openGauss AI特性创新实践课



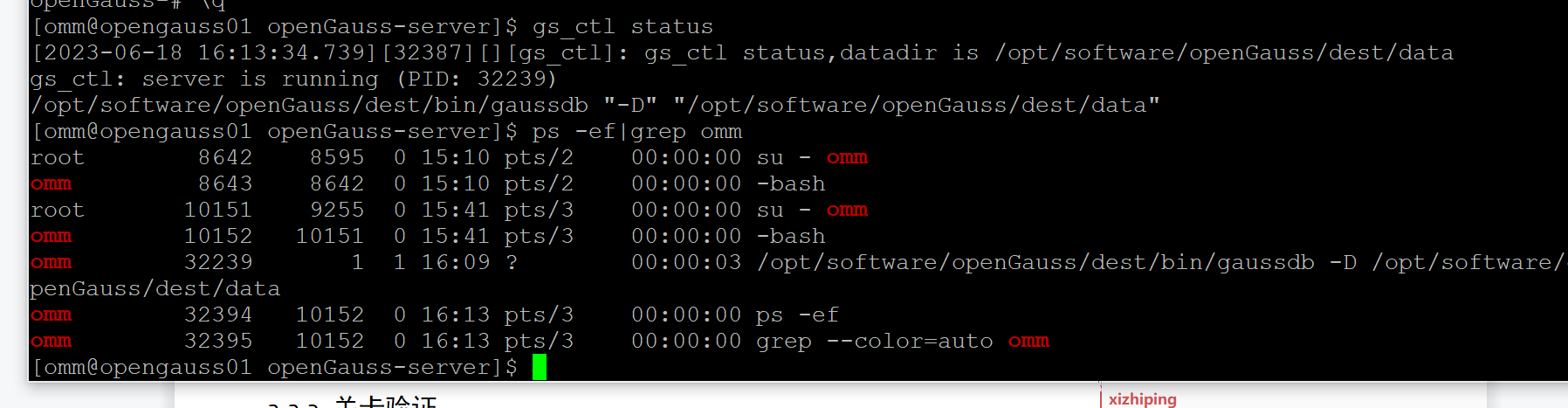
华为技术有限公司

# 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：

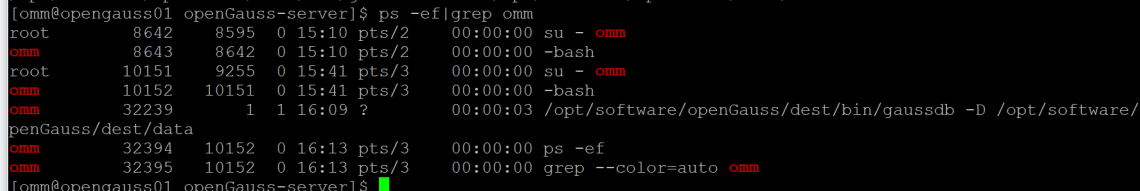
任务一：数据库状态验证

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？

定制化：通过源码编译，可以自定义编译选项以满足特定需求。可以启用或禁用某些功能，进行性能优化，或者修改默认设置，以适应的应用场景。

平台适配性：通过源码编译，可以确保数据库系统能够在特定的操作系统和硬件平台上运行。这是因为源码编译会根据目标平台的特性进行优化，以获得更好的性能和兼容性。

最新功能和修复：通过编译源码，可以获取到数据库的最新版本，并获得最新的功能增强和错误修复。这对于追求最新功能、性能改进或安全修复的用户来说非常有吸引力。

安全性和可信度：源码编译允许审查和验证数据库系统的源代码。这样可以确保代码的完整性和安全性，减少潜在的漏洞或后门存在的风险。

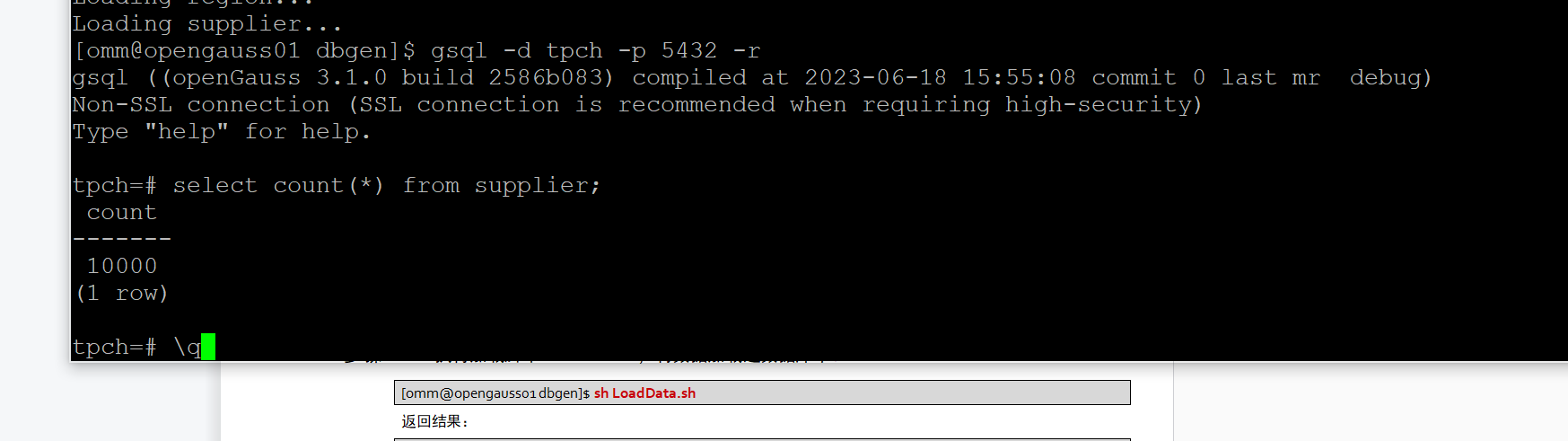
扩展性和自定义：通过源码编译安装数据库，可以更方便地编写和安装自定义的插件、扩展或模块。这样可以根据特定需求进行定制开发，以满足应用程序的特殊要求。

# 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';

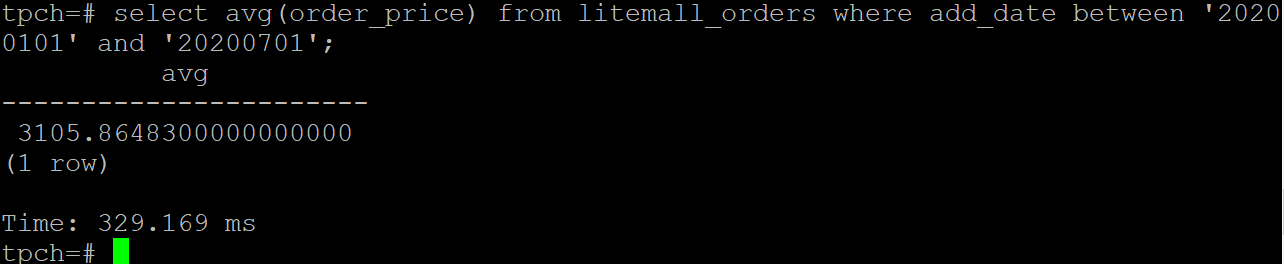


select sum (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

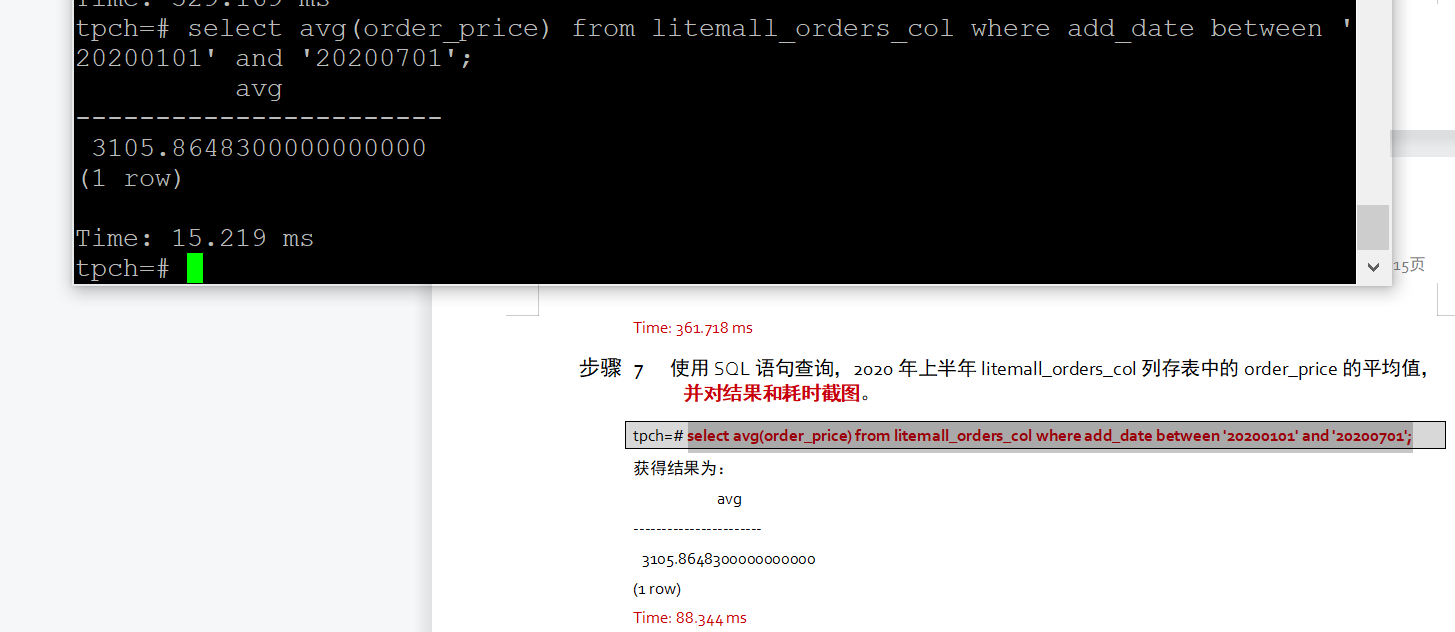


2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

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

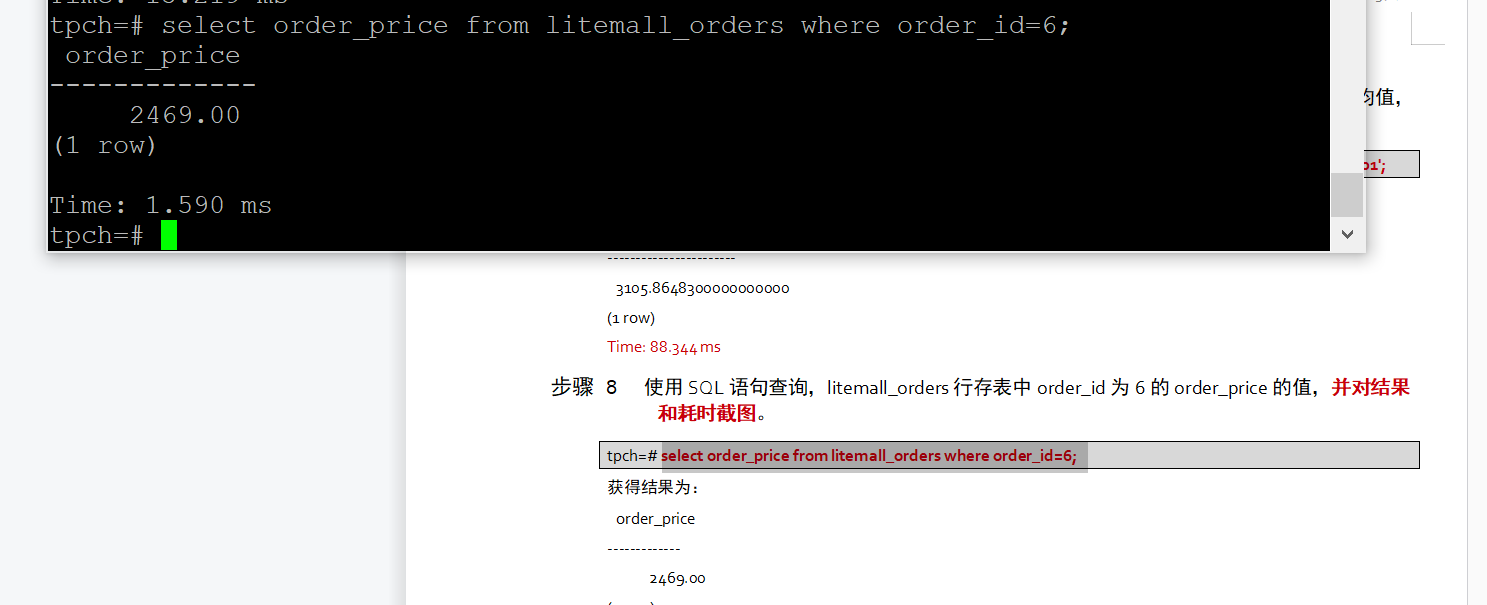


select avg (order\_price) from litemall\_orders\_col where add\_date between '20200101' and '20200701';

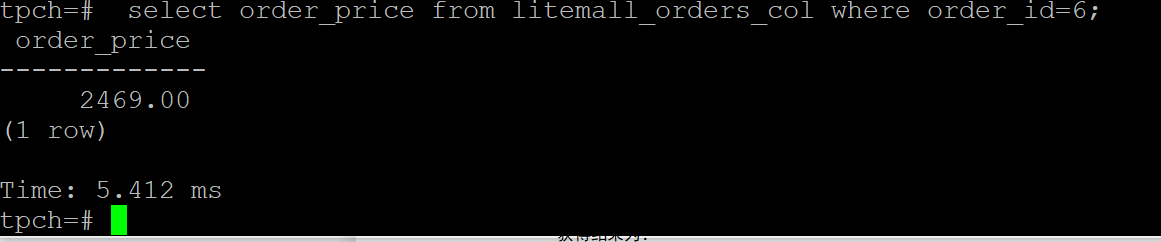


3. 查询litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price的值，并对比执行效率截图。

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

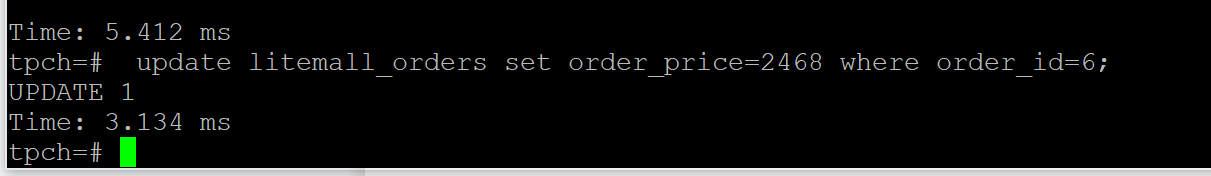


select order\_price from litemall\_orders\_col where order\_id=6;

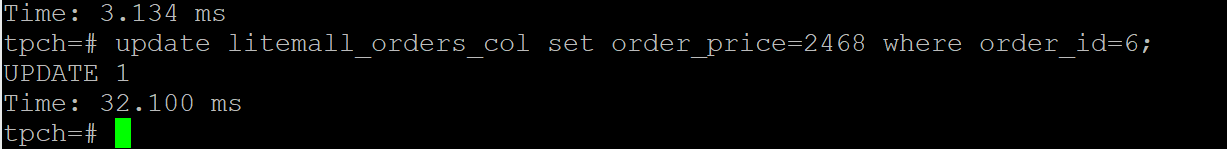


4. 将litemall\_orders行存表与litemall\_orders\_col列存表中order\_id为6的order\_price修改为2468，并对比执行效率截图。

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



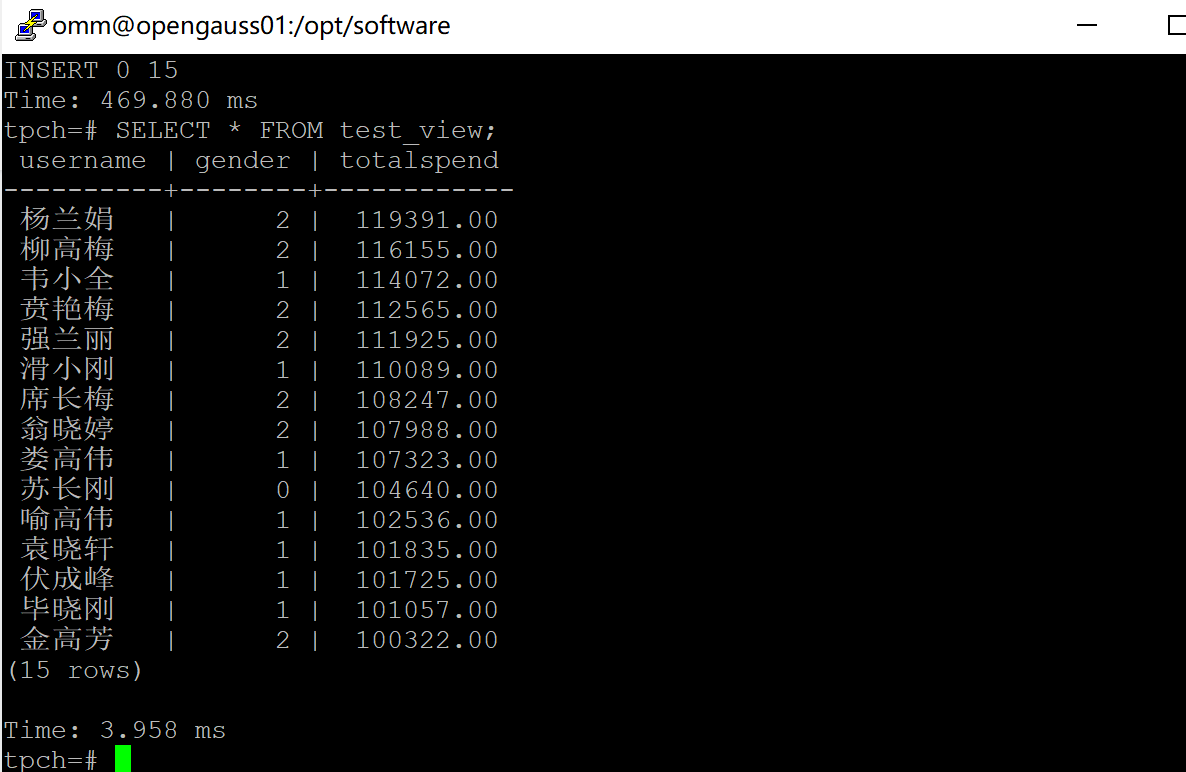
update litemall\_orders\_col set order\_price=2468 where order\_id=6;



任务三：物化视图的使用

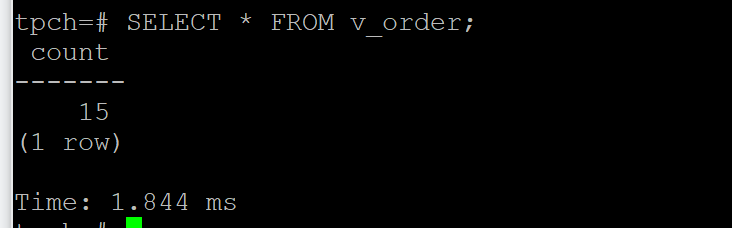
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



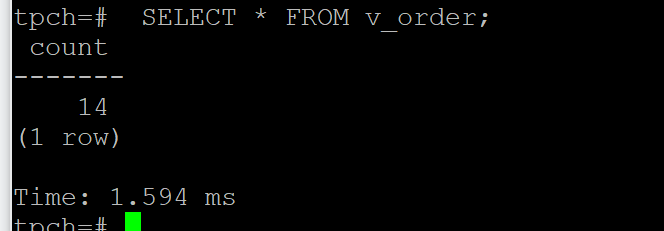
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

SELECT \* FROM v\_order;



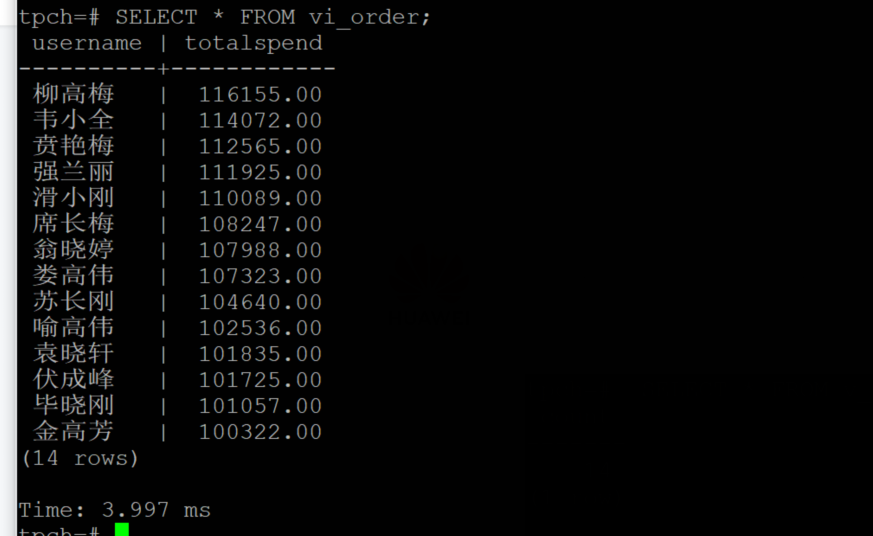
3. 对表进行操作后，刷新物化视图，查询物化视图结果，将执行结果截图。

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，将同一行的所有列存储在一起。这种方式适合事务处理（OLTP）场景，其中频繁地进行插入、更新和删除操作。在执行以下类型的SQL时，行存表效率通常更高：

单条记录的读写操作：当需要快速读取或修改单个记录时，行存表的存储方式更加高效，因为可以直接访问一整行的数据。

需要频繁更新的操作：行存表适合进行频繁的更新操作，因为它可以更快地定位和修改特定行的数据。

列存表将数据按列存储，即将同一列的数据放在一起。这种方式适合分析型查询（OLAP）场景，其中需要大量的聚合计算和扫描大量数据的操作。

在执行以下类型的SQL时，列存表效率通常更高：

聚合查询和分析操作：当需要进行聚合函数、统计和分析操作时，列存表能够更快地读取和处理大量的列数据，因为它只需要访问所需的列，而不需要读取整行的数据。

大数据量的扫描操作：当需要扫描大量数据的查询时，列存表能够更高效地处理，因为它只需要读取所需的列，而不必加载整行数据。

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：

刷新机制：全量物化视图在刷新时会重新计算和重新获取所有的数据，并将完整的结果存储在物化视图中。这意味着在每次刷新时，全量物化视图都会从源表中重新读取数据并计算新的结果。

数据内容更新：全量物化视图的数据内容是完整的、全量的，并且与源表中的数据一致。每次刷新后，物化视图中的数据会完全替换为最新的数据。

增量物化视图：

刷新机制：增量物化视图在刷新时只计算和获取源表中发生更改的数据，并将这些更改应用到物化视图中。这意味着增量物化视图只处理源表中发生变化的数据，而不需要重新计算和获取所有的数据。

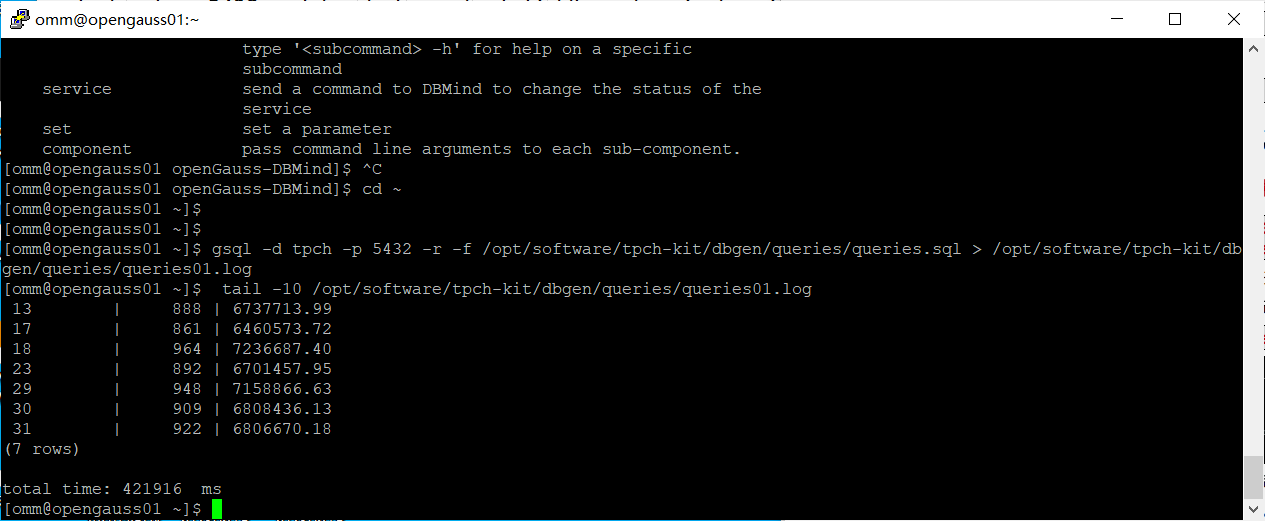
数据内容更新：增量物化视图只包含源表中发生更改的数据，而不是完整的数据集。它只记录和维护那些发生了变化的数据，并将这些变化应用到物化视图中，以保持数据的最新状态。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

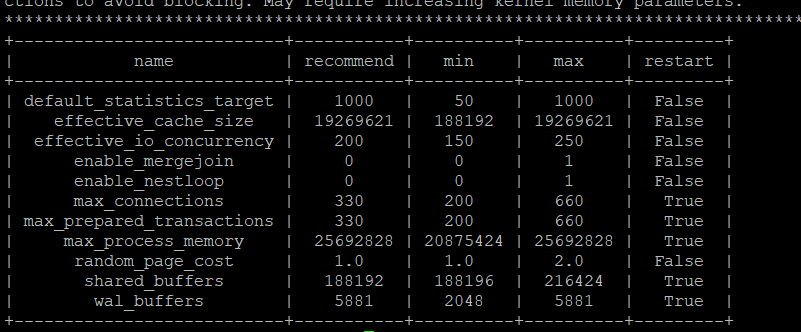
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

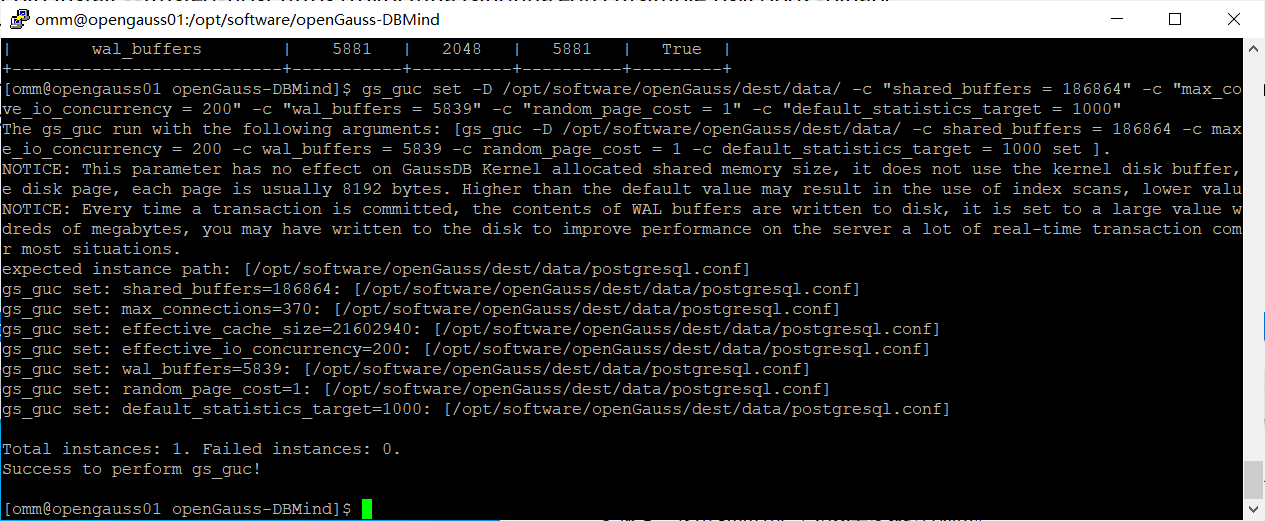
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries01.log



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

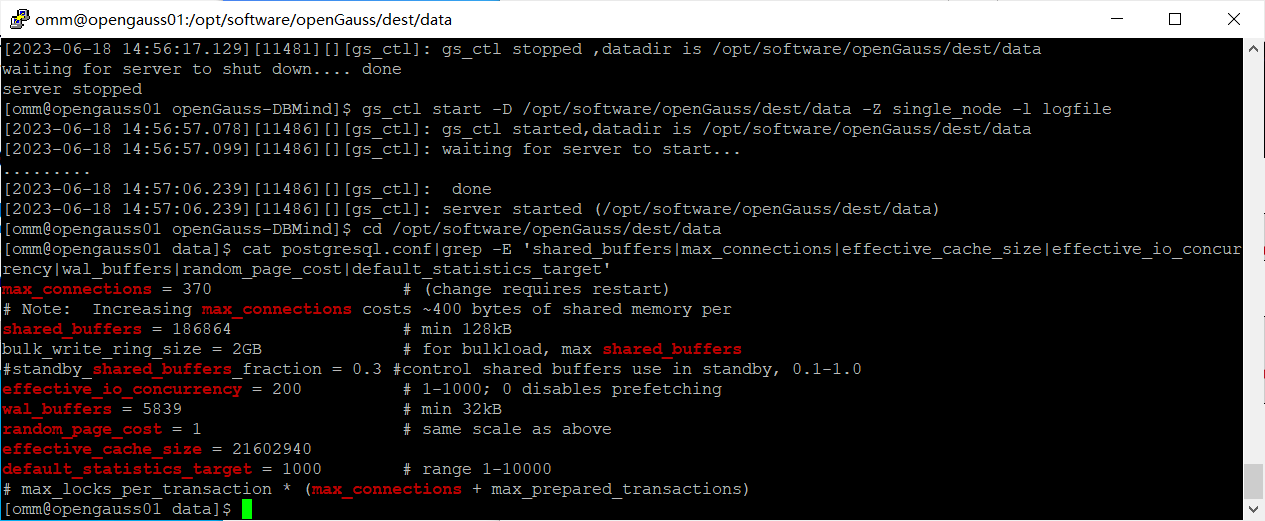




3.重启完成后，获取参数值：

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

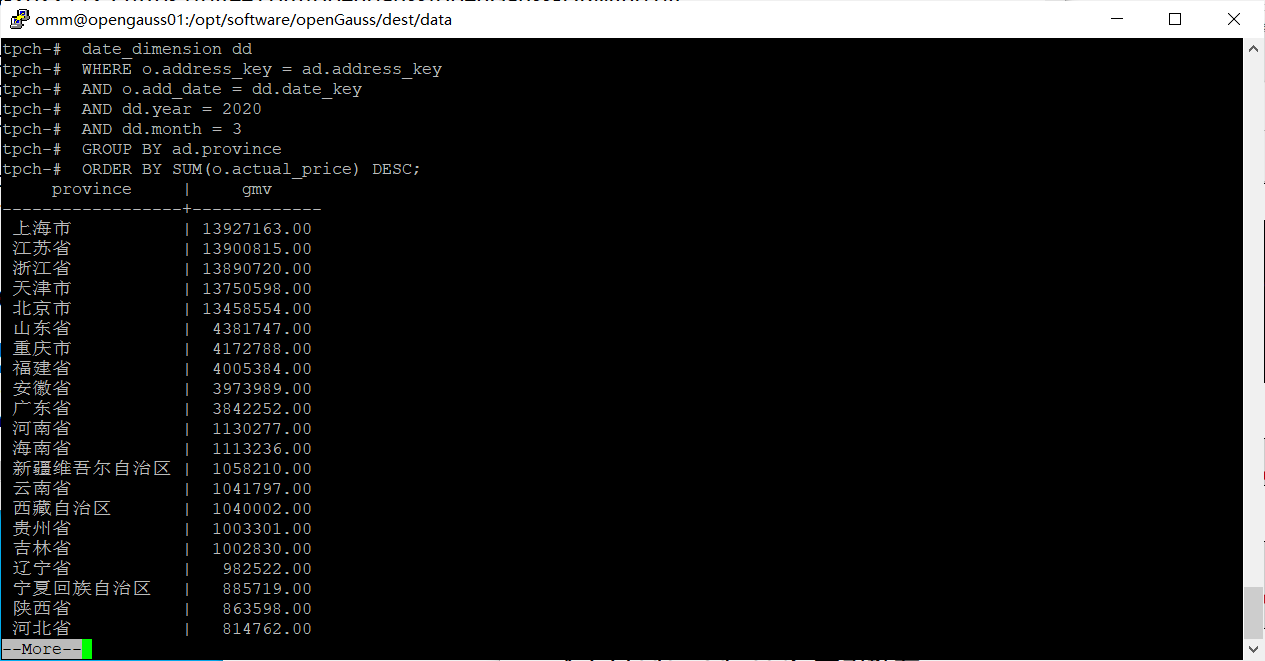
AND o.add\_date = dd.date\_key

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

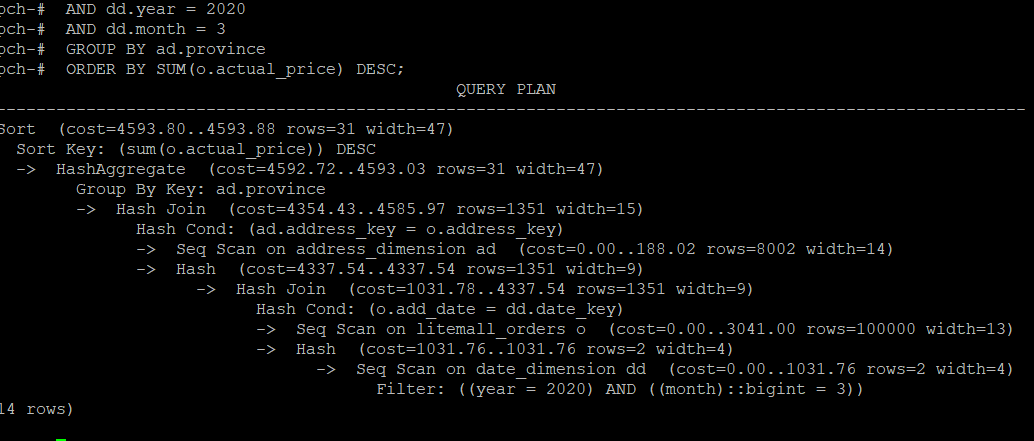
AND o.add\_date = dd.date\_key

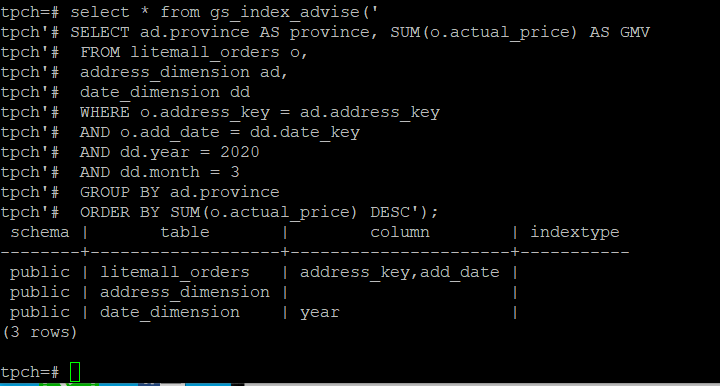
AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

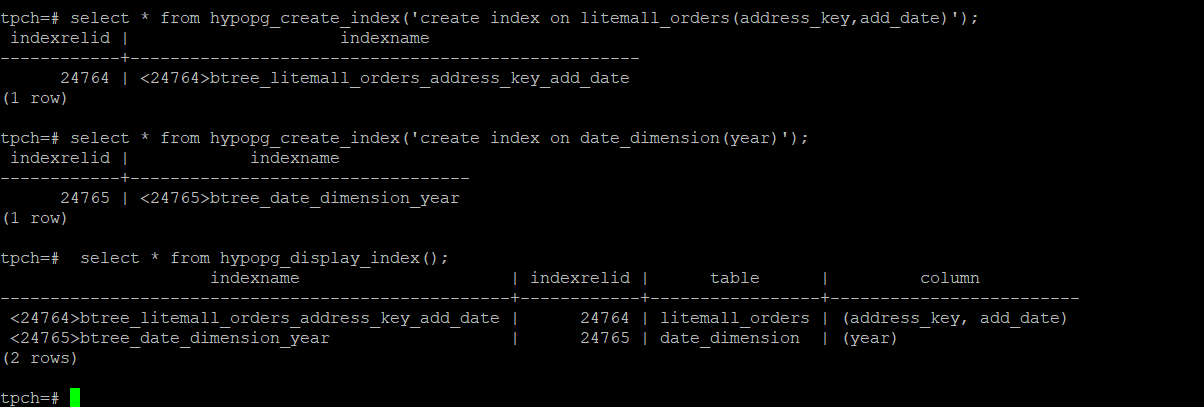
ORDER BY SUM(o.actual\_price) DESC');





3. 查看创建的虚拟索引列，将执行结果截图。

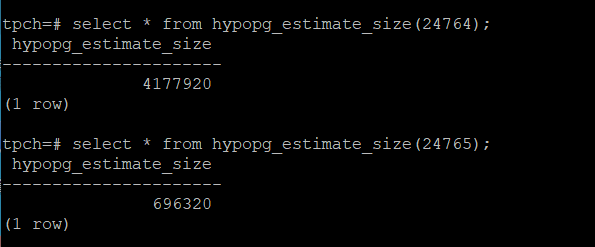
select \* from hypopg\_display\_index();



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

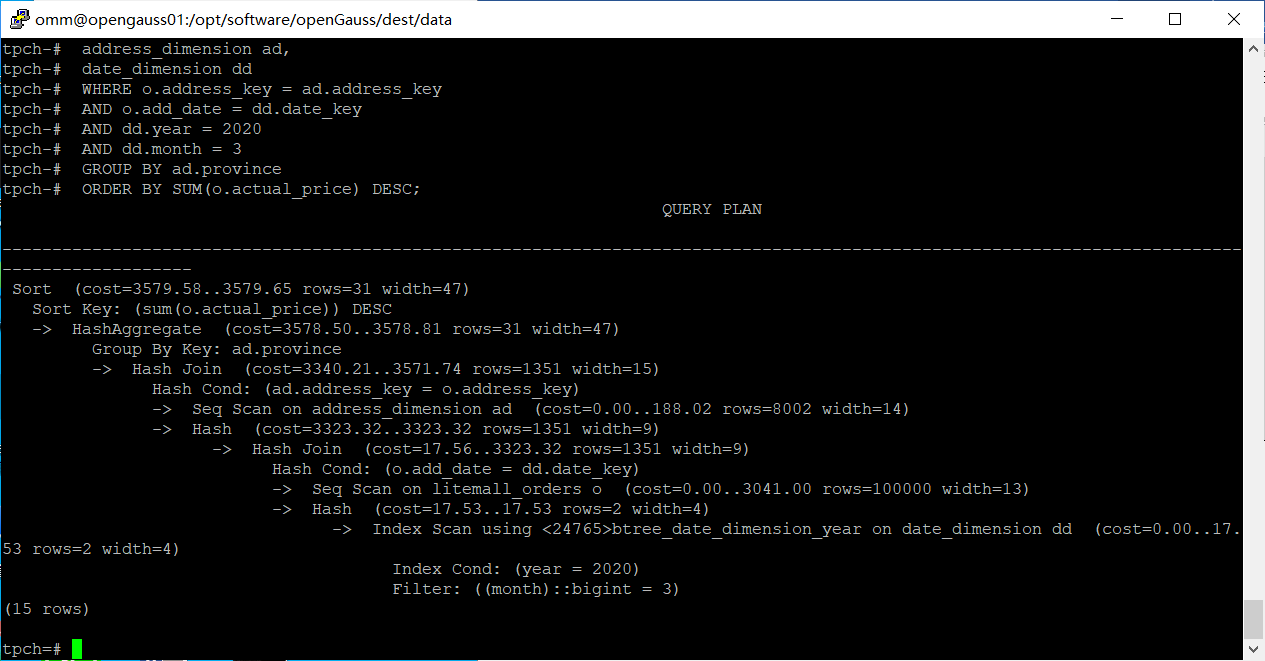
AND o.add\_date = dd.date\_key

AND dd.year = 2020

AND dd.month = 3

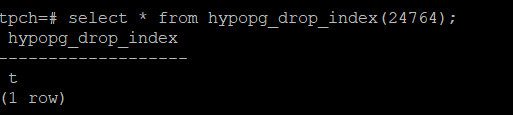
GROUP BY ad.province

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



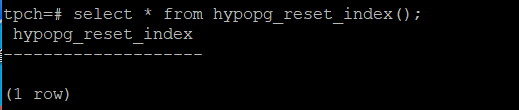
6. 删除某一个索引虚拟列，将执行结果截图。

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



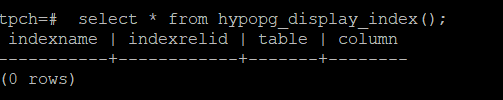
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

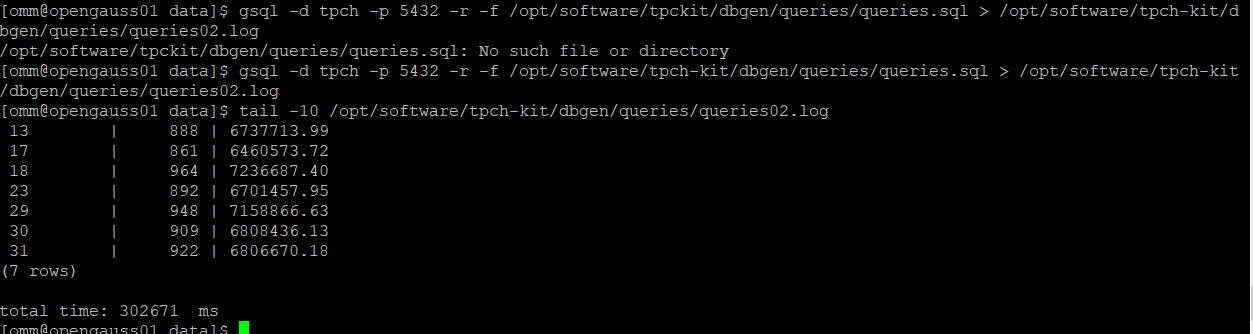
select \* from hypopg\_display\_index();



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：

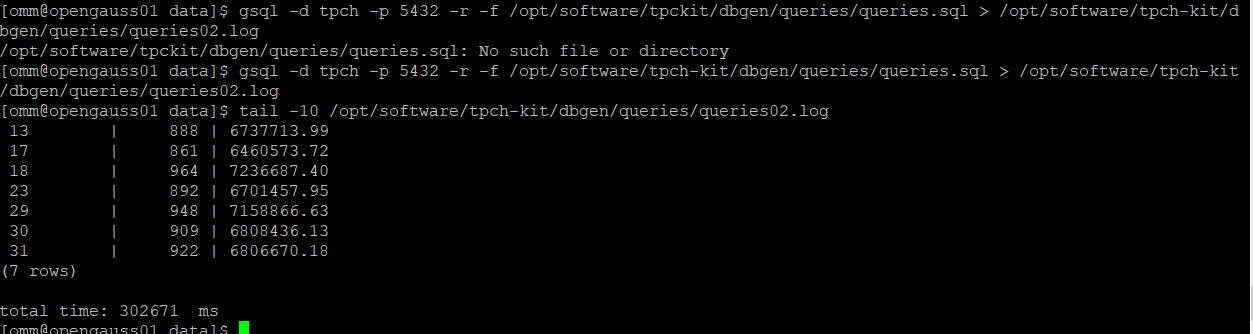
gsql -d tpch -p 5432 -r -f /opt/software/tpch-kit/dbgen/queries/queries.sql > /opt/software/tpch-kit/dbgen/queries/queries02.log



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？

缓冲区大小（Buffer size）：通过调整缓冲区大小，可以影响数据库系统在内存管理和磁盘操作方面的性能。优化此参数可以提高系统的吞吐量和响应时间，减少磁盘I/O操作，从而改善系统性能。

并发连接数（Concurrent connections）：调整并发连接数可以控制数据库系统同时处理的客户端连接数量。通过优化此参数，可以平衡系统的并发性能和资源利用率，避免过多的连接导致性能下降或资源耗尽。

查询优化器相关参数（Query optimizer parameters）：数据库查询优化器负责生成最优的查询计划。通过调整查询优化器相关参数，可以影响查询计划的生成方式和执行策略，从而改善查询性能和执行效率。

内存参数（Memory parameters）：数据库系统中的内存管理对性能至关重要。通过优化内存参数，如共享内存大小、排序缓冲区大小等，可以提高数据库的内存使用效率和查询性能。

日志相关参数（Log-related parameters）：数据库系统的事务日志对数据的持久性和恢复能力起着重要作用。优化日志相关参数可以改善事务处理的性能和可靠性。

这些只是一些可能进行优化的参数示例，实际的参数优化范围取决于具体的应用程序和系统环境。通过优化这些参数，可以根据实际需求和系统特点，提高系统的性能、可靠性和资源利用效率。 X-Tuner利用自动化技术，可以根据实时监测和反馈的性能指标，通过尝试不同的参数组合来找到最佳的参数配置，以实现性能优化的目标。

窗体顶端

Regenerate response

窗体底端

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？

索引在执行SQL查询时具有以下好处：

提高查询性能：索引可以加快SQL查询的速度，因为它们提供了快速的数据查找路径。通过使用索引，数据库可以更快地定位和检索符合查询条件的数据，减少全表扫描的开销。

减少磁盘I/O操作：索引可以减少数据库系统需要从磁盘读取的数据量。它们创建了数据的副本或引用，使得数据库可以更有效地利用内存中的数据，减少磁盘I/O操作的频率，提高查询的响应速度。

改善数据一致性和完整性：索引可以用作唯一约束或主键约束的支持，确保数据的一致性和完整性。通过创建索引并强制约束，数据库可以防止重复数据或无效数据的插入。

除了使用索引和参数优化外，还有其他一些方面可以对数据库进行优化：

数据库设计：良好的数据库设计是优化的关键。合理的表结构、关系建立和数据模型可以提高查询性能和数据操作效率。

正规化和反规范化：根据具体需求，对数据库进行正规化（降低冗余和重复数据）或反规范化（提高查询性能）可以优化数据库的效率。

查询优化：通过优化SQL查询语句、选择合适的连接方式、使用适当的查询条件和运算符，可以提高查询性能。还可以通过使用查询提示（query hints）或查询重写（query rewriting）来改善查询计划。

硬件和存储优化：合理配置硬件资源、磁盘子系统和存储设备，例如使用RAID技术、调整磁盘缓存大小等，可以提高数据库的读写性能和数据存储效率。

并发控制和事务管理：优化并发控制策略、锁定机制和事务管理可以提高多用户并发访问时的性能和数据一致性。

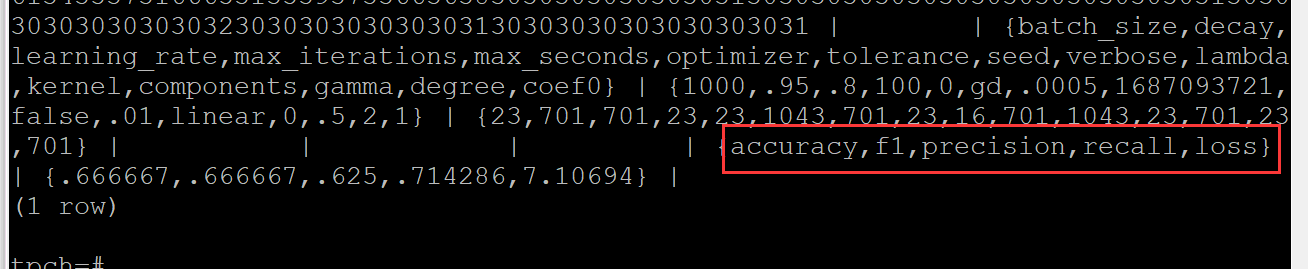
定期维护和性能监控：定期执行数据库维护任务，如索引重建、统计信息收集和日志清理等，以保持数据库的健康状态。同时，监控数据库的性能指标和查询执行计划，及时识别和解决性能瓶颈。

这些是对数据库进行优化的一些常见方面。具体的优化策略和技术取决于数据库系统的特性、应用程序的需求和环境的限制。

# 关卡四、openGauss的DB4AI特性应用

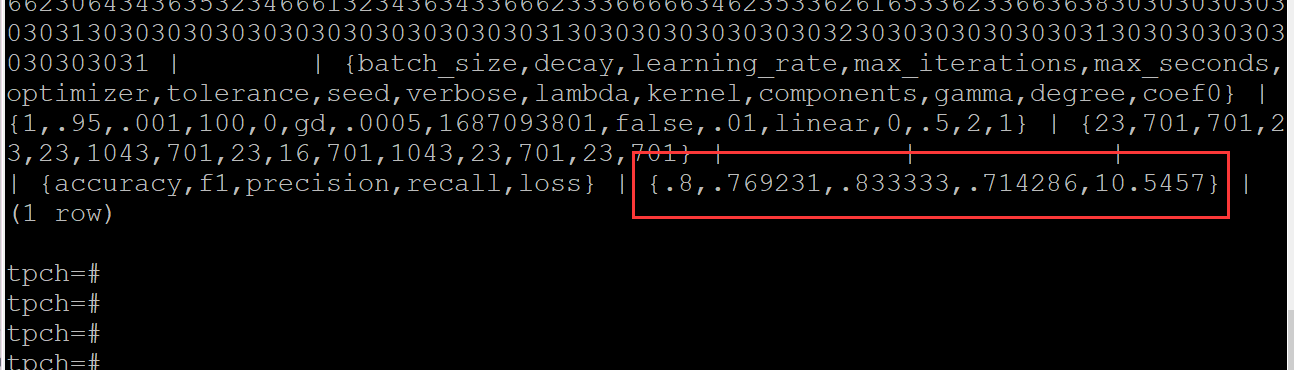
任务一：在gs\_model\_warehouse系统表中查看训练后的模型信息，将执行结果截图：

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



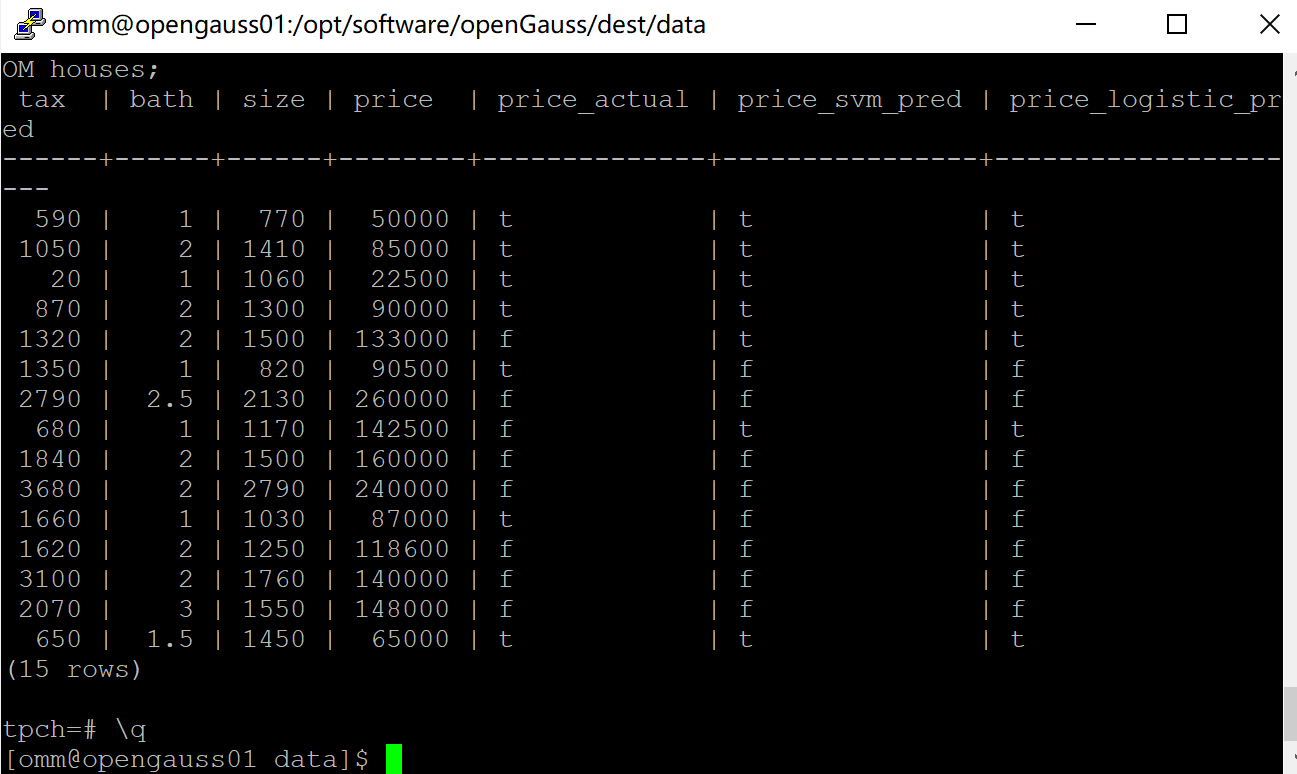
任务二：观察新模型的信息，将执行结果截图。

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（Classification Model）：

目标变量：分类模型用于预测离散的目标变量或标签，它将输入数据映射到不同的类别或类标签。类别可以是二元的（例如是/否、真/假），也可以是多元的（例如猫、狗、鸟等）。

预测任务：分类模型的主要任务是将新的输入样本分配到已知的类别中。它通过学习从已有的标记样本中提取特征，并根据这些特征对未标记样本进行分类。

回归模型（Regression Model）：

目标变量：回归模型用于预测连续的目标变量，它根据输入的特征变量预测一个数值。目标变量可以是实数，如房价、销售额等。

预测任务：回归模型的主要任务是根据输入特征的值预测目标变量的数值。它通过学习从已有的训练数据中捕捉特征与目标变量之间的关系，并用于预测新的未知样本

实践思考题2：什么是SVM算法？

SVM（支持向量机，Support Vector Machine）是一种常用的机器学习算法，用于进行分类和回归分析。它是一种有监督学习算法，基于统计学习理论和结构风险最小化原则。

SVM的主要思想是找到一个最优的超平面，将不同类别的数据点在特征空间中划分开来。在二分类问题中，SVM通过构建一个最大间隔的超平面，使得两个不同类别的数据点距离超平面最远。这个最大间隔的超平面被称为决策边界，而离决策边界最近的数据点被称为支持向量。支持向量是决定决策边界位置和形状的关键点。

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之一，表示分类器正确预测的样本数占总样本数的比例。它可以衡量分类器整体的预测准确程度。准确率计算公式为：准确率 = (预测正确的样本数) / (总样本数)。

精确率（Precision）：精确率衡量分类器预测为正例的样本中真正为正例的比例。它关注的是分类器预测为正例的准确性。精确率计算公式为：精确率 = (真正例的样本数) / (预测为正例的样本数)。

召回率（Recall）：召回率衡量实际为正例的样本中被分类器正确预测为正例的比例。它关注的是分类器对正例样本的覆盖程度。召回率计算公式为：召回率 = (真正例的样本数) / (实际为正例的样本数)。

F1分数（F1 Score）：F1分数是精确率和召回率的调和平均值，综合考虑了分类器的准确性和覆盖度。它是一个综合评价指标，用于平衡精确率和召回率之间的权衡。F1分数计算公式为：F1分数 = 2 \* (精确率 \* 召回率) / (精确率 + 召回率)。

特异度（Specificity）：特异度衡量分类器在预测为负例时的准确性，即预测为负例且实际为负例的样本比例。它是衡量分类器对负例样本的识别能力。特异度计算公式为：特异度 = (真负例的样本数) / (实际为负例的样本数)。

ROC曲线和AUC值：ROC曲线（Receiver Operating Characteristic curve）是以真正例率（True Positive Rate，召回率）为纵轴，假正例率（False Positive Rate）为横轴绘制的曲线。ROC曲线可以展示分类器在不同阈值下的分类性能。AUC（Area Under the Curve）是ROC曲线下的面积，用于衡量分类器的整体性能。AUC值越大，表示分类器的性能越好。

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）：均方误差是回归问题中最常用的评价指标之一，它衡量预测值与真实值之间的平均差异程度。计算MSE时，首先计算每个样本的预测值与真实值之差的平方，然后求平均值。均方误差越小，表示预测结果与真实值之间的差异越小。

平均绝对误差（Mean Absolute Error，MAE）：平均绝对误差是另一个常用的回归评价指标，它计算预测值与真实值之间的绝对差的平均值。平均绝对误差衡量了预测结果与真实值之间的平均偏差程度。平均绝对误差越小，表示预测结果与真实值之间的偏差越小。

决定系数（Coefficient of Determination，R^2）：决定系数是衡量回归模型拟合优度的指标，表示回归模型能够解释因变量方差的比例。决定系数的取值范围在0到1之间，值越接近1表示模型对观测数据的拟合程度越好，值越接近0表示模型对观测数据的拟合程度越差。

解释方差分数（Explained Variance Score）：解释方差分数衡量了回归模型能够解释因变量方差的比例，它表示预测值与真实值之间的方差比例。解释方差分数的取值范围在0到1之间，值越接近1表示模型对观测数据的解释能力越强。