openGauss AI特性创新实践课



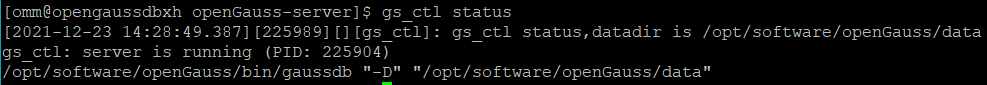
华为技术有限公司

# 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：

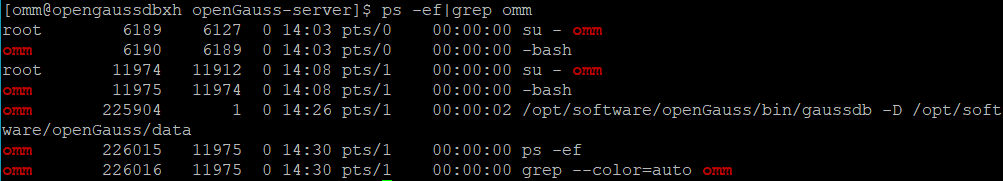
任务一：数据库状态验证

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：满足不同的运行平台，由于Linux发行的版本众多，而每个版本采用的软件或者内核版本都不一样，而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

安装步骤：1. 在服务器登录root用户，创建数据库的安装用户及其属组，并修改用户密码；

2. 创建数据库源码存放文件夹及其安装路径；

3. 下载第三方编译库并解压，下载数据库源码，并从本地上传cmake包后解压；

4. 用yum命令安装所需依赖包后替换python版本并检查

5. 修改用户所属组及其权限；

6. 切换为数据库的安装用户，并添加环境变量，然后令其生效

7. 在数据库源码安装路径下生成配置文件，并用make命令编译安装

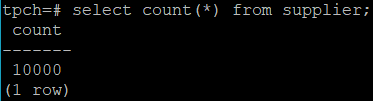
8. 初始化数据库并启动，登录后检查数据库版本，确认后安装结束

# 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，并将结果进行截图：

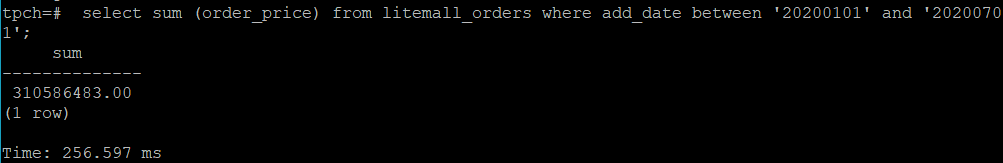
select count(\*) from supplier;



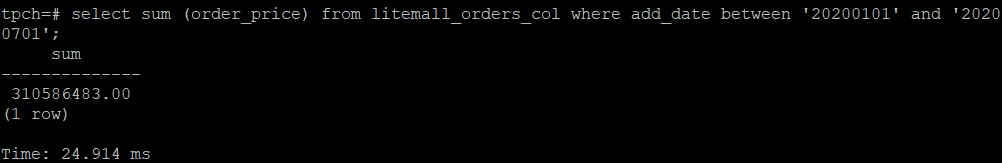
任务二：行存表与列存表执行效率对比

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';



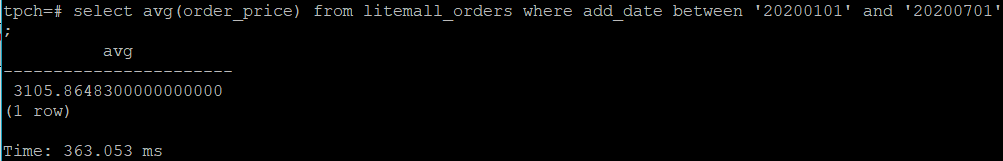
select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



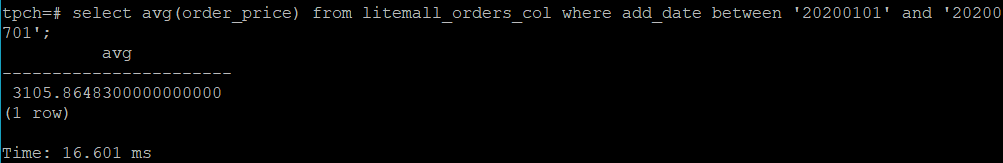
可以看出，在执行总和查询语句时行存表的运行时间接近列存表的十倍，因此在执行总和查询语句时列存表效率较高。

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';



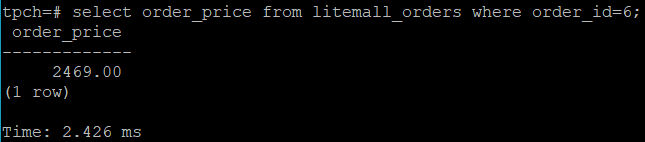
select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';



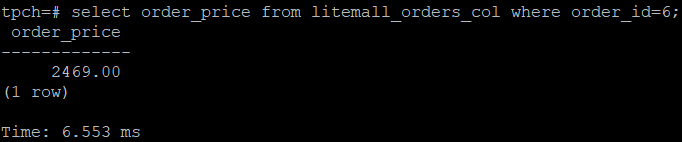
可以看出，在执行平均值查询语句时行存表的运行时间接近列存表的二十二倍，因此在执行平均值查询语句时列存表效率较高。

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;



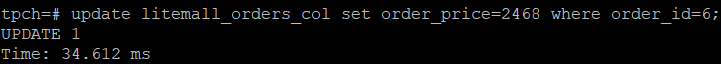
可以看出，在执行查询语句时行存表的运行时间比列存表的更短，因此在执行查询语句时行存表效率较高。

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;

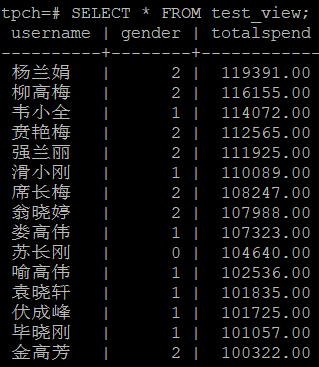


可以看出，在执行更新语句时列存表的运行时间接近行存表的六倍，因此在执行更新语句时行存表效率较高。

任务三：物化视图的使用

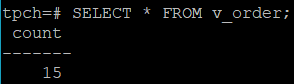
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



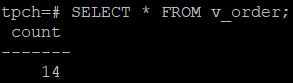
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

在数据写入时，行存表的写入是一次完成，而列存表由于需要把一行记录拆分成单列保存，写入次数明显比行存储多（意味着磁头调度次数多，而磁头调度是需要时间的，一般在1ms~10ms)，再加上磁头需要在盘片上移动和定位花费的时间，实际时间消耗会更大；但在数据读取时，行存表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，而列存表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。

因此，在执行简单的查询、更新语句时行存表效率更高；在执行求和、取平均等这种涉及聚合函数的查询语句时列存表效率更高。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

对于物化视图，它建立在基础表之上，当基础表的数据发生改变（增删改）时，物化视图需要进行刷新才能响应基础表的更改，而在刷新时就有全量刷新和增量刷新两种方式。

全量刷新：不管使用哪种刷新方式，物化视图的首次刷新都使用全量刷新的方式。全量刷新机制是首先物化视图对应表中的数据采用delete全部删除，然后再从原表中使用insert把数据重新插入。

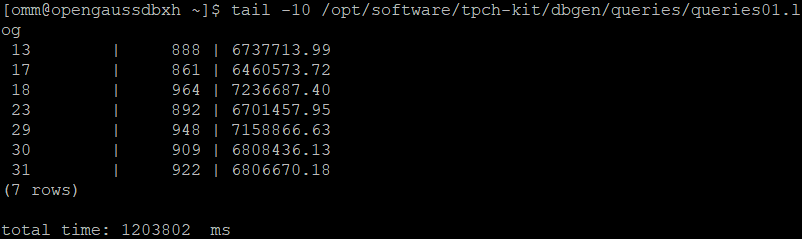
增量刷新：增量刷新是基于主表上的物化视图日志（MLOG$\_tablename）进行刷新的，主表上每插入或删除一条数据，对应物化视图日志中同样会插入一条数据（主表更新一条数据，对应物化视图中会插入两条记录数据），物化视图刷新后主表上物化视图日志记录信息会被清空，重新开始记录后面的更新。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

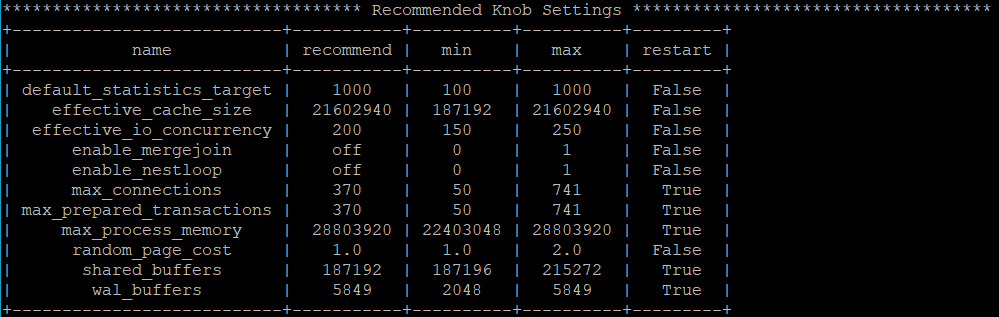
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

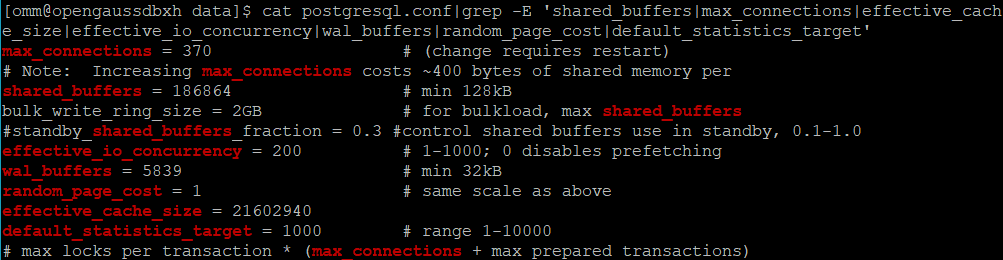
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

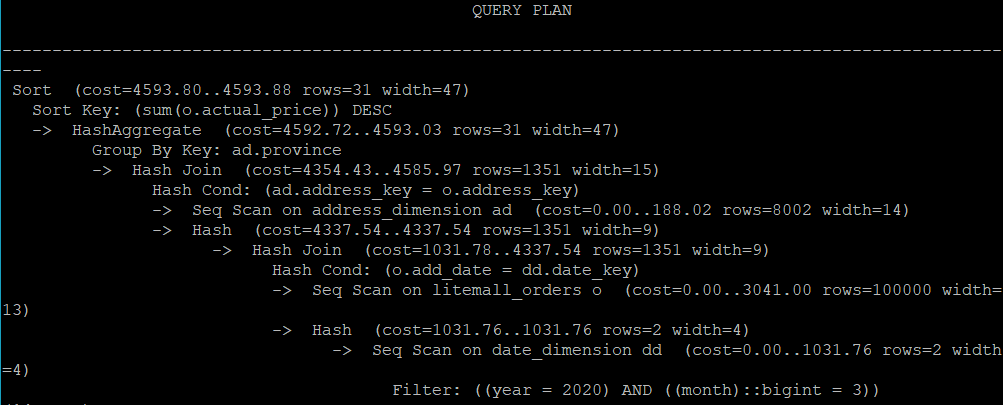
AND o.add\_date = dd.date\_key

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

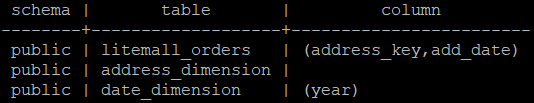
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

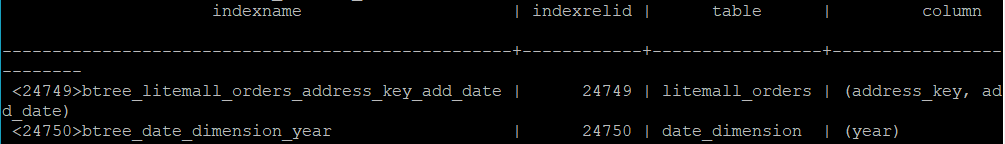
GROUP BY ad.province

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。

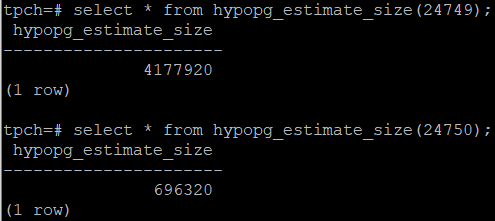
select \* from hypopg\_display\_index();



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(24749);

select \* from hypopg\_estimate\_size(24750);



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

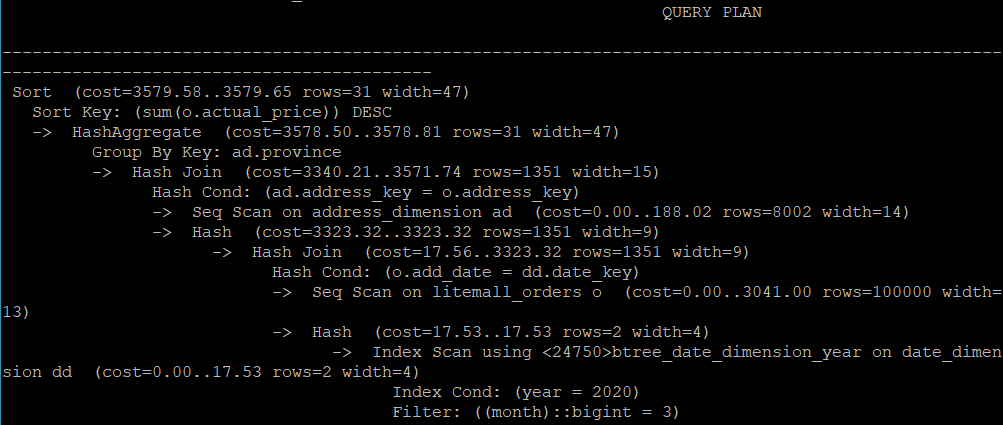
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

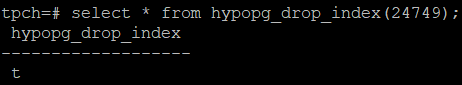
GROUP BY ad.province

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



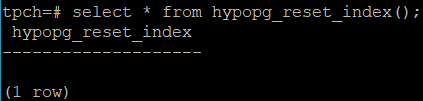
6. 删除某一个索引虚拟列，将执行结果截图。

select \* from hypopg\_drop\_index(24749);



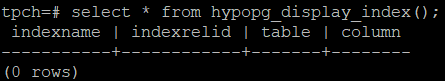
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

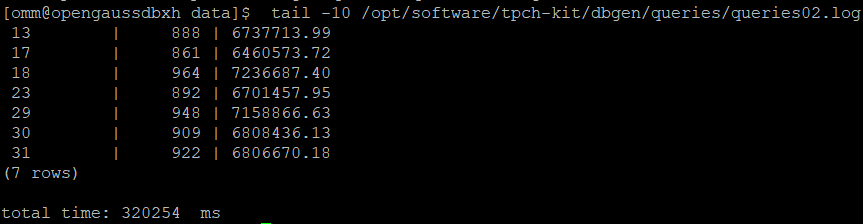
select \* from hypopg\_display\_index();



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？

根据X-Tuner给出的参数优化，在default statistics target、effective cache size、effective io concurrency、enable mergejoin、enable nestloop、max connections、max prepared transactions、max process memory、random page cost、shared buffers、wal buffers等参数上进行了优化。优化的理由主要是优化用户的体验，而且这些参数都与查询语句的运行有关，如高速缓存大小、最大进程内存大小等，选择适合的参数可以大大提高运行效率。

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？

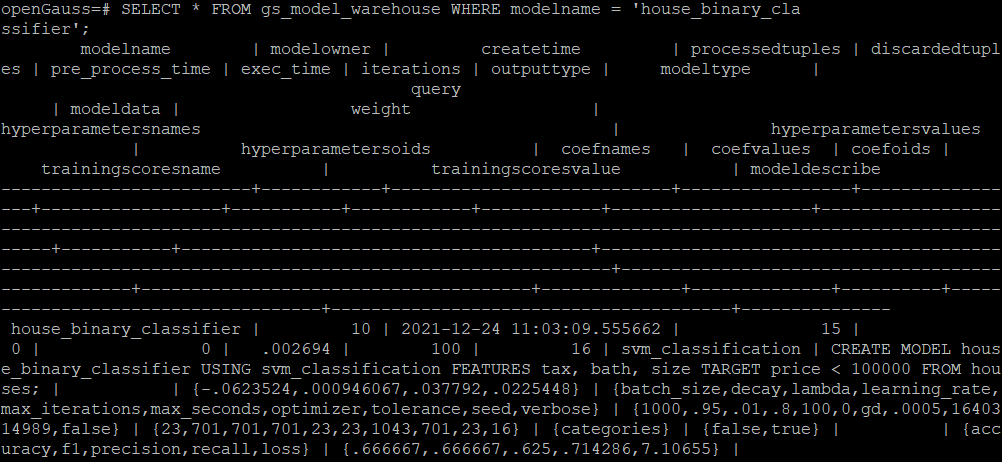
数据库在执行一条Sql语句的时候，默认的方式是根据搜索条件进行全表扫描，遇到匹配条件的就加入搜索结果集合。如果我们对某一字段增加索引，查询时就会先去索引列表中一次定位到特定值的行数，大大减少遍历匹配的行数，所以能明显增加查询的速度。

除了使用索引和参数外，还可以从计算机硬件、系统配置、数据库表结构等方面对数据库进行优化。

# 关卡四、openGauss的DB4AI特性应用

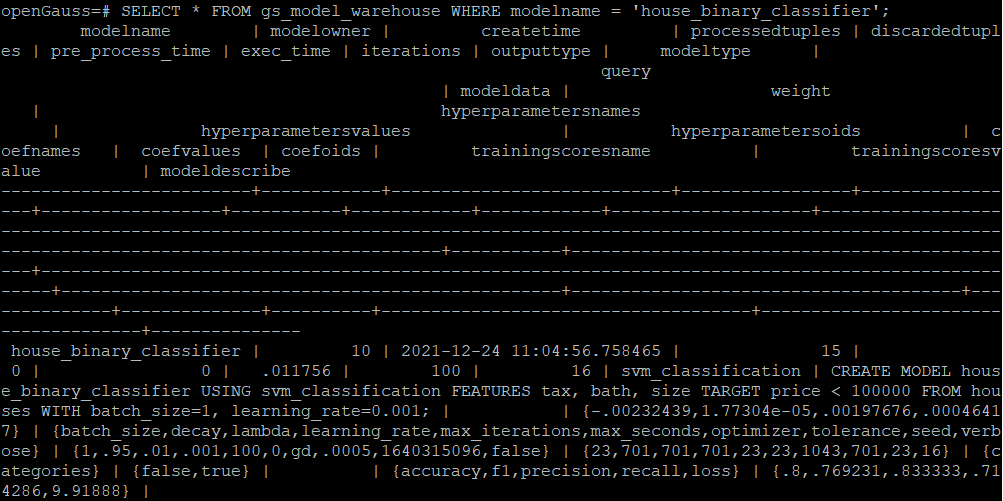
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



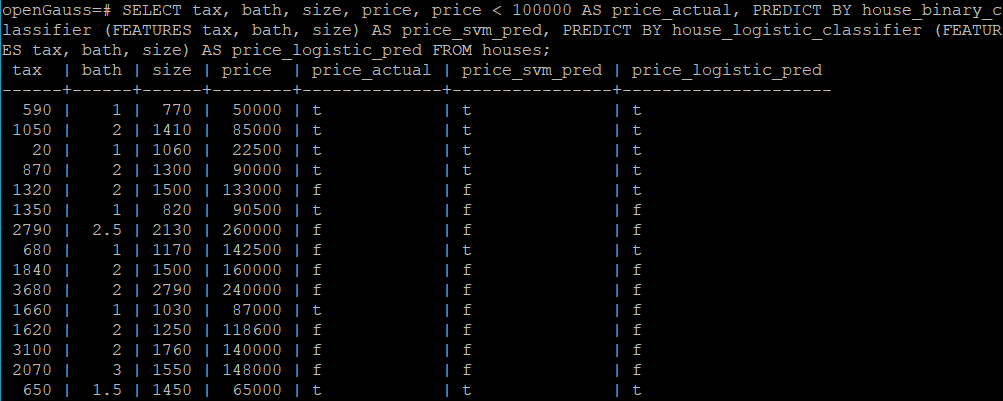
任务二：观察新模型的信息，将执行结果截图。

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？

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。

回归问题与分类问题本质上都是要建立映射关系，而两者的区别在于：

对于回归问题，其输出空间B是一个度量空间，即所谓“定量”。也就是说，回归问题的输出空间定义了一个度量 去衡量输出值与真实值之间的“误差大小”。例如：预测一瓶700毫升的可乐的价格（真实价格为5元）为6元时，误差为1；预测其为7元时，误差为2。这两个预测结果是不一样的，是有度量定义来衡量这种“不一样”的。（于是有了均方误差这类误差函数）。

对于分类问题，其输出空间B不是度量空间，即所谓“定性”。也就是说，在分类问题中，只有分类“正确”与“错误”之分，至于错误时是将Class 5分到Class 6,还是Class 7，并没有区别，都是在error counter上+1。

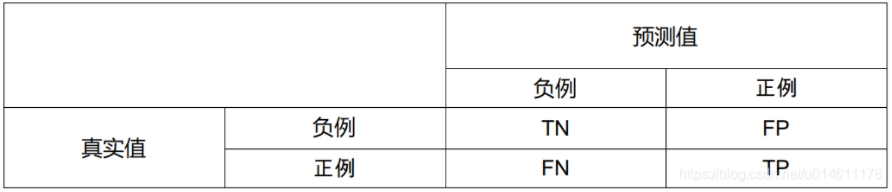
实践思考题2：什么是SVM算法？

SVM的全称是Support Vector Machine，即支持向量机。SVM是90年代中期发展起来的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？

分类问题的评价指标有：混淆矩阵、正确率、真阳率和假阳率、ROC/AUC、精准率和召回率、F1分数、Matthews相关系数等。

其中，混淆矩阵的定义如图：



正确率是猜对的样本占总样本的比例；

真阳率是正样本中猜对的比例，假阳率是负样本中猜错的比例；

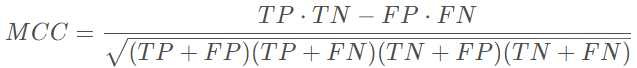
ROC（Receiver Operating Characteristic）曲线是把所有阈值对应的TPR和FPR求出来，画个直角坐标系，以FPR为横轴，TPR为纵轴，把不同阈值下的（FPR，TPR）坐标点标上并连起来，就能得到TPR和FPR的整个变化曲线，而ROC曲线下方与坐标轴围成的区域的面积，称为AUC（Area Under Curve）。

精准率是预测为正的样本中猜对的比例，召回率是：实际为正的样本中被猜对的比例；

P-R（精确率-召回率）曲线是算出不同阈值下的Precision和Recall，然后以Recall为横轴，以Precision为纵轴画出的曲线图；

F1分数的计算公式：

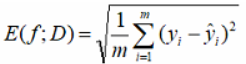
Matthews相关性系数（Matthews Correlation Coefficient, MCC）的公式：



其为特殊的Pearson相关系数，Matthews相关系数针对的是X和Y都是0-1分布的情况。

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？

回归问题的评价指标有：RMSE(平方根误差)、MAE（平均绝对误差）、MSE(平均平方误差)、Coefficient of determination (决定系数)。

其中RMSE的公式：

MAE又称L1范数损，其公式：

MSE的公式：

Coefficient of determination (决定系数) 是相关系数的平方。 相关系数是用来描述两个变量之间的线性关系的，但决定系数的适用范围更广，可以用于描述非线性或者有两个及两个以上自变量的相关关系。