



Database Test Guide

BorayDataRapidsDB POC Test Process

Weixun BorayData Technology (Beijing) Co., Ltd.

Building Poly International Plaza, No.7 Area of Wangjing

East Garden, Chaoyang District, Beijing

Telephone: +01064700868

Website: <http://www.boraydata.cn>

Copyright

Software copyright, copyright and intellectual property right involved herein have been legally registered according to applicable laws, are legally possessed by Weixun BorayData Technology (Beijing) Co., Ltd., and are protected by *Copyright Law of the People's Republic of China*, *Regulations on Computer Software Protection*, *Regulations on the Protection of IP Rights* and applicable international copyright treaties, laws, regulations and other intellectual property laws and treaties. It shall not be used illegally without authorization.

Disclaimer

Copyrights of BorayData involved herein are legally possessed by BorayData and are protected by laws. BorayData bears no responsibility for data not belonged to BorayData that are possibly involved herein. You

may inquire in the scope approved by laws and can only copy and print this document in the legal scope specified by *Copyright Law of the People's Republic of China*. Without written authorization of BorayData, any part or content of this document can't be used, modified or re-published by any organization or individual, otherwise it shall be regarded as infringement and BorayData has the right to investigate their responsibilities according to laws.

Information involved herein can be updated without further notice. In case of any problem, please directly inform or inquire Weixun BorayData Technology (Beijing) Co., Ltd.

All rights not expressly granted by the Company are reserved.

Communication mode is as follows:

Weixun BorayData Technology (Beijing) Co., Ltd. Room 503, No.19 Building (T1), Poly International Plaza, No.7 Area of Wangjing East Garden, Chaoyang District, Beijing Telephone: +01064700868 Website: http://www.boraydata.cn
--

Trademark Declaration



is registered by Weixun BorayData Technology (Beijing) Co., Ltd. at Trademark Office of China National Intellectual Property Administration. Its exclusive use right is legally possessed by BorayData and protected by laws. Without written approval of BorayData, any part of the trademark shall not be used, copied, modified, transmitted, transcribed or bundled with other parts for sales by any organization or individual in any way. Any infringement of BorayData trademark right will be investigated legally by BorayData.

Version Declaration

Products involved herein are Boray full-memory distributed analytical database (RpdSql version) and the version of relevant dependent packages is specified herein.

Version	Date of Issue	Effective Date	Author	Reviewer	Approver	Current Status
V1.0	April 6, 2021	April 6, 2021	Arman			Official
						Official
						Official
						Official
						Official
						Official
						Official

Notes:

1. *When using, please update “cover” and “revision history” to the cover and revision history of this document, and delete the explanatory text.*
2. *V1.0 corresponds to 001 on the page footer, and V2.0 corresponds to 002, and so on.*
3. *If there is no reviewer and approver, please fill in the corresponding cell with “/”.*

Contents

Copyright	I
Disclaimer	II
Trademark Declaration	III

Version Declaration	IV
1. Test Preparation	1
1.1 Hardware environment	1
1.2 Software environment	2
1.3 Recommended configuration	3
2. Resource Monitoring	4
3. Login and Use	5
3.1 RapidsDB user	5
3.2 Connector	5
3.3 RPDSQL privileges	6
4. Basic Function Test	9
5. Test Case	13
5.1 Table creating statements	14
5.2 Data preparation	21
5.3 Data import	21
5.4 Test statement	21
5.5 Test results	23
6. TPCCH Test	24
6.1 Data generation	24
6.2 Table creating statements	25
6.3 Data import	29
6.4 Data query	29
6.5 Query script	39
7. POC Test Optimization	40
7.1 Data partition	40
7.2 Shard key	41
7.2.1 Type of shard key	41
7.2.2 Functions of shard key	42
7.2.3 Selection of shard key	44
7.3 Row table	45
7.3.1 Create row table	45
7.3.2 Row table index	46
7.3.3 Selection of table index	48
7.4 Column table	48
7.4.1 Create column table	48
7.4.2 Selection of table index	50
7.5 Unique key	51
7.6 Data skew	53
7.6.1 Understand data skew	53
7.6.2 View data skew	54
7.7 Leaf and aggregator	56
7.8 Code generation	58
8. High availability test	61
9. Concurrent Test	63
9.1 Window installation and use	63
9.2 Linux installation and use	68
9.3 Create data by JMeter	70
10. Backup and Restoration	71
11. Connect JAVA	72

1. Test Preparation

Before POC test, tester should check software and hardware of the server, [including memory, CPU, disk, system version, network etc.](#), keep records of server configuration and prepare for follow-up test plan and test optimization at the same time.

1.1 Hardware environment

Now, let's take configuration of node1 server as an example:

- Check CPU model:

```
[root@node1~]# cat /proc/cpuinfo | grep name | cut -f2 -d: | uniq -c
72 Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz
```

- Check the number of physical CPU: 2

```
[root@node1~]# cat /proc/cpuinfo | grep 'physical id' | sort | uniq | wc -l
2
```

- Check the number of logic CPU: 72

```
[root@node1~]# cat /proc/cpuinfo | grep 'processor' | wc -l
72
```

- Check the memory size: 256GB

```
[root@node1~]# free -g
      total used free shared buff/cache available
Mem:   253 230 6 0 16 10
Swap:  3 0 3
```

- Check network card: network card enp59s0f0 is 10 gigabit lan

```
[root@node1~]# ethtool enp59s0f0
Settings for enp59s0f0:
Supported ports: [ FIBRE ]
Supported link modes:  1000baseT/Full
                      1000baseKX/Full
                      10000 baseKR/Full
```

An example of hardware information collection table is shown below:

Server	Configuration
CPU	40 Intel(R) Xeon(R) Silver 4210R

	CPU @ 2.40GHz
Memory	128 GB
Hard disk	7.3 TB
Network	Gigabit lan

1.2 Software environment

During operation according to this document, RapidsDB cluster has been installed and arranged successfully in the test server (if not installed and arranged, please refer to *RapidsDBv4.2.3 Installation Manual*) and starting and stop commands have been known (please refer to *RapidsDBv4.2.3 Operation and Maintenance Manual*) by default.

- Check version of operating system: CentOS7.6

```
[root@node1~]#cat/etc/centos-release
CentOS Linux release 7.6.1810 (Core)
```

- Check JDK and install openjdk:

```
[root@node1~]#java -version
openjdk version "1.8.0_282"
OpenJDK Runtime Environment (build 1.8.0_282-b08)
OpenJDK 64-Bit Server VM (build 25.282-b08, mixed mode)
[root@node1~]# yum install java-1.8.0-openjdk-devel //yum install openjdk
```

- Check IP address and hostname of the cluster:

```
[root@node1~]#cat /etc/hosts
192.168.0.A node1
192.168.0.Bnode2
```

- Turn off the firewall:

```
[root@node1~]#systemctl stop firewalld
```

- Start zookeeper:

```
[root@node1 ~]# cd /opt/zookeeper/bin
[root@node1 bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /data/zookeeper/bin/./conf/zoo.cfg
Starting zookeeper ... STARTED
[root@host20 bin]# ./zkServer.sh start
ZooKeeper JMX enabled by default
Using config: /data/zookeeper/bin/./conf/zoo.cfg
Starting zookeeper ... already running as process 20188. //it means successful startup
```

- Start RapidsDB cluster:

```
[root@node1 ~]# cd /opt/rdp/current
[root@node1 current] ./bootstrapper.sh -a start
```

An example of cluster information collection table is show below:

Node Name	Host	Role	Number of Leaves
node01	128.64.252.1	Master	N
node02	128.64.252.2	Leaf	N
node03	128.64.252.3	Leaf	N

An example of cluster information collection table is show below:

Software Name	Description
Operating system	CentOS 7.6
Database	RapidsDB v4.2.3
JDK	1.8.0_282

1.3 Recommended configuration

	Minimum Configuration	Recommended Configuration
CPU	2×2 cores	2×72 cores
Memory	4 GB	512 GB
Disk	100 GB	SSD 2TB
Network card	1000MB	10GB/25GB
Operating system	Above CentOS/RH 6.5	Above CentOS7.6

2、 Resource Monitoring

During testing, the customer may require monitoring resource utilization during usage of database or data import. If there isn't other monitoring software or Rapids Manager, following methods can be used for monitoring easily:

- Test the network speed:

```
[root@node1 ~]#sar -n DEV 1 100
//it represents one time executed per second, and 100 times executed in total.
```


Time	IFACE	rxpck/s	txpck/s	rxkB/s	txkB/s	rxcmp/s	txcmp/s	rxmcast/s
09时46分58秒	IFACE							
09时47分08秒	enp95s0f0	0.00	0.00	0.00	0.00	0.00	0.00	0.00
09时47分08秒	enp94s0f1	0.00	0.00	0.00	0.00	0.00	0.00	0.00
09时47分08秒	enp176s0f0	0.00	0.00	0.00	0.00	0.00	0.00	0.00
09时47分08秒	eno1	1.10	0.00	0.36	0.00	0.00	0.00	0.00
09时47分08秒	eno2	0.00	0.00	0.00	0.00	0.00	0.00	0.00
09时47分08秒	eno3	0.00	0.00	0.00	0.00	0.00	0.00	0.00
09时47分08秒	eno4	0.00	0.00	0.00	0.00	0.00	0.00	0.00
09时47分08秒	enp176s0f1	1.00	0.00	0.33	0.00	0.00	0.00	0.00
09时47分08秒	enp94s0f0	7284.70	16952.00	486.62	23990.16	0.00	0.00	0.00
09时47分08秒	lo	1801.40	1801.40	8670.13	8670.13	0.00	0.00	0.00
09时47分08秒	enp95s0f1	0.00	0.00	0.00	0.00	0.00	0.00	0.00

It represents that bandwidth currently used by network card enp94s0f0 is about: $7.2+1.8+16.9+1.8+0.48+8.67+23.99+8.67=70$ MB/S

- Test I/O of the disk:

```
[root@node1 ~]#iostat-d -k 1 100
```

// it represents one time executed per second, and 100 times executed in total.

Device:	tps	kB_read/s	kB_wrtn/s	kB_read	kB_wrtn
sda	0.00	0.00	0.00	0	0
sdb	517.10	41420.80	1761.60	414208	17616
dm-0	0.00	0.00	0.00	0	0
dm-1	0.00	0.00	0.00	0	0
dm-2	0.00	0.00	0.00	0	0

As database is included in the disk sdb, we can see that total i/o of the disk is about 43MB/S (41.4+1.7) during data import.

3、 Login and Use

3.1 RapidsDB user

The user can log in and out through rapids-shell at the terminal and the default user name is **RAPIDS** and the initial password is **rapids**.

RAPIDS user has all privileges:

```
[root@node1~]# cd /data/rdp/current
[root@node1current]# ./rapids-shell.sh
Please enter a username > rapids
Please enter the password for user 'RAPIDS' >
rapids >
```

We can alter the initial user password into 123456:

```
rapids > ALTER USER rapids PASSWORD '123456';
0 row(s) returned (1.22 sec)
```

Initial user can create and delete new user:

```

rapids > create user bigdata password 'bigdata';
0 row(s) returned (0.45 sec)
rapids > drop user bigdata ;
0 row(s) returned (1.12 sec)

```

3.2 Connector

We can firstly create database in the storage engine RPDSQL:

```

[root@node1~]# cd /data/rdp/rpdsq-ops
[root@node1rpdsq-ops]# ./rapids -P13306 //port number of master node during installation
of rpdsq
rapids > create database boraydata ;
Query OK, 1 row affected (14.36 sec)
rapids >quit ;

```

After quitting RPDSQL, enter RapidsDB again to create the connector:

```

[root@node1rpdsq-ops]# cd /data/rdp/current
[root@node1current ]# ./rapids-shell.sh
Please enter a username> rapids
Please enter the password for user 'RAPIDS' >
rapids > CREATE CONNECTOR BORAYDATA TYPE RPDSQL WITH
HOST='192.168.252.101',PORT=13306,USER='root',DATABASE='boraydata' NODE *
CATALOG * SCHEMA * TABLE *;
//where BORAYDATA is the name of the connector
//HOST is IP of node to be connected by the connector
//PORT is port number to be connected by the connector, and MASTER node or AGGREGATOR
node is used generally
//DATABASE is the name of database to be connected by the connector
rapids > show connectors; //check the information of the connector
rapids > use connector BORAYDATA; //use the connector
rapids > use connector rapids; //return to the no-connector status (otherwise, the system
determines that the connector BORAYDATA is being used, and the command of show connector
can't be used)
rapids > drop connector BORAYDATA; //delete the connector

```

3.3 RPDSQL privileges

The following shows setting of user and privileges in RPDSQL,

which can be logged in through RPDSQL without password in the initial state:

```
[root@node1~]# cd /data/rdp/rpdsq1-ops
[root@node1rpdsq1-ops]# ./rapids -P13306 //port number of master node during installation
of rpdsq1

rapids > //login successfully

rapids >create user test1 identified by '123456'; //create new user and add a password

rapids >create user test2 ; //create new user without password

rapids > show users; //check the number of current users

+-----+-----+-----+-----+-----+
| User | Type | Connections | Is deleted | Default resource pool |
+-----+-----+-----+-----+-----+
| 'root'@'%' | Native | 65 | | |
| 'test1'@'%' | Native | 0 | | |
| 'test2'@'%' | Native | 0 | | |
+-----+-----+-----+-----+-----+

rapids >quit; //quit
```

Then, we log in database through user test1:

```
[root@node1rpdsq1-ops]# ./rapids -P13306 -u test1 -p //specify a user name
Enter password://enter password 123456

rapids > //login successfully
```

User privileges, statements and relevant parameters are shown below:

```
GRANT priv_type [, priv_type [ ... ]] ON priv_level
TO user_or_role [, user_or_role [ ... ]]
[WITH GRANT OPTION]
[REQUIRE {SSL | NONE}]

priv_type:
ALL PRIVILEGES | SELECT | INSERT | UPDATE | DELETE | CREATE | DROP | RELOAD |
PROCESS | FILE READ | FILE WRITE | INDEX | ALTER | SHOW METADATA | GRANT
OPTION | SUPER | CREATE TEMPORARY TABLES | LOCK TABLES | CREATE VIEW |
SHOW VIEW | CREATE USER | CLUSTER | ALTER VIEW | DROP VIEW | BACKUP |
CREATE DATABASE | DROP DATABASE | CREATE PIPELINE | DROP PIPELINE | ALTER
PIPELINE | START PIPELINE | SHOW PIPELINE | EXECUTE | CREATE ROUTINE | ALTER
ROUTINE | CREATE LINK | DROP LINK | SHOW LINK

priv_level:
*
| *.*
| database.*
| database.table

user_or_role:
user [, user]
| role

user:
'user name'@'host name' [IDENTIFIED BY 'password']
```

role:

ROLE'role_name'

Now we log in the database at test1, its detailed information is as follows and there are no privileges:

```
rapids> show databases;
+-----+
| Database      |
+-----+
| information_schema |
+-----+
1 row in set (0.00 sec)
```

We switch to root user to grant **all** privileges, operated as follows:

```
[root@node1rpdsq-ops]# ./rapids -P13306
rapids> GRANT ALL PRIVILEGES ON *.* TO 'test1'@'%' identified by '123456' WITH
GRANT
        OPTION;           //represent granting all privileges
Query OK, 0 rows affected (0.27 sec)
rapids>quit;
```

After switching back to test1 user:

```
[root@node1rpdsq-ops]# ./rapids -P13306 -u test1 -p
Enter password://enter password 123456
rapids> show databases;
+-----+
| Database      |
+-----+
| cluster       |
| dbtest        |
| information_schema |
| test1         |
+-----+
9 rows in set (0.00 sec)
```

For new user, we should give special instructions during creation of the connector, for example:

```
[root@node1rpdsq-ops]# cd /data/rdp/current
[root@node1current]# ./rapids-shell.sh
Please enter a username > rapids
Please enter the password for user 'RAPIDS' >
rapids > CREATE CONNECTOR TEST TYPE RPDSQL WITH
HOST='192.168.252.101',PORT=13306,USER='test1',PASSWORD='123456',DATABASE='bora
ydata' NODE * CATALOG * SCHEMA * TABLE *;
rapids > use connector TEST;
```

4、 Basic Function Test

We firstly test basic functions of database, including addition, deletion and alteration, and transaction support etc.

- In case of bulk writing from other database, database name should be added before the table name:

```
rapids > insert into testdataselect * from boray.bigdata
//structure of two table should be consistent and testdata of the table should be included in
the current database
```

- Bulk deletion of data:

```
rapids > delete from testdata where id<100
//delete data about specific row
```

- Bulk alteration of data:

```
rapids > update testdata price = replace ( price , '50','100' );
//alter the value in the column of price, and value of primary key can't be altered
```

- Truncate table data:

```
rapids > truncate table testdata;
//truncate all (please use with caution if in a non-transaction situation)
```

- Data import

Parameters for importing CSV and TXT files are shown below:

```
LOAD DATA [LOCAL] INFILE '<file_name>'
[REPLACE | IGNORE | SKIP { ALL | CONSTRAINT | DUPLICATE KEY | PARSER }
ERRORS]
INTO TABLE <table_name>
[CHARACTER SET <character_set_name>]
[ {FIELDS | COLUMNS}
[TERMINATED BY '<string>' [[OPTIONALLY] ENCLOSED BY '<char>'] [ESCAPED BY
'<char>'] ]
[LINES
[STARTING BY '<string>']
[TERMINATED BY '<string>'] ]
```

```

[TRAILING NULLCOLS]
[NULL DEFINED BY <string> [OPTIONALLY ENCLOSED]]
[IGNORE <number> LINES]
[ ( {<column_name> | @<variable_name>}, ... ) ]
[SET <column_name> = <expression>,...]
[WHERE <expression>,...]
[MAX_ERRORS <number>]
[ERRORS HANDLE <string>])
  
```

Example for importing CSV file also applies to TXT file:

```

rapids > LOADDATA INFILE 'foo.csv' INTO TABLE foo (fourth, third, second, first);
//if column sequence is inconsistent with column sequence of data file during table creation,
you can name them explicitly.
  
```

```

rapids > LOAD DATA INFILE 'foo.txt' INTO TABLE foo (bar, @, @, baz);
//you can use @ symbol to skip the column in the source file. Only the first and the fourth
columns are imported to the table.
  
```

```

rapids >LOAD DATA INFILE 'foo.csv' INTO TABLE foo COLUMNS TERMINATED BY
',';
//data representing each column should be separated by commas, and COLUMNS can be
replaced by FIELDS
  
```

```

rapids > LOAD DATA INFILE 'numbers.txt' INTO TABLE foo COLUMNS TERMINATED
BY ','
TRAILING NULLCOLS; //it means that NULL is used to complement the null
value
  
```

```

rapids > load data infile "/tmp/harara.sql" replace into table orders fields terminated by ','
enclosed by '"' escaped by '\';
//it means enclosing data content in the text to prevent interference by other
characters
  
```

Parameters for importing JSON files are shown below:

```

LOAD DATA [LOCAL] INFILE 'file_name'
[REPLACE | SKIP { CONSTRAINT | DUPLICATE KEY } ERRORS]
INTO TABLE tbl_name
  
```

FORMAT JSON

subvalue_mapping

[SET col_name = expr,...]

[WHERE expr,...]

[MAX_ERRORS number]

[ERRORS HANDLE string]

subvalue_mapping:

({col_name | @variable_name} <- subvalue_path [DEFAULT literal_expr], ...)

subvalue_path:

{% | [%::]ident [::ident ...]}

An example for importing JSON file is shown below:

```
//for the following data
{"a":{"b":1}, "c":null}
{"a":{"b":2}, "d":null}

rapids > create table t (a INT);
rapids > LOAD DATA LOCAL INFILE "example.json" INTO TABLE t(a <- a:b) FORMAT
JSON;
rapids > select * from t;
a
-
2
1

2 row(s) returned (0.01 sec)
```

- Data export

Let's take export of CSV file as an example:

```
rapids > SELECT * FROM testdata INTO OUTFILE '/data/testdata.csv' FIELDS
TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' LINES TERMINATED
BY '\n';
//you can specify the contents, and specify which separator is used to divide the data in each row.
Row change character is used directly in this example.
```

- Transaction support

Now, we test the simple transaction support, as follows:

```
rapids > SELECT * FROM test1;
id name
-- ----
40 Nick
```

30 Jim

We give UPDATE test1:

```

rapids >BGIN;
rapids > update test1 set name = "John" where ID=300; //operate without error
rapids > SELECT * FROM test1; //as there isn't matched ID, UPDATE fails
id name
-- ----
40 Nick
30 Jim
rapids >ROLLBACK //so, rollback is operated

```

Then, we give UPDATE test2:

```

rapids >BGIN;
rapids > update test1 set name = "Jimmmy" where ID=30; //it prompts returning to 1 row
sucessfully
1 row(s) returned (0.00 sec)
rapids > SELECT * FROM test1;
id name
-- ----
40 Nick
30 Jimmmy
rapids >ROLLBACK //as value is altered incorrectly, i.e. typing Jimmy to Jimmmy, rollback
is operated.
rapids > SELECT * FROM test1; //return to the initial value
id name
-- ----
40 Nick
30 Jim

```

Finally, we give UPDATE test3:

```

rapids >BGIN;
rapids > update test1 set name = "Jimmy" where ID=30; //it prompts returning to 1 row
sucessfully
1 row(s) returned (0.00 sec)
rapids > SELECT * FROM test1;
id name
-- ----
40 Nick
30 Jimmy
rapids >COMMIT //all is well and transaction is committed.
rapids > SELECT * FROM test1; //altered results
id name
-- ----
40 Nick
30 Jimmy

```


5、 Test Case

The following is about history test cases. User test case in the business industry is selected. Test background is that the customer wants to optimize performance of accurate analysis of user portraits held by their customer managers, as traditional database is suffering from larger analysis pressure for query with randomly associated conditions from multiple large wide tables, and less analysis performance, with expansion of business index and business data. By virtue of distributed system structure based on memory, RapidsDB shows excellent performance advantages. On the test site, the customer provides three physical machines for preparation, detailed as follows:

5.1 Table creating statements

In the early stage of the test, it's necessary to communicate with the customer and obtain table structure of business data to be tested in the first place, and analyze whether table structure needs to be adjusted or optimized (considering creation of row table or column table). Specific table creating statements are shown below (statements for creating column table are shown below):

```
create table CST_BSC_INF_DPLT(
CST_ID CHAR(18),
IP_ID VARCHAR(19),
IP_NM_LND_CD CHAR(2),
CST_LGL_NM VARCHAR(600),
ID_INF_NO VARCHAR(120),
IDY_CD VARCHAR(5),
ENTP_SZ_CD CHAR(2),
CPCT_TPCD CHAR(6),
CPCT_CCB_OPNACC_YRLMT NUMERIC(3),
CPCT_CRGRD_CD CHAR(4),
KEY_CST_TPCD CHAR(2),
HQ_STRTG_CST_IND CHAR(1),
CST_BLIY_CGY_CD VARCHAR(5),
CST_BLIY_LRGCLSS_CD VARCHAR(5),
CST_BLIY_MDLCLS_CD VARCHAR(5),
ITNLSETLKEYCSTTPCD CHAR(4),
XAREAGRP_IND CHAR(1),
SPCL_IDY_IND CHAR(1),
HV_CRPND_RULE_IND CHAR(1),
GRPCST_IND CHAR(1),
CRCST_IND CHAR(1),
SML_ECST_IND CHAR(1),
CPCT_REL_MGT_FCN_CD CHAR(2),
CST_HOST_INSID CHAR(9),
EMPID CHAR(8),
```

ORG_CHAR_CD CHAR(4),
 BSC_DEP_ACC_DEPBNK_CD CHAR(6),
 LGL_RPRS_NM VARCHAR(180),
 IP_EXT_ID VARCHAR(240),
 RGST_CPAMT NUMERIC(19,3),
 RGST_CPTL_CCYCD CHAR(3),
 CMNT_2 VARCHAR(600),
 ORG_ESTB_DT DATE,
 ORG_OPRT_SCOP_DSC VARCHAR(900),
 ORGACINCCPTL_AMT NUMERIC(19,3),
 ENTFNDDPSNECNMPCD CHAR(3),
 CTYRGON_CD CHAR(3),
 ADIV_CD CHAR(6),
 DEPT_CGY_DSC VARCHAR(10),
 ORG_MAINBSN_DSC VARCHAR(600),
 ORG_MIX_BSN_DSC VARCHAR(600),
 ORGQF_GRD_CD VARCHAR(4),
 PNPPD_AND_BRND_DSC VARCHAR(240),
 MKT_LOT NUMERIC(10),
 FST_CR_REL_DT DATE,
 HV_LGLPSN_QUA_IND CHAR(1),
 DEP_ACC_TPCD CHAR(3),
 PARCO_IND CHAR(1),
 GRP_ID VARCHAR(19),
 CST_CSTGRP_REL_TPCD CHAR(7),
 CST_MGRP_ID CHAR(20),
 PRJ_LGLPSN_IND CHAR(1),
 LCL_YRMO_DAY DATE,
 LCL_HR_GRD_SCND CHAR(9),
 MULTI_TENANCY_ID CHAR(5),
 CST_CUR_STCD CHAR(6),
 POD NUMERIC(8,5),
 CRLNASTRSK_CL_RSLT_CD CHAR(2),
 STDT DATE,
 EXDAT DATE,
 LOAD_TM DATE,
 BTCH_NO VARCHAR(100),
 PRJ_FNL_RTG_GRD_CD CHAR(4),
 TMS TIMESTAMP,
 RMRK_1 VARCHAR(3000),
 RMRK_2 VARCHAR(3000),
 INPT_TM TIMESTAMP,
 TXN_DT DATE,
 LAST_UDT_DT_TM DATE,
 CPRSVCHRTC_CST_GRD_CD CHAR(3),
 CST_SCRN_RSLT_TPCD CHAR(1),
 CST_SCRN_RSLT_DSC VARCHAR(600),
 CPCT_ISBOND_IND VARCHAR(8),
 BSC_DAT_DEPBNK_ENG_NM VARCHAR(128),
 BSC_DEP_ACC_DPBK VARCHAR(600),
 CST_ENG_NM VARCHAR(600),
 CST_LO_ZON VARCHAR(600),
 CST_WBT_CITY VARCHAR(600),

```

CST_WBT_PROV VARCHAR(600),
AGNCBNK_IND VARCHAR(1),
ENLND_BK_IND VARCHAR(1),
IDY_RTG VARCHAR(2),
ASPD_ID VARCHAR(8),
TRDPT_CST_ID VARCHAR(23),
/*!90621 UNIQUE KEY pk (`CST_ID`,`MULTI_TENANCY_ID`) UNENFORCED RELY, */
/*!90618 SHARD */ KEY (`CST_ID`) /*!90619 USING CLUSTERED COLUMNSTORE */
);

```

```

create table CST_CPRSV_INF (
CST_ID CHAR(18),
DATA_DT DATE,
CST_NM VARCHAR(240),
CPCT_CRGRD_CO CHAR(4),
ORGQF_GRO_CD VARCHAR(4),
RGST_CPTL_NVAL NUMERIC(19,2),
GRP_NM VARCHAR(600),
CRGLN NUMERIC(19,2),
USED_LMT NUMERIC(19,2),
DEP_BAL NUMERIC(15,2),
DEP_AADBAL NUMERIC(17,2),
DEP_INTEXP_AMT NUMERIC(17,2),
CPCT_OTHR_LNBAL NUMERIC(19,2),
CPCT_LQUD_FNDS_LNBAL NUMERIC(21,4),
CPCT_SSPCS_CGY_LNBAL NUMERIC(21,4),
CPCT_FCS_CGY_LNBAL NUMERIC(21,4),
CPCT_FIX_AST_LNBAL NUMERIC(19,2),
CPCT_LNBAL NUMERIC(15,2),
COLL_INT_BAL NUMERIC(19,2),
CPCT_SCND_CGY_LNBAL NUMERIC(21,4),
CPCT_BDLNRT NUMERIC(7,5),
OFFBALSHET_BAL NUMERIC(17,2),
CPCTNONLGOFBALSHETBAL NUMERIC(19,2),
CPCT_LGNT_BAL NUMERIC(19,2),
CPCT_RGLR_CGY_LNBAL NUMERIC(21,4),
RECINT NUMERIC(15,2),
LC_BAL NUMERIC(15,2),
CPCT_BL_DSCT_LNBAL NUMERIC(21,4),
CPCT_LOSS_CGY_LNBAL NUMERIC(21,4),
CPCT_HOLD_PD_NUM NUMERIC(10),
FNC_TNUM_BAL NUMERIC(17,2),
CRNYR_DBT_TDNUM NUMERIC(10),
CRNYR_DHAMT NUMERIC(19,2),
CRNYR_CR_TDNUM NUMERIC(10),
CRNYR_CR_HPNUM NUMERIC(19,2),
INR_SFT_EXPN_AMT NUMERIC(31,10),
INR_SFT_INCMAM NUMERIC(17,2),
CST_CTB_NVAL NUMERIC(15,2),
CPCT_DEP_GRSINCMAMT NUMERIC(15,2),
LNIN_INCMAM NUMERIC(17,2),
CPCT_ASTCGYPD_INCMAM NUMERIC(19,2),
CPCT_INTRBSN_INCM_PCT NUMERIC(19,8),
CRNMO_INTRBSN_INCMAM NUMERIC(31,10),

```

```

INTRBSN_EXPN_AMT NUMERIC(17,2),
CPCTINTRBSNGRSINCMAMT NUMERIC(15,2),
RT12MOCPCIBGRSINCMAMT NUMERIC(17,2),
CPCT_PREV_ANUL_SETAMT NUMERIC(19,2),
CPCTPREVANULSETL_DNUM NUMERIC(10),
CPCTRTOLYCPSAMTINCRRT NUMERIC(10,5),
CPCTRTOLYCPSDNMINCRRT NUMERIC(10,5),
CPCT_CRNYR_CNY_SETAMT NUMERIC(19,2),
CPCT_BADB_LNBAL NUMERIC(21,4),
CPCT_WTRCSM NUMERIC(17,4),
CPCT_ELCCSM NUMERIC(17,4),
CPCT_PRJ_RSRV_NUM NUMERIC(10),
LOAD_TM DATE,
BICH_NO VARCHAR(100),
DEP_MO_DABAL NUMERIC(17,2),
LN_AADBAL NUMERIC(17,2),
LN_MO_DABAL NUMERIC(17,2),
CPCT_EVA_VAL NUMERIC(19,2),
CRNYR_FNCLTX_DNUM NUMERIC(17,2),
INTRBSN_NET_INCMAM NUMERIC(17,2),
CCB_BSCACC_DPSKINNO CHAR(9),
MULTI_TENANCY_ID CHAR(5),
CPCT_RAROC_RATE NUMERIC(19,8),
CRNYR_INTRBSN_INCMAM NUMERIC(15,2),
LAST_UDT_DT_TM DATE,
RCRD_UDT_LCL_DT DATE,
RCRD_UDT_LCL_TM CHAR(6),
TXN_DT DATE,
OBSBSN_AADBAL NUMERIC(15,2),
CR_BAL NUMERIC(15,2),
CRNMO_CF_CVR NUMERIC(19,2),
LYSP_CR_HP NAM NUMERIC(19,2),
CRNMO_CR_HP NAM NUMERIC(19,2),
LYSP_CR_TDNUM NUMERIC(10),
CRNMO_CR_TDNUM NUMERIC(10),
LYSP_DHAMT NUMERIC(19,2),
CRNMO_DHAMT NUMERIC(19,2),
LYSP_DBT_TDNUM NUMERIC(10),
CRNMO_DBT_TDNUM NUMERIC(10),
/*!90621 UNIQUE KEY pk (`CST_ID`,`DATA_DT`) UNENFORCED RELY, */
/*!90618 SHARD */ KEY (`CST_ID`) /*!90619 USING CLUSTERED COLUMNSTORE */
);

```

```

CREATE TABLE LNACC_INF (
ACC_ID VARCHAR(40),
DATA_DT DATE,
CST_ID CHAR(18),
CCYCD CHAR(3),
DEPBANK_ID VARCHAR(19),
CR_ACCNM VARCHAR(600),
CRG_VRTY_ID CHAR(8),
CR_VRTY_NM VARCHAR(600),
CPCT_LNBAL NUMERIC(15,2),
IN_AADBAL NUMERIC(17,2),

```

```

CTR_ID VARCHAR(240),
CTR_AMT NUMERIC(15,2),
LOAN_YR_INTRT NUMERIC(10,6),
ACC_ST CHAR(7),
STDT DATE,
EXDAT DATE,
FST_DSBR_DT DATE,
PNP_RCYC_AMT NUMERIC(19,2),
NONACRALRDY_EXP_PNAMT NUMERIC(15,2),
ODUAMT NUMERIC(19,8),
LOAD_TM DATE,
BTCH_NO VARCHAR(100),
MULTI_TENANCY_ID CHAR(5),
LDGR_INSID CHAR(9),
LAST_UDT_DT_TM DATE,
RCRD_UDT_LCL_DT DATE,
RCRD_UDT_LCL_TM CHAR(6),
TXN_DT DATE,
GRTSTL_CMNT VARCHAR(150),
LNACC_INTICM_AMT NUMBER(19,2),
/*!90621 UNIQUE KEY pk (`ACC_ID`, `CST_ID`) UNENFORCED RELY, */
/*!90618 SHARD */ KEY (`ACC_ID`) /*!90619 USING CLUSTERED COLUMNSTORE */
);

```

```

CREATE TABLE INST_CST_DLY_INF (
CST_ID CHAR(18),
INST_ECD CHAR(9),
DATA_DT DATE,
CST_NM VARCHAR(240),
CST_SZ_CD CHAR(2),
CORP_DEP_BAL NUMERIC(17,2),
CORP_DEP_AADBAL NUMERIC(17,2),
ACC_NUM NUMERIC(6),
CRN_BAL NUMERIC(15,2),
TRM_BAL NUMERIC(15,2),
SMBSN_DEP_BAL NUMERIC(17,2),
SMBSN_DEP_AADBAL NUMERIC(17,2),
FNCL_DEP_BAL NUMERIC(17,2),
FNCL_DEP_AADBAL NUMERIC(17,2),
CORP_LNBAL NUMERIC(17,2),
CORP_LN_AADBAL NUMERIC(17,2),
TRDFNC_AADBAL NUMERIC(17,2),
BRKEVN_CHRTC_BAL NUMERIC(17,2),
NON_BRKEVN_CHMTPD_BAL NUMERIC(17,2),
CPCT_NON_DSCT_LNBAL NUMERIC(19,2),
CPCT_LQUOD_FNDS_LNBAL NUMERIC(21,4),
CPCT_FIX_AST_LNBAL NUMERIC(19,2),
CPCTRLESTDVPOPRTLNBAL NUMERIC(21,4),
CPCTIDFCYHS_CGY_LNBAL NUMERIC(19,2),
CPCTFCTRNAVPMYMTITMBAL NUMERIC(19,2),
CPCT_BL_DSCT_LNBAL NUMERIC(21,4),
CPCTNONDSCT_LN_AADBAL NUMERIC(19,2),
CPCTLQUODFNDSL_N_AADBAL NUMERIC(19,2),

```

```

CPCTFIX_AST_LN_AADBAL NUMERIC(19,2),
CPCTRLESTCGYLN_AADBAL NUMERIC(19,2),
CPCTIDFCYHSCGYLAADBAL NUMERIC(19,2),
CPCTFCTRNAVPTIAADBAL NUMERIC(19,2),
CPCT_DSCT_AADBAL NUMERIC(19,2),
CORP_LNACC_NUM NUMERIC(10),
CPCTOTHRINSTLNACC_NUM NUMERIC(10),
CPCTACPT_DRFTBILL_BAL NUMERIC(19,2),
CPCT_DMST_LGNT_BAL NUMERIC(19,2),
CPCT_OVSEA_LGNT_BAL NUMERIC(19,2),
CPCT_RGLR_CGY_LNBAL NUMERIC(21,4),
CPCT_SCND_CGY_LNBAL NUMERIC(21,4),
CPCT_SSPCS_CGY_LNBAL NUMERIC(21,4),
CPCT_LOSS_CGY_LNBAL NUMERIC(21,4),
CPCT_FCS_CGY_LNBAL NUMERIC(21,4),
LOANTODEPRTO NUMERIC(19,2),
CPCT_HOLD_PD_NUM NUMERIC(10),
LOAD_TM DATE,
BTCH_NO VARCHAR(100),
MULTI_TENANCY_ID CHAR(5),
CPCTBRKEVNCFADCMDPBAL NUMERIC(17,2),
CPCT_ADVDEP_BAL NUMERIC(21,4),
CPCT_AGRM_DEP_BAL NUMERIC(21,4),
CPCTBIGAMTCTFOFDEPBAL NUMERIC(21,4),
CPCT_STC_DEP_BAL NUMERIC(21,4),
CPCT_NRA_DEP_BAL NUMERIC(21,4),
CPCT_INST_DEP_BAL NUMERIC(21,4),
CPCT_PRVDFND_DEP_BAL NUMERIC(21,4),
CPCT_FTA_DEP_BAL NUMERIC(21,4),
LAST_UDT_DT_TM DATE,
RCRD_UDT_LCL_DT DATE,
RCRD_UDT_LCL_TM CHAR(6),
TXN_DT DATE,
DEP_MO_DABAL NUMERIC(13,2),
CORP_CR_AADBAL NUMERIC(19,2),
CORP_CR_BAL NUMERIC(19,2),
NEW_LOANTODEPRTO NUMERIC(19,2),
/*!90621 UNIQUE KEY pk ('CST_ID','DATA_DT') UNENFORCED RELY, */
/*!90618 SHARD */ KEY ('CST_ID') /*!90619 USING CLUSTERED COLUMNSTORE */
);
    
```

5.2 Data preparation

Table Name	Column Number	Data Size	Data Volume
CST_BSC_INF_DPLT	84	11GB	15,867,399
CST_CPRSV_INF	82	5.8GB	17,322,047
INST_CST_DLY_INF	65	14GB	53,328,065
LNACC_INF	30	7.6GB	27,097,498

Actual data volume of the business is shown above. If the tester wants to copy the test process, please use JMeter to create data to fill the

table according to 9.3.

5.3 Data import

```

rapids > load data infile'/home/data/CST_BSC_INF_DPLT.csv' into table
CST_BSC_INF_DPLT
        fields terminated by '|' lines terminated by '\n'; //alter the delimiter according to
actual data
15,867,399 row(s) returned (26.55 sec)

rapids > load data infile'/home/data/CST_CPRSV_INF.csv' into table CST_CPRSV_INF
fields
        terminated by '|' lines terminated by '\n'; //alter the delimiter according to actual
data
17,322,047 row(s) returned (28.24 sec)

rapids > load data infile'/home/data/INST_CST_DLY_INF.csv' into table
INST_CST_DLY_INF
        fields terminated by '|' lines terminated by '\n'; //alter the delimiter according to
actual data
53,328,065 row(s) returned (74.36 sec)

rapids > load data infile'/home/data/LNACC_INF.csv' into table LNACC_INF
        fields terminated by '|' lines terminated by '\n'; //alter the delimiter according to
actual data
27,097,498 row(s) returned (40.11 sec)

```

5.4 Test statement

In the early stage of the test, it's necessary to ask for SQL of business to be tested along with communication about customer demand and table structure, prepare in advance, and check whether all functions can be supported and whether SQL statement needs to be altered or optimized. Actual business statements of the customer are shown below:

```

SQL 1:
select t.CST_ID,
t.IP_ID,
t.CST_LGL_NM,
t.ID_INF_NO,
t.CPCT_TPCD,
t1.DEP_BAL,
t1.DEP_AADBAL,
t1.DEP_INTEXP_AMT,
t1.CPCT_OTHR_LNBAL,

```

```

t1.CPCT_LQUID_FNDS_LNBAL,
t2.CTR_AMT,
t2.STDT,
t2.FST_DSBR_DT,
t2.PNP_RCYC_AMT,
t2.ODUAMT,
t3.CORP_DEP_BAL,
t3.ACC_NUM,
t3.CRN_BAL,
t3.TRM_BAL,
t3.SMBSN_DEP_BAL
from CST_BSC_INF_DPLT t
left join CST_CPRSV_INF t1 on t.CST_ID = t1.CST_ID and DATA_DT ='2020-06-30'
left join (select CST_ID,
sum(CTR_AMT) as CTR_AMT,
min(STDT) AS STDT,
min(FST_DSBR_DT) AS FST_DSBR_DT,
sum(PNP_RCYC_AMT) AS PNP_RCYC_AMT,
sum(ODUAMT) AS ODUAMT
from LNACC_INF where DATA_DT='2020-06-20' AND CCYCD='156' AND
ACC_ST='0002002'
group by CST_ID) t2
on t.CST_ID=t2.CST_ID
left join (select CST_ID,
sum(CORP_DEP_BAL) AS CORP_DEP_BAL,
sum(ACC_NUM) AS ACC_NUM,
sum(CRN_BAL) AS CRN_BAL,
sum(SMBSN_DEP_BAL) AS SMBSN_DEP_BAL,
sum(CPCT_HOLD_PD_NUM) AS CPCT_HOLD_PD_NUM
from INST_CST_DLY_INF where DATA_DT='2020-07-29' GROUP
BY CST_ID) t3
on t.CST_ID=t3.CST_ID
where t.ENTP_SZ_CD = '30'
AND t.CPCT_TPCD = '010000'
and t1.CRGLN>= 0
and t3.CPCT_HOLD_PD_NUM >=3
and t3.CORP_DEP_BAL >=1000000

```

```

SQL 2:
select t.CST_ID,
t1.CST_NM
from CST_BSC_INF_DPLT t
left join CST_CPRSV_INF t1
on t.CST_ID=t1.CST_ID
and t1.DATA_DT=date('2020-06-30')
and t1.DEP_BAL>=10000
and t1.CPCT_LNBAL>=10000
left join (select CST_ID from LNACC_INF
where DATA_DT=date('2020-06-20')
and CPCT_LNBAL>=10000
and LN_AADBAL>=10000
group by CST_ID) t2
on t.CST_ID=t2.CST_ID

```



```

left join (select CST_ID from INST_CST_DLY_INF
           where DATA_DT=date('2020-07-29')
             and CRN_BAL>=10000
             AND CORP_LNBAL>=10000
           group by CST_ID) t3
  on t.CST_ID=t3.CST_ID
 WHERE t1.CST_ID is not null
    and t2.CST_ID is not null
    and t3.CST_ID is not null

```

5.5 Test results

Result comparison:

SQL	Row Table	Column Table
SQL 1	5.82 sec	0.86 sec
SQL 2	2.68 sec	2.76 sec

Result analysis:

It can be found through result comparison that a quick query speed can be kept in the column table by SQL1, and there is no difference in SQL2. By comparing SQL statements, we can find that query performance of column table is better when querying among many columns, and query performance of row table is outstanding when complicatedly querying among less columns due to limit of its index number. So, in different business cases, different table creating methods can be flexibly selected for optimization.

As Boray distributed memory database is based on the advantage of memory, we can find that two query methods are very fast and give response in seconds. Therefore, time difference between two tables is jointly determined by their respective strengths and weaknesses, as well as the volume of the data.

6、TPCH Test

Common POC test is based on TPC Benchmark H (TPC-H). TPCH

is a decision support criteria and composed of a series of altered query and parallel data oriented to business application. Query selected in the benchmark and data that constitutes the database are widely representative in business and easy to implement, and support the analysis of large amounts of data, the execution of highly complicated query, and the answering of key and frequently answered business questions.

6.1 Data generation

TPCH data generation can be conducted by generating the script tpch-tool.3.0.tar file from specific TPCH data. The file can be downloaded from the file server (download address: http://192.168.10.6:8080/sales_presales/%e4%ba%a7%e5%93%81/%e4%ba%a7%e5%93%81%e5%ae%89%e8%a3%85%e6%96%87%e6%a1%a3/TPCH%e6%b5%8b%e8%af%95/%e5%ae%89%e8%a3%85%e4%bb%8b%e8%b4%a8/). After the compressed package is uploaded to the virtual machine and decompressed, it can be used. As data generation needs a large space, it is recommended to decompress the file to a directory with a large disk capacity. In the following, we take the data directory as an example, which enters the file directory dbgen after decompression:

```
[root@node1~]# cd /data
[root@node1 data]# tar -xvf tpch-tool.3.0.tar
[root@node1 data]# cd /tpch-tools/dbgen
```

Now, we provide two data generation modes, i.e. common mode and running in the background. If the amount of data generated is too large, it's recommended to take the second mode to prevent disconnection with the server due to computer sleep. Generation of 100GB data is taken as an example below:

```
[root@node1 dbgen]# ./dbgen -vf -s 100
[root@node1 dbgen]# nohup ./dbgen -vf -s 100 &
```

After data generation, eight text files with the suffix .tbl will appear in the file directory to represent eight tables during TPCH test. Then, it's necessary to transform it into CSV files and store in the specified path:

```
[root@node1 dbgen]# rename .tbl .csv *.tbl
[root@node1 dbgen]# mv *.csv /data/tpch100g //create data storage file in advance
```

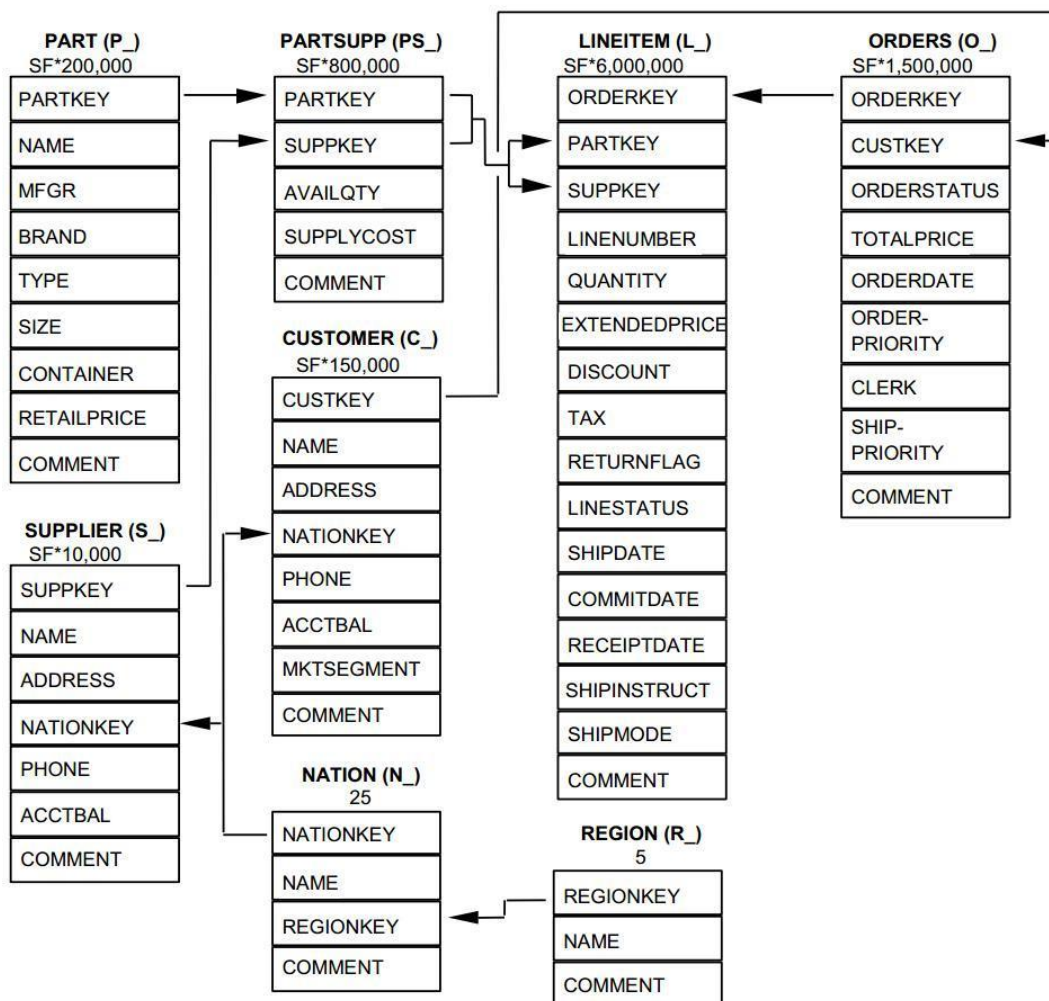
6.2 Table creating statements

Name of all tables involved during TPCH test and table relationship are

shown below:

No.	Table Name	No.	Table Name
1	customer	5	part
2	lineitem	6	partsupp
3	nation	7	region
4	orders	8	supplier

Figure 2: The TPC-H Schema



During TPC-H, as test data is large in most cases, we create column table for large table and replication table for small table. Table creating statements are shown below:

```

rapids > CREATE TABLE `customer` (
  `c_custkey` int(11) NOT NULL,
  `c_name` varchar(25) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
  `c_address` varchar(40) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,

```

```

`c_nationkey` int(11) NOT NULL,
`c_phone` char(15) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`c_acctbal` decimal(15,2) NOT NULL,
`c_mktsegment` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
`c_comment` varchar(117) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
UNIQUE KEY pk (`c_custkey`) UNENFORCED RELY,
SHARD KEY (`c_custkey`) USING CLUSTERED COLUMNSTORE
);

```

```

rapids >CREATE TABLE `lineitem` (
`l_orderkey` bigint(11) NOT NULL,
`l_partkey` int(11) NOT NULL,
`l_suppkey` int(11) NOT NULL,
`l_linenumber` int(11) 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) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`l_linestatus` char(1) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`l_shipdate` date NOT NULL,
`l_commitdate` date NOT NULL,
`l_receiptdate` date NOT NULL,
`l_shipinstruct` char(25) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
`l_shipmode` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`l_comment` varchar(44) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
UNIQUE KEY pk (`l_orderkey`, `l_linenumber`) UNENFORCED RELY,
SHARD KEY (`l_orderkey`) USING CLUSTERED COLUMNSTORE
);

```

```

rapids >CREATE reference TABLE nation (
n_nationkeyint(11) NOT NULL,
n_namevarchar(25) NOT NULL,
n_l_regionkeyint(11) NOT NULL,
n_commentvarchar(152) NOT NULL,
PRIMARY KEY (n_nationkey) );

```

```

rapids >CREATE TABLE `orders` (
`o_orderkey` bigint(11) NOT NULL,
`o_custkey` int(11) NOT NULL,
`o_orderstatus` char(1) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
`o_totalprice` decimal(15,2) NOT NULL,
`o_orderdate` date NOT NULL,
`o_orderpriority` char(15) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
`o_clerk` char(15) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`o_shippriority` int(11) NOT NULL,

```

```

`o_comment` varchar(79) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
UNIQUE KEY pk (`o_orderkey`) UNENFORCED RELY,
SHARD KEY (`o_orderkey`) USING CLUSTERED COLUMNSTORE
);

```

```

rapids >CREATE TABLE `part` (
`p_partkey` int(11) NOT NULL,
`p_name` varchar(55) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`p_mfgr` char(25) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`p_brand` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`p_type` varchar(25) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`p_size` int(11) NOT NULL,
`p_container` char(10) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`p_retailprice` decimal(15,2) NOT NULL,
`p_comment` varchar(23) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
UNIQUE KEY pk (`p_partkey`) UNENFORCED RELY,
SHARD KEY (`p_partkey`) USING CLUSTERED COLUMNSTORE
);

```

```

rapids >CREATE TABLE `partsupp` (
`ps_partkey` int(11) NOT NULL,
`ps_suppkey` int(11) NOT NULL,
`ps_availqty` int(11) NOT NULL,
`ps_supplycost` decimal(15,2) NOT NULL,
`ps_comment` varchar(199) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
UNIQUE KEY pk (`ps_partkey`, `ps_suppkey`) UNENFORCED RELY,
SHARD KEY(`ps_partkey`),
KEY (`ps_partkey`, `ps_suppkey`) USING CLUSTERED COLUMNSTORE
);

```

```

rapids >CREATE reference TABLE region (
r_regionkeyint(11) NOT NULL,
r_namevarchar(25) NOT NULL,
r_commentvarchar(152) NOT NULL,
PRIMARY KEY (r_regionkey)
);

```

```

rapids >CREATE TABLE `supplier` (
`s_suppkey` int(11) NOT NULL,
`s_name` char(25) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`s_address` varchar(40) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
`s_nationkey` int(11) NOT NULL,
`s_phone` char(15) CHARACTER SET utf8 COLLATE utf8_general_ci NOT NULL,
`s_acctbal` decimal(15,2) NOT NULL,
`s_comment` varchar(101) CHARACTER SET utf8 COLLATE utf8_general_ci NOT
NULL,
UNIQUE KEY pk (`s_suppkey`) UNENFORCED RELY,

```

```
SHARD KEY ('s_supplekey') USING CLUSTERED COLUMNSTORE
);
```

6.3 Data import

After table creation, it comes to data import. We can use connector created previously for data import:

```
rapids > use connector BORAYDATA;
```

Please note that file path can't be incorrect and please wait patiently during import. In case of cleared environment network or poor disk reading and writing, it may take a long time for import, rather than crash.

```
rapids > load data infile '/data/tpch100g/region.csv' into table region fields terminated by '|'
lines terminated by '\n';
```

6.4 Data query

All query statements (22 SQL in total, please refer to query.txt file) involved during TPCB test and SQL statements are detailed below:

Q1: Query of price statistics report

Q1 statement is a pricing summary report for querying lineitem. It queries the time period from the lineitem of the single table, and gives statistics of paid goods and delivered goods and other goods, including billing of business volume, delivery, discount, tax and average price etc.

Q1 statement is characterized by single table query operation with coexistence of grouping, sorting and aggregation. The query may cause reading of 95%~97% rows of data on the table.

Q1 query statement is as follows:

```
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<= '1998-12-01 00:00:00' and l_shipdate>= '1998-09-01 00:00:00' group
by l_returnflag, l_linestatus order by l_returnflag, l_linestatus limit 1;
```

Q2: Query of suppliers with the minimum cost

Q2 statement can be used to query the supplier with the minimum cost. In a given region, a given part (a part of a certain type and size) can be ordered from the supplier who can supply it at the lowest price.

Q2 statement is characterized by multi-table query operation with coexistence of sorting, aggregation and subquery. The query statement does not syntactically limit the number of tuples to return. But, the TPC-H standard stipulates that only the first 100 rows (usually depending on the application program) are returned as query result.

Q2 query statement is as follows:

```
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 = 15 and p_type like '%brass' 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 1;
```

Q3: Query of shipping priority

Q3 statement can be used to query the order not delivered and with revenue ranking the top 10. It specifies the shipping priority (orders are sorted in descending order of the revenue) of the order with the largest revenue among orders not shipped before the specified date and the potential revenues (potential revenues are the sum of $l_extendedprice * (1-l_discount)$).

Q3 statement is characterized by three-table query with coexistence of grouping, sorting and aggregation. The query statement does not syntactically limit the number of tuples to return. But, the TPC-H standard stipulates that only the first 10 rows (usually depending on the application program) are returned as query result.

Q3 query statement is as follows:

```
select l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue, o_orderdate, o_shippriority
from customer, orders, lineitem where c_mktsegment = 'building' and c_custkey = o_custkey and
l_orderkey = o_orderkey and o_orderdate < '1995-03-15' and l_shipdate > '1995-03-15' group by
l_orderkey, o_orderdate, o_shippriority order by revenue desc, o_orderdate limit 1;
```

Q4: Query of order priority

Q4 statement can be used to query the statistics value of order priority. It calculates the quantity of orders in the specified three months and at least includes one row in each order which will be received by the customer after the submission date.

Q4 statement is characterized by single table query operation with coexistence of grouping, sorting, aggregation and subquery. Subquery is relevant subquery.

Q4 query statement is as follows:

```
select o_orderpriority, count(*) as order_count from orders where o_orderdate>= '1993-07-01' and o_orderdate<'1993-10-01' and exists ( select * from lineitem where l_orderkey = o_orderkey and l_commitdate<l_receiptdate ) group by o_orderpriority order by o_orderpriority limit 1;
```

Q5: Query of revenue created by a supplier in an area for the Company

Q5 statement can be used to query the statistical revenue (calculated according to sum (l_extendedprice * (1 -l_discount))) obtained from part supplier in a certain area. It can be used to determine whether a local distribution center needs to be established in the given area.

Q5 statement is characterized by multi-table join query operation with coexistence of grouping, sorting, aggregation and subquery.

Q5 query statement is as follows:

```
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 = 'asia' and o_orderdate>= '1994-01-01' and o_orderdate<'1995-01-01' group by n_name order by revenue desc limit 1;
```

Q6: Query of forecast revenue change

Q6 statement can be used to query incremental revenue brought about by changing the discount in a given year. It's a typical "what-if" judgement and can be used to find way to increase the revenue. Query of forecast revenue change considers all delivered orders at a discount between "DISCOUNT-0.01" and "DISCOUNT+0.01" in a given year, and can be used to find the amount of increase in total revenue after elimination of discount of orders with l_quantity less than quantity.

Q6 statement is characterized by single table query operation with aggregation. Query statement includes BETWEEN-AND operator and some database can be used to optimize BETWEEN-AND.

Q6 query statement is as follows:


```
select sum(l_extendedprice*l_discount) as revenue from lineitem where l_shipdate>=
'1994-01-01' and l_shipdate< '1995-01-01' and l_discount between 0.06 - 0.01 and 0.06 + 0.01 and
l_quantity< 24 limit 1;
```

Q7: Query of shipping profits

Q7 statement can be used to query the sales profits obtained between nation of the supplier and nation for selling goods. It can be used to determine the volume of shipped goods between two nations, so as to help negotiation about shipping contract.

Q7 statement is characterized by multi-table query with coexistence of grouping, sorting, aggregation and subquery. Parent query of subquery is a relatively simple subquery without other query objects.

Q7 query statement is as follows:

```
select supp_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 = 'france' and n2.n_name =
'germany') or (n1.n_name = 'germany' and n2.n_name = 'france')) and l_shipdate>= '1995-01-01'
and l_shipdate<= '1996-12-31') as shipping group by supp_nation, cust_nation, l_year order by
supp_nation, cust_nation, l_year limit 1;
```

Q8: Query of market share held by the nation

Q8 statement can be used to query the change of market share of a given part in a certain region of a certain nation in the past two years.

Q8 statement is characterized by query with coexistence of grouping, sorting, aggregation and subquery. Parent query of subquery is a relatively simple subquery without other query objects. But, subquery is a multi-table join query.

Q8 query statement is as follows:

```
select o_year, sum(case when nation = 'brazil' 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 = 'america' and  
s_nationkey = n2.n_nationkey and o_orderdate >= '1995-01-01' and o_orderdate <=  
'1996-12-31' and p_type = 'economy anodized steel' ) as all_nations group by o_year  
order by o_year;
```

Q9: Query of profit estimation by product type

Q9 statement can be used to query the annual total profits of all parts ordered in each nation in each year.

Q9 statement is characterized by query with coexistence of grouping, sorting, aggregation and subquery. Parent query of subquery is a relatively simple subquery without other query objects. But, subquery is a multi-table join query. LINK operator is used during subquery. Some query optimizer can't be used to optimize the LINK operator.

Q9 query statement is as follows:

```
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 '%green%' ) as profit group by nation,  
o_year order by nation, o_year desc limit 1;
```

Q10: Query of shipping problems

Q10 statement can be used to query the customers with shipping problems in three months from a certain time and accrued losses in each country.

Q10 statement is characterized by multi-table join query with coexistence of grouping, sorting and aggregation. The query statement does not syntactically limit the number of tuples to return. But, the TPC-H standard stipulates that only the first 10 rows (usually depending on the application program) are returned as query result.

Q10 query statement is as follows:

```
select c_custkey, c_name, sum(l_extendedprice * (1 - l_discount)) as revenue, 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 >= '1993-10-01' and o_orderdate <
```

```
'1994-01-01' 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 1;
```

Q11: Query of inventory value

Q11 statement can be used to query the value of parts in stock supplied by a certain nation.

Q11 statement is characterized by multi-table join query with coexistence of grouping, sorting, aggregation and subquery. Subquery is included in the HAVING condition of grouping.

Q11 query statement is as follows:

```
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 = 'germany' group by ps_partkey having sum(ps_supplycost * ps_availqty) > ( select sum(ps_supplycost * ps_availqty) * 0.0000010000 from partsupp, supplier, nation where ps_suppkey = s_suppkey and s_nationkey = n_nationkey and n_name = 'germany' ) order by value desc limit 1;
```

Q12: Query of shipping mode and order priority

Q12 statement can be used to query the obtained shipping mode and order priority. It can be used to help decide whether selection of cheap shipping mode can cause customers to receive the goods after the contract date, thus leaving negative influence on emergency priority order.

Q12 statement is characterized by two table join query with coexistence of grouping, sorting and aggregation.

Q12 query statement is as follows:

```
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 ('mail', 'ship') and l_commitdate < l_receiptdate and l_shipdate < l_commitdate and l_receiptdate >= '1994-01-01 00:00:00' and l_receiptdate < '1995-01-01 00:00:00' group by l_shipmode order by l_shipmode;
```

Q13: Query of order quantity from customers

Q13 statement can be used to query the quantity of orders from the customers, including customers without order records in the past and at present.

Q13 statement is characterized by query with coexistence of grouping, sorting, aggregation, subquery and left outer join.

Q13 query statement is as follows:

```
select c_count, count(*) as custdist from ( select c_custkey, count(o_orderkey) as c_count from
customer left outer join orders on c_custkey = o_custkey and o_comment not like
'%special%requests%' group by c_custkey ) as c_orders group by c_count order by custdist desc,
c_count desc limit 1;
```

Q14: Query of promotion effect

Q14 statement can be used to query the percentage among monthly revenue from parts promotion, so as to monitor the market reaction brought by promotion.

Q14 statement is characterized by query with coexistence of grouping, sorting, aggregation, subquery and left outer join..

Q14 query statement is as follows:

```
select 100.00 * sum(case when p_type like 'promo%' 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>= '1995-09-01' and l_shipdate< '1995-10-01';
```

Q15: Query of top suppliers

Q15 statement can be used to query the information of supplier (ranked first) with the largest contribution to the total revenues in a time, so as to decide which first-class suppliers can be granted with rewards, more orders, specific certification or encouragement.

Q15 statement is characterized by join operation of common table and view with coexistence of sub-sorting, aggregation and aggregation subquery.

Q15 query statement is as follows:

```
with revenue0 (supplier_no, total_revenue) as (select l_suppkey, sum(l_extendedprice * (1 -
l_discount)) from lineitem where l_shipdate>= '1996-01-01'and l_shipdate< '1996-01-01' +
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;
```

Q16: Query of part/supplier relationship

Q16 statement can be used to query the quantity of suppliers that can supply parts at

the specified contribution conditions, so as to determine whether there are sufficient suppliers when there is large order quantity and urgent task.

Q16 statement is characterized by two-table join operation with coexistence of grouping, sorting, aggregation, weight adjusting, and NOT IN subquery.

Q16 query statement is as follows:

```
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#45' and p_type not like 'medium polished%'
and p_size in (49,14,23,45,19,3,36,9) 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 limit 1;
```

Q17: Query of income from small orders

Q17 statement can be used to query small batch orders less than 20% of average supply amount. For parts with specific trademark and package type, it can be used to determine the average number of items (past and pending) of such order parts among all orders in a seven-year database, and determine the average annual loss if orders of parts less than 20% of the average will not be accepted. So, this query can be used to get the loss of average annual revenue if without small orders (as management fees can be reduced for shipping of large volume of goods).

Q17 statement is characterized by two-table join operation with coexistence of aggregation and aggregation subquery.

Q17 query statement is as follows:

```
select sum(l_extendedprice) / 7.0 as avg_yearly from lineitem, part where p_partkey = l_partkey
and p_brand = 'brand#23' and p_container = 'med box' and l_quantity = ( select 0.2 *
avg(l_quantity) from lineitem where l_partkey = p_partkey );
```

Q18: Query of customers with large orders

Q18 statement can be used to query the information of suppliers with supply quantity above the specified quantity. It can be used to determine whether there are sufficient suppliers when there is large order quantity and urgent task.

Q18 statement is characterized by three-table join operation with coexistence of grouping, sorting, aggregation and IN subquery. The query statement does not syntactically limit the number of tuples to return. But, the TPC-H standard stipulates

that only the first 100 rows (usually depending on the application program) are returned as query result.

Q18 query statement is as follows:

```
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) > 300 ) 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 1;
```

Q19: Query of discount revenue

Q19 statement can be used to query the total discount revenues from all orders of three types of parts transported by air or by hand. Parts are selected by considering its brand, package and size. This query is an example of formatting code generated by data mining tools.

Q19 statement is characterized by three-table join operation with coexistence of grouping, sorting, aggregation and IN subquery.

Q19 query statement is as follows:

```
select sum(l_extendedprice * (1 - l_discount) ) as revenue from lineitem, part where ( p_partkey =
l_partkey and p_brand = 'brand#12' and p_container in ( 'sm case', 'm box', 'sm pack', 'sm pkg' )
and l_quantity >= 1 and l_quantity <= 1 + 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#23' and p_container in ('med bag', 'med box', 'med pkg', 'med pack') and l_quantity >= 10
and l_quantity <= 10 + 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#34' and
p_container in ( 'lg case', 'lg box', 'lg pack', 'lg pkg' ) and l_quantity >= 20 and l_quantity <= 20 + 10
and p_size between 1 and 15 and l_shipmode in ('air', 'air reg') and l_shipinstruct = 'deliver in
person' );
```

Q20: Query of supplier competitiveness

Q20 statement can be used to query the supplier from the given nation that can provide a more competitive price for a part in a given year. The so-called supplier with more competitiveness refers to suppliers with surplus parts. That quantity of parts is more than 50% of a part received by a given nation from the supplier in a year means surplus.

Q20 statement is characterized by two-table join operation with coexistence of sorting, aggregation, IN subquery and common subquery.

Q20 query statement is as follows:

```
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 'forest%' ) and
ps_availqty > ( select 0.5 * sum(l_quantity) from lineitem where l_partkey = ps_partkey and
l_suppkey = ps_suppkey and l_shipdate >= '1994-01-01' and l_shipdate < '1995-01-01' ) ) and
s_nationkey = n_nationkey and n_name = 'canada' order by s_name limit 1;
```

Q21: Query of suppliers failed to deliver on time

Q21 statement can be used to query the suppliers failing to deliver on time.

Q21 statement is characterized by four-table join operation with coexistence of grouping, sorting, aggregation, EXISTS subquery, and NOT EXISTS subquery. The query statement does not syntactically limit the number of tuples to return. But, the TPC-H standard stipulates that only the first 100 rows (usually depending on the application program) are returned as query result.

Q21 query statement is as follows:

```
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 = 'saudi Arabia' group by s_name order by numwait desc,
s_name limit 1;
```

Q22: Query of global sales opportunities

Q22 statement can be used to query geographical distribution of possible purchase by customers. It can be used to calculate the number of consumers in a given nation, having a more positive attitude than the average, and without placing seven-year orders. It can reflect the attitude of common consumers, i.e. purchase intention.

Q22 statement is characterized by four-table join operation with coexistence of grouping, sorting, aggregation, EXISTS subquery and NOT EXISTS subquery.

Q22 query statement is as follows:

```
select entrycode, count(*) as numcust, sum(c_acctbal) as totacctbal from ( select
substring(c_phone from 1 for 2) as entrycode, 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 entrycode order by entrycode limit 1;
```

6.5 Query script

All 22 SQL statements are written into a txt file. Let's take query.txt as an example, it's necessary to create an empty result.txt file and place them in the directory of /data/rdp/current jointly, and then prepare a script file:

```
[root@node1current]# vi test.sh
#!/bin/bash
export RDP_USERNAME=rapids
export RDP_PASSWORD=rapids
time echo "run /opt/rdp/current/query.txt;" | ./rapids-shell.sh 2>&1 | tee result.txt
```

The script can be executed after the file is saved and granted with privileges, and results will be recorded in the result.txt file:

```
[root@node1current]#chmod 777 test.sh
[root@node1current]# ./test.sh
```

7、 POC Test Optimization

Apart from pressure test based on TPCH, the customer may require onsite test or simulation test, and provide table structure and business logic. In this condition, it should be tested flexibly according to machine configuration and data volume. The following describes the database table structure optimization plan and problem solving method that may be used during test.

7.1 Data partition

As distributed database, database can be divided into several partitions during database creation by RapidsDB. The number of partitions can be a value automatically calculated according to the current scale of the cluster. The user can use the command of show partitions to see the current partition conditions. The user can also add partition=n after the statement during database creation to specify the number of partitions of the database. When the cluster scale is large and network link is good, the number of partitions should be larger as recommended.


```
rapids > create database test partitions = 100;
```

7.2 Shard key

Each table should have a shard key or partition index, i.e. shard key, which is similar with table index of common table and may include rows in any quantity. When create table is operated by the user to create a table, a shard key can be specified for the table. If no shard key is specified, default shard key should be used.

7.2.1 Type of shard key

The following shows the type of shard key:

- Default shard key

No specific shard key, or specify an empty shard key:

```
rapids > create table t1(a INT,b INT);
rapids > create table t1(a INT,b INT, shard key ());
```

In most conditions, keyless partition can cause even distribution of rows in the partitions.

- Primary key as shard key

If a table with primary key and without explicit shard key is created by the user, primary key will be used as shard key by default.

```
rapids > CREATE TABLE clicks (
    click_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    page_id INT,
    ts TIMESTAMP );
```

- Non-unique shard key

```
rapids > CREATE TABLE clicks (
    click_id BIGINT AUTO_INCREMENT,
    user_id INT,
    page_id INT,
    ts TIMESTAMP,
```

```
SHARD KEY (user_id),
PRIMARY KEY (click_id, user_id );
```

Please note, even if `click_id` is unique, `user_id` should also be included in the primary key.

- Common shard key:

```
rapids > CREATE TABLE clicks2 (
    click_id BIGINT AUTO_INCREMENT,
    user_id INT,
    page_id INT,
    ts TIMESTAMP,
    SHARD KEY (user_id);
```

7.2.2 Functions of shard key

Shard key can be used to determine which column can be used as benchmark and basis for importing data and stored in our partition. The column selected as shard key will directly affect the distribution of data. Rows with same shard key will be stored in the same partition. So, it's of great importance to efficiently select which column or columns can be used as shard key.

For example: shard key in the following table only include the column `first`, and all persons with same name will be stored in the same partition (four partitions in total):

people

people_1			people_2			people_3			people_4		
user	first	last	user	first	last	user	first	last	user	first	last
areece	alex	reece	jdoe	john	doe	ndrew	nancy	drew	amons	amanda	monson
anick	alex	nick	jsmith	john	smith						
amace	alex	mace									
thom	tom	holmes									

Shard key should be specified during table creation. Once created, shard key of the table can't be altered at will. During data partition, two competitive factors should be considered:

- Data are distributed averagely across partitions;

- Data on the column screened or connected frequently are partitioned.

Firstly, data distributed evenly can facilitate planning of the volume, as the resources will be used by the system uniformly. If data skew or is distributed unevenly, its query performance may be affected to a certain degree, as there may be some partitions with low speed. Query speed of multiple partitions isn't quicker than lowest operation speed of the partition.

Then, if the optimizer can accurately know which partition is used, query performance can be improved significantly, and the resources used will also be reduced. If query filter matches the keyword of shard key, RapidsDB only needs to contain one partition, so as to greatly reduce the usage of customer resources at a high concurrent workload. Similarly, multiple tables with same shard key will be connected locally on the leaf partition, rather than re-distribution of data in the cluster, thus causing quicker connection with high concurrent performance. Tables with different shard keys can be connected or distributed connection may be much slower and consume more resources.

For example: the following query is considered as a “single partition”, as screen clause (where first = 'john') includes the column of shard key, and aggregator only needs to have dialogue with a partition to obtain data:

people

people_1			people_2			people_3			people_4		
user	first	last	user	first	last	user	first	last	user	first	last
areece	alex	reece	jdoe	john	doe	ndrew	nancy	drew	amons	amanda	monson
anick	alex	nick	jsmith	john	smith						
amace	alex	mace									
thom	tom	holmes									

In case of data deviation, these two problems generally have conflicts. For example, `first` may be an awful shard key in the table, as some names are more common than other names. In this condition, it's more important to distribute data evenly, so as to not deplete the capacity by the cluster. `User` may be a better choice, as it will be distributed more evenly and may be the filter targeted on table query. If the user hopes that a table can get a truly uniform partition, one can give partition on an auto incrementing column.

7.2.3 Selection of shard key

Faced with selection of shard key, you may refer to the followings:

Q: Does the table have a primary key?

A: It's necessary to ensure that there is a shard key, which is a sub-cluster of primary key.

Q: Do you often add a specific set of columns during query? For example: `where users.id = action.user_id and users.country = action.country`

A: Please try to make column of shard key a subset of columns for connection.

Q: Do you often screen a specific column during query? For example: `where user_id = 17 and date = '2007-06-14'`

A: Please try to make column of Shard Key a subset of screening column.

Q: Do users need highly concurrent queries?

A: Select a Shard Key that allows such queries to be a single partition.

Q: Does user data deviate from the currently selected Shard Key?

A: Please try to add extra columns to the Shard Key to ensure even

distribution.

Q: Do users need to update or alter any fields of Shard Key?

A: Remove these fields from Shard Key.

Q: If you have a small table with little change, do you need to be on each node of the cluster to ensure local connection?

A: Use reference table rather than partition table.

7.3 Row table

Row storage is the default table storage format. Generally, the user will specify a shard key and one or more indexes for the row table. The shard key and index are optional and not mandatory. The user can also add a primary key that is mandatory to be unique.

7.3.1 Create row table

The following is the example of statements for creating a row table:

```
rapids > CREATE TABLE products (  
          ProductId INT,  
          Price INT,  
          dt DATETIME,  
          KEY (Price),  
          SHARD KEY (ProductId) );
```

All data in the row table will be written into the memory, so the memory capacity of the server is very important, and we need to reserve part of the memory space for cache storage and processing (about 20% - 30%). So when the machine memory is 256GB, we can only import 180GB data at most, so as to avoid performance degradation and full memory

The advantage of row table is that it can establish multiple indexes, and has outstanding performance in addition, deletion, alteration and

query. It is more suitable for multi-column random search, more frequent addition and deletion of tables, smaller data volume and business scenarios with complex logic.

7.3.2 Row table index

Row table index has two storage types: lock free skip column table index and lock free Hash Table. In these two cases, we use lock free data structure to optimize concurrent update performance of the table.

- By default, the index is stored as a skip table, which has similar functions and performance characteristics to B-trees in other databases. Skip table is a data structure optimized for ordered data, which stores rows in smaller and smaller ordered table sets. Queries can quickly find data by binary search using table at different sizes, and can quickly scan data scope by traversing the largest table. For multi-column index, the query filter must match the prefix of the index column list, so as to use the index.

```
rapids > CREATE TABLE products (  
          ProductId INT,  
          Price INT,  
          KEY (Price) using BTREE ,  
          SHARD KEY (ProductId) );
```

- Hash Table is a data structure optimized for fast query. It stores rows in a sparse bucket array. These buckets are indexed by hash functions on related columns. A query can quickly find the exact matching data by only checking the buckets identified by hash functions, but it cannot easily scan a subset of the table. For multi-column indexes, the query filter must match all index columns before using the indexes, which causes inflexibility. So we do not encourage the use of hash

indexes. They are only be used when there are clear demands and measurable benefits from specific data sets and workloads of the user.

```
rapids > CREATE TABLE products2 (
    ProductId INT,
    Price INT,
    KEY (Price) using HASH,
    SHARD KEY (ProductId) );
```

Another factor to consider when selecting an index is the cost incurred from addition of another index. For each additional index, extra memory will be used for additional data structures (about 40 bytes per row on average) and data insertion is slowed down slightly.

In each row table, there is at most one primary key or multiple primary keys. The scanning of primary key is usually faster than that of secondary key. For example, if data is inserted in the order of primary key, then for primary key, rows are inserted in memory order and the cache location of rows is better than that of secondary key.

7.3.3 Selection of table index

The following is a common problem during creation of row table index:

Q: For table `CREATE TABLE t(a INT, b INT, KEY (a, b))`, will the query of `SELECT SUM(a) FROM t WHERE b = 3` benefit from the index?

A: No, as the unique column `b` in the filter list isn't the prefix of button `(a, b)`, query can't benefit from the index. In case of `SELECT SUM(a) FROM t WHERE a = 3`, it's ok.

7.4 Column table

The column table has only one index, which is used by the column storage as a keyword to divide all rows into logic segments after sorting. These logic segments contain data of many rows. Data in the data

segments are stored in the data segment files on the disk, which contain the same fields of many rows. This realizes two important functions: one is to scan each column separately; in essence, it can and only can scan the columns required by query with high locality. The other is that column storage is very suitable for compressing data. For example, duplicate and similar data can be easily compressed together.

In addition, RapidsDB stores the metadata of each row segment in memory, including the minimum and maximum values of each column contained in the segment. The metadata is used to determine whether a segment may match the filter during query, which is called segment elimination.

7.4.1 Create column table

For example, we create the following column table:

```
rapids > CREATE TABLE products (
    ProductId INT,
    Color VARCHAR(10),
    Price INT,
    Quantity INT,
    KEY ('Price') USING CLUSTERED COLUMNSTORE );
```

15 rows of data are temporarily used for demonstration:

ProductId	Color	Price	Qty
1	Red	10	2
2	Red	20	2
3	Black	20	2
4	White	30	2
5	Red	20	2
6	Black	10	2
7	White	25	2
8	Red	30	2
9	Black	50	2
10	White	15	2
11	Red	5	2
12	Red	20	2
13	Black	35	2
14	White	30	2
15	Red	4	2

Row segment group 1 for sorting:

Price 4-15	ProductId 1-15	Color	Qty 2-2
4	15	Red×2	2×5
5	11	Black	
10×2	6	Red	
15	1	White	
	10		

Row segment group 2:

Price 20-25	ProductId 2-12	Color	Qty 2-2
20×4	3	Black	2×5
25	2	Red×3	
	5	White	
	12		
	7		

Row segment group 3:

Price 30-50	ProductId 4-14	Color	Qty 2-2
30×3	8	Red×3	2×5
35	4	White×2	
50	14	Black×2	
	13		
	9		

“×N means the value is repeated for N times”

In this example, we can see that there are three segments containing different ranges of Price column (4-15, 20-25 and 30-50). Each segment has the same number of rows and data organized by columns. For each column, the minimum and maximum values in the segment are stored as metadata.

The most important consideration of column table key is to add deletion quantity of the segment. Metadata of minimum/maximum values can be used to determine whether a segment matches the filter during query; if not, it's necessary to completely skip this segment and not check data.

A good key of column table is used to segment the data through filter, so as to add the optimization possibility. Segment deletion is very effective for query by screening key column of column table index, because row segments in each row segment group don't have overlapping segments. For example, in the above table, query statement `SELECT AVG(Price), AVG(Qty) FROM Products WHERE Price BETWEEN 1 AND 10;` will eliminate all segments except for the price in the row segment group of 4-15.

7.4.2 Selection of table index

The following are several common problems during creation of column table index:

Q: Is data always filtered by a column (for example, insertion timestamp or event type)

A: Make sure that all public columns for query are in the key columns of column table to improve segment elimination.

Q: Is data usually inserted in column order (for example, insertion timestamp)

A: It is best to put the column first in the key column of column table to minimize the workload required for background segment merging.

Q: Does one column of the user's key have a higher base than the other?

A: It is best to put the column with the minimum base first to increase the possibility of segment deletion affecting the later columns.

Q: Can the column storage key be different from the shard key?

A: Of course, they don't have any relationship. Usually, it's recommended to select shard key matching with other row storage table (such as event_id) to improve connection performance, and select uncorrelated column storage key matching with common filter (such as event_timestamp or event_type) in the table.

7.5 Unique key

The user may specify the cancelation of enforcement option on the unique constraint to disable enforcement of the constraint. This means that RapidsDB doesn't prevent users from inserting duplicate rows and doesn't ensure that the constraint is true. The unenforced unique

constraint is informative: query planners can use the unenforced unique constraint as a cue for selecting a better query plan.

The unenforced unique constraint is useful for column table, because column table does not support general unique constraints. In the column table, the unenforced unique constraint is logical indexes only, which has no physical storage.

In row table, the unenforced unique constraint is physically stored as a regular non-unique index.

“Dependency” and “non-dependency” options specify how the query planner uses the unenforced unique constraint:

- “Non-dependency” NORELY option is a default option, which specifies that query semantics should not depend on the unenforced unique constraint - whether the unenforced unique constraint is true or not, the query will return the correct results. Rapidsdb may use an unenforced unique constraint to inform statistical estimates.
- “Dependency” rely option specifies that RapidsDB can choose a query plan with the assumption that the unenforced unique constraint is true - only when the constraint is true, the query can ensure that the correct results are returned. If there are duplicate rows, incorrect results may be returned during query. Only when the user knows that the constraint is valid, for example, one process of the user application enforces the integrity of the constraint, this option can be used, just as that a dependent unenforced unique constraint can be used by RapidsDB to notify the statistical estimates and eliminate redundant connection.

Statement example

To specify the unique constraint as enforcement cancellation, please add the option of unenforced into the index declaration, for example:

```
UNIQUE KEY (col1, col2) UNENFORCED
```

Syntax of unenforced clause is:

```
UNENFORCED [RELY | NORELY]
```

If RELY or NORELY isn't specified, it should be NORELY by default.

Complete statements during table creation should be:

```
rapids > CREATE TABLE t (id INT,
                        KEY USING CLUSTERED COLUMNSTORE (id),
                        UNIQUE KEY (id) UNENFORCED RELY);
```

Note: column storage index table doesn't support unenforced unique constraint. It doesn't support addition, deletion or alteration of unenforced unique constraint on the column storage table.

7.6 Data skew

One important characteristic of distributed database is that the distribution of data is more or less evenly. In very few cases, data may be unbalanced. In this section, we mainly discuss how skew occurs, how to be detected and how to solve it

7.6.1 Understand data skew

The “asymmetry” herein refers to the imbalance of table data among partitions of the cluster. Although it is unlikely, a small amount of skew is inevitable and harmless. By default, RapidsDB can distribute data based on the hash value of the primary key. As the hash value is unique and evenly distributed, this will keep the minimum deviation.

Users can split tables by a set of columns rather than primary keys. For example, a URL table from a WEB domain can be defined as follows:

```
rapids > CREATE TABLE urls ( id BIGINT,
                            domain_id BIGINT,
                            path VARCHAR(8192),
                            first_seen INT UNSIGNED NOT NULL,
                            crawl count INT UNSIGNED NOT NULL, ...
                            SHARD KEY (domain_id));
```

As shown, domain_id is partitioned by database. Query for this table

will be compiled and run well. However, it is likely that the number of URLs for some domain names will be many times more than this. For example, the partition containing the “youtube.com” link will almost certainly have more rows than the average level, which will break the balance of the cluster. The bad luck partition containing “youtube.com” will not only store more data, but also assume more work during the selection, update, and deletion of query tasks as required. To achieve the best performance, users should choose a shard key that can minimize data asymmetry.

After a large amount of load data, recovery from restart, or deletion of large amounts of data, a partition may temporarily display more memory usage and memory skew than other partitions, because memory allocation and garbage collection may not take effect immediately between partitions. This is normal. As the system reaches a new stable state, it will self-correct as time goes on.

7.6.2 View data skew

The user may measure and test data skew with following methods:

- Method 1: check memory allocation

Variables, i.e. `maximum_table_memory` and `Alloc_table_memory` (memory used by the table) on each node are compared manually:

```
rapids > SHOW VARIABLES LIKE "maximum_%";

Variable_name  Value
-----
maximum_memory 762244
maximum_table_memory 686019

2 row(s) returned (0.00 sec)

rapids > SHOW STATUS EXTENDED LIKE "Alloc_table_memory";

Variable_name  Value
-----
Alloc_table_memory 46.807 MB

1 row(s) returned (0.00 sec)
```

If memory consumption on a particular node is significantly higher, skew will occur. If memory consumption is fairly even between leaves, skew will not exist.

- Method 2: find rows and memory usage in each partition

The built-in database contains metadata about tables, columns, indexes and partitions. In particular, each table partition contains a row, which can be used to check whether the number of rows in a given partition is more than the average:

```
rapids >SELECT
```

```

DATABASE_NAME,
TABLE_NAME,
ORDINAL AS PARTITION_ID,
ROWS,
MEMORY_USE
FROM INFORMATION_SCHEMA.TABLE_STATISTICS
WHERE TABLE_NAME = 'ta';
```

DATABASE_NAME	TABLE_NAME	PARTITION_ID	ROWS	MEMORY_USE
ccb ta	0	2084703	0	
ccb ta	1	2085621	0	
ccb ta	2	2085460	0	
ccb ta	3	2083063	0	
ccb ta	4	2084806	0	
ccb ta	5	2082339	0	
ccb ta	6	2083658	0	

RapidsDB distributes data according to the shard key specified in the table structure. If the shard key is not specified explicitly, the primary key will act as the default shard key. If the shard key splits on the low base sequence, the data will be accumulated on some nodes. To solve this problem, users should first export the data, modify the mode, and then reload the data. After data backup, one can delete the table and create a table of shard keys with high base. When the data is reloaded, RapidsDB will automatically partition the data according to the new shard key.

7.7 Leaf and aggregator

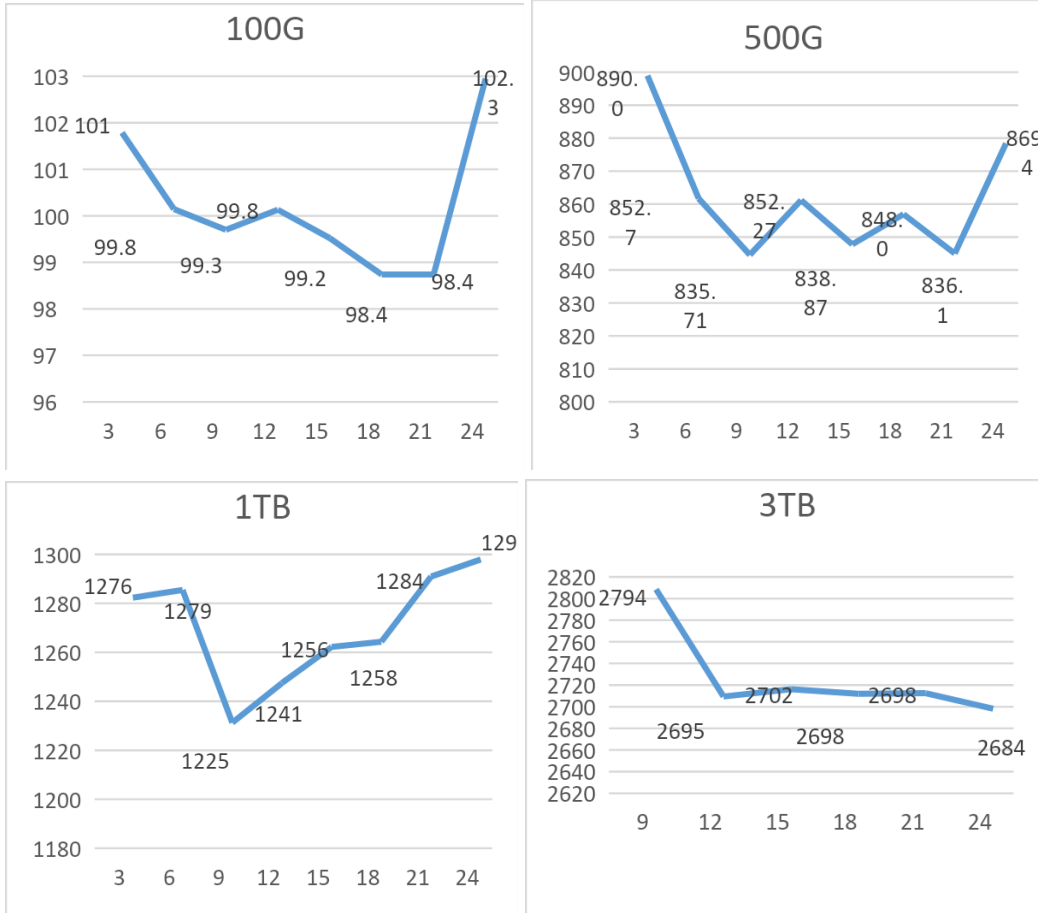
Leaf node is the node for storing subset of cluster data. Aggregator is a node for querying the route to middle leaves, aggregating the middle results, and sending the results back to the client. There are two kinds of aggregators: main aggregator and sub-aggregator. A cluster contains one main aggregator and zero or more sub-aggregators (depending on the query volume).

For minimum setting, RapidsDB cluster only has an aggregator (main aggregator) and a leaf. The user can add the aggregator, which can be used to read metadata from main aggregator and operate DML command on the leaf.

Storage size and performance of the cluster are determined by the

quantity of allocated aggregators and leaf nodes. For typical allocation, the proportion of leaf node to aggregator is 5:1, which can be improved if higher storage requirements are needed.

When network environment is gigabit lan, the curve graph of query time changing with the number of leaves is shown below:

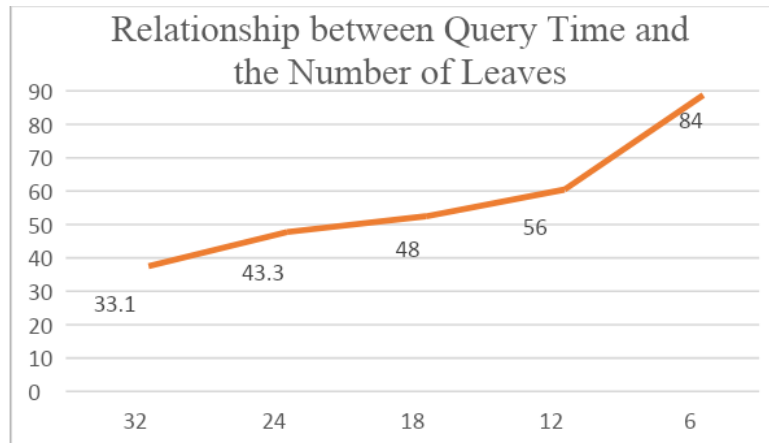


If there is too little leaf, the advantages of distributed mode can't be reflected and query speed is low.

If there is too many leaves, query speed is also affected due to network speed limit.

In the common environment of Gigabit lan, it's recommended to establish 5-7 leaves for each server.

When network environment is 10 gigabit lan, the curve graph of query time changing with the number of leaves is shown below:



In the 25G optical fiber network environment, as the network speed has been greatly improved, the data transmission and sharing rate between the master node and the slave node has been greatly improved. At this time, the increase in the number of leaves does not affect the query rate, but increases the query rate.

But it is worth noting that the number of leaves should be set according to the hardware configuration of the machine, which shouldn't be too much sometimes.

7.8 Code generation

One of main reason for excellent query performance of RapidsDB is its code generation system. Compared with traditional database execution model based on the interpreter, RapidsDB is embedded with an industrial compiler to generate efficient machine code, so as to achieve low-level optimization. This process can't be realized by query through interpreting only. By default, the query is interpreted firstly and then compiled asynchronously in the background, for future use. This speeds up the execution of long-time complex query, and provides an efficient query plan for future use.

As mentioned above, when RapidsDB encounters a given query shape for the first time, it will asynchronously optimize and compile the

query for future use, which can minimize the expenditure and depend on query complexity, rather than the size of data to be processed. Code generation process includes parameter extraction from query, and transformation of standard query into an intermediate representation of single storage database customized by the system. Subsequent requests with the same shape can reuse the plan for fast and consistent completion.

Code generation applies to query with all data operation languages (DML). In addition, RapidsDB produces code during CREATE TABLE and ALTER TABLE. Code generated from query of data definition language (DDL) can be used to reduce the compilation time of DML query and table checking in the future.

Allowed settings include the following modes:

1. llvm or compile: the query is compiled into machine code.
2. mbc or interpret: the query is explanatory, rather than compiled.
3. interpret_first: the query starts from interpretation and is dynamically switched to compilation during the first query. This mode helps improve the performance of ad hoc query. By default, interpretation first mode is turned on and can be used during production deployment.

This variable can also be achieved by adding OPTION (interpreter_mode = { interpret | compile | interpret_first}) at the end of the query. It's not operational on the node, because it is forwarded from the aggregator to the leaf node.

- When interpreter_mode is set as compile:

In this mode, RapidsDB can compile query shape during the first time it encounters the query; as shown below, query SELECT * FROM t

WHERE col = 1; it takes more time to complete the first operation; this is the result of the compilation overhead in the first step. It should also be noted that during the second and third queries, WHERE clause is different, but the shapes of two queries are the same. Therefore, the second query plan is reused by RapidsDB for the third query:

```
rapids > SELECT * FROM t WHERE col = 1;
Empty set (0.13 sec)
rapids > SELECT * FROM t WHERE col = 1;
Empty set (0.00 sec)

rapids > SELECT * FROM t WHERE col = 100000;
Empty set (0.00 sec)
```

- When `interpreter_mode` is set as `interpret_first`:

In this mode, RapidsDB can automatically interpret and compile the query shape during the first time it encounters the query shape. The query is operated in interpretation mode, until completion of compilation of query shape. In the following example, the speed for firstly querying `SELECT * FROM t WHERE col = 1;` is larger than the first query speed in the previous example:

```
rapids > SELECT * FROM t WHERE col = 1;
Empty set (0.02 sec)

rapids > SELECT * FROM t WHERE col = 1;
Empty set (0.00 sec)

rapids > SELECT * FROM t WHERE col = 100000;
Empty set (0.00 sec)
```

View current mode:

```
rapids> show variables like '%interpreter_mode%';
+-----+-----+
| Variable_name          | Value          |
+-----+-----+
| interpreter_mode       | INTERPRET_FIRST |
| interpreter_mode_sampling_threshold | 1000000       |
+-----+-----+
2 rows in set (0.00 sec)
```

Change current mode:

```
rapids> set interpreter_mode=compile;
Query OK, 0 rows affected (0.00 sec)
```

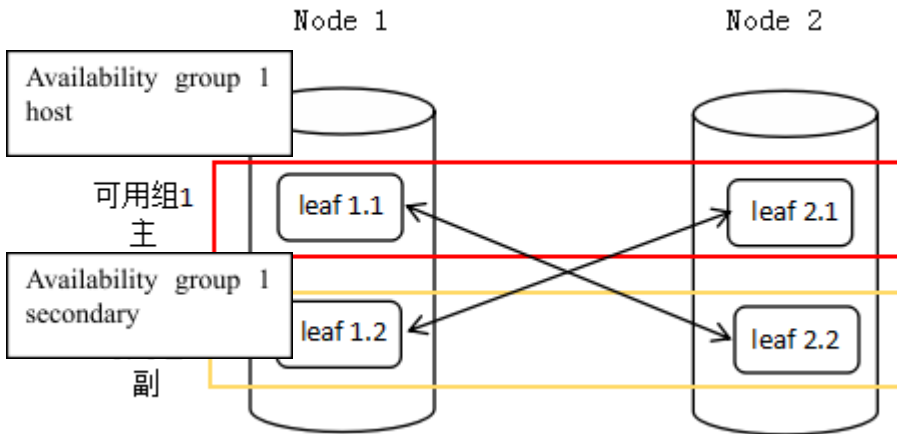
```

rapids> show variables like '%interpreter_mode%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| interpreter_mode       | LLVM  |
| interpreter_mode_sampling_threshold | 1000000 |
+-----+-----+
2 rows in set (0.00 sec)
    
```

8、 High availability test

Availability group is composed of a group of leaves that store redundant data to ensure high availability. Each availability group contains the copy of each partition in the system, some as primary partition and some as copy partition. At present, RapidsDB can support two availability groups at most. The user can use `redundancy_level` on the main aggregator to set the number of availability groups.

Partition placement mode of RapidsDB is paired mode, where each leaf of availability group has a corresponding paired node in another availability group. The same partition set is shared by this leaf and its paired leaves, but main partition is averagely distributed among them. In case of fault, copy partition of paired leaves of the leaf is automatically updated by RapidsDB.



There are three nodes at the cluster, and two leaf nodes at each node:

```

rapids > show leaves;
    
```

Host	Port	Availability_Group	Pair_Host	Pair_Port	State	Opened_Connections	Average_Roundtrip_Latency_ms	NodeId
192.168.0.88	3307	1	NULL	NULL	online	9	0.264	5
192.168.0.88	3308	1	NULL	NULL	online	9	0.218	7
192.168.0.89	3307	1	NULL	NULL	online	9	0.202	9
192.168.0.89	3308	1	NULL	NULL	online	9	0.192	11
192.168.0.93	3307	1	NULL	NULL	online	9	0.172	13

```
192.168.0.93 3308 1 NULL NULL online 9 0.149 15
6 row(s) returned (0.00 sec)
```

Firstly, we prepare the null leaf node, i.e. leaf node installed and without allocated group. Among above six leaf nodes, we remove three leaf nodes, which constitute three null leaf nodes:

```
rapids > remove leaf '192.168.0.88':3308;
rapids > remove leaf '192.168.0.89':3308;
rapids > remove leaf '192.168.0.93':3308;
```

Then, variable is set:

```
rapids > SET @@GLOBAL.redundancy_level = 2;
```

After setting, leaf node should be added to different groups (please remember the sequence for adding leaf node and don't pair leaf nodes on the same node. For example, leaf node at 88 nodes matches leaf node at 89 nodes):

```
rapids > ADD LEAF root@'192.168.0.89':3308 INTO GROUP 2;
rapids > ADD LEAF root@'192.168.0.93':3308 INTO GROUP 2;
rapids > ADD LEAF root@'192.168.0.88':3308 INTO GROUP 2;
```

We can view specific information through show leaves. Each leaf node has a corresponding backup leaf node. Then, we can create the table and database normally:

```
rapids > show leaves;
      Host          Port      Availability_GroupPair_HostPair_Port      State
Opened_ConnectionsAverage_Roundtrip_Latency_msNodeId
-----
192.168.0.88 3307 1 192.168.0.89 3308 online 9 0.264 5
192.168.0.89 3307 1 192.168.0.93 3308 online 9 0.202 9
192.168.0.93 3307 1 192.168.0.88 3308 online 9 0.172 13
192.168.0.89 3308 2 192.168.0.88 3307online 9 0.2223
192.168.0.93 3308 2 192.168.0.89 3307online 9 0.1142
192.168.0.88 3308 2 192.168.0.93 3307online 9 0.3618
6 row(s) returned (0.00 sec)
```

If you want to view partition of the host and the secondary, please enter show partitions on tpch; and be sure to enter the command of **rebalance partitions on tpch**; after data loading to balance the data. After server downtime and completion of cluster recovery, rebalance should be implemented. (note: tpch is the name of database)

9、 Concurrent Test

Concurrent performance can be tested through JMeter, which can also be used as a tool for making numbers randomly to satisfy the simulation test of customer business data. Download address is shown below. For installation package and specific jar package used during

configuration, please refer to the appendix:

http://192.168.10.6:8080/sales_presales/%e4%ba%a7%e5%93%81/%e4%ba%a7%e5%93%81%e5%ae%89%e8%a3%85%e6%96%87%e6%a1%a3/RapidsDB%e5%8e%8b%e5%8a%9b%e6%b5%8b%e8%af%95/%e5%ae%89%e8%a3%85%e4%bb%8b%e8%b4%a8/

The following describes the installation and use of JMeter.

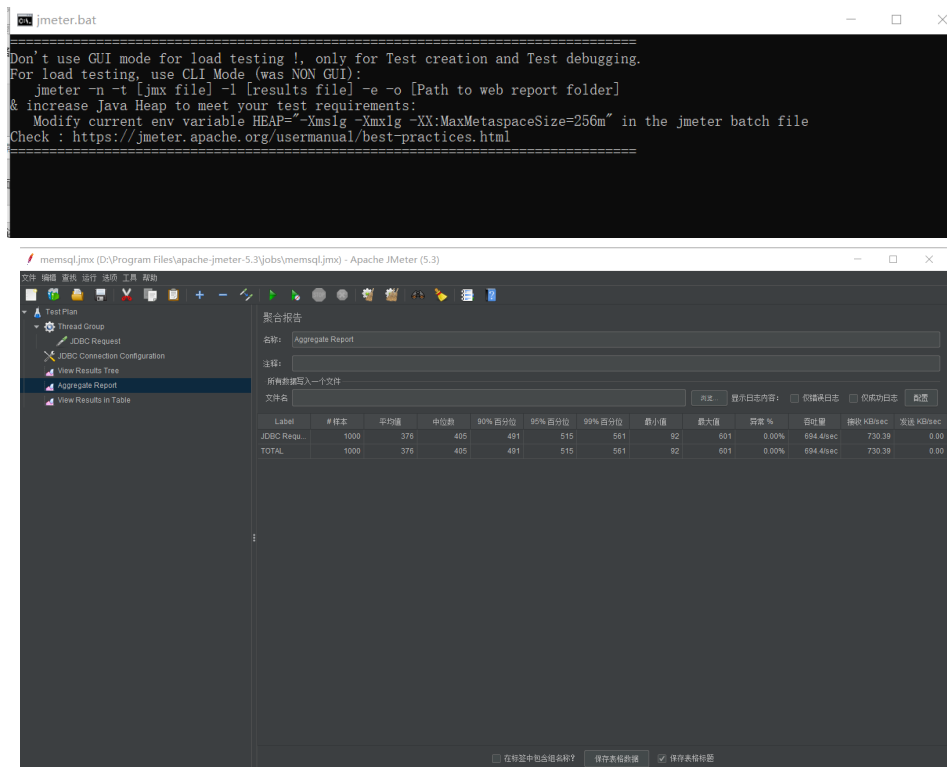
9.1 Window installation and use

Firstly, apache-jmeter-5.3.zip downloaded and installed well is unzipped to any non-Chinese disk directory. Then, link variables are configured:

Computer desktop----computer----attribute----advanced system ---advanced---
 system environment variable

Secondly, among user variables, we create the variable named “JMETER_HOME”, and variable value is E:\apache-jmeter-4.0 (i.e. the address of directory for unzipping files)

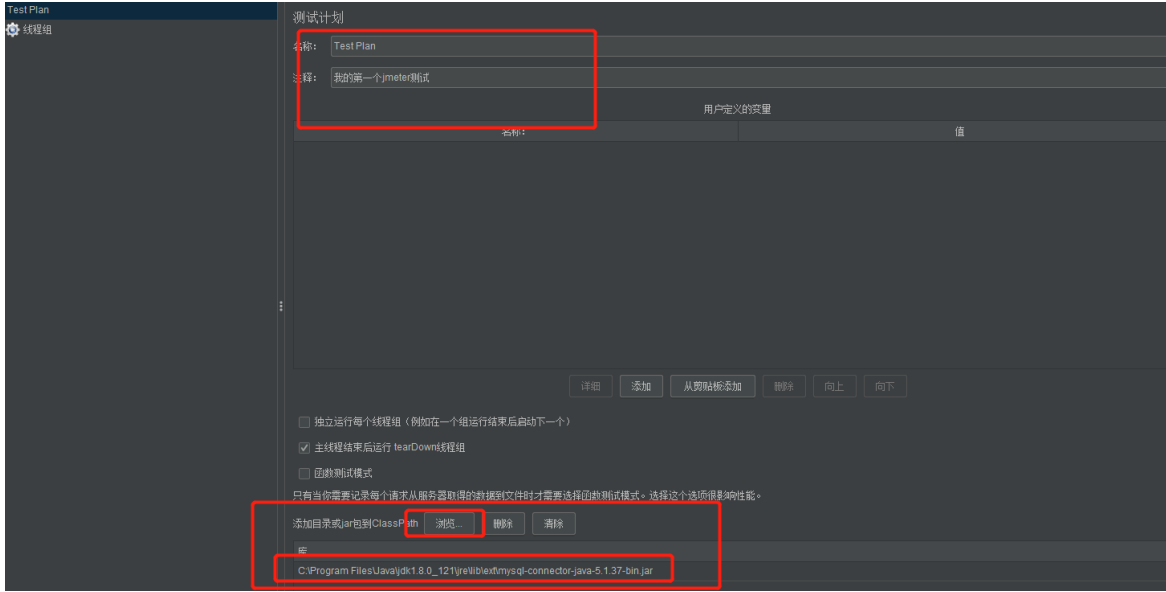
Finally, jmeter.bat in the directory of bin is operated to pop up two screens, i.e. command window and JMeter window, which means successful installation.



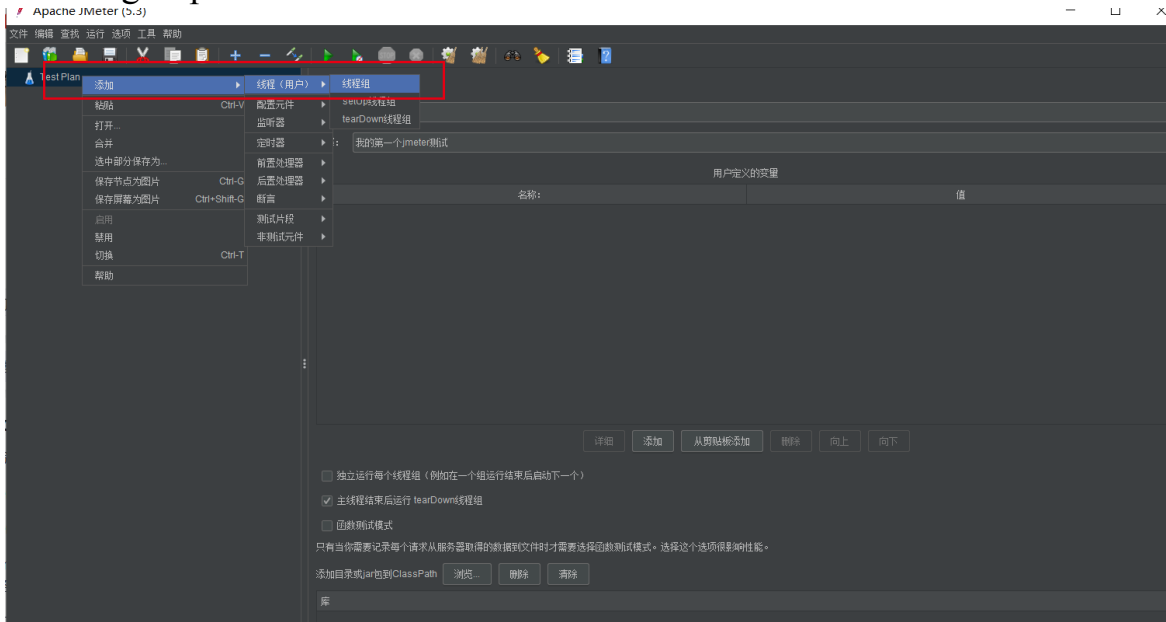
Step 1: Options-choose language- select simplified Chinese.

Step 2: create test plan, name TestPlan and add the driver. Among them, if RapidsDB needs to be tested, RapidsDBjdbc driver should be added, such as rapids-jdbc-4.0.jar. In case of rpdsq1 test, you may add

rpdsq.jar or directly add mysql driver package. The figure shows cases for adding mysql driver packages.



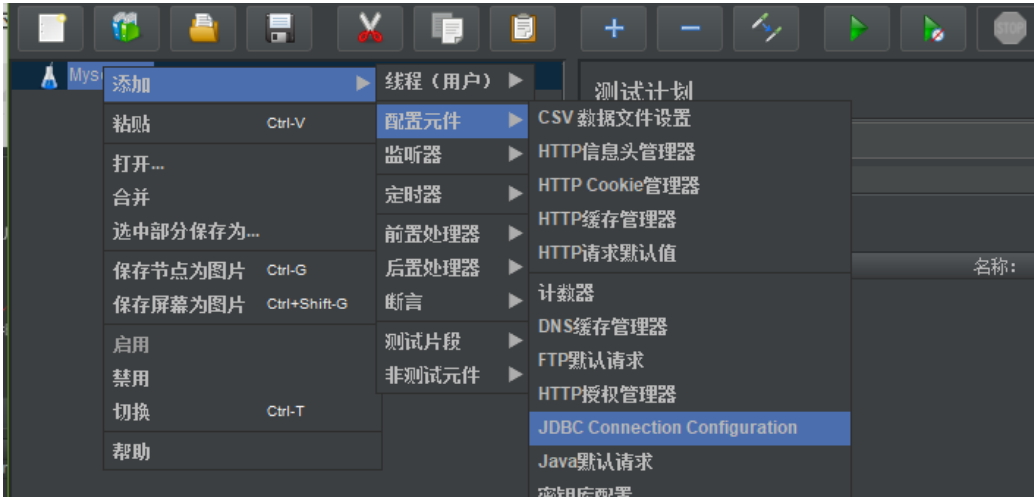
Step 3: right-click “TestPlan”, click “Add”“Thread”, and select a thread group.



Fill in the number of threads (number of concurrent threads), and cycle number:



Step 4: add JDBC Connection Configuration



Test configuration of connector named con among RapidsDB:

DatabasesURL: jdbc:rdp://192.168.20.42:4333/connector=conn

JDBC Driver class: com.rapidsdata.jdbcdriver.Driver

Username: RAPIDS

Password: rapids

Test configuration of database named test among rpdsq1:

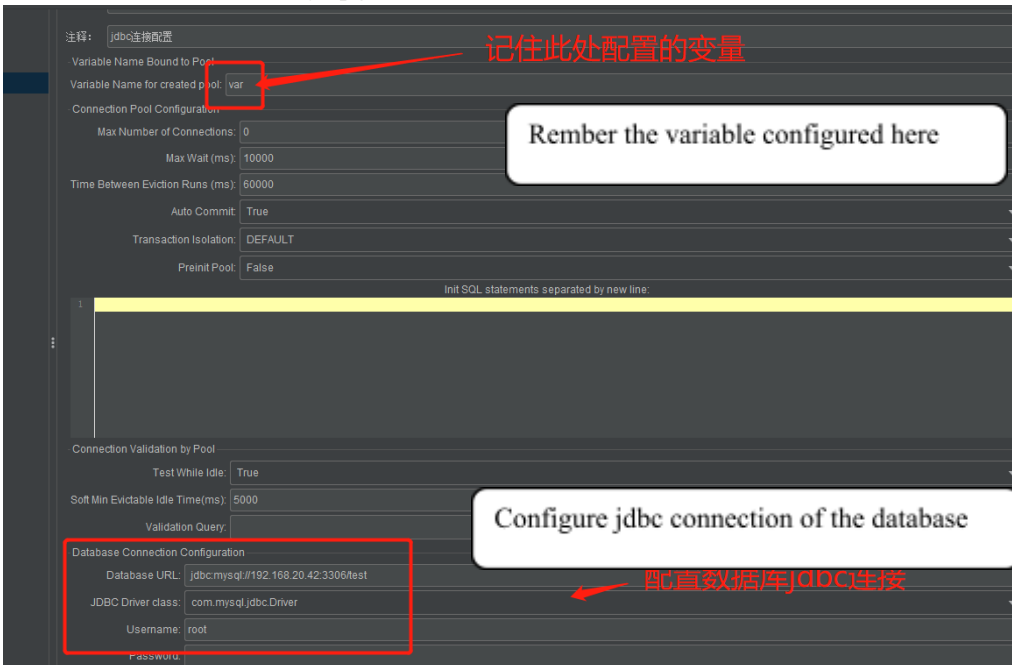
DatabasesURL: jdbc:rpdsq1://192.168.20.42:3306/test

JDBC Driver class : com.rpdsq1.jdbc.Driver

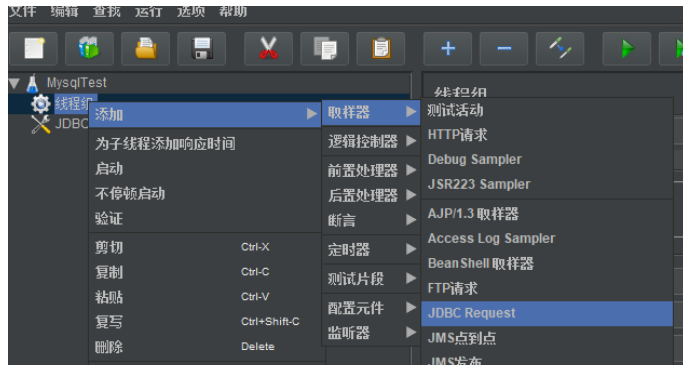
or (please note that there is no blank during filling-in)

DatabasesURL: jdbc:mysql://192.168.20.42:3306/test

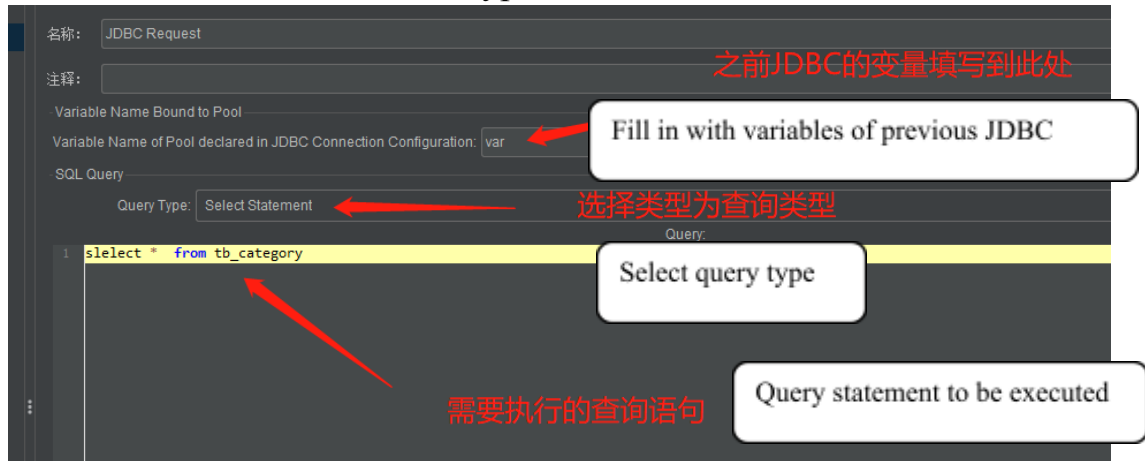
JDBC Driver class : com.mysql.jdbc.Driver



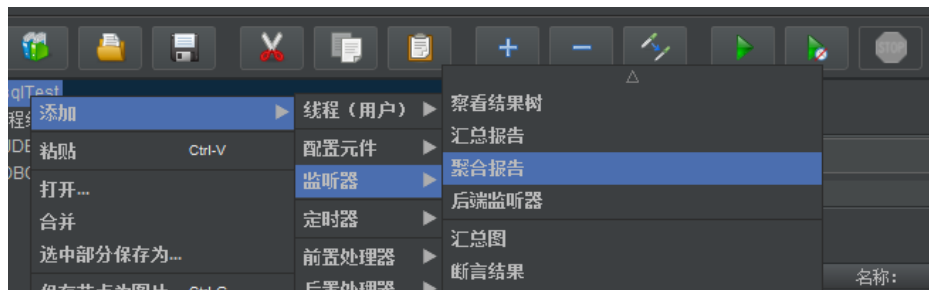
Step 5: right-click “thread group”, select “add”, “sampler” and click JDBC Request



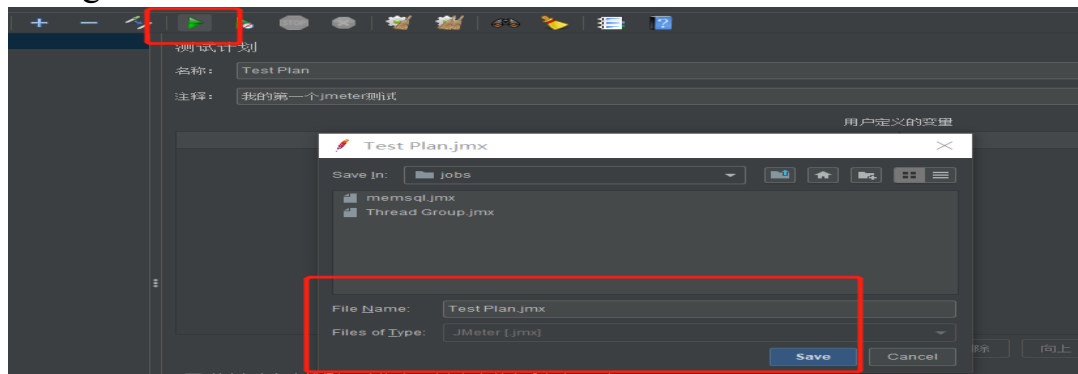
Fill in SQL statement and type to be executed:



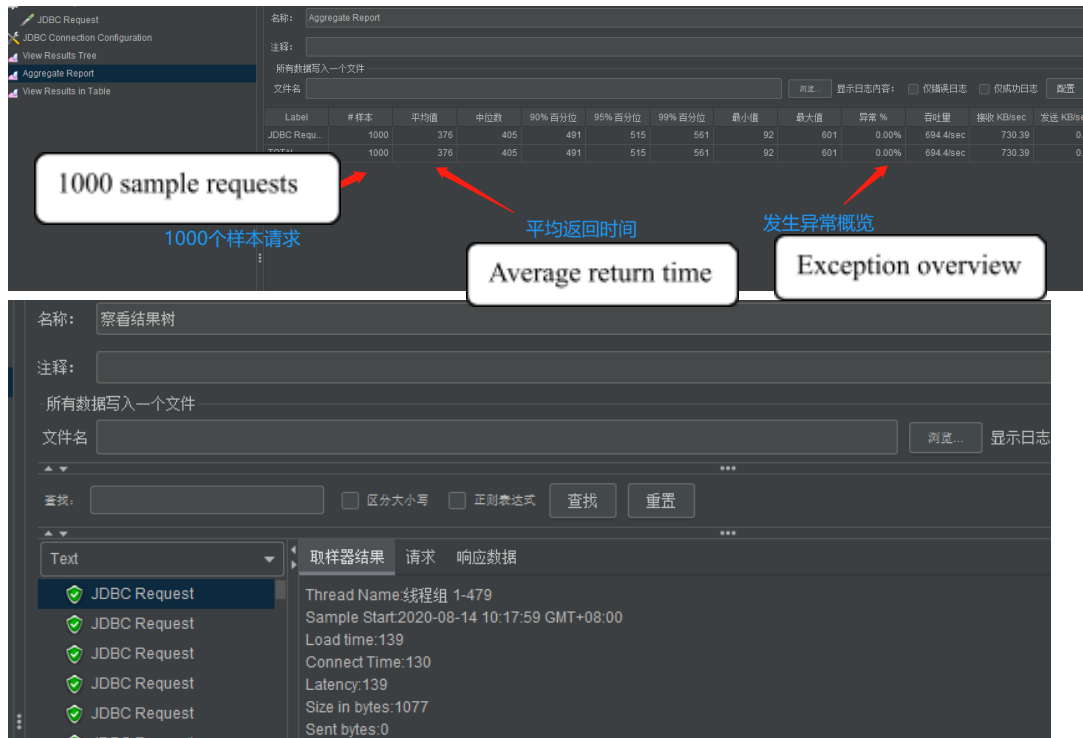
Step 6: right-click TestPlan to select “add”, “monitor”, and report result, such as aggregation report, view observation tree, and view the form.



Step 7: click the execution button, i.e. green triangle, and save files during the first run.



View various report results:



9.2 Linux installation and use

Firstly, downloaded apache-jmeter-5.3.tgz is uploaded to server/opt/software:

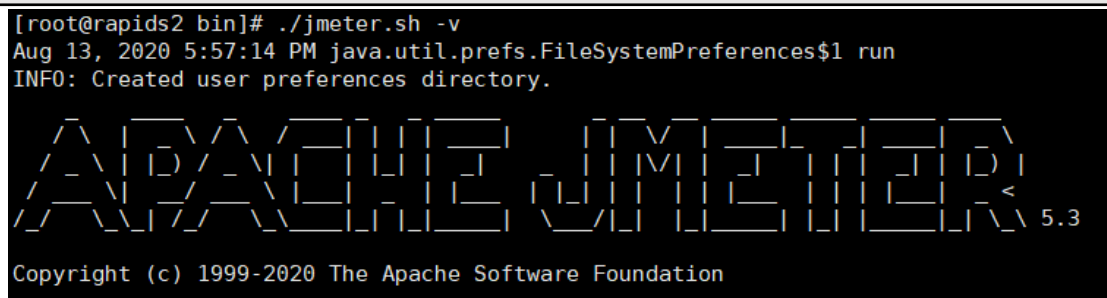
```
[root@node1 software]# ll
-rw-r--r--. 1 root root 67364344 Aug 13 11:16 apache-jmeter-5.3.tgz
```

Secondly, it's unzipped to /opt/module:

```
[root@node1 software]# tar -zxvf apache-jmeter-5.3.tgz -C /opt/module/
```

Finally, it's necessary to operate the script and view the version:

```
[root@node1 software]# cd /opt/module/apache-jmeter-5.3/bin
[root@node1 software]# ./jmeter.sh -v
```



Instructions on parameters of script operation:

- h help ->print the useful information and quit
- n non-GUI mode ->operate JMeter in non-GUI mode
- t test file ->JMeter test script file to be operated
- l log file ->files recording results

- r remote execution ->start remote service
- H proxy host ->set the proxy host used by JMeter
- P proxy port ->set port number of proxy host used by JMeter

The format of script file in Linux to be executed is .jmx. Project configured in Windows can be exported to a jmx format file and then uploaded to the server for execution.



Execute the script and view the results:

```
[root@node1 apache-jmeter-5.3]# bin/jmeter -n -t ./jobs/test.jmx -l ./result/res1.txt &
```

```
[root@rapids2 apache-jmeter-5.3]# bin/jmeter -n -t ./jobs/memsql.jmx -l ./result/res1.txt &
[2] 17637
[root@rapids2 apache-jmeter-5.3]# Creating summariser <summary>
Created the tree successfully using ./jobs/memsql.jmx
Starting standalone test @ Fri Aug 14 13:06:10 CST 2020 (1597381570896)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
summary = 1000 in 00:00:05 = 192.1/s Avg: 1995 Min: 31 Max: 4286 Err: 0 (0.00%)
Tidying up ... @ Fri Aug 14 13:06:16 CST 2020 (1597381576706)
... end of run
```

```
[root@node1 result]# cat res1.txt
```

```
1597381572362,1309,JDBC Request,200,OK,Thread Group 1-418,text,true,,1077,0,675,675,null,1298,0,1198
1597381572345,1253,JDBC Request,200,OK,Thread Group 1-404,text,true,,1077,0,675,675,null,1251,0,1215
1597381572743,941,JDBC Request,200,OK,Thread Group 1-465,text,true,,1077,0,675,675,null,940,0,836
1597381572093,1682,JDBC Request,200,OK,Thread Group 1-256,text,true,,1077,0,675,675,null,1680,0,1572
1597381571597,2112,JDBC Request,200,OK,Thread Group 1-13,text,true,,1077,0,673,673,null,2110,0,1962
1597381572750,1015,JDBC Request,200,OK,Thread Group 1-471,text,true,,1077,0,670,670,null,1014,0,838
1597381571649,2128,JDBC Request,200,OK,Thread Group 1-85,text,true,,1077,0,669,669,null,2119,0,2077
1597381571621,2149,JDBC Request,200,OK,Thread Group 1-65,text,true,,1077,0,669,669,null,2146,0,2096
1597381572861,1767,JDBC Request,200,OK,Thread Group 1-273,text,true,,1077,0,671,671,null,1767,0,1736
1597381572303,1542,JDBC Request,200,OK,Thread Group 1-379,text,true,,1077,0,671,671,null,1541,0,1479
1597381571619,2244,JDBC Request,200,OK,Thread Group 1-61,text,true,,1077,0,670,670,null,2206,0,2168
1597381571708,2166,JDBC Request,200,OK,Thread Group 1-115,text,true,,1077,0,669,669,null,2164,0,2138
1597381572062,1812,JDBC Request,200,OK,Thread Group 1-275,text,true,,1077,0,669,669,null,1811,0,1736
1597381571646,2232,JDBC Request,200,OK,Thread Group 1-79,text,true,,1077,0,667,667,null,2164,0,2136
```

9.3 Create data by JMeter

Firstly, we introduce the data creation in WINDOW, and number in the range is simulated by the system and inserted into the table. When table ta is taken as an example, ta table structure is as follows:

```
create table ta (
  uuidvarchar(10),companyid int(1),loansign int(1),a001 varchar(10),a002 varchar(10),a003
```

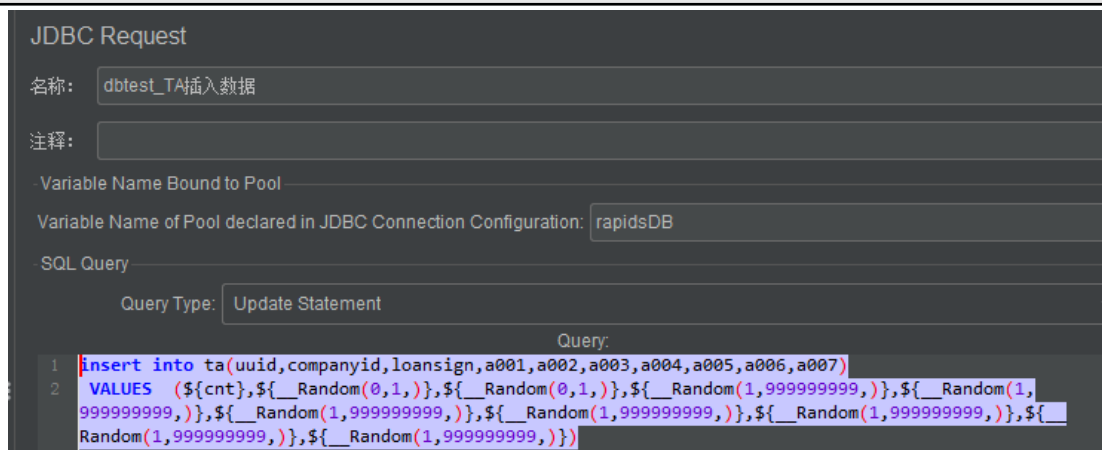
```

varchar(10),a004  varchar(10),a005  varchar(10),a006  varchar(10),a007  varchar(10),a008
varchar(10),a009  varchar(10),a010  varchar(10),key (companyid),key (loansign),
primary key (uuid));
  
```

In the “thread group”, select “add”, “sampler” and click JDBC Request and enter in sql (note: there may be multiple JDBC Requests, which can be manually opened and closed; if they are open, it’s necessary to concurrently write-in along with synchronous execution of multiple tables):

```

insert into ta(uuid,companyid,loansign,a001,a002,a003,a004,a005,a006,a007,a008,a009,a010)
VALUES
(${cnt},${__Random(0,1)},$__Random(0,1)},$__Random(1,999999999)},$__Random(1,99
9999999)},$__Random(1,999999999)},$__Random(1,999999999)},$__Random(1,999999
99)},$__Random(1,999999999)},$__Random(1,999999999)},$__Random(1,99999999
9)},$__Random(1,999999999)},$__Random(1,999999999)},$__Random(1,99999999
9)},$__Random(1,999999999)},$__Random(1,999999999)}
  
```



The number of threads is set as 100, which represents 100 concurrent write-ins:



It’s faster to write-in through script execution in Linux. It’s recommended to export jmx files after configuration of Windows, and upload the file to the server for execution, which can save more time.

10、Backup and Restoration

Firstly, we backup the database (backup the single table):

```
rapids > BACKUP testdata TO "/home/testdata";
```

```
2338b28fa85d1cbcf341117d797c090975cd26822
e9fe47c1baa326e03a98967517a36fd54906dfb25
8f7fd43fe6881adef9d0b315a71fbb3f3f0efae44
83fe4e40eb6c24416034c11f66fe6994a5e0536e9
2f9682649dc8411318c53c61d8b7041ae48e6feb3
1 row(s) returned (11.90 sec)
```

Then, database is restored:

```
rapids > drop database testdata;
```

```
10 row(s) returned (3.03 sec)
```

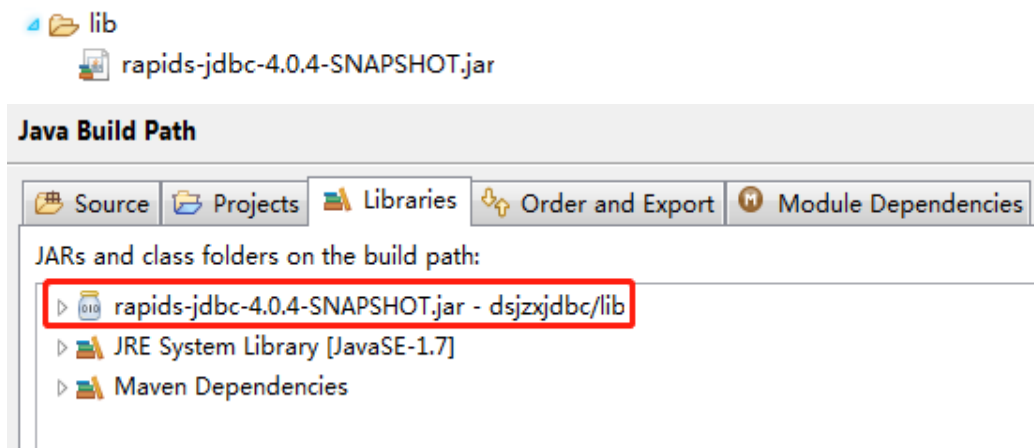
```
rapids > RESTORE DATABASE testdata from "/home/testdata";
```

```
1 row(s) returned (170.85 sec)
```

11、Connect JAVA

JDBC driver package is provided by the database for connecting JAVA. The following describes database connection and usage at JAVA. For specific jar package and project files, please refer to the appendix.

After creation of new project, it's necessary to firstly associate the latest JDBC driver package to the library, right-click lib file to click Configure Build Path in the Build Path, and then add the latest JDBC to them:



After environment configuration, it can be seen from JAVA class code that corresponding url, user name, password and name of the connector have been entered, and RapidsDB has been installed at IP:

192.168.252.101 of url, with port 4333, user name of rapids, and password of rapids, and the connector named DSJZX has been created in RapidsDB:

```
public class jdbcRapids {

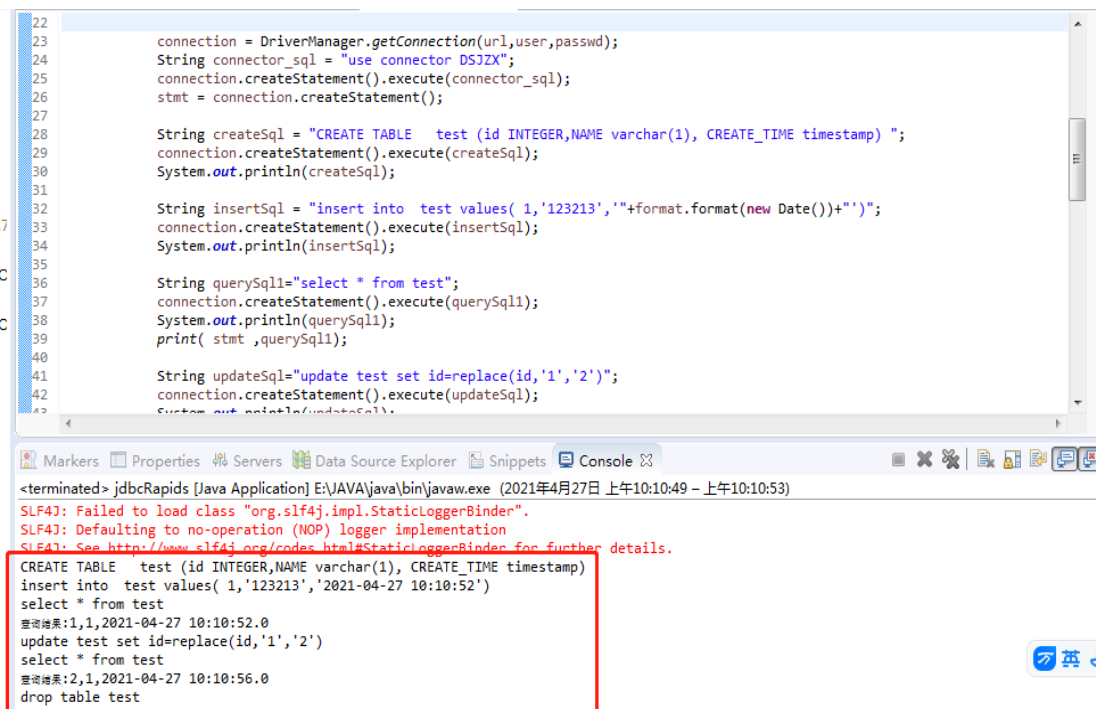
    public static void main(String[] args) {
        java.sql.Connection connection = null;
        Statement stmt = null;
        try {
            String ip = "192.168.252.101";
            String port = "4333";
            String user = "RAPIDS";
            String passwd = "rapids";
            String driver = "com.rapidsdata.jdbcdriver.Driver";
            String url = "jdbc:rdp://" + ip + ":" + port;
            /* 加载jdbc驱动 */
            Class.forName(driver);
            SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

            connection = DriverManager.getConnection(url,user,passwd);
            String connector_sql = "use connector DSJZX";
            connection.createStatement().execute(connector_sql);
            stmt = connection.createStatement();

            String createSql = "CREATE TABLE test (id INTEGER,NAME varchar(1), CREATE_TIME timestamp) ";
            connection.createStatement().execute(createSql);
            System.out.println(createSql);

            String insertSql = "insert into test values( 1,'123213','"+format.format(new Date())+"')";
            connection.createStatement().execute(insertSql);
            System.out.println(insertSql);
```

Then, it's necessary to execute SQL statement for adding, deletion, alteration and query, and test whether connection is completed. Results printed below can be used to prove successful connection of database and execution:



```
22
23 connection = DriverManager.getConnection(url,user,passwd);
24 String connector_sql = "use connector DSJZX";
25 connection.createStatement().execute(connector_sql);
26 stmt = connection.createStatement();
27
28 String createSql = "CREATE TABLE test (id INTEGER,NAME varchar(1), CREATE_TIME timestamp) ";
29 connection.createStatement().execute(createSql);
30 System.out.println(createSql);
31
32 String insertSql = "insert into test values( 1,'123213','"+format.format(new Date())+"')";
33 connection.createStatement().execute(insertSql);
34 System.out.println(insertSql);
35
36 String querySql1="select * from test";
37 connection.createStatement().execute(querySql1);
38 System.out.println(querySql1);
39 print( stmt ,querySql1);
40
41 String updateSql="update test set id=replace(id,'1','2')";
42 connection.createStatement().execute(updateSql);
43 System.out.println(updateSql);
```

```
<terminated> jdbcRapids [Java Application] E:\JAVA\java\bin\javaw.exe (2021年4月27日 上午10:10:49 - 上午10:10:53)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
CREATE TABLE test (id INTEGER,NAME varchar(1), CREATE_TIME timestamp)
insert into test values( 1,'123213','2021-04-27 10:10:52')
select * from test
查询结果:1,1,2021-04-27 10:10:52.0
update test set id=replace(id,'1','2')
select * from test
查询结果:2,1,2021-04-27 10:10:56.0
drop table test
```

