）曲线，以灵敏度（真阳性率）为纵坐标，以1减去特异性（假阳性率）为横坐标绘制的性能评价曲线；

(11) AUC就是ROC曲线下的面积，即ROC的积分。衡量学习器优劣的一种性能指标。

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) MSE：均方误差，用 真实值-预测值 然后平方之后求和平均；

(2) RMSE：均方根误差，预测过程中预测错误的标准偏差；

(3) MAE：平均绝对误差，对绝对误差损失的预期值；

(4) R-Squared：拟合优度，反映的是自变量x对因变量y的变动的解释的程度。