openGauss AI特性创新实践课



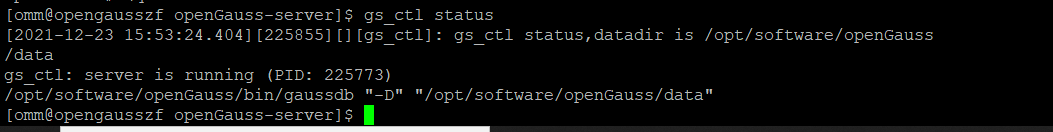
华为技术有限公司

# 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：

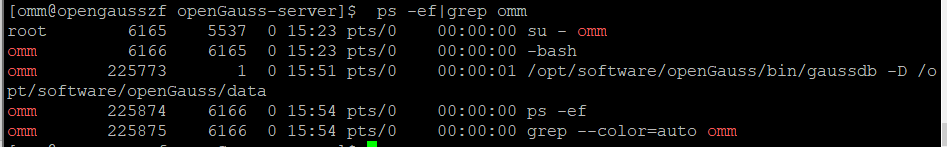
任务一：数据库状态验证

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？

答：

1：满足不同的运行平台，我们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，而我们的二进制包所依赖的环境不一定能够正常运行，所以大部分软件直接提供源码

 2：方便定制，满足不同的需求，很多时候我们所需要的软件都是可以定制的，我需要什么就安装什么，大多数二进制代码都是一键装全，所以自由度并不高！

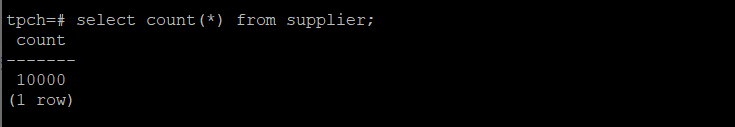
3：方便运维、开发人员维护，我们的源码是可以打包二进制的，但是对于这个软件的打包都会有一份代价不小的额外工作，包括维护，所以如果是源码的话，软件产商会直接维护，但是如果是二进制的话，一般都是Linux发行商提供

# 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：

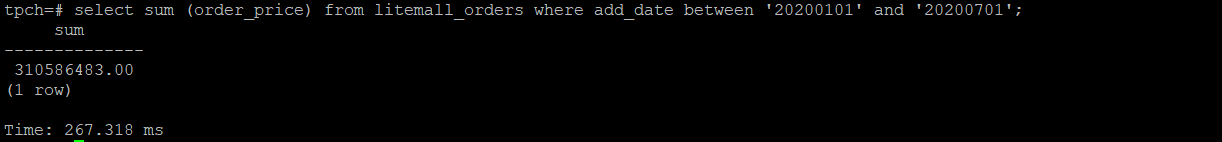
select count(\*) from supplier;;



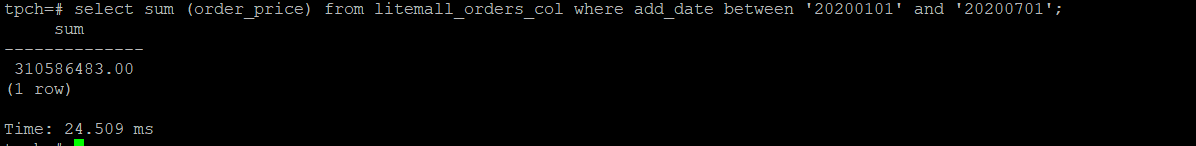
任务二：行存表与列存表执行效率对比

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';

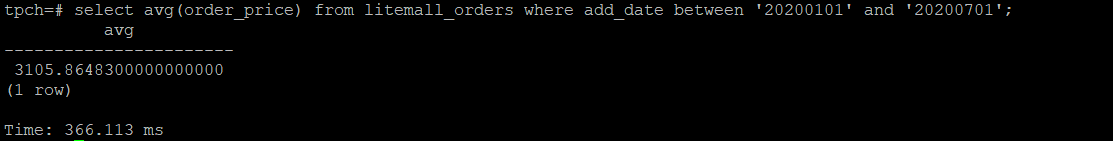


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

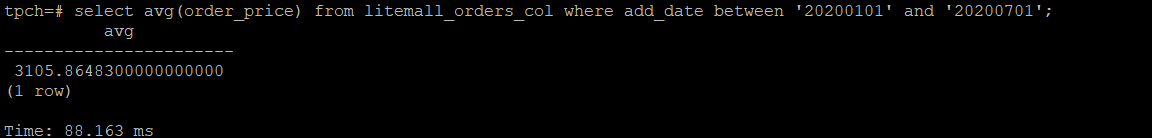


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

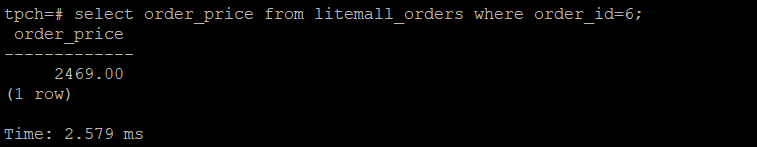


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

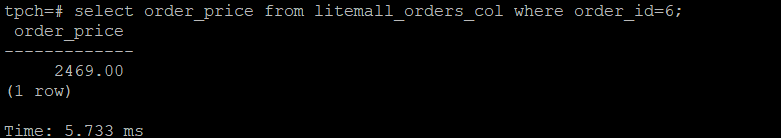


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

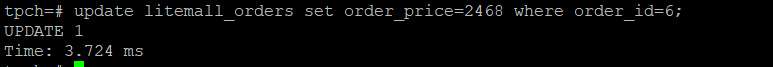


select order\_price from litemall\_orders\_col where order\_id=6;

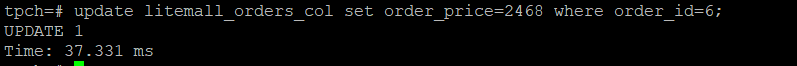


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



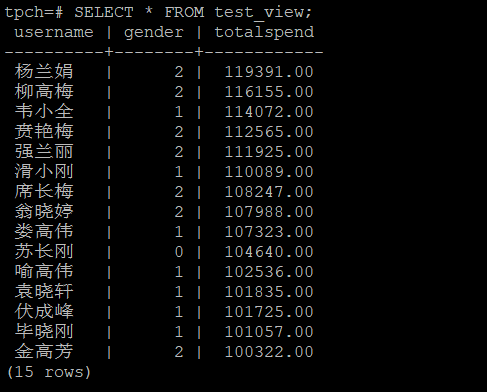
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

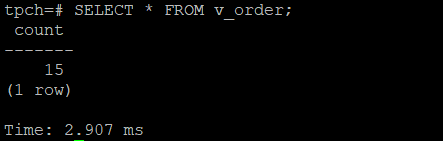
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



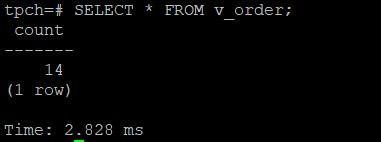
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



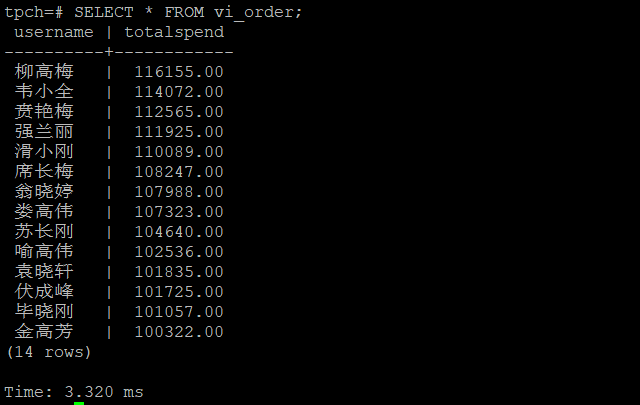
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



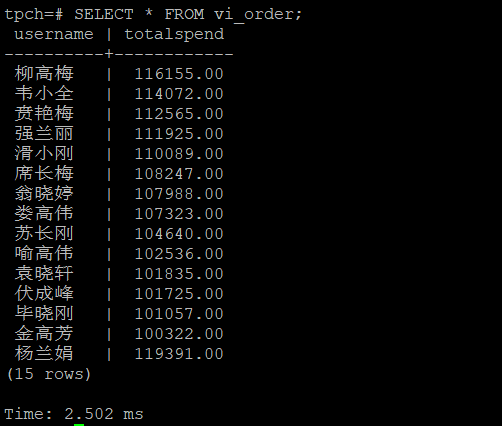
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

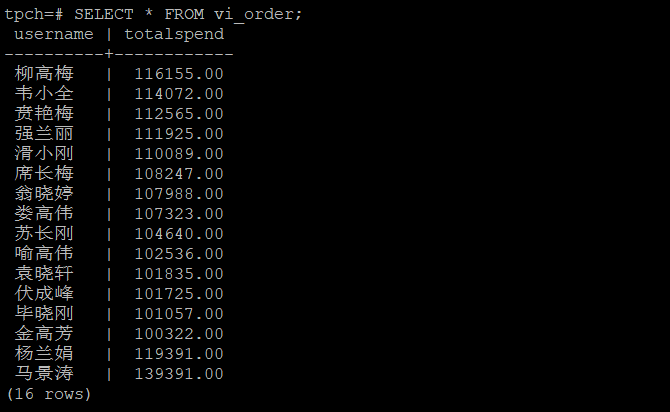
SELECT \* FROM vi\_order;



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？

答：因为两者的数据存储方式不同，行式存储是将一行中的数值串在一起存储起来，然后再存储下一行；列式存储把一列中的数值串在一起存储起来，然后再存储下一列。

当对行数据进行操作时，行存表效率更高，因为只有一个页面被放到了内存中；当对表中某一列的数据做一些操作，列存表效率更高，因为每一列的信息都是存储在一起的。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

答：全量物化视图只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。

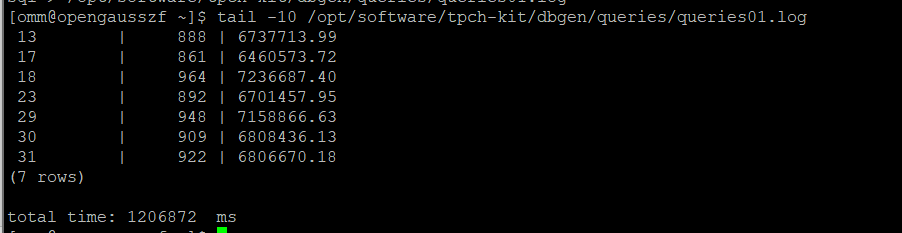
增量物化视图支持物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

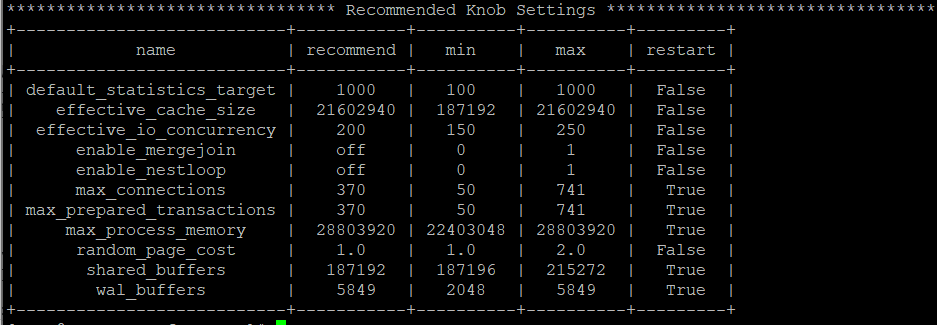
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

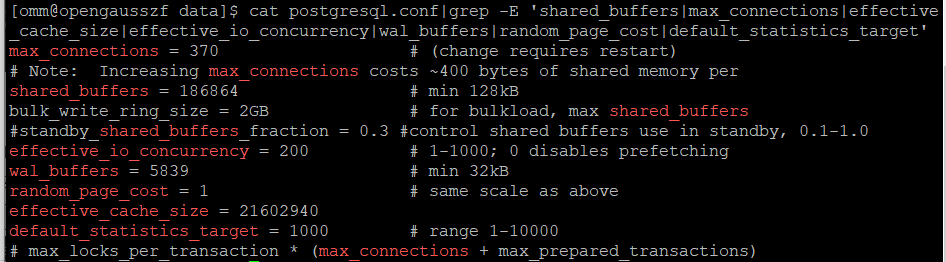
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

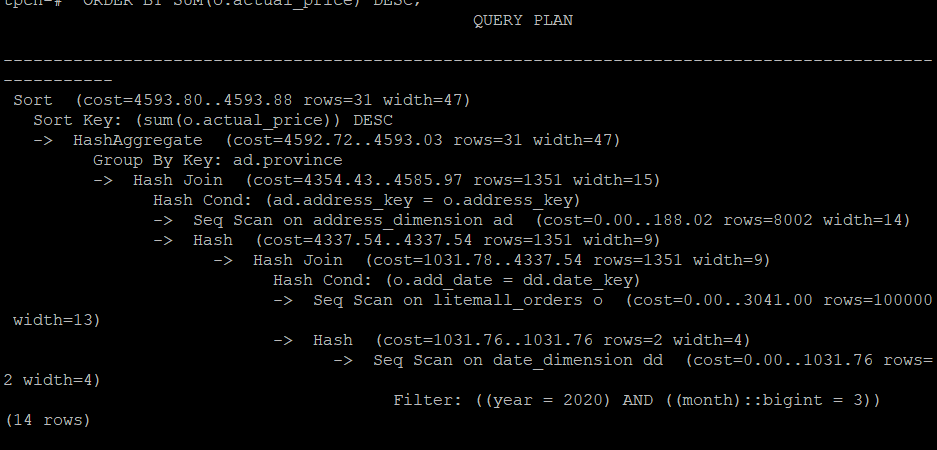
AND o.add\_date = dd.date\_key

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

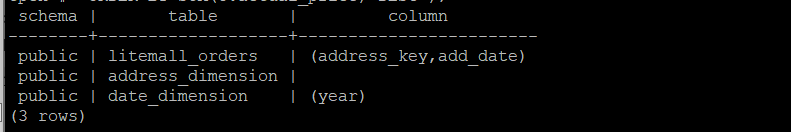
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

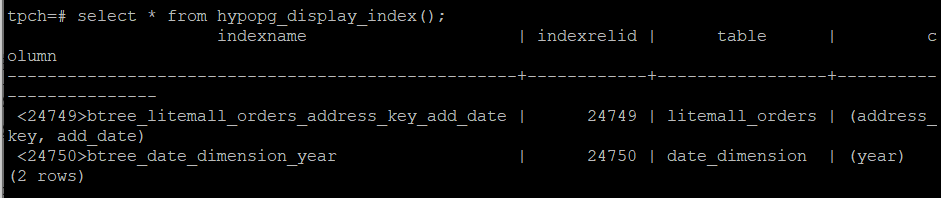
GROUP BY ad.province

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。

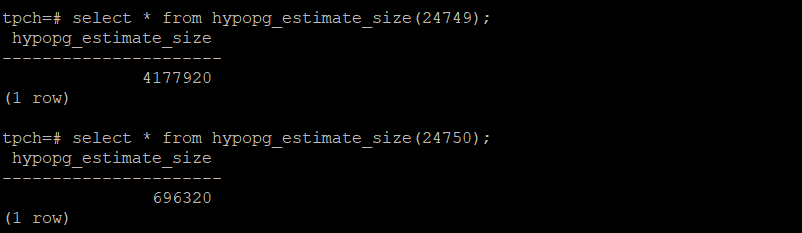
select \* from hypopg\_display\_index();



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

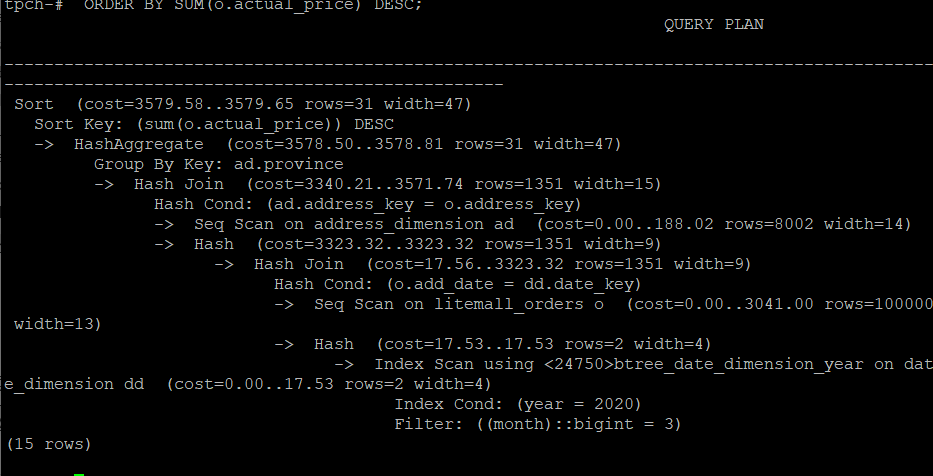
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

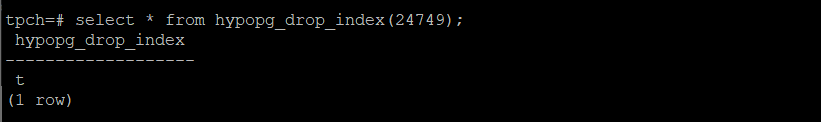
GROUP BY ad.province

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



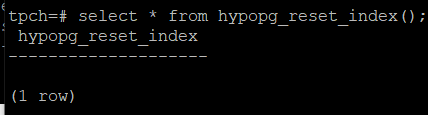
6. 删除某一个索引虚拟列，将执行结果截图。

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



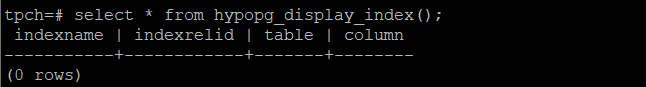
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

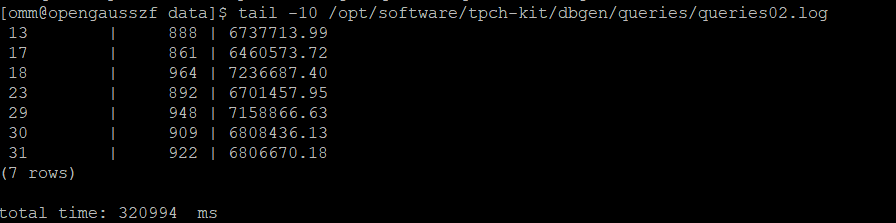
select \* from hypopg\_display\_index();



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？

答：优化参数为：shared\_buffers, max\_connections, effective\_cache\_size, effective\_io\_concurrency, wal\_buffers, random\_page\_cost, default\_statistics\_target.

优化目的：让数据库的效率更高。

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：

好处：可以加快数据的检索速度，加快表之间的连接，减少分组和排序时间以及使用优化隐藏器提高系统性能，同时通过唯一性索引可确保数据的唯一性。

优化方法：

选取最适用的字段属性。

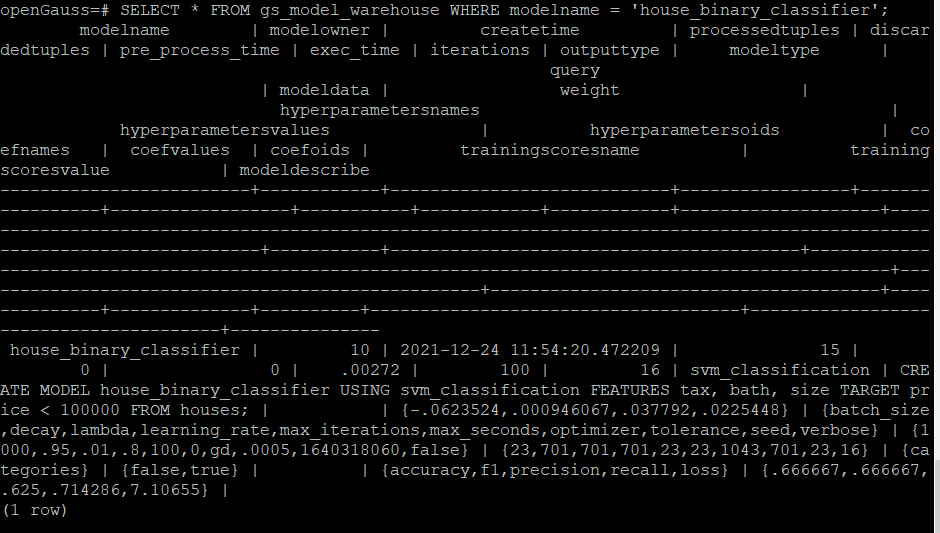
使用缓存，把经常访问到的数据而且不需要变化的数据放到缓存中。

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出来。

# 关卡四、openGauss的DB4AI特性应用

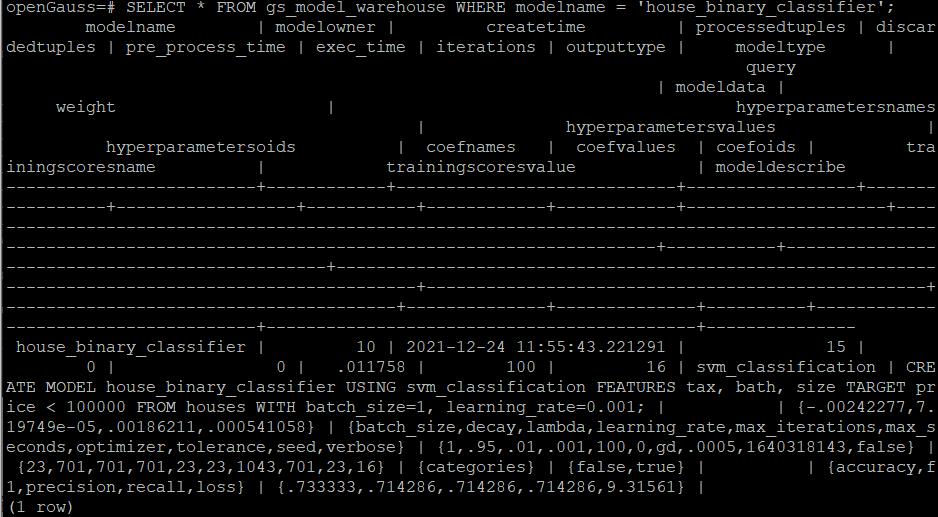
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



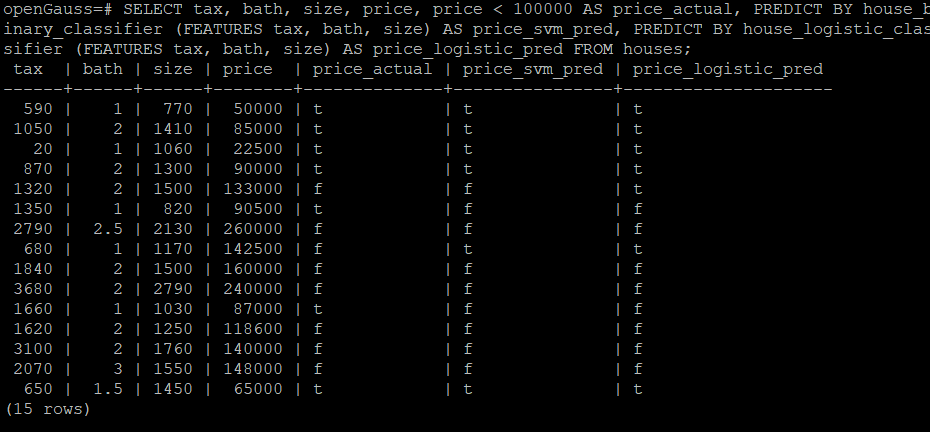
任务二：观察新模型的信息，将执行结果截图。

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：这两个的的预测目标变量类型不同，回归问题是连续变量，分类问题是离散变量；回归问题是定量问题，分类问题是定性问题；同时，回归与分类的根本区别在于输出空间是否为一个度量空间。

实践思考题2：什么是SVM算法？

答：Support Vector Machine，支持向量机，主要被用来解决模式识别领域中的数据分类问题，是一种有监督学习算法。

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：准确率（Accuracy）、精确率（Precision）、召回率（Recall）、P-R曲线（Precision-Recall Curve）、F1 Score、混淆矩阵（Confuse Matrix）、ROC、AUC；

准确率是分类问题中最为原始的评价指标，准确率的定义是预测正确的结果占总样本的百分比。

精准率（Precision）又叫查准率，它是针对预测结果而言的，它的含义是在所有被预测为正的样本中实际为正的样本的概率。

召回率（Recall）又叫查全率，它是针对原样本而言的，它的含义是在实际为正的样本中被预测为正样本的概率。

P-R曲线（Precision Recall Curve）正是描述精确率/召回率变化的曲线，P-R曲线定义如下：根据学习器的预测结果（一般为一个实值或概率）对测试样本进行排序，将最可能是“正例”的样本排在前面，最不可能是“正例”的排在后面，按此顺序逐个把样本作为“正例”进行预测，每次计算出当前的P值和R值

在一些场景下要兼顾精准率和召回率，最常见的方法就是F-Measure，又称F-Score。F-Measure是P和R的加权调和平均

混淆矩阵（Confusion Matrix）又被称为错误矩阵，通过它可以直观地观察到算法的效果。它的每一列是样本的预测分类，每一行是样本的真实分类（反过来也可以），顾名思义，它反映了分类结果的混淆程度。

AUC(Area Under Curve)又称为曲线下面积，是处于ROC Curve下方的那部分面积的大小。对于ROC曲线下方面积越大表明模型性能越好，于是AUC就是由此产生的评价指标

ROC（Receiver Operating Characteristic）曲线，又称接受者操作特征曲线。该曲线最早应用于雷达信号检测领域，用于区分信号与噪声。后来人们将其用于评价模型的预测能力。

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：残差平方和，决定系数，校正决定系数；

残差平方和：根据n个观察值拟合适当的模型后，余下未能拟合部份(ei=yi一y平均)称为残差，其中y平均表示n个观察值的平均值，所有n个残差平方之和称误差平方和。

决定系数：在线性回归中，回归平方和与总离差平方和之比值，其数值等于相关系数的平方。

校正决定系数：决定系数R可以用来评价回归方程的优劣，但随着自变量个数的增加，R将不断增大。