openGauss AI特性创新实践课



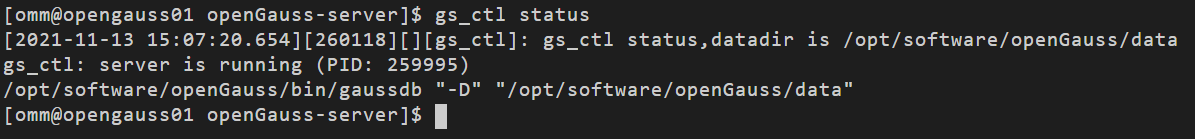
华为技术有限公司

# 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：

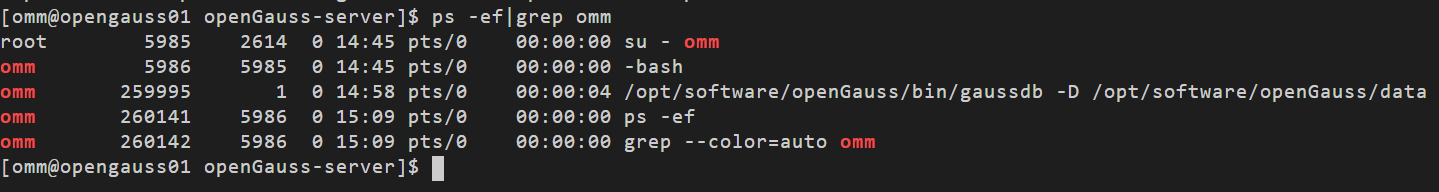
任务一：数据库状态验证

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？简述安装步骤

通过源码编译的时候配置环境，可以方便定位到依赖包所在的具体位置。因为不同操作系统源码编译环境是不一样的，导致数据库需要的依赖包在主机位置不同。

而且通过源码安装，比使用镜像文件简单，速度快，可读性更强。

安装步骤：

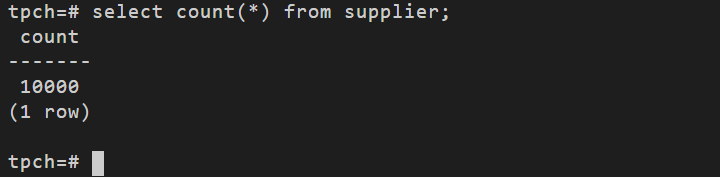
1. 配置华为云环境。先购买openEuler弹性云服务器，然后创建VPC环境，配置VPC，子网和安全组。
2. 通过源码安装部署opengauss数据库。
3. 在编译前登录ECS，创建用户组、用户，下周源码，创建目录。
4. 解压源码，检查python版本（为什么需要python？）,检查权限。
5. 安装编译，ok

# 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：

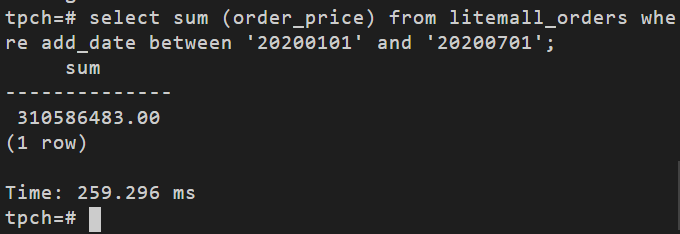
select count(\*) from supplier;;



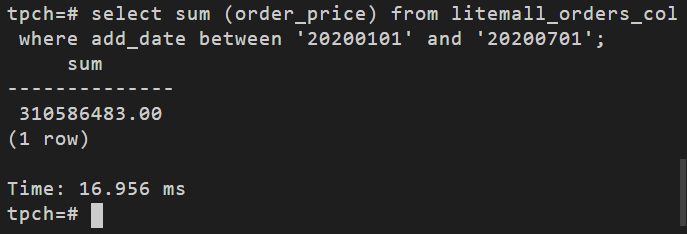
任务二：行存表与列存表执行效率对比

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';

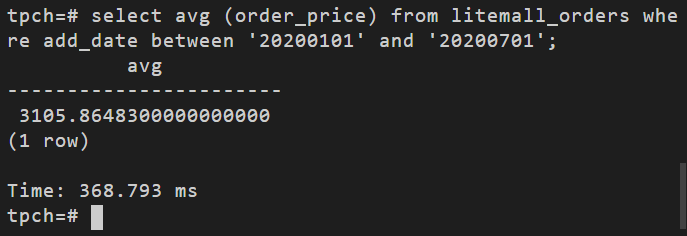


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

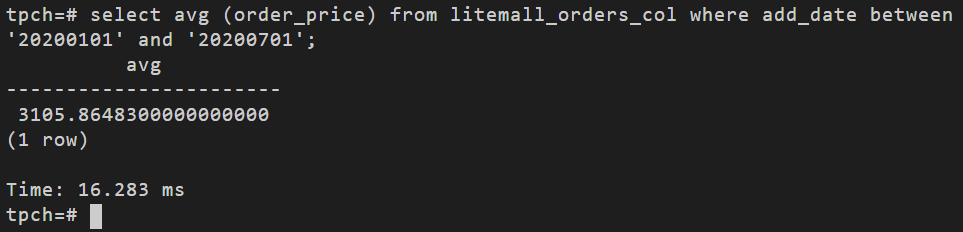


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

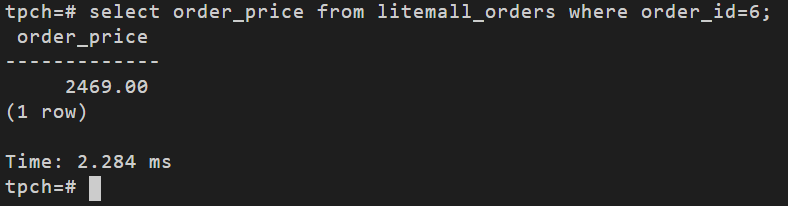


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

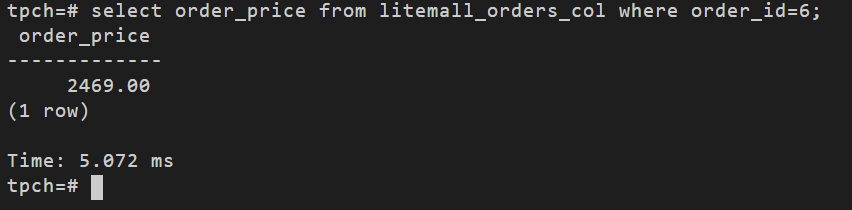


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

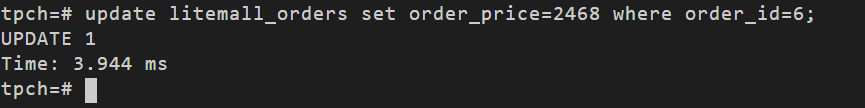
select order\_price from litemall\_orders where order\_id=6;



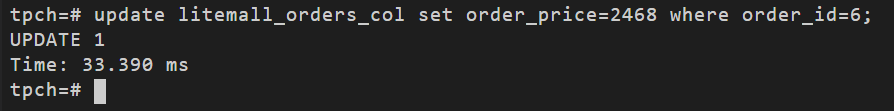
select order\_price from litemall\_orders\_col where order\_id=6;

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;



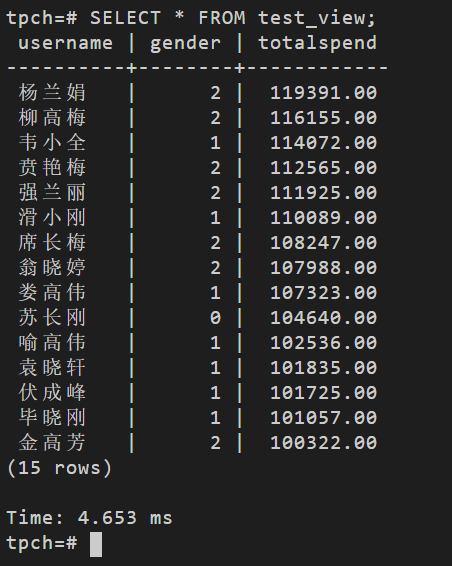
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

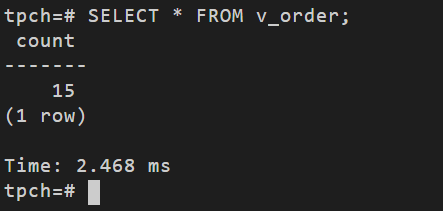
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



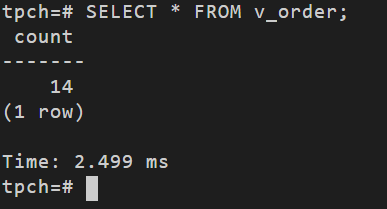
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



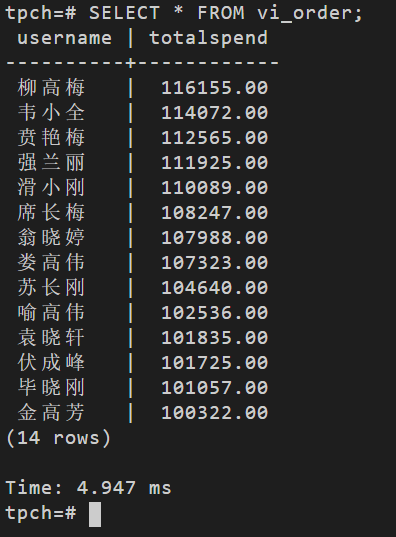
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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？

执行时间不同因为行存表与列存表的数据存储方式和执行机制不同。

行存表的数据是按行存储的，没有索引的查询使用大量I/O。比如一般的数据库表都会建立索引，通过索引加快查询效率。在执行随机增删改查操作，或需要在行中选取某个属性或所有属性的查询，行存储效率更高。

列存表数据按列存储，数据就是索引，当进行查询只访问查询涉及的列，可以大量降低IO，且每个列由一个线程处理，查询并发处理性能高。在执行只需要对几列数据进行查询，数据类型一致，特征相似，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？

增量物化视图可以对物化视图增量刷新，需要手动执行语句完成对物化物化视图更新。

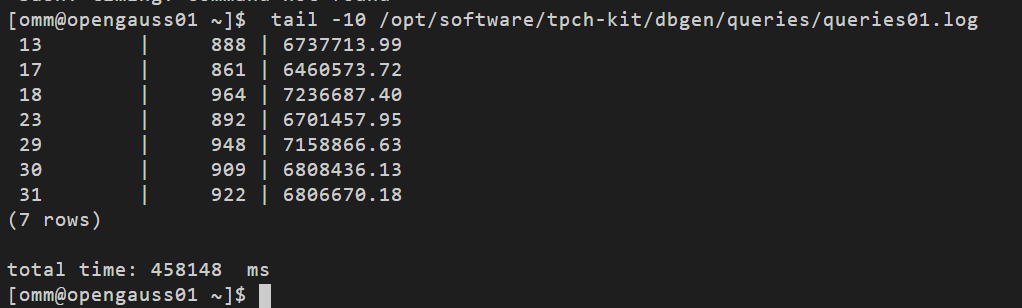
全量物化视图仅支持对创建好的物化视图做全量更新，而不支持做增量更新。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

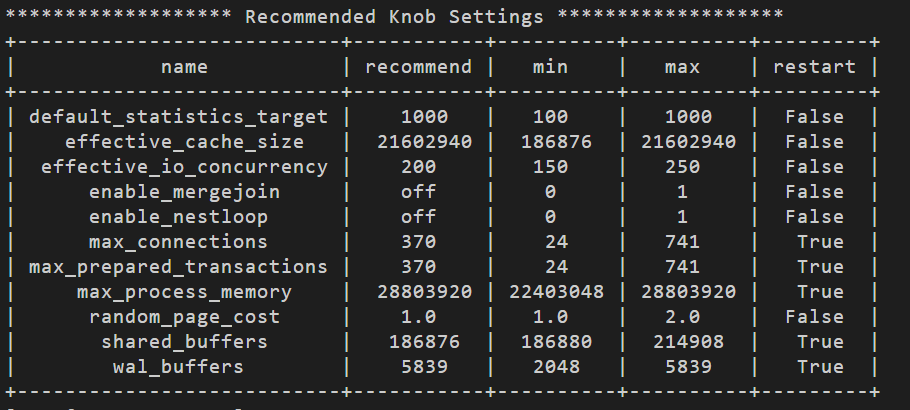
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

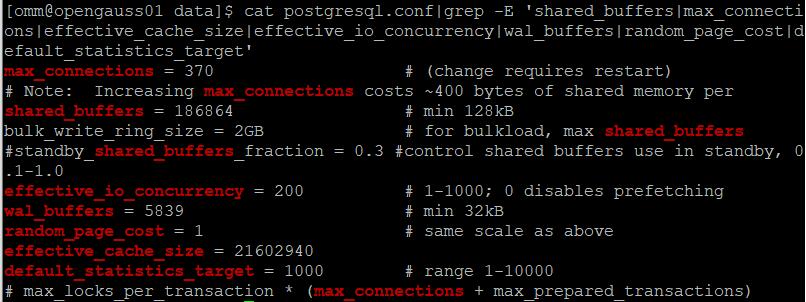
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

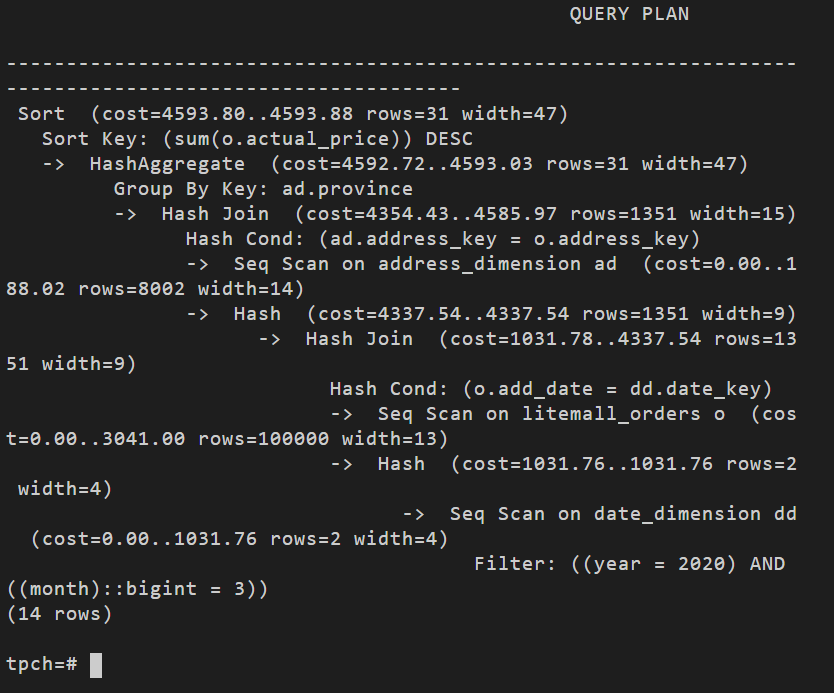
AND o.add\_date = dd.date\_key

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

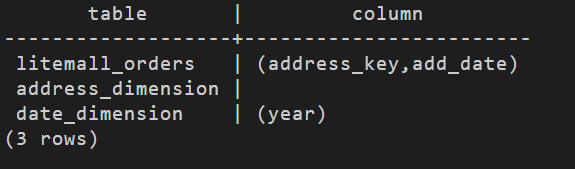
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

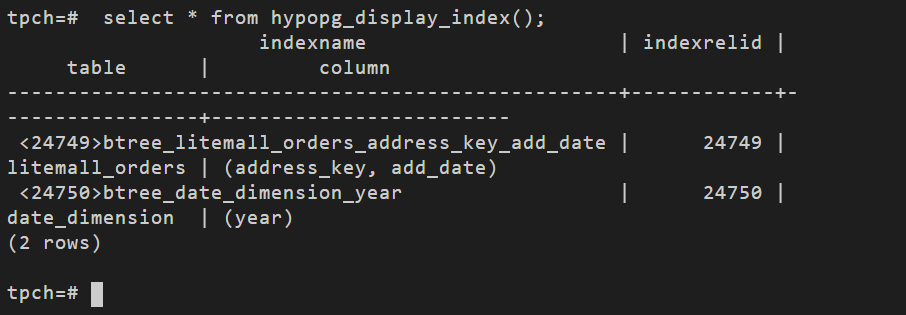
GROUP BY ad.province

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。

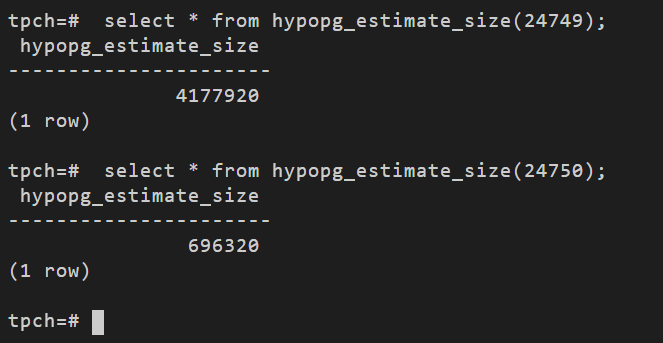
select \* from hypopg\_display\_index();



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

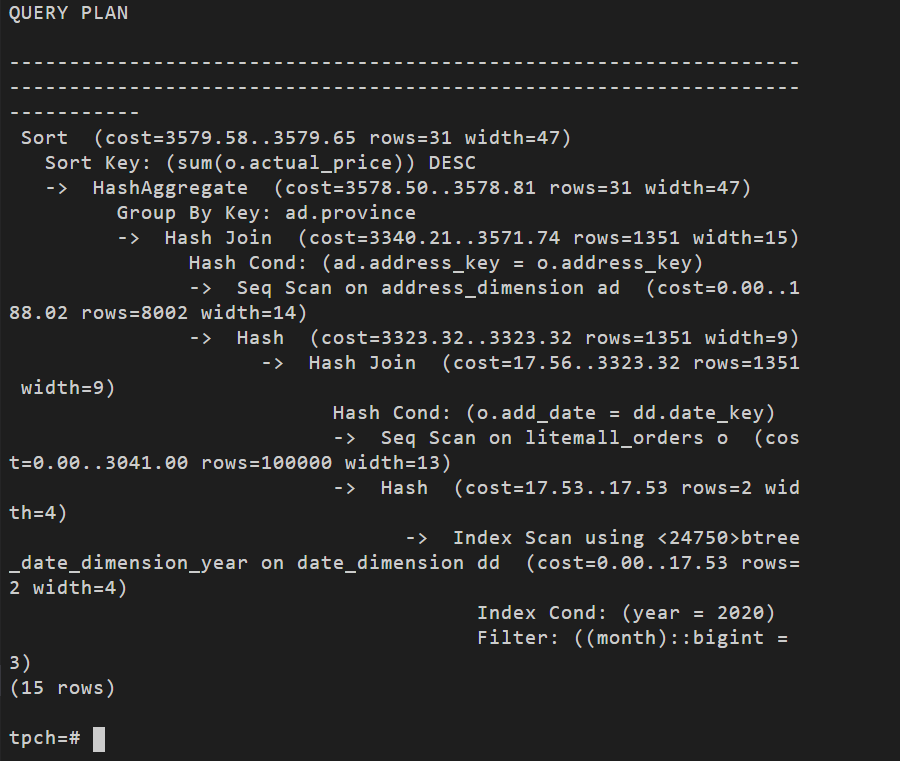
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

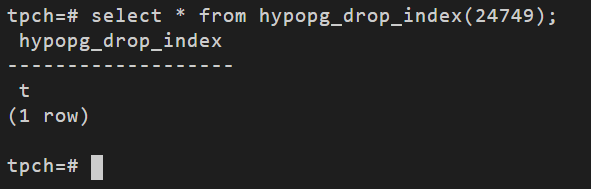
GROUP BY ad.province

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



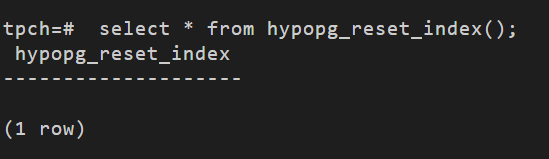
6. 删除某一个索引虚拟列，将执行结果截图。

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



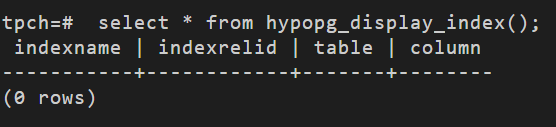
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

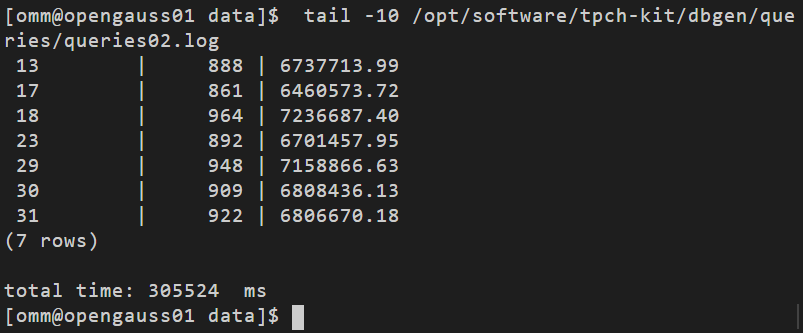
select \* from hypopg\_display\_index();



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：

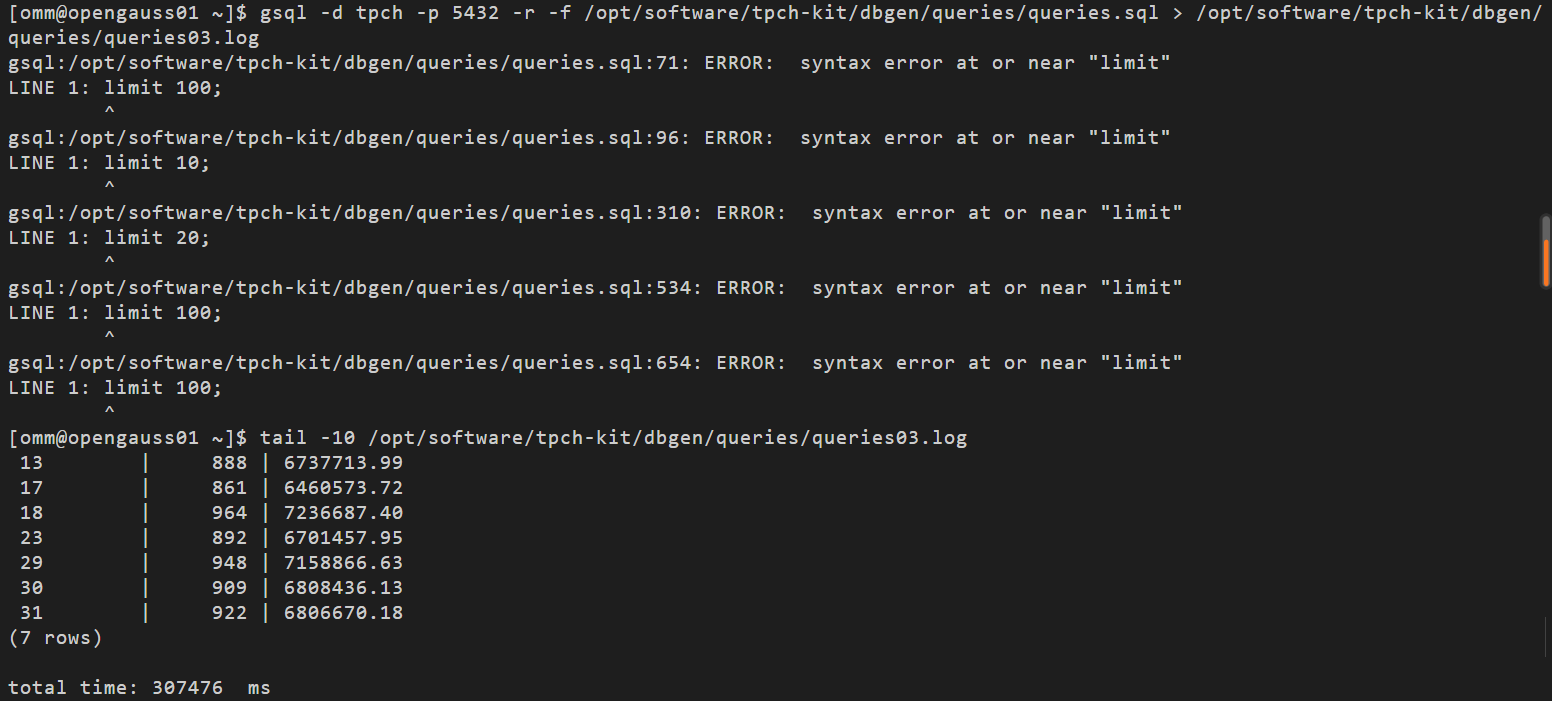
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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：

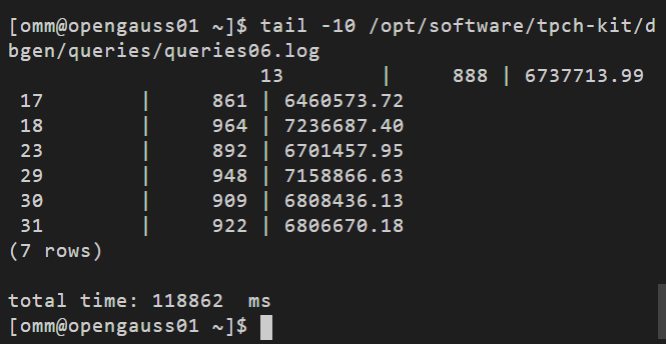
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries03.log



对queries.sql的多个语句创建了虚拟索引，但是耗费时间并没有减少。猜测是数据量的问题，数据量不是很大，所以即使增加了索引，也没有加快效率。

虽然优化索引不可以提高性能，那么可以提高数据库并行效率，使用SMP提高数据库性能。

在这里设置 query\_dot=6



可以看的运行时间缩短到了118862ms，效率提升了3倍。

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？

在shared\_buffers,max\_connections, effective\_cache\_size, effective\_io\_concurrency, wal\_buffers, random\_page\_cost, default\_statistics\_target这些参数上进行了优化。

Shared\_buffers参数决定数据库服务器缓存数据大小，max\_connections是最大连接数，effective\_io\_concurrency控制IO性能，effective\_cache\_size提供可用于磁盘高速缓存的内存量的估计值等。

优化这些数据库参数是为了提高数据库总运行性能。

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？

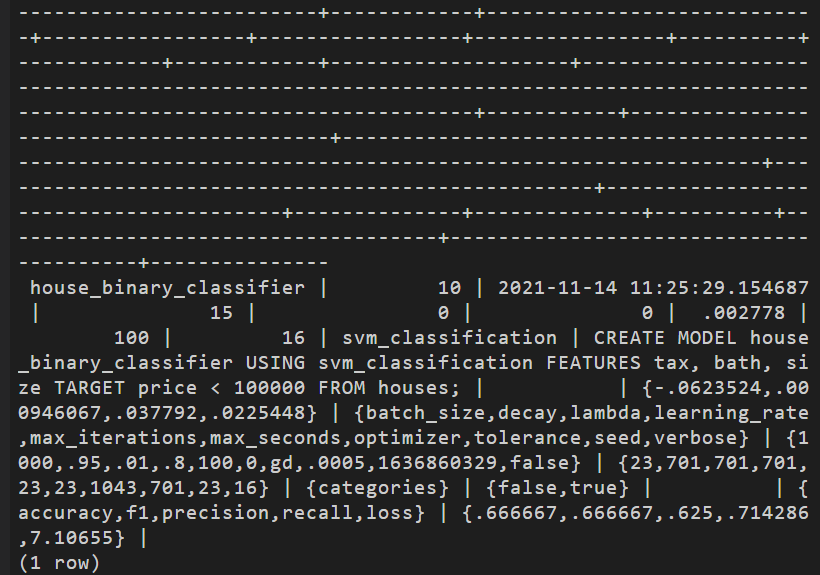
使用索引可以加快查找效率。如果数据库没有索引，在查找是使用顺序查找，时间fuzadu1是O(n)。添加索引后，在进行查询时会优先对添加的索引列进行查询，在检测到该列对应的值时就可以查询到该行，减少了全局遍历的次数以及数量。

还可以使用优化查询后的SQL语句以及在设计数据库时使用范式设计对数据库表进行优化。

# 关卡四、openGauss的DB4AI特性应用

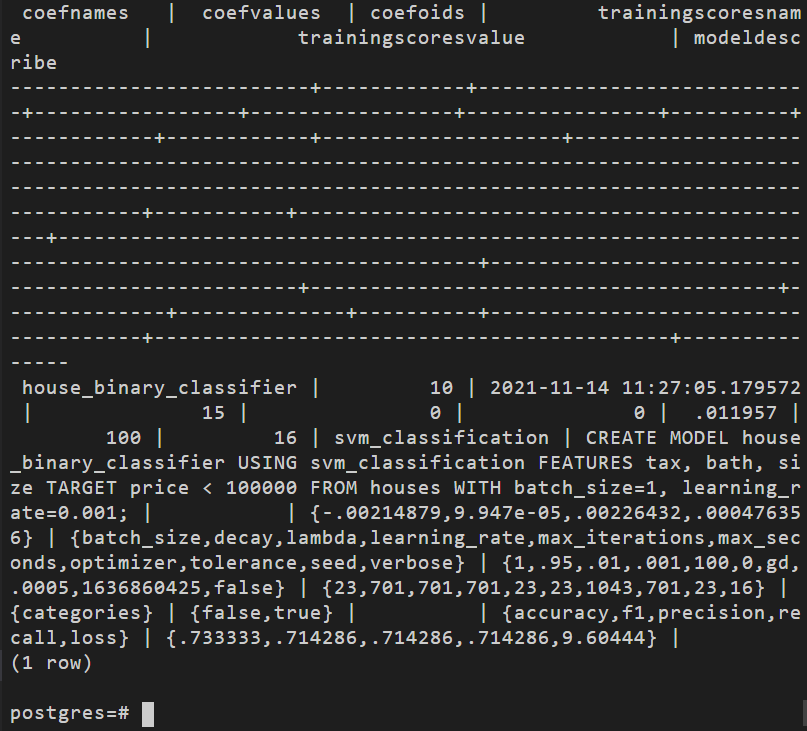
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



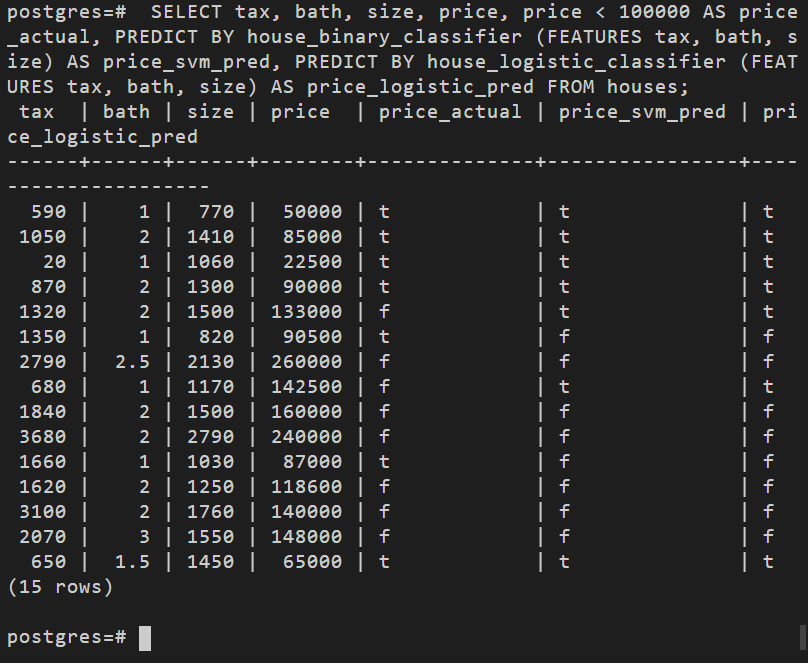
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型的标签值时离散的，回归模型的标签值是连续的。

实践思考题2：什么是SVM算法？

SVM算法是支持向量机算法，通过查找数据集中的一个总距离最小的超平面把数据进行划分。可以运用于分类与回归问题。

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？

精确率：自分类正确的正样本个数占分类器判定为正样本的样本个数的比例 。

召回率：分类正确的正样本个数占真正的正样本个数的比例 。

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？

均方误差：真实值与预测值之间的差求和平均。

决定系数：用来描述两个变量之间的相关性