openGauss AI特性创新实践课



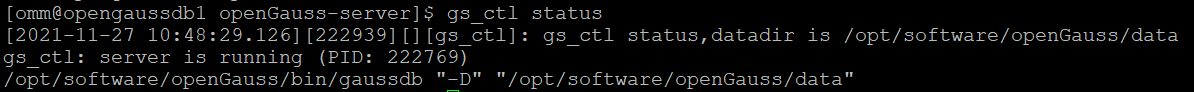
华为技术有限公司

# 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：

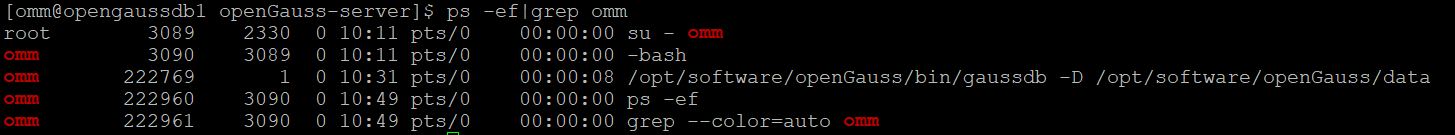
任务一：数据库状态验证

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）



任务三：安装数据库所需要的步骤

① 编译前准备

包括：登录ECS，创建用户omm，下载第三方编译库，下载安装依赖包

② 数据库安装编译

包括：设置并添加环境变量、编译和安装数据库

③ 数据库初始操作

包括：初始化数据库、登录数据库、查看了解数据库

实验思考题：为什么需要通过源码编译，安装数据库？

源码编译安装的优点：

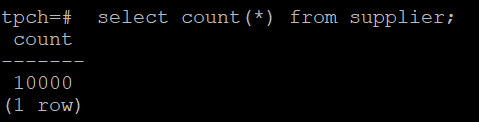
1. 安装的过程、软件更适合自己的机器配置
2. 一般安装路径为/usr/local 目录，若以后重装系统，可以把软件程序原封不动拷贝过去
3. 编译安装的版本比编译好的版本更高级
4. 不用的用户对软件的需求不同，源码编译可以满足实现针对用户的定制版软件
5. 可以满足不同版本的linux的运行条件，而安装编译好的软件（一个二进制包）
6. 方便管理维护

# 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：

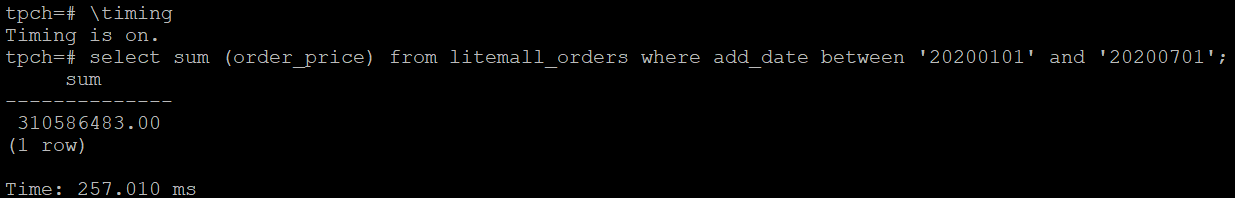
select count(\*) from supplier;;



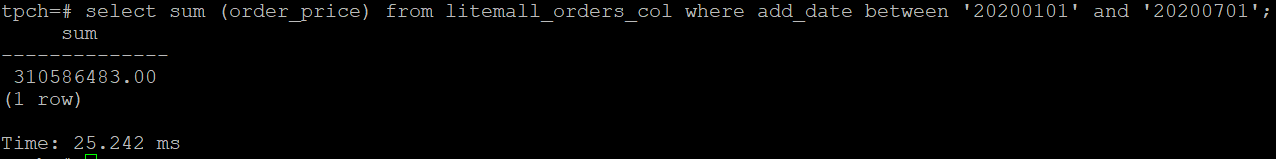
任务二：行存表与列存表执行效率对比

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';

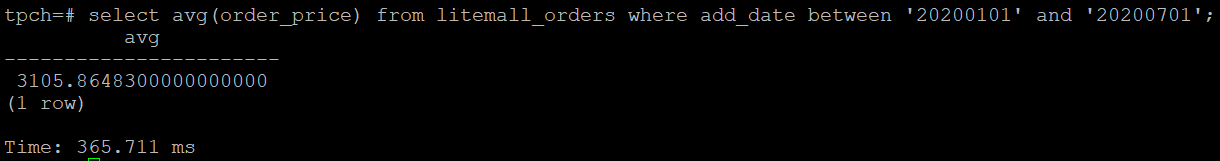


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

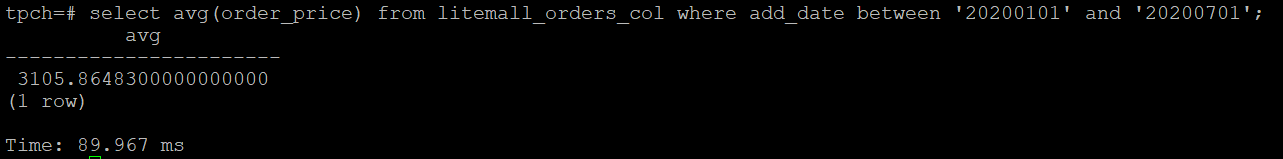


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

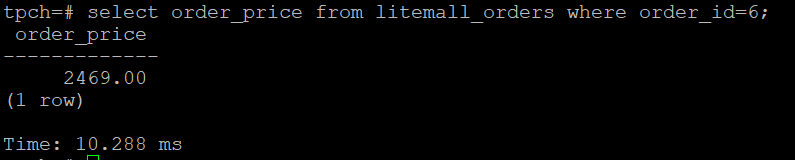


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

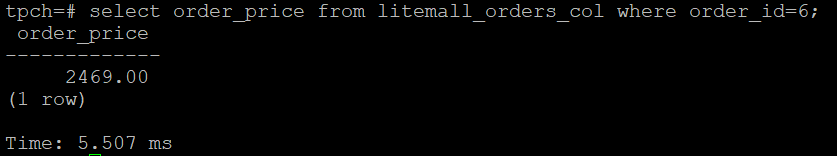


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

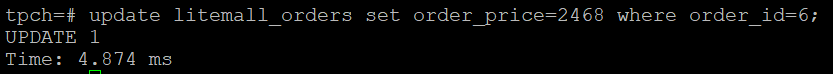


select order\_price from litemall\_orders\_col where order\_id=6;

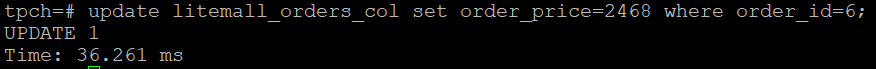


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

update litemall\_orders set order\_price=2468 where order\_id=6;



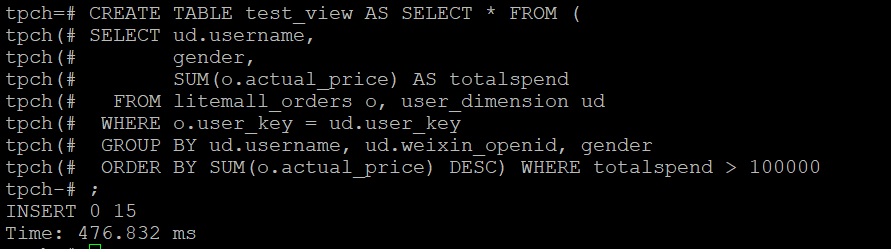
update litemall\_orders\_col set order\_price=2468 where order\_id=6;

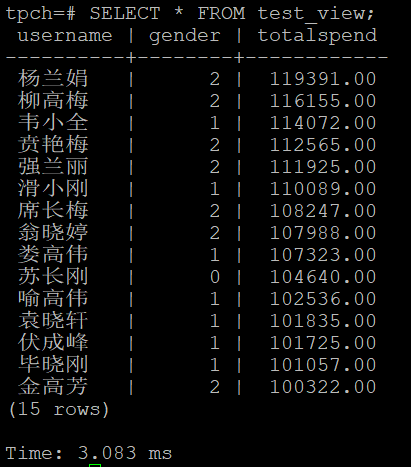


任务三：物化视图的使用

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：

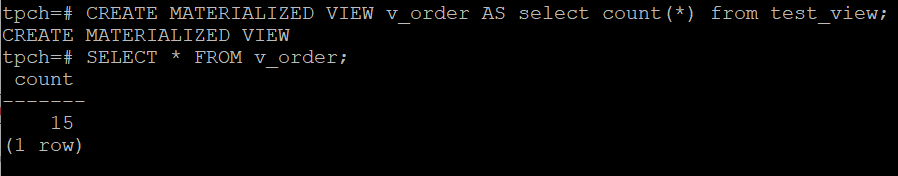
SELECT \* FROM test\_view;





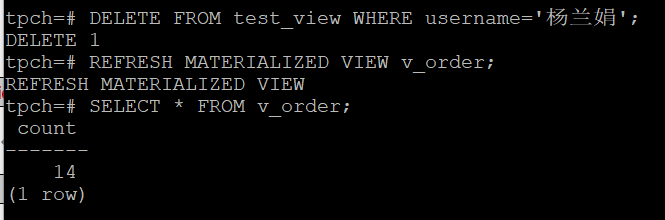
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



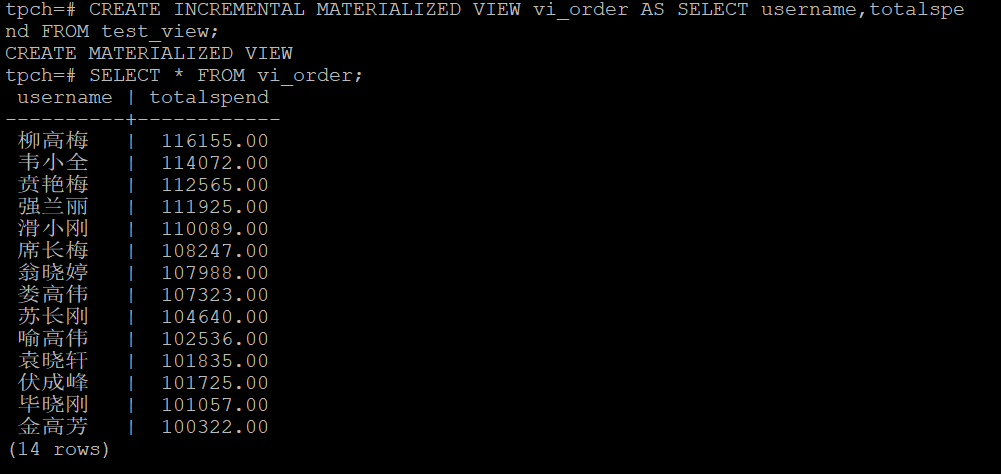
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



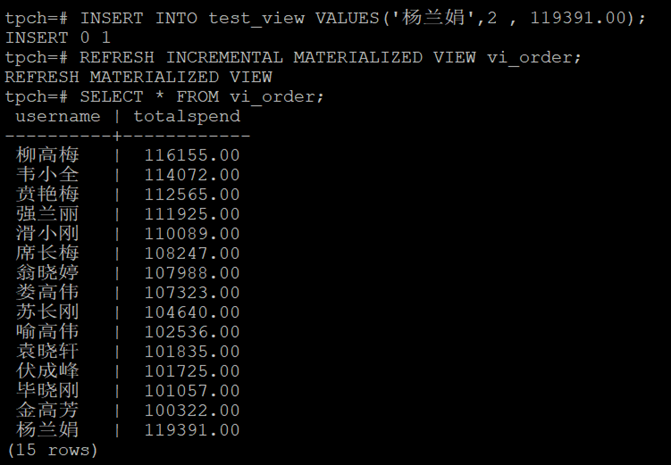
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

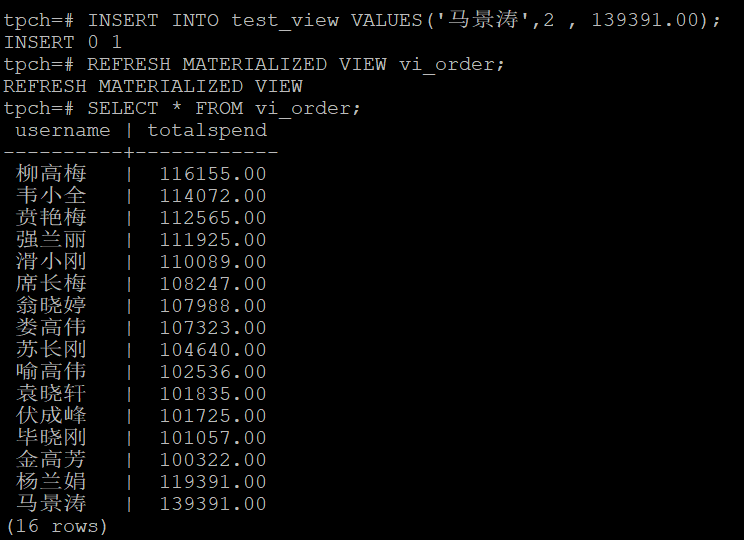
SELECT \* FROM vi\_order;



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？

数据总结：单位：ms

|  |  |  |
| --- | --- | --- |
|  | 行存表 | 列存表 |
| sum | 257.010 | 25.242 |
| avg | 365.711 | 89.967 |
| Order\_id=6 | 10.288 | 5.507 |
| update | 4.874 | 36.261 |

从数据来看，列存表在求和、求均值、求特殊选择的时候效率比行存表高，而在更新数据时行存表效率比列存表效率高。

行存表是把一行数据（一个元组）储存为一个单元，而列存表是把一列数据（同一属性）储存为一个单元。从读数据和处理数据来看，行存表一个单元是不同数据类型，而列存表一个单元是相同的数据类型，没用冗余数据，不管是数据压缩还是数据分析，都比行存表高效。从写入和修改来看，行存表可以一次性输入完整数据，而列存表需要先将数据拆分成单列保存才能输入完整数据，是需要多次输入，所以从这个角度来看，行存表更高效。这与实验数据是一致的。

下表总结了行存表和列存表的优缺点和适用场景：

|  |  |  |
| --- | --- | --- |
|  | 行存表 | 列存表 |
| 优点 | 一次性写入、修改，效率高 | 读取数据没用冗余，读取速度快；数据类型相同，容易压缩，适用于大数据的存储 |
| 缺点 | 读取数据存在冗余，降低了读取数据的效率 | 写入修改需要先拆分再多次插入，耗时长 |
| 适用场景 | 需要进行频繁的更新操作，关注整张表的内容 | 需要进行频繁的读取数据、聚集运算操作，关注某几列的内容 |

实践思考题2：全量物化视图与增量物化视图有哪些差别？

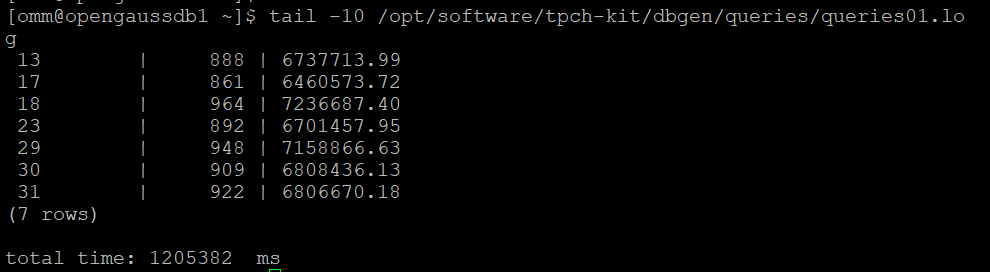
1. 全量物化视图只支持全量刷新，增量物化视图支持全量刷新和增量刷新，全量刷新会刷新整个视图，而增量刷新只会刷新上次刷新后修改的部分。
2. 增量物化视图创建语句仅支持基表扫描语句或UNION ALL语句。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

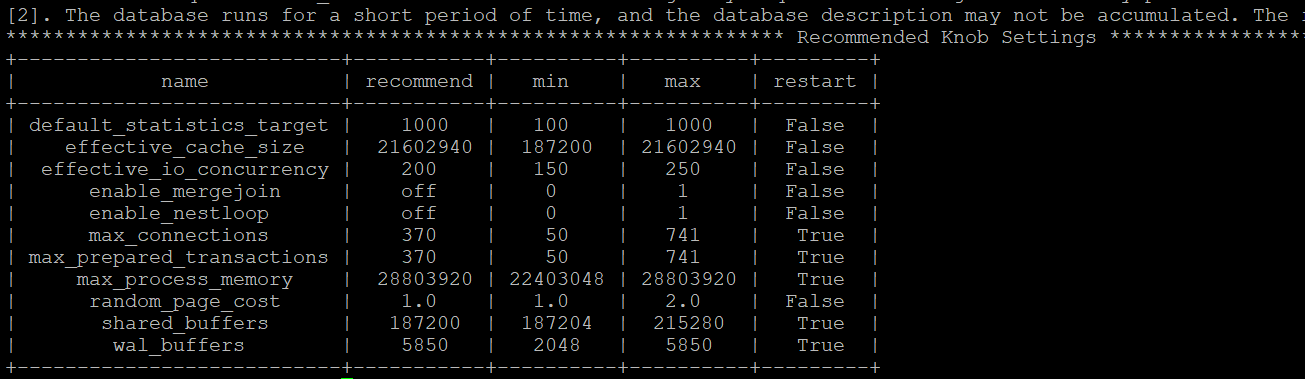
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

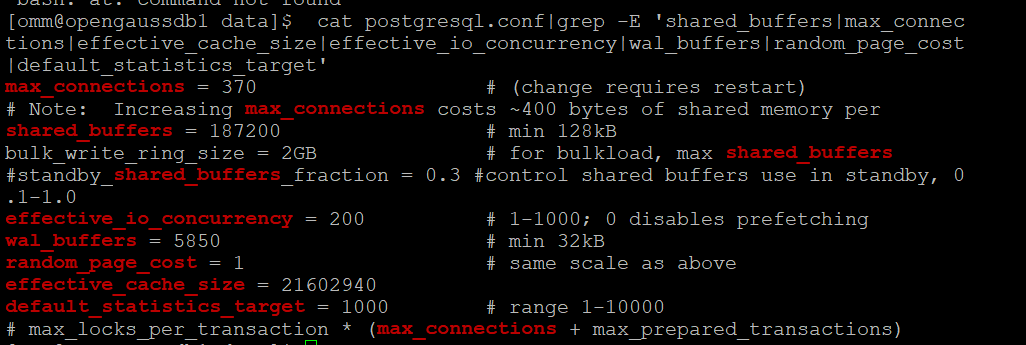
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

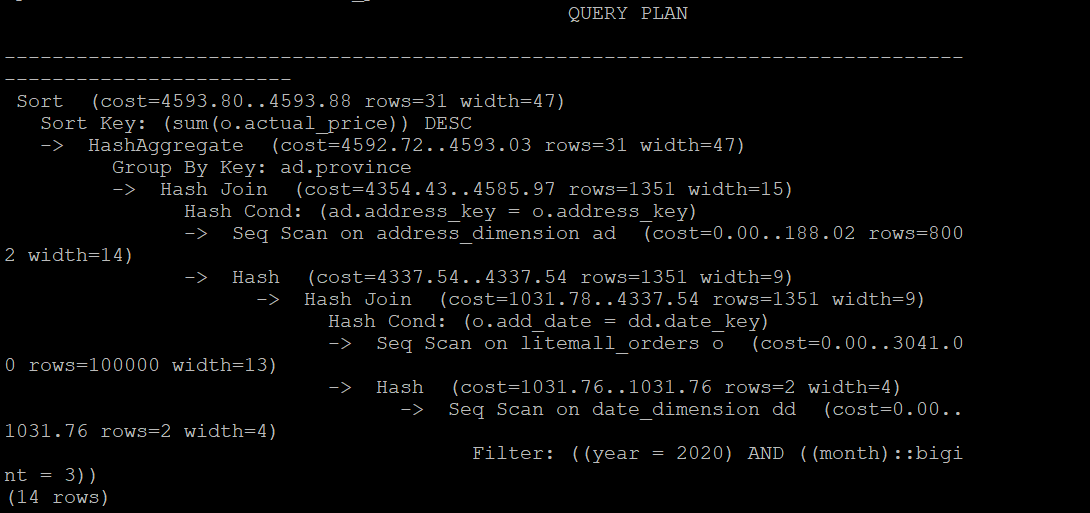
AND o.add\_date = dd.date\_key

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

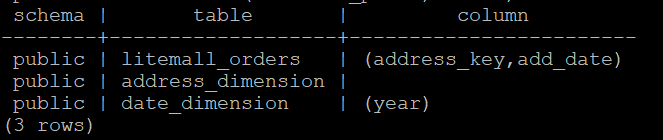
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

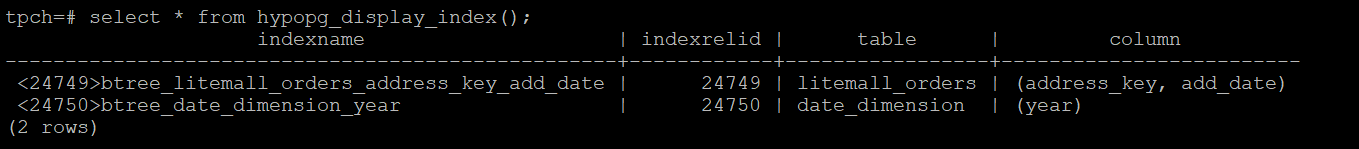
GROUP BY ad.province

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。

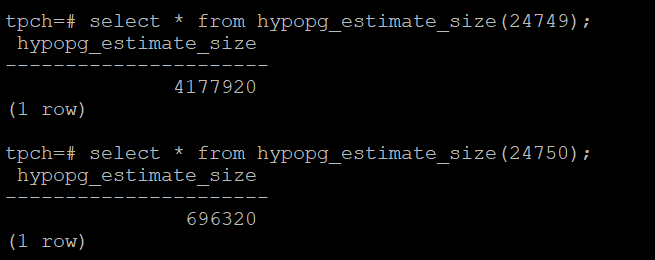
select \* from hypopg\_display\_index();



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(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

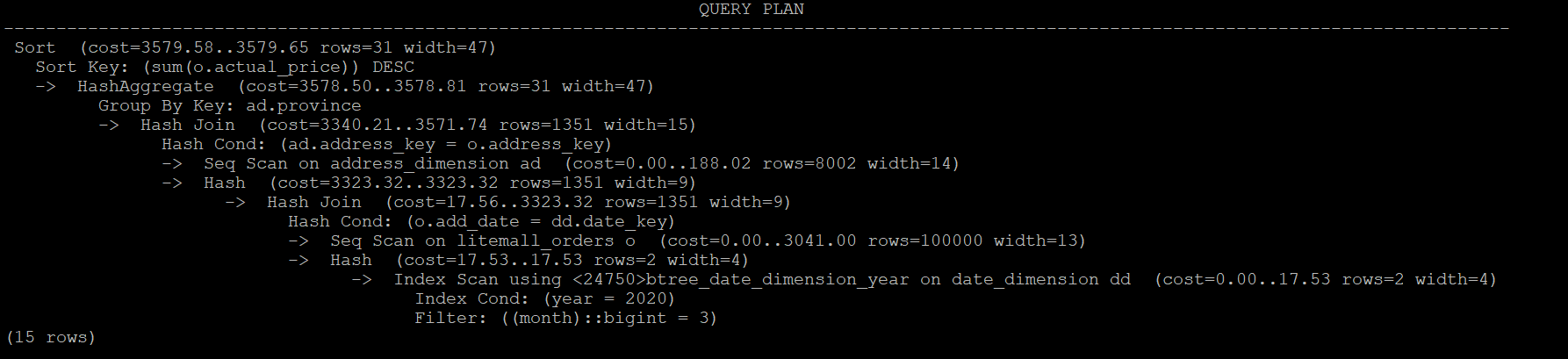
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

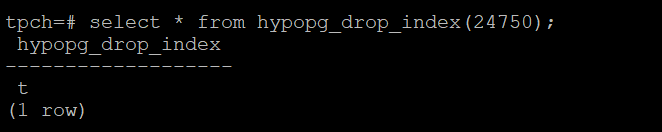
GROUP BY ad.province

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



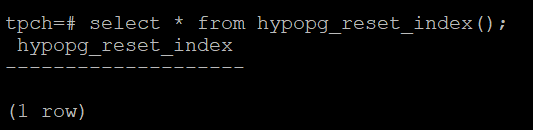
6. 删除某一个索引虚拟列，将执行结果截图。

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



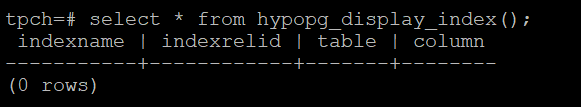
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

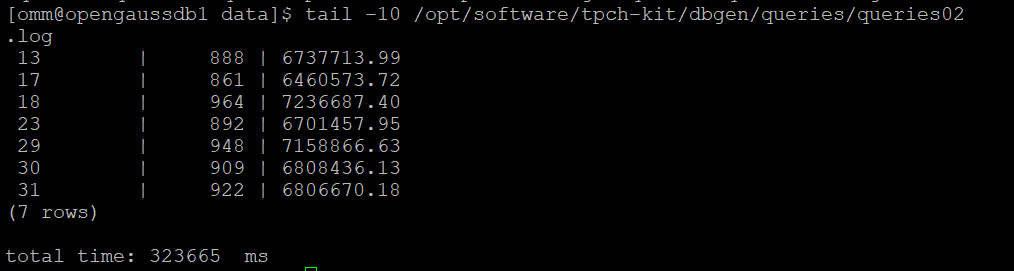
select \* from hypopg\_display\_index();



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？

|  |  |  |
| --- | --- | --- |
| 参数 | 修改前（默认值） | 修改后 |
| shared\_buffers | 1024 | 186864 |
| max\_connections | 100 | 370 |
| effective\_cache\_size | 4GB | 21602940 |
| effective\_io\_concurrency | 1 | 200 |
| wal\_buffers | -1（-1） | 5839 |
| random\_page\_cost | 4 | 1 |
| default\_statistics\_target | 100 | 1000 |

default\_statistics\_target：这个参数定义了在最常用值列表中存储多少个值，并且还指示了通过ANALYZE进程检查的行数。。更大的值可能会改善规划器的估计质量。一开始就将值增加到1000，并根据EXPLAIN ANALYZE查询结果增加/减少值进行进一步优化。

random\_page\_cost：设置规划器对一次非顺序获取磁盘页面的代价估计。减少这个值将导致系统更倾向于索引扫描，由于该数据库内容较少，小于服务器总内存，故可以把数据完全放在高速缓存中，在此情况下，随机读取并不比顺序读取开销大，这就降低了随机读取的代价。

wal\_buffers：将其WAL（预写日志）记录写入缓冲区，然后将这些缓冲区刷新到磁盘。如果有大量并发连接的话，则设置为一个较高的值可以提供更好的性能

effective\_cache\_size：可用于磁盘高速缓存的内存量的估计值，更高的数值会使得索引扫描更可能被使用，更低的数值会使得顺序扫描更可能被使用

effective\_io\_concurrency：可以同时被执行的并发磁盘 I/O 操作的数量，将它设得更大，甚至完全代替磁盘预读，可以提高I/O速率。

max\_connections：决定数据库的最大并发连接数，如果服务器的并发连接请求量比较大，建议调高此值，以增加并行连接数量

shared\_buffers：使用的共享内存缓冲区量，一般会根据需求适当调大内容缓冲空间。

**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、索引的好处：

① 提高了检索速度和性能

② 使用优化隐藏器提高系统性能

③ 加强了表之间的联系

④ 减少分组和排序时间

但是索引也不是越多越好，建立索引本身就是一个大工程：建立索引耗时、I/O操作耗时，索引建立以后占据内存空间，调入进程以后会占据CPU，索引的维护也是一个成本很高的工作。

2、其他优化方法

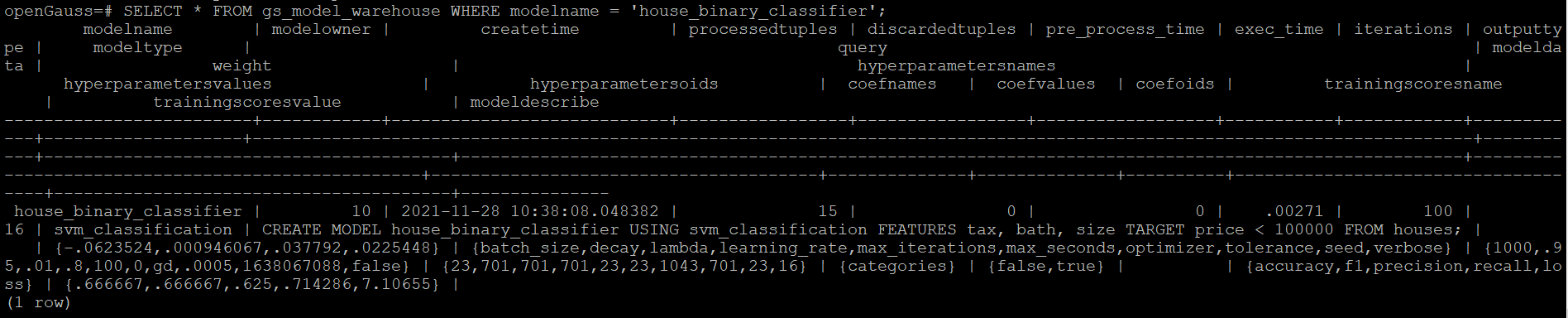
① 设置外键

② 优化SQL语句

# 关卡四、openGauss的DB4AI特性应用

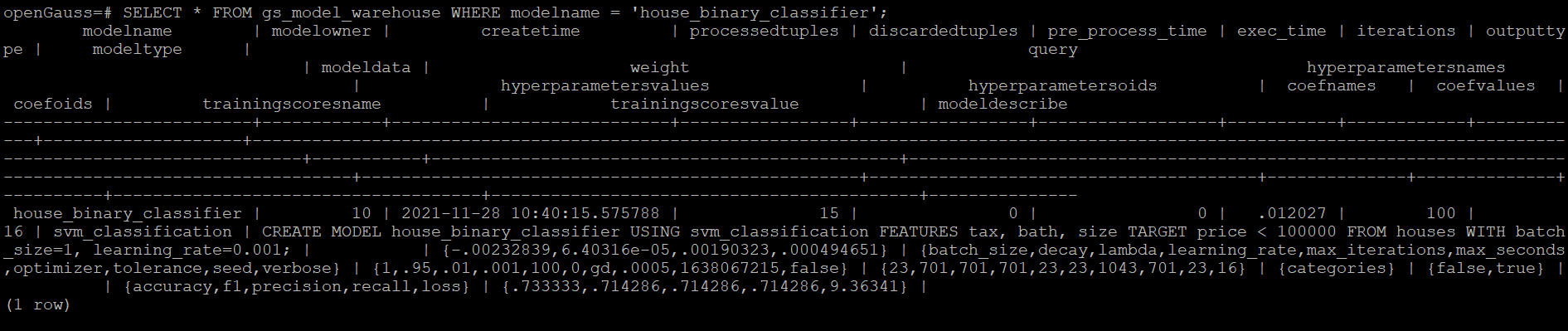
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



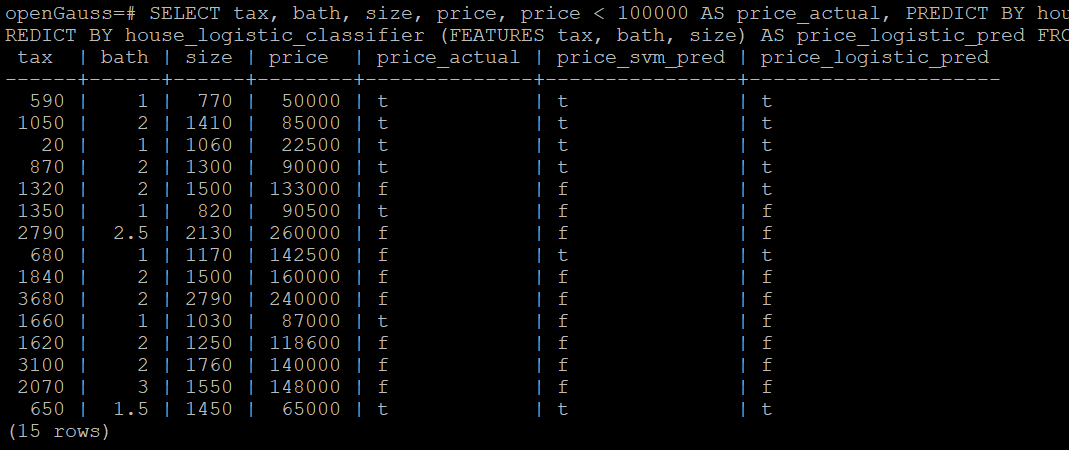
任务二：观察新模型的信息，将执行结果截图。

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？

分类是将预测的结果划分为几个离散的值，而回归是预测出未来的结果，是一个连续的值。区分分类和回归看的是输出结果是否是连续数据。

|  |  |  |
| --- | --- | --- |
|  | 分类 | 回归 |
| 数据 | 离散数据 | 连续数据 |
| 预测目的 | 寻找一个分类标准（决策边界） | 找到最优拟合，使预测值与真实值的误差最小化 |

实践思考题2：什么是SVM算法？

SVM算法是一个二分类算法，它的基本模型是定义在特征空间上的间隔最大的线性分类器。

模型假设有一条表达式为W•X+b =0的最优分割线将数据分为两类，使得离分割线比较近的点能有更大的间距。SVM包括核函数，这使它成为实质上的非线性分类器。SVM的的学习策略就是间隔最大化，可形式化为一个求解凸二次规划的问题。

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？

混淆矩阵

|  |  |  |  |
| --- | --- | --- | --- |
| 实际值\预测值 | + | - |  |
| + | TP | FN | TP+FN |
| - | FP | TN | FP+TN |
|  | TP+FP | FN+TN | TOTAL |

|  |  |  |
| --- | --- | --- |
| 评价指标 | 含义 | 公式 |
| 精确率（Accuracy） | 分类正确的样本数占总样本数的比例 | (TP+TN)/TOTAL |
| 准确率（Precision） | 预测为正的样本中预测正确的概率 | TP / (TP + FP) |
| 召回率（Recall） | 预测为正的样本占据所有实际为正的yan根本的比例 | TP / (TP + FN) |

除此之外还有其他指标：

ROC 曲线：

|  |  |  |
| --- | --- | --- |
|  | 含义 | 公式 |
| FPR | 将负例错分为正例的概率 | FPR=FP/(FP+FN) |
| TPR | 能将正例分对的概率 | TPR=TP/(TP+FN) |

分类器对应的ROC曲线应该尽可能靠近坐标轴的左上角， 而对角线的位置意味着分类器的效果和随机猜测一样的差。

AUC:

定义：ROC 曲线下的面积

评判标准：

* AUC = 1： 完美分类器， 采用该模型，不管设定什么阈值都能得出完美预测
* 0.5 < AUC < 1： 优于随机猜测，分类器好好设定阈值的话，有预测价值
* AUC = 0.5： 跟随机猜测一样，模型没有预测价值
* AUC < 0.5 ：比随机猜测还差，但是如果反着预测，就优于随机猜测。

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？

|  |  |  |
| --- | --- | --- |
| 指标 | 含义 | 公式 |
| 均方误差（MSE） | 真实值和预测值之间误差的平方值 |  |
| 均方根误差（RMSE） | 均方误差的平方根 |  |
| 平均绝对误差（MAE） | 真实值和预测值之间误差的绝对值 |  |
| 确定系数（R Squared） | SSR（预测数据与原始数据均值之差的平方和）和SST（原始数据和均值之差的平方和）的比值 |  |