openGauss AI特性创新实践课



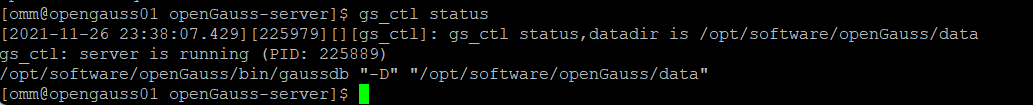
华为技术有限公司

# 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：

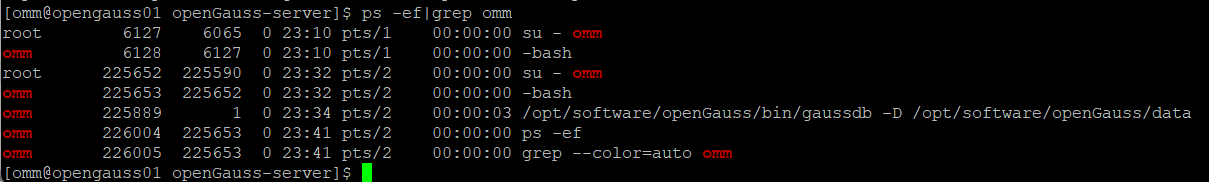
任务一：数据库状态验证

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. 源码安装在编译安装过程，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。

2. 数据库在编译期间需要配置

3. 数据库需要统一安装路径

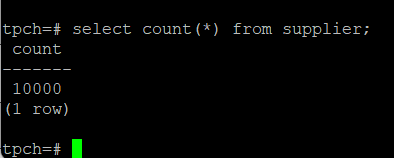
4. 编译安装可以针对特定的硬件进行优化，以获得更好的性能表现。

# 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：

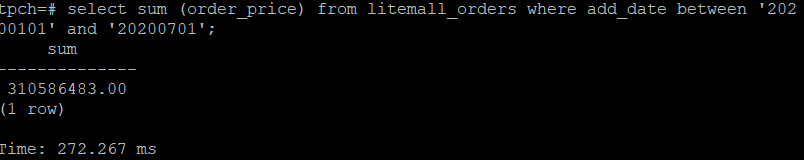
select count(\*) from supplier;;



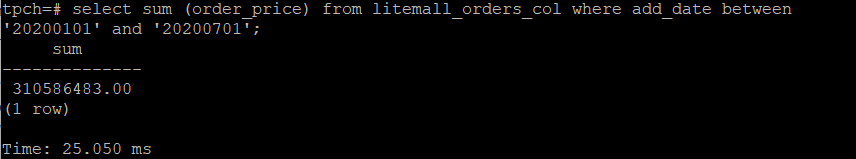
任务二：行存表与列存表执行效率对比

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';

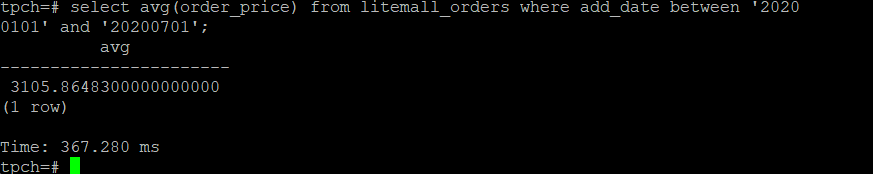


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

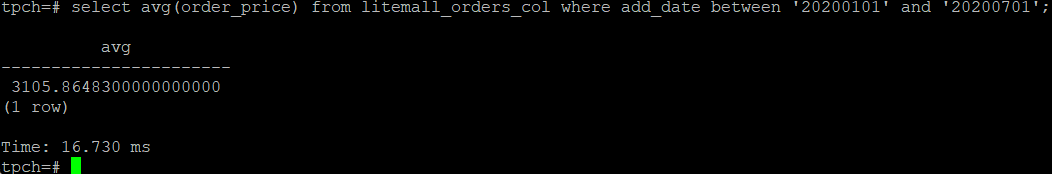


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

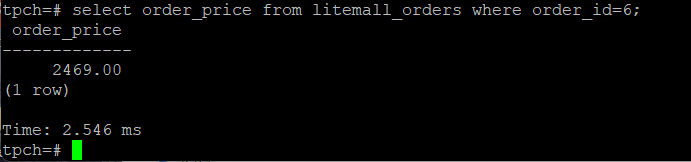


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

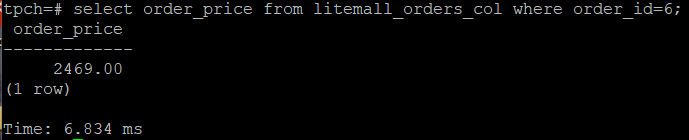


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

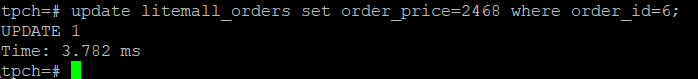


select order\_price from litemall\_orders\_col where order\_id=6;

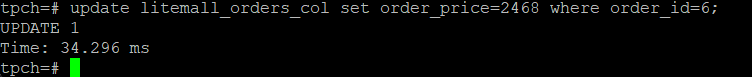


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

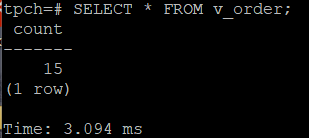
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



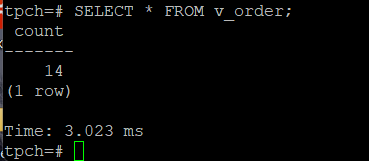
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



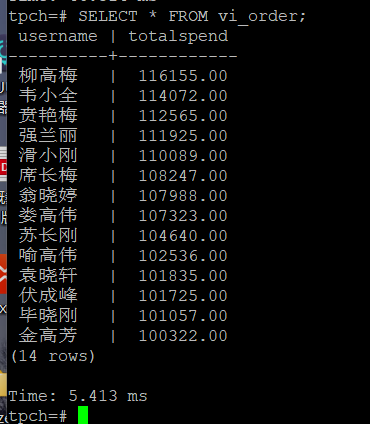
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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？

1. 读取数据时，行存储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，就会存在冗余列，出于缩短处理时间的考量，消除冗余列的过程通常是在内存中进行的。
2. 列存储每次读取的数据是集合的一段或者全部，不存在冗余性问题。

两种存储的数据分布。由于列存储的每一列数据类型是同质的，不存在二义性问题。这使数据解析变得十分容易。相比之下，行存储则要复杂得多，因为在一行记录中保存了多种类型的数据，数据解析需要在多种数据类型之间频繁转换，这个操作很消耗CPU，增加了解析的时间。所以，列存储的解析过程更有利于分析大数据。

根据实验结果，在进行修改和选择语句时，行存储的数据库效率更高；然而在进行求和，求平均值等类似操作时，列存储数据库速度更快。

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。**

创建全量物化视图语法和CREATE TABLE AS语法一致。即当底层表发生任何改变时，如果想让物化视图与底层表一致，需要重新基于底层表来构造数据。相当于把旧的数据做truncate操作，然后重新生成新数据。

全量刷新代价较大，即使底层表只增加了一条记录，或者更新、删除了一条记录，都会对底层表进行全量分析来构造新数据。

**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。**

当底层表发生改变时，不需要对旧的数据做truncate操作，而是在旧数据的基础上，只对新的，修改的数据做增量的刷新。

1. 增量刷新的速度优于全量刷新，因为该方式不需要基于全量的底层表进行数据的构造，而只是针对改变的部分。但增量刷新需要一定的代价，需要物化视图日志的支持。而且即使有物化视图日志的支持，也不是所有的视图，或者底层表进行了任何操作后都能快速刷新。

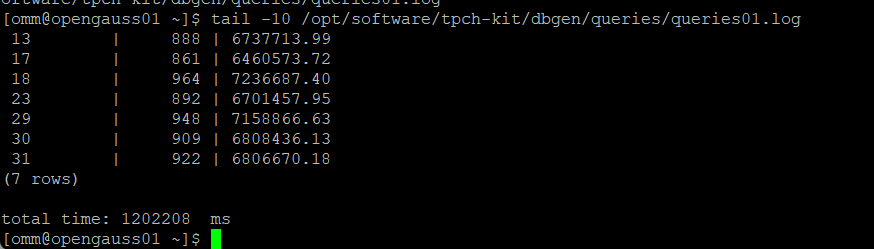
2. 与全量创建物化视图不同在于目前增量物化视图所支持场景较小，目前物化视图创建语句仅支持基表扫描语句或者UNION ALL语句。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

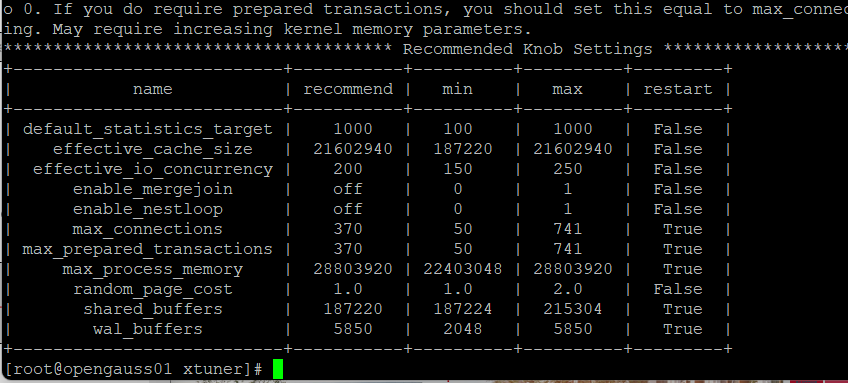
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

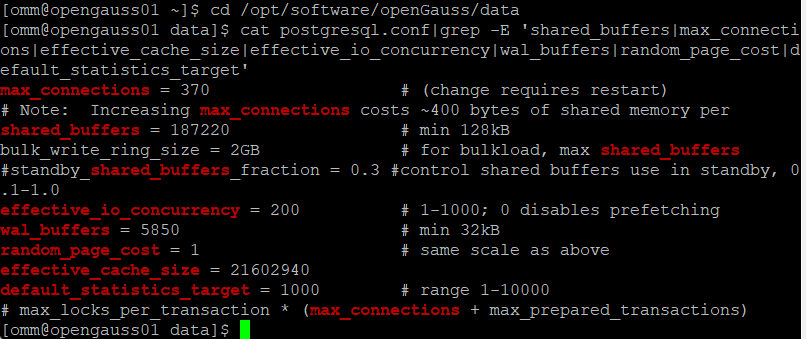
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

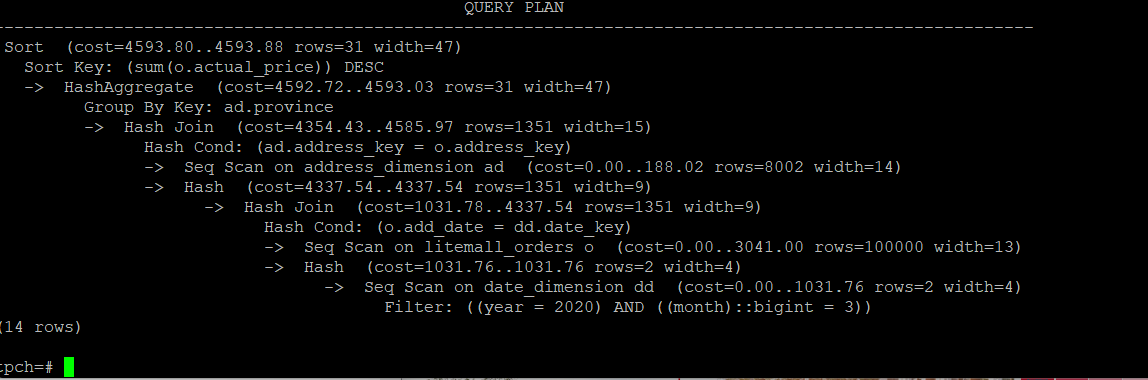
AND o.add\_date = dd.date\_key

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

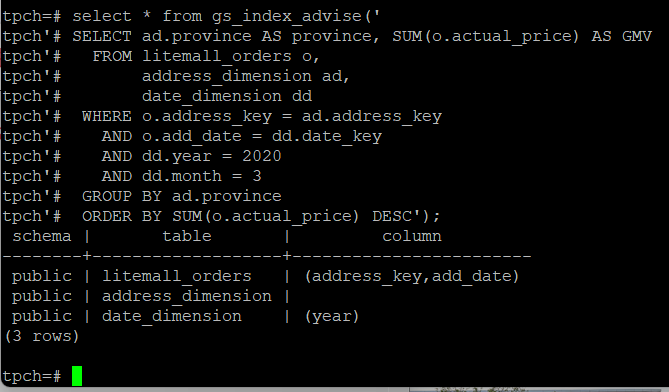
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

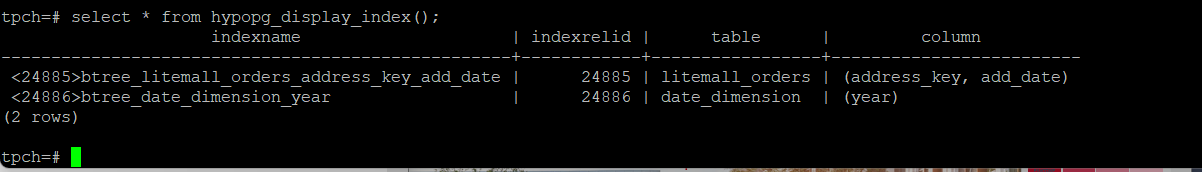
GROUP BY ad.province

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。

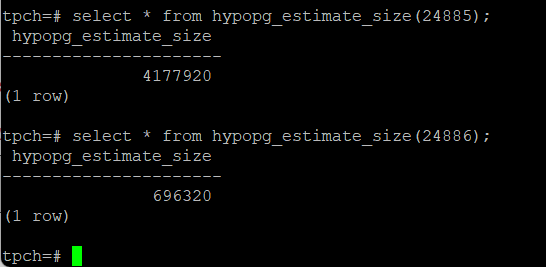
select \* from hypopg\_display\_index();



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(24885);

select \* from hypopg\_estimate\_size(24886);



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

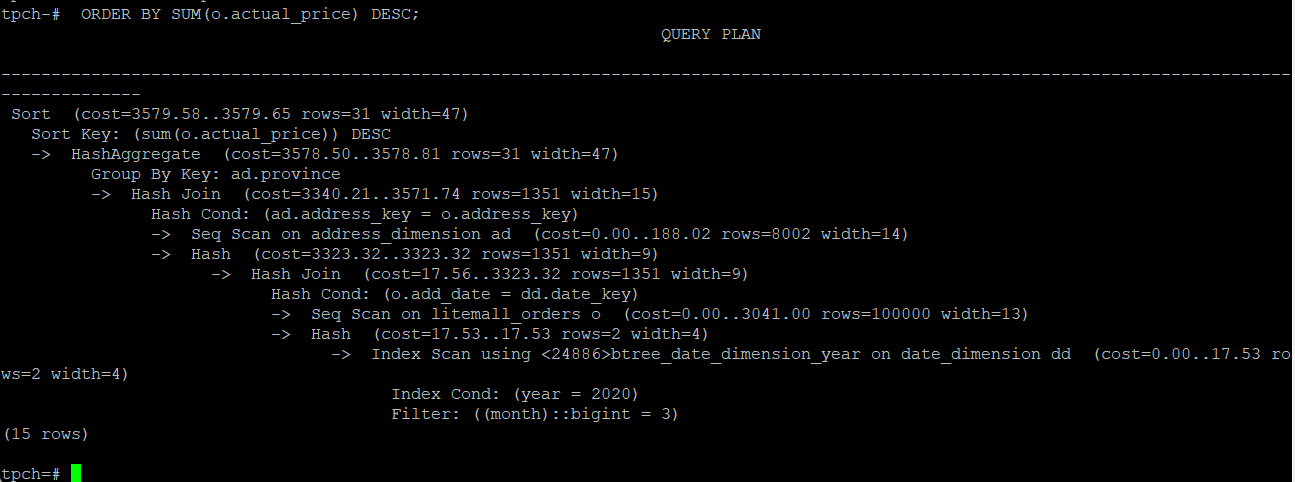
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

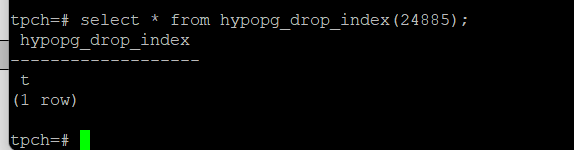
GROUP BY ad.province

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



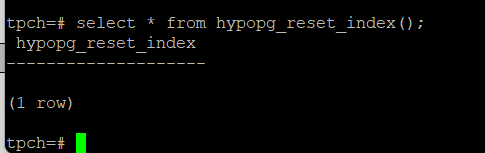
6. 删除某一个索引虚拟列，将执行结果截图。

select \* from hypopg\_drop\_index(24885);



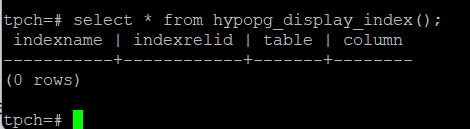
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

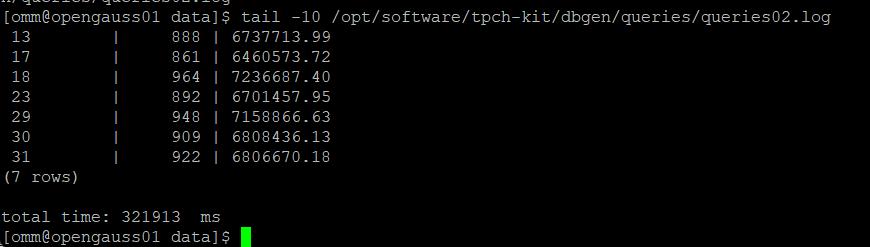
select \* from hypopg\_display\_index();



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：

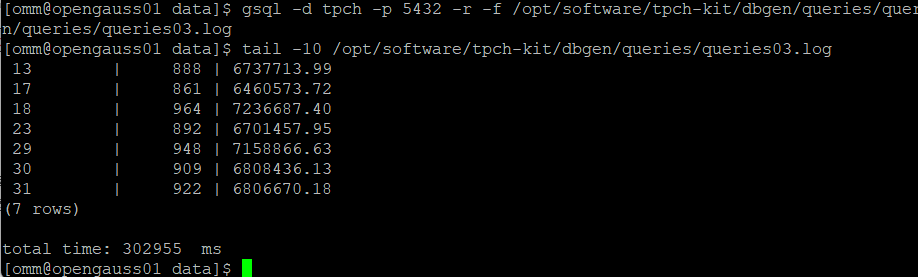
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？**

X-Tuner优化了name栏中的内容例如：effective\_cache\_size,effective\_io\_concurrency、shared\_buffers、max\_connections、wal\_buffers、random\_page\_cost、default\_statistics\_target,可以提高数据库运行效率，减少资源消耗。

**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？**

1. 索引的使用对执行SQL的好处：

创建索引可以大大提高系统的性能。

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，可以大大加快 数据的检索速度，这也是创建索引的最主要的原因。

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。

第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，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。

第五，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。

2. 优化数据库的方面：

1. 架构优化：在高并发的场景下对架构层进行优化其效果最为明显，常见的优化手段有：分布式缓存，读写分离，分库分表等，每种优化手段又适用于不同的应用场景。

2. 硬件优化：更换数据库的硬盘等手段

3. DB优化：SQL执行慢有时候不一定完全是SQL问题，手动安装一台数据库而不做任何参数调整，再怎么优化SQL都无法让其性能最大化。要让一台数据库实例完全发挥其性能，就得先优化数据库的实例参数。

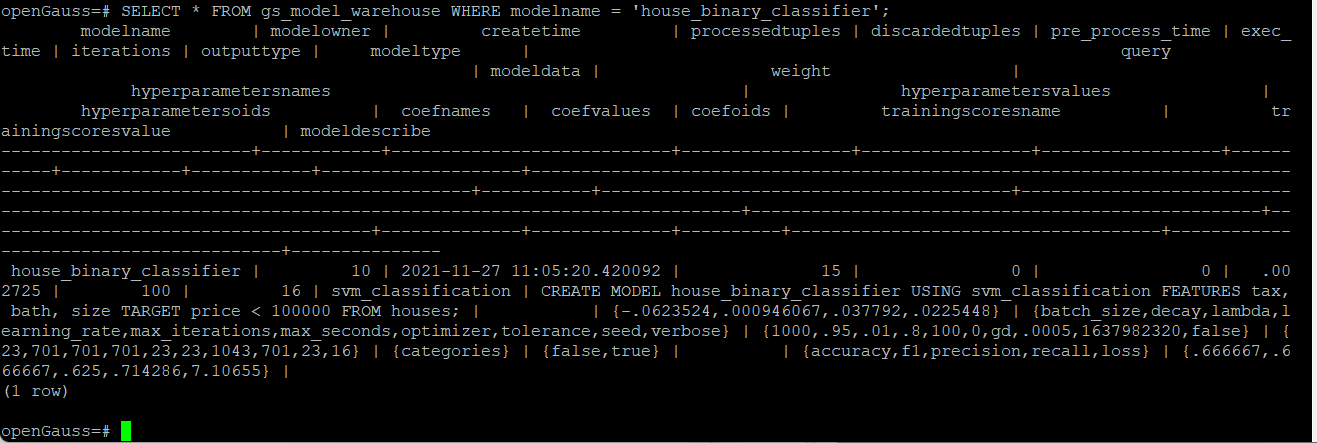
4. SQL优化：通过给查询字段添加索引或者改写SQL提高其执行效率，一般而言，SQL编写有以下几个通用的技巧：

合理使用索引，使用UNION ALL替代UNION，避免select \* 写法，JOIN字段建议建立索引等等。

# 关卡四、openGauss的DB4AI特性应用

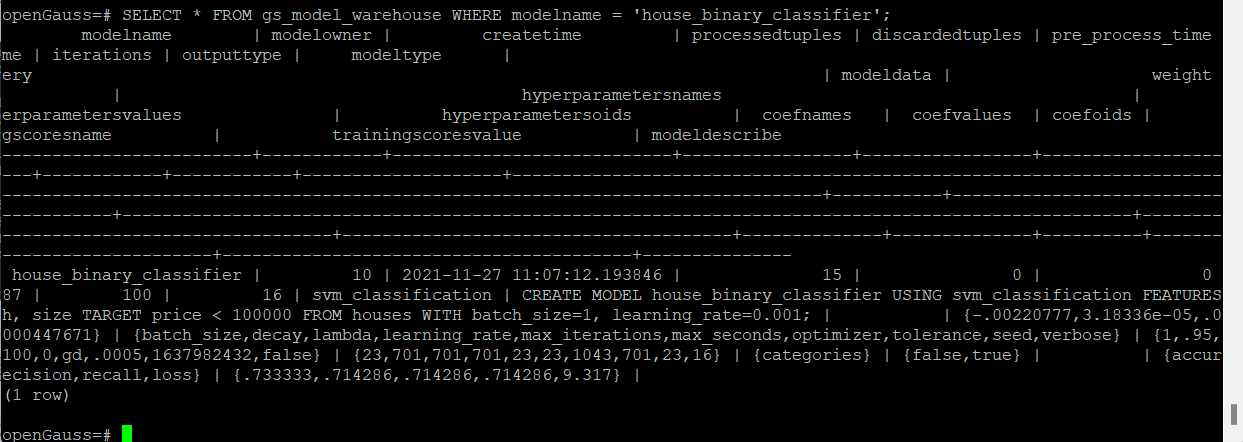
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



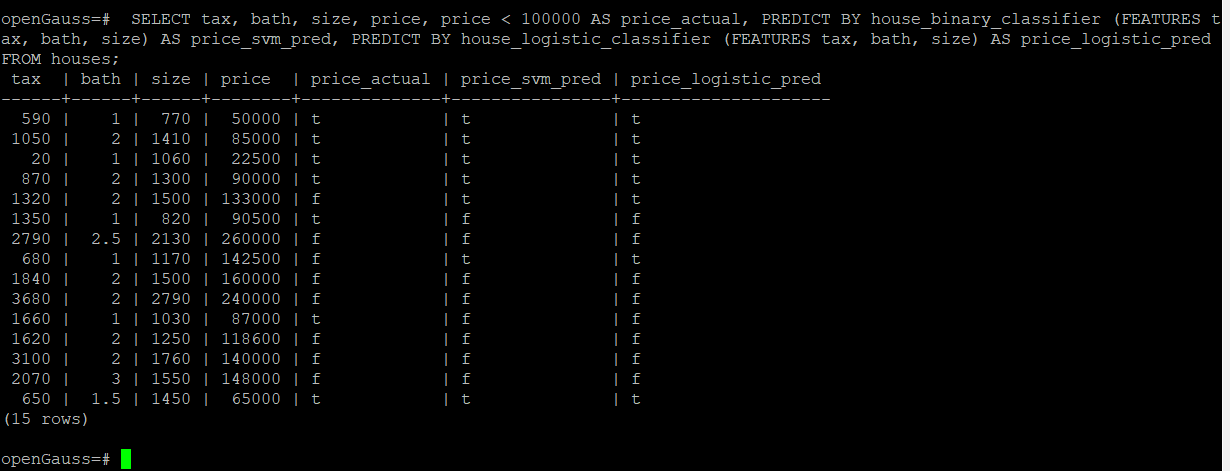
任务二：观察新模型的信息，将执行结果截图。

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？**

定量输出称为回归，或者说是连续变量预测；  
定性输出称为分类，或者说是离散变量预测。

根本上来说分类是关于预测标签，而回归是关于预测数量。

**实践思考题2：什么是SVM算法？**

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？**

准确率、精准度、召回率、F1值，根据混淆矩阵有：

准确率 = 算法分类正确的数据个数/输入算法的数据的个数A=(TP+TN)/(T+F)

精确率 = 预测为正的样本占所有正样本的比重P=TP/(TP+FP)

召回率 = 正确预测的数据在总样本中的比重R=TP/(TP+FN)=TP/T

F1值 = 精确率和召回率的兼顾指标，是精确率和召回率的调和平均数。调和平均数的性质，只有当精确率和召回率二者都非常高的时候，它们的调和平均才会高。如果其中之一很低，调和平均就会被拉得接近于那个很低的数 F1 = 2\*precision\*recall / precision+recall.

**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？**

评价指标有：MSE，RMSE，MAE、R-Squared

RMSE：均方根误差衡量观测值与真实值之间的偏差。常用来作为机器学习模型预测结果衡量的标准。

MSE：均方误差MSE是真实值与预测值的差值的平方然后求和平均。

通过平方的形式便于求导，所以常被用作线性回归的损失函数。

MAE：平均绝对误差是绝对误差的平均值。可以更好地反映预测值误差的实际情况。

SD：标准差方差的算术平均根。用于衡量一组数值的离散程度。