DBMSI Project Report

Phase-1

*Submitted by* Spoorthi Karnati

Group members:

1. Ashish Agarwal
2. Supriyaa Damodaraswamy
3. Spoorthi Karnati
4. Naga Subrahmanya Sai Tarun Malladi
5. Thiviyakalyani Navaneethan
6. Sucharitha Rumesh
7. Hariharan Adhithya Venkatraman

**ABSTRACT**

This project is about observing a Minibase which acts as a small database management system. This mini base is coded in JAVA and it gives us a clear picture of how disc space is managed, how data is stored using several storage mechanisms like Heap files, B+ trees, how buffer pool is managed and several other functions. This Mini base also has a menu with several menu items to test the functionalities of B+ tree. It includes creating/ opening/ deleting/ closing files and inserting/ deleting one or several records in a file. All these tests output get stored in an output file called ‘Typescript’.

**INTRODUCTION**

Mini base is a database management system which has several components like parser, optimizer, buffer pool manager, storage mechanisms (heap files, secondary indexes based on B+ Trees), and a disk space management system. With the help of mini base, we can perform several tests on each of its components to understand its functionalities. The next pages of this document drive us to set of tests performed on each component along with its results.

**Description of the testcases:**

After running the command “make test”, all the tests inside it are run one after the other in the order: bmtest -> dbtest -> hftest ->bttest ->indextest ->jointest ->sorttest -> sortmerge

**Test1:** **BMTest** – involves tests about buffer manager operations

Tests involved in it are:

* Creating a new page
* Adding bunches of new pages to it (called as pinning)
* Removing the pages from it (called as unpinning)
* Reading data from the pages
* Freeing up the pages
* Handling exceptional errors that occur when trying to bundle up more pages or unpin more pages.

All these functions are user defined functions and they are defined in “bufmgr” package. The code in BMTEST uses all these functions by importing ‘bufmgr’ package. All the pages are created and handled using Hash Tables. Common functions are declared in ‘global’ package. All the exceptions are handled by declaring each of them in a separate Java file inside ‘bufmgr’.

**Test2: DBTest** – tests involve disk management operations

Tests involved in it:

* Creating a new database
* Adding file entries by allocating id to each page
* Throws exception errors if pages are not allocated or unable to add file entry.
* Allocating more number of run pages starting from given page id.
* Storing data into the created file.
* De-allocating the unused files.
* Deleting file entries
* Finding the file entries that are undeleted (or currently available in database)
* Reading the data that is already stored.
* Displaying error messages when trying to
  + delete a deleted entry or a non-existent file entry
  + look up for a non-existent file entry or look up for a deleted file entry
  + add a file entry that exists already
  + add a file entry with too long name
  + allocate a run of pages which is too long
  + allocate and deallocate negative run of pages
  + allocate one more page after DB overhead is accounted
* Freeing some of the allocated pages, adding enough file entries for directory to surpass the page.

These user defined functions are defined and declared in DBTest.java and DB.java

**Test 3: HFTest** – tests involves heap file operations

Tests involved in it:

* Creating a new heap file and inserting records into it.
* Scanning the records inserted in the heap file
* Deleting a fixed set of records
* Updating fixed set of records
* Error messages are displayed when trying to
* Change the record length either by lengthening or shortening it
* Insert a record that is too long

All these functionalities are defined and declared in HFTest.java and Scan.java etc.

**Test 4: BTTest** – tests involves various B-tree operations

**Description of the Test cases**

Menu contains total 20 menu options where each option performs a function. (Below test cases examples are referred from ‘Typescript file’)

**Test case0:** Naïve Delete (new file)

* This creates a new file with a name assigned to it.
* For example, when I make a choice ‘0’ and press enter, a new file name ‘AAA1’ gets created.
* No redistribution or merging of pages are done when the number of records in a page are less than threshold limit value.

**Test case1:** Full Delete (default) (new file)

* This also creates a new file but redistribution and merging of pages is done when the number of records in a page are than the threshold limit value.
* For example, when I make a choice ‘1’ and press enter, a new file name ‘AAA2’ gets created.

**Test case2:** Insert a Record

* By choosing option ‘5’, a single new record can be inserted into the recently created file ‘AAA2’.
* Each record that is inserted gets amended to the existing record in the file.
* Data gets stored in B+ tree s tructure.
* For example, I have inserted the following records into AAA2 file. [1,12,3,14,5,16]

**Test case3:** Print the B+ tree structure

* By taking option ‘2’, the B+ tree structure gets printed as shown below.

----------------------------The B+ Tree Structure --------------------------

1. 3

----------------------------End-------------------------------------------------

* 1 is a constant and 3 is the page id.

**Test case4:** Print all Leaf pages

* By taking option ‘3’, all the records that are inserted so far gets printed in ascending order.
* The output for the file AAA2 is [1,3,5,12,14,16].

**Test case5:**  Delete a record

* By choosing option ‘6’ we can delete a record in a file by inputting the integer key.
* For example, the integer key given is 3, then 3 gets deleted from the file.
* File after deleting looks like [1,5,12,14,16].
* In this way we can delete a particular record irrespective of its position in the file.

**Test case6:** Insert n records in order

* We can insert more than one record in ascending order by choosing option ‘7’.
* A new file gets created and we have to input the number of keys that are to be inserted into this new file.
* For example,

Please input the number of keys to insert: 5

The values that gets inserted into new file AAA1 are [0,1,2,3,4].

**Test case7:** Insert n records in reverse order

* We can insert more than one record in reverse order by choosing option ‘8’.
* A new file gets created and we have to input the number of keys that are to be inserted into this new file.
* For example,

Please input the number of keys to insert: 5

The values that gets stored into file are [5,4,3,2,1] and when we try to print them, the values get sorted in ascending order and prints [1,2,3,4,5].

**Test case8:** Insert n records in random order

* We can insert more than one record in random order by choosing option ‘9’.
* A new file gets created and we have to input the number of keys that are to be inserted into this new file.
* For example,

Please input the number of keys to insert: 5

The values are stored randomly into the file and when we try to print them, the values get sorted in ascending order and prints [1,2,3,4,5].

**Test case9:** Insert n records in random order and delete m records randomly

* We can insert and delete more than one record in random order by choosing option ‘10’.
* A new file gets created and we have to input the number of keys that are to be inserted randomly into this new file and also input the number of keys that are to be deleted randomly from this file.
* For example,

Please input the number of keys to insert: 5

Please input the number of keys to delete:2

The values are stored randomly into the file and gets deleted randomly. When we try to print them, the values get sorted in ascending order and prints [0,2,3].

**Test case10:** Choose a page to print

* All the new files that are created get stored in a page with unique page id’s.
* If we want to print the data of an old file, choose option ‘4’.
* Input the page number and see the records in the respective file.
* For example, given the input number = 7, output is [1,2,3,4,5]

**Test case11:** Delete some records

* Choose option ‘11’ to delete a set of records in a given range.
* We have to input the lower and higher integer key values so that the values in this integer key value range gets deleted.
* Given the lower integer = 2 and higher integer = 4, output that gets printed is [ 1,5].

**Test case12:** Initialize a scan

* We can scan records for a certain range by choosing option ‘12’.
* By inputting the lower and higher integer key values, say 1 and 3, all the values in the given range gets scanned.

**Test case13:** Scan the next record

* Option ‘13’ scans the first record with the integer key value same as lower integer key value that is inputted in option 12 and prints it.

**Test case14:** Delete the just scanned record

* By taking option ‘14’, we can delete the just scanned record and to see the remaining records in that file, take option ‘3.

**Test case15:** option ’15’ is same as 10.

**Test case16:** Close the file

* Choose option ’16’ to close a file that is currently opened.

**Test case17:** open which file (input an integer for the file name)

* If we want to open any file that is created previously, choose option 17 and input the last digit of the file name.

**Test case18:** destroy which file (input an integer for the file name)

* Input the last digit of the file name to destroy a file by choosing option ‘18’.
* File once destroyed cannot be retrieved and throws an exception error when tried to display its records or open it.

**Test case19:** Quit

------------------------End of B+ tree tests --------------------------------------

**Test 5: IndexTest** – tests on BTree index filed

Tests involved in it:

* Creating a new BTree Index file
* Opening the newly created BTree Index file
* Checking whether BTree index is created successfully or not.
* Checking whether BTree index file is created successfully or not.

**Test 6: JoinTest** – Tests on Joins are done using three relations – sailors, boats, reserves and how different joins are performed on those relations. Joins that are used for demonstration are – sort-merge, nested-loops and index nested- loops.

*Query1:* Query is to find the sailor names who has reserved boat number 1 and print the dates on which reservation is done.

*Ans:* Given query performs a sort merge join considering outer relation with sailors and inner relation with reserves. Sailor name and reservation date are projected at the end the join operation.

*Query2*: Find the name of sailors who have reserved a red boat and return the names in alphabetical order

*Ans:* Two nested loop joins are used – with the result of the sailors and reserves join forming the outer input of the second join. The boats are the inner relation. Tests filescan, projection, index selection, simple nested-loop joins and sorting are used.

*Query3:* Find the names of sailors who have reserved a boat.

*Ans*: Filescan, projection and sortmerge join are used to project the names of the sailors who have reserved a boat.

*Query4:* Find the names of sailors who have reserved a boat. and print each name once.

*Ans:* Filescan, projection, sort-merge join and duplication elimination are used to find the names of sailors who have reserved a boat.

*Query5*: 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.)

*Ans:* Filescan, multiple selection, projection, and sort-merge join are used to find the old sailors who have reserved a boat with rating less than 7.

*Query6:* Find the names of sailors with a rating greater than 7 who have reserved a red boat and print them out in sorted order.

*Ans:* Filescan, multiple selection, projection, sorting, and index-nested-loop join are used to find names of sailors who have reserved red boat with rating greater than 7.

**Test7: SortTest –**

**Test 1:** A given array is sorted by storing it in a Heap file in ascending order based on the string field. Tests are done for boundary conditions on size and sorted order.

**Test 2:** A given array is sorted by storing it in Heap file in descending order based on the string field. Tests are done on boundary conditions on size and sorted order.

**Test 3:** A given array is sorted by storing it in Heap file in ascending order based on an Integer field. Tests are done on boundary conditions on size and sorted order. Sort this array again in descending order based on Float field. Tests for boundary conditions on size and sorted order.

**Test 4:** Sorts given array by storing in Heap file in ascending order once and then in descending order based on a String field. Tests for boundary conditions on size and sorted order.

**Test8: Sort Merge Join Tests**

***Query 1:*** Find the names of the sailors who have reserved boat number one and the date of their reservation.

***Ans:*** Sort-Merge Join is done where sailors is the outer relation and reserves is the inner relation and the required output is projected at the end

***Query 3*:** Find names of sailors who have reserved a boat.

***Ans:*** Sort-Merge Join is done where Sailors is the outer relation and Reserves is the inner relation and the sailors’ names are projected at the end.

***Query 4:*** Find names of sailors who have reserved a boat and print each name once.

***Ans:*** Sort-Merge Join is done where Sailors is the outer relation and Reserves is the inner relation and the sailors’ names are projected at the end. We use distinct option to eliminate duplicate values to display the final result.

***Query 5:*** Find the names of old sailors or sailors with a rating less than 7, who have reserved a boat.

***Ans:*** Sort-Merge Join is done where Sailors is the outer relation and Reserves is the inner relation. Test for the given two conditions, Sailor age > 40 and rating less than 7 and project name, age and rating of records that satisfy either.

**Conclusion:**

Minibase which is a database management system consisting of several components are studied in detail. Functionalities and importance of each and every component are understood by performing several tests. These tests helped us to gain knowledge on a database management system and further modify or add additional functionalities to the existing Minibase to meet any specific requirement.

**Bibliography:**

1. <http://research.cs.wisc.edu/coral/mini_doc/minibase.html>
2. <http://www.cs.sfu.ca/CourseCentral/354/zaiane/material/notes/Chap7/node11.html>