openGauss AI特性创新实践课



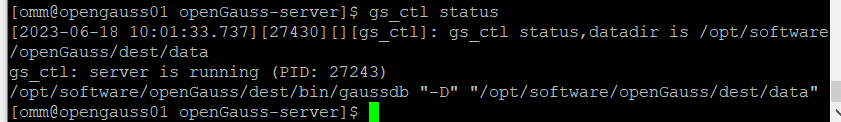
华为技术有限公司

# 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：

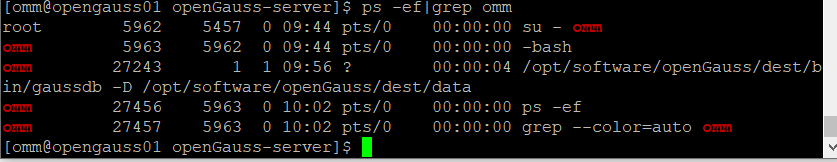
任务一：数据库状态验证

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？

安装mysql在编译期间需要配置，定制存储引擎等，软件安装不在默认路径需要特定的

路径也需要惊醒源码编译，素以指定数据库的安装路径，为更新删除数据库提供便利的

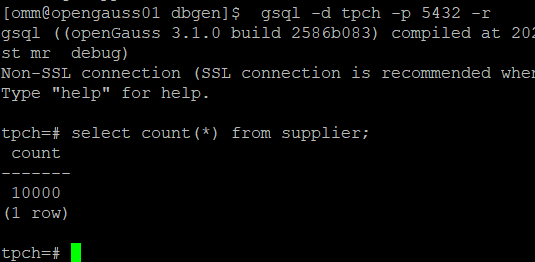
条件。类似于自定义安装。

# 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：

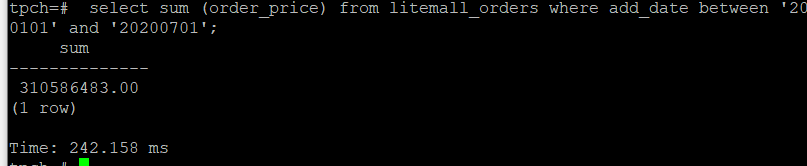
select count(\*) from supplier;;



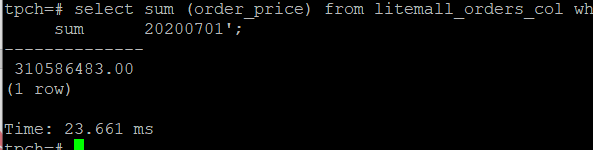
任务二：行存表与列存表执行效率对比

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';

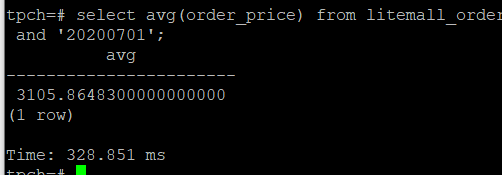


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

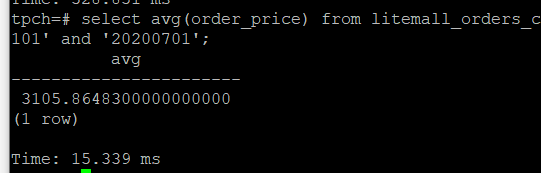


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

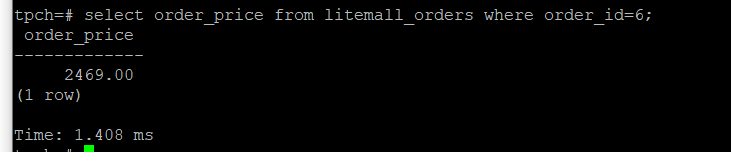


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

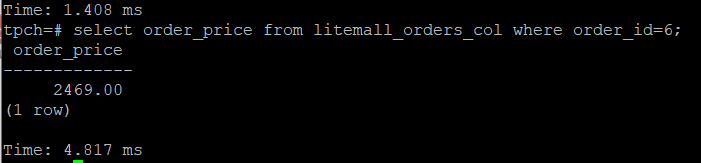


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

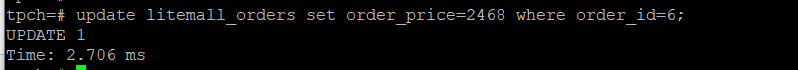


select order\_price from litemall\_orders\_col where order\_id=6;

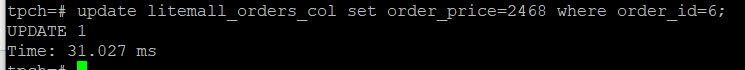


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



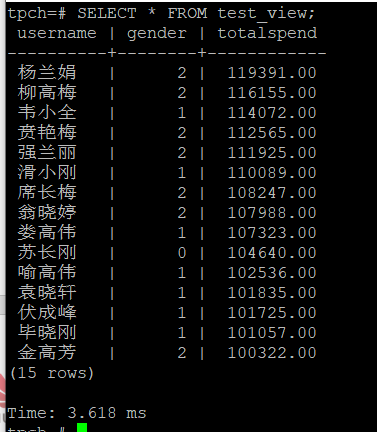
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

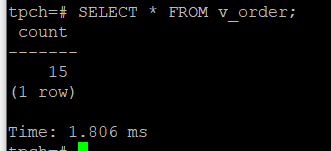
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



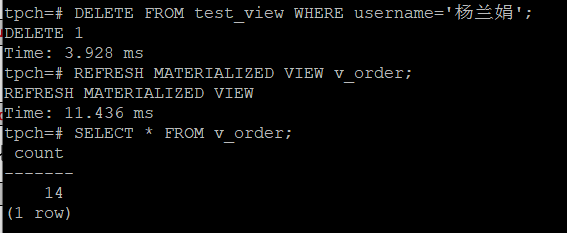
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



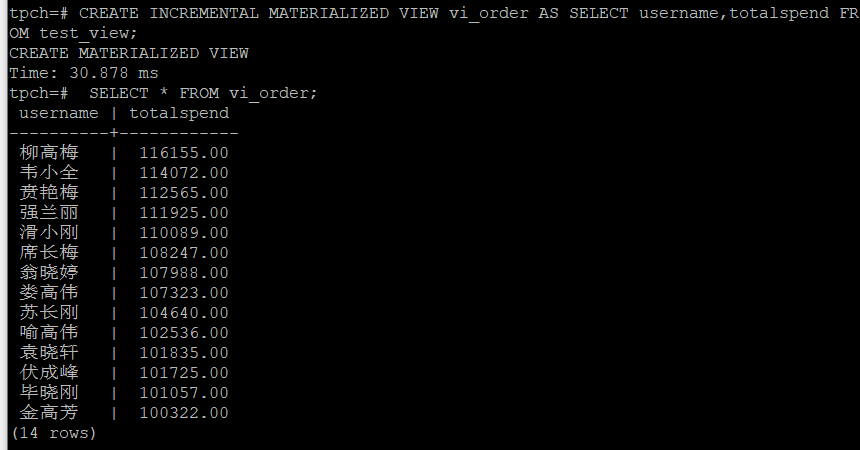
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



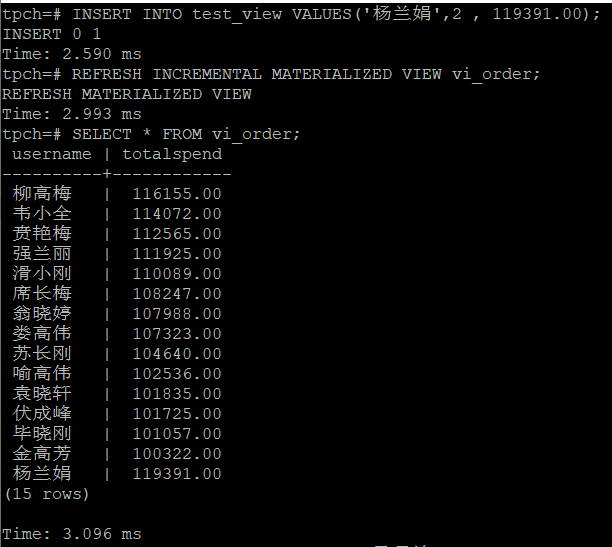
4. 创建增量物化视图，查询物化视图结果，将执行结果截图。

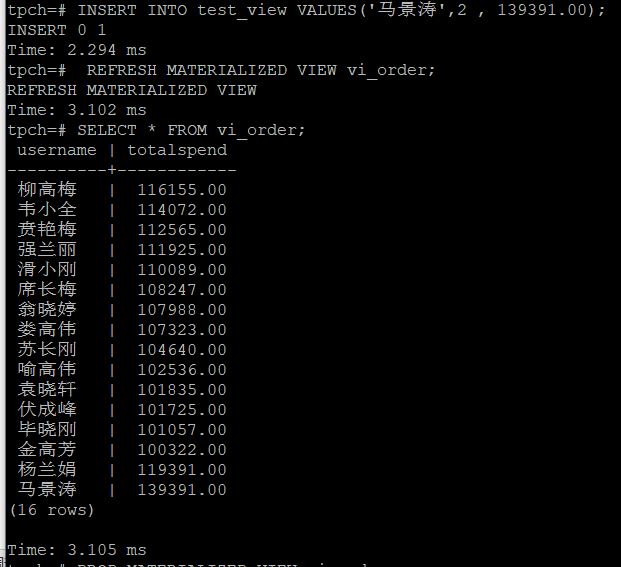
SELECT \* FROM vi\_order;



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？

行存表和列存表是两种不同的数据存储方式，在执行相同的SQL语句时可能会出现时间上的差异。行存表将每一行的所有数据都存储在一起。列存表将同一列的数据都存储在一起。

在进行数据操作时，行存表需要读取整行的数据，而列存表只需要读取相关列的数据。列存表将同一列的数据都存储在一起，在进行数据操作时可以更好地使用压缩和向量化等技术。

行存表在执行以下几种类型SQL时效率更高：

频繁更新和插入操作的数据：插入和更新的是所有列的信息，行存储是包含所有列的信息的，并且只需在当前页内的空闲区域写入新数据。

小规模数据的查询：当单次查询的数据量较少时，行存表优势明显，因为它可以快速地定位到一行数据并取出其中的所有列信息。

列存表适合执行以下类型的 SQL：

大批量更新和删除：对表中大量数据进行更新或删除时，列存储包含的数据更多。而且列存储可以跳过不需要的列，只请求要更新或删除的列。

数据仓库：列存储数据库更适合用作数据仓库。由于有些数据库多数包含非结构化数据，而列存储数据库能够更好地支持非结构化数据，并且在处理大量数据时具有更好的可伸缩性和性能。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

全量物化视图和增量物化视图在数据集中预先计算并缓存查询结果。他们的缓存方式和更新频率不同。

全量物化视图：全量物化视图是在一段时间后进行刷新。当指定事件或条件满足时，会对整个视图进行重新计算并更新存储结果。因此，它包含了所有的查询结果，不管是否需要或者数据是否过时。全量物化视图对数据量比较大的场景有很好的性能提升作用。

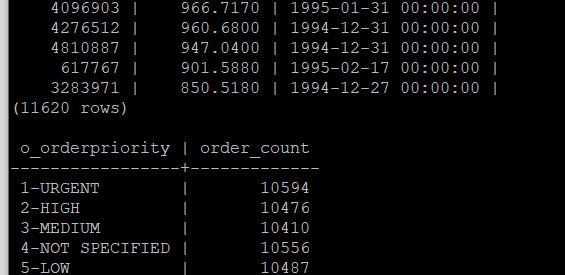
增量物化视图：增量物化视图则只保存部分的查询结果，并且它们的刷新频率更高。与全量物化视图相反，它们只包含那些经常被查询的数据，并且只在相关数据有变更时进行刷新。增量物化视图适合于频繁查询、数据量较小的场景。

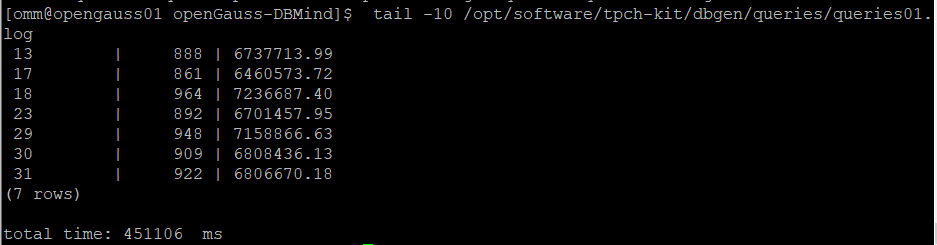
# 关卡三、openGauss的AI4DB特性应用

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：

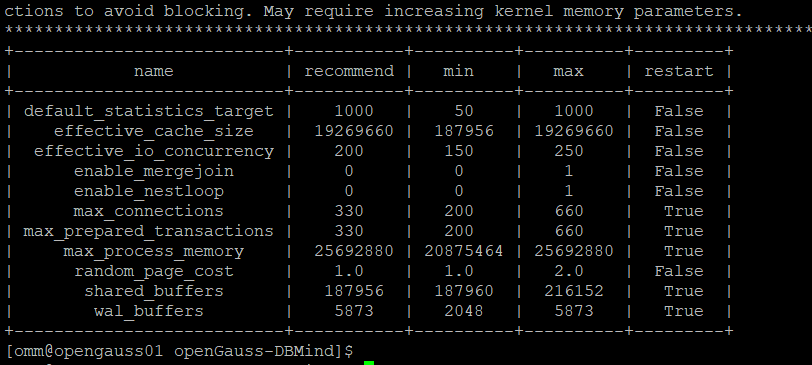
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log





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

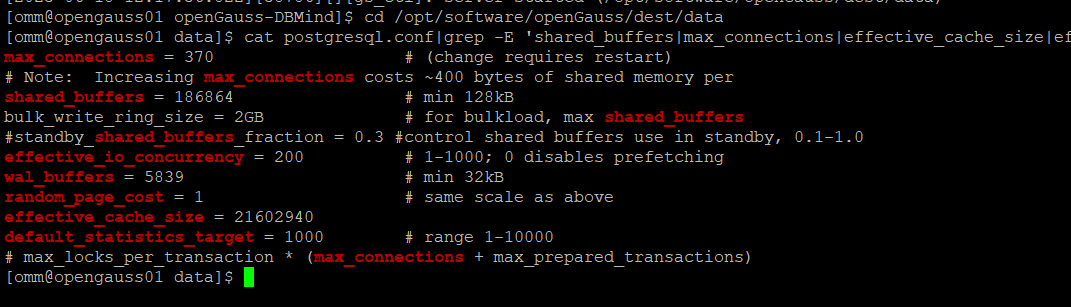
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

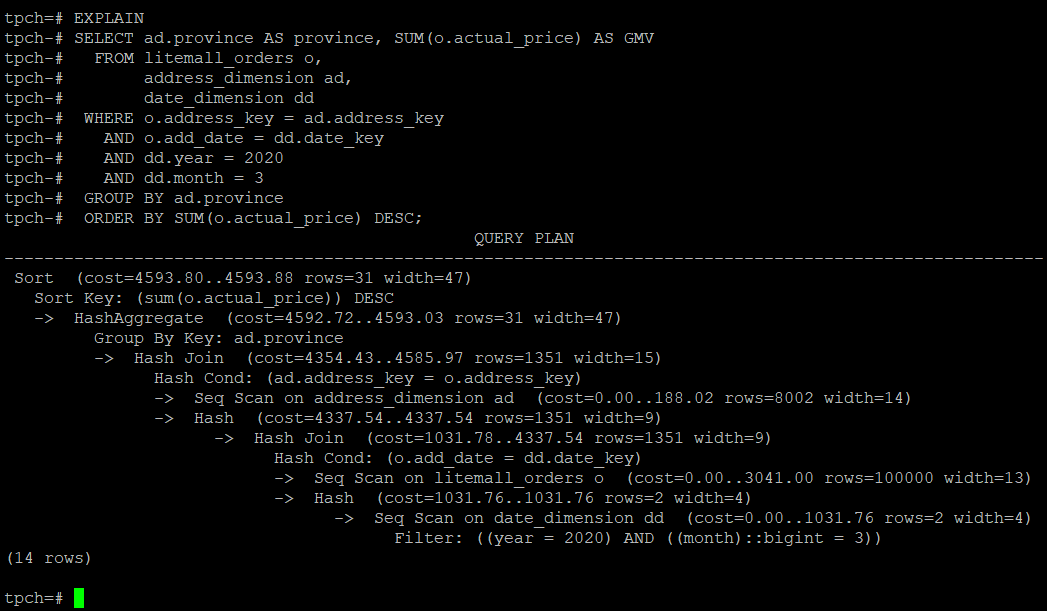
AND o.add\_date = dd.date\_key

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

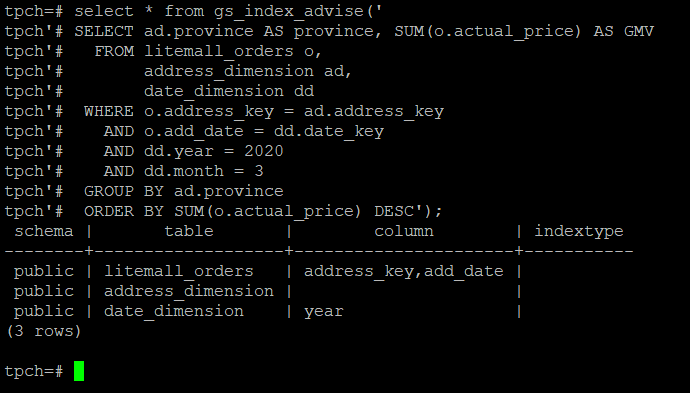
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

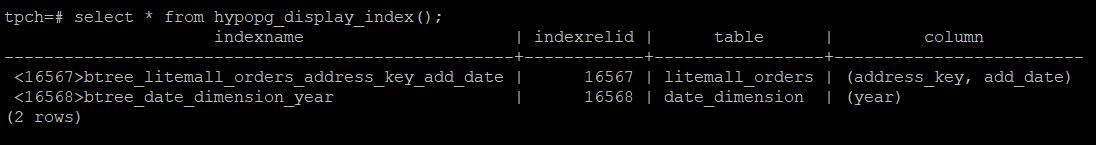
GROUP BY ad.province

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。

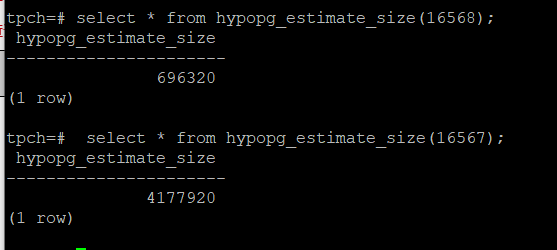
select \* from hypopg\_display\_index();



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

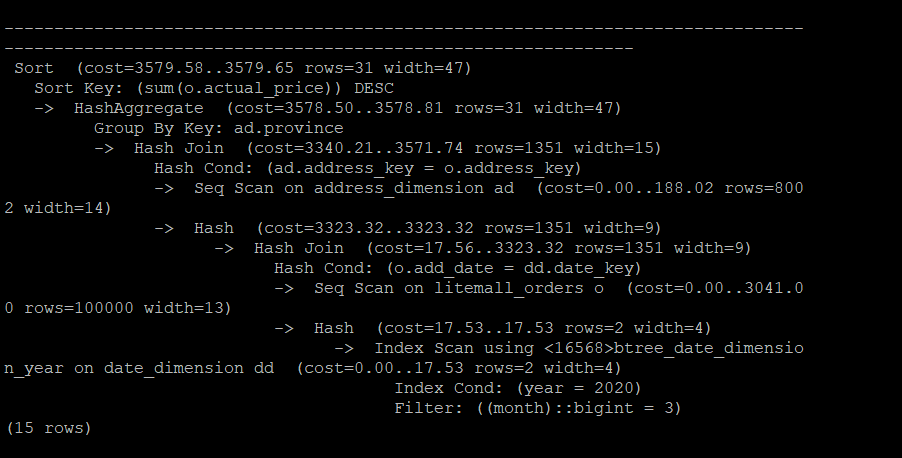
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

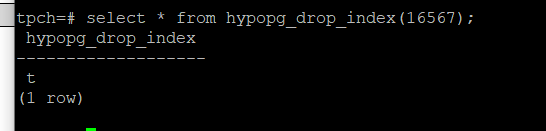
GROUP BY ad.province

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



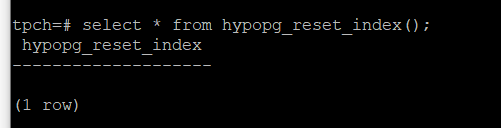
6. 删除某一个索引虚拟列，将执行结果截图。

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



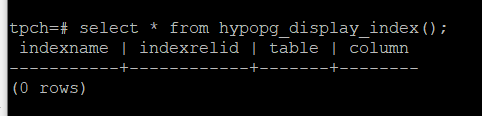
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

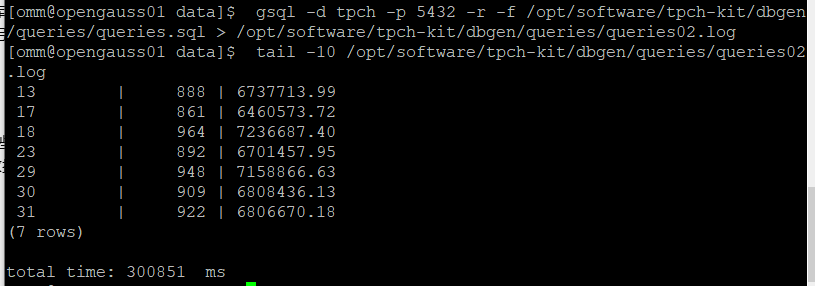
select \* from hypopg\_display\_index();



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：

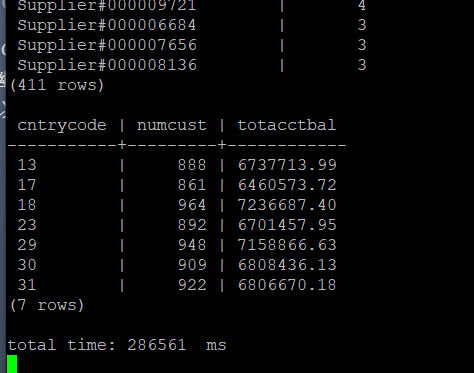
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

shared\_buffers: 将该值设置为18686，提高该参数大小可以增加数据库系统性能。这个参数用于指定数据库在内存中分配给共享缓存区域的最大字节数量。

max\_connections: 这个命令没有提供max\_connections的设置。max\_connections用来限制连接到数据库的最大并发数量。如果超过该数量，会拒绝新的连接请求，从而保护数据库免受资源饱和和崩溃等问题。

wal\_buffers: 将该值设置为5839，增加wal\_buffers大小有助于减少写日志期间的磁盘I/O操作。这个参数指定重新启动之前一个事务块产生的WAL记录缓冲池中持久存储的字节数。

random\_page\_cost: 将该值设置为1，会让Planner将随机IO的代价设置为与顺序IO相同。这个参数用于确定在随机读取每个块的情况下需要经过多少磁盘寻道时间，以及读取所需的额外CPU处理时间。

default\_statistics\_target: 将该值设置为1500，可以提高查询优化器的准确性。默认情况下，该参数设置为100，表示PostgreSQL收集用于调整查询计划的统计信息。但是，当该值太小时，收集的相应样本可能无法反映整个数据库，从而可能导致错误的查询计划。因此，增加值可以提高统计信息的精度，进而提高查询优化器的性能。

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？

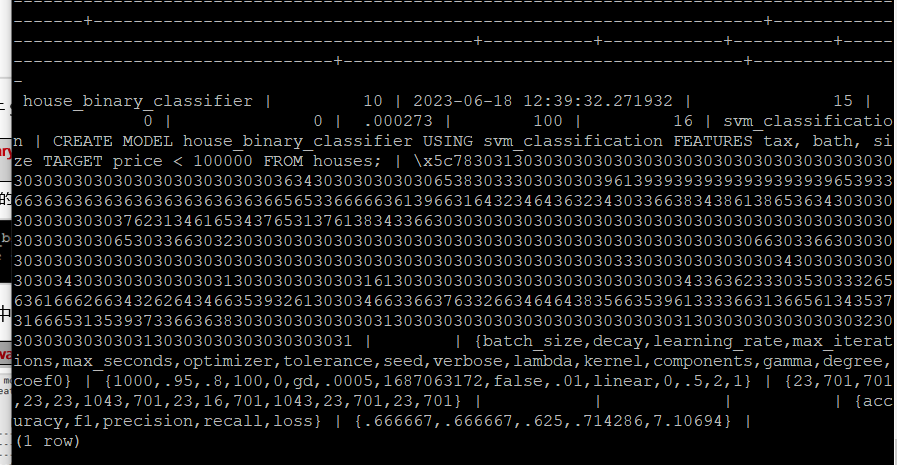
索引是一种用于加速数据库操作的数据结构，它可以使对数据库表的查找和筛选操作更加高效。使用索引可以减少查询中需要扫描的数据量，从而提升查询速度，并且可以降低数据库在处理大量数据时的索引是一种用于加速数据库操作的数据结构，它可以使对数据库表的查找和筛选操作更加高效。使用索引可以减少查询中需要扫描的数据量，从而提升查询速度，并且可以降低数据库在处理大量数据时的负载。

除索引外还有硬件配置，优化sql语句，优化数据库表的设计等。

# 关卡四、openGauss的DB4AI特性应用

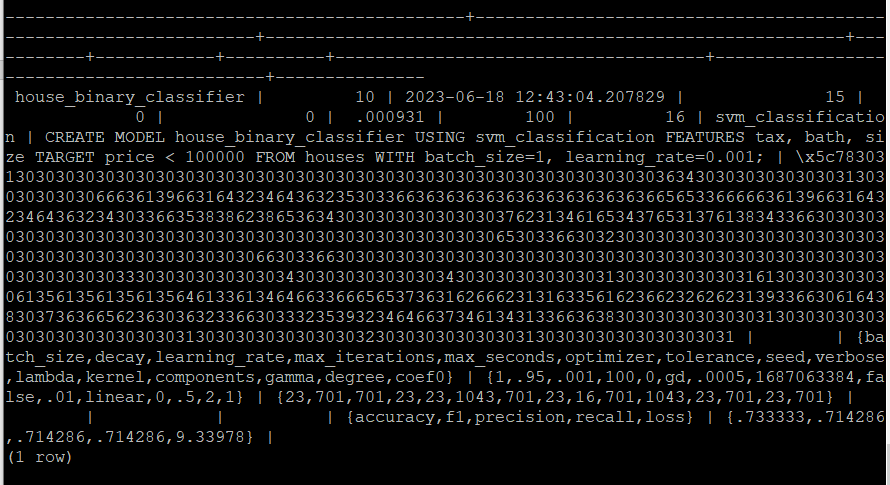
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



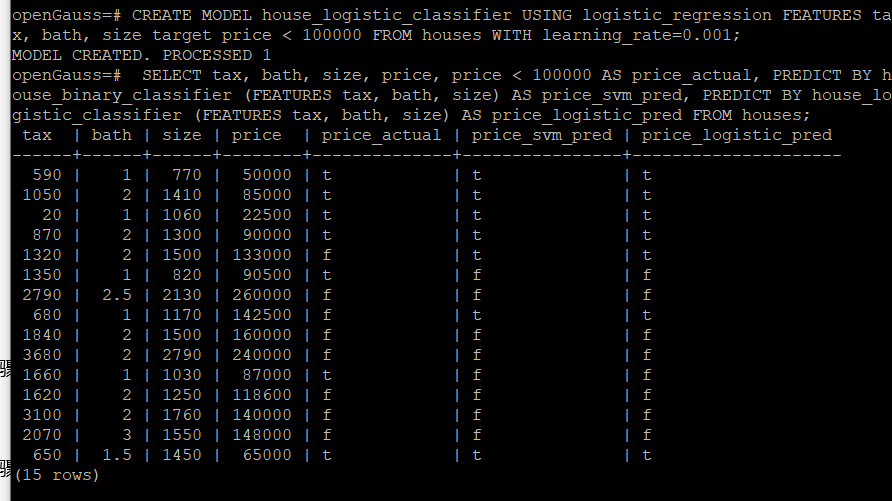
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型和回归模型都是机器学习中的重要模型。

费雷模型可以分为多分类和二分类，分类模型是针对数据对数据的类型进行预测。在给定数据的情况下，预测新数据属于哪个类别。

回归模型则是预测一个数值型目标变量的函数关系。 回归模型被用于建立因变量和自变量之间的关系。

实践思考题2：什么是SVM算法？

SVM是一种对高维数据的降维方法。低维映射高维。将数据集映射到高维空间中，并找到一个最优的超平面，以将数据集分为两个类别。SVM 的目标是找到这个最优的超平面，使其具有最大的间隔，从而可以更好地进行分类。

SVM 会根据已知的数据，找到一条分割线，并且该分割线与离它最近的数据点之间的距离最大。这些最近的数据点被称作 “支持向量”，因此 SVM 的名称也来自于此。

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？

准确率：预测值与真实值的关系，模型预测正确数量与总样本数之比

混淆矩阵：分类正确与错误的对比图。很直观。

查找率和召回率：前者为模型所预测结果为正（阳性）并且实际上也为正（真实阳性）的样本数占预测为正的样本数的比例。模型所预测结果为正（阳性）并且实际上也为正（真实阳性）的样本数占预测为正的样本数的比例。

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（Mean Squared Error）：MSE 是最常见的回归指标之一，它度量观测值与真实值之间差异的平方和的均值。

平均绝对误差（Mean Absolute Error）：与 MSE 类似，MAE 也度量观测值与真实值之间的差异。然而，它是误差的绝对值之和的平均值，因此不像 MSE 那样敏感于异常值。

平均平方根误差（Root Mean Squared Error）：这个指标常用于 MSE 的同比率版本。其优点是使用了相同的单位作为预测变量，更易于比较不同模型之间的性能。