Deep Bodra

5801 1841

**REPORT**

# GROUP MEMBER NAMES

Individual Project

# 

# RUNNING THE CODE

## Run Test (test.cc)

make test.out

./test.out

# 

## Run gtest (gtest\_sortedfile.cc)

make gtest\_sortedfile.out

## ./gtest\_sortedfile.out

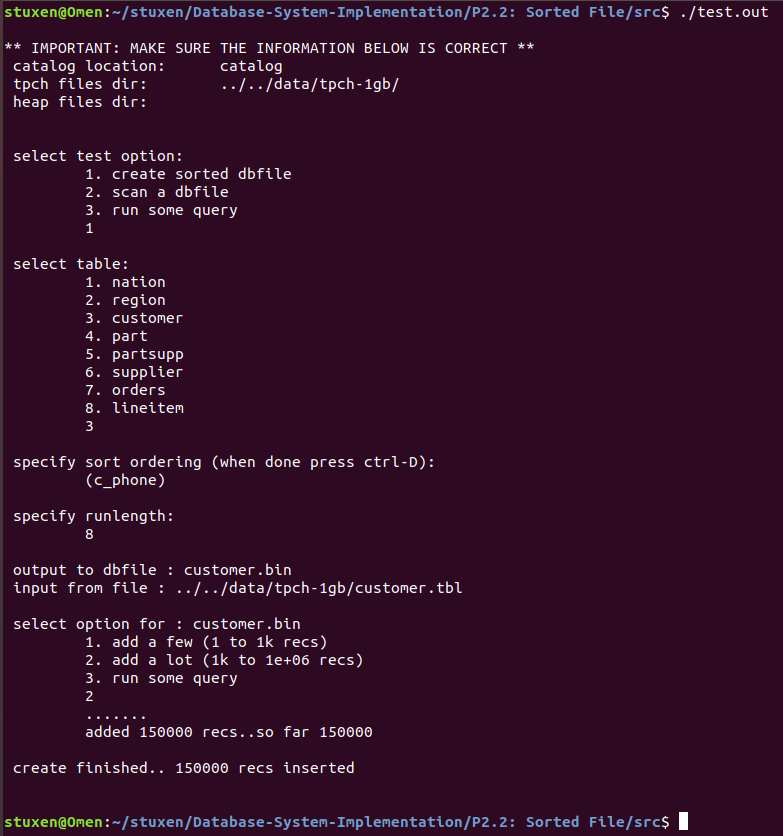
# FUNCTION DESCRIPTION

1. Create
   1. It creates a file to which the data will be written to
   2. It also stores a pointer to SortInfo for tracking runLength and OrderMaker instance
2. Open
   1. It first opens the meta file in binary mode and reads runLength and sortOrder
   2. It then opens the actual sorted file
3. Load
   1. If the fileMode is not write then it changes to WRITE mode and sets up the inputPipe, outputPipe and BigQ instance
   2. It reads the records from the input file and adds to the input pipe
4. Add
   1. If the fileMode is not write then it changes to WRITE mode, sets up the inputPipe, outputPipe and BigQ instance (if not set); and creates a pthread for TPMMS
   2. Irrespective of the fileMode it inserts the record into the inputPipe
5. MoveFirst
   1. If the fileMode is not READ then it changes to READ and merges Pipe and SortedFile using MergeFileAndPipe (discussed later)
   2. Otherwise it loads the first page and first record right away
6. Close
   1. If the fileMode is WRITE then it changes to READ and merges Pipe and SortedFile using MergeFileAndPipe (discussed later)
   2. It closes the SortedFile and writes out metadata about this file (fileType, orderMaker, runLength)
7. GetNext (without CNF)
   1. If the fileMode is not READ then it changes to READ and merges Pipe and SortedFile using MergeFileAndPipe (discussed later)
   2. Otherwise, it is same as that of HeapFile i.e loading next record from the page while loading next page if it gets empty
8. GetNext (with CNF)
   1. If the fileMode is not READ then it changes to READ and merges Pipe and SortedFile using MergeFileAndPipe (discussed later)
   2. If the order maker is not compatible, then it gets the first record that matches the literal by performing a sequential scan starting from ptrCurrentRecord
   3. If the order maker is compatible then load the page that might have the record using LoadProspectivePage (discussed later)
   4. If the returned record matches literal then we stop
   5. Otherwise, we scan this page sequentially until we find the record or the page exhausts
9. LoadProspectivePage
   1. If the CNF changed then currentPage is the lower page for binary search and lastPage is the higher page for BinarySearch
   2. The BinarySearch returns the page which might have the record we want (prospectivePage)
   3. We search this page sequentially and stop until we find a record that matches the literal
10. Binary Search
    1. If the lowerPageNumber matches midPageNumber then we have found the prospectivePage and return it
    2. If the first record in midPage is less than literal then we search in [lowerPageNumber, midPageNumber-1]
    3. If they match then we have found the prospectivePage and return it
    4. Otherwise, we search in [midPageNumber+1, higherPageNumber-1]
11. MergeFileAndPipe
    1. It shuts down the inputPipe and reads from outputPipe so that we get records from pipe in sorted order
    2. It maintains 2 Record instances, one recordFromFile and another recordFromPipe
    3. It uses a simple 2 way merge strategy
    4. If pushes the smaller record and replenishes this exhausted record with next record in the file/pipe
    5. This is done until both file and pipe get empty

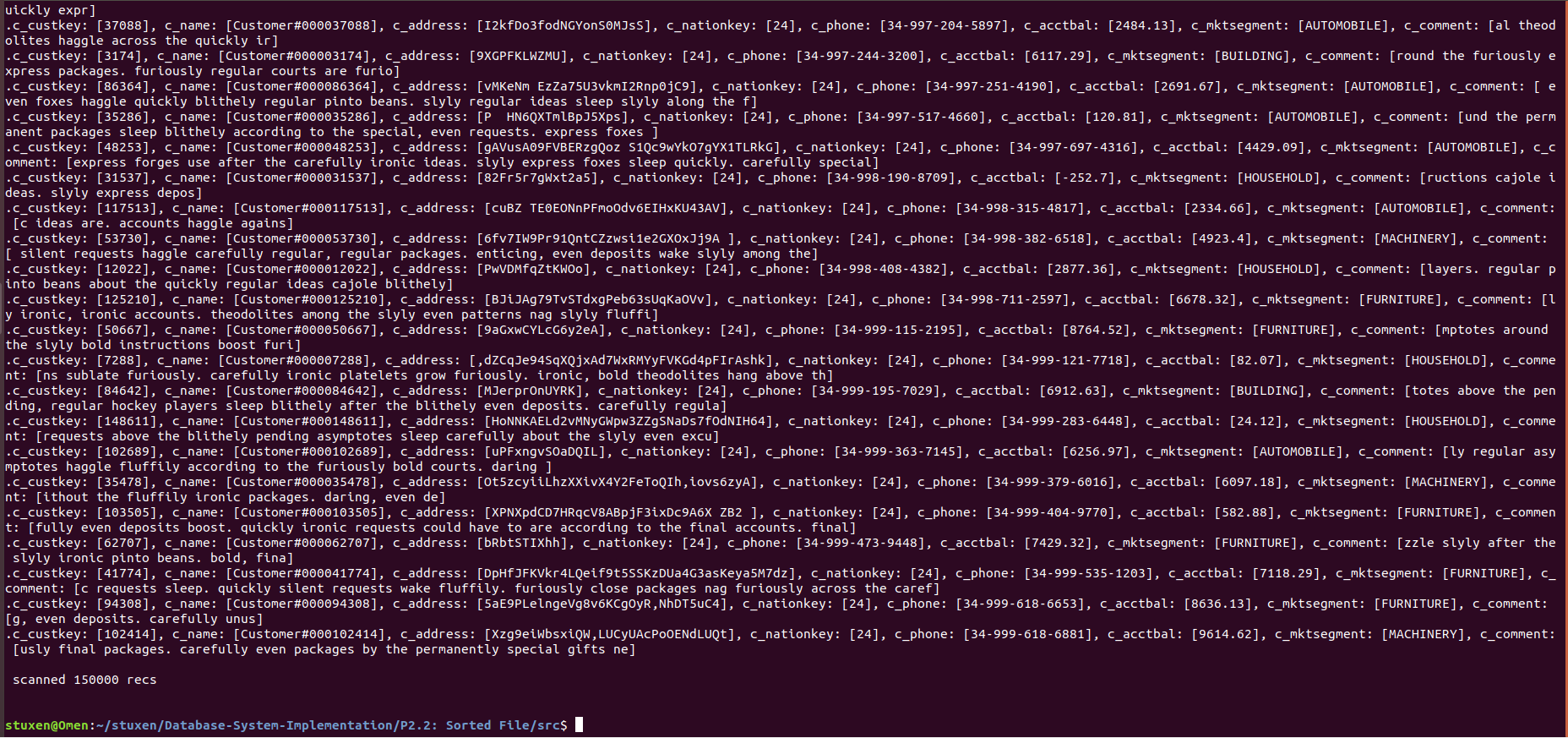
# TEST CASES

TPCH data size=1GB

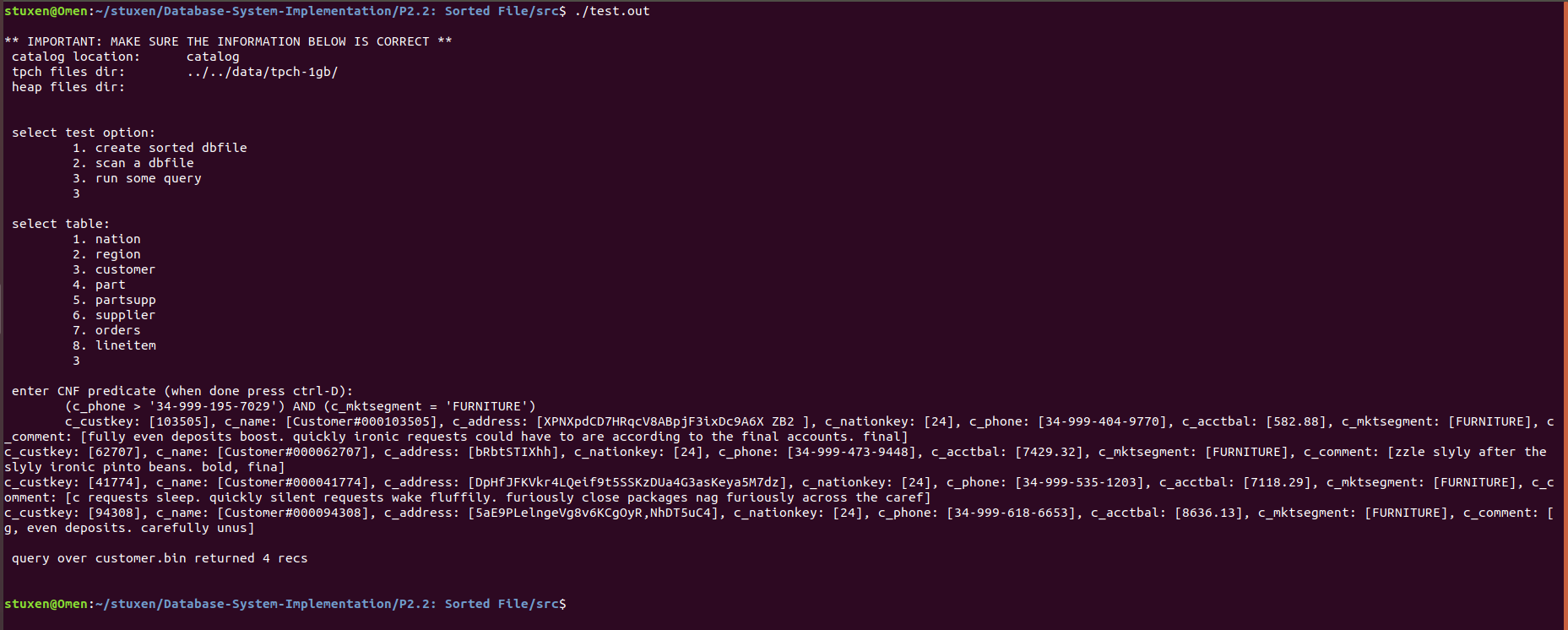
## Test Case 1



## Test Case 2



## Test Case 3

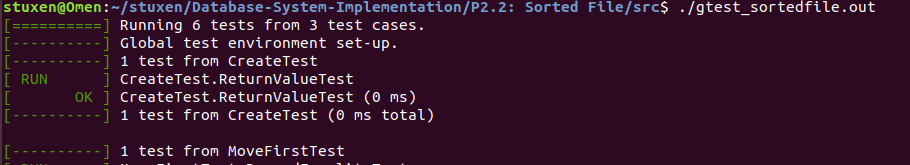


# GTEST

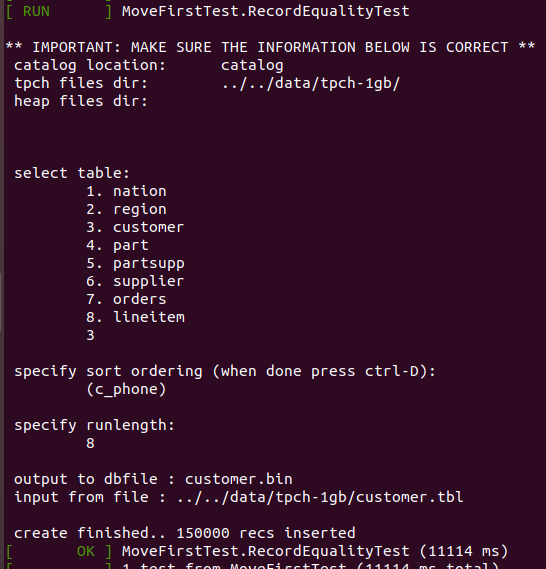
## Gtests for part 2

I have written 2 new gtests for this part.

1. ReturnValueTest
   1. It tests whether the return value after the creation of SortedFile is 1 or not



1. RecordEqualityTest
   1. This test is a bit involved and not so trivial
   2. It tests MoveFirst function
   3. It adds records to the file and calls MoveFirst (which triggers merging of pipe and file) before closing the file. It retrieves the first record
   4. It then closes, reopens the file and calls MoveFirst. It retrieves the first record
   5. The two records should be same



## Gtests from previous part

**Note**

1. Code has been modified a bit to accommodate changes as a result of the introduction of SortedFile
2. Also the output is generated again and not copied from previous report

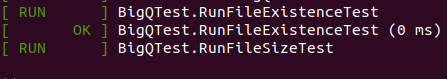
The thread TwoPassMultiwayMergeSort generates an intermediate file called the *run file* in the first phase. This file is named “yolo.runfile” in this project. I have written 4 gtests which test these files for various aspects.

**Notes before executing gtests**:

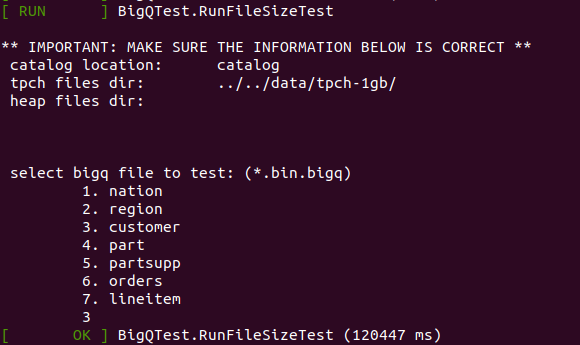
1. These tests are to be run after the test.cc has been executed with 1. Create sorted dbfile option
2. CNF for gtest must be same as you used when you executed test.cc

**Tests**

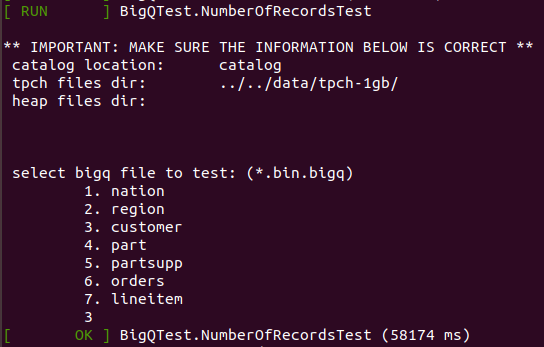
1. RunFileExistenceTest
   1. It tests whether the run file by the name “yolo.runfile” was created or not



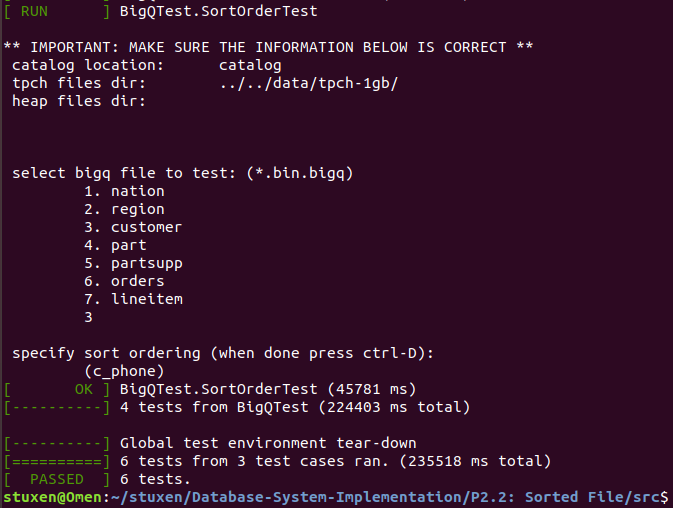
1. RunFileSizeTest
   1. It tests whether the file size of the *runfile* is as at least as that of the input *bin file* or not



1. NumberOfRecordsTest
   1. It tests whether the number of input records is equal to that of the output records



1. SortOrderTest
   1. It tests whether the number of records that are out of order is 0 or not



# CONCLUSION

We now have sorted file, a more sophisticated version of the database and will hopefully incorporate B+ trees into the project