openGauss AI特性创新实践课



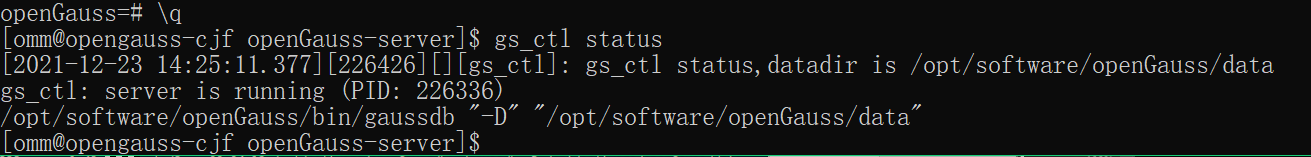
华为技术有限公司

# 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：

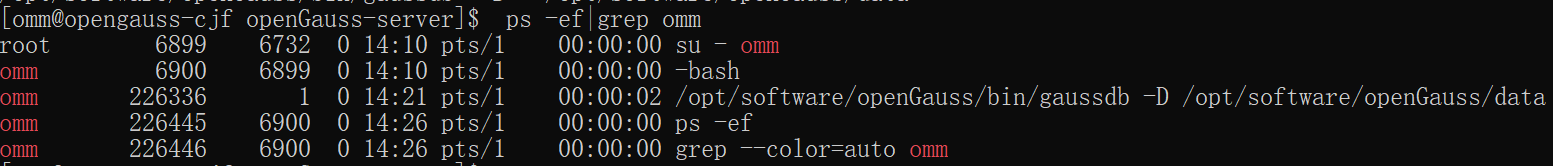
任务一：数据库状态验证

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？

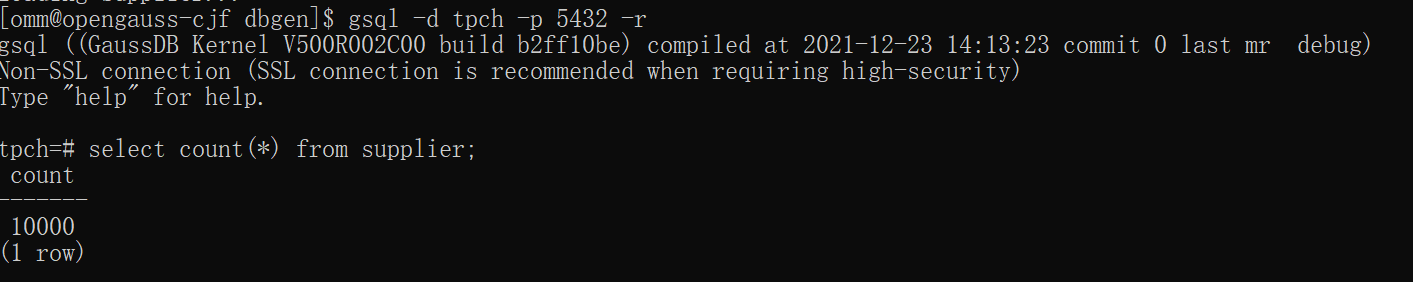
答：通过源码编译安装数据库，可以满足不同的运行平台、源码安装兼容性较好，对于不同配置的机器可以起到很好的兼容作用，并且可以对该机器进行较好的适配，方便定制，满足不同的需求，避免下载了不必要的冗余、方便运维、开发人员维护。

# 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：

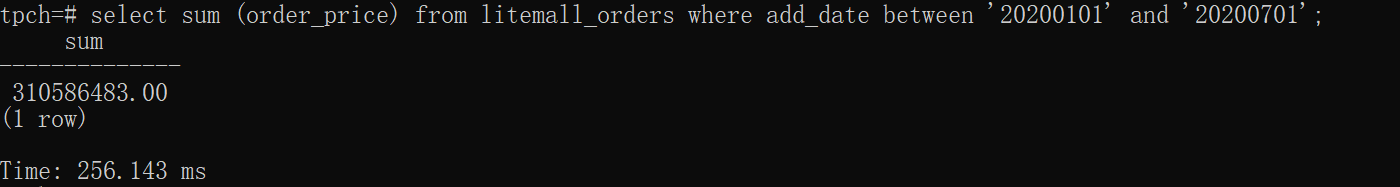
select count(\*) from supplier;;



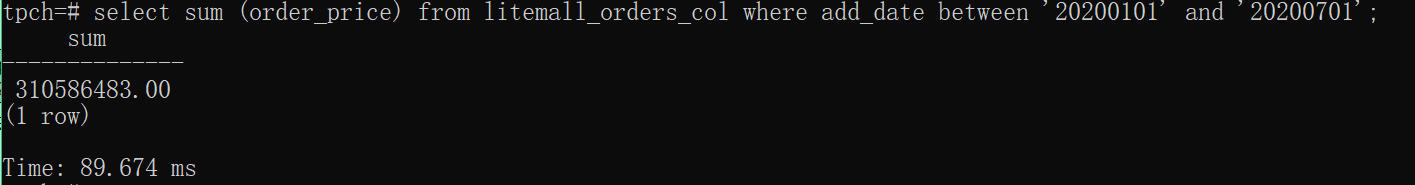
任务二：行存表与列存表执行效率对比

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';

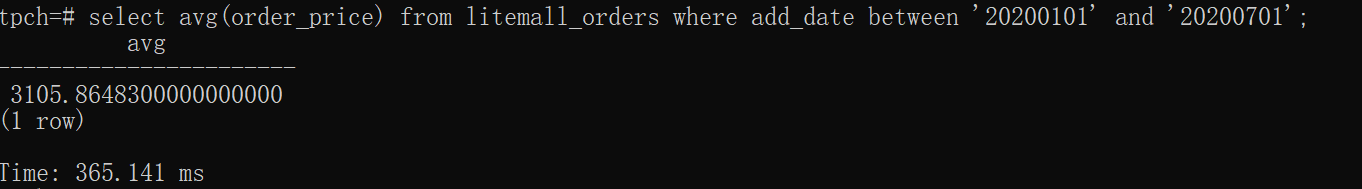


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

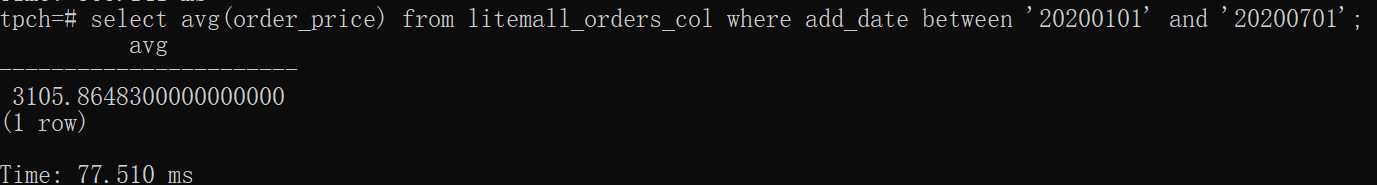


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

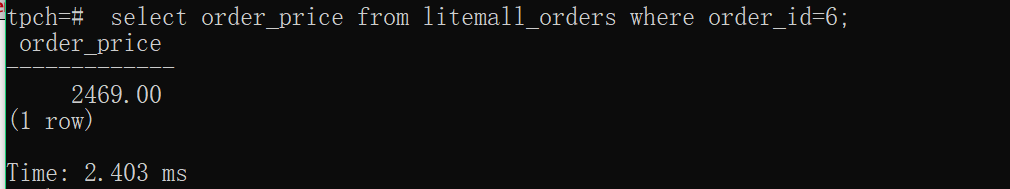


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

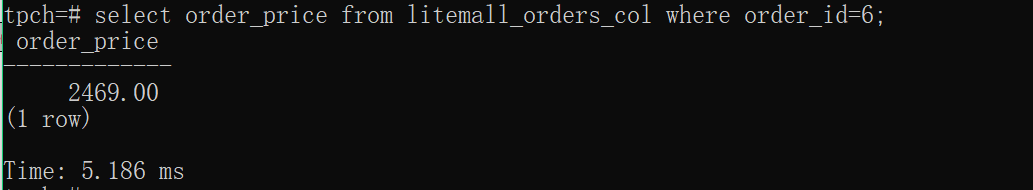


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

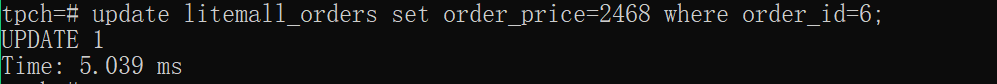


select order\_price from litemall\_orders\_col where order\_id=6;

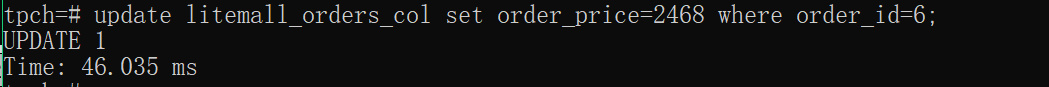


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

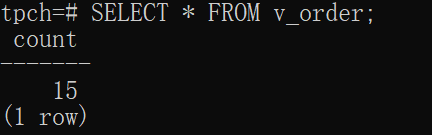
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



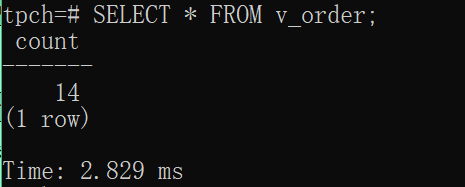
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



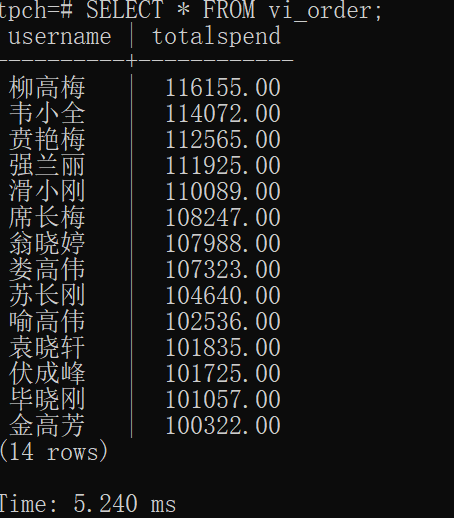
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



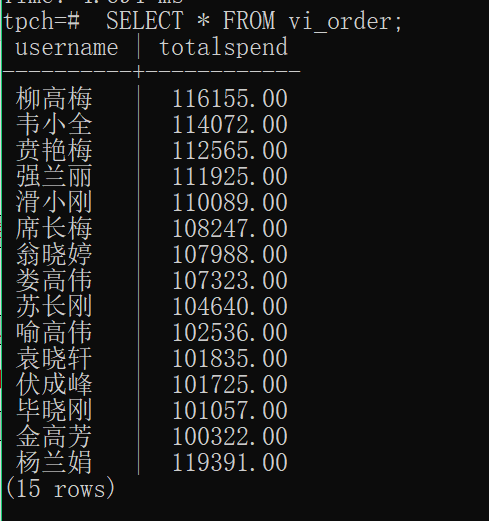
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

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？

行存表与列存表在执行相同的SQL语句时，执行的时间不同，是因为行存表和列存表存储数据的方式不同。行存表数据按行存储，列存表数据按列进行存储。

在执行一些随机的增删改查操作、频繁插入或更新的操作，需要在行中选取所有属性的查询操作，其操作与索引和行的大小更为相关的SQL语句时行存表效率更高。

在执行聚集函数相关的与列的数据操作（例如AVG、SUM等聚集函数）时，列效率较高，列存储可以直接读取这些值而不用遍历所有的行。

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，而不支持做增量更新。全量物化在刷新时是对全表数据进行刷新，一般时间比较均匀，

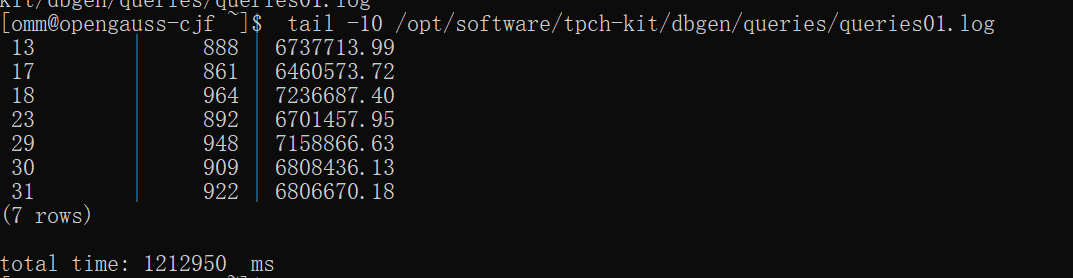
增量物化视图可以对物化视图增量更新，只刷新自上次刷新以来发生变化的部分。对大量表数据进行改变时，所花费的时间相较全量更新会较长；对某些数据项进行少量的更新或增加时，相对而言只对改变的数据进行刷新，时间一般会比全量更新更小

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

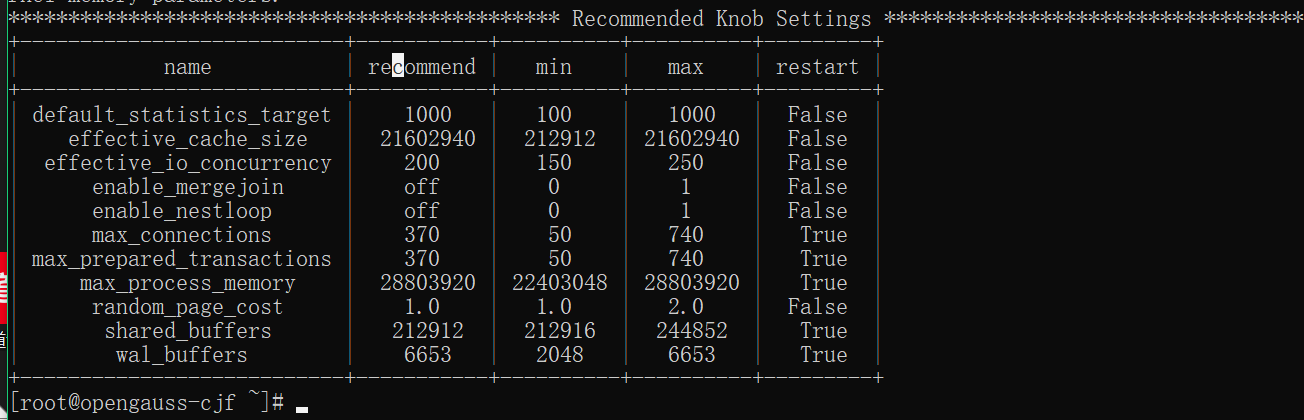
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

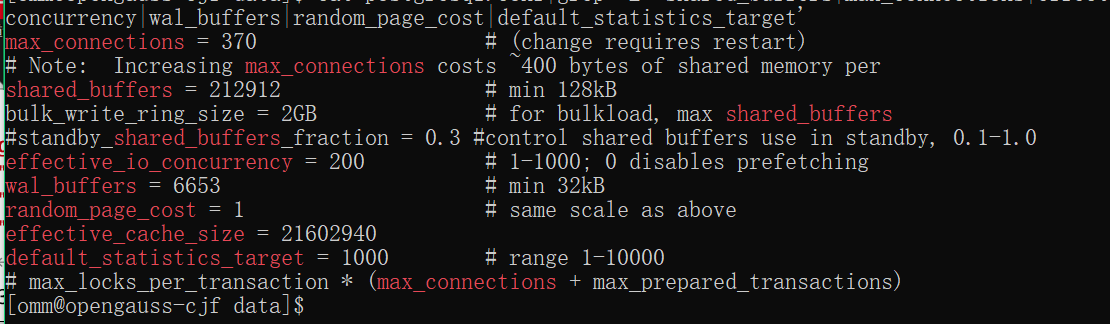
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

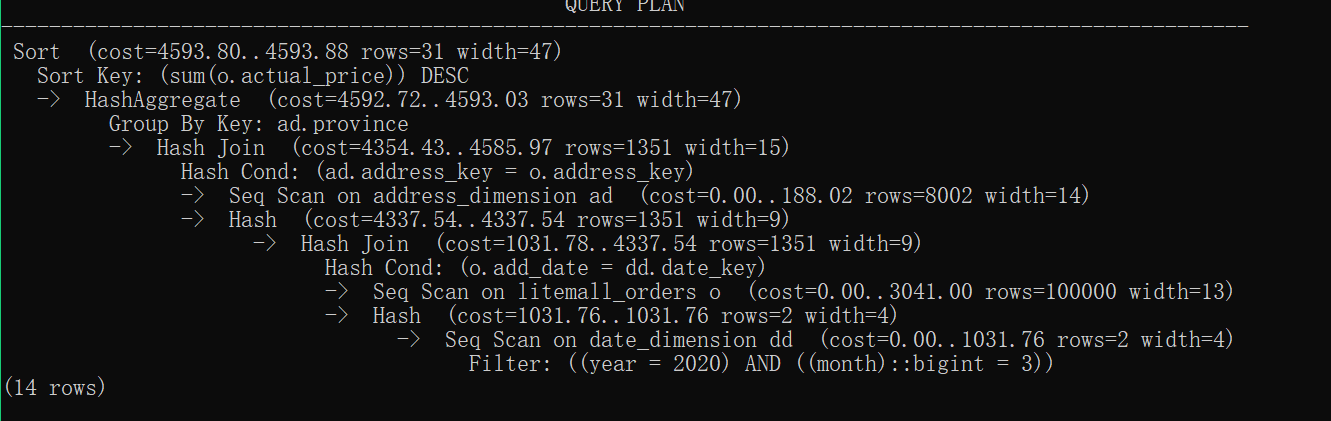
AND o.add\_date = dd.date\_key

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

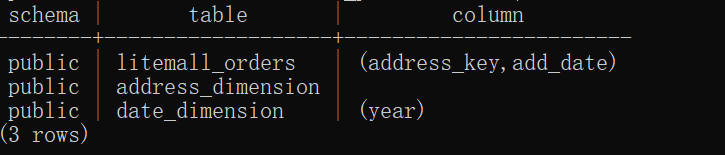
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

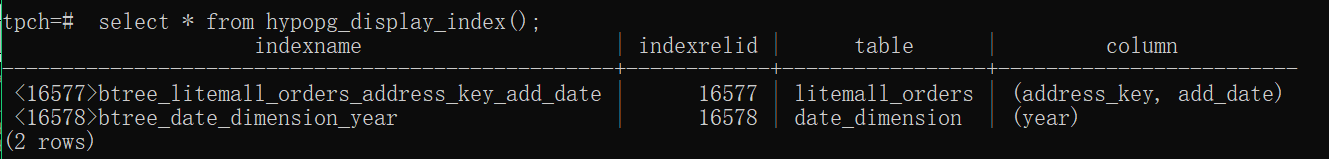
GROUP BY ad.province

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();

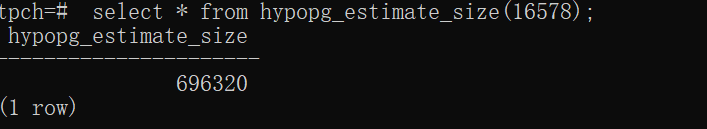


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

select \* from hypopg\_estimate\_size(16577);

select \* from hypopg\_estimate\_size(16578);





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

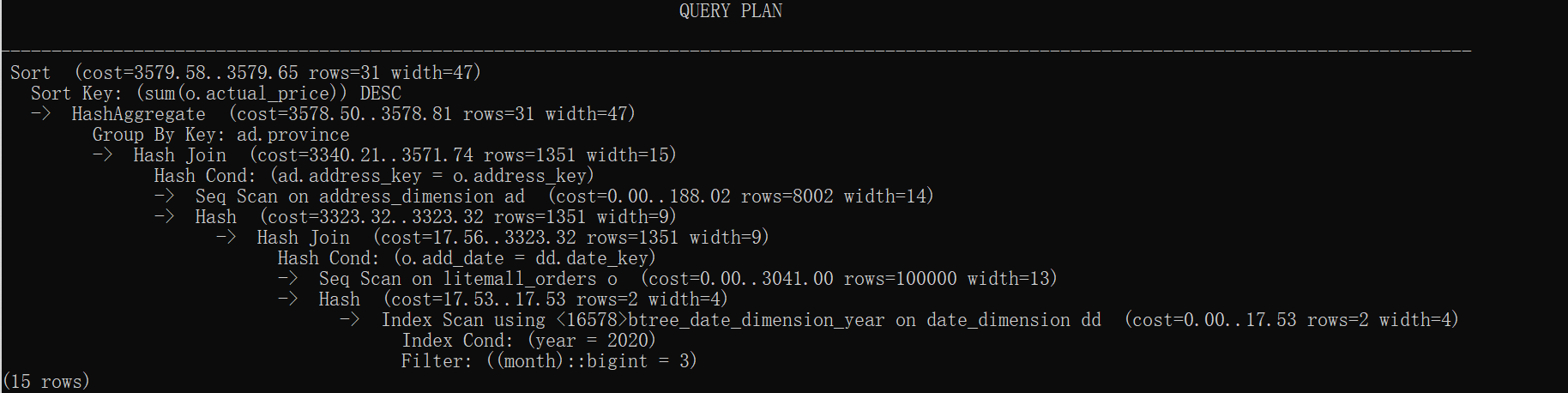
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

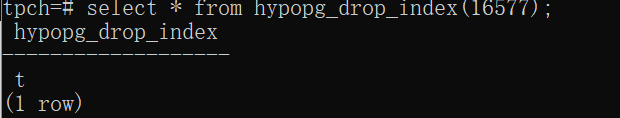
GROUP BY ad.province

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



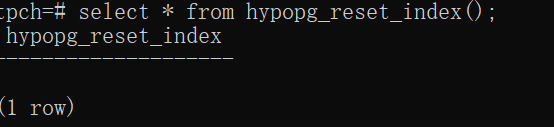
6. 删除某一个索引虚拟列，将执行结果截图。

select \* from hypopg\_drop\_index(16577);



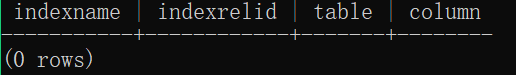
7. 删除所有索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

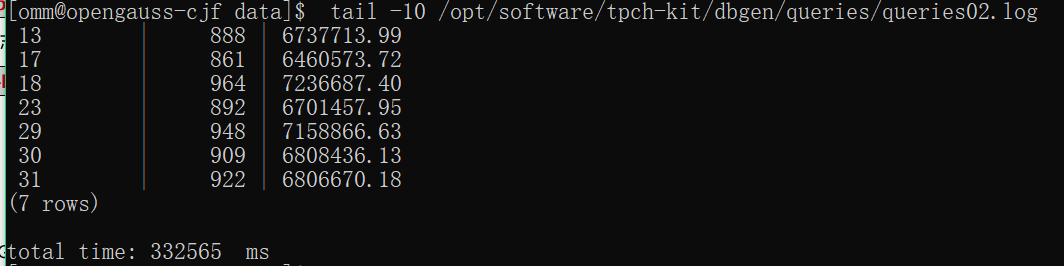
select \* from hypopg\_display\_index();



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：

根据索引推荐添加索引

CREATE INDEX idx\_l\_returnflag\_l\_linestatus ON lineitem(l\_returnflag,l\_linestatus);

CREATE INDEX idx\_p\_partkey\_p\_size\_p\_type ON part(p\_partkey,p\_size,p\_type);

CREATE INDEX idx\_s\_suppkey\_s\_nationkey ON supplier(s\_suppkey,s\_nationkey);

CREATE INDEX idx\_s\_acctbal\_s\_name ON supplier(s\_acctbal,s\_name);

CREATE INDEX idx\_ps\_partkey\_ps\_suppkey ON partsupp(ps\_partkey,ps\_suppkey);

CREATE INDEX idx\_n\_name ON nation(n\_name);

CREATE INDEX idx\_r\_regionkey ON region(r\_regionkey);

CREATE INDEX idx\_l\_orderkey ON lineitem(l\_orderkey);

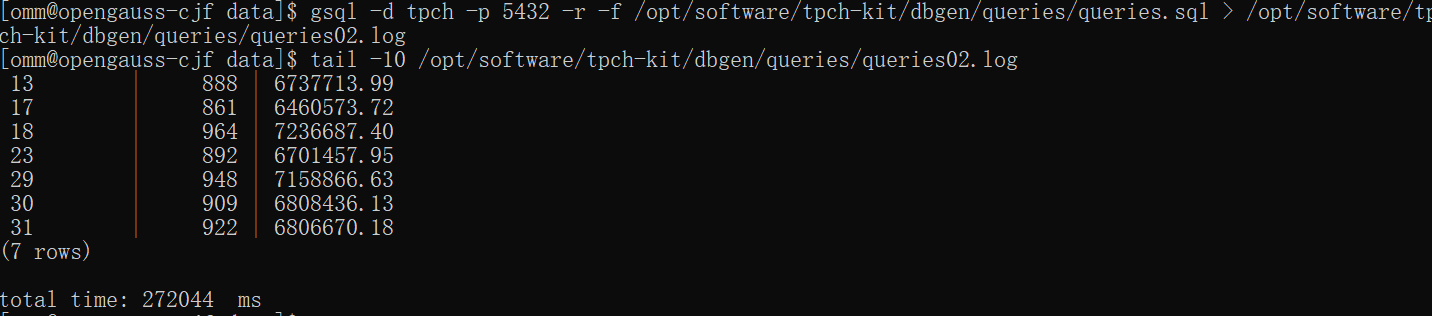
CREATE INDEX idxc\_mktsegment ON customer(c\_mktsegment);

CREATE INDEX idxc\_custkey\_c\_nationkey ON customer(c\_custkey,c\_nationkey);

重新执行queries.sql查询(由于手快了，这里还是用的quries02.log)

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

可以看出运行时间大大减少



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，主要是对于shared\_buffers、 max\_connections 、effective\_cache\_size、effective\_io\_concurrency、wal\_buffers、random\_page\_cost 、default\_statistics\_target等参数进行了优化。

TPC-H 是一个面向分析型业务(AP)的基准测试，它由一系列热点查询组成，这些热点查询都是高度复杂的，因此执行时间往往都比较长。

实验中，默认TPC-H数据库的表缺少索引，数据库的参数并没有做任何优化，因此执行效率会比较差。所以基于openGauss的X-Tuner参数自调优功能，对于数据库的运行性能，在修改这些参数至推荐最优值后能够有效提升数据库的运行性能。

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：

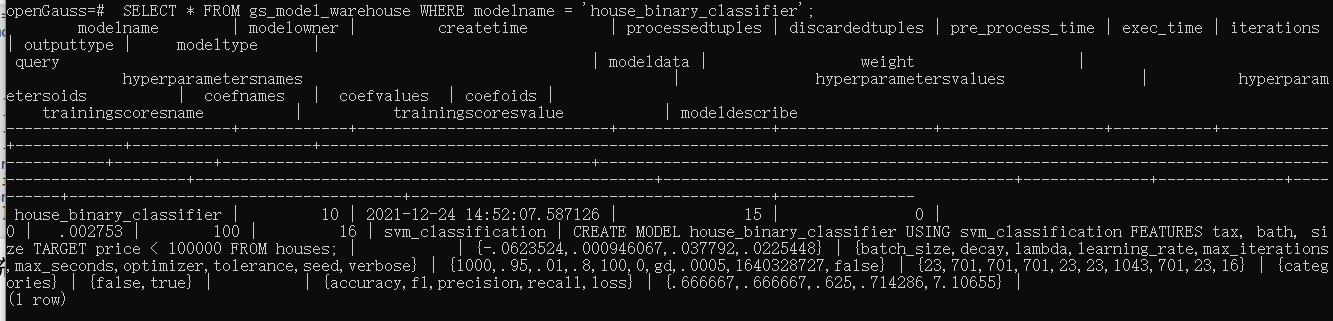
使用索引的优点：通过唯一性索引（unique）可确保数据的唯一性、加快数据的检索速度、加快表之间的连接、减少分组和排序时间、使用优化隐藏器提高系统性能

除了使用索引和参数，还可以通过优化SQL语句、优化数据表结构、优化数据库的配置、使用缓存、优化硬件等方法对数据库进行优化。

# 关卡四、openGauss的DB4AI特性应用

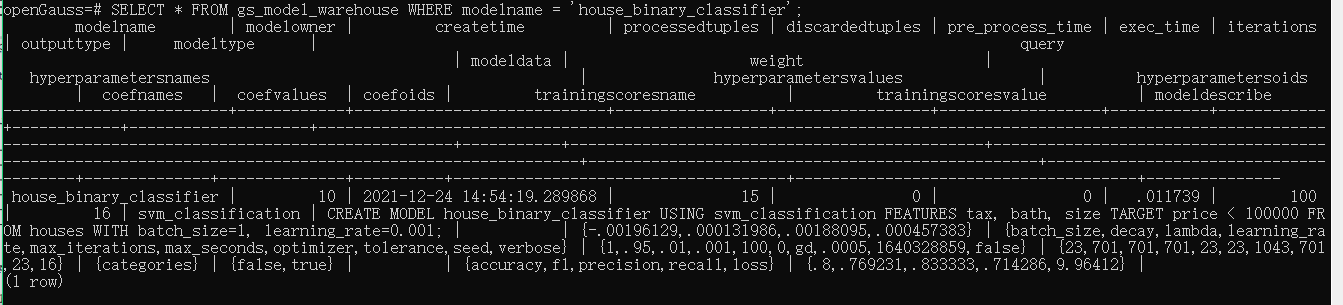
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



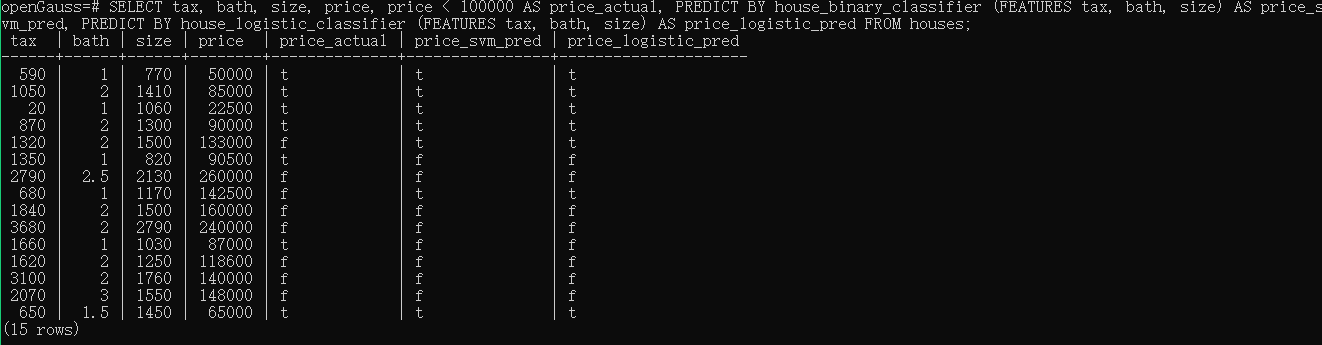
任务二：观察新模型的信息，将执行结果截图。

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？

答：

分类模型与回归模型的本质一致，其主要区别在于输出变量的类型不同；也在于其输出空间是否为一个度量空间：

分类的输出为定性的。它是用于对离散的变量进行预测，即可以将其他的模型输出离散化。其常用于对事物打上一个标签，没有逼近的概念，对于最终的分类结果只有一个正确与否的判定。

回归的输出为定量的，它是用于对连续的变量进行预测，即可以将其他的模型输出连续化。其是对真实值的一种逼近预测，常用来预测一个值，即会在一个度量空间中去衡量逼近真实值。回归模型会减少所谓的输出值与真实值之间的“误差大小”。

实践思考题2：什么是SVM算法？

SVM即支持向量机，是在分类与回归分析中分析数据的监督式学习模型与相关的学习算法。每个训练实例都会被标记为属于两个类别中的一个或另一个，使其成为非概率的二元线性分类器。它的目标就是为了使训练样本分类正确，让分类函数划分的超平面尽可能地将训练样本分割开。而对于距离超平面最近的几个训练样本点可以称其为“支持向量”，两个异类支持向量到超平面的距离之和被称为“间隔”，通过优化使这个“间隔”最大化的算法便是SVM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？

1. 准确率（Accuracy）：

即对于给定的数据，分类正确的样本数占总样本数的比例。

1. 精确率（Precision）：

假设我们将关注的类定义为正类，其他类定义为负类。通过分类器在数据集上的分类正确错误，我们定义以下4种情况：

* TP ：True Positive， 把正类预测为正类；
* FP ：False Positive， 把负类预测为正类；
* TN：True Negative， 把负类预测为负类；
* FN：False Negative，把正类预测为负类。

则精确率就是指在预测为正类的样本中真实类别为正类所占的比例，即

1. 查全率（召回率 Recall）

召回率是指在所有的正类中被预测为正类的比例，即：

1. F1-Score

即精确率和召回率的调和平均值，即：

1. ROC曲线和AUC

真阳率TPR和假阳率FPR分别为：

使用FPR为横坐标，TPR为纵坐标得到的ROC曲线中，其下的面积即为AUC。因为ROC曲线越靠近左上角，分类器效果越好，所以AUC越大越好。

1. Kappa系数

对总体分类精度和，即第i类真实样本数\*第i类预测出的样本数之和除以样本总数平方。K表示为：

K越大分类效果越好。

1. 海明距离

其衡量了预测标签与真实标签之间的距离，取值在0~1之间。距离为0说明预测的结果与真实结果完全相同，距离为1就说明模型的预测每次恰好为相反的类别。

1. Jaccard相似系数

其与海明距离的不同之处在于分母。当系数为1时，预测结果与实际情况完全相符时；当系数为0时，预测结果与实际情况完全不符。

1. Hinge loss

主要用以使“边缘最大化”，该损失取值在0-1之间。当取值为0时，表示多分类模型分类完全准确；当取值为1时，表示模型的分类结果没有任何意义。

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？

1. 平均绝对误差（MAE）

即对绝对误差损失的平均值，反映预测值误差的实际情况：

1. 平均绝对百分比误差（MPAE）

即对相对误差损失的预期值。其中相对误差就是绝对误差和真实结果值的百分比：

1. 均方误差（MSE）

即观测值与真实值的偏差的平方和与观测次数的比值，回归过程中尽可能使之最小：

1. 均方根误差（RMSE)

即对平均误差的算术平方根的期望值，衡量观测值与真实值之间的偏差：

1. 均方误差对数（MSLE）

即对平方对数差的期望值：

1. 中位绝对误差（MedAE）

即取目标和预测之间的所有绝对差值的中值来计算损失：

1. R2 Score

也被成为R-Squared，为可决系数，也叫拟合优度，反映自变量x对因变量y的变动的解释程度。分子为训练得到模型的预测误差和，分母为随机取值下得到的误差和，一般为观测值的平均值。越接近于1,拟合越好。一般认为超过0.8的模型就拟合优度较高：