Project Report: Database System Implementation (COP6726)

Project 2 Part 1: Implementing a Sorted File

Name: Vikas Chaubey**, UFID:** 3511 5826**, Email:** vikas.chaubey@ufl.edu

I have done this project alone without any project partner. Hence, I am submitting it on my behalf.

1) **Compile and Run Instructions:** This program has been compiled, run and tested on Ubuntu 18.04 operating system.

Step.1 – Run loadtables.out:
In order to run the program first we need to load the dbfiles using tpch-dbgen tables. For that purpose, loadtables.out should be run. Please make sure that variables dbfile_dir, tpch_dir and catalog_path with the appropriate location in the files "loadtables.cc" and "loadtables.h" . please use below commands sequentially and follow onscreen instruction to load bin files using tpch data tables.

```
> make clean
> make loadtables.out
> ./loadtables.out
```

Step.2 - Run test.out:
In order to run **test.out** first we need to make sure that step 1 is completed successfully and all heap files (.bin files) are available in dbfile directory. Please make sure that in test.h file dbfile_dir, tpch_dir and catalog_path are also updated correctly with right locations. please use below commands sequentially and follow onscreen instruction to run **test.out** file.

```
> make clean
> make test.out
> ./test.out
```

Step.3 - Run runTestCases.sh:
In order to run **runTestCases.sh** first we need to make sure that step 1 is completed successfully and all heap files (.bin files) are available in dbfile directory. Please make sure that in "test.h" file dbfile_dir, tpch_dir and catalog_path are also updated correctly with right locations. please use below commands sequentially and follow onscreen instruction to run "**runTestCases.sh**" file.

```
> make clean
> make test.out
> ./runTestCases.sh
```

Step.4 - To run gtest (google tests):
In order to run **gtest** first we need to make sure that step 1 is completed successfully and all heap files (.bin files) are available in dbfile directory. Please make sure that in "gtest.cc" file dbfile_dir, tpch_dir and catalog_path are also updated correctly with right locations. Pass gtest_arg.txt as argument to run gtests. This file I present in the folder. please use below commands sequentially to run gtest.

```
> make clean
> make gtest.out
> ./gtest < gtest_arg.txt
```

2) Project files, structure, classes and functions: The BigQ file has following classes and structs:

a) Structs Created:

1. **QueueObj:** This struct encapsulates data for priority Queue
2. **RunFileObj:** This struct encapsulates data for Run manager.
3. **ThreadData:** This struct encapsulates data passed in BigQ class constructor.

```
// ------------------------------------------------------------------
// This structure is created to encapsulate Data for Priority Queue
typedef struct{
    //variable for runId
    int runIdValue;
    //Variable to save records
    Record * record;
} QueueObj;


// This structure is created to encapsulate Data for Runmanager
typedef struct{
    //variable  to save start page index
    int startPageInd;
    //variable  to save current page index
    int currentPageInd;
    //variable  to save end page index
    int endPageInd;
    //variable to save runID
    int runIdValue;
} RunFileObj;


// This structure is created to encapsulate Data Passed to BigQ's Constructor
typedef struct {
    Pipe * in;
    Pipe * out;
    OrderMaker * sortorder;
    int runlen;
} ThreadData;
// ------------------------------------------------------------------
```

b) Classes Created:

1. **Run Class:** The class is used to encapsulate a collection of pages called runs which are used as the unit of operation to implement the external sort.

```
// ------------------------------------------------------------------
class run {
    //variable to save run length
    int runSequenceLength;
    //orderMaker variable
    OrderMaker * sortorder;
    //vector to save pages
    vector <Page *> pages;
    public:
        //constructor to instantiate run class
        run(int runSequenceLength);
        //constructor to instantiate run class
        run(int runSequenceLength,OrderMaker * sortorder);
        //This function is getter method to get runSize.
        int getRunSize();
        vector<Page*> getPages();
        //this function is gtter method to getPages For Priority Queue.
        void getPages(vector<Page*> * pagevector);
        bool customRecordComparator(Record &left, Record &right);
        //this function is method to writeRun to File after Sorting.
        int writeRunToFile(File *file);
        void AddPage();
        //this function is method to add page
        void AddPage(Page *p);
        //this function is method to check if the run if full.
        bool checkRunFull();
        //this function is method to add record to the page.
        int addRecordAtPage(long long int pageCount, Record *rec);
        //this function is method to clear pages.
        bool clearPages();

};
// ------------------------------------------------------------------
```

2. **RunManager Class:** The class is used to fetch data for each run from the file. The Tournament Tree class uses this to suck in data from various runs and sort it.

```
// ------------------------------------------------------------------
// Class for managing runs
class RunManager{
    //variable to save number of runs
    int numberOfRuns;
    //variable to save run length
    int runSequenceLength;
    //variable to save total pages
    int totalNumberOfPages;
    //variable to save file object
    File file;
    //variable to save file path
    char * f_path;
    //unordered_map
    unordered_map<int,RunFileObj> runLocation;
public:
    //RunManager class cinstructor
    RunManager(int runSequenceLength,char * f_path);
//  This Function is a getter method to get Inital Set of Pages
    void getPages(vector<Page*> * PageVectorObj);
//  This Function is a getter method to get Next Page for a particular Run
    bool getNextPageOfRun(Page * page,int runNo);

    ~RunManager();
    int getNoOfRuns();
    int getRunLength();
    int getTotalPages();


};
// ------------------------------------------------------------------
```

3. **CustomComparator Class:** The class is used to implement custom comparator for vector sorting and priority queue sort.

```
// ------------------------------------------------------------------
// Class to implement custom comparator for vector sorting and priority queue sorting.
class CustomComparator{
    //variable to save comparisonenginer object
    ComparisonEngine ComparisonEngineObj;
    //variable to save ordermaker
    OrderMaker * OrderMakerObj;
public:
    //constructor to create custom comparator class
    CustomComparator(OrderMaker * sortorder);
    //this method sorts the vector of records
    bool operator()( Record* lhs,   Record* rhs);
    //this function sorts the priority queue
    bool operator()(QueueObj lhs,  QueueObj rhs);

};
// ------------------------------------------------------------------
```

4. **TournamentTree Class:** The class is used to sort a single run and is also used to merge the various sorted runs present in the file using a priority queue. The class has a output buffer in which the sorted data is put which is the size of a page.

```
// ------------------------------------------------------------------
// Class used to sort the records within a run or across runs using priority queue.
class TournamentTree{
    //variable to save ordermaker object
    OrderMaker * OrderMakerObj;
    //variable to save run manager object
    RunManager * RunManagerObj;
    //variable to save vector of pages
    vector<Page*> PageVectorObj;
    //variable to save pages
    Page PageOutputBuffer;
    // RUnManager availability checker
    bool isRunManagerAvailable;
    priority_queue<QueueObj,vector<QueueObj>,CustomComparator> * myQueue;
//  This Function initiates and processes the queue with records pulled from runManager
    void InititateQueue();
//  This Function initiates and processes the queue with records pulled from run
    void InititateQueueForRun();
public:
    //constructor to create tournamentTree
    TournamentTree(run * run,OrderMaker * sortorder);
    TournamentTree(RunManager * manager,OrderMaker * sortorder);
//  This method refills the output buffer using RunManger.
    void RefillOutputBufferUsingRunManager();
//  this function is a getter for sorted output buffer and this method also refills buffer again
    bool GetSortedPageAndRefill(Page * *p);
//  this function is a getter for refill output buffer using Run.
    void RefillOutputBuffer();
//  this function to get sorted output buffer and refill buffer again
    bool GetSortedPageForRunAndRefill(Page * *p);
};
// ------------------------------------------------------------------
```

5. **BigQ Class:** The class is sort the heap dB files. The class reads in records from an input pipe, the uses TPMMS algorithm to sort the records in a separate thread spawned by it and pushes the resultant record on an output pipe.

```
// ---------------------------------------------------------------------
// Class used to sort the heap binary files.
class BigQ {
    //variable to save thread data
     ThreadData ThreadDataObj;
     //variable to save tournament tree object
     TournamentTree * TreeObj;
     //variable to save pthread obj
     pthread_t pThreadObj;
     //variable to save total number of threads
     int totalNumberOfRuns;
     File myFile;
     char * f_path;
//   function to implement phase1 of TPMMS algorithm
     void TPMMSPhase1();
//   function to implement phase2 of TPMMS algorithm
     void TPMMSPhase2();

public:
    void sortCompleteRun(run *run, OrderMaker *sortorder);
    //   public static function to drive the TPMMS algorithm.
     static void* Driver(void*);
    //   constructor
     BigQ (Pipe &in, Pipe &out, OrderMaker &sortorder, int runlen);
    //   destructor
     ~BigQ ();
};
// ---------------------------------------------------------------------
```

c) BigQ Class Function Descriptions:

```
1) BigQ (Pipe &in, Pipe &out, OrderMaker &sortorder, int runlen);
```

Description: All the arguments of this constructor of BigQ class are stored in a struct called ThreadData. When this constructor is called a new thread is spawned, this thread then calls another static function which is Driver() method. And to Driver method this pointer is passed as an argument.

```
2) static void* Driver(void*);
```

Description: In the driver function, thread passes the pointer as an argument, then in this method the pointer is type casted to the instance. And then this method calls two different functions which implement TPMMS algorithm. These functions basically implement two phases in TPMMS algorithm. These functions are TPMMSPhase1( ) and TPMMSPhase2( ).

```
3) void TPMMSPhase1();
```

Description: In the phase1 of TPMMS algorithm implementation, this function removes the records from the pipe continuously. These records which are fetched from the pipe are then filled in the instance of Run Class. As the instance of the run class gets full to its capacity then this filled run class instance is passed to a method called "sortCompleteRun". This method sorts the records present in run object and returns the sorted object .once the run is sorted , it is written to .xbin file.

```
4) void TPMMSPhase2();
```

Description: This function implements the second phase of TPMMS algorithm, in this second phase this function creates the object for run manager class. To create this object this class constructor is fed with length of the run and path for the .xbin file which was created in the previous phase. Using these arguments the run manager approximates a structure , that structure stores run Location offsets in the unit of pages for all the runs. Once this is done a new instance of tournament class is created, to create this tournament class object run manager instance is fed to the constructor. then the function continuously fetches the page which contains sorted records using a while loop. and these records are then written into an output pipe. Once the input pipe gets empty and filled in the instance of a Run Class. the TournamentTree is empty and it closes.

```
5) void sortCompleteRun(Run *r);
```

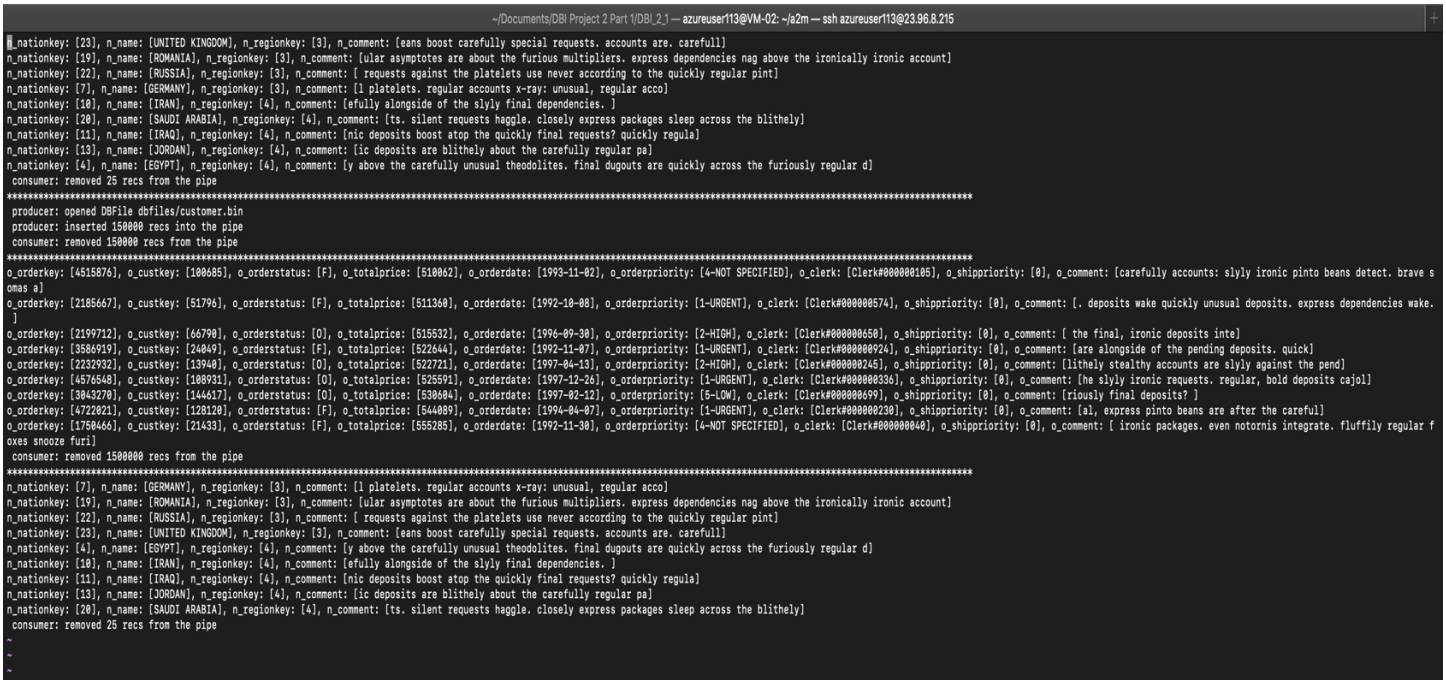
Description: This function sorts the records in the run class instance using a TournamentTree instance.This TournamentTree instance is built on a priority queue.The queue is fill up with all the records present in the pages and then the queue is popped off to get a sorted run.

```
6) ~BigQ ();
```

Description: In the destructor function the .xbin files which were created are removed.

3) Results Obtained on Program Execution: (All result screenshots are present in screenshots folder)

Output1.txt: (Program execution on tpch 1 GB data)



4) Gtest Results: