openGauss AI特性创新实践课



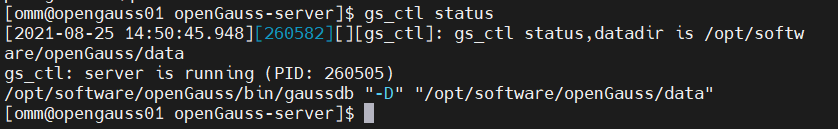
华为技术有限公司

# 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：

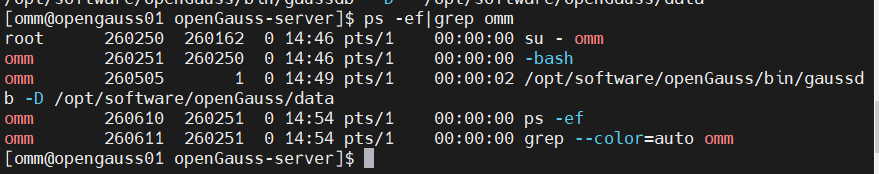
任务一：数据库状态验证

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）



任务三：步骤 3 简单描述关卡一中，安装数据库所需要的步骤。完成以上信息，表明关卡一完成。

1. 使用putty和购买服务器时所用的账号密码远程登录服务器  
2. 创建用数据的安装用户omm以及附属组，并修改用户密码  
3. 使用mkdir命令创建gauss数据库源码的存放文件夹  
4. 使用wget下载并解压第三方编译库，并重命名为binarylibs  
5. 下载openGauss源码  
6. 下载cmake并解压  
7. 使用yum下载python依赖包，并将自带python换成新下载的python3  
8. 切换至用户omm，配置数据库安装所需的环境变量并应有  
9. 进入数据库源码目录下，生成配置文件  
10. 使用make命令对源码进行编译以及安装  
11. 配置数据库hostname以及密码  
12. 启动数据库，数据库安装完成

实验思考题：为什么需要通过源码编译，安装数据库？

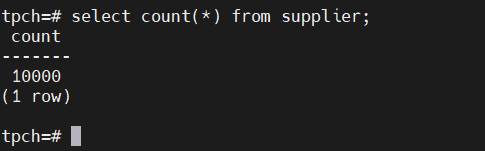
通过源码编译，安装数据库可以在编译与安装过程根据实际需求设定数据库的参数，安装版本等内容，更加灵活。

# 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：

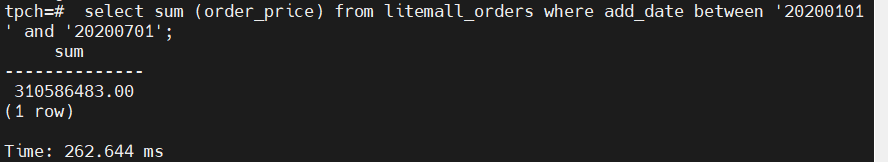
select count(\*) from supplier;;



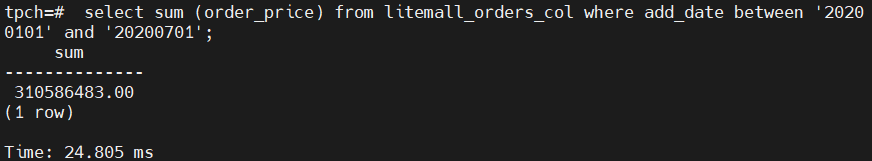
任务二：行存表与列存表执行效率对比

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';

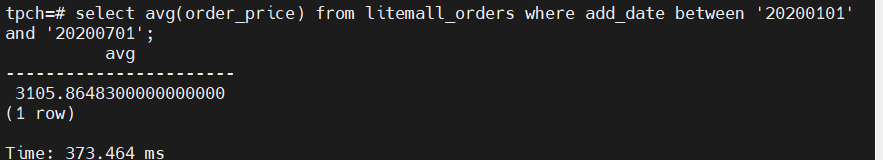


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

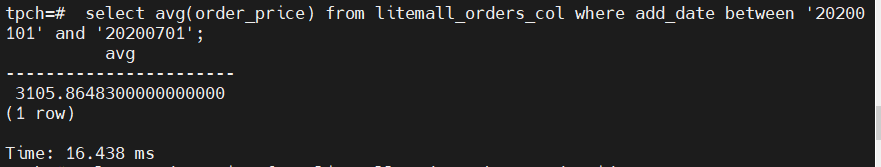


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

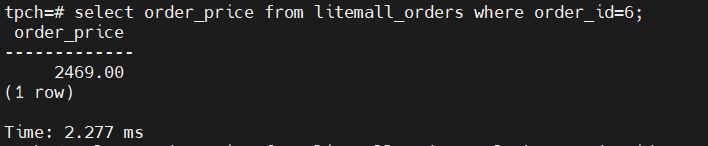


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

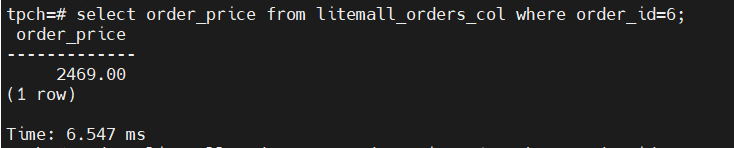


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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



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

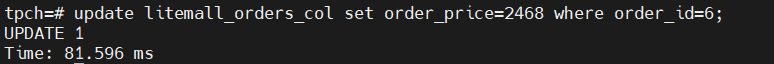


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

QQ截图20210825155559

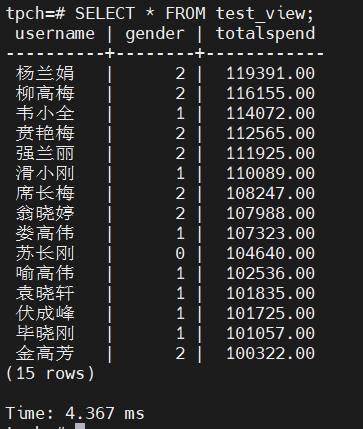
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

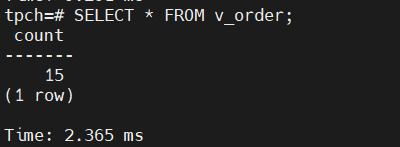
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



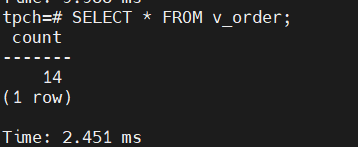
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



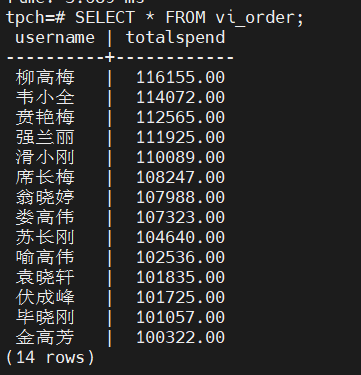
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



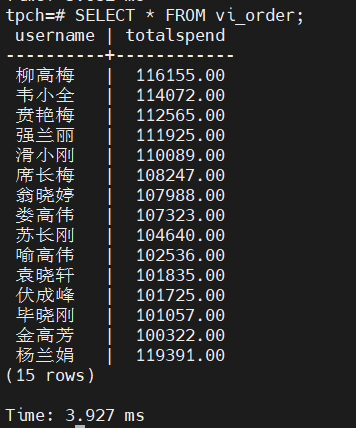
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;

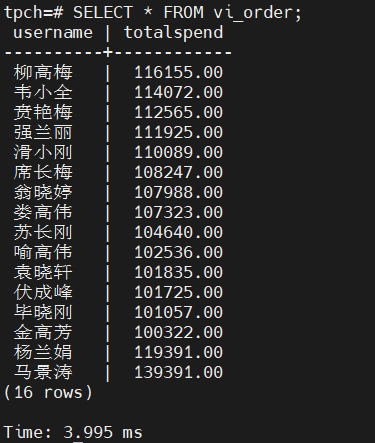


5. 对表进行操作后，刷新增量物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM vi\_order;



6. 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。



思考题

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？全量物化视图与增量物化视图有哪些差别？

答：

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？

按行存储对应没有索引的查询会使用大量IO，建立索引和物化视图时也需要花费大量时间和资源，在单列查询时会读取一些不必要的数据  
按列存储数据本身即是索引，且查询时只访问设计的列，可以大量降低系统IO，但是在整行读取时，可能需要多次IO操作

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？  
行存表效率更高：进行整行修改，增加的操作  
列存表效率更高：面向单列数据的读取，统计等操作

全量物化视图与增量物化视图有哪些差别？

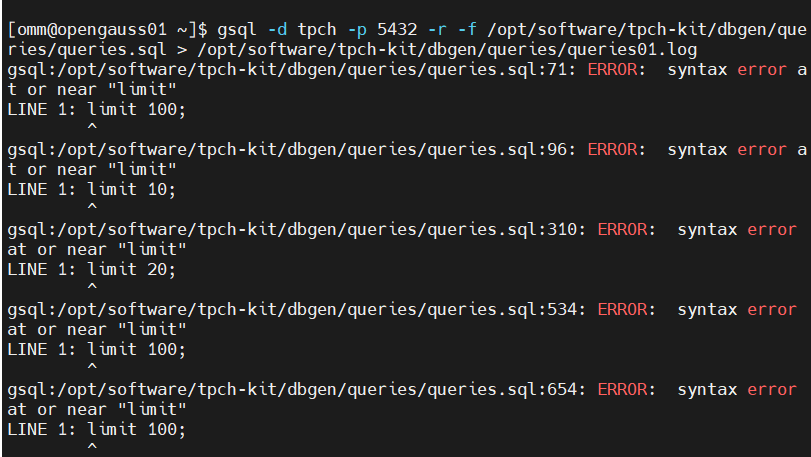
全量物化视图仅支持对创建好的物化视图做全量更新，不支持做增量更新。  
增量物化视图可以对物化视图进行增量刷新，用户执行语句时便对一段时间内数据作增量更新  
在数据量不大时使用全量刷新和增量刷新区别不大，但是当数据量过大时，选择全量刷新会耗费大量的时间，此时若是使用增量刷新可以减少操作时间

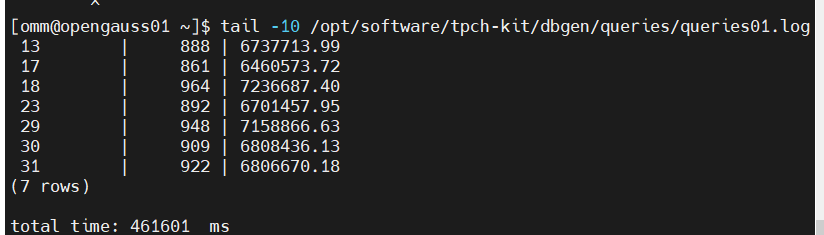
# 关卡三、openGauss的AI4DB特性应用

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：

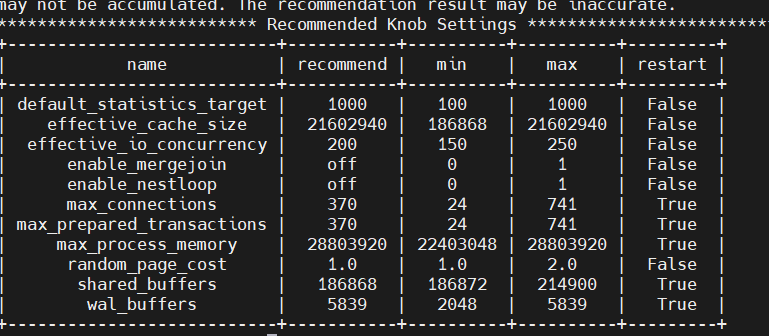
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log





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

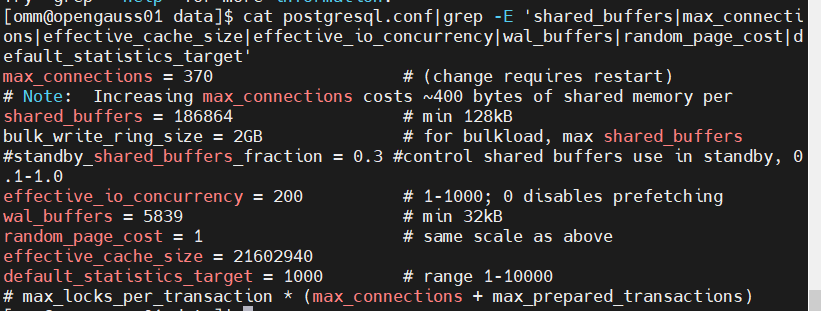
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

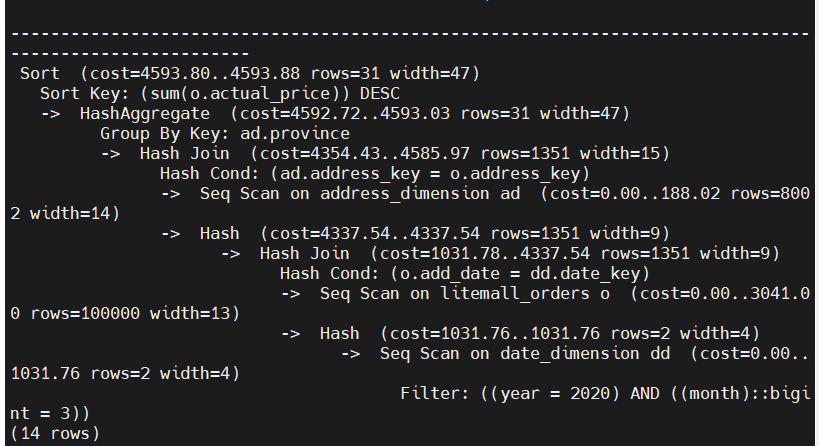
AND o.add\_date = dd.date\_key

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

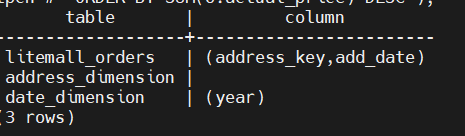
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

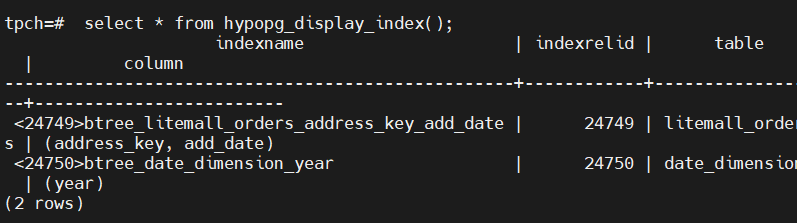
GROUP BY ad.province

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。

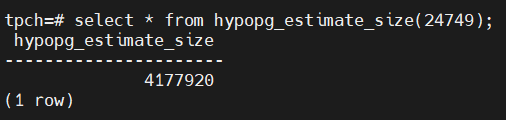
select \* from hypopg\_display\_index();

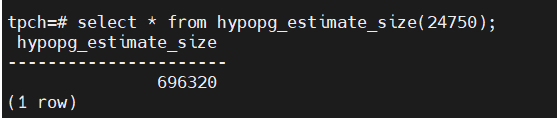


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

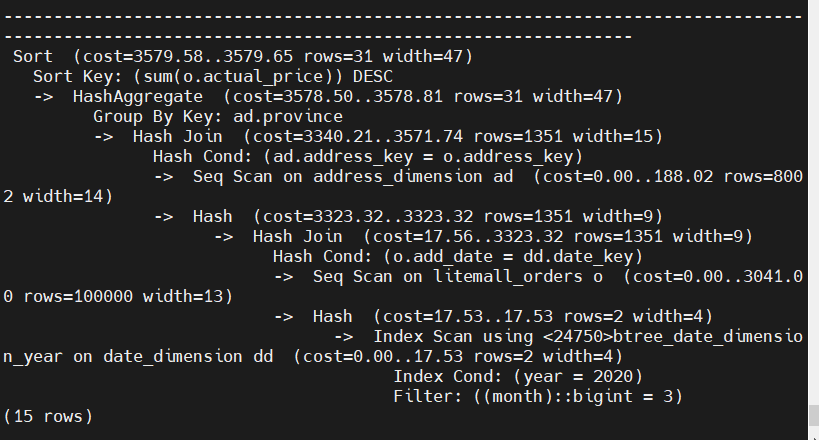
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

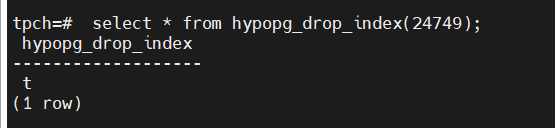
GROUP BY ad.province

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



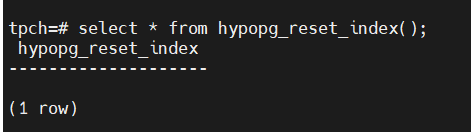
6. 删除某一个索引虚拟列，将执行结果截图。

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



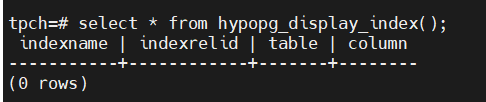
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

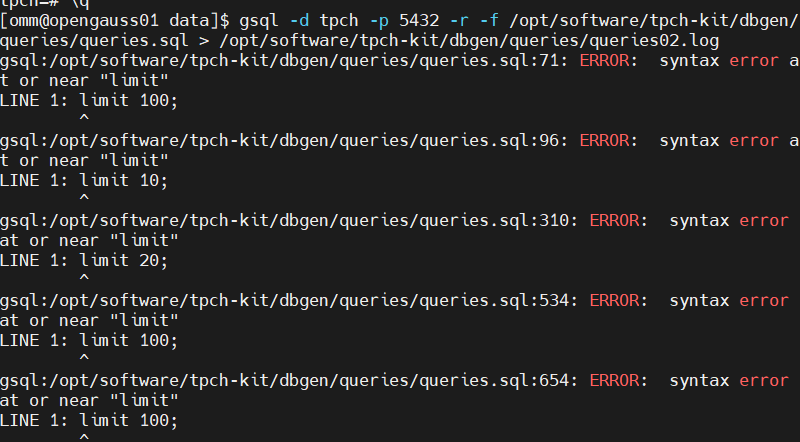
select \* from hypopg\_display\_index();

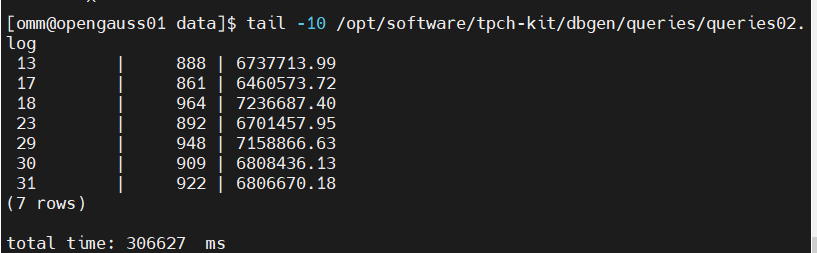


任务三：通过创建索引，对queries.sql中的SQL语句进行优化，并对比优化前后queries.sql执行的时间。

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





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？

答：对索引索引进行了优化，通过使用虚拟索引推荐，得到了虚拟索引列。以提高查询效率，实现查询的优化。

因为这些参数对于查询时间的影响更大，因而需要对其进行索引优化

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？

答：索引提供指向存储在表的指定列中的数据值的指针，能够加快表的查询速度。

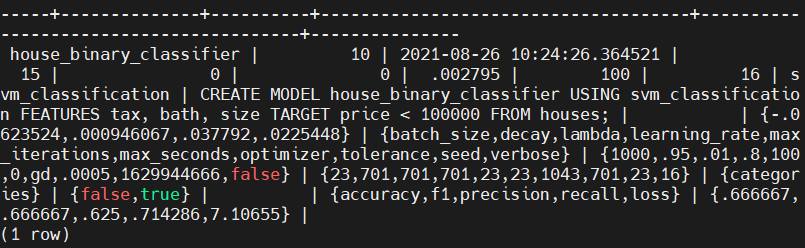
数据库的优化还可以从很多方面进行考虑，比如：

1. 数据类型的选择。如：int，bigint等类型，如何选择
2. 范式的选择。
3. 索引的优化，删去或增加冗余的索引以提高查询效率
4. 优化查询语句，如加入LIMIT,GROUP BY等

# 关卡四、openGauss的DB4AI特性应用

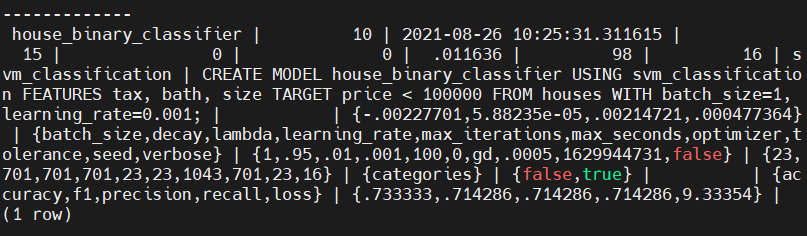
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



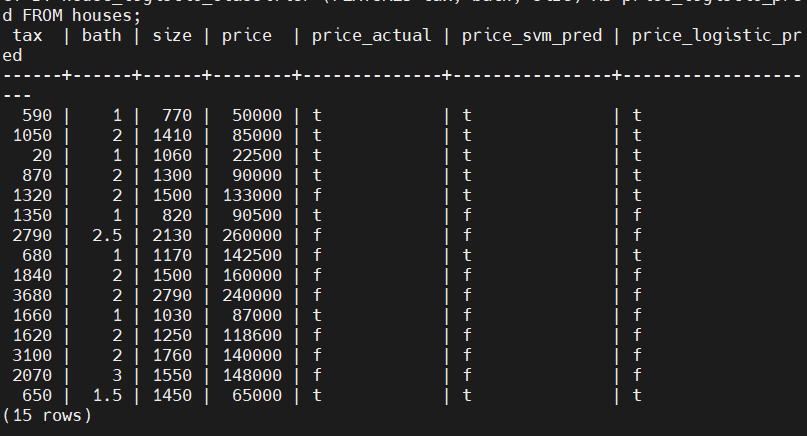
任务二：观察新模型的信息，将执行结果截图。

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？

答：

1.两者的预测目标变量类型不同，回归问题是连续变量，分类问题离散变量

2. 回归问题是定量问题，分类问题是定性问题。

3.回归模型一般用于预测一个值，例如预测房价，未来天气，分类模型一般用于给事物打上标签，例如判断西瓜是否成熟，图片中的生物是猫还是狗

实践思考题2：什么是SVM算法？

答：SVM算法是经典的二维分类算法，旨在通过找到一个合适的超平面将数据分为两类并使得类间间距最大

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？

答：有精度、召回率、精确率、ROC-AUC 、混淆矩阵、PRC。

精度：被正确分类的样本占总样本的比。

精确率：意义：查准。就是预测出来为正样本的结果中，有多少是正确分类。

召回率：真实为正样本的结果中，有多少是正确分类。

ROC：ROC叫做受试者工作特性曲线，反应敏感度和特异度连续变量的综合指标。纵坐标为敏感度，横坐标是特异度。ROC是一个以TPR(真正率)为纵坐标，FPR(假正率)为横坐标构造出来的一幅图。

TPR越高覆盖率越高；我们也希望FPR越低精确率越高

AUC：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。

混淆矩阵：混淆矩阵（Confusion Matrix），又称为可能性矩阵或错误矩阵，主要用于比较分类结果和实际测得值，可以把分类结果的精度显示在一个混淆矩阵里面

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？

答：有SSE(误差平方和)，R-square(决定系数)，Adjusted R-square

SSE(误差平方和)：SSE的值代表误差，SSE越小，误差越小，模型效果越好

R-square(决定系数)：R值越接近1，模型对数据拟合的也越好；越接近0，表明模型拟合的越差

Adjusted R-Square (校正决定系数）用于抵消样本数量对R-Square的影响。