CSE 510 DBMSI PROJECT PHASE 1

Group 12

Vivek Sankara Menon 1215099119

Meghna Prabhu 1215160622

Akhil Jayaraj M 1215160609

Mahathi Srinivasan 1215187155

Eldo Jacob Mathews 1215090292

Renil Joseph 1213158685

ABSTRACT

In this world collecting and storing data is a challenging task as it is heterogeneous and voluminous. Given the fact that data is heterogeneous and can come from various sources, it is a lot harder to put the needed info from it together in order to interpret and make sense of it. Databases make it easier by efficiently storing and retrieving data for analytical purposes. I this phase, the Minibase Database is setup and its different functionalities are explored and tested using the interactive and non-interactive tests provided in Minibase.

Keywords: Database Management System, Relational Database, Heap Files, B-Tree, Buffer Management, Sort, Merge, Disk Space Manager, Java, Linux

1. INTRODUCTION

Minibase is a database management system used for educational purposes only. It comprises a parser, optimizer, buffer pool manager, storage mechanisms such as heap files, secondary indexes based on B-trees and a disk space management system [1]. A relational database is a collection of data items with pre-existing relationships between them. The data items are organized as a set of tables with rows and columns [2]. Relational databases also contain schemas, views, reports, queries and other items. A relational database is one that offers extremely complex and sophisticated queries and searches thanks to two factors: tables and cross-referencing. It stores data as tables rather than plain lists, making it easier to filter individual elements of each record. It also allows cross-referencing between different sets of data [3].

1.1 TERMINOLOGY

B-Tree-: It is a data structure that is a generalization of a binary search tree. It is self-balancing in nature and can stores sorted data for sequential access, insertions and deletions in logarithmic time [4]. A B+ Tree is an extension of B tree that allows efficient insertion, deletion and search operations [5]. It can be viewed as a B-tree in which each node contains only keys

(not key–value pairs), and to which an additional level is added at the bottom with linked leaves [6].

Buffer Manager-: It is the software component of the DBMS that handles all of the data requests. It also manages page replacement [7].

Database-: An organized collection of data [8].

Database Management System (DBMS)-: A system Software that can create and manage databases [9].

Heap files-: An unordered set of records that can created & destroyed, opened & closed, inserted and deleted [10].

Relational Database: A collection of multiple datasets organized by tables, records and columns [11]. It aids in creating and maintaining a well-established relationship between tables.

SQL-: Structured Query Language for Relational database management and Data Manipulation [12]

1.2 GOAL DESCRIPTION

The aim of this work is to setup the minibase in a Linux environment, understand the components of minibase and test the capabilities provided by the interactive and non-interactive test cases of minibase.

1.3 ASSUMPTIONS

1. Minibase is installed in a Linux Environment

2. For the B+ Trees tests, only positive values are entered and the choice made must be from 0 to 19

3. For choosing the file to be opened by specifying ‘k’, positive integers are given and are not strings or any invalid value.

1.4 TEST DESCRIPTION

After setting up minibase in a Linux environment, two set of tests need to be completed:

1. Non-interactive tests: Such tests don’t need the user’s input and display results on the terminal. Ex: Buffer management tests, Disk Management Tests, Heap files, Index Test, Sort Test, Sort-Merge tests, Join tests

2. Interactive tests: Such tests require the user’s input and display results of the test cases on the terminal. Ex: B+ Tree Tests

1.4.1 BUFFER MANAGEMENT TESTS

TEST 1:

- This test aims at checking the number of free frames and allocates pages based on the number of free frames.
- The function 'SystemDefs.JavabaseBM.getNumUnpinnedBuffers()' returns the number of pages to be allocated and JavabaseBM.newPage() is used to allocated the required number of pages.
- Each page is pinned to the buffer and data is written into it. Here, the data is written as page number+99999.
- After the data is entered in all of the pages, the program checks if the data entered in each page is correct. If not, it throws an error.
- The program also frees a page by checking if its page Id is within a range i.e pid.pid < lastPid.pid.
- An error is thrown if there is an exception on the reading, writing and comparing processes.

TEST 2:

- This test aims at checking if more pages can be pinned than the maximum number of frames available.
- Based on the value returned by SystemDefs.JavabaseBM.getNumUnpinnedBuffers(), the allowed number of pages are pinned until the Page ID exceeds the max. ID and an exception is thrown
- The program also tests if a double pinned page can be freed. As expected, it prints an error message

TEST 3:

- This program creates 60 pages by default. The goal is to allocate and dirty some pages while leaving some pages pinned
- Data in the form of Page ID+99999 written in all pages.
- If the Page ID mod 20! = 12, the page is unpinned else its pinned.

1.4.2 DISK MANAGEMENT TESTS

TEST 1:

- The program creates a new database and allocates new pages. File entries 'file+i'(0<=i<=5) are added.
- The program then requests the database to allocate run of the pages with size set to 30.
- After the run allocation, string 'Ai', 0<=i<=20 is written to some pages and rest of the pages are deallocated.

TEST 2:

- The program aims to delete specific files that were created in the database used in Test 1
- File0, File1, File2 are deleted. If a file which is to be deleted is not found, an error message is displayed.
- File3, Fil4, File5 remain in the database and the program checks if they are present.
- The data written in Test 1 are read

TEST 3:

- The program aims to delete a deleted file, delete a non-existent file, add a duplicate entry and perform run allocation/deallocation with invalid sizes.
- While trying to delete File0, it fails as expected.
- While trying to delete 'blargle' (non-existent), it fails as expected.
- While trying to add File3, it fails as expected.
- The program tries to add a file whose name exceeds the MAX_NAME by 5. It fails as expected
- A run allocation with size set to 9000 is attempted. It's not allowed.
- A run allocation with size set to -10 is attempted. It's not allowed.
- A run deallocation with size set to -10 is attempted. It's not allowed.

TEST 4:

- The program tests for some boundary conditions
- No unpinned pages are checked by comparing the number of unpinned buffers and total number of buffers.
- When there is a DB overhead, the program aims to create a new DB in order to accommodate the remaining pages. The database is big such that it needs 2 pages to hold its space map.
- A run is allocated with a set size just smaller than the newly created Database.
- Since an overhead is encountered in the previous step, allocating a new page will result in an error.
- Next, some pages are dellocated. In this ace, pages 3-9 and 33-40 are deallocated.
- In two continuous runs, pages 11-17 and 18-28 are freed.
- Next, the programs tries to allocate the freed pages (11-28) in a sequence. Some files entries (file3-file5) are deleted. The required number of files are added to the directory.
- The programs tries to allocate more pages than what's available (trying to allocate 7 pages when 6 are available).
- All pages are claimed when running the above test. Allocating one more page fails, as expected.
- As per the boundary condition in the space map, the last two pages are freed.

1.4.3 HEAP FILE TESTS

TEST 1:

- The program does an insertion operation of fixed-size records and then scans them.
- A heap file (file_1) is created using the command 'New Heapfile'.
- Then, the program inserts 100 records into the Heapfile.
- After insertion, a scan is done to check if the records did not change during insertion. The scan need not be in the order of that of the insertion. The scan should return the insertion order since fixed-length records are scanned.
- A test checks whether there are any pages pinned during insertion and scanning

TEST 2:

- The program aims to delete some records in file_1.
- It makes use of a variable i. If I is odd, the corresponding record is deleted. As a result, only even numbered records remain
- The programs then scans the remaining records

TEST 3:

- The program aims to update the remaining fixed-size records in the heap file.
- The heap file 'file_1' is opened and the entries are scanned. The fval of each entry is updated as 7*i.
- The file is scanned again to check if the updated values are correct for all entries.

TEST 4:

- The programs tries to alter the length of the records found in the heap file 'file_1'.
- First, it tries to shorten the first record and put the updated one into the heap file. It fails as expected.
- Then, it tries to lengthen the record. This one fails as well.
- Finally, it attempts to insert a record whose size is MINIBASE_PAGESIZE+4 (The maximum page size is MINIBASE_PAGESIZE=1024). This operation fails as expected.

1.4.4 INDEX TESTS

TEST 1:

- The program tries to create and insert a new index file.
- Two string arrays data1 and data2 are created. Both String arrays store a set of names. A heapfile 'test1.in' and a tuple are created. The heapfile is scanned and errors are thrown, if any.
- The strings in data1 are inserted into the B+ tree. Each leaf node of the B+ tree points to the data from the tuple with the record id for each value.

TEST 2:

- The program opens the heapfile 'test1.in' and the newly created index file 'BtreeIndex', which was created in TEST 1.
- For the string 'dsilva' a match operation is performed on the Btree index file after a successful index scan.
- If no tuples are selected the message 'Test2 – error in identity search' is printed
- If there are many matches the message 'Test2 – OOPS! Too many records' is printed
- The above steps are repeated with a limited search on the strings 'dsilva' and 'yuc'
- If the number of matches is less than the size of data2 i.e count<NUM_RECORDS -3, then the message 'Test2 – Oops! Too few records' is printed.

TEST 3:

- The program creates an unsorted data file ‘test3.in’. Inum and fnum are two numbers generated in randomn.
- Inum=random1.nextInt(); fnum=random2.nextFloat();
- An index is created on the integer field and an index scan is performed on the key. Since there are 1000 record, the key values returned must be in the range 100-900. If values returned are below 100 or greater than 900, error messages are printed.

1.4.5 JOIN TESTS

QUERY 1:

- Query: Find the names of sailors who have reserved boat number 1 and print out the date of reservation.
- SQL QUERY: SELECT S.sname, R.date FROM Sailors S, Reserves R WHERE S.sid = R.sid AND R.bid = 1
- The desired records are selected from the Sailors table & Reservation table, sorted and merged. The results are then projected

QUERY 2:

- Query: Find the names of sailors who have reserved a red boat and return them in alphabetical order.
- SQL QUERY: SELECT S.sname FROM Sailors S, Boats B, Reserves R WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red' ORDER BY S.sname
- The Btree index is built on sailors.sid. A join operation is performed on the Sailors table and the Reserves Table. Another join operation is perfomed when the B.color is set to Red.
- The snames are displayed in the ascending order.

QUERY 3:

- Query: Query: Find the names of sailors who have reserved a boat.
- QUERY executed: SELECT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
- The above query displays the list of sailors who have reserved a boat based on the merged data from Sailors and Reserves

QUERY 4:

- Query: Find the names of sailors who have reserved a boat and print each name once.
- QUERY executed: SELECT DISTINCT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
- The above query is similar to query 3 except that duplicate names are removed by using the ‘DupELim’ class and the unique names are projected

QUERY 5:

- Query: Find the names of old sailors or sailors with a rating less than 7, who have reserved a boat, (perhaps to increase the amount they have to pay to make a reservation).
- QUERY executed: SELECT S.sname, S.rating, S.age FROM Sailors S, Reserves R WHERE S.sid = R.sid and (S.age > 40 || S.rating < 7)
- The sailors with a rating <7 or age>40 are selected. The corresponding entries from the Reserves table are selected. After a sortmerge operation, the names, ratings and age of the sailors are projected

QUERY 6:

- Query: Find the names of sailors with a rating greater than 7 who have reserved a red boat, and print them out in sorted order.
- SQL QUERY: SELECT S.sname FROM Sailors S, Boats B, Reserves R WHERE S.sid = R.sid AND S.rating > 7 AND R.bid = B.bid AND B.color = 'red' ORDER BY S.name
- The above query selects sailor names with a rating greater than 7 using a file scan. In the Reserves table, records having color set to Red are read. Then using a nested loop join, the two results are merged and the names are displayed in ascending order.

1.4.6 SORT-MERGE JOIN TESTS

QUERY 1:

- Query: Find the names of sailors who have reserved boat number 1 and print out the date of reservation.
- SQL QUERY: SELECT S.sname, R.date FROM Sailors S, Reserves R WHERE S.sid = R.sid AND R.bid = 1
- The above query is executed to select sailors and their reservation date from tables Sailors and Reserves if they have reserved boat number 1 i.e R.bid=1.
- Using sortmerge sm, the results from the two tables are sorted, merged together and displayed.

QUERY 3:

- Query: Find the names of sailors who have reserved a boat.
- SQL QUERY: SELECT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
- Using a file scan, the names of the sailors who have reserved a boat are read from the tables Sailors and Reserves.
- Using sortmerge sm, the results from the two tables are sorted, merged together and displayed.

QUERY 4:

- Query: Find the names of sailors who have reserved a boat and print each name once.

- SQL QUERY: SELECT DISTINCT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
- The above query is similar to the one in Query 3 except that unique names are projected

QUERY 5:

- Query: Find the names of old sailors or sailors with a rating less than 7, who have reserved a boat, (perhaps to increase the amount they have to pay to make a reservation).
- SQL QUERY: SELECT S.sname, S.rating, S.age FROM Sailors S, Reserves R WHERE S.sid = R.sid and (S.age > 40 || S.rating < 7)
- The above query selects the sailors' names, rating and age whose rating is less than 7 or age is greater than 40. Using sortmerge sm, the results from the two tables are sorted, merged together and displayed.

1.4.7 SORT TESTS

TEST 1:

- There are 2 string arrays data1 and data2, unsorted and sorted respectively.
- A tuple is created and an unsorted heapfile 'test1.in' is created. The entries from data1 are inserted into 'test1.in'.
- This heapfile is sorted in the line: sort=new Sort(attrType, (short) 2, attrSize, fscan, 1, order[0], REC_LEN1, SORTPGNUM);
- Here order [0] is ascending, order[1] is descending
- The sorted heapfile is compared with data2. If the number of entries in data2 don't match the number of entries in the heapfile, error messages such as 'Too many records' or 'Too few records' are thrown accordingly.
- If the number of entries match, the entries in data2 are compared with the same in the heap file. If there is no match, it means that the entries in the heap file aren't sorted in the right order and an error message is printed.

TEST 2:

- This is similar to TEST1 except that an unsorted heap file 'test2.in' is created to store the entries in data1 and that the order to be followed is descending order.

TEST 3:

- The program creates a heap file 'test3.in' and sets the values in the tuple with the following:
  t.setStrFld(1, data1[i%NUM_RECORDS]); t.setIntFld(3, inum); t.setFloFld(4, fnum);
- inum and fnum are int and float values generated at random

- The records of data1 are inserted into the heap file. The entries are sorted based on IntFld in ascending order and based on FloFld in descending order.

TEST 4:

- The two strings arrays used in TEST 1 and TEST 2 is used in this test case.
- Two unsorted heap files 'test4-1.in' and 'test4-2.in' are created.
- Every record in data1 is inserted into both of the heap files.
- The heap file 'test4-1.in' is sorted in ascending order while 'test4-2.in' is sorted in descending order.
- The entries in both of the sorted heap files are compared with data2 to check if the number of entries match and if the entries are sorted accordingly. Based on the status of the operations, the appropriate messages are printed

1.4.8 B+ TREES TESTS

TEST 0:

- The program creates a file of the form 'AAAk', if a file of such a form doesn't exist.
- It performs a naïve delete wherein redistribution or merging of pages doesn't happen if the number of records in a page fall below a threshold.

TEST 1:

- Here, the program creates a file of the form 'AAAk', if a file of such a form doesn't exist.
- It performs a Full delete operation. There is redistribution or merging of pages if the number of records in a page fall below the given threshold.

TEST 2:

- It prints the entire B+Tree
- The print statement displays two columns. The first column denotes the depth and the second column denotes the current page ID

TEST 3:

- All leaf pages are printed
- The print statement displays the current page Id, left link, right link, key, page number and slot number.

TEST 4:

- A specific leaf page, as chosen by the user is printed
- The details printed is similar to the one in TEST 3

TEST 5:

- The program allows the user to insert a record into the B+ Tree
- Only positive key values are allowed. If its negative, the error message isn't printed but the condition breaks
- The keys are sorted and stored in the B+tree. Duplicate values can be entered as well

TEST 6:

- The program delets a record from the B+tree, as specified by the user.
- If there are duplicate values, only one element is deleted.

TEST 7:

- The user mentions 'n', the number of records to be inserted in order.
- Keys from 0 to n-1 are inserted into a page in ascending order.
- A filename of the form 'AAAk' is created.

TEST 8:

- The user mentions 'n', the number of records to be inserted in descending order.
- Keys from 0 to n-1 are inserted into a page in descending order.
- But on printing the B+ Tree, the records are displayed in ascending order since it is self-balancing in nature.

TEST 9:

- The user mentions 'n', the number of records to be inserted in random order.
- Keys from 0 to n-1 are inserted into a page in random order.
- But on printing the B+ Tree, the records are displayed in ascending order since it is self-balancing in nature.

TEST 10:

- The user is asked to insert n records randomly
- Keys from 0 to n-1 are inserted into a page in random order.
- User then mentions m records to be deleted
- Then m random records are deleted

TEST 11:

- The program deletes some records from the B+Tree
- The user mentions the values for the lower interger key and upper integer key. Both keys must be greater than or equal to zero.
- Keys with values within the lower and upper integer key are deleted. But if the lower inter key is greater than upper integer key, no deletion occurs.

TEST 12:

- The program initializes a scan on the B+Tree
- The user mentions the values lower interger key and upper integer key.
- The key with values within the lower and upper integer key are retrieved and used for TESTS 13 and 14

TEST 13:

- The program prints the next record based on the records retrieved in TEST 12
- The scan pointer points to the next element and print its record details till it reaches the upper bound key.

TEST 14:

- The program deletes the record which was last scanned in TEST 13.
- If TEST 13 is run again, it shows" AT THE END OF SCAN!"
- Now, on running TEST 14, it shows,"No record to delete, BTree.ScanDeletionException".

TEST 15:

- This is similar to TEST 10 except that string values are used for the keys. But in TEST 10, integer values are used.

TEST 16:

- The program closes the current file. Any insert/delete operation on the file closed results in an error

TEST 17:

- The program opens a file 'AAAk' as per the integer k, given by the user.
- When the file is open, any operation can be performed. If the file doesn't exist, it displays an error message

TEST 18:

- The user is asked to specify integer k so that file 'AAAk' is deleted.
- If the file doesn't exist, an error message is thrown

TEST 19:

- The user exits from the tests provided by the program

2. EXECUTION STEPS

1. Download the minjava.tar.gz and extract the files in a Linux System or Windows Subsystems for Linux

2. In the MakeFile of every folder, modify the classpath and javapath according to the system's directory structure

3. In the Command Prompt, get to the src folder and run bash in order to build the source

4. To create a typescript file type 'script'

5. Run 'make db'

6. To run all tests run 'make test'

7. Verify all tests provided. Interact with the B+ Tree tests

8. Exit from the running command prompt

3. CONCLUSION

The various components of a relational database were studied. With the help of interactive and non-interactive tests, we were able to explore all the components. We gained an understanding on the implementation and working of Buffer Management Systems, Disk Management Systems, Heap files, Sort, Indexing and Join operations. The structure and organization of B+Trees were understood as well. In the later phases we will learn and explore more functionalities and other capabilities of the minibase and use it to develop a database system.

BIBLIOGRAPHY

[1] The Minibase Homepage- http://research.cs.wisc.edu/coral/mini_doc/minibase.html

[2] https://aws.amazon.com/relational-database/

[3] https://itstillworks.com/purpose-features-relational-database-1961.html

[4] Wikipedia- https://en.wikipedia.org/wiki/B-tree

[5] https://www.javatpoint.com/b-plus-tree

[6] Wikipedia-https://en.wikipedia.org/wiki/B%2B_tree

[7] https://searchsqlserver.techtarget.com/definition/database-management-system

[8] https://en.wikipedia.org/wiki/Database

[9] https://www.chegg.com/homework-help/function-buffer-manager-serve-request-data-chapter-16-problem-29rq-solution-9780133970777-exc

[10] http://pages.cs.wisc.edu/~dbbook/openAccess/Minibase/spaceMgr/heap_file.html

[11] https://www.techopedia.com/definition/1234/relational-database-rdb

[12] https://www.techopedia.com/definition/1245/structured-query-language-sql