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

文本

描述已自动生成

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）

文本

描述已自动生成

实验思考题：为什么需要通过源码编译，安装数据库？

通过源码编译和安装数据库，可以让用户更好地理解和掌握数据库的运行原理和细节，也可以根据自己的需求进行定制化配置和优化。此外，源码编译和安装也可以提高系统的安全性和稳定性

# 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;;

文本

描述已自动生成

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';

文本

描述已自动生成

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

文本

描述已自动生成

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';

文本

描述已自动生成

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

文本

描述已自动生成

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;

文本

描述已自动生成

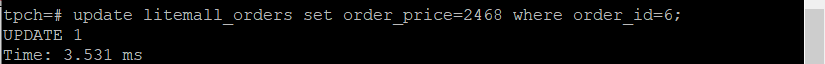
select order\_price from litemall\_orders\_col where order\_id=6;

文本

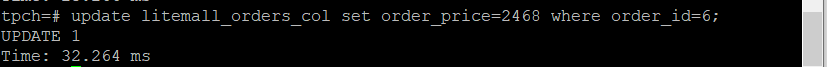
描述已自动生成

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;

图片包含 文本

描述已自动生成

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;

形状

中度可信度描述已自动生成

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;

形状

中度可信度描述已自动生成

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;

图片包含 文本

描述已自动生成

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;

文本

中度可信度描述已自动生成

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？

因为行存表和列存表的存储方式不同。

行存列表在执行以下类型的SQL时效率更高：

1. 频繁的更新、插入、删除操作；2. 查询结果集较小的SQL语句；3. 需要同时查询多个表的SQL语句。

列存表在执行以下类型的SQL时效率更高：

1. 大量数据的聚合查询、分组查询、统计查询等SQL语句；2. 需要查询大量列数据的SQL语句；3. 需要进行数据分析、挖掘等复杂计算的SQL语句。

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的主要差别在于物化视图的刷新方式和数据量。 全量物化视图每次刷新都需要重新计算和加载所有数据，这种视图适用于数据量较小或者数据更新频率较低的情况；增量物化视图每次刷新只需要计算和加载新增或修改的数据，这种视图适用于数据量较大或者数据更新频率较高的情况，可以提高查询效率和降低刷新成本。总的来说，全量物化视图适用于数据量较小或者数据更新频率较低的情况，而增量物化视图适用于数据量较大或者数据更新频率较高的情况。

# 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

电脑萤幕截图

描述已自动生成

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

图片包含 图形用户界面

描述已自动生成

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'

图形用户界面, 文本

描述已自动生成

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;

文本

描述已自动生成

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');

文本

描述已自动生成

3. 查看创建的虚拟索引列，将执行结果截图。

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

电脑屏幕的手机截图

描述已自动生成

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);

文本

描述已自动生成

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

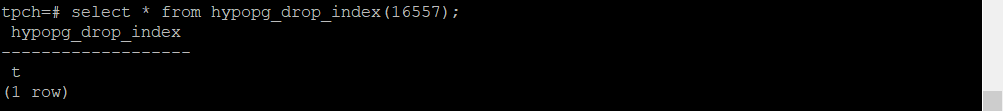
ORDER BY SUM(o.actual\_price) DESC;

文本

描述已自动生成

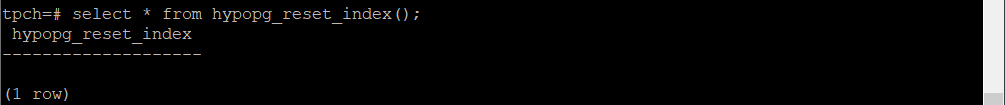
6. 删除某一个索引虚拟列，将执行结果截图。

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



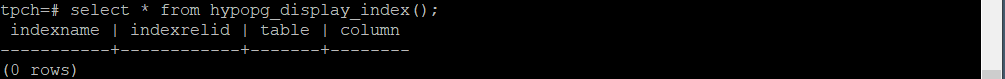
7. 删除某一个索引虚拟列，将执行结果截图。

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

电脑屏幕截图

描述已自动生成

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

文本

描述已自动生成

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？

X-Tuner在shared\_buffers ，max\_connections，effective\_cache\_size，effective\_io\_concurrency，wal\_buffers，random\_page\_cost，default\_statistics\_target参数上进行优化，X-Tuner对这些参数进行优化是因为它们对于PostgreSQL数据库的性能和稳定性有着重要的影响。以下是每个参数的作用和为什么需要进行优化：

1. shared\_buffers：这个参数控制PostgreSQL在内存中缓存数据的大小。如果设置得太小，会导致频繁的磁盘读写，从而影响性能。2. max\_connections：这个参数控制PostgreSQL可以同时处理的客户端连接数。如果设置得太小，会导致连接被拒绝，从而影响应用程序的可用性。3. effective\_cache\_size：这个参数控制PostgreSQL在查询时使用的缓存大小。如果设置得太小，会导致频繁的磁盘读取，从而影响性能。4. effective\_io\_concurrency：这个参数控制PostgreSQL在执行磁盘操作时使用的并发度。如果设置得太低，会导致磁盘操作的效率低下，从而影响性能。5. wal\_buffers：这个参数控制PostgreSQL在写入WAL日志时使用的缓存大小。如果设置得太小，会导致频繁的磁盘写入，从而影响性能。6. random\_page\_cost：这个参数控制PostgreSQL在查询时对随机I/O操作的成本估计。如果设置得太低，会导致PostgreSQL错误地选择使用随机I/O操作，从而影响性能。7. default\_statistics\_target：这个参数控制PostgreSQL在收集表统计信息时使用的样本大小。如果设置得太低，会导致统计信息不准确，从而影响查询优化器的决策。

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？

使用索引可以提高SQL查询的效率，因为索引可以帮助数据库快速定位需要查询的数据，而不必扫描整个表。这样可以大大减少查询所需的时间和资源。

除了使用索引和参数外，还有以下方面可以对数据库进行优化： 1. 数据库设计优化：合理的数据库设计可以减少数据冗余和重复，提高数据的一致性和完整性，从而提高数据库的性能。 2. 查询优化：优化SQL查询语句，避免使用复杂的子查询和连接操作，尽量使用简单的查询语句。 3. 缓存优化：使用缓存可以减少数据库的访问次数，提高查询效率。 4. 硬件优化：增加内存、CPU等硬件资源可以提高数据库的性能。 5. 数据库服务器优化：对数据库服务器进行优化，如调整参数、优化配置等，可以提高数据库的性能。 6. 数据库备份和恢复优化：合理的备份和恢复策略可以保证数据的安全性和可靠性，同时也可以提高数据库的性能。

# 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';

文本

描述已自动生成

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';

文本

描述已自动生成

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;

图形用户界面

中度可信度描述已自动生成

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的主要区别在于输出类型的不同。分类模型输出离散的标签或类别，而回归模型输出连续的数值。

实践思考题2：什么是SVM算法？

SVM算法是一种监督学习算法，用于分类和回归分析。它的基本思想是将数据映射到高维空间中，找到一个最优的超平面，将不同类别的数据分开。

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？

分类问题的评价指标包括准确率、精确率、召回率、F1值和ROC曲线等。

1. 准确率：分类正确的样本数占总样本数的比例。准确率越高，分类器的性能越好。

2. 精确率：真正例占预测为正例的比例。精确率衡量的是分类器预测为正例的准确性。

3. 召回率：真正例占实际为正例的比例。召回率衡量的是分类器对正例的识别能力。

4. F1值（：精确率和召回率的调和平均数，F1值越高，分类器的性能越好。

5. ROC曲线：ROC曲线是以假正例率为横轴，真正例率为纵轴，绘制出的曲线。ROC曲线下的面积越大，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？

1. 均方误差：均方误差是预测值与真实值之间差值的平方的平均值。均方误差越小，说明模型的预测结果越接近真实值。

2. 均方根误差：均方根误差是均方误差的平方根，它的单位与预测值的单位相同。均方根误差越小，说明模型的预测结果越接近真实值。

3. 平均绝对误差：平均绝对误差是预测值与真实值之间差值的绝对值的平均值。平均绝对误差越小，说明模型的预测结果越接近真实值。

4. 决定系数：决定系数用于衡量模型对数据的拟合程度，其取值范围为0到1。决定系数越接近1，说明模型对数据的拟合程度越好。

5. 相关系数：相关系数用于衡量两个变量之间的线性关系强度和方向，其取值范围为-1到1。相关系数越接近1或-1，说明两个变量之间的线性关系越强；相关系数越接近0，说明两个变量之间的线性关系越弱。