Report Assignment 2 Part 1:
Implementing a Sorted File (BigQ)

Group Members:

1. Dhiraj Mahesh Paryani, UFID: 1692 1261
2. Yogesh Laxman, UFID: 9451 2517

Instructions for Execution/ Code compilation and run tests:

I. Extracting folders

a. Extract the contents of the folder. DhirajMaheshParyani_YogeshLaxman_p2.zip
b. Open the terminal and navigate to the extracted folder.

II. Creating binary files from tpch files

a) Execute the following commands. Please note that the tpch files should be generated using the dbgen program before execution. Please change the tpch, dbfile output and catalog directories if necessary, in the a1test.cc file.
    1. $ make clean
    2. $ make a1test.out
    3. $ ./a1test.out
b) This would give a menu-based interface the performs the following three options that can test the code:
    1. load (read a tpch file and outputs a binary heap DBFile)
    2. scan (read records from an existing heap DBFile)
    3. scan & filter (read records from an existing heap DBFile and filter using a CNF predicate)
c) Select 1 to generate binary heap file in “db-files” directory.
d) Now select the required table from the list of tables.
e) The selected table would create a binary heap “table_name.bin” file in the db-files directory.

III. Running tests

a) Execute the following commands. Please note that the tpch files should be generated using the dbgen program before execution. Please change the tpch, dbfile output and catalog directories if necessary, in the test.h file.
    1. $ make clean
    2. $ make test.out
    3. $ ./test.out

    If you want to run all the predefined test cases, please run ./runTestCases.sh
b) This would give a menu-based interface the performs the following three options that can test the code:
    1. sort
    2. sort + display
    3. sort + write
c) Provide inputs for the menu-based interface, table name, run length and sort order as required.

Brief explanation of the methods implemented and how it works:

Our BigQ class consist of three data structures:

1. WorkerThreadData: This provides input arguments to the worker thread.
2. RecordComparator: This uses the OrderMaker class to compare two records based on the provided CNF sort order.
3. PriorityQueueItem: This structure maintains the order of the record within the priority queue.

The constructor of the BigQ class spawns a worker thread which calls TPMMS method. TPMMS method generates sorted runs of run-length number of pages, merges the sorted runs into the final sorted output and then performs cleanup of the intermediate files that were produced during the process of sorting. Below mentioned are a few methods that we have implemented within the BigQ class:

1. TPMMS(): This is the key function that is called by the worker thread from the BigQ constructor. This method runs the InitializeWorkerThreadData, RunGeneration, MergeRuns and CleanUp methods one after the other.

2. InitializeWorkerThreadData(): It creates buffer page arrays to store the records of the ongoing run and then create temporary files to store runs.

3. RunGeneration(): This function extracts records from the input pipe into run pages until the pipe is empty. If the current run page is full, then a new run page is created until the number of the run pages becomes equal to the given run length. When run-length pages of records are full, it calls CreateRun() method to create a sorted run of those pages and stores sorted pages in the temporary file.

4. CreateRun(): It sorts the records and stores the current run page when the current page is full and then update the worker thread data.

5. AddRecordToCurrentRun(): This method adds a new record to the run page by using the append method of the page.

6. SortAndStoreCurrentRun(): This calls LoadCurrentRunPriorityQueue() and WritePriorityQueueContentToBigQFile().

7. LoadCurrentRunPriorityQueue(): It loads the records present in the current run page into the priority queue until the current run page is empty.

8. WritePriorityQueueContentToBigQFile(): This method will fetch records from the priority queue, insert sorted records into the buffer page, and when the buffer page is full it will write that page into the temporary BigQ file. This will happen iteratively for run length of pages.

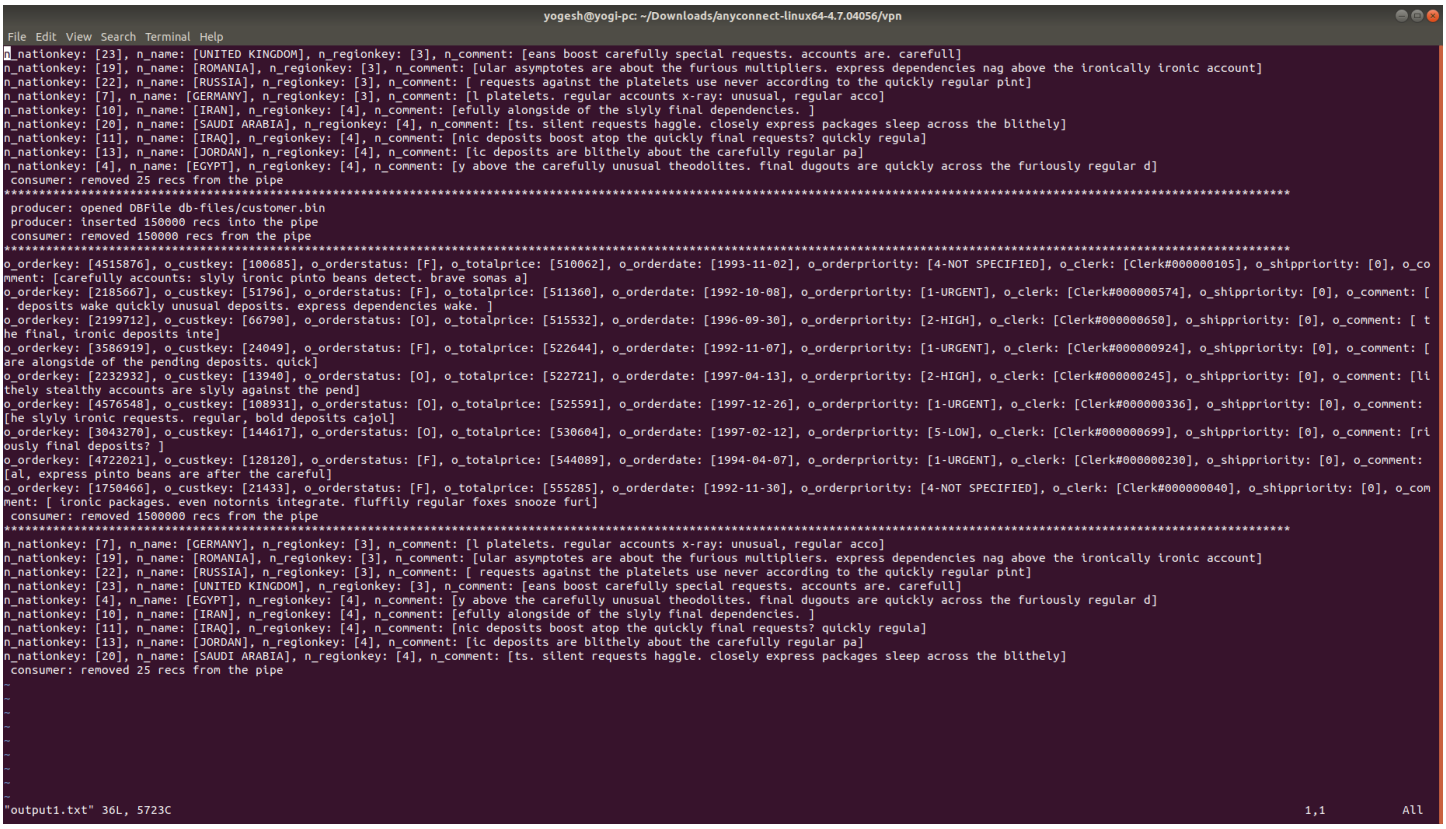
9. MergeRuns(): This function merges the sorted run pages into final output by calling LoadMergeRunPriorityQueue() and LoadOutputPipeWithPriorityQueueData() method sequentially.

10. LoadMergeRunPriorityQueue(): It takes the head of every run and appends it to the priority queue.

11. LoadOutputPipeWithPriorityQueueData(): It uses the priority queue data and fetches the next sorted element onto the Output pipe.

12. CleanUp(): It will BigQFile() is closed and then deleted. It then shut down the output file to free up space.

Screenshot of output1.txt generated from code:



Instructions to run GTest:

a) Execute the following commands. Please note that the tpch files should be generated using the dbgen program before execution. Please change the tpch, dbfile output and catalog directories if necessary, in the BigQGTest.cc file.

I. $ make clean
II. $ make gTestBigQ.out
III. $ ./gTestBigQ.out

Screenshots of GTest results that match results generated by code:

1. TestInitializeWorkerThreadDataMethod function: This function tests the correctness of InitializeWorkerThreadData() function which initializes the current run page number, number of runs and BigQ file. The test cases consist of both positive and negative testing in which currentRunPageNumber and numberOfRuns should be equal to 0. This should match the actual status within the EXPECT_EQ method.
2. TestCleanUpMethod function: This function tests the correctness of CleanUp() function and check if BigQFile() is closed and then deleted. This should match the actual status within the EXPECT_EQ method.

3. TestAddRecordToCurrentRunMethod function: This function checks the functioning of AddRecordToCurrentRun() method by checking if the number of records fetched from the DBFile match the number of records added in the current run.

4. TestLoadCurrentRunPriorityQueueMethod function: This function checks the functioning of LoadCurrentRunPriorityQueue() method as it loads into the priority queue the records present in the current run page into the priority queue until the current run page is empty. It checks if the number of records added is equal to the number of items within the priority queue and finally checks if no records are present in the array of current run pages.

GTest Output:

```
dhirajmaheshparyani@Dhirajs-MBP Project % make gTestBigQ.out
g++ -O2 -Wno-deprecated -g -c gtests/BigQGTests.cc
g++ -O2 -Wno-deprecated -o gTestBigQ.out Record.o Comparison.o ComparisonEngine.o Schema.o File.o BigQ.o DBFile.o Pipe.o y.tab.o lex.yy.o  BigQGTests.o -ll -lpthread -lgtest
dhirajmaheshparyani@Dhirajs-MBP Project % ./gTestBigQ.out
[==========] Running 4 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 4 tests from BigQTest
[ RUN      ] BigQTest.TestInitializeWorkerThreadDataMethod
[       OK ] BigQTest.TestInitializeWorkerThreadDataMethod (0 ms)
[ RUN      ] BigQTest.TestCleanUpMethod
[       OK ] BigQTest.TestCleanUpMethod (0 ms)
[ RUN      ] BigQTest.TestAddRecordToCurrentRunMethod
[       OK ] BigQTest.TestAddRecordToCurrentRunMethod (1 ms)
[ RUN      ] BigQTest.TestLoadCurrentRunPriorityQueueMethod
[       OK ] BigQTest.TestLoadCurrentRunPriorityQueueMethod (0 ms)
[----------] 4 tests from BigQTest (1 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test case ran. (2 ms total)
[  PASSED  ] 4 tests.
dhirajmaheshparyani@Dhirajs-MBP Project %
```