openGauss AI特性创新实践课



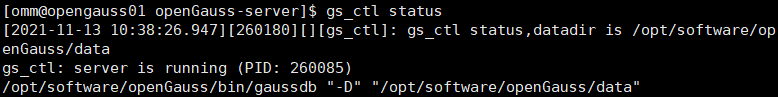
华为技术有限公司

# 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：

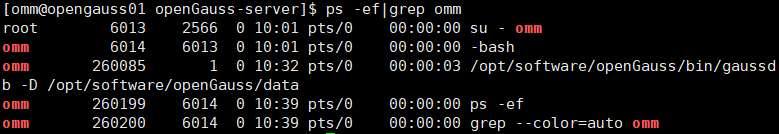
任务一：数据库状态验证

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：

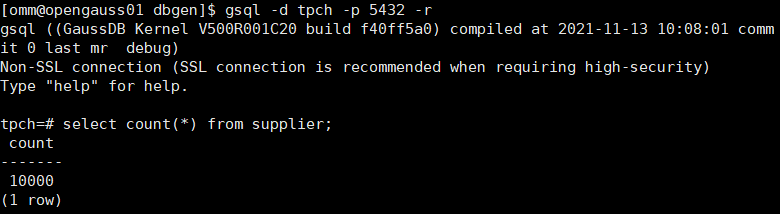
1. 因为通过源码编译安装数据库比较灵活，可以自定义安装的大部分参数，包括安装路径和安装的数据库版本，通过源码编译安装还可以定义编译参数和其他配置，因此通过源码编译安装数据库方便实现自己的配置需求。
2. 安装数据库所需要的步骤：
3. 使用根用户登录ECS
4. 创建openGauss数据库的安装用户及其属组
5. 创建openGauss源码存放及openGauss安装路径
6. 下载、解压第三方编译库
7. 下载openGauss源码
8. 上传cmake包并解压
9. 使用yum安装依赖包
10. 替换python版本
11. 修改/opt/software路径的用户所属组及权限
12. 切换至安装用户、设置其环境变量、使环境变量生效
13. 进入openGauss源码下，生成配置文件
14. 执行make命令进行编译
15. 执行make install完成安装数据库

# 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：

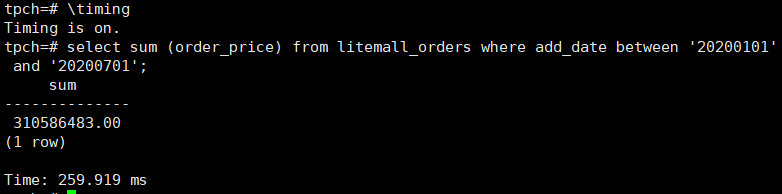
select count(\*) from supplier;;



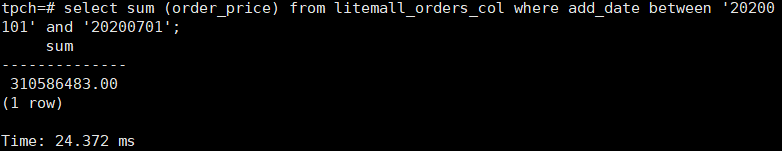
任务二：行存表与列存表执行效率对比

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';

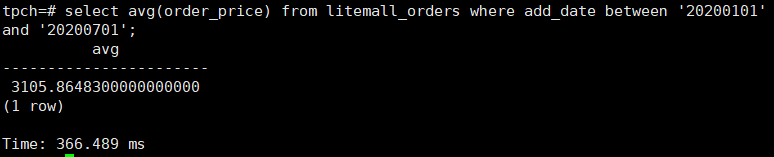


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

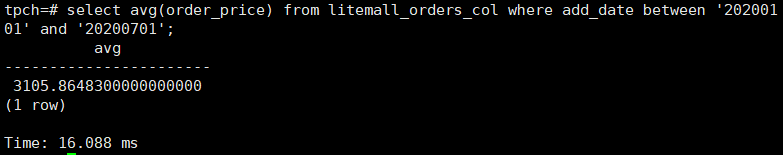


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

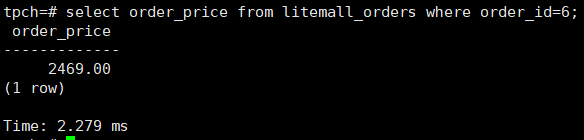


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

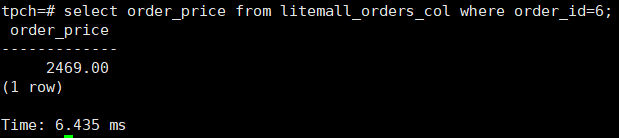


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

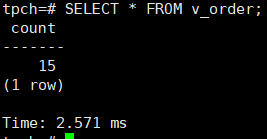
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



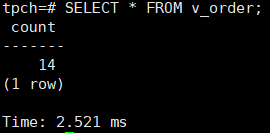
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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：

1. a、行存表和列存表在存储介质中的连续存储形式不同，行存表以行数据为基本逻辑存储单元进行存储，一行中的数据在存储介质中以连续存储形式存在，列存表以列数据为基本逻辑存储单元进行存储，一列中的数据在存储介质中以连续存储形式存在，因此在执行SQL语句时，行存表和列存表的读写方式不同执行时间也不同；
2. 执行写类型的SQL时行存表的速度一般会比列存表的速度快（按行写入），执行读类型的SQL时列存表的速度一般比行存表的速度快，因为在读取特别是搜索过程中，行存表通常要一行一行读出数据产生冗余列，而列存储一次读取集合的一段不产生冗余问题

实践思考题2：全量物化视图与增量物化视图有哪些差别？

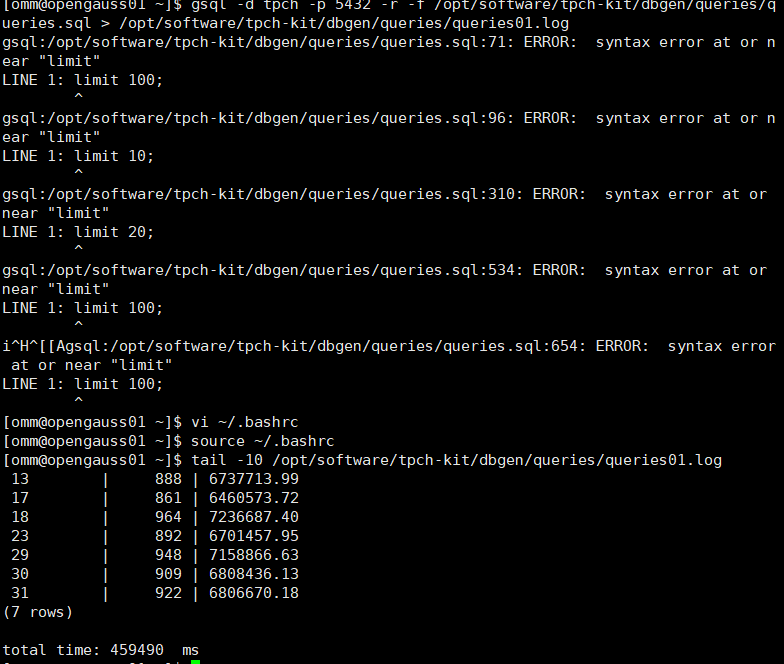
答：全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新，但比目前增量物化视图所支持的场景更大；增量物化视图可以对物化视图增量刷新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新，与全量物化视图相比目前增量物化视图所支持场景较小

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

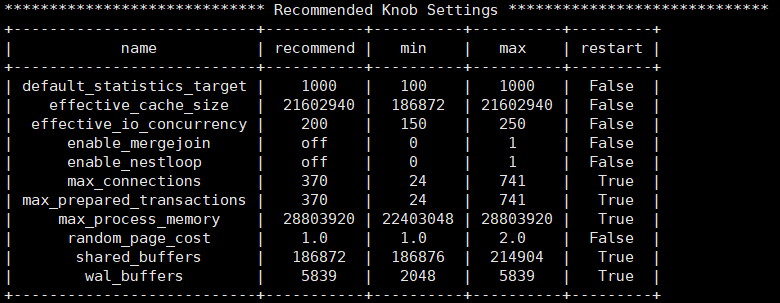
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

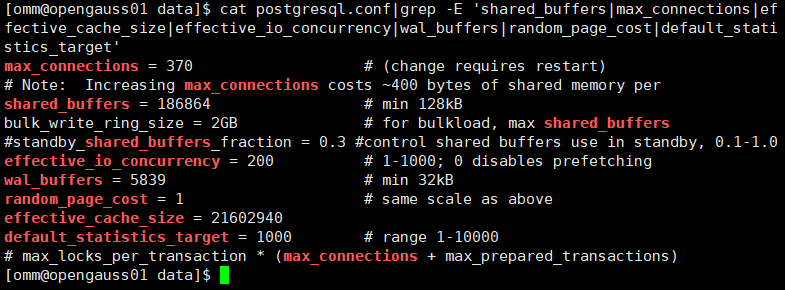
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

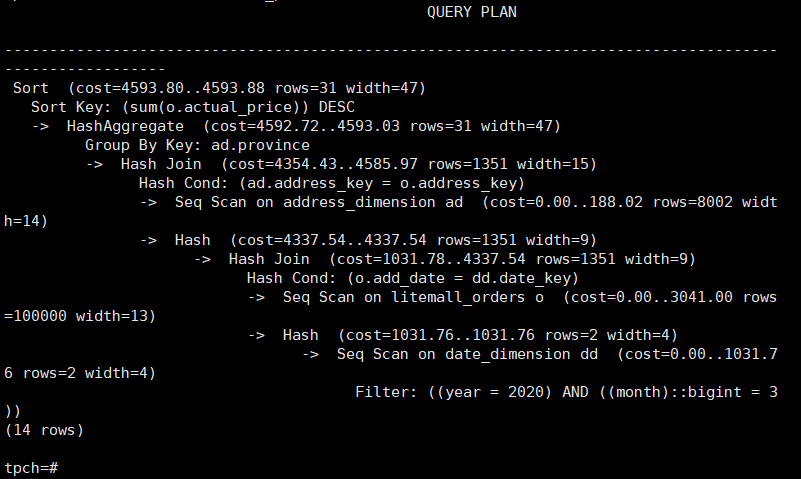
AND o.add\_date = dd.date\_key

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

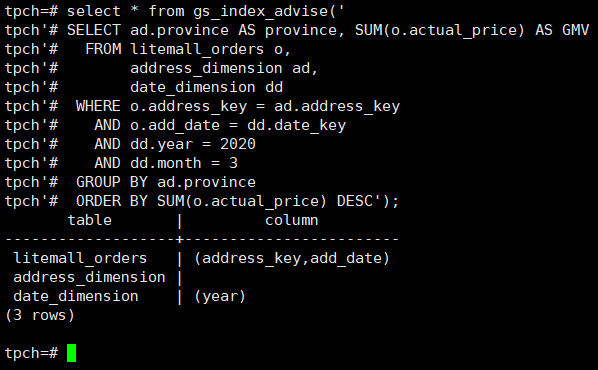
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

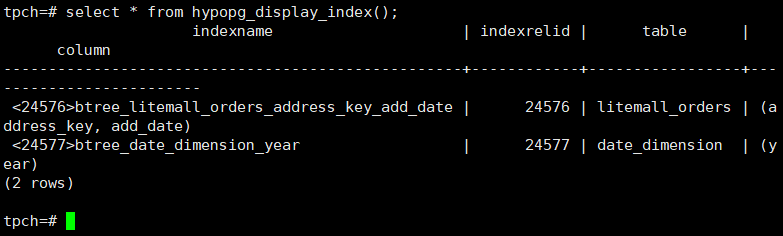
GROUP BY ad.province

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。

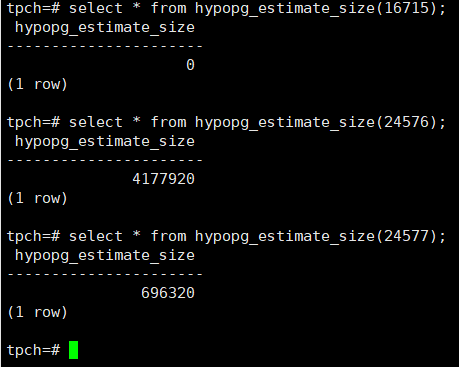
select \* from hypopg\_display\_index();



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

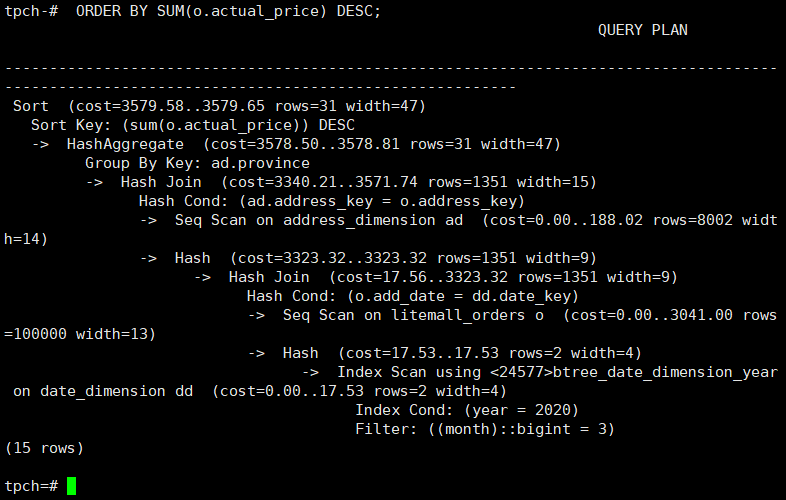
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

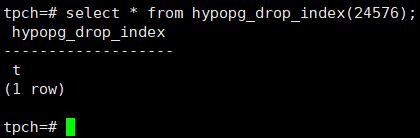
GROUP BY ad.province

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



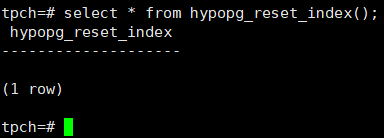
6. 删除某一个索引虚拟列，将执行结果截图。

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



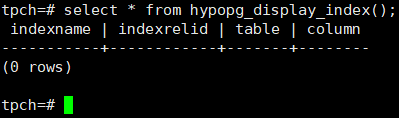
7. 删除所有索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

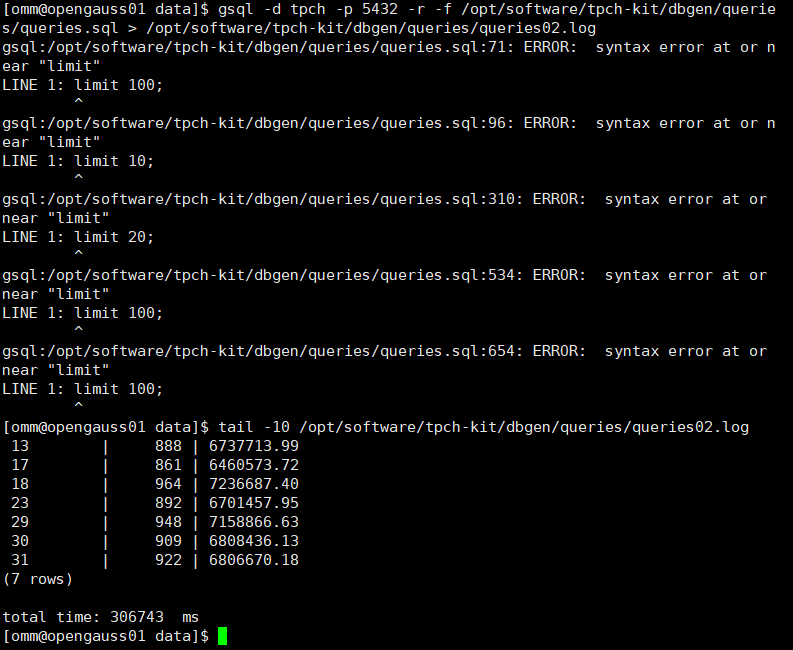
select \* from hypopg\_display\_index();



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：

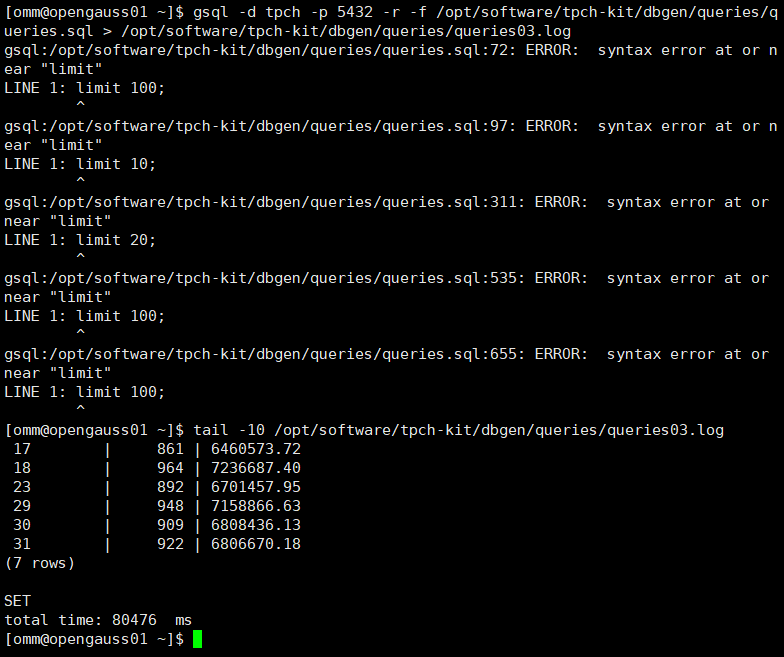
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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：

1. X-Tuner进行了优化的参数有：shared\_buffers、max\_connections、effective\_cache\_size、effective\_io\_concurrency、wal\_buffers、random\_page\_cost、default\_statistics\_target
2. 这些参数是数据库执行任务时涉及的系统参数，通过优化这些参数让数据库运行时充分运用系统资源，提高数据库运行性能

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？

答：

1. 使用索引对执行SQL的好处有：大大提高系统的性能，提高数据的检索速度，加快表连接操作的速度，通过创建唯一性索引保证数据库中每一行数据的唯一性
2. 其它的优化方面可以考虑两个方向：物理优化和代数优化。

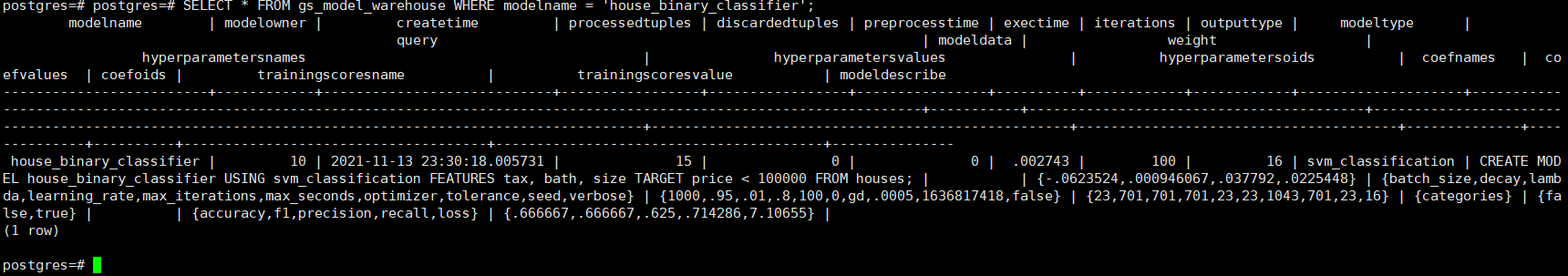
物理优化：可以对存储路径选择优化、对系统参数进行优化、提高运行并行度等

代数优化：可以建立查询树进行SQL语句优化、修改表的定义适当增加或减少冗余字段、建立分区表、改变范式级别拆分或合并表、变换SQL语句使用的关键字和函数（例如in改用exists等）、建立临时表等

# 关卡四、openGauss的DB4AI特性应用

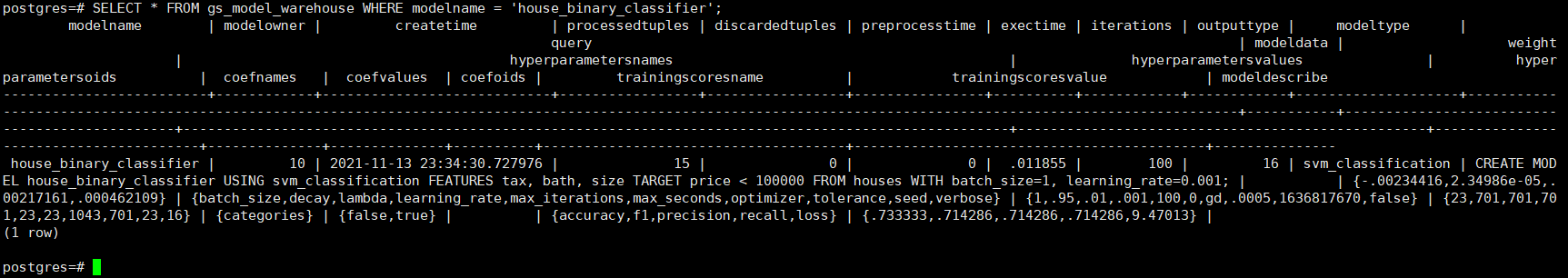
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



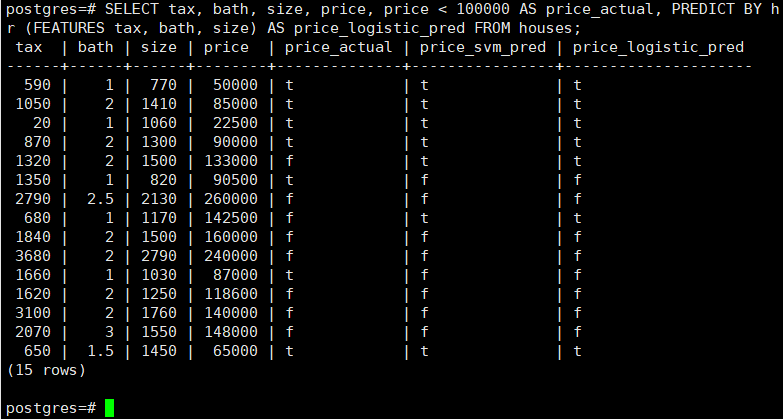
任务二：观察新模型的信息，将执行结果截图。

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：分类模型是给定待预测数据得到结果类别，其中结果的类别只能在某一分类集合中取值，这个分类集合的元素是离散值；回归模型是给定待预测数据得到结果值，其中结果的取值范围在某些连续区间上。回归模型也可以用于分类任务上，比如logistic回归算法，其待求函数是，通过输入x向量，得到结果y的值在(0,1)之间，再判断y更偏向于0还是1得到一个分类（二分类）

实践思考题2：什么是SVM算法？

答：SVM即支持向量机，是一种二分类模型，属于监督学习算法，其基本模型是定义在特征向量上的大间隔分类器，基础的SVM是线性分类器（无核函数或线性内核函数，直接使用预测），增添了核函数（比如高斯内核函数）后SVM可支持非线性预测。

大间隔分类器中的大间隔是指样本到决策界限的间距，大间隔分类器通常使用尽可能大的间距分隔不同种类的样本，即大间隔分类器从众多分类界限中挑选出一个最好的界限，挑选依据是与超平面（界限）最近的样本点的距离，然后要让这个距离尽可能大。

核函数的使用使得本来线性不可分的数据通过扩展高维的方式后变得线性可分，再运用线性分类器像逻辑回归一样进行分类。

SVM与逻辑回归实际上相类似，大部分能使用SVM和逻辑回归的问题中结果都比较相近。

SVM的优化问题是凸优化问题，因此好的SVM一般能找到接近全局最优的解。

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：分类问题的评价指标有：准确率、精确率、召回率、**F-Score、ROC曲线及其AUC值、PR曲线等。**

准确率：**对于给定的数据，分类正确的样本数占总样本数的比例**

**精确率：**在预测为正类的样本中真正类所占的比例

召回率：在所有的正类中被预测为正类的比例

F1-Score：精确率和召回率的调和平均

ROC曲线和AUC：ROC纵轴是正类的召回率、横轴是1-负类的召回率，AUC是ROC曲线与横轴、直线FPR = 1围成的面积，ROC曲线越偏向左上方，AUC越大模型效果越好

PR曲线：准确率-召回率曲线，在极度不平衡的数据下，PR曲线可能比ROC曲线更实用

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：回归问题有哪些评价指标有：MSE均方误差、RMSE、MAE平均绝对误差、MAPE、MSPE

MSE均方误差：描述了样本真实值与预测值差方求和的平均值，均方误差最小化优化模型的方法称为“最小二乘法”

RMSE：对MSE开平方根

MAE平均绝对误差：真实值与预测值差的绝对值和求平均

MAPE：求所有样本真实值与预测值差绝对值与真实值的比例的和求平均

MSPE：求所有样本的真实值与预测值的差与真实值的比例求平方的和求平均