openGauss AI特性创新实践课



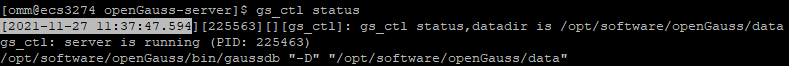
华为技术有限公司

# 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：

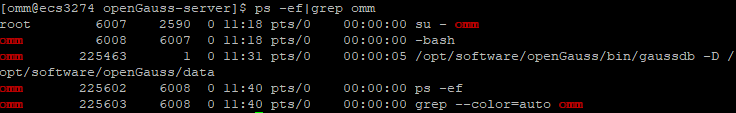
任务一：数据库状态验证

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？

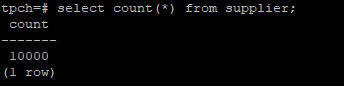
1. 源码编译安装可以按照自己的需求，进行定制化编译参数设定，能更确切的满足需求；相比之下包管理器安装则只能按照已封装好的参数进行安装，未必完全贴合需求；
2. 源码编译安装通常可以获取到更新的软件版本，并且选择安装版本时也更加自由；反之包管理器安装则只能选择被封装的版本，且一般更新相对较慢；
3. 源码编译安装可以自定义部署路径，进行统一管理，减轻了后期维护的工作量；然而包管理器安装不能自定义部署路径，后期维护工作量会更大；
4. 如果通过源码编译安装到用户目录，则安装时不需要有管理员权限，也不必考虑会对系统造成的污染；但包管理器安装则必须要求管理员权限，如果没有管理员权限将无法进行安装。

# 关卡二、openGauss数据导入及基本操作

任务一：数据初始化验证

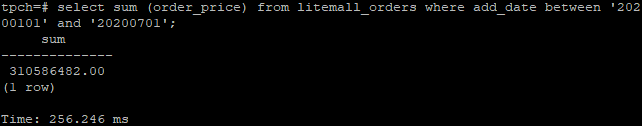
1. 查询supplier表的行数，并将结果进行图：

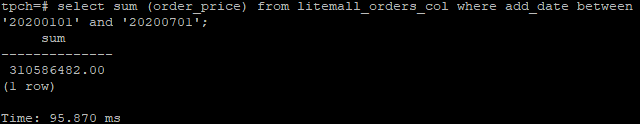
select count(\*) from supplier;



任务二：行存表与列存表执行效率对比

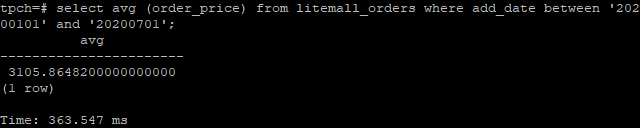
1. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的总和查询，并对比执行效率截图

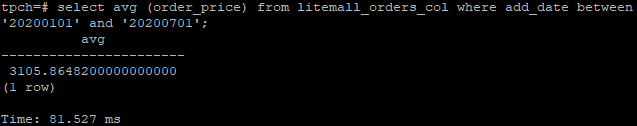
select sum (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';

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

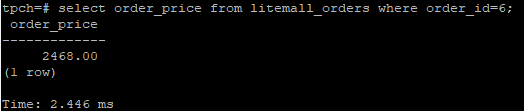
2. 2020年上半年litemall\_orders行存表与litemall\_orders\_col列存表中的order\_price的平均值查询，并对比执行效率截图

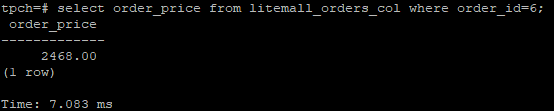
select avg (order\_price) from litemall\_orders where add\_date between '20200101' and '20200701';



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

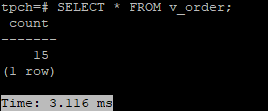
1. 创建物化视图所需要的表后，对表内容进行查询，对查询结果截图：

SELECT \* FROM test\_view;



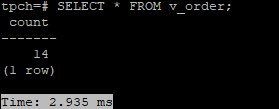
2. 使用物化视图统计人数，查询物化视图结果，将执行结果截图。

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？

1. 对于数据读取而言：
   1. 行存表通常将一行数据完全读出，若只需要其中部分列，则会存在冗余列；而列存表每次读取的数据是一列的一段或者全部，不存在冗余列；
   2. 行存表的一行数据中包含不同类型的数据，在数据解析时需要在多种数据类型之间进行转换；而列存表中的每一列数据类型都是相同的，数据解析更容易；
   3. 行存表若没有建立索引，查询时要耗费大量的I/O，但建立索引又要花费大量的时间与资源；而列存表中的数据本身即是索引，大大降低了I/O。
2. 对于数据写入而言：
   1. 行存表的写入是一次性完成一行的写入；而列存表需要把一行记录拆分为不同列保存，相比之下要花费更多的写入次数；
   2. 行存表在数据修改时对磁盘上指定位置做删除与写入一次；而列存表需要将磁盘定位到多个列上分别删除与写入。

实践思考题2：全量物化视图与增量物化视图有哪些差别？

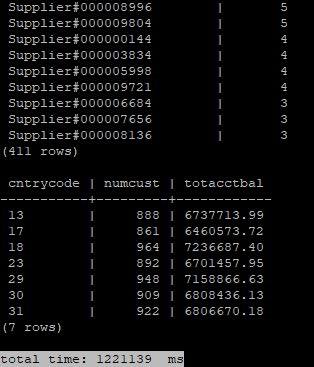
1. 全量物化视图即仅支持对创建好的物化视图做全量更新，而不支持做增量更新；而增量物化视图可以对物化视图进行增量更新，需要用户手动执行语句完成对物化视图在一段时间内的增量数据进行刷新。增量更新建立在全量更新的基础之上；
2. 目前而言全量物化视图所支持的功能与场景较广泛，而增量物化视图所支持场景较小；
3. 全量更新需要一次性将全部数据进行同步，因此需要占用大量时间和资源；而增量更新只抓取某个检查点之后的数据进行更新，占用时间与资源相对较少，但需要更新端记录或可以查询更新日志确定更新部分的数据。

# 关卡三、openGauss的AI4DB特性应用

任务一：使用X-Tuner进行参数优化

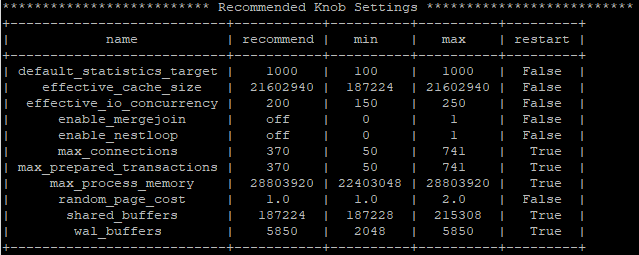
1. 执行TPCH脚本，获得测试时间，将执行结果截图：

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



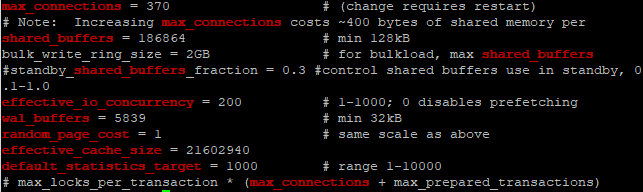
2. 使用root用户，执行X-Tuner进行参数建议优化，将执行结果截图

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

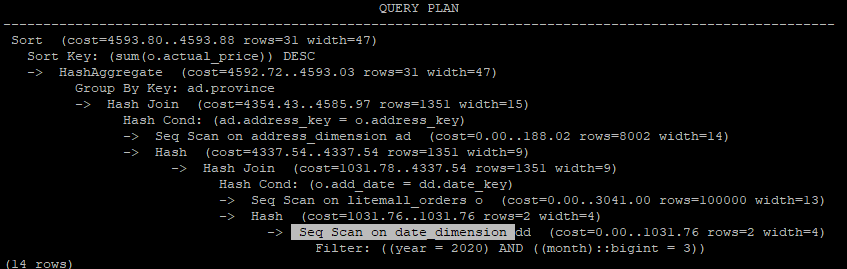
WHERE o.address\_key = ad.address\_key

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

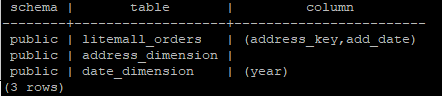
AND o.add\_date = dd.date\_key

AND dd.year = 2020

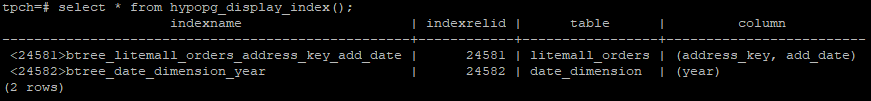
AND dd.month = 3

GROUP BY ad.province

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



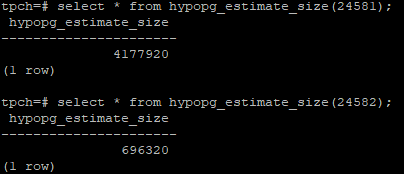
3. 查看创建的虚拟索引列，将执行结果截图。

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

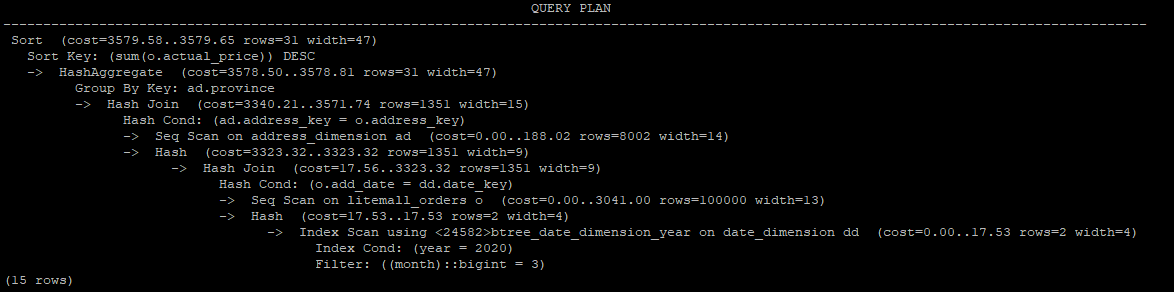
WHERE o.address\_key = ad.address\_key

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

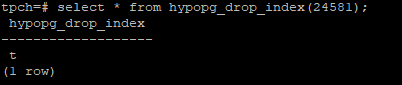
AND dd.year = 2020

AND dd.month = 3

GROUP BY ad.province

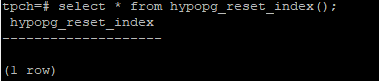
 ORDER BY SUM(o.actual\_price) DESC;

6. 删除某一个索引虚拟列，将执行结果截图。

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

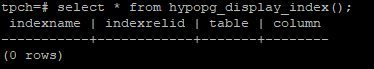
7. 删除某一个索引虚拟列，将执行结果截图。

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



8. 查看索引虚拟列，将执行结果截图。

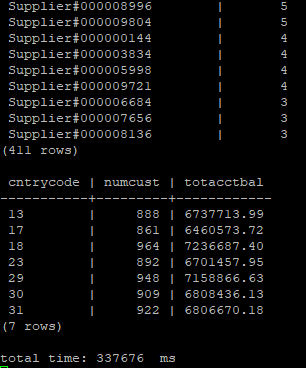
select \* from hypopg\_display\_index();



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

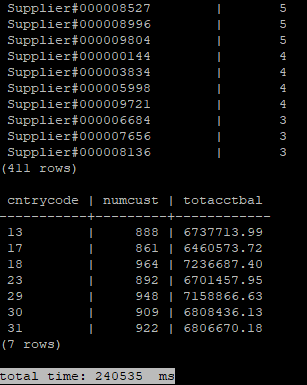


挑战一：进一步优化queries.sql中的查询语句，使得前后执行时间出现倍数级的提升。

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

利用EXPLAIN查看SELECT语句的开销，对于其中开销较大的语句，使用gs\_index\_advise()查看其推荐索引，并创建其推荐索引。



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？

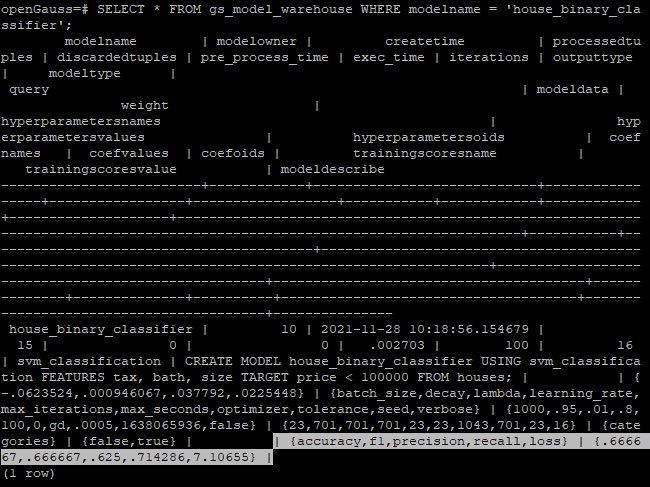
1. 参数：default\_statistics\_target、effective\_cache\_size、effective\_io\_concurrency、enable\_mergejoin、enable\_nestloop、max\_connections、max\_prepared\_transactions、max\_process\_memory、random\_page\_cost、shared\_buffers、wal\_buffers；
2. 原因：数据库在执行增、删、改、查语句时会产生不同的开销，如果能找到最影响性能的开销瓶颈点，对相应的瓶颈点进行参数优化，则可以获得较大的性能提升。

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. 索引的好处：
   1. 大大提高了数据检索的效率，降低数据库的I/O成本；
   2. 显著减少查询时分组和排序的时间；
   3. 可以加速表和表的连接。
2. 优化方法：
   1. 进行数据分页处理；
   2. 在输入SELECT语句时去除不必要的返回字段；
   3. 减少与服务器的交互次数。

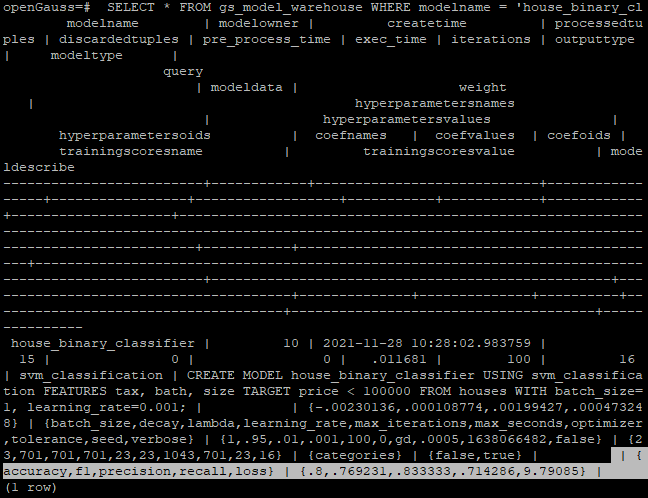
# 关卡四、openGauss的DB4AI特性应用

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：

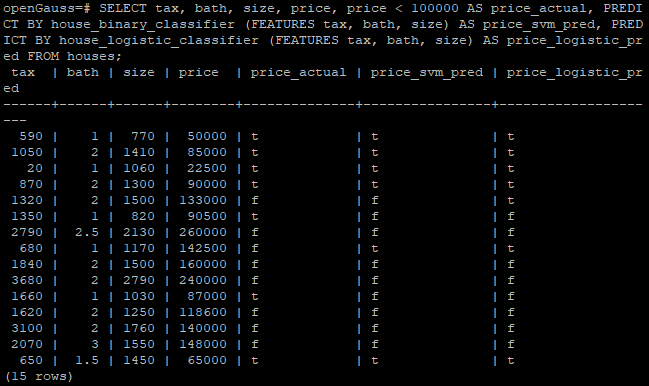
postgres=# SELECT \* FROM gs\_model\_warehouse WHERE modelname = 'house\_binary\_classifier';

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？

1. 首先分类模型与回归模型本质上解决的都是监督学习的问题；
2. 分类模型的输出是离散的值，即离散变量的预测，可以看作是“定性”输出，其目的是找到决策边界；
3. 回归模型的输出是连续的值，即连续变量的预测，可以看作是“定量”输出，其目的是找到最优拟合；

实践思考题2：什么是SVM算法？

支持向量机（SVM）的基本思想是求解能够正确划分训练数据集并且几何间隔最大的分离超平面。若假设数据集线性可分，则此类超平面有无穷多个，但几何间隔最大的分离超平面是唯一的，其关于所有样本点的几何间隔的最小值即为所谓支持向量到超平面的距离。因此SVM算法可视为求解凸二次规划的最优化算法。对于线性不可分的数据集，引入“软间隔”的概念，允许默写点不满足约束，并采用hinge损失改写原优化问题，用核函数替换内积，再求解。

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？

（主要以二分类问题进行分析，将实例分为正例与负例）

1. 混淆矩阵：对于模型分类的结果，将会出现真正类(True Postive TP)、假负类(False Negative FN)、假正类(False Postive FP)、真负类(True Negative TN)，基于此有以下标准：
   1. 准确率：
   2. 查准率（精确率）：
   3. 查全率（召回率）：
   4. 图：查准率—查全率曲线
   5. 曲线：“受试者工作特征”曲线，以“真正例率”（）为纵轴、“假正例率”（）为横轴。真正例率即查全率，假正例率为
   6. ：曲线下的面积
2. 代价矩阵：对于模型分类的结果，将会出现第类样本被预测为第类样本的情况，此时出现的代价记为。

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. （误差平方和）：
2. （决定系数）：
3. （校正决定系数）：