openGauss AI特性创新实践课



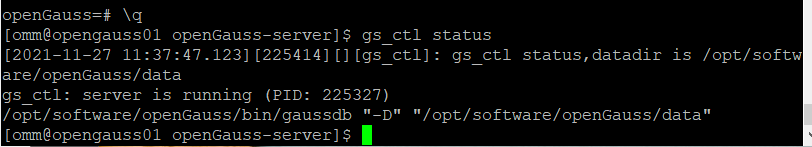
华为技术有限公司

# 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：

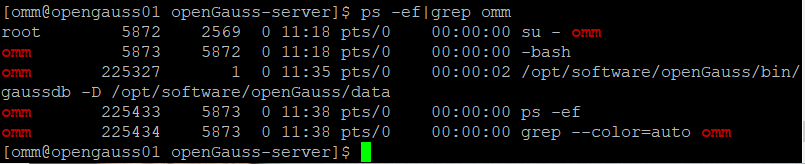
任务一：数据库状态验证

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）



在关卡一中，安装数据库主要分为两个大步骤，分别是进行编译前的准备以及添加环境变量配置文件等。其中，编译前需要登陆虚拟机、创建新的用户、创建openGauss的安装路径、下载第三方编译库、上传cmake包、安装yum安装依赖包、检验python版本、修改权限。之后关于openGauss数据库的初始化主要包括添加环境变量、生成配置文件、编译、安装、初始化。之后就可以启动、登录、退出、正常使用数据库了。

实验思考题：为什么需要通过源码编译，安装数据库？

在这个过程中，主要是通过源码编译完成的数据库安装，主要原因有三个：

1、为了满足不同的运行平台。由于Linux版本众多，但每个版本采用的软件或内核都不同，二进制包所依赖的环境不一定能够正常运行，所以可以直接通过软件提供的源码进行安装；

2、方便定制、满足不同的需求。主要是需要什么就安装什么，而大多数二进制代码都是一键装全，所以自由度并不是很高；

3、方便运维、开发人员维护。所有的源码都可以进行打包转化为二进制，但这是一个额外的工作，代价比较大。所以软件厂商会直接对源码进行维护，而二进制是Linux还行上进行维护。

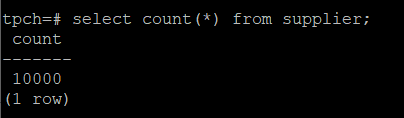
4、不会对系统造成污染。

# 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：

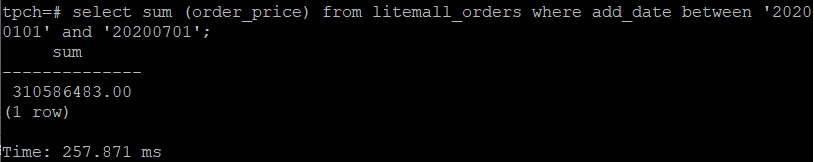
select count(\*) from supplier;;



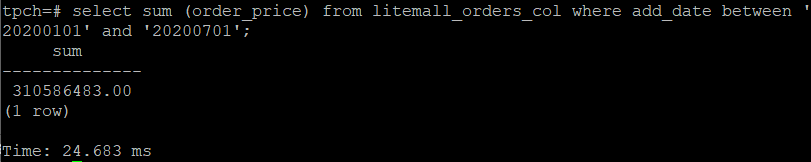
任务二：行存表与列存表执行效率对比

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';

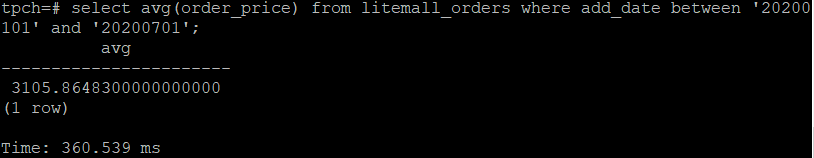


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

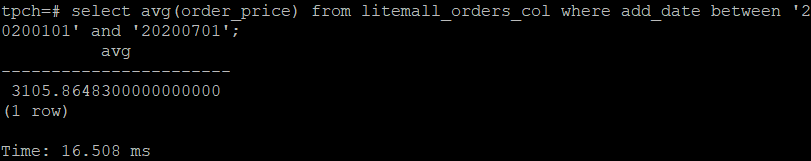


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

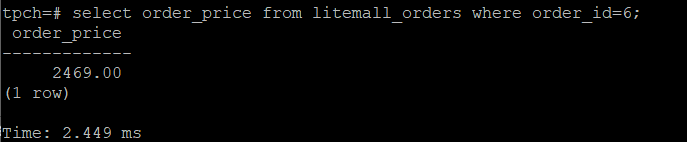


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

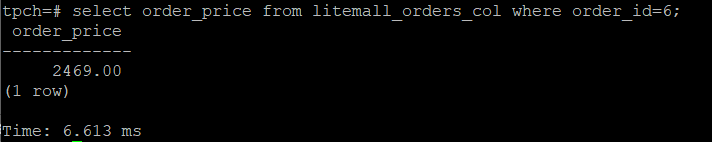


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

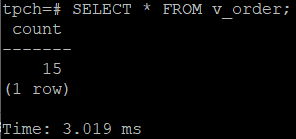
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

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。

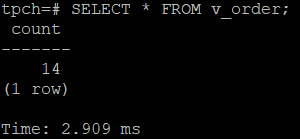
SELECT \* FROM v\_order;



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;

进行数据删除后：



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;

插入数据后：



再次插入数据后：



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？

从实验结果可知，在求某一个属性的简单操作时，列存表的运算速度比行存表要快许多（24.603<257.871；16.508<360.539），但一旦涉及先查询、筛选再输出、修改的操作，行存表的操作要远远快于列存表（2.449<6.613；3.736<77.045）。

从数据写入、修改的角度看，行存表的写入是一次完成的。如果这种写入建立在操作系统的文件系统上，可以保证写入过程的成功或者失败，数据库的完整性可以因此确定；而列存表是需要把一行记录拆分成单列保存，写入的次数明显比行存表多，实际消耗更大。同理数据修改的过程。所以行存表在数据写入、修改上的优势远大于列存表。

从数据读取角度来说，行存表通常将一行数据完全读出，如果只需要其中几列数据的情况，就容易出现冗余列，处理这些冗余数据就会让时间变长一些；而列存表每次读取的数据是集合的一段或全部，不存在冗余性问题。

所以总结来说，行存表的优点是：写入是一次性完成，消耗的时间比列存表少，并且能够保证数据地完整性，缺点是：数据读取过程中会产生冗余数据，如果只有少量数据可能影响较小，但数据量大，处理数据的效率可能会降低。列存表的优点是读取过程中不会产生冗余数据，对于数据完整性要求不高的大数据处理领域比较重要；可以针对各列的运算并发执行，无需维护索引、减小无关IO、避免全表扫描，可能降低查询响应时间；在执行压缩操作时效率更高，提高物理存储利用率；如果某一行的某一列没有数据，那么在列存储时就可以不存储该列的值，这比行存储更节省时间。

行存储的使用场景：

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。

列存储的使用场景：

①提高OLAP大数据量查询的效率

②统计分析类查询（group、join多的场景）

③即席查询（查询条件列不稳定，行存无法确定索引）

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。

在两个物化视图进行查询的过程中，全量物化视图查询仅输出了数据的个数，而增量物化视图会输出数据的具体内容。带来的效果就是全量物化视图的查询速度较快。

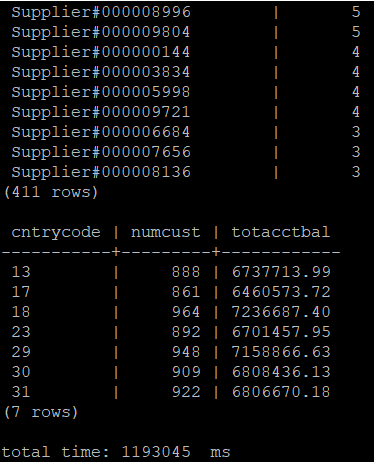
# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

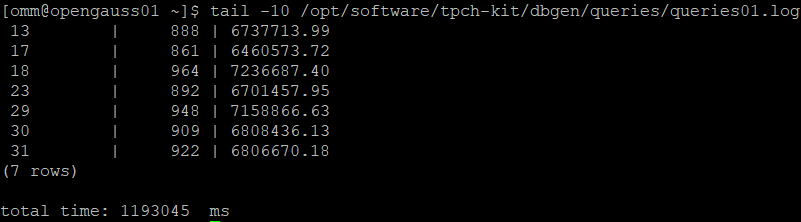
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

新窗口：

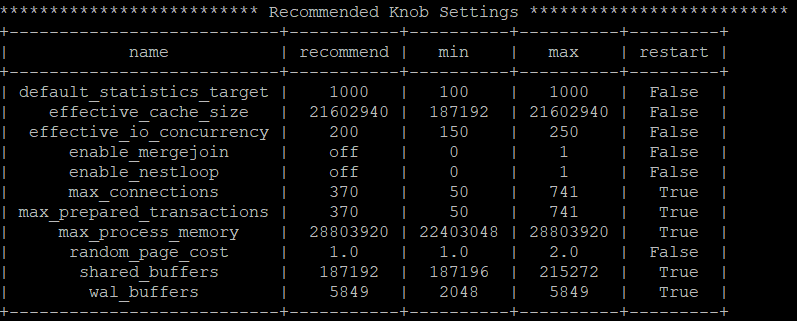


原来的窗口：



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

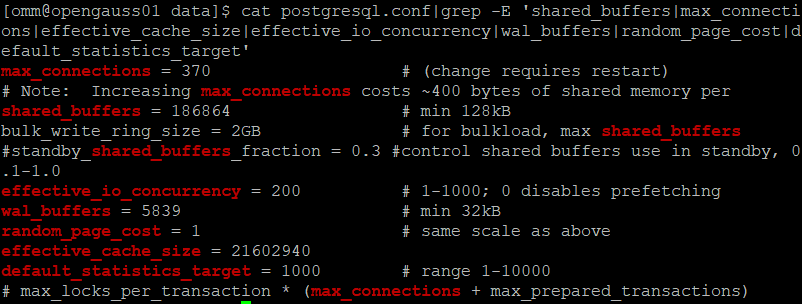
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

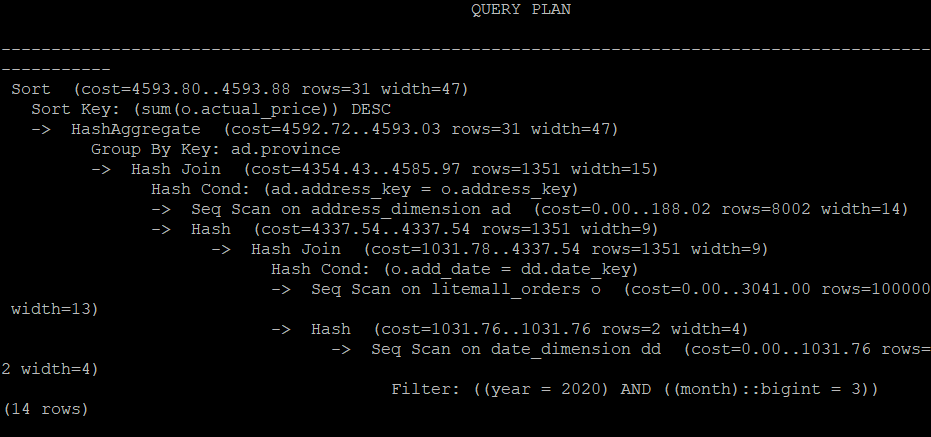
AND o.add\_date = dd.date\_key

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

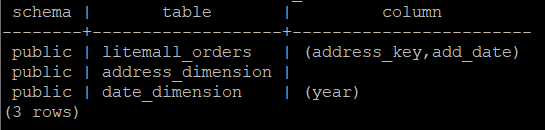
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

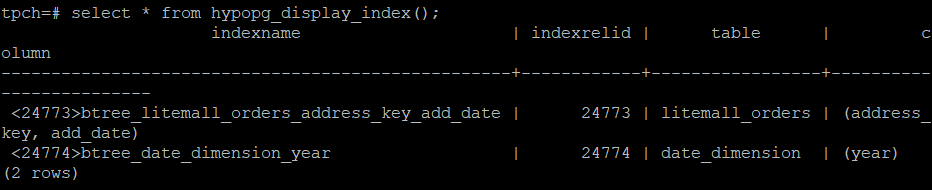
GROUP BY ad.province

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。

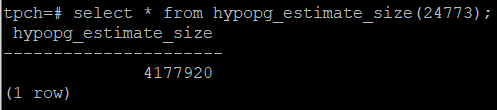
select \* from hypopg\_display\_index();

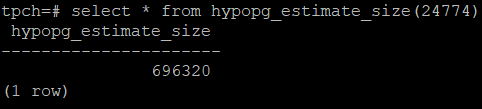


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

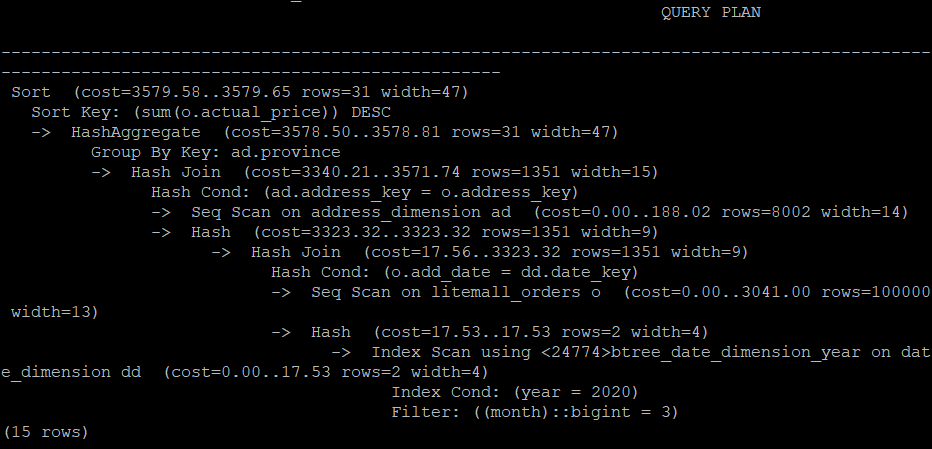
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

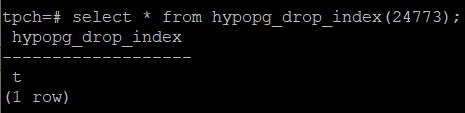
GROUP BY ad.province

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。

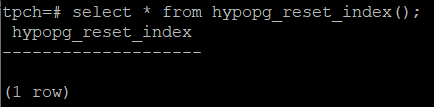
select \* from hypopg\_drop\_index(16715);



t表示已经将这个虚拟索引列删掉了

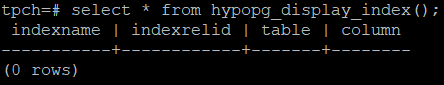
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

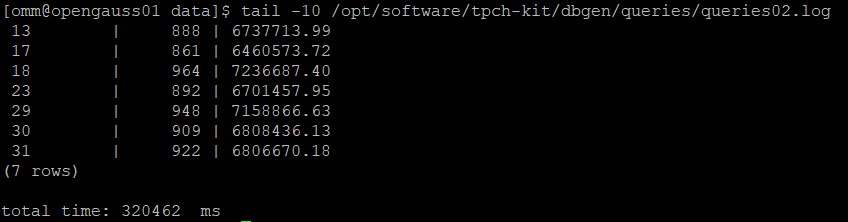
select \* from hypopg\_display\_index();



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？

在shared\_buffers、max\_connections、effective\_cache\_size、effective\_io\_concurrency、wal\_buffers、random\_page\_cost、default\_statistics\_target的recommend上尝试进行了优化。

recommend是获取当前正在运行的workload特征信息，根据上述特征信息生成参数推荐报告。报告当前数据库中不合理的参数配置和潜在风险等；输出当前正在运行的workload行为和特征；输出推荐的参数配置。该模式是秒级的，不涉及数据库的重启操作，其他模式可能需要反复重启数据库。

1、shared\_buffer决定有多少内存可以被PostgreSQL用于缓存数据。在大内存环境下，shared\_buffer占比可更高一些，小内存情况下占比应下调。PostgreSQL既使用自身的缓冲区，也使用内核缓冲IO。这意味着数据会在内存中存储两次，首先是存入PostgreSQL缓冲区，然后是内核缓冲区。这被称为双重缓冲区处理。对大多数操作系统来说，这个参数是最有效的用于调优的参数。

2、max\_connections是允许客户端的最大并发连接数。

3、effective\_cache\_size是优化器假设一个查询可以用的最大内存，它只是一个建议值，而不是确切分配的内存或缓存大小。它不会实际分配内存，而是会告知优化器内核中可用的缓存量。在一个索引的代价估计中，更高的数值会使得索引扫描更可能被使用，更低的数值会使得顺序扫描更可能被使用。

4、effective\_io\_concurrency是设置PostgreSQL可以同时被执行的并发磁盘I/O操作的数量的参数。调高这个值，可以增加任何单个PostgreSQL会话试图并行发起的I/O操作的数目。允许的范围是1到1000，或0表示禁用异步I/O请求。

5、wal\_buffers指定事务日志缓冲区中包含的数据块的个数，默认大小为16MB，如果调优的系统有大量并发连接，那么该值越高，性能越好。

6、random\_page\_cost是从数据文件上随机读取一个数据块的执行成本，默认是4.0，在这里改为了1。此设置规定，非连续牵强磁盘页面将有成本，而且直接影响到查询规划决策。使用高等待时间的存储时，如旋转磁盘用保守的值会是特别重要的。

7、default\_statistics\_target设置默认的收集优化器统计数据的目标值。它的值越大，ANALYZE操作的执行的时间越长，扫描的数据行的个数也就越多，得到的优化器统计数据就越准确。默认值是10。

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？

创建索引的好处：

①通过创建唯一性索引，可以保证数据库中每一行数据的唯一性；

②可以大大加快数据的检索速度，这是最主要的原因

③可以加速表和表之间的连接，特别是在实现数据地参考完整性方面特别有意义

④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

⑤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。

数据库优化的其他方法：

①设计表的时候严格根据数据的设计规范来设计数据库

②使用缓存，把经常访问到的且不需要变化的数据放在缓存中

③使用固态硬盘

④垂直分表，把一些不经常用的数据放到一个表中，节约磁盘的IO

⑤主从分离读写，采取主从复制把数据库的读操作和写操作分离

⑥选取最适用的字段属性

⑦使用Union来代替手动创建的临时表

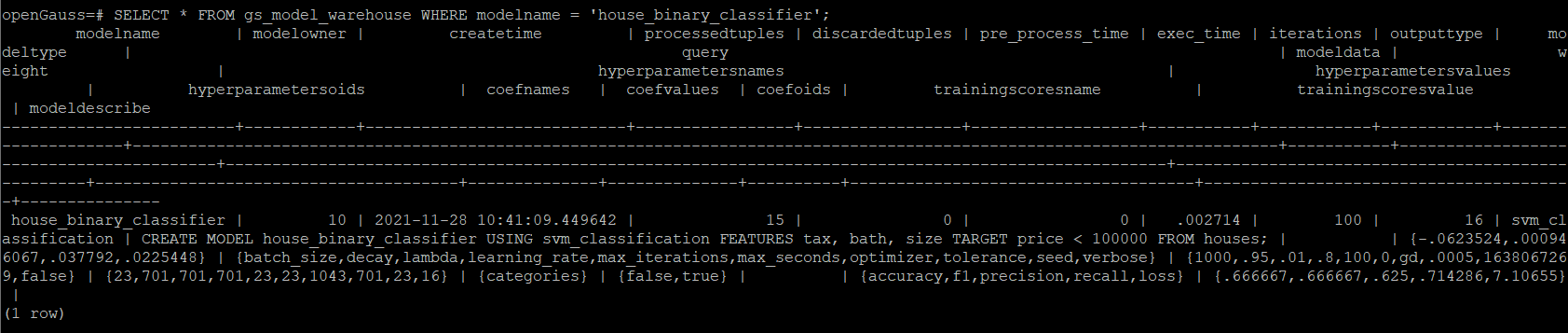
⑧使用事务、外键，或锁定表

⑨优化查询语句

# 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：

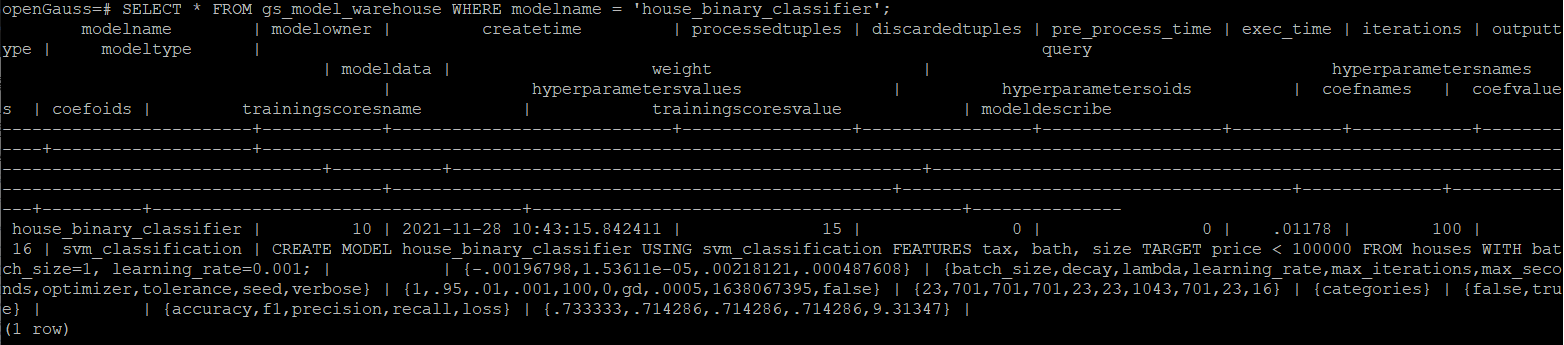
postgres=# SELECT \* FROM gs\_model\_warehouse WHERE modelname = 'house\_binary\_classifier';



可见训练过程中的预测准确率为0.666667

任务二：观察新模型的信息，将执行结果截图。

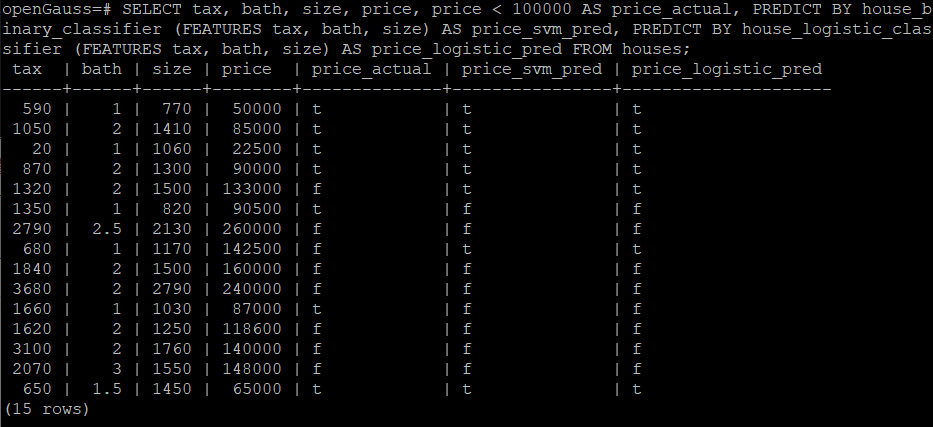
postgres=# SELECT \* FROM gs\_model\_warehouse WHERE modelname = 'house\_binary\_classifier';



可见整体准确率相对默认参数有提升。

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？

分类模型与回归模型的区别在于输出变量的类型，定量输出称为回归，又叫连续变量预测；定性输出称为分类，又叫离散变量预测。

回归问题通常应用于预测房价、未来天气状况等，是对真实值的一种逼近预测；

分类问题通常应用于判断一张照片是猫还是狗这种问题，相当于给事物打上一个标签，没有逼近的概念，最终正确结果只有一个

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。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？

①Accuracy：是正确预测的样本数除以所有的样本数，表示的是预测正确的概率；

②Precision：精确率，是检索出相关文档数与检索出的文档总数的比率（正确分类的正例个数占分类为正例的实例个数的比例），衡量的是检索系统的查准率。

③Recall：召回率，是指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（正确分类的正例个数占实际正例个数的比例），衡量的是检索系统的查全率。

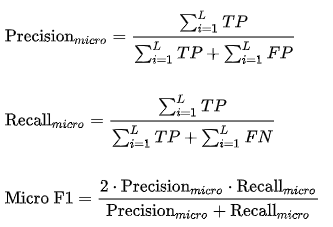
④F1-Score：



⑤Macro F1：宏平均，本质上是所有类别的统计指标的算术平均值来求得的，这样单纯的平均忽略了样本之间分布可能存在极大不平衡的情况。

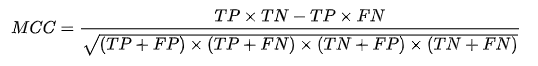


⑥Micro F1 ：微平均，在计算Precision与Recall时会将所有类直接放到一起来计算。



⑦MCC：马修斯相关系数

主要用于衡量二分类问题，其综合考虑了 TP TN, FP , FN，是一个比较均衡的指标，对于样本不均衡情况下也可以使用。MCC的取值范围在 [-1, 1]，取值为1表示预测与实际完全一致，取值为0表示预测的结果还不如随机预测的结果，-1表示预测结果与实际的结果完全不一致。因此我们看到，MCC本质上描述了预测结果与实际结果之间的相关系数。



⑧ROC 曲线，横坐标为FPR，纵坐标为TPR，FPR代表将负例错分为正例的概率，TPR 表示能将正例分对的概率。分类器对应的ROC曲线应该尽可能靠近坐标轴的左上角，而对角线的位置意味着分类器的效果和随机猜测一样的差。

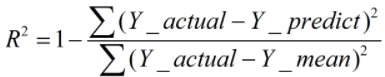
⑨AUC：ROC曲线下的面积，这个面积的数值介于0到1之间，能够直观的评价出分类器的好坏，AUC的值越大，分类器效果越好。

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？

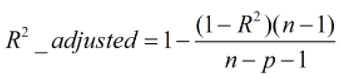
①误差平方和SSE，同样数据集的情况下，SSE越小，误差越小，模型效果越好。



②R-square（决定系数），分母理解为原始数据的离散程度，分子预测为预测数据和原始数据的误差，二者相除可以消除原始数据离散程度的影响。越接近1，表示方程的变量对y的解释能力越强，这个模型对数据拟合的也较好；越接近0模型的拟合效果就越差。



③Adjusted R-Square (校正决定系数)：消除了样本数量和特征数量的影响



④Mean Absolute Error(MAE)：平均绝对误差指预测值与真实值之间平均相差多大，平均绝对误差能更好地反映预测值误差的实际情况。



⑤（Mean Squared Error，MSE）均方误差：观测值与真值偏差的平方和与观测次数的比值，这也是线性回归中最常用的损失函数，线性回归过程中尽量让该损失函数最小。那么模型之间的对比也可以用它来比较。MSE可以评价数据的变化程度，MSE的值越小，说明预测模型描述实验数据具有更好的精确度。

