openGauss AI特性创新实践课

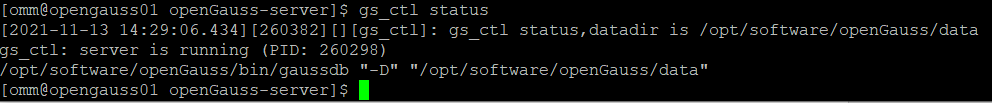


华为技术有限公司

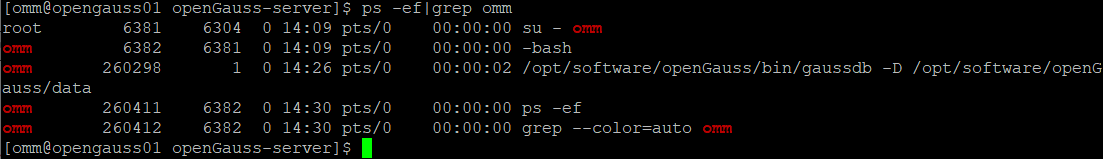
# 关卡一、openGauss数据安装及基本操作

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）

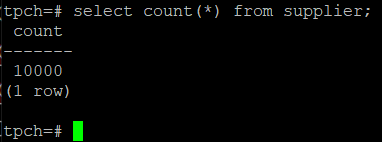
实验思考题：为什么需要通过源码编译，安装数据库？

答：通过源码编译安装数据库，可以自定义地调整数据库使得运行在一个更适合自己机器的设置之下；同时自定义的安装路径也便于再换盘、重装系统等情况下直接拷贝使用。而面对开源数据库的时候，源码编译安装下也便于用户针对自己的需求修改、使用源码。

# 关卡二、openGauss数据导入及基本操作

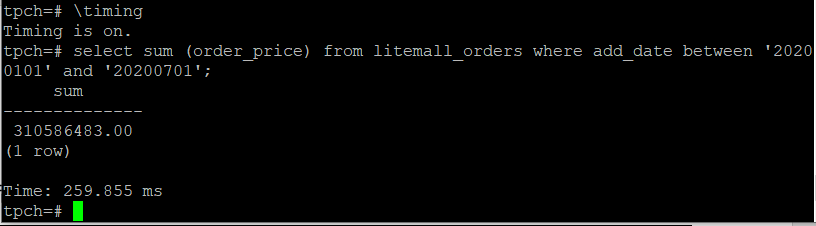
任务一：数据初始化验证

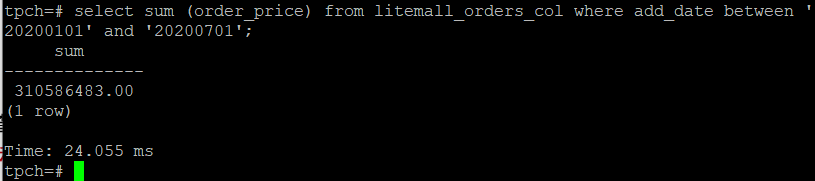
1. 查询supplier表的行数，并将结果进行图：

select count(\*) from supplier;;

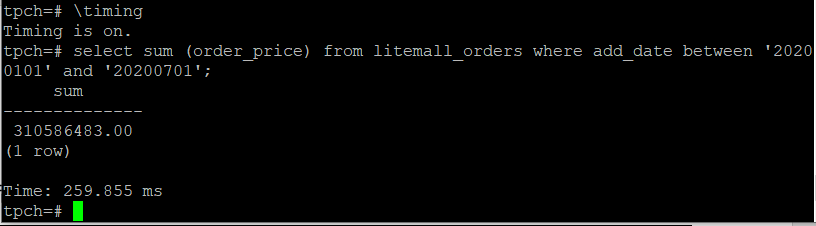
任务二：行存表与列存表执行效率对比

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

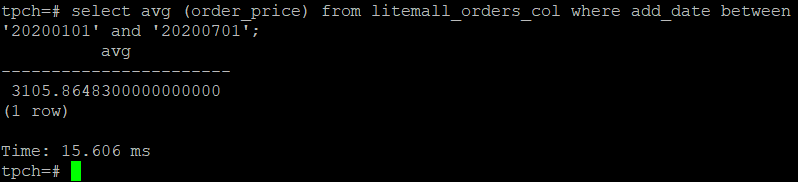
select sum (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';

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

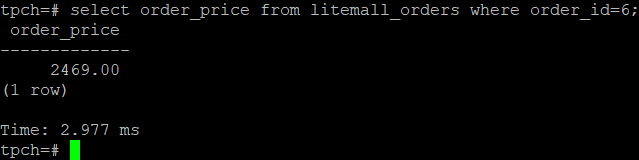
2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

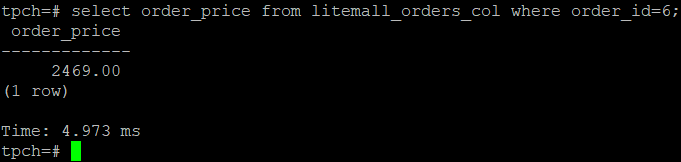
select avg (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';

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

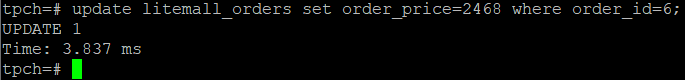


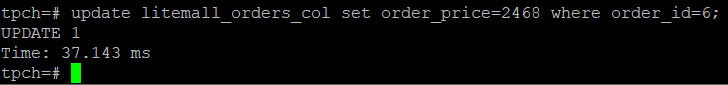
3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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;

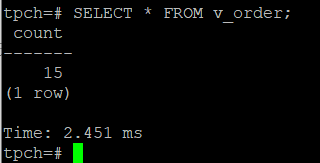
任务三：物化视图的使用

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：

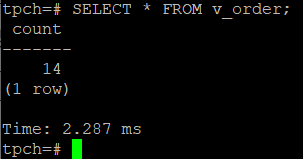
SELECT \* FROM test\_view;



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;

第二次插入后的查询：

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：行存表与列存表存储数据的结构有区别，行存表中的一组数据按照规则拆成多列排成一行，而列存表中一组数据拆成多行存为一列。因此前者的物理存储格式中同一组数据都相邻，后者则是同一类型的数据都相邻。物理存储结构的差异导致了对表进行操作的时候产生效率的区别。

读取数据时，行存表会把整行数据读出，如果只需要一部分的数据，就会造成冗余浪费内存降低效率，而列存表则可以精确地读出需要的数据。

写入数据时，列存表要把数据拆开分别存入不同的行的末端，行存表可以直接写入数据，效率更高。

修改数据时，行存表只需定位出数据的位置写入一次即可，列存表需要定位的次数更多，效率低于行存表。

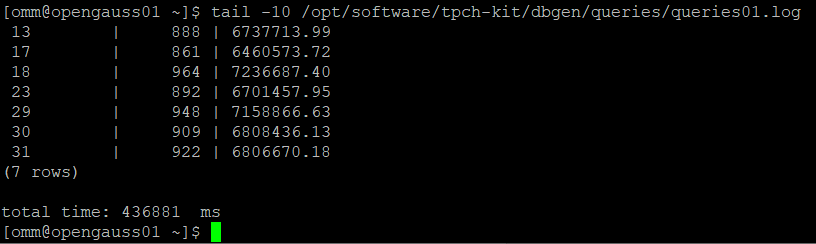
实践思考题2：全量物化视图与增量物化视图有哪些差别？

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。增量物化视图可以进行增量更新，但是目前支持场景较小。

# 关卡三、openGauss的AI4DB特性应用

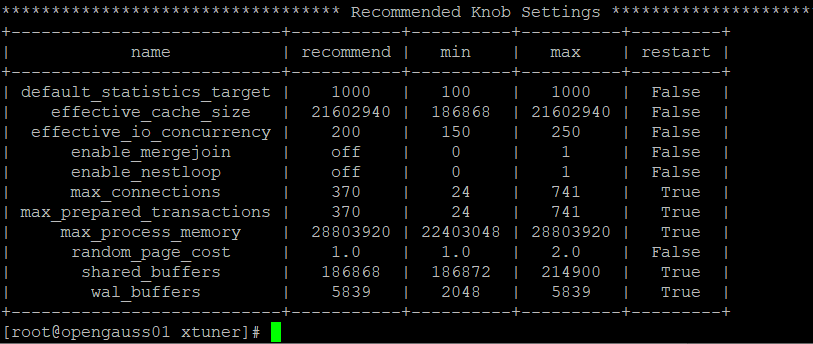
任务一：使用X-Tuner进行参数优化

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

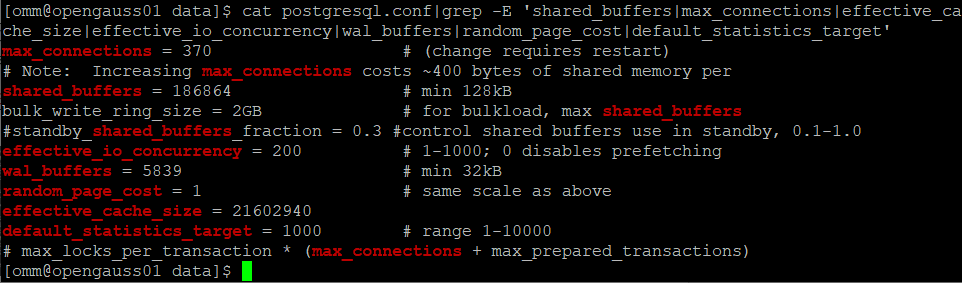
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

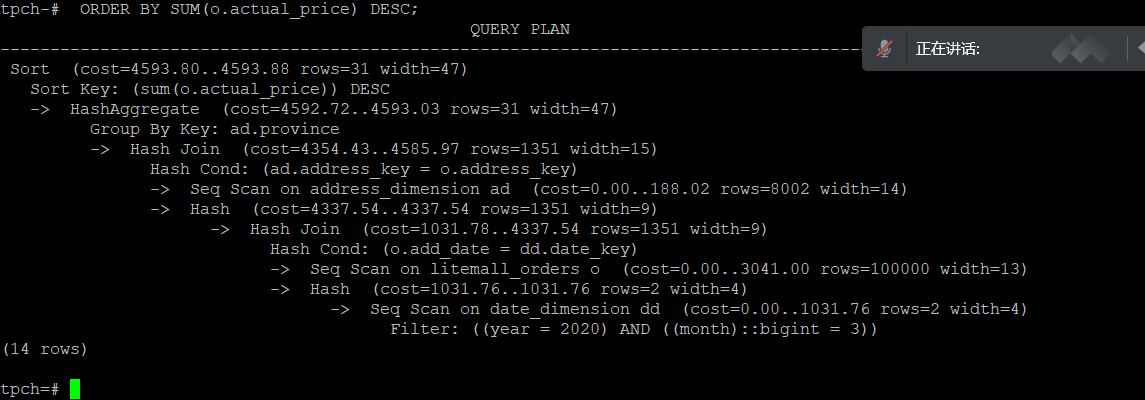
AND o.add\_date = dd.date\_key

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

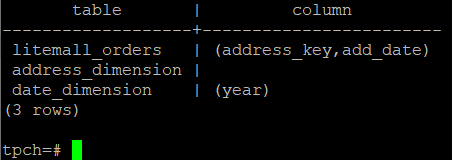
WHERE o.address\_key = ad.address\_key

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

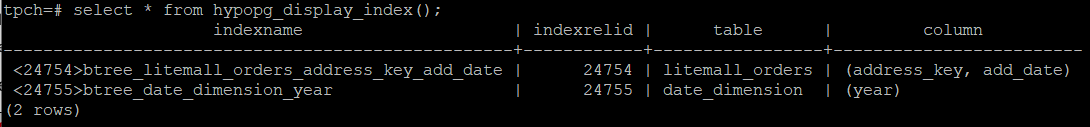
AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

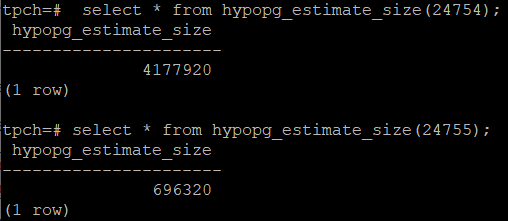
 ORDER BY SUM(o.actual\_price) DESC');

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

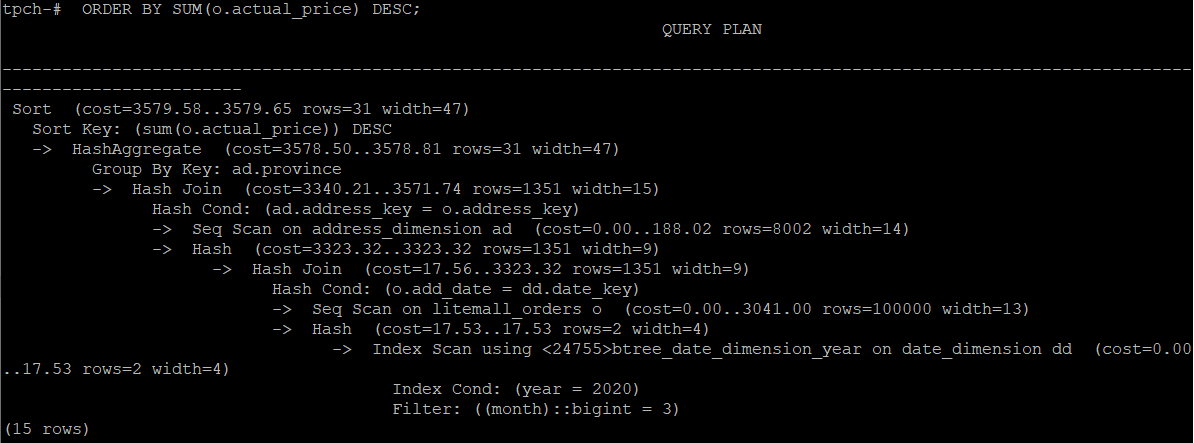
WHERE o.address\_key = ad.address\_key

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

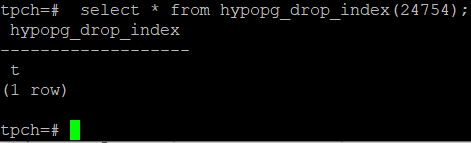
AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

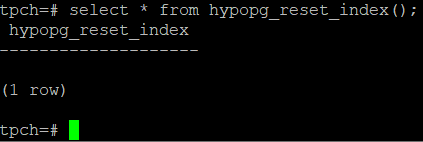
 ORDER BY SUM(o.actual\_price) DESC;

6. 删除某一个索引虚拟列，将执行结果截图。

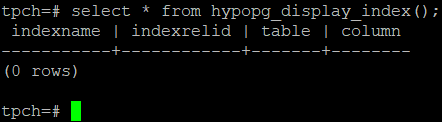
select \* from hypopg\_drop\_index(16715);

7. 删除全部索引虚拟列，将执行结果截图。

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

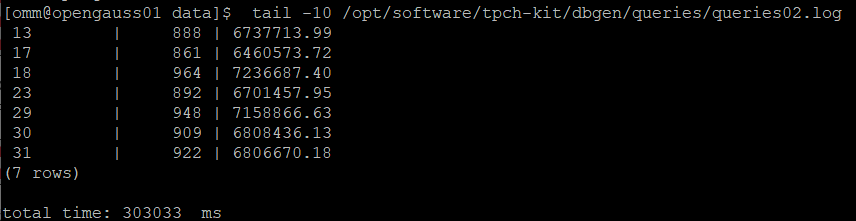


8. 查看索引虚拟列，将执行结果截图。

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. shared\_buffers：数据库在从磁盘中查询数据前，会先查找shared\_buffers的页，若命中则返回可以避免磁盘查找。调整该参数可以调节缓冲区的大小。通常来说该值应该高于默认值来获得更好的性能，如果服务器RAM>=1GB，建议该参数设置为系统内存的25%-40%之间 。
  2. max\_connections：最大连接数，应该根据服务器负载的上限进行设置。
  3. effective\_cache\_size：系统能提供的cache大小总和，该参数会被用到使用索引的成本考虑之中，值越大，使用索引的可能性越大。
  4. effective\_io\_concurrency：控制I/O性能的参数，为0时禁止I/O并发查询。
  5. wal\_buffers：WAL（预写日志）记录写入的缓冲区的大小，设置该值大一些可以在有大量并发连接的时候提供更好的性能。
  6. random\_page\_cost：用于决定使用索引的代价是否值得；如果系统Cache使用状况良好，那么random\_page\_cost可以适当降低。
  7. default\_statistics\_target：告诉数据库应该抽样多少数据来填充存储元数据的表。较大的值会增加ANALYZE所需的时间，但可能会提高计划者评估的质量

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：索引的好处：

1. 提高数据查询的效率
2. 保证数据的唯一性，避免冲突
3. 加速表与表之间的连接
4. 面对ORDER BY、GROUP BY子句可以显著提高效率

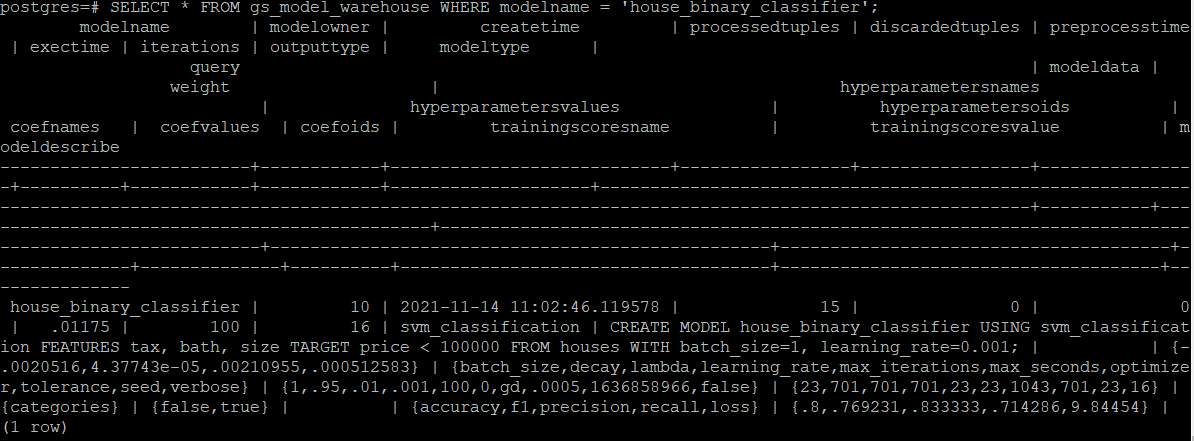
其他优化方法：

1. 精心选取合适的字段属性
2. 使用事务来进行多表、多语句的操作
3. 使用外键保证数据关联性

# 关卡四、openGauss的DB4AI特性应用

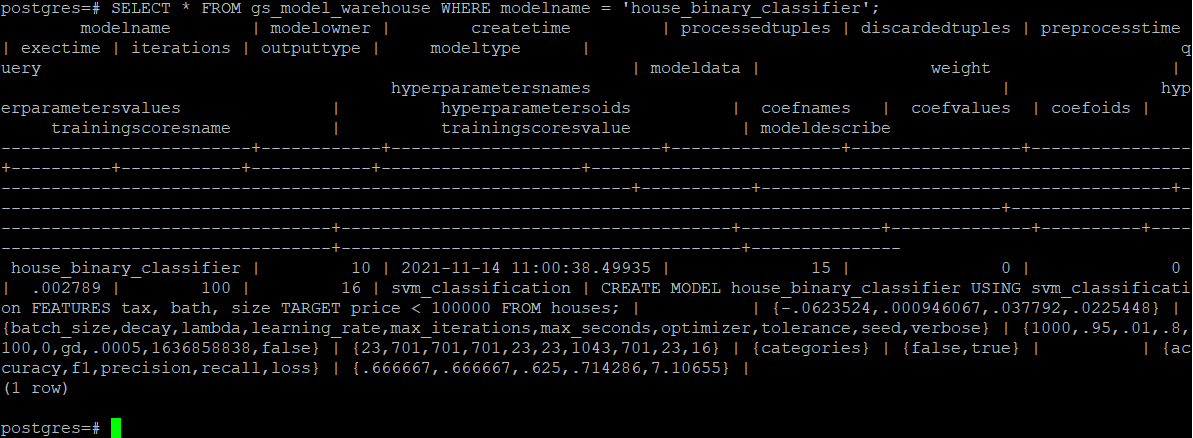
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



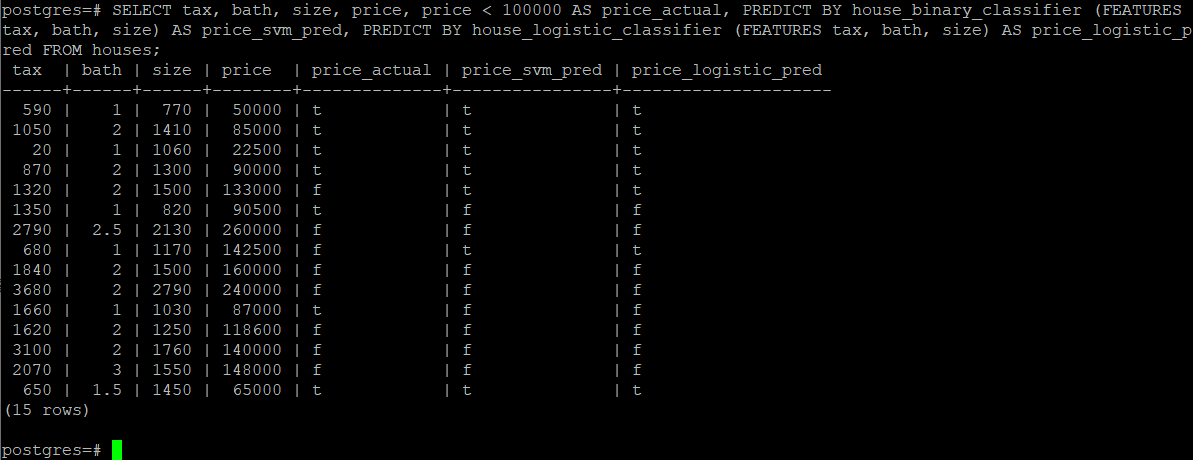
任务二：观察新模型的信息，将执行结果截图。

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：分类模型为定性输出，回归模型为定量输出。分类模型的输出相比回归模型的更加离散化；回归模型的输出更加连续化。前者更适合预测离散类标签的任务，后者更适合预测连续数量的任务。

实践思考题2：什么是SVM算法？

答：SVM算法（支持向量机），是基于统计学习理论的一种机器学习方法，通过寻求结构化风险最小来提高学习机泛化能力，实现经验风险和置信范围的最小化，从而达到在统计样本量较少的情况下，亦能获得良好统计规律的目的。

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：Accuracy：准确率

Precision：精确率

Recall：召回率

F-Measure（又称为F-Score）：精确率与召回率的综合考虑的参数（以一定的权衡）

Macro F1： 宏平均

Micro F1 ：微平均

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：MSE （Mean Squared Error）：均方误差

RMSE（Root Mean Squard Error）：均方根误差

MAE：平均绝对误差

SSE：和方差

R Squared：R方，线性回归的衡量标准