openGauss AI特性创新实践课



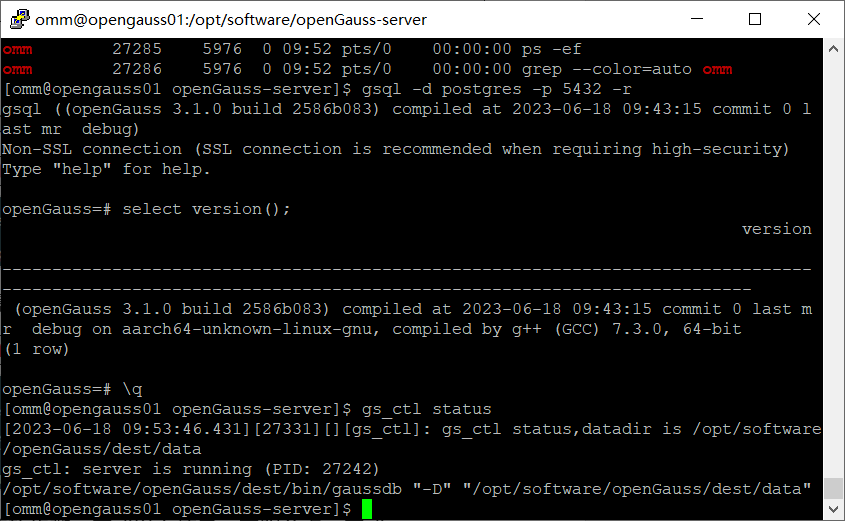
华为技术有限公司

# 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：

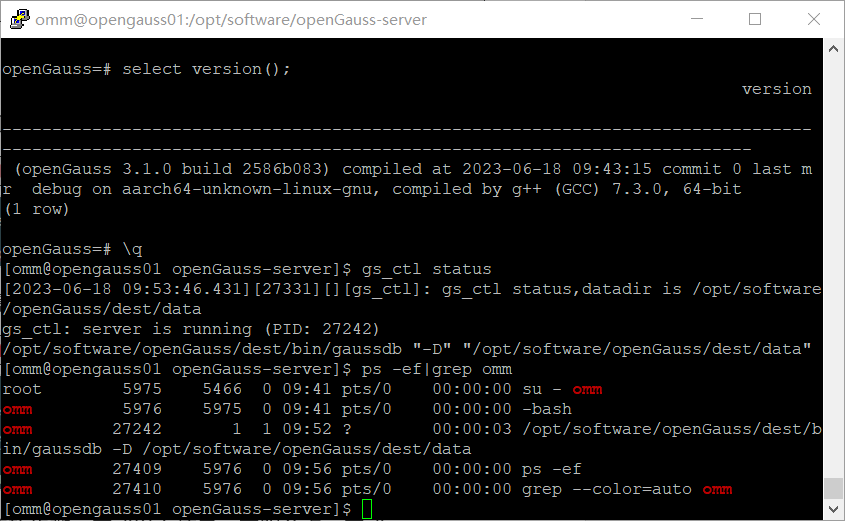
任务一：数据库状态验证

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？

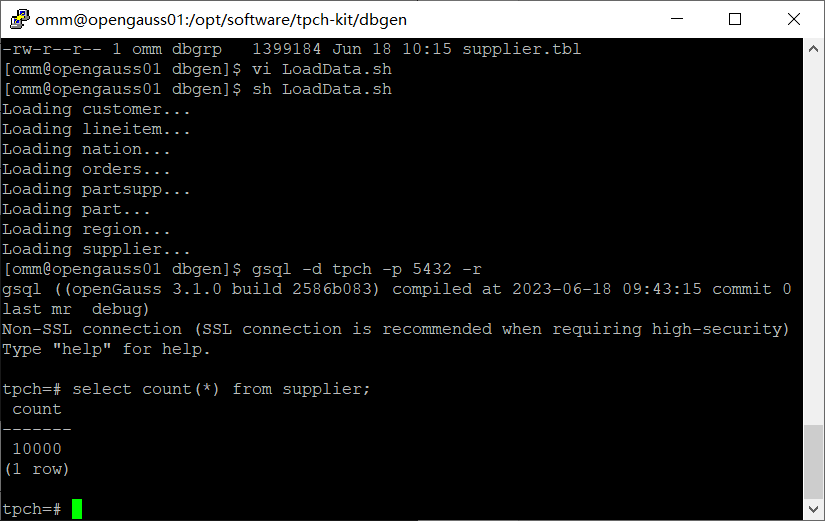
1. 软件在编译期间需要配置。
2. 软件需要统一安装路径
3. 需要安装指定版本

# 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：

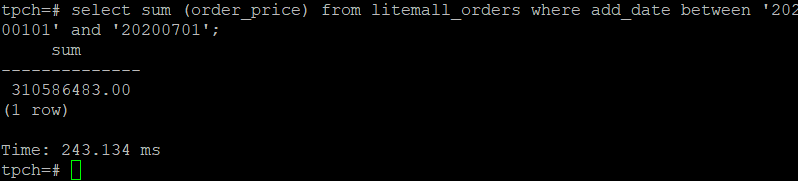
select count(\*) from supplier;;



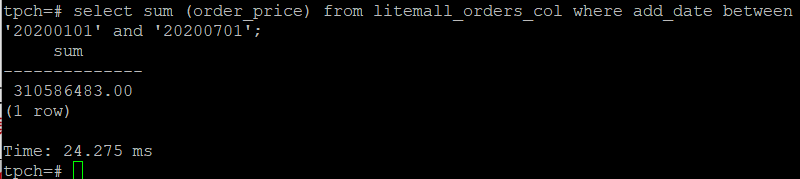
任务二：行存表与列存表执行效率对比

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';

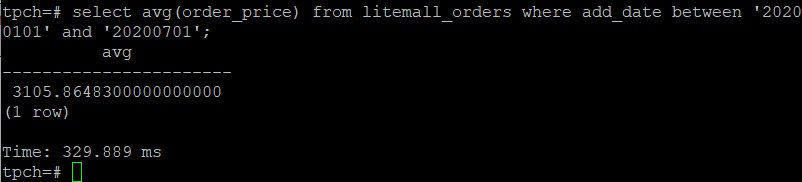


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

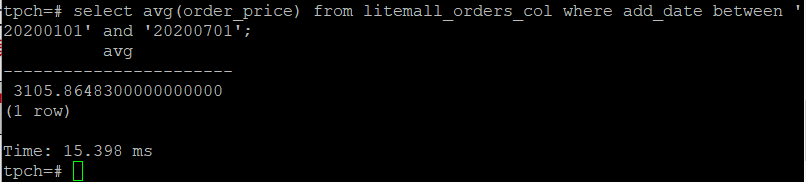


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

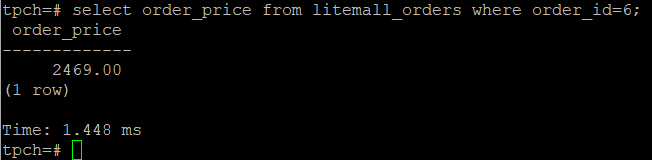


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

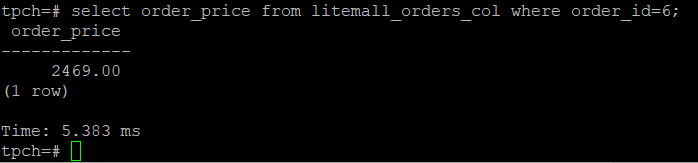


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

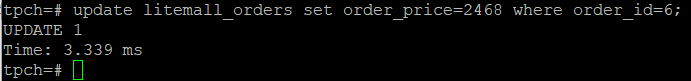


select order\_price from litemall\_orders\_col where order\_id=6;

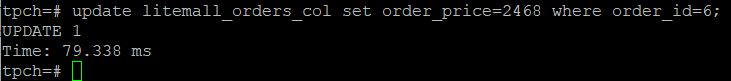


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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

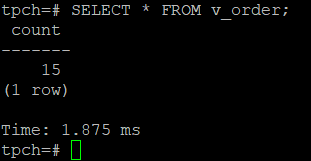
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



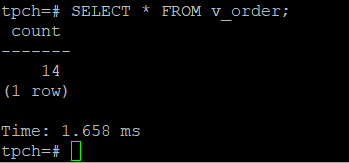
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

行存表以行为单位存储数据，将每一行的数据存储在一起。这种方式适合OLTP（联机事务处理）工作负载，其中频繁地进行插入、更新和删除操作。

适用场景：在执行以下类型的SQL时，行存表通常效率更高：

单条记录查询：例如根据主键查询单个记录。

少量列查询：例如只查询表中的几列数据。

插入、更新和删除操作：由于行存表以行为单位存储数据，对单个记录的操作更加高效。

列存表将每一列的数据存储在一起，而不是按照行存储。这种方式适合OLAP（联机分析处理）工作负载，其中需要对大量数据进行聚合、分析和报表生成。

适用场景：在执行以下类型的SQL时，列存表通常效率更高：

多列聚合查询：例如对表中的多个列进行聚合计算，如SUM、COUNT、AVG等。

大规模数据分析：例如需要对大数据集进行复杂查询和分析操作。

列压缩和向量化处理：由于列存表存储相同类型的数据在一起，可以应用更有效的压缩算法和向量化处理，提高查询性能。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

1、计算方式：

全量物化视图：全量物化视图计算和存储了基础表的完整结果集。当基础表数据发生变化时，全量物化视图需要重新计算和更新，以保持数据的一致性。

增量物化视图：增量物化视图只计算和存储基础表数据的部分结果集，通常是根据指定的增量条件和增量规则来决定哪些数据需要计算和更新。它仅对变化的数据进行更新，可以减少计算和存储的开销。

2、更新频率：

全量物化视图：由于需要重新计算和更新完整的结果集，全量物化视图的更新通常需要在基础表发生变化后手动触发或定期执行。更新频率较低，适合对数据要求不太严格的场景。

增量物化视图：增量物化视图的更新可以更频繁地进行，因为只需要处理变化的数据。它可以在基础表发生变化时自动触发或按照设定的规则进行增量更新，使得视图的数据更加实时和准确。

3、查询性能：

全量物化视图：全量物化视图的查询性能较高，因为它存储了完整的结果集，可以直接返回预计算的数据，减少了查询时的计算开销。

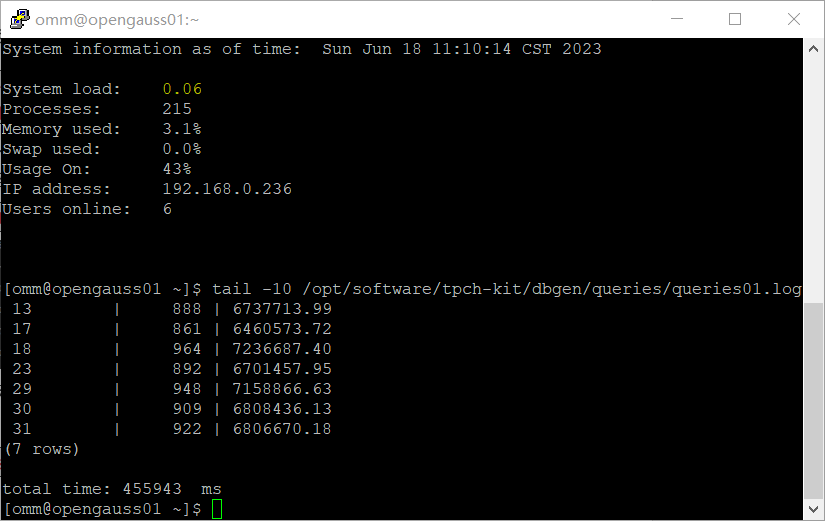
增量物化视图：增量物化视图的查询性能可能较全量物化视图略低，因为它存储的是部分结果集，查询时可能需要进行更多的计算操作。然而，增量物化视图可以提供更实时的数据，并且在处理大型数据集时可能更高效。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

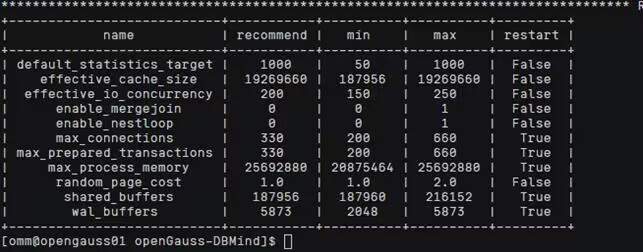
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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

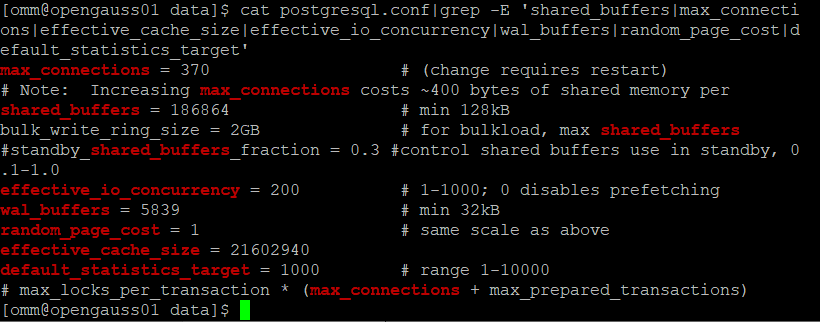
gs\_xtuner recommend --db-name tpch --db-user omm --port 5432 --host 127.0.0.1 --host-user omm



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

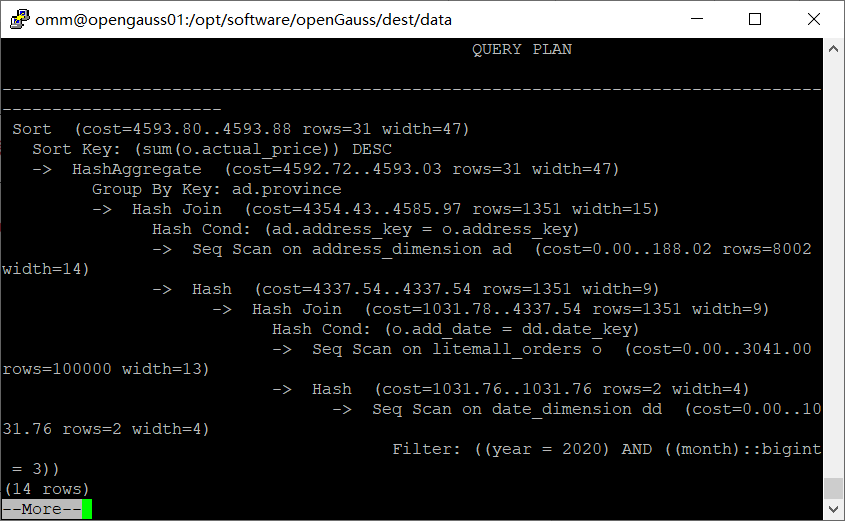
AND o.add\_date = dd.date\_key

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

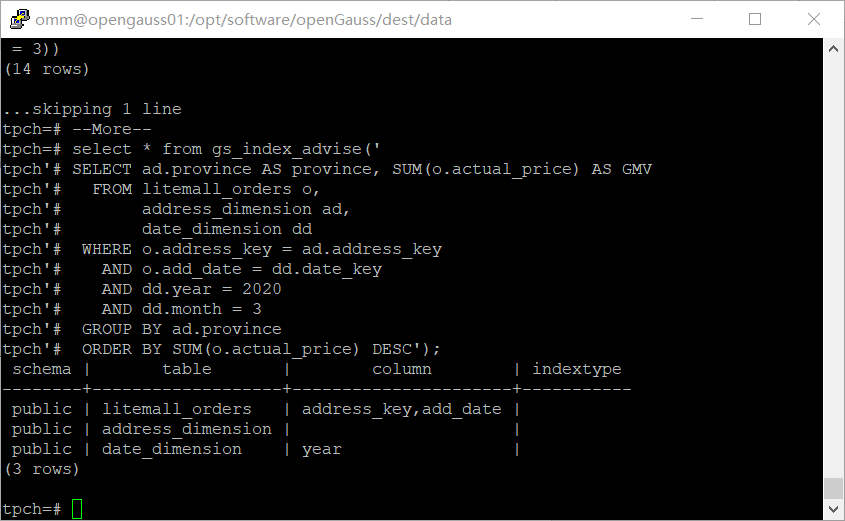
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

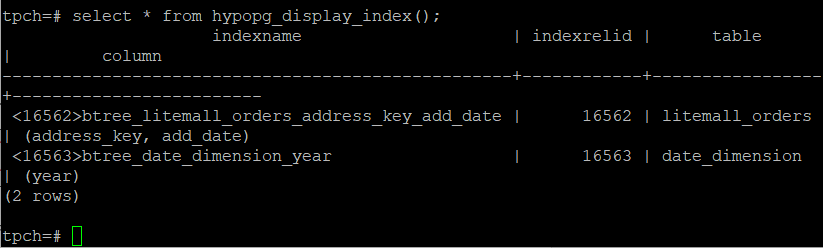
GROUP BY ad.province

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。

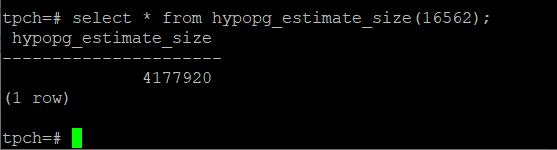
select \* from hypopg\_display\_index();

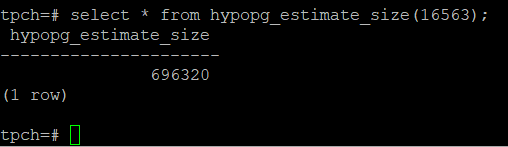


4. 获取索引虚拟列大小结果（单位为：字节），将执行结果截图。

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

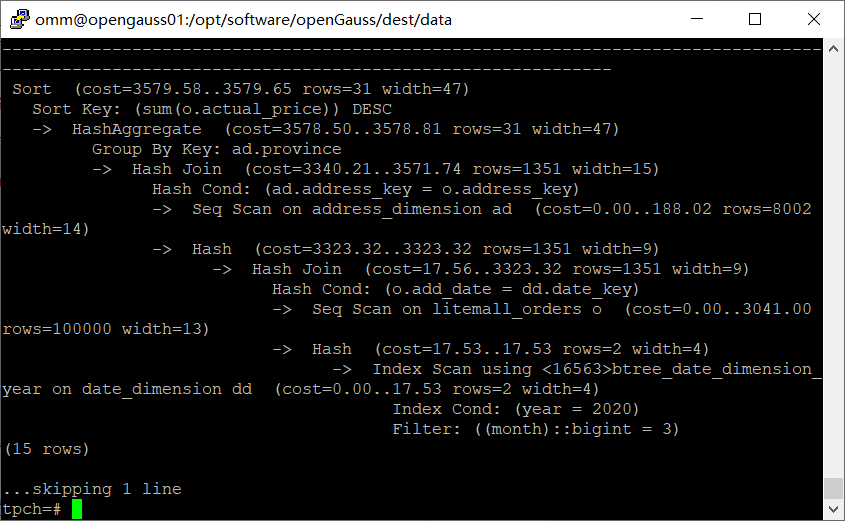
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

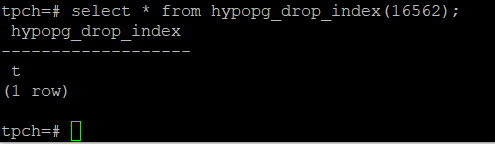
GROUP BY ad.province

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



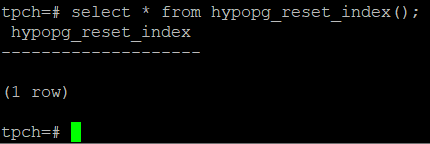
6. 删除某一个索引虚拟列，将执行结果截图。

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



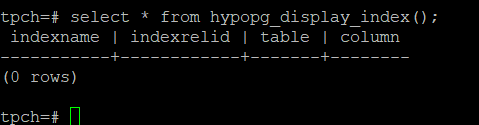
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

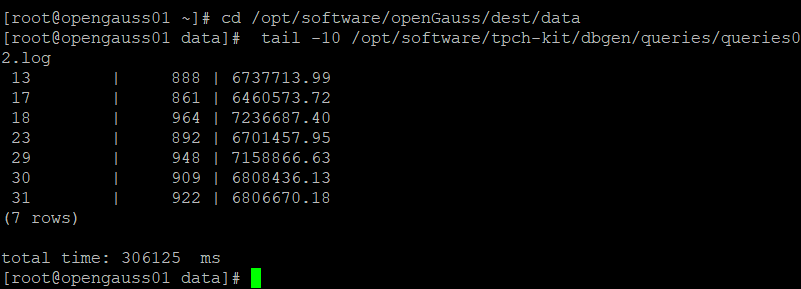
select \* from hypopg\_display\_index();



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：

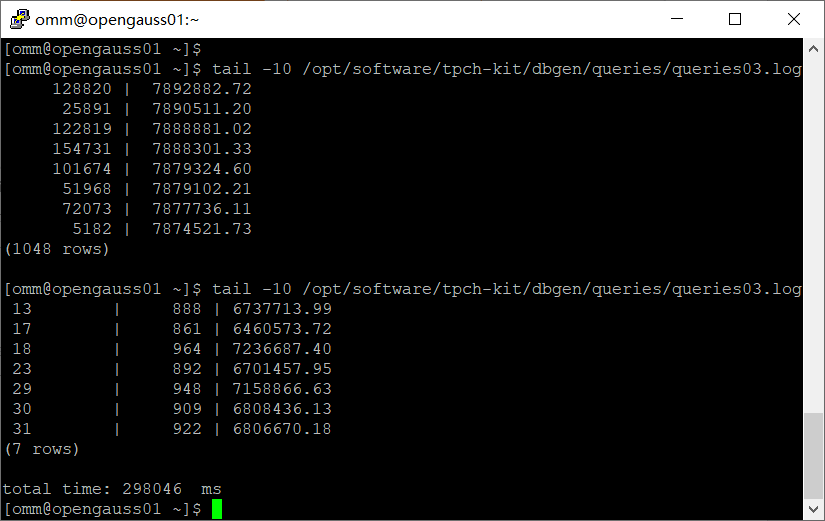
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

default\_statistics\_target：

优化原因：default\_statistics\_target是控制查询优化器使用何种统计信息来生成查询计划的参数。通过增加该参数的值，可以提高查询计划的准确性和性能。

effective\_cache\_size：

优化原因：effective\_cache\_size参数用于指定PostgreSQL可用的预计算磁盘缓存大小。通过适当设置该参数的值，可以提高查询的缓存命中率，从而减少磁盘I/O操作，提升性能。

effective\_io\_concurrency：

优化原因：effective\_io\_concurrency参数用于指定系统对I/O操作的并发度。通过调整该参数的值，可以优化数据库对磁盘I/O的利用率，从而提高查询的响应速度。

enable\_mergejoin：

优化原因：enable\_mergejoin参数用于控制是否启用合并连接（Merge Join）算法。通过启用该算法，可以在某些情况下改善连接操作的性能。

max\_connections：

优化原因：max\_connections参数用于限制数据库服务器同时接受的最大连接数。通过适当设置该参数的值，可以控制数据库服务器的并发连接数量，避免资源过度消耗和性能下降。

max\_prepared\_transactions：

优化原因：max\_prepared\_transactions参数用于限制服务器可以准备的最大事务数。通过适当调整该参数的值，可以控制服务器中同时运行的事务数量，以避免资源竞争和性能问题。

max\_process\_memory：

优化原因：max\_process\_memory参数用于限制数据库服务器进程的最大内存使用量。通过控制进程的内存使用，可以避免系统资源耗尽和性能下降。

random\_page\_cost：

优化原因：random\_page\_cost参数用于指定随机I/O相对于顺序I/O的成本。通过调整该参数的值，可以更准确地反映系统中随机I/O操作的成本，从而优化查询计划生成。

shared\_buffers：

优化原因：shared\_buffers参数用于设置PostgreSQL服务器使用的共享内存缓冲区大小。通过增加该参数的值，可以提高数据库对常用数据的缓存效果，减少磁盘I/O操作，提升查询性能。

wal\_buffers：

优化原因：wal\_buffers参数用于设置WAL（Write-Ahead Log）缓冲区的大小。通过适当设置该参数的值，可以提高写操作的性能，减少WAL写入磁盘的频率。

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、提高查询性能：索引可以加快数据检索速度，通过减少需要扫描的数据量来提高查询效率。索引可以将查询的时间复杂度从全表扫描的O(n)降低到O(log n)或O(1)的级别。

2、减少磁盘I/O操作：索引可以将数据存储在物理上相邻的位置，减少磁盘I/O操作次数。通过减少磁盘的读取操作，可以显著提高查询性能。

3、优化排序和分组操作：索引可以加快排序和分组操作的速度。当SQL语句包含ORDER BY、GROUP BY或聚合函数时，索引可以提供已排序的数据或预计算的结果，减少排序和分组所需的计算量。

除了使用索引和参数，还可以通过以下方面对数据库进行优化：

1、优化数据库设计：良好的数据库设计可以提高查询和事务的性能。包括合理的表结构设计、范式化和反范式化的选择、数据类型的选择等。

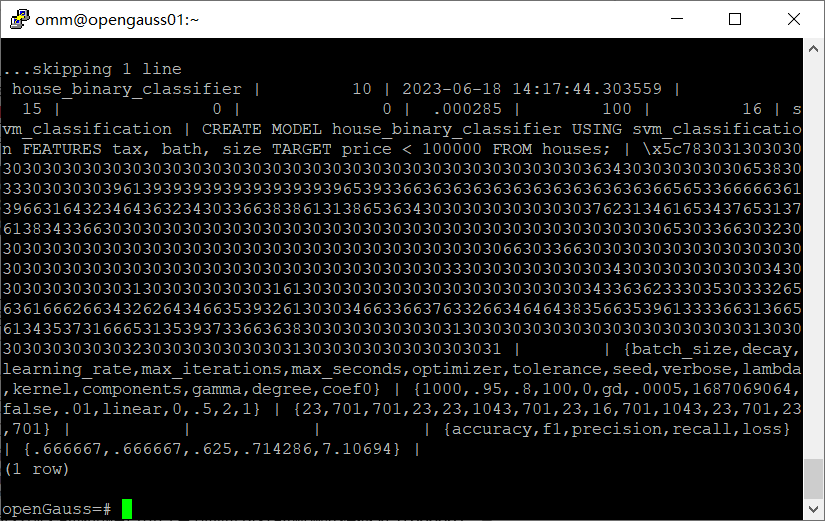
2、合理使用数据库范式：对于OLTP场景，使用适当的范式设计可以减少数据冗余和更新异常，提高事务性能。对于OLAP场景，冗余数据和反范式化设计可以提高查询性能。

3、适当的索引设计：根据查询频率、查询条件和排序需求，设计合理的索引。避免创建过多的索引，以免降低写操作性能和增加存储空间的开销。

# 关卡四、openGauss的DB4AI特性应用

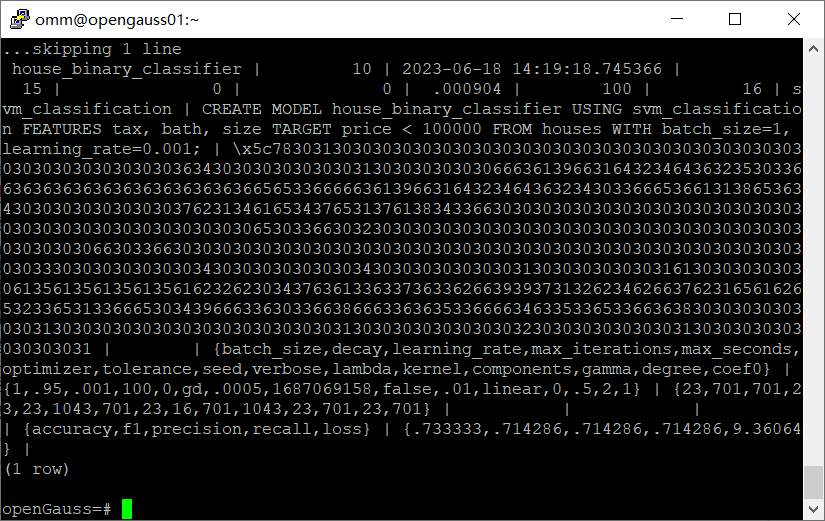
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



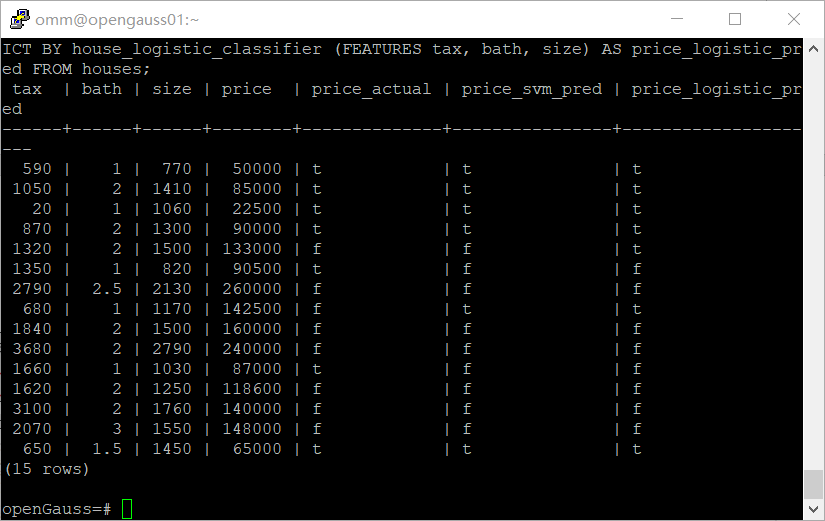
任务二：观察新模型的信息，将执行结果截图。

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？

分类模型用于预测离散的类别或标签。它的输出是一个离散的类别或标签。

回归模型用于预测连续的数值。它的输出是一个连续的数值，可以是实数或整数。

实践思考题2：什么是SVM算法？

SVM（Support Vector Machines）是一种监督学习算法，广泛应用于分类和回归问题。SVM的主要目标是找到一个最优的超平面（或决策边界），将不同类别的样本分隔开来。

在SVM中，样本被表示为特征空间中的点，每个点被赋予一个特定的类别标签。SVM的目标是找到一个能够将不同类别的样本分隔开的超平面，使得不同类别的样本点尽可能远离该超平面。这样的超平面被称为最大间隔超平面。

SVM的基本原理是通过对训练数据进行非线性映射，将其映射到高维特征空间，使得样本在该空间中更容易线性分割。在特征空间中，SVM寻找一个最优的超平面，使得离该超平面最近的样本点（称为支持向量）到该超平面的距离最大化。

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（Accuracy）：准确率是最常见的分类评价指标，表示分类器正确预测的样本数与总样本数之间的比例。

精确率（Precision）和召回率（Recall）：精确率和召回率是一对相关的指标，常用于评估二分类问题。精确率衡量预测为正例的样本中真正为正例的比例，召回率衡量实际为正例的样本中被正确预测为正例的比例。

F1值（F1 Score）：F1值综合考虑了精确率和召回率，是精确率和召回率的调和平均值，用于平衡分类器的性能。

特异度（Specificity）：特异度衡量了分类器对负例样本的预测能力，即预测为负例的样本中真正为负例的比例。

AUC-ROC（Area Under the Receiver Operating Characteristic Curve）：AUC-ROC（Area Under the Receiver Operating Characteristic Curve）：

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）：均方误差衡量了预测值与真实值之间的平均差异程度，即预测值与真实值之间差异的平方的均值。

均方根误差（Root Mean Squared Error，RMSE）：均方根误差是均方误差的平方根，衡量了预测值与真实值之间的平均差异程度，具有与真实值相同的单位。

平均绝对误差（Mean Absolute Error，MAE）：平均绝对误差衡量了预测值与真实值之间的平均差异程度，即预测值与真实值之间差异的绝对值的均值。

决定系数（Coefficient of Determination，R²）：决定系数衡量了回归模型对因变量变异性的解释程度，即模型解释的方差比例。