openGauss AI特性创新实践课



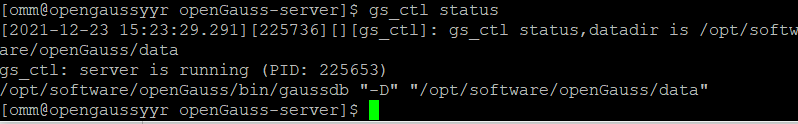
华为技术有限公司

# 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：

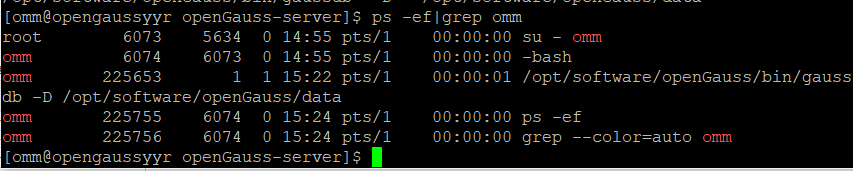
任务一：数据库状态验证

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？简单描述安装数据库所需要的步骤。

使用源码编译的安装过程，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，没有烦人的依赖关系。

安装数据库需要的步骤：

下载opengauss的源码和编译库后

1.切换omm用户

2.设置用户环境变量

3.添加环境变量

4.进去opengauss源码下生成配置文件

5.编译

6.安装opengauss

7.初始化数据库

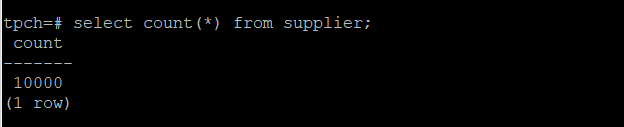
8.启动

# 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：

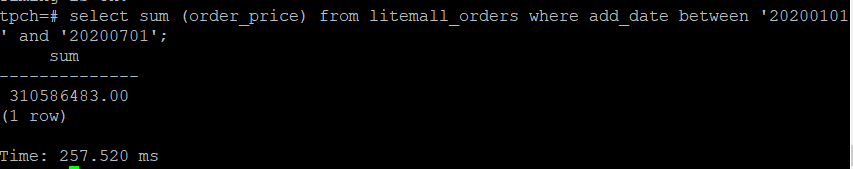
select count(\*) from supplier;;



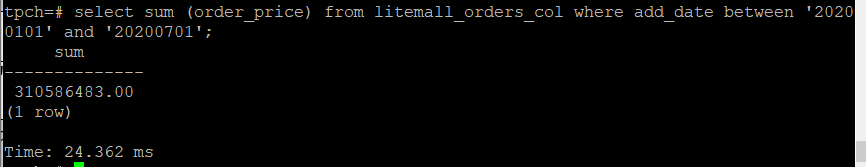
任务二：行存表与列存表执行效率对比

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';

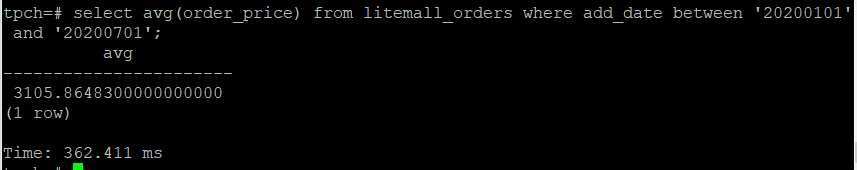


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

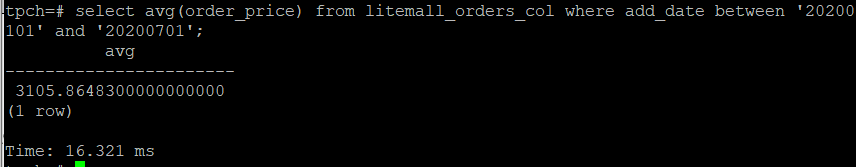


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

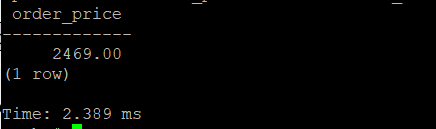


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

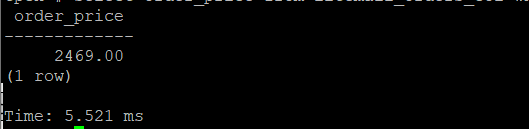


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

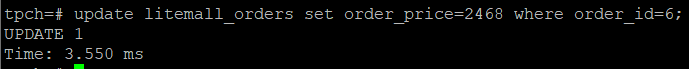


select order\_price from litemall\_orders\_col where order\_id=6;

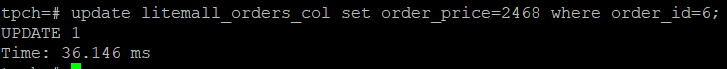


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

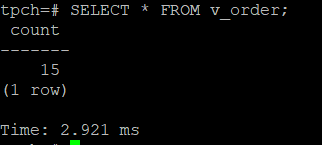
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



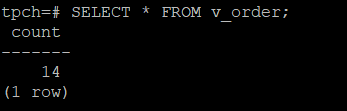
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

在基于行式存储的数据库中， 数据是按照行数据为基础逻辑存储单元进行存储的， 一行中的数据在存储介质中以连续存储形式存在。在基于列式存储的数据库中， 数据是按照列为基础的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。可见行式存储下一张表的数据都是放在一起的，但列式存储下都被分开保存了。所以执行sql语句时，数据读取顺序不同，所以执行时间也不同。

行存表。默认创建表的类型。数据按行进行存储，即一行数据紧挨着存储。行存表支持完整的增删改查。适用于对数据需要经常更新的场景。

列存表。数据按列进行存储，即一列所有数据紧挨着存储。单列查询IO小，比行存表占用更少的存储空间。适合数据批量插入、更新较少和以查询为主统计分析类的场景。列存表不适合点查询，insert插入单条记录性能差。

行存表和列存表的选择原则如下：

更新频繁程度。数据如果频繁更新，选择行存表。

插入频繁程度。频繁的少量插入，选择行存表。一次插入大批量数据，选择列存表。

表的列数。表的列数很多，选择列存表。

查询的列数。如果每次查询时，只涉及了表的少数（<50%总列数）几个列，选择列存表。

压缩率。列存表比行存表压缩率高。但高压缩率会消耗更多的CPU资源。

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创建好的物化视图做全量更新，而不支持做增量更新。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。

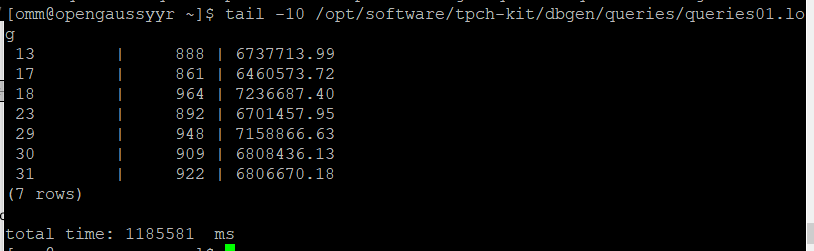
增量物化视图顾名思义就是可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。与全量创建物化视图不同在于目前增量物化视图所支持场景较小，目前物化视图创建语句仅支持基表扫描语句或者UNION ALL语句。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

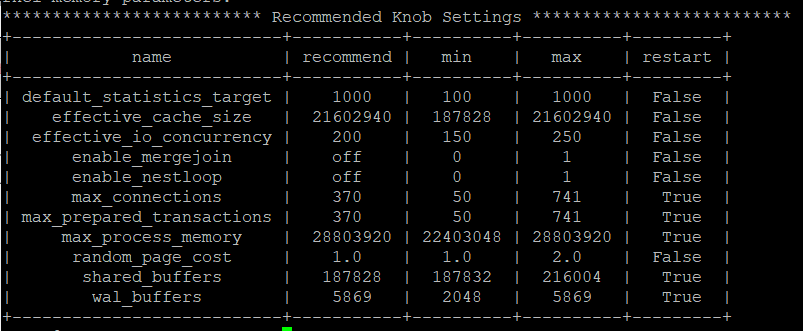
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

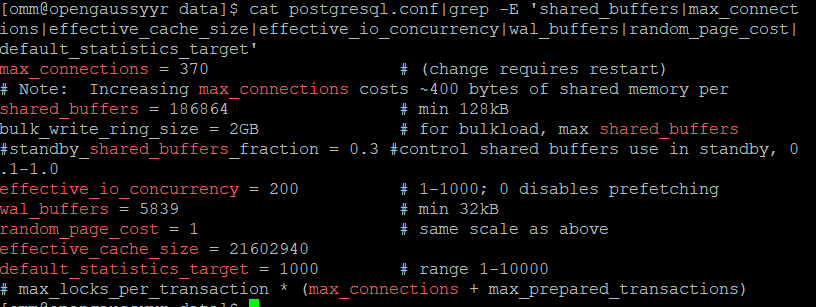
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

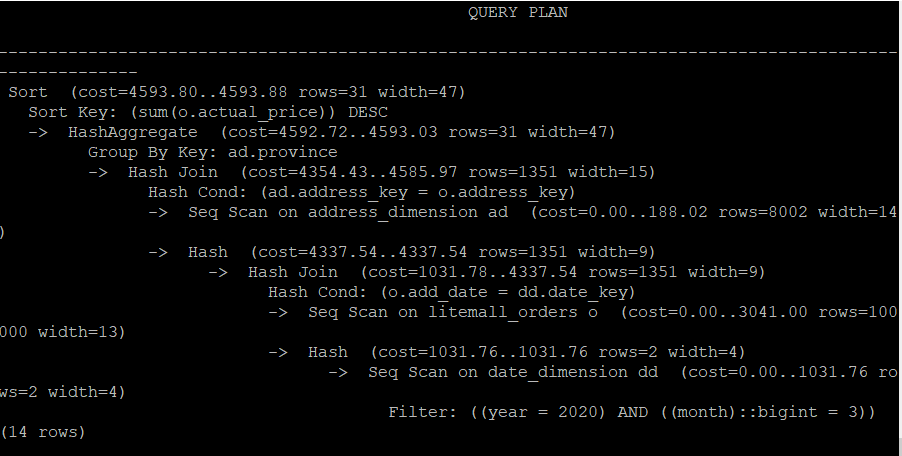
AND o.add\_date = dd.date\_key

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

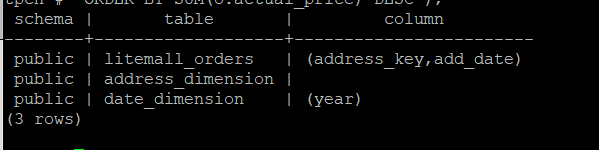
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

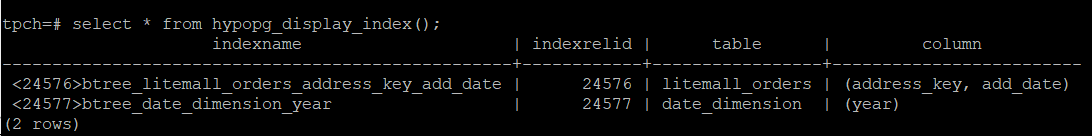
GROUP BY ad.province

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。

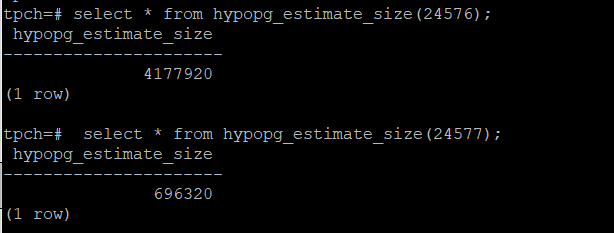
select \* from hypopg\_display\_index();



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

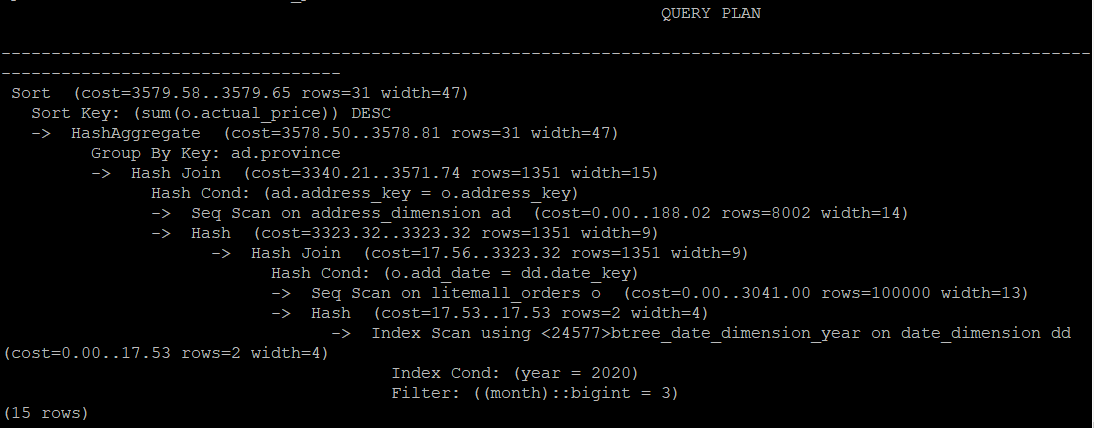
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

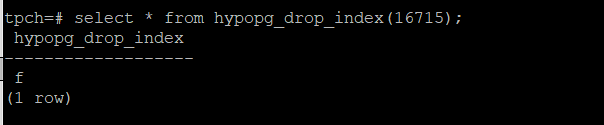
GROUP BY ad.province

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



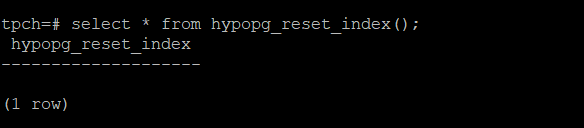
6. 删除某一个索引虚拟列，将执行结果截图。

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



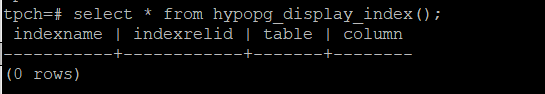
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

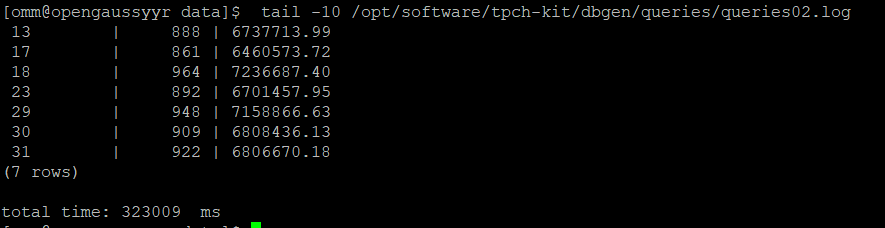
select \* from hypopg\_display\_index();



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：

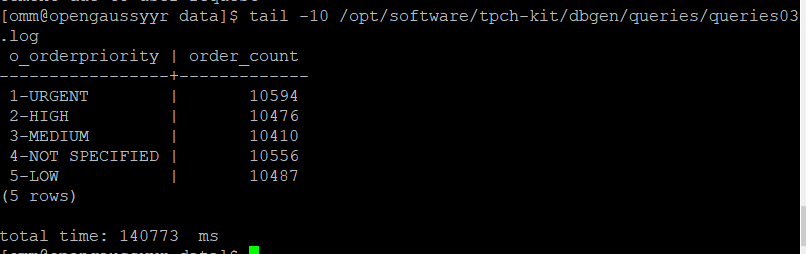
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



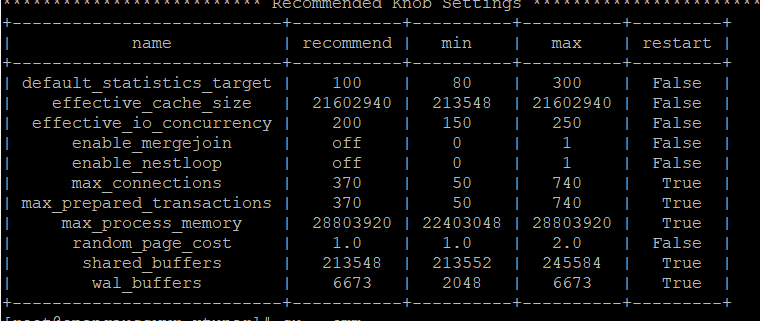
挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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？



gs\_guc set -D /opt/software/openGauss/data/ -c "shared\_buffers = 213548" -c "max\_connections = 370" -c "effective\_cache\_size = 21602940" -c "effective\_io\_concurrency = 200" -c "wal\_buffers = 6673" -c "random\_page\_cost = 1" -c "default\_statistics\_target = 1000"

对数据库进行参数优化，以提升数据库运行性能

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？

DB在执行一条Sql语句的时候，默认的方式是根据搜索条件进行全表扫描，遇到匹配条件的就加入搜索结果集合。如果我们对某一字段增加索引，查询时就会先去索引列表中一次定位到特定值的行数，大大减少遍历匹配的行数，所以能明显增加查询的速度。

数据库优化：

表的设计合理化，符合三大范式（3NF)

1NF是对属性的原子性约束，要求属性(列)具有原子性，不可再分解；(只要是关系型数据库都满足1NF)

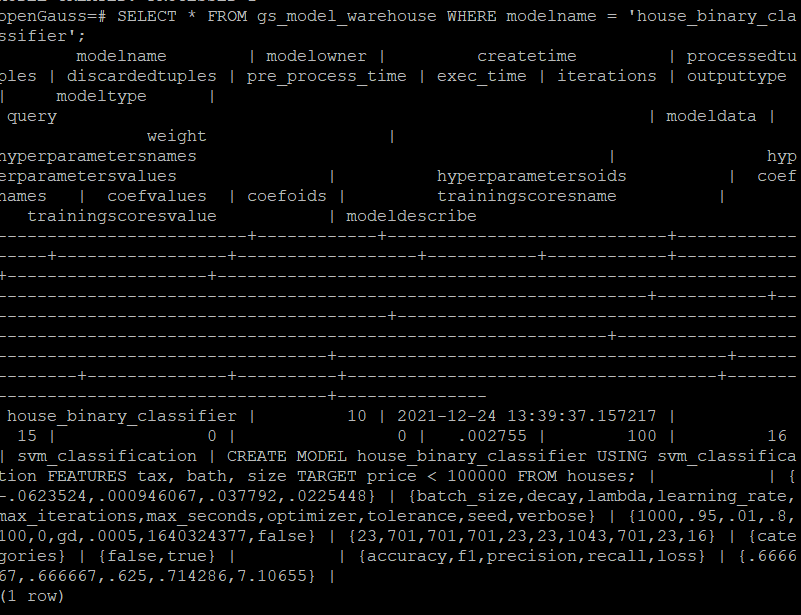
2NF是对记录的惟一性约束，要求记录有惟一标识，即实体的惟一性；

3NF是对字段冗余性的约束，它要求字段没有冗余。 没有冗余的数据库设计可以做到。

# 关卡四、openGauss的DB4AI特性应用

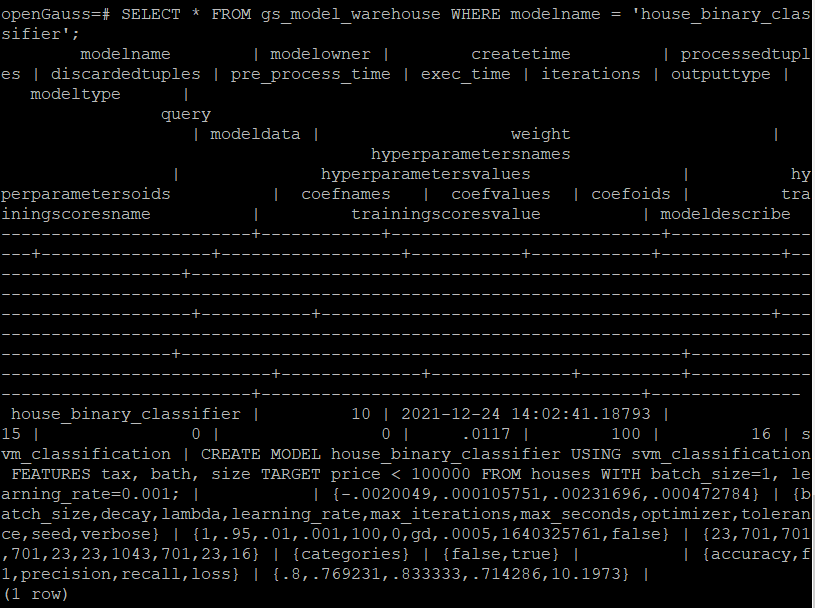
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



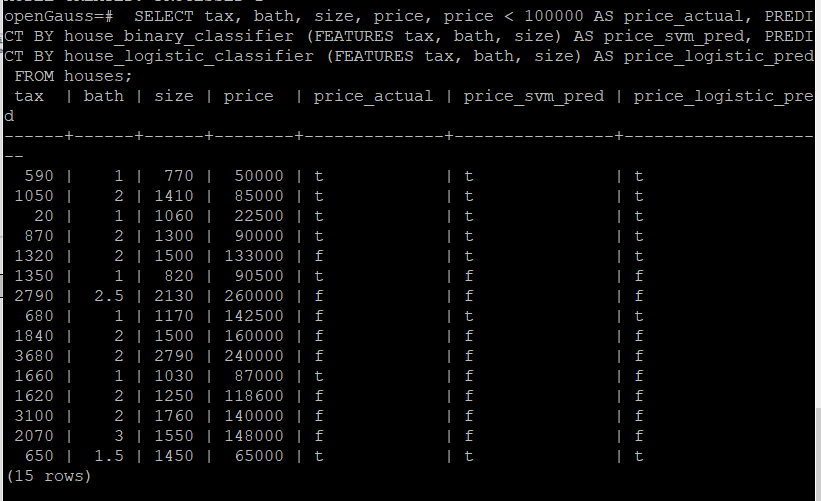
任务二：观察新模型的信息，将执行结果截图。

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](https://www.zhihu.com/search?q=%E5%88%86%E7%B1%BB%E6%A8%A1%E5%9E%8B&search_source=Entity&hybrid_search_source=Entity&hybrid_search_extra=%7B%22sourceType%22%3A%22answer%22%2C%22sourceId%22%3A151216012%7D)和回归模型本质一样，分类模型可将回归模型的输出离散化，回归模型也可将分类模型的输出连续化

实践思考题2：什么是SVM算法？

SVM的全称是Support Vector Machine，即支持向量机，主要用于解决模式识别领域中的数据分类问题，属于有监督学习算法的一种。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核非线性两大类。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，并且使得本本集中所有数据到这个超平面的距离最短。

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？

 AUC（Area under curve）顾名思义就是ROC曲线下的面积，衡量学习器优劣的一种性能指标

F1分数是一个常用指标，F1值是精确率和召回率的调和均值

召回率(Recall)是针对原样本而言的，其含义是在实际为正的样本中被预测为正样本的概率

精确率(Precision)是针对预测结果而言的，其含义是在被所有预测为正的样本中实际为正样本的概率

准确率(Accuracy)预测正确的结果占总样本的百分比

二分类混淆矩阵的一般定义只是将1和0叫做正例和负例，把4种结果的样本数量用符号来表示

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？

均误差方（MSE）：所有样本的样本误差的平方的均值

平均绝对误差（MAE）：所有样本的样本误差的绝对值的均值

平均绝对比例误差（MAPE）：所有样本的样本误差的绝对值占实际值的比例，mape越接近0，模型越准确。

R方：因变量的方差能被自变量解释的程度，R方越接近1，则代表自变量对因变量的解释度越高