openGauss AI特性创新实践课



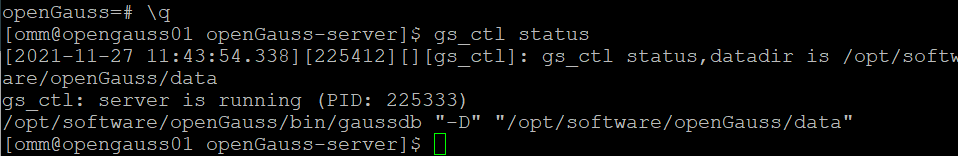
华为技术有限公司

# 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：

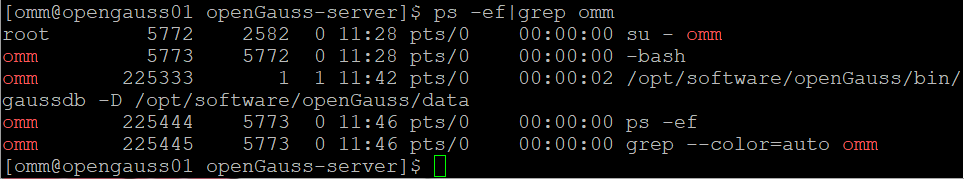
任务一：数据库状态验证

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？

一、源码安装数据库的原因：

1. 更换硬盘或重装系统时可直接/usr/local下的程序进行拷贝，便于迁移；
2. 可通过直接删除 /usr/local/bin或 /usr/local/lib 及/usr/local/share的文件删除程序；
3. 编译安装过程，可以设定参数，根据需求进行安装，并且自己选择安装版本，灵活性比较大；
4. 可以更清楚地理解自己的操作，发现安装时存在的问题，便于运维。

二、数据库安装所需步骤：

1. 编译前准备：

登录弹性云服务器

创建数据库安装用户及属组并设计密码

创建数据库源码存放及安装路径

下载第三方编译库、解压重命名

下载数据库源码

上传cmake包并进行解压

使用yum安装依赖包

替换python版本

修改用户所属组及权限

1. 编译及安装：

设定omm用户环境变量

添加omm用户环境变量并使其生效

进入数据库源码生成配置文件

make命令进行编译

make install命令安装

1. 初始化数据库

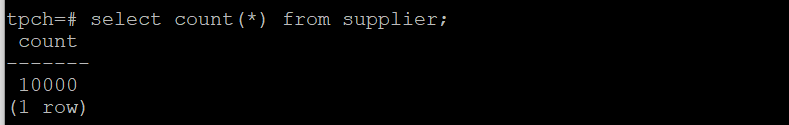
启动、登录、查询

# 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：

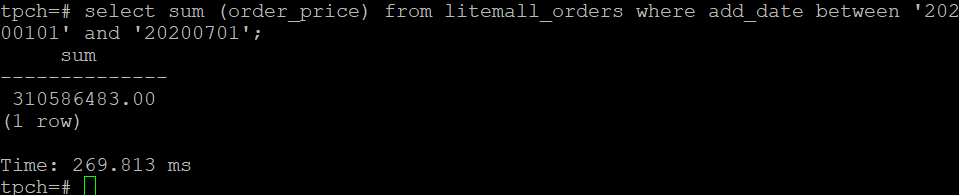
select count(\*) from supplier;;



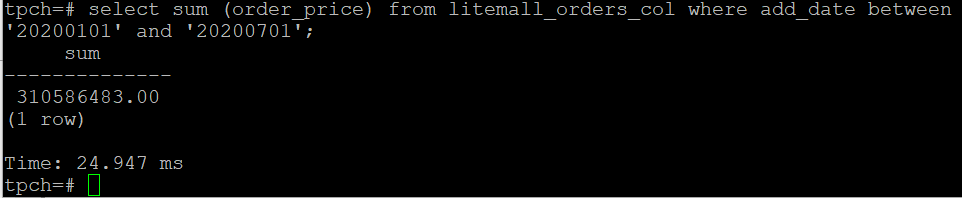
任务二：行存表与列存表执行效率对比

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';

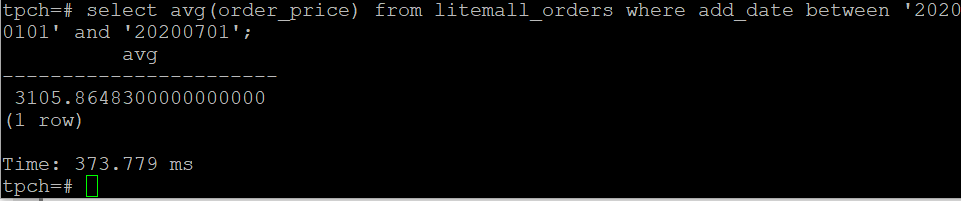


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

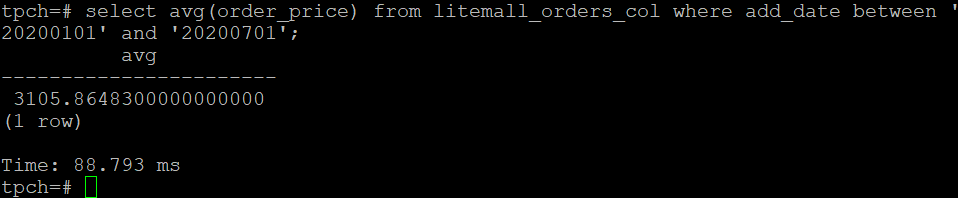


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

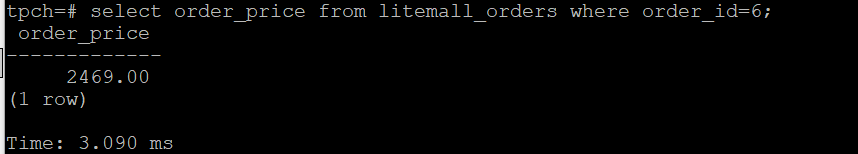


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

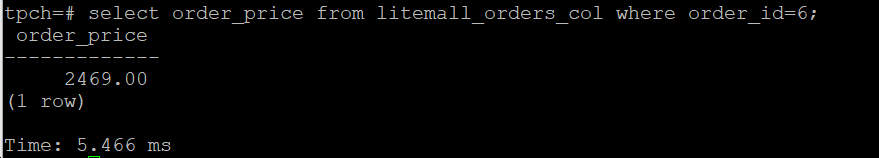


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

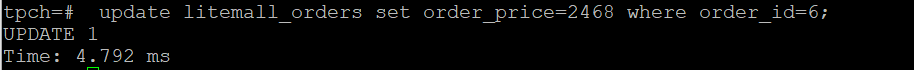


select order\_price from litemall\_orders\_col where order\_id=6;

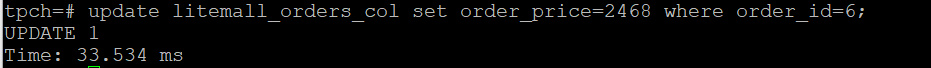


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



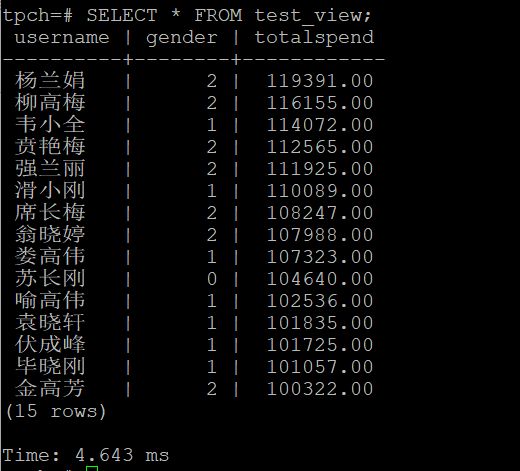
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

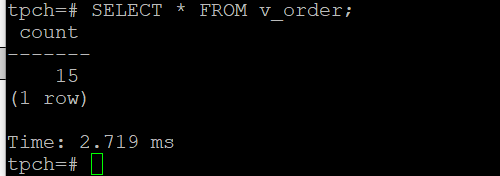
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



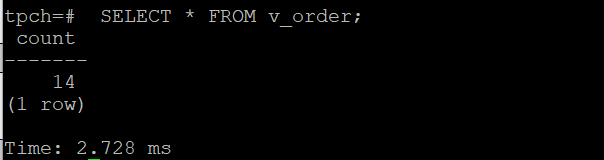
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



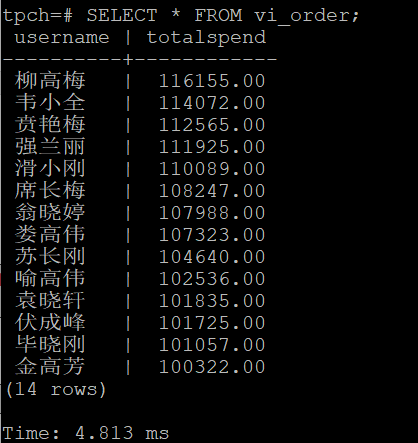
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



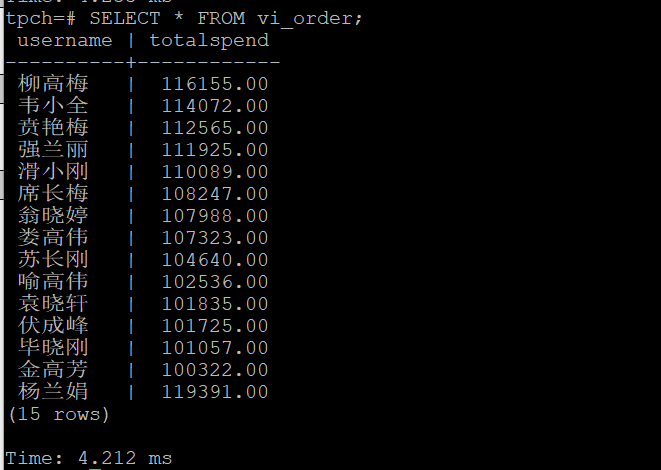
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

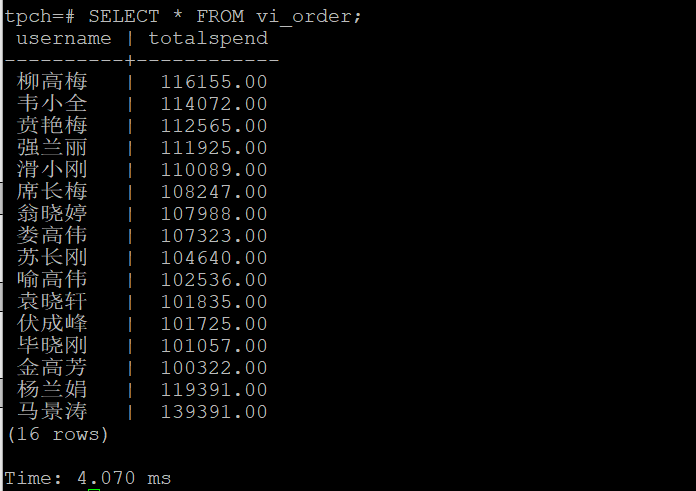
SELECT \* FROM vi\_order;



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？

**行存储及列存储的执行时间对比如图：**

|  |  |  |
| --- | --- | --- |
| 操作 | 行存储表格耗时 | 列存储表格耗时 |
| order\_price总和 | 269.813ms | 24.947ms |
| order\_price平均值 | 373.779ms | 88.793ms |
| 条件查询 | 3.090ms | 5.466ms |
| order\_price修改 | 4.792ms | 33.534ms |

在执行选取行中所有属性的条件查询、修改操作中，行存储的效率更高；

在执行求和、求平均等操作中，列存储的效率更高。

**行存储及列存储的区别：**

1. 数据写入上：

1）行存储的写入是一次完成，可以保证写入过程的成功或者失败，从而确定数据完整性。

2）列存储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，实际时间消耗更大。

3）数据修改是对磁盘上的记录做删除标记。行存储是在指定位置写入一次，列存储是将磁盘定位到多个列上分别写入，这个过程仍是行存储的列数倍。

1. 数据读取上：

1）数据读取时，行存储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，则会存在冗余列。为缩短处理时间，通常在内存中消除冗余列。

2）列存储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。

3）由于列存储的每一列数据类型是同质的，不存在二义性，便于数据解析。行存储在一行记录中保存了多种类型的数据，数据解析需要在多种数据类型之间频繁转换，比较消耗CPU。

1. 优缺点：
   * + 1. 行存储

行存储写入一次性完成，效率高；

保证数据完整性；

数据读取存在冗余，影响计算速度；

* + - 1. 列存储：

读取过程不存在冗余；

可针对某列数据类型、数据量大小等选择压缩算法，提高物理存储效率；

若某行的某一列无数据，列存储可以不用存储该列值，节省存储空间；

数据缺乏完整性；

数据写入效率低；

**行存储及列存储的应用场景：**

1. 列存储：
   1. 查询过程中，可针对各列的运算并发执行(SMP)，在内存中聚合完整记录集，降低查询响应时间;
   2. 可在数据列中高效查找数据，无需维护索引，查询过程中能够尽量减少无关IO，避免全表扫描;
   3. 因为各列独立存储，且数据类型已知，可以针对该列的数据类型、数据量大小等因素动态选择压缩算法，以提高物理存储利用率;如果某一行的某一列没有数据，那在列存储时，就可以不存储该列的值，这将比行式存储更节省空间；
   4. 主要适用于OLAP。
2. 行存储：
   1. 适合随机的增删改查操作；
   2. 需要在行中选取所有属性的查询操作；
   3. 需要频繁插入或更新的操作，其操作与索引和行的大小更为相关；
   4. 主要适用于OLTP。

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：

仅支持对创建好的物化视图做全量更新；

Select \* from返回得到的是整个表格的行数count;

1. 增量物化视图：

可以对物化视图进行增量刷新；

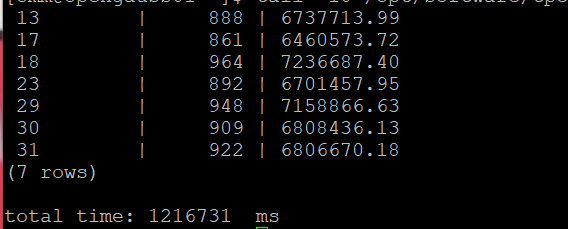
Slect \* from得到的是整个表格的返回结果；

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

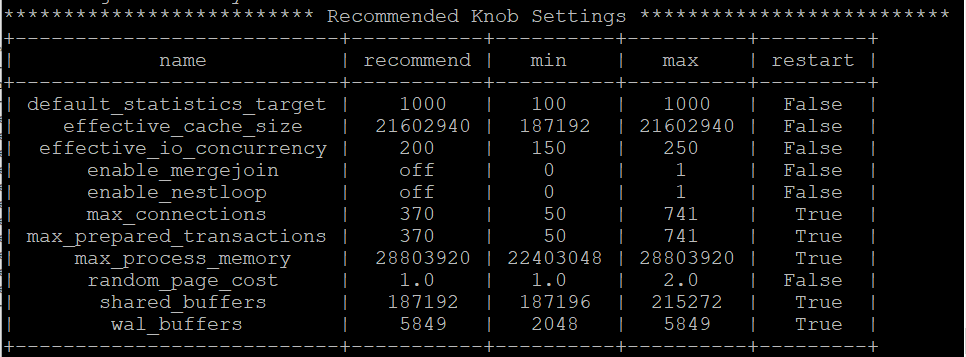
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

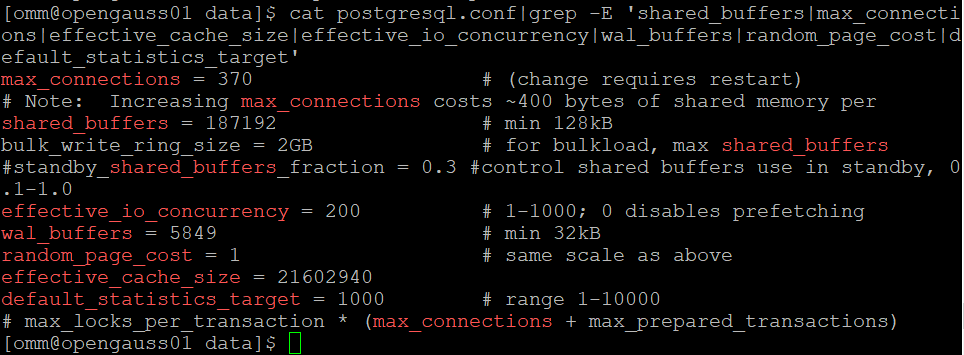
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

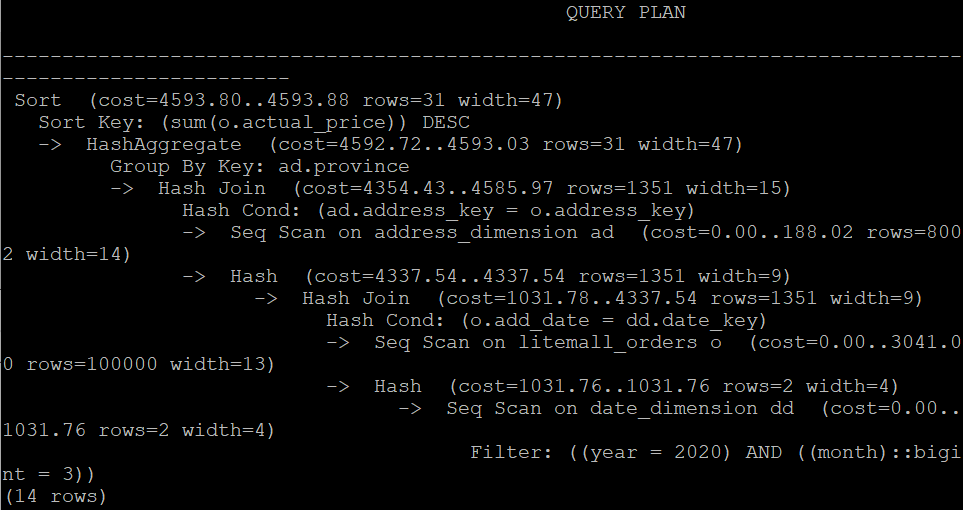
AND o.add\_date = dd.date\_key

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

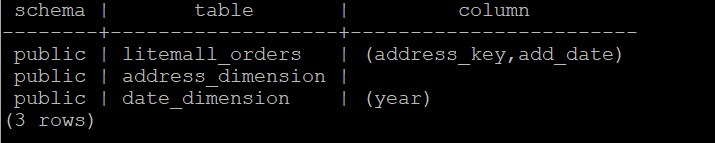
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

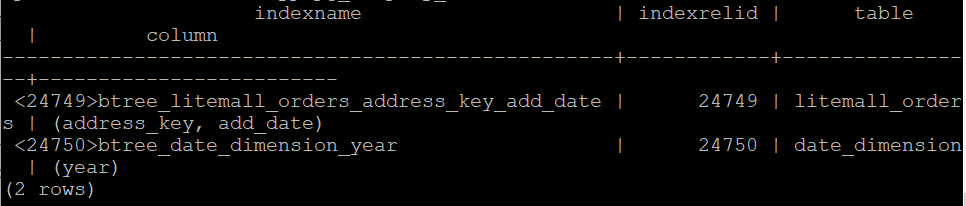
GROUP BY ad.province

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。

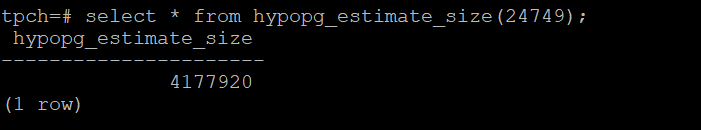
select \* from hypopg\_display\_index();

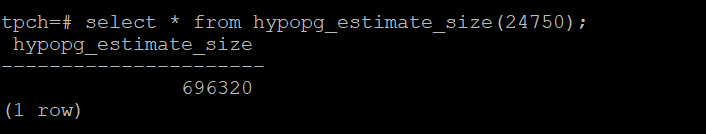


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

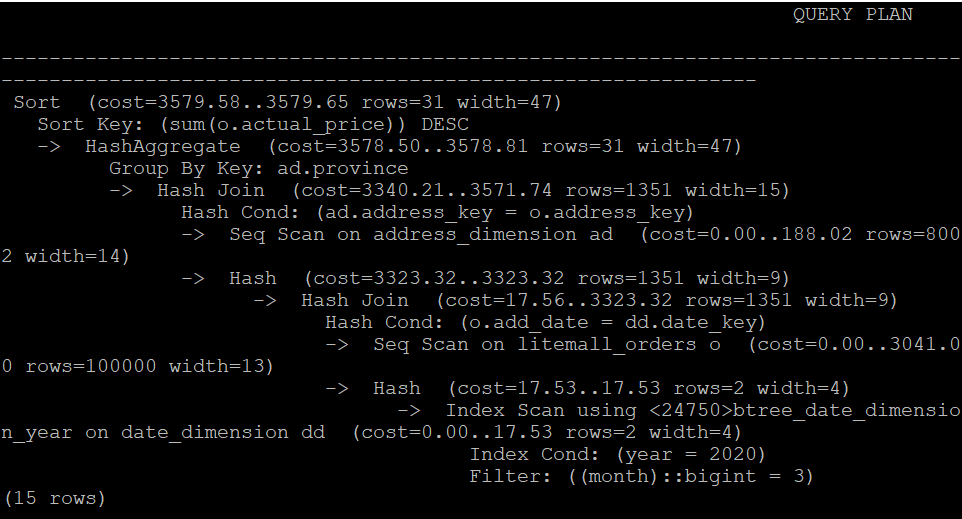
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

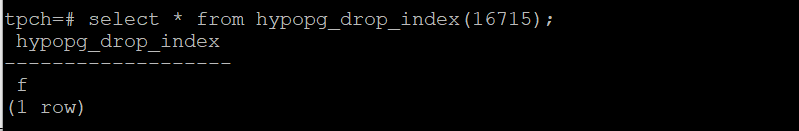
GROUP BY ad.province

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



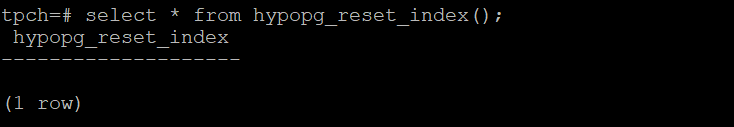
6. 删除某一个索引虚拟列，将执行结果截图。

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



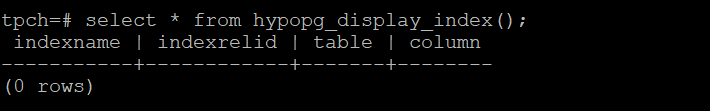
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

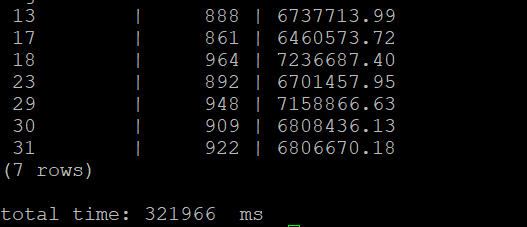
select \* from hypopg\_display\_index();



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

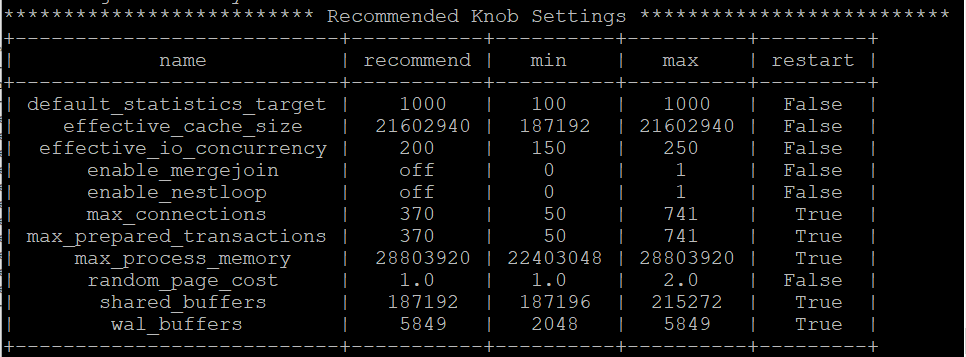


挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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？



根据对数据库的诊断了解到数据库metric中相关的指标，根据该指标的结果形成参数调优的推荐列表，从而提升数据库的性能。

调整的各项参数的含义：

Defult\_statistics\_target:采样颗粒度，越高统计估算的准确性越高，消耗时间也更长；

Effective\_cache\_size:单个查询的磁盘缓存的有效大小，值越大，使用索引扫描的可能性越大，反之，使用顺序扫描的可能性越大；

Effective\_io\_concurrency: 设置PostgreSQL可以同时被执行的并发磁盘 I/O 操作的数量;

Enable\_mergejoin: 允许或禁用查询计划程序对合并联接计划类型的使用（一般在对查询进行故障排除时使用）；

Enable\_nestloop: 启用或禁用查询计划程序使用嵌套循环联接计划，最终可能导致无法有效使用索引；

Max\_connections: 参数用来设置最大连接（用户）数，增加该参数并不会占据很多资源；

Max\_prepared\_transactions: 设置可以同时处于“准备”状态的最大事务数。设置为0可防止意外创建预备事务，若使用预备事务，该参数至少和max\_connections一样大；

Max\_process\_memory:数据库可用内存上限；

Random\_page\_cost: 设置计划程序对非按顺序读取的磁盘页的成本的估计（默认值为 4.0），参数值越低，系统首，选索引扫描；数据完全位于缓存中或相对于顺序具有较低随机读取成本的存储，可以使用较低的random\_page\_cost值；

Shared\_buffers:存储索引、表数据、执行计划的数组，访问数据先加载到os再加载到shared\_buffers，查找数据先从shared\_buffers中查看是否有缓存再到os中找；

Wal\_buffers: 用于尚未写入磁盘的 WAL 数据的共享内存量，提高该值可以提高许多客户端同时提交的繁忙服务器上的写入性能，但由于WAL 缓冲区的内容在每次事务提交时都会写出到磁盘，不宜过大。

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？

**使用索引对SQL的好处：**

1、通过唯一性索引（unique）可确保数据的唯一性；

2、加快数据的检索速度；

3、加快表之间的连接；

4、减少分组和排序时间；

5、使用优化隐藏器提高系统性能。

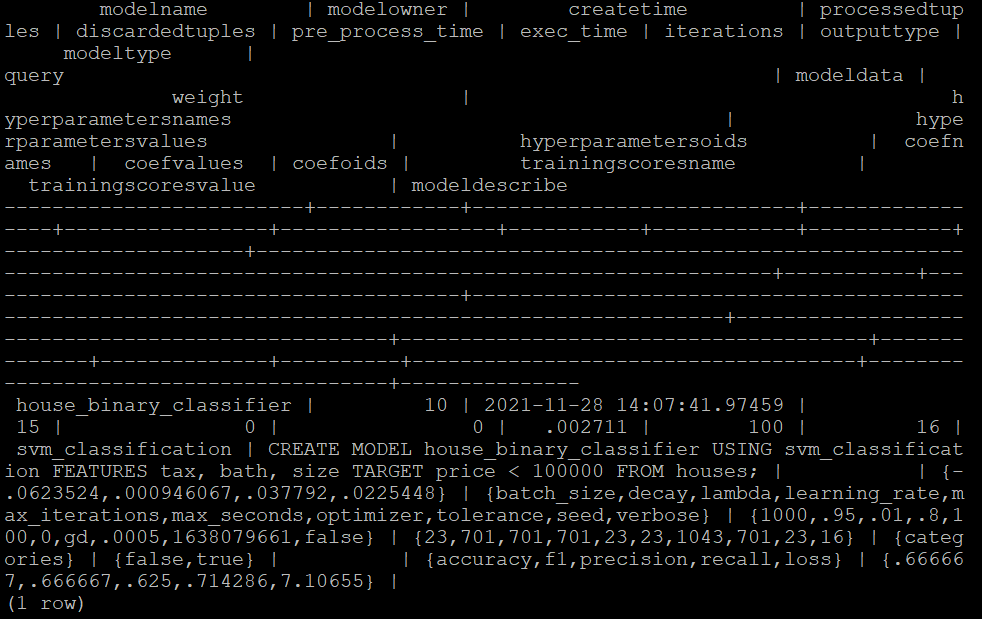
**其他优化数据库方法：**

1. 选取合适的字段属性；
2. 使用联合（union）代替手动创建的临时表；
3. 事务查询；
4. 锁定表；
5. 使用外键；
6. 优化查询语句：比如：
   1. 使用连接（join）替代子查询；
   2. Where句子避免使用!=和< >；
   3. Where句子使用or优化
   4. Where句子使用in、not in优化；
   5. Where句子使用is null或is not null优化；

# 关卡四、openGauss的DB4AI特性应用

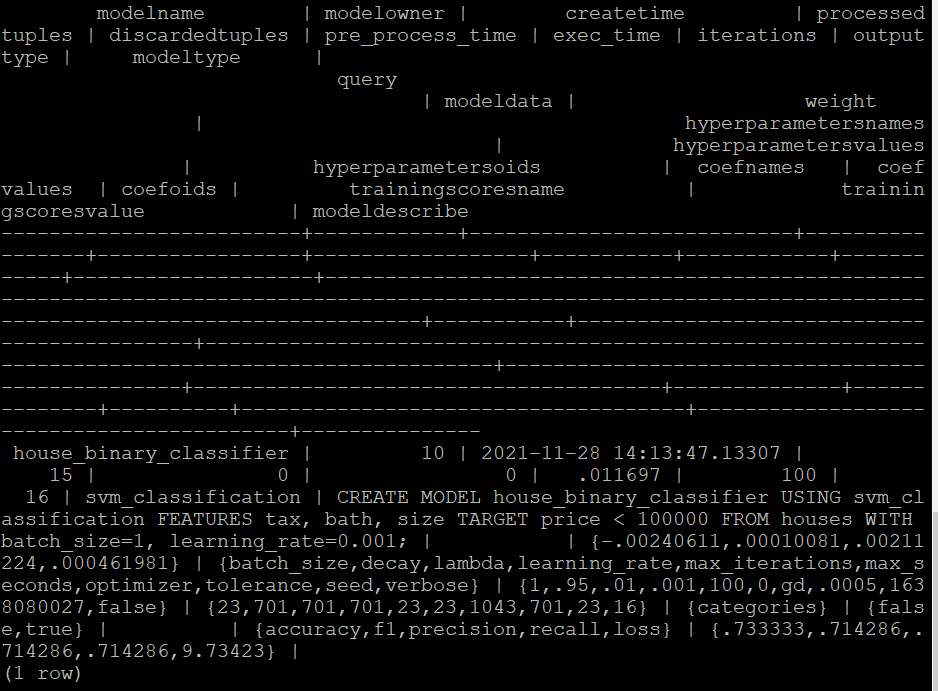
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



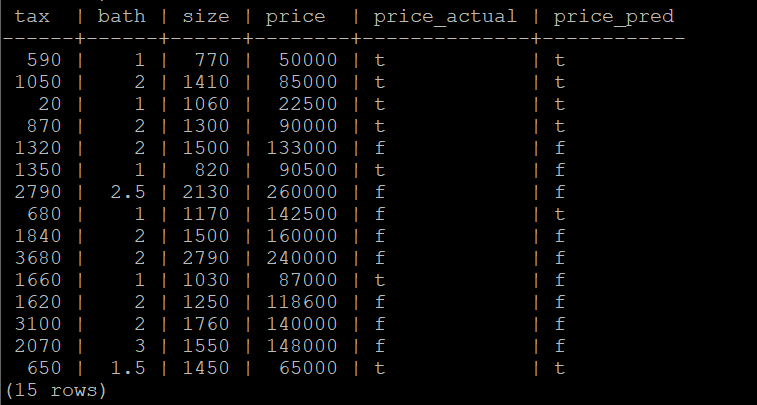
任务二：观察新模型的信息，将执行结果截图。

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？

|  |  |  |
| --- | --- | --- |
| 特性 | 分类 | 回归 |
| 输出类型 | 离散数据 | 连续数据 |
| 目的 | 寻找决策边界 | 找到最优拟合 |
| 评价方法 | 精度、混淆矩阵 | SSE或拟合优度 |

实践思考题2：什么是SVM算法？

SVM又叫支持向量机，模型可以找到多个可以分类的超平面将数据分开，并使得数据和超平面保持一定距离，即找到能使支持向量和超平面间最小距离的最大值，最终实现数据分类（其中：和超平面平行的保持一定的函数距离的这两个超平面对应的向量，我们定义为支持向量）。

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. 混淆矩阵：

|  |  |
| --- | --- |
| TP | 实际为正，判定为正 |
| FP | 实际为负，判定为正 |
| FN | 实际为正，判定为负 |
| TN | 实际为负，判定为负 |

* 1. Accuracy:
  2. precision:
  3. recall:
  4. F1：

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. 平均绝对值误差（MAE）：
2. 均方误差（MSE）：
3. 均方根误差（RMSE）：
4. :