ALIBABA CLOUD

阿里云

云原生数据仓库 AnalyticDB PostgreSQL 版

性能指标

文档版本：20201015

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	**单击设置> 网络> 设置网络类型。**
粗体	表示按键、菜单、页面名称等UI元素。	**在结果确认页面，单击确定。**
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.TPC-C

AnalyticDB for PostgreSQL 6.0 (简称ADBPG6.0)在事务处理性能上相比上个版本ADBPG4.3有了质的飞跃，本文将以TPC-C业界标准事务性能测试benchmark来展示ADBPG6.0在事务上的处理能力。

TPC-C简介

TPC-C是由TPC(Transaction Processing Performance Council，事务处理性能委员会)提供的专门针对联机交易处理系统的规范，TPC-C模拟的是一个大型的商品批发销售公司交易负载。这个事务负载主要由9张表组成，主要涉及5类交易类型：新订单生成（New-Order）、订单支付（Payment）、发货（Delivery）、订单状态查询（Order-Status）、和库存状态查询（Stock-Level）。TPC-C测试使用吞吐量指标（Transaction per minute，简称tpmC)来衡量系统的性能，其中所统计的事务指的是新订单生成的事务，即以每分钟新订单生成的事务数来衡量系统的性能指标（在标准的TPC-C测试中，新订单的事务数量占总事务数的45%左右）。

测试环境

- 测试数据

使用开源的benchmarksql工具生成1000个warehouse的数据集，各个表的数据量如下表所示:

表名	数据量（行数）
bmsql_warehouse	1000
bmsql_district	10000
bmsql_customer	30000000
bmsql_history	30000000
bmsql_new_order	9000000
bmsql_oorder	30000000
bmsql_order_line	299976737
bmsql_item	100000
bmsql_stock	100000000

- 测试集群

测试的ADBPG6.0集群由16个计算节点，每个计算节点有4个CPU核，32GB内存，存储类型为高性能SSD，具体规格选用参考：规格及选型。

另外在同一个域里面建了一个ECS（放在与测试ADBPG实例相同的VPC中），然后在这个ECS上运行benchmarksql工具来生成TPC-C负载，向ADBPG实例发送请求。

- 集群配置参数

要获取极致的TP性能，需要对以下集群参数进行修改。其中有些参数用户无法自行设置，请联系ADBPG值班人员进行修改。

参数	说明
set optimizer = off	关闭ADBPG针对AP场景的orca优化器，对TP性能更友好。
set shared_buffers = 8GB	将ADBPG的数据共享缓存调大。修改该参数需要重启实例
set wal_buffers = 256MB	将ADBPG的WAL日志缓存调大。修改该参数需要重启实例。
set log_statement = none	将日志输出关闭。
set random_page_cost = 10	将随机访问代价开销调小，有利于查询走索引。
set gp_resqueue_priority = off set resource_scheduler = off	将ADBPG的resource queue关闭。需要重启实例。

测试结果

测试在不同并发度情况下，ADBPG6.0的性能结果，结果如下图所示，其中横坐标代表不同的并发数，纵坐标代表TPC-C性能（单位为tpmC），从图中可以看到，ADBPG6.0的峰值性能可以达到 101231.3 tpmC。

如何使用benchmarksql进行ADBPG的TPC-C测试

- 工具下载

使用ADBPG提供的benchmarksql，相比于开源版本，差异在于：1、由于ADBPG是分布式数据库，在建表语句时指定了分布键，一方面可以使得数据分布更加均匀，另一方面使得两阶段事务涉及计算节点数更少以便减少分布式事务的通信开销。2、修复了在生成csv数据集的时候，数据字段与建表字段不匹配的问题。

- 数据生成加载

1. 生成数据

使用如下命令可以生成TPC-C中各个表的csv数据，然后可以使用COPY方式将数据加载到数据库中。

```
# adbpg.properties为配置文件
./runLoader.sh adbpg.properties
```

在执行前，需要修改adbpg.properties中的如下内容：

```
db=postgres // 使用默认的postgres
driver=org.postgresql.Driver // 使用默认的org.postgresql.Drive
conn=jdbc:postgresql://xxxxx.xxxx.xxxx.rds.aliyuncs.com:3432/benchmarksql //adbpg实例连接地址
user=tpcc_test // 连接adbpg实例测试的用户
password=your_password // 对应用户的密码

warehouses=1000 // 指定生成数据集warehouse的数量
loadWorkers=16  // 指定生成数据的线程数量，如果CPU core和内存够用可以将其大小调大，这样速度更快
fileLocation=/data/tpcc_1000w/ // 指定生成数据的存储目录
```

2. 数据加载

(1) 建表: 执行benchmarksql工具提供的tableCreates.sql脚本。

(2) 数据加载: 使用copy命令将上节中生成的csv数据导入到ADBPG对应的表中。

```
# 使用COPY方式将数据加载到数据库中
psql -h conn_addr -d benchmarksql -p your_port -U tpcc_test -c "\copy bmsql_history from '/data/tpcc_
1000w/cust-hist.csv' with delimiter ',' null ';'"
```

(3) 创建索引: 执行benchmarksql工具提供的indexCreates.sql脚本。

(4) 执行analyze: 数据加载完并创建好索引后，需要对所有表执行analyze，以便ADBPG的优化器能做出好的执行计划。

- 执行测试

执行benchmarksql的如下脚本, 其中adbpg.properties为配置文件

```
./runBenchmark.sh adbpg.properties
```

在执行前，需要修改adbpg.properties中的如下内容

```
db=postgres // 使用默认的postgres
driver=org.postgresql.Driver // 使用默认的org.postgresql.Drive
conn=jdbc:postgresql://xxxxx.xxxx.xxxx.rds.aliyuncs.com:3432/benchmarksql //adbpg实例连接地址
user=tpcc_test // 连接adbpg实例测试的用户
password=your_password // 对应用户的密码

terminals=128 // TPC-C测试的并发数
//To run specified transactions per terminal- runMins must equal zero
runTxnsPerTerminal=0 // 使用默认的0
//To run for specified minutes- runTxnsPerTerminal must equal zero
runMins=10 // TPC-C执行时间，单位分钟
//Number of total transactions per minute
limitTxnsPerMin=30000000 // 使用默认的30000000

// 下面配置各个事务的比重，也使用如下给定的默认值
//The following five values must add up to 100
//The default percentages of 45, 43, 4, 4 & 4 match the TPC-C spec
newOrderWeight=45
paymentWeight=43
orderStatusWeight=4
deliveryWeight=4
stockLevelWeight=4
```

运行上面的脚本后，就会在屏幕上实时输出数据库当前处理的事务总数，等运行完runMins指定的时间后，屏幕就会输出如下所示的数据库TPC-C性能的tpmC数值：

Term-00, Running Average tpmTOTAL: 225114.24 Current tpmTOTAL: 7445652 Memory Usage: 116MB / 168MB18:28:56,388 [Thread-127] INFO jTPCC : Term-00,18:28:56,388 [Thread-127] INFO jTPCC : Term-00,18:28:56,388 [Thread-127] INFO jTPCC : Term-00, Measured tpmC (NewOrders) = 101231.3318:28:56,388 [Thread-127] INFO jTPCC : Term-00, Measured tpmTOTAL = 224908.1418:28:56,388 [Thread-127] INFO jTPCC : Term-00, Session Start = 2020-02-24 18:23:5618:28:56,388 [Thread-127] INFO jTPCC : Term-00, Session End = 2020-02-24 18:28:5618:28:56,388 [Thread-127] INFO jTPCC : Term-00, Transaction Count = 1125698

2.TPC-B

AnalyticDB for PostgreSQL 6.0 (简称ADBPG6.0)在事务处理性能上相比上个版本ADBPG4.3有了质的飞跃，本文将以TPC-B业界标准事务性能测试benchmark来展示ADBPG6.0在事务上的处理能力。

TPC-B简介

TPC-B是由TPC(Transaction Processing Performance Council，事务处理性能委员会)提供的benchmark，主要用于衡量一个系统每秒能够处理的并发事务数。TPC-B不像TPC-C那样模拟了现实生活中一个具体的交易场景，其中的事务都是由简单SQL构成的没有语义的事务（事务中混杂了大表与小表的插入、更新与查询操作），而且每个client的请求间也不会像TPC-C那样会有一个human think time的间隔时间，而是一旦前一个事务执行完成，立马会有下一个事务请求发出。因此，TPC-B经常用于对数据库系统的事务性能压测。TPC-B性能的衡量指标是每秒处理的事务数量，即TPS（Transactions per Second）。

测试环境

- 测试数据

使用PostgreSQL提供的开源工具pgbench来生成TPC-B的测试数据，填充因子为100，比例因子为11424。因此，最后TPC-B各个表的数据量如下：

表名	数据量（行数）
pgbench_accounts	1142400000
pgbench_branches	11424
pgbench_history	0
pgbench_tellers	114240

- 测试集群

测试的ADBPG6.0集群由16个计算节点，每个计算节点有4个CPU核，32GB内存，存储类型为高性能SSD，具体规格选用参考：规格及选型。另外，在同一个域里面建了一个ECS（放在与ADBPG实例相同的VPC中），在上面运行pgbench工具来生成TPC-B负载，向ADBPG发送请求。

- 集群配置参数

需要获取极致的TP性能，需要对一下参数进行修改。其中有些参数用户无法自行设置，请联系ADBPG值班人员进行修改。

参数	说明
set optimizer = off	关闭ADBPG针对AP场景的orca优化器，对TP性能更友好。
set shared_buffers = 8GB	将ADBPG的数据共享缓存调大。修改该参数需要重启实例。
set wal_buffers = 256MB	将ADBPG的WAL日志缓存调大。修改该参数需要重启实例。
set log_statement = none	将日志输出关闭。

参数	说明
set random_page_cost = 10	将随机访问代价开销调小，有利于查询走索引。
set gp_resqueue_priority = off set resource_scheduler = off	将ADBPG的resource queue关闭。需要重启实例

测试结果

测试在不同并发度情况下，ADBPG6.0的性能结果，结果如下图所示，其中横坐标代表不同的并发数，纵坐标代表测试性能（单位为TPS，transactions per second）。除了对ADBPG的TPC-B的性能测试之外，还使用pgbench对只读、只更新和只插入进行了测试。从下面的测试性能结果图中可以看出，ADBPG的TPC-B峰值性能可以达到 5923 TPS，只读峰值性能为 150084 TPS， 只更新峰值性能为31023 TPS，只插入的峰值性能为 60367 TPS。

- TPC-B性能

并发	1	8	16	32	48	64	96	128	168	192
TPS	199	1481	2031	2568	2795	3236	3622	5923	5037	5211
RT(ms)	5.02	5.4	7.88	12.5	17.2	19.8	26.5	21.6	33.3	36.8

- 只读性能

并发	1	8	16	32	48	64	96	128	160	192
TPS	1875	14226	26784	51115	72767	92370	125708	143297	150084	139637
RT(ms)	0.53	0.56	0.6	0.63	0.66	0.69	0.76	0.89	1.07	1.37

- 只更新性能

并发	1	8	16	32	48	64	96	128	160	192
TPS	787	6486	12127	19333	21083	24199	31023	27464	24362	23600
RT(ms)	1.27	1.23	1.32	1.66	2.28	2.64	3.09	4.66	6.57	8.14

- 只插入性能

并发	1	8	16	32	48	64	96	128	160	192
TPS	1497	11463	21771	40543	56887	59967	60367	59365	53375	48927

并发	1	8	16	32	48	64	96	128	160	192
RT(ms)	0.69	0.7	0.73	0.79	0.84	1.07	1.59	2.16	3.0	3.9

如何使用pgbench进行ADBPG的TPC-B测试

- 工具下载安装

有两种方式对pgbench工具进行安装：

1. 源码安装：下载开源数据库PostgreSQL源码，然后到pgbench对应的目录中单独对pgbench进行编译生成可执行的二进制文件。
2. 二进制安装：可以先直接yum install postgresql-server来安装PostgreSQL程序，此过程会自动安装pgbench工具。

- 数据加载

执行如下命令，会自动将数据生成加载到ADBPG实例中。

```
# 其中-F参数就是上文说的装填因子，-s值得是比例因子
./pgbench -i -F 100 -s 11424 -p port -h con_addr -U user_name db_name
```

- 执行测试

执行TPC-B负载

```
# -c指定了连接数据库client数量，-j指定了建立连接使用的线程数量，推荐将两者设置成一样
# -T指定了测试执行时间，单位为秒
./pgbench -h con_addr -p port -r -n -c 96 -j 96 -T 120 -f all.sql -U user_name db_name
```

其中all.sql为TPC-B负载，其内容如下：

```
\set scale 11424
\set nbranches 1 * :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000

BEGIN;
 UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
 SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
 UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
 UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
 INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta, CURRENT_TIME
STAMP);
END;
```

- 执行只读负载

```
# -c指定了连接数据库client数量，-j指定了建立连接使用的线程数量，推荐将两者设置成一样
# -T指定了测试执行时间，单位为秒
./pgbench -h con_addr -p port -r -n -c 96 -j 96 -T 120 -f select.sql -U user_name db_name
```

其中select.sql为只读负载，其内容如下：

```
\set scale 11424
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts

SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
```

- 执行只更新负载

```
# -c指定了连接数据库client数量，-j指定了建立连接使用的线程数量，推荐将两者设置成一样
# -T指定了测试执行时间，单位为秒
./pgbench -h con_addr -p port -r -n -c 96 -j 96 -T 120 -f update.sql -U user_name db_name
```

其中update.sql为只更新负载，其内容如下：

```
\set scale 11424
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom delta -5000 5000
UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
```

- 执行只插入负载

```
# -c指定了连接数据库client数量，-j指定了建立连接使用的线程数量，推荐将两者设置成一样
# -T指定了测试执行时间，单位为秒
./pgbench -h con_addr -p port -r -n -c 96 -j 96 -T 120 -f insert.sql -U user_name db_name
```

其中insert.sql为只插入负载，其内容如下：

```
\set scale 11424
\set nbranches 1 * :scale
\set ntellers 10 * :scale
\set naccounts 100000 * :scale
\setrandom aid 1 :naccounts
\setrandom bid 1 :nbranches
\setrandom tid 1 :ntellers
\setrandom delta -5000 5000
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta, CURRENT_TIME
STAMP);
```

3.TPC-H

AnalyticDB for PostgreSQL 6.0 （简称ADBPG6.0）在事务处理性能上相比上个版本ADBPG4.3有了质的飞跃，本文将以TPC-H业界标准事务性能测试benchmark来展示ADBPG6.0在事务上的处理能力。

TPC-H 简介

TPC-H（商业智能计算测试）是美国交易处理效能委员会（TPC,TransactionProcessing Performance Council）组织制定的用来模拟决策支持类应用的一个测试集。目前在学术界和工业界普遍采用它来评价决策支持技术方面应用的性能。TPC-H是根据真实的生产运行环境来建模，模拟了一套销售系统的数据仓库。其共包含8个基本关系，数据量可设定从1G~3T不等。其基准测试共包含了22个查询，主要评价指标各个查询的响应时间，即从提交查询到结果返回所需时间。其测试结果可综合反映系统处理查询时的能力。详情参考TPCH Specification

8张表逻辑关系

测试数据量的说明

数据量的大小对查询速度有直接的影响，TPC-H 中使用SF描述数据量，1SF 对应1GB 单位。1000SF，即1TB。1SF对应的数据量只是8个表的总数据量不包括索引等空间占用，准备数据时需预留更多空间。

AnalyticDB for PostgreSQL 6.0 规格选择

选择性价比适中的规格：

- ADB PG版 6.0 SSD存储+单节点4核+实例节点数32
- 使用官方标准TPC-H SQL

测试步骤

1. **开通一个ECS实例**

   准备一台ECS（建议规格：ecs.g6.4xlarge规格、CentOS系统、ESSD 2T数据盘，建议与AnalyticDB for PostgreSQL 6.0 实例用相同region和VPC网络），用于1T数据生成、数据上传/入库、客户端测试。

2. **开通一个AnalyticDB for PostgreSQL 6.0 实例**

   需要与ECS实例用相同区域、相同可用区和VPC网络。

3. **生成TPC-H 1T 测试数据**

   - SSH进入到ECS实例，下载TPC-H dbgen代码，编译后dbgen目录生成 dbgen/qgen 执行程序

     ```
     git clone https://github.com/gregrahn/tpch-kit.git
     cd tpch-kit/dbgen
     make
     ```

   - **生成1T数据，运行**

     ```
     ./dbgen --help
     ```

   - 查看如何生成，命令参考：

```
./dbgen -vf -s 1000
```

- 也可以并行生成分片文件，如下shell脚本参考（10个分片文件）：

```
for((i=1;i<=10;i++));
do
  ./dbgen -s 1000 -S $i -C 10 -f &
done
```

- 处理生成的 tbl 文件，tbl文件每行最后会多1个'|'，可以用seed命令将每行后面的'|'去掉，shell脚本参考：

```
sed -i 's/.$//' ./region.tbl &
sed -i 's/.$//' ./nation.tbl &
for((i=1;<=10;i++));
do
   sed -i 's/.$//' ./lineitem.tbl.$i &
   sed -i 's/.$//' ./orders.tbl.$i &
   sed -i 's/.$//' ./customer.tbl.$i &
   sed -i 's/.$//' ./partsupp.tbl.$i &
   sed -i 's/.$//' ./part.tbl.$i &
   sed -i 's/.$//' ./supplier.tbl.$i &
done
```

建表

列存表适合向量计算、JIT架构。对大批量数据的访问和统计，效率更高。因此建表语句中使用了

- AO列存表
- 不开压缩
- 设置复制表

```
create table nation (
   n_nationkey  integer not null,
   n_name       char(25) not null,
   n_regionkey  integer not null,
   n_comment    varchar(152))
with (appendonly=true, orientation=column)
distributed REPLICATED;

create table region (
   r_regionkey  integer not null,
   r_name       char(25) not null,
   r_comment    varchar(152))
```

```
with (appendonly=true, orientation=column)
distributed REPLICATED;

create table part (
    p_partkey     integer not null,
    p_name        varchar(55) not null,
    p_mfgr        char(25) not null,
    p_brand       char(10) not null,
    p_type        varchar(25) not null,
    p_size        integer not null,
    p_container   char(10) not null,
    p_retailprice DECIMAL(15,2) not null,
    p_comment     varchar(23) not null)
with (appendonly=true, orientation=column)
distributed by (p_partkey);

create table supplier (
    s_suppkey     integer not null,
    s_name        char(25) not null,
    s_address     varchar(40) not null,
    s_nationkey   integer not null,
    s_phone       char(15) not null,
    s_acctbal     DECIMAL(15,2) not null,
    s_comment     varchar(101) not null)
with (appendonly=true, orientation=column)
distributed by (s_suppkey);

create table partsupp (
    ps_partkey     integer not null,
    ps_suppkey     integer not null,
    ps_availqty    integer not null,
    ps_supplycost  DECIMAL(15,2)  not null,
    ps_comment     varchar(199) not null)
with (appendonly=true, orientation=column)
distributed by (ps_partkey);

create table customer (
    c_custkey     integer not null,
    c_name        varchar(25) not null,
    c_address     varchar(40) not null,
    c_nationkey   integer not null,
```

```
    c_phone       char(15) not null,
    c_acctbal     DECIMAL(15,2)  not null,
    c_mktsegment  char(10) not null,
    c_comment     varchar(117) not null)
with (appendonly=true, orientation=column)
distributed by (c_custkey);

create table orders (
    o_orderkey       bigint not null,
    o_custkey        integer not null,
    o_orderstatus    char(1) not null,
    o_totalprice     DECIMAL(15,2) not null,
    o_orderdate      date not null,
    o_orderpriority  char(15) not null,
    o_clerk          char(15) not null,
    o_shippriority   integer not null,
    o_comment        varchar(79) not null)
with (appendonly=true, orientation=column)
distributed by (o_orderkey);

create table lineitem (
    l_orderkey    bigint not null,
    l_partkey     integer not null,
    l_suppkey     integer not null,
    l_linenumber  integer not null,
    l_quantity    DECIMAL(15,2) not null,
    l_extendedprice  DECIMAL(15,2) not null,
    l_discount    DECIMAL(15,2) not null,
    l_tax         DECIMAL(15,2) not null,
    l_returnflag  char(1) not null,
    l_linestatus  char(1) not null,
    l_shipdate    date not null,
    l_commitdate  date not null,
    l_receiptdate date not null,
    l_shipinstruct char(25) not null,
    l_shipmode     char(10) not null,
    l_comment      varchar(44) not null)
with (appendonly=true, orientation=column)
distributed by (l_orderkey);
```

导入数据

准备工作就绪，可以开始导入数据了，导入数据有两种方式：

- 通过 copy from导入
- 通过OSS外表方式导入

下面分别介绍两种导入方法

COPY方式导入

SQL脚本参考：

```
\copy nation from '/data/tpch_1t/nation.tbl' DELIMITER '|';
\copy region from '/data/tpch_1t/region.tbl' DELIMITER '|';
\copy supplier from '/data/tpch_1t/supplier.tbl' DELIMITER '|';
\copy part from '/data/tpch_1t/part.tbl' DELIMITER '|';
\copy partsupp from '/data/tpch_1t/partsupp.tbl' DELIMITER '|';
\copy customer from '/data/tpch_1t/customer.tbl' DELIMITER '|';
\copy orders from '/data/tpch_1t/orders.tbl' DELIMITER '|';
\copy lineitem from '/data/tpch_1t/lineitem.tbl' DELIMITER '|';
```

tbl 路径以实际路径为准，导入shell脚本参考创建表的shell脚本（或psql进入数据库执行）。为提高导入效率（ECS网络带宽保障），可以把SQL拆开并发导入。

使用OSS外表方式导入数据

将生成的数据文件上传到oss

```
./ossutil64 cp -r <tbl文件目录> oss://<oss bucket>/<目录>/
                -i <AccessKey ID> -k <Access Key Secret>
                -e <EndPoint>
```

OSS外表文档参考：OSS外表高速导入或导出OSS数据

创建OSS外部表的建表语句示例

```
create readable external table ext_nation ( n_nationkey int, n_name varchar(25), n_regionkey integer,
   n_comment varchar(152))
   location('oss://oss-cn-beijing.aliyuncs.com
   filepath=data/tpch_data_1000x/nation.tbl
   id=$AccessKey key=$AccessKeySecret
   bucket=oss-y') FORMAT 'TEXT' (DELIMITER '|' ) ;


create readable external table ext_region ( R_REGIONKEY int, R_NAME CHAR(25),R_COMMENT VARCHAR(
152))
   location('oss://oss-cn-beijing.aliyuncs.com
```

```
  location('oss://oss-cn-beijing.aliyuncs.com
  filepath=data/tpch_data_1000x/region.tbl
  id=$AccessKey key=$AccessKeySecret
  bucket=oss-y') FORMAT 'TEXT' (DELIMITER '|' ) ;

CREATE readable external TABLE ext_lineitem ( l_orderkey bigint, l_partkey bigint, l_suppkey bigint,
 l_linenumber bigint, l_quantity double precision, l_extendedprice double precision,
 l_discount double precision, l_tax double precision, l_returnflag CHAR(1),
 l_linestatus CHAR(1), l_shipdate DATE, l_commitdate DATE, l_receiptdate DATE,
 l_shipinstruct CHAR(25), l_shipmode CHAR(10), l_comment VARCHAR(44))
  location('oss://oss-cn-beijing.aliyuncs.com
  filepath=data/tpch_data_1000x/lineitem.tbl
  id= $AccessKey key= $AccessKeySecret
  bucket=oss-y ') FORMAT 'TEXT' (DELIMITER '|' ) ;

CREATE readable external TABLE ext_orders ( o_orderkey bigint , o_custkey bigint , o_orderstatus CHA
R(1) ,
 o_totalprice double precision, o_orderdate DATE , o_orderpriority CHAR(15) , o_clerk CHAR(15) ,
 o_shippriority bigint , o_comment VARCHAR(79) )
  location('oss://oss-cn-beijing.aliyuncs.com
  filepath=data/tpch_data_1000x/orders.tbl
  id=$AccessKey key=$AccessKeySecret
  bucket=oss-y') FORMAT 'TEXT' (DELIMITER '|' ) ;

CREATE readable external TABLE ext_part ( p_partkey bigint , p_name VARCHAR(55) , p_mfgr CHAR(25) ,

 p_brand CHAR(10) , p_type VARCHAR(25) , p_size bigint , p_container CHAR(10) ,
 p_retailprice double precision , p_comment VARCHAR(23) )
  location('oss://oss-cn-beijing.aliyuncs.com
  filepath=data/tpch_data_1000x/part.tbl
  id= $AccessKey key= $AccessKeySecret
  bucket=oss-y') FORMAT 'TEXT' (DELIMITER '|' ) ;

CREATE readable external TABLE ext_partsupp ( ps_partkey bigint , ps_suppkey bigint ,
 ps_availqty bigint , ps_supplycost double precision , ps_comment VARCHAR(199) )
  location('oss://oss-cn-beijing.aliyuncs.com
  filepath=data/tpch_data_1000x/partsupp.tbl
  id= $AccessKey key= $AccessKeySecret
  bucket=oss-y') FORMAT 'TEXT' (DELIMITER '|' ) ;

CREATE readable external TABLE ext_supplier ( s_suppkey bigint , s_name CHAR(25) ,
```

```
 s_address VARCHAR(40) , s_nationkey bigint , s_phone CHAR(15) , s_acctbal DECIMAL(15,2) ,
 s_comment VARCHAR(101) )
  location('oss://oss-cn-beijing.aliyuncs.com
  filepath=data/tpch_data_1000x/supplier.tbl
  id= $AccessKey key= $AccessKeySecret
  bucket=oss-y') FORMAT 'TEXT' (DELIMITER '|' ) ;

CREATE readable external TABLE ext_customer ( c_custkey bigint , c_name VARCHAR(25) ,
 c_address VARCHAR(40) , c_nationkey bigint , c_phone CHAR(15) , c_acctbal double precision ,
 c_mktsegment CHAR(10) , c_comment VARCHAR(117) )
  location('oss://oss-cn-beijing.aliyuncs.com
  filepath=data/tpch_data_1000x/customer.tbl
  id= $AccessKey key= $AccessKeySecret
  bucket=oss-y') FORMAT 'TEXT' (DELIMITER '|' ) ;
```

从OSS外表写入TPC-H数据到AnalyticDB for PostgreSQL

```
insert into nation select * from ext_nation;
insert into region select * from ext_region;
insert into lineitem select * from ext_lineitem;
insert into orders select * from ext_orders;
insert into customer select * from ext_customer;
insert into part select * from ext_part;
insert into partsupp select * from ext_partsupp;
insert into supplier select * from ext_supplier;
```

至此 数据导入完毕，进入查询执行阶段

收集统计信息

```
analyze nation;
analyze region;
analyze lineitem;
analyze orders;
analyze customer;
analyze part;
analyze partsupp;
analyze supplier;
```

执行查询

使用如下shell脚本测试，也可以通过psql等其他客户端逐条执行查询SQL。具体的22条SQL语句见本文最后。

查询加速

特别的，AnalyticDB for PostgreSQL 6.0 的向量计算加速引擎，可大幅提升查询性能，在TPC-H场景下可提升1倍左右。

使用方法：

在session级别，修改参数 enable_odyssey为on，可开启加速引擎 。即执行如下SQL

```
set enable_odyssey = on;
```

如需关闭加速引擎，设置该参数为off。

```
set enable_odyssey = off;
```

如果使用如下脚本执行22条TPCH SQL，需要在每个Query文件开始出增加一行 `set enable_odyssey = on;`

执行全部查询，并记录每条耗时和总耗时

```
total_cost=0

for i in {1..22}
do
    echo "begin run Q${i}, query/q$i.sql , `date`"
    begin_time=`date +%s.%N`
    #psql -h ${实例连接地址} -p ${端口号} -U ${数据库用户} -f query/q${i}.sql > ./log/log_q${i}.out
    rc=$?
    end_time=`date +%s.%N`
    cost=`echo "$end_time-$begin_time"|bc`
    total_cost=`echo "$total_cost+$cost"|bc`
    if [ $rc -ne 0 ] ; then
        printf "run Q%s fail, cost: %.2f, totalCost: %.2f, `date`\n" $i $cost $total_cost
     else
        printf "run Q%s succ, cost: %.2f, totalCost: %.2f, `date`\n" $i $cost $total_cost
     fi
done
```

测试结果

各表数据量说明，1TB数据 (不含索引)

表名	数据条目数
customer	15000w

表名	数据条目数
lineitem	5999989709
nation	25
orders	150000w
part	20000w
partsupp	80000w
region	5
supplier	1000w

执行时间统计

SQL运行总时长（单位：秒）	4core * 32节点 SSD型	4core * 32节点 SSD型 - 自研向量化引擎
Total	**2179.85**	**1258.24**
Q1	399.38	171.05
Q2	25.32	12.24
Q3	56.91	38.26
Q4	54.26	20.20
Q5	145.64	118.72
Q6	30.61	21.19
Q7	71.43	63.79
Q8	73.58	37.84
Q9	174.09	169.28
Q10	51.56	36.96

SQL运行总时长（单位：秒）	4core * 32节点 SSD型	4core * 32节点 SSD型 - 自研向量化引擎
Q11	11.63	4.56
Q12	44.25	27.74
Q13	59.13	40.00
Q14	27.90	15.18
Q15	48.62	26.27
Q16	19.15	13.02
Q17	294.83	178.73
Q18	293.15	98.39
Q19	41.84	48.15
Q20	61.87	32.22
Q21	151.44	58.85
Q22	43.26	25.60

22个SQL

```
-- Q1
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
```

```
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= date '1998-12-01' - interval '93 day'
group by
    l_returnflag,
    l_linestatus
order by
    l_returnflag,
    l_linestatus;

-- Q2
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 23
    and p_type like '%STEEL'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
```

```
  and r_name = 'EUROPE'
  and ps_supplycost = (
    select
      min(ps_supplycost)
    from
      partsupp,
      supplier,
      nation,
      region
    where
      p_partkey = ps_partkey
      and s_suppkey = ps_suppkey
      and s_nationkey = n_nationkey
      and n_regionkey = r_regionkey
      and r_name = 'EUROPE'
  )
order by
  s_acctbal desc,
  n_name,
  s_name,
  p_partkey
limit 100;

-- Q3
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
  l_orderkey,
  sum(l_extendedprice * (1 - l_discount)) as revenue,
  o_orderdate,
  o_shippriority
from
  customer,
  orders,
  lineitem
where
  c_mktsegment = 'MACHINERY'
  and c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and o_orderdate < date '1995-03-24'
```

```
    and l_shipdate > date '1995-03-24'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
limit 10;

-- Q4
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date '1996-08-01'
    and o_orderdate < date '1996-08-01' + interval '3' month
    and exists (
        select
            *
        from
            lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority;

-- Q5
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    n_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue
```

```
from
   customer,
   orders,
   lineitem,
   supplier,
   nation,
   region
where
   c_custkey = o_custkey
   and l_orderkey = o_orderkey
   and l_suppkey = s_suppkey
   and c_nationkey = s_nationkey
   and s_nationkey = n_nationkey
   and n_regionkey = r_regionkey
   and r_name = 'MIDDLE EAST'
   and o_orderdate >= date '1994-01-01'
   and o_orderdate < date '1994-01-01' + interval '1' year
group by
   n_name
order by
   revenue desc;

-- Q6
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
   sum(l_extendedprice * l_discount) as revenue
from
   lineitem
where
   l_shipdate >= date '1994-01-01'
   and l_shipdate < date '1994-01-01' + interval '1' year
   and l_discount between 0.06 - 0.01 and 0.06 + 0.01
   and l_quantity < 24;

-- Q7
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
   supp_nation,
   cust_nation,
```

```
    cust_nation,
    l_year,
    sum(volume) as revenue
from
    (
        select
            n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            extract(year from l_shipdate) as l_year,
            l_extendedprice * (1 - l_discount) as volume
        from
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2
        where
            s_suppkey = l_suppkey
            and o_orderkey = l_orderkey
            and c_custkey = o_custkey
            and s_nationkey = n1.n_nationkey
            and c_nationkey = n2.n_nationkey
            and (
                (n1.n_name = 'JORDAN' and n2.n_name = 'INDONESIA')
                or (n1.n_name = 'INDONESIA' and n2.n_name = 'JORDAN')
            )
            and l_shipdate between date '1995-01-01' and date '1996-12-31'
    ) as shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by
    supp_nation,
    cust_nation,
    l_year;

-- Q8
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
```

```
select
  o_year,
  sum(case
    when nation = 'INDONESIA' then volume
    else 0
  end) / sum(volume) as mkt_share
from
  (
    select
      extract(year from o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) as volume,
      n2.n_name as nation
    from
      part,
      supplier,
      lineitem,
      orders,
      customer,
      nation n1,
      nation n2,
      region
    where
      p_partkey = l_partkey
      and s_suppkey = l_suppkey
      and l_orderkey = o_orderkey
      and o_custkey = c_custkey
      and c_nationkey = n1.n_nationkey
      and n1.n_regionkey = r_regionkey
      and r_name = 'ASIA'
      and s_nationkey = n2.n_nationkey
      and o_orderdate between date '1995-01-01' and date '1996-12-31'
      and p_type = 'STANDARD BRUSHED BRASS'
  ) as all_nations
group by
  o_year
order by
  o_year;

-- Q9
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
```

```
select
  nation,
  o_year,
  sum(amount) as sum_profit
from
  (
    select
      n_name as nation,
      extract(year from o_orderdate) as o_year,
      l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
    from
      part,
      supplier,
      lineitem,
      partsupp,
      orders,
      nation
    where
      s_suppkey = l_suppkey
      and ps_suppkey = l_suppkey
      and ps_partkey = l_partkey
      and p_partkey = l_partkey
      and o_orderkey = l_orderkey
      and s_nationkey = n_nationkey
      and p_name like '%chartreuse%'
  ) as profit
group by
  nation,
  o_year
order by
  nation,
  o_year desc;

-- Q10
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
  c_custkey,
  c_name,
  sum(l_extendedprice * (1 - l_discount)) as revenue,
  c_acctbal
```

```
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= date '1994-08-01'
    and o_orderdate < date '1994-08-01' + interval '3' month
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc
limit 20;

-- Q11
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    ps_partkey,
    sum(ps_supplycost * ps_availqty) as value
from
    partsupp,
    supplier,
    nation
where
    ps_suppkey = s_suppkey
```

```
    and s_nationkey = n_nationkey
    and n_name = 'INDONESIA'
group by
    ps_partkey having
        sum(ps_supplycost * ps_availqty) > (
            select
                sum(ps_supplycost * ps_availqty) * 0.0001000000
            from
                partsupp,
                supplier,
                nation
            where
                ps_suppkey = s_suppkey
                and s_nationkey = n_nationkey
                and n_name = 'INDONESIA'
        )
order by
    value desc;

-- Q12
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    l_shipmode,
    sum(case
        when o_orderpriority = '1-URGENT'
            or o_orderpriority = '2-HIGH'
            then 1
        else 0
    end) as high_line_count,
    sum(case
        when o_orderpriority <> '1-URGENT'
            and o_orderpriority <> '2-HIGH'
            then 1
        else 0
    end) as low_line_count
from
    orders,
    lineitem
where
    o_orderkey = l_orderkey
```

```
    and l_shipmode in ('REG AIR', 'TRUCK')
    and l_commitdate < l_receiptdate
    and l_shipdate < l_commitdate
    and l_receiptdate >= date '1994-01-01'
    and l_receiptdate < date '1994-01-01' + interval '1' year
group by
    l_shipmode
order by
    l_shipmode;

-- Q13
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    c_count,
    count(*) as custdist
from
    (
        select
            c_custkey,
            count(o_orderkey)
        from
            customer left outer join orders on
                c_custkey = o_custkey
                and o_comment not like '%pending%requests%'
        group by
            c_custkey
    ) as c_orders (c_custkey, c_count)
group by
    c_count
order by
    custdist desc,
    c_count desc;

-- Q14
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    100.00 * sum(case
        when p_type like 'PROMO%'
            then l_extendedprice * (1 - l_discount)
```

```
        then l_extendedprice * (1 - l_discount)
      else 0
    end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
    lineitem,
    part
where
    l_partkey = p_partkey
    and l_shipdate >= date '1994-11-01'
    and l_shipdate < date '1994-11-01' + interval '1' month;

-- Q15
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
create view revenue0 (supplier_no, total_revenue) as
    select
        l_suppkey,
        sum(l_extendedprice * (1 - l_discount))
    from
        lineitem
    where
        l_shipdate >= date '1997-10-01'
        and l_shipdate < date '1997-10-01' + interval '3' month
    group by
        l_suppkey;
select
    s_suppkey,
    s_name,
    s_address,
    s_phone,
    total_revenue
from
    supplier,
    revenue0
where
    s_suppkey = supplier_no
    and total_revenue = (
        select
            max(total_revenue)
        from
            revenue0
```

```
    )
order by
    s_suppkey;
drop view revenue0;

-- Q16
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    p_brand,
    p_type,
    p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
    partsupp,
    part
where
    p_partkey = ps_partkey
    and p_brand <> 'Brand#44'
    and p_type not like 'SMALL BURNISHED%'
    and p_size in (36, 27, 34, 45, 11, 6, 25, 16)
    and ps_suppkey not in (
        select
            s_suppkey
        from
            supplier
        where
            s_comment like '%Customer%Complaints%'
    )
group by
    p_brand,
    p_type,
    p_size
order by
    supplier_cnt desc,
    p_brand,
    p_type,
    p_size;

-- Q17
-- 开启向量加速引擎，并设置开关变量为on
```

```
set enable_odyssey = on;
select
  sum(l_extendedprice) / 7.0 as avg_yearly
from
  lineitem,
  part
where
  p_partkey = l_partkey
  and p_brand = 'Brand#42'
  and p_container = 'JUMBO PACK'
  and l_quantity < (
    select
      0.2 * avg(l_quantity)
    from
      lineitem
    where
      l_partkey = p_partkey
  );

-- Q18
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
  c_name,
  c_custkey,
  o_orderkey,
  o_orderdate,
  o_totalprice,
  sum(l_quantity)
from
  customer,
  orders,
  lineitem
where
  o_orderkey in (
    select
      l_orderkey
    from
      lineitem
    group by
```

```
        l_orderkey having
            sum(l_quantity) > 312
    )
    and c_custkey = o_custkey
    and o_orderkey = l_orderkey
group by
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice
order by
    o_totalprice desc,
    o_orderdate
limit 100;

-- Q19
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    sum(l_extendedprice* (1 - l_discount)) as revenue
from
    lineitem,
    part
where
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#43'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 5 and l_quantity <= 5 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#45'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 12 and l_quantity <= 12 + 10
        and p_size between 1 and 10
```

```
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
  )
  or
  (
    p_partkey = l_partkey
    and p_brand = 'Brand#11'
    and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
    and l_quantity >= 24 and l_quantity <= 24 + 10
    and p_size between 1 and 15
    and l_shipmode in ('AIR', 'AIR REG')
    and l_shipinstruct = 'DELIVER IN PERSON'
  );

-- Q20
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
  s_name,
  s_address
from
  supplier,
  nation
where
  s_suppkey in (
    select
      ps_suppkey
    from
      partsupp
    where
      ps_partkey in (
        select
          p_partkey
        from
          part
        where
          p_name like 'magenta%'
      )
      and ps_availqty > (
        select
          0.5 * sum(l_quantity)
```

```
            from
              lineitem
            where
              l_partkey = ps_partkey
              and l_suppkey = ps_suppkey
              and l_shipdate >= date '1996-01-01'
              and l_shipdate < date '1996-01-01' + interval '1' year
          )
    )
    and s_nationkey = n_nationkey
    and n_name = 'RUSSIA'
order by
    s_name;

-- Q21
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
    s_name,
    count(*) as numwait
from
    supplier,
    lineitem l1,
    orders,
    nation
where
    s_suppkey = l1.l_suppkey
    and o_orderkey = l1.l_orderkey
    and o_orderstatus = 'F'
    and l1.l_receiptdate > l1.l_commitdate
    and exists (
        select
            *
        from
            lineitem l2
        where
            l2.l_orderkey = l1.l_orderkey
            and l2.l_suppkey <> l1.l_suppkey
    )
    and not exists (
```

```
        select
          *
        from
          lineitem l3
        where
          l3.l_orderkey = l1.l_orderkey
          and l3.l_suppkey <> l1.l_suppkey
          and l3.l_receiptdate > l3.l_commitdate
    )
    and s_nationkey = n_nationkey
    and n_name = 'MOZAMBIQUE'
group by
    s_name
order by
    numwait desc,
    s_name
limit 100;

-- Q22
-- 开启向量加速引擎，并设置开关变量为on
set enable_odyssey = on;
select
        cntrycode,
        count(*) as numcust,
        sum(c_acctbal) as totacctbal
from
        (
                select
                        substring(c_phone from 1 for 2) as cntrycode,
                        c_acctbal
                from
                        customer
                where
                        substring(c_phone from 1 for 2) in
                                ('13', '31', '23', '29', '30', '18', '17')
                        and c_acctbal > (
                                select
                                        avg(c_acctbal)
                                from
                                        customer
                                where
```

```
                        c_acctbal > 0.00
                        and substring(c_phone from 1 for 2) in
                            ('13', '31', '23', '29', '30', '18', '17')
                )
                and not exists (
                    select
                        *
                    from
                        orders
                    where
                        o_custkey = c_custkey
                )
        ) as custsale
group by
        cntrycode
order by
        cntrycode;
```