CSE 510: Database Management Systems Implementation

Project Phase 1

Submitted by Kyle Stevens (1214241534)

Group Members Group 1:

Aryyama Kumar Jana

Sahil Santosh Patil

Neha Dandekar

Kyle Stevens

Jasmin Kantibhai Trada

Venkataramana Balaji Rajendran

Abstract

The purpose of phase 1 of this project is to introduce, reintroduce as the case might be, students to the idea of a DBMS, specifically in the case of this course the java-based DBMS Minibase. This gives students an opportunity to ensure their understanding of relational database functionality and familiarize themselves with the visualization and interface of Minibase. Through the analysis of the included Minibase tests the user also can gain an understanding of the limitations and capabilities of the Minibase DBMS and DBMS systems in general. Analysis of the Buffer Manager Test allows the user to focus on learning the operations that can be done on the buffer and what limitations exist on the buffer. This is also true of the Heap File Test, Disk Manager Test, and B+ Tree Test. The phase also provides the user with an introduction to B+ tree data structures if they are unfamiliar with it, or a refresh for those who have encountered the data structure before. This phase serves to test to ensure that the user can implement basic tests, has a general understanding of DBMS capabilities, and is able to use both the Java programming environment and a Unix-based Linux environment.

Introduction

Phase 1 of this project covers various test cases for features found in a database management system (DBMS); presenting both a live teaching tool and opportunity to refine or refresh basic database knowledge and the component of a DBMS system. Not only does this phase serve as a method of introduction to DBMS systems, but also to a specific version used for education and visualization in a universal platform in the form the Minibase java DBMS.

Minibase is a Java based single user DBMS, intended primarily for use in education and teaching. Research provides some basic knowledge of the system at a surface level, single user with no concurrency control or recovery support. It is under development by the University of Konstanz and based on the Minibase of Java written by Chris Mayfield, which was an extended and redesigned version of the initial Java port of the first C/C++ version of Minibase developed by Raghu Ramakrishnan to accompany the book Database Management Systems.

The buffer manager is responsible in this system for a few primary tasks. First, the database buffer is temporary storage in memory that is quick to access, but very limited in size. The manager is responsible for allocating space in the buffer to store data, providing block addresses when they are available and moving the block to the buffer if it is not. When there is no free space in the buffer the manager will use one of a series of replacement algorithms to decide which blocks are replaced in the buffer; least recently used(LRU) is one such common algorithm often used for buffer management. The buffer is also a key component in the systems recovery and concurrency of data in the event of a system crash.

The disk space manager works closest to the operating system of all the DBMS components since it is responsible for the management, creation, and manipulation of disk space to write the data for the database. In databases the concept of a “page” is created as a unit of data. Generally, the page size as less than that of the block size mentioned before to facilitate retrieval in one disk I/O operation. The disk manager handles the allocation of disk space, as well as certain standard practices for data storage, such as storing data accessed sequentially in contiguous block sequences. In terms of continued use, the disk manager handles the manipulation of the blocks and pages as data is created and removed from the database.

Heap file organization, at the most basic level means that files are unordered and inserted at the file's end. Records are not sorted or ordered. When a data block is full in heap file organization the new record is stored in a different block, and due to the structure, this does not have to be the next sequential block. While this works fine for small databases, such as those that Minibase mostly handles, for large databases this quickly becomes extremely inefficient; while insertion is not an extraordinary issue retrieving datafiles and records is time consuming as the entire file must be searched from the beginning for the record's unique id.

Minibase does utilize a B+ tree structure which allows for efficient insertion, deletion, and search operations. Extending the functionality of a B Tree, the B+ Tree stores records exclusively in the leaf nodes of internal nodes which contain key values. Like a directory at the mall that tells you what each section of the mall caters to. If you are shopping for clothes at Kohls, the directory tells you where you would find Kohls, but not the specific location of Kohls. Like this, a B+ Tree's internal node stores key values with pointers to the leaf nodes containing the actual records. So if you were to search for the key value 55, and your internal nodes contains the values 50 and 75, the DBMS would search the branch that exists between the 50 and 75 key nodes to find the leaf which contains that record. Insertion and deletion require some manipulation of tree structure in certain cases and this is done to maintain a key B Tree property; that a B Tree is always balanced.

Description of Tests

Buffer Manager Test(BMTest):

There are three buffer manager tests included in the automated tests of Minibase. Test 1 tests simple operations as described by the system. Analyzing the test shows that the first test allocates a number of pages equal to the number of unpinned buffers plus one, so that it can test the unpinning of all buffers, writing new data, and then unpinning the 'dirty' buffers to write back to the disk. It then finishes operations by freeing all pages that were allocated for the test.

Test 2 expects failures by trying to perform illegal operations on the buffer, including loading, or pinning, more pages to the buffer than exist free buffers. This should fail as there are not enough locations in the buffer to hold those pages. After testing the buffer bounds this test also attempts to free a pinned page with the expectation of failure as the DBMS should prevent a page that has not been written back to disk and unpinned from being freed. The final portion of this test is to try and unpin a page not held in the buffer pool; this should fail as pages that are pinned are defined as being pages that are in use and any pages not in the buffer are not in use. Operations of the test are then finished, and the system cleaned up for the next test.

Test 3 begins with a test of internals by first allocating new pages and altering them in the buffer so that they are 'dirty'; the system then attempts to unpin the pages left pinned and with dirty bits. This should take these pages and overwrite them on the disk with there new modified value. The second part of this test attempts to read data back from pages and ensure that the data is correct before unpinning the pages and completing the test.

Disk Space Management Tests(DBTest):

The tests performed for Disk Space Management build off the previous ones by generally testing in these ways. Test 1 is a test of allocating pages and files, writing to some of them, and then deallocating some pages.

Test 2 uses the database created in the previous test and performs deletions of some file entries before then looking up the file entries that should remain. Test 2 ends by reading data written to the pages in test 1.

Test 3 tests for error conditions in the disk manager, beginning with searching for deleted file entries. Next the test tries to delete entries already removed, and entries that did not exist in the database at any time. There are also testing of the look up for nonexistent files, the addition of already existing files, and boundary conditions on page allocation in terms of both name length and page runs.

Test 4 also tests boundary conditions in a very implementation specific way, beginning with attempting to allocate more pages than is space available on the disk. The tests continue to manipulate the number of entries and pages to test the boundary conditions of the systems disk space manager until another final check is done to ensure that no allocation of pages beyond those available is allowed; this includes testing boundaries at the maximum and minimum capacity on the disk.

Heap File Tests(HFTest):

The heap file is tested in this test, beginning with test 1 which performs some standard heap file operations. A heap file is first created with a set size before records are added to the file. This test ends after scanning the records and ensuring that they were all created. Test 2 begins by opening the file created prior and deleting half of the records before scanning the remaining records on the file. Test 3 finally tests the manipulation of data on the heap file by changing the records stored on the heap file and scanning to ensure that the record values were updated.

Test 4 tests error conditions for the heap file by attempting to change the size of the records without adding or deleting records; the records are first shortened, then lengthened. The final test is then ended after the system checks the error condition for record length by attempting to insert a record that is too long.

B+ Tree Test(BTTest):

The B+ Tree Test is the only test which required interaction by the user, this was done as the B+ tree is visualized by Minibase, and the user is allowed to determine the number of records to allocate and delete, while giving them experience with the concept of B+ trees. Test 1 in the system allows the user to insert **n** records in order, and this allocates n records beginning at 1 and iterating to **n**. Test 2 and test 3 follow the same format as test 1 but allocate from **n** to 1, and at random in the range (1..**n**) instead. These tests are meant to show the B+ tree format as the number of records are increased and to test and ensure proper sorting and ordering within the data structure. Test 4 and 5 are also similar in that they randomly add **n** records and randomly delete **m** records; ensuring proper addition and deletion of records which also test the reallocation of pages as the number of records are mutated. The significant difference between these two tests being that test 5 utilizes string-based keys as compared to the integer keys used previously. The utilization of string-based keys changes the way in which the records should be sorted and tests the DBMS management of keys of this type.

Index Test(IndexTest):

The index tests are simple tests to ensure that the DBMS is properly indexing records and data. Test 1 checks that the index scan returns the index sorted in order, while also ensuring that there are the proper number of records existing in the database. Throwing an erro if too few or too many records are found. Test 2 scans for specific indexes and a range of indexes to ensure that both functionalities are properly scanning the database. Test 3 does an index scan on integer keys to ensure that these are also properly found.

Join and Sort-Merge Tests(JoinTest & SM_JoinTest):

The join tests are defined as a set of queries, and as are the sort-merge tests. In the case of the join tests, there exists sailors, boats, and reserves groups and these are joined in the same database. The queries search this joined database and checks that all expected values are returned. Query 1 searches for all sailors who have reserved boat number 1 printing out their reservation dates, query 2 searches for all sailors who had reserved a boat that was red and returns them in alphabetical order. Queries 3 and 4 both search for all sailors who have reserved a boat, however query 4 prints that names of each unique sailor. Query 5 finds the names of old sailors and those with poor ratings, a score under 7, from the database. Query 6 is the last query in the join test and finds the name of all sailors who have reserved a red boat and have a rating greater than 7.

The queries for the sort-merge tests are identical to those of join, but for two exceptions; the sort-merge test does not implement queries 2 or 6.

Sort Tests(SortTest):

The sort tests are used to check the sorting capabilities and algorithms of the DBMS. The first two tests sort the records and run a check to ensure that the sorted records are the same as the expected records order. Test 3 sorts in descending order the float field, and in ascending order the int field. Test 4 tests the sorting algorithm on multiple files rather than the previous tests which only sorted one file.

Conclusion

Phase 1 of the project covered specific uses of the Minibase DBMS and tests for those specific functionalities and components of a DBMS; these included the buffer manager, disk manager, heap file management, sorting functions, and both join and sort-merge join functionalities. Beyond merely running these tests and familiarizing the user with the interface of Minibase and its visualization functions, this phase also introduced major topics and ideas that are included in the standard DBMS and that are important and key topics to understand when dealing with databases. It forced research into topics that while specific to Minibase and this project, also are of importance to dealing with databases in general. The research done and study done when viewing the various tests also served to present the limitations and capabilities of the various features of not only Minibase, but DBMS software in general.

Bibliography

Raghu Ramakrishnan, Minibase, The Minibase Home Page, 1996

https://research.cs.wisc.edu/coral/minibase/minibase.html

JavaTPoint, B+ Tree, JavaTPoint, 2018

https://www.javatpoint.com/b-plus-tree/

JavaTPoint, DBMS B+ Tree, JavaTPoint, 2018

https://www.javatpoint.com/dbms-b-plus-tree/

Sagnik Adhikary, Introduction of B Tree, GeeksForGeeks, 2020

https://www.geeksforgeeks.org/introduction-of-b-tree/

JavaTPoint, DBMS Heap file organization, JavaTPoint, 2018

https://www.javatpoint.com/dbms-heap-file-organization/

JavaTPoint, Database Buffer, JavaTPoint, 2018

https://www.javatpoint.com/database-buffer/

DBMSInternals, Disk Space Management And Pages, DBMSInternals, 2020

http://www.dbmsinternals.com/database-fundamentals/data-storage/disk-space-management-pages/

Appendix

```
Script started on 2021-01-29 22:50:17-0700

[37;1mkyle-stevens@[35msrc[0m$ make test

cd tests; make bmtest dbtest; whoami; make hftest bttest indextest jointest sorttest sortmerge

make[1]: Entering directory
'/mnt/c/Users/kyleb/Desktop/GITHUB_REPOS/CSE_510_SPRING_21/minjava/javaminibase/sr
c/tests'

/usr/lib/jvm/java-11-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java BMTest.java

/usr/lib/jvm/java-11-openjdk-amd64/bin/java  -classpath .:.. tests.BMTest


Running Buffer Management tests....

Replacer: Clock




  Test 1 does a simple test of normal buffer manager operations:

  - Allocate a bunch of new pages

  - Write something on each one

  - Read that something back from each one

   (because we're buffering, this is where most of the writes happen)

  - Free the pages again

  Test 1 completed successfully.


  Test 2 exercises some illegal buffer manager operations:

  - Try to pin more pages than there are frames

*** Pinning too many pages

  --> Failed as expected


  - Try to free a doubly-pinned page

*** Freeing a pinned page
```

```
  --> Failed as expected


  - Try to unpin a page not in the buffer pool
*** Unpinning a page not in the buffer pool
  --> Failed as expected


  Test 2 completed successfully.


  Test 3 exercises some of the internals of the buffer manager
  - Allocate and dirty some new pages, one at a time, and leave some pinned
  - Read the pages
  Test 3 completed successfully.


...Buffer Management tests completely successfully.


/usr/lib/jvm/java-11-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java DBTest.java
/usr/lib/jvm/java-11-openjdk-amd64/bin/java  -classpath .:.. tests.DBTest


Running Disk Space Management tests....


Replacer: Clock




  Test 1 creates a new database and does some tests of normal operations:
  - Add some file entries
  - Allocate a run of pages
  - Write something on some of them
  - Deallocate the rest of them
  Test 1 completed successfully.


  Test 2 opens the database created in test 1 and does some further tests:
  - Delete some of the file entries
  - Look up file entries that should still be there
  - Read stuff back from pages we wrote in test 1
  Test 2 completed successfully.


  Test 3 tests for some error conditions:
  - Look up a deleted file entry
**** Looking up a deleted file entry
  --> Failed as expected


  - Try to delete a deleted entry again
**** Delete a deleted file entry again
  --> Failed as expected


  - Try to delete a nonexistent file entry
**** Deleting a nonexistent file entry
  --> Failed as expected


  - Look up a nonexistent file entry
**** Looking up a nonexistent file entry
  --> Failed as expected


  - Try to add a file entry that's already there
**** Adding a duplicate file entry
  --> Failed as expected
  - Try to add a file entry whose name is too long
**** Adding a file entry with too long a name
  --> Failed as expected

  - Try to allocate a run of pages that's too long
**** Allocating a run that's too long
  --> Failed as expected

  - Try to allocate a negative run of pages
**** Allocating a negative run
  --> Failed as expected

  - Try to deallocate a negative run of pages
**** Deallocating a negative run
  --> Failed as expected

  Test 3 completed successfully.

  Test 4 tests some boundary conditions.
    (These tests are very implementation-specific.)
  - Make sure no pages are pinned
  - Allocate all pages remaining after DB overhead is accounted for
  - Attempt to allocate one more page
**** Allocating one additional page
  --> Failed as expected

  - Free some of the allocated pages
  - Allocate some of the just-freed pages
  - Free two continued run of the allocated pages
  - Allocate back number of pages equal to the just freed pages

  - Add enough file entries that the directory must surpass a page
  - Make sure that the directory has taken up an extra page: try to
    allocate more pages than should be available
**** Allocating more pages than are now available
   --> Failed as expected

  - At this point, all pages should be claimed.  Try to allocateone more.
**** Allocating one more page than there is
   --> Failed as expected

  - Free the last two pages: this tests a boundary condition in the space map.
  Test 4 completed successfully.

...Disk Space Management tests completely successfully.

make[1]: Leaving directory
'/mnt/c/Users/kyleb/Desktop/GITHUB_REPOS/CSE_510_SPRING_21/minjava/javaminibase/sr
c/tests'
kyle-stevens
make[1]: Entering directory
'/mnt/c/Users/kyleb/Desktop/GITHUB_REPOS/CSE_510_SPRING_21/minjava/javaminibase/sr
c/tests'
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java HFTest.java
/usr/lib/jvm/java-11-openjdk-amd64/bin/java  -classpath .:.. tests.HFTest

Running Heap File tests....
```

```
Replacer: Clock

 Test 1: Insert and scan fixed-size records
 - Create a heap file
 - Add 100 records to the file
 - Scan the records just inserted
 Test 1 completed successfully.

 Test 2: Delete fixed-size records
 - Open the same heap file as test 1
 - Delete half the records
 - Scan the remaining records
 Test 2 completed successfully.

 Test 3: Update fixed-size records
```

```
  - Open the same heap file as tests 1 and 2

  - Change the records

  - Check that the updates are really there

  Test 3 completed successfully.


  Test 4: Test some error conditions

  - Try to change the size of a record

**** Shortening a record
  --> Failed as expected

**** Lengthening a record
  --> Failed as expected

  - Try to insert a record that's too long

**** Inserting a too-long record
  --> Failed as expected

  Test 4 completed successfully.


...Heap File tests completely successfully.
```

```
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java BTTest.java
/usr/lib/jvm/java-11-openjdk-amd64/bin/java  -classpath .:.. tests.BTTest
Replacer: Clock


Running  tests....

 **************** The file name is: AAA0  **********
------------------------- MENU ------------------



[0]   Naive delete (new file)
[1]   Full delete(Default) (new file)

[2]   Print the B+ Tree Structure
[3]   Print All Leaf Pages
[4]   Choose a Page to Print

        ---Integer Key (for choices [6]-[14]) ---

[5]   Insert a Record
[6]   Delete a Record
[7]   Test1 (new file): insert n records in order
[8]   Test2 (new file): insert n records in reverse order
[9]   Test3 (new file): insert n records in random order
[10]  Test4 (new file): insert n records in random order
    and delete m records randomly
```

```
[11]  Delete some records


[12]  Initialize a Scan
[13]  Scan the next Record
[14]  Delete the just-scanned record


          ---String Key (for choice [15]) ---


[15]  Test5 (new file): insert n records in random order
       and delete m records randomly.


[16]  Close the file
[17]  Open which file (input an integer for the file name):
[18]  Destroy which file (input an integer for the file name):


[19]  Quit!
Hi, make your choice :2
The Tree is Empty!!!
------------------------- MENU -----------------



[0]   Naive delete (new file)
[1]   Full delete(Default) (new file)


[2]   Print the B+ Tree Structure
[3]   Print All Leaf Pages
[4]   Choose a Page to Print
```

```
        ---Integer Key (for choices [6]-[14]) ---

[5]   Insert a Record
[6]   Delete a Record
[7]   Test1 (new file): insert n records in order
[8]   Test2 (new file): insert n records in reverse order
[9]   Test3 (new file): insert n records in random order
[10]  Test4 (new file): insert n records in random order
    and delete m records randomly
[11]  Delete some records

[12]  Initialize a Scan
[13]  Scan the next Record
[14]  Delete the just-scanned record

        ---String Key (for choice [15]) ---

[15]  Test5 (new file): insert n records in random order
     and delete m records randomly.

[16]  Close the file
[17]  Open which file (input an integer for the file name):
[18]  Destroy which file (input an integer for the file name):

[19]  Quit!
Hi, make your choice :7
Please input the number of keys to insert:
200
```

```
 **************** The file name is: AAA1  **********
------------------------- MENU -----------------



[0]   Naive delete (new file)
[1]   Full delete(Default) (new file)

[2]   Print the B+ Tree Structure
[3]   Print All Leaf Pages
[4]   Choose a Page to Print

          ---Integer Key (for choices [6]-[14]) ---

[5]   Insert a Record
[6]   Delete a Record
[7]   Test1 (new file): insert n records in order
[8]   Test2 (new file): insert n records in reverse order
[9]   Test3 (new file): insert n records in random order
[10]  Test4 (new file): insert n records in random order
      and delete m records randomly
[11]  Delete some records

[12]  Initialize a Scan
[13]  Scan the next Record
[14]  Delete the just-scanned record

          ---String Key (for choice [15]) ---
```

```
[15]  Test5 (new file): insert n records in random order
        and delete m records randomly.

[16]  Close the file
[17]  Open which file (input an integer for the file name):
[18]  Destroy which file (input an integer for the file name):

[19]  Quit!
Hi, make your choice :2



---------------The B+ Tree Structure---------------
1     6
2           4
2           5
2           7
2           8
2           9
2           10
--------------- End ---------------


-------------------------- MENU ------------------


[0]   Naive delete (new file)
[1]   Full delete(Default) (new file)
```

[2]   Print the B+ Tree Structure

[3]   Print All Leaf Pages

[4]   Choose a Page to Print

---Integer Key (for choices [6]-[14]) ---

[5]   Insert a Record

[6]   Delete a Record

[7]   Test1 (new file): insert n records in order

[8]   Test2 (new file): insert n records in reverse order

[9]   Test3 (new file): insert n records in random order

[10] Test4 (new file): insert n records in random order
and delete m records randomly

[11] Delete some records

[12] Initialize a Scan

[13] Scan the next Record

[14] Delete the just-scanned record

---String Key (for choice [15]) ---

[15] Test5 (new file): insert n records in random order
and delete m records randomly.

[16] Close the file

[17] Open which file (input an integer for the file name):

[18] Destroy which file (input an integer for the file name):

```
[19]  Quit!
Hi, make your choice :8
Please input the number of keys to insert:
100
 **************** The file name is: AAA2  **********
------------------------- MENU -----------------



[0]   Naive delete (new file)
[1]   Full delete(Default) (new file)

[2]   Print the B+ Tree Structure
[3]   Print All Leaf Pages
[4]   Choose a Page to Print

        ---Integer Key (for choices [6]-[14]) ---

[5]   Insert a Record
[6]   Delete a Record
[7]   Test1 (new file): insert n records in order
[8]   Test2 (new file): insert n records in reverse order
[9]   Test3 (new file): insert n records in random order
[10]  Test4 (new file): insert n records in random order
    and delete m records randomly
[11]  Delete some records

[12]  Initialize a Scan
```

```
[13]  Scan the next Record
[14]  Delete the just-scanned record

          ---String Key (for choice [15]) ---

[15]  Test5 (new file): insert n records in random order
      and delete m records randomly.

[16]  Close the file
[17]  Open which file (input an integer for the file name):
[18]  Destroy which file (input an integer for the file name):

[19]  Quit!
Hi, make your choice :2



---------------The B+ Tree Structure---------------
1    14
2         12
2         15
2         13
--------------- End ---------------


-------------------------- MENU ------------------
```

```
[0]   Naive delete (new file)
[1]   Full delete(Default) (new file)

[2]   Print the B+ Tree Structure
[3]   Print All Leaf Pages
[4]   Choose a Page to Print

            ---Integer Key (for choices [6]-[14]) ---

[5]   Insert a Record
[6]   Delete a Record
[7]   Test1 (new file): insert n records in order
[8]   Test2 (new file): insert n records in reverse order
[9]   Test3 (new file): insert n records in random order
[10]  Test4 (new file): insert n records in random order
      and delete m records randomly
[11]  Delete some records

[12]  Initialize a Scan
[13]  Scan the next Record
[14]  Delete the just-scanned record

            ---String Key (for choice [15]) ---

[15]  Test5 (new file): insert n records in random order
        and delete m records randomly.

[16]  Close the file
```

```
[17]  Open which file (input an integer for the file name):
[18]  Destroy which file (input an integer for the file name):

[19]  Quit!
Hi, make your choice :9
Please input the number of keys to insert:
50
 ***************** The file name is: AAA3  **********
------------------------- MENU -----------------



[0]   Naive delete (new file)
[1]   Full delete(Default) (new file)

[2]   Print the B+ Tree Structure
[3]   Print All Leaf Pages
[4]   Choose a Page to Print

        ---Integer Key (for choices [6]-[14]) ---

[5]   Insert a Record
[6]   Delete a Record
[7]   Test1 (new file): insert n records in order
[8]   Test2 (new file): insert n records in reverse order
[9]   Test3 (new file): insert n records in random order
[10]  Test4 (new file): insert n records in random order
    and delete m records randomly
[11]  Delete some records
```

```
[12]  Initialize a Scan
[13]  Scan the next Record
[14]  Delete the just-scanned record

          ---String Key (for choice [15]) ---

[15]  Test5 (new file): insert n records in random order
       and delete m records randomly.

[16]  Close the file
[17]  Open which file (input an integer for the file name):
[18]  Destroy which file (input an integer for the file name):

[19]  Quit!
Hi, make your choice :10
Please input the number of keys to insert:
200
Please input the number of keys to delete:
400
 ***************** The file name is: AAA4  **********
-------------------------- MENU ------------------



[0]   Naive delete (new file)
[1]   Full delete(Default) (new file)

[2]   Print the B+ Tree Structure
```

[3]   Print All Leaf Pages

[4]   Choose a Page to Print

---Integer Key (for choices [6]-[14]) ---

[5]   Insert a Record

[6]   Delete a Record

[7]   Test1 (new file): insert n records in order

[8]   Test2 (new file): insert n records in reverse order

[9]   Test3 (new file): insert n records in random order

[10] Test4 (new file): insert n records in random order
and delete m records randomly

[11] Delete some records

[12] Initialize a Scan

[13] Scan the next Record

[14] Delete the just-scanned record

---String Key (for choice [15]) ---

[15] Test5 (new file): insert n records in random order
and delete m records randomly.

[16] Close the file

[17] Open which file (input an integer for the file name):

[18] Destroy which file (input an integer for the file name):

[19] Quit!

```
Hi, make your choice :2

The Tree is Empty!!!

-------------------------- MENU ------------------



[0]   Naive delete (new file)

[1]   Full delete(Default) (new file)


[2]   Print the B+ Tree Structure

[3]   Print All Leaf Pages

[4]   Choose a Page to Print


        ---Integer Key (for choices [6]-[14]) ---


[5]   Insert a Record

[6]   Delete a Record

[7]   Test1 (new file): insert n records in order

[8]   Test2 (new file): insert n records in reverse order

[9]   Test3 (new file): insert n records in random order

[10]  Test4 (new file): insert n records in random order

    and delete m records randomly

[11]  Delete some records


[12]  Initialize a Scan

[13]  Scan the next Record

[14]  Delete the just-scanned record


        ---String Key (for choice [15]) ---
```

```
[15]  Test5 (new file): insert n records in random order
      and delete m records randomly.

[16]  Close the file
[17]  Open which file (input an integer for the file name):
[18]  Destroy which file (input an integer for the file name):

[19]  Quit!
Hi, make your choice :12
Please input the LOWER integer key (null if -3):
0
Please input the HIGHER integer key (null if -2):
150
------------------------- MENU ------------------



[0]   Naive delete (new file)
[1]   Full delete(Default) (new file)

[2]   Print the B+ Tree Structure
[3]   Print All Leaf Pages
[4]   Choose a Page to Print

        ---Integer Key (for choices [6]-[14]) ---

[5]   Insert a Record
[6]   Delete a Record
```

```
[7]   Test1 (new file): insert n records in order
[8]   Test2 (new file): insert n records in reverse order
[9]   Test3 (new file): insert n records in random order
[10]  Test4 (new file): insert n records in random order
      and delete m records randomly
[11]  Delete some records

[12]  Initialize a Scan
[13]  Scan the next Record
[14]  Delete the just-scanned record

          ---String Key (for choice [15]) ---

[15]  Test5 (new file): insert n records in random order
        and delete m records randomly.

[16]  Close the file
[17]  Open which file (input an integer for the file name):
[18]  Destroy which file (input an integer for the file name):

[19]  Quit!
Hi, make your choice :19

... Finished .
```

```
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java IndexTest.java
/usr/lib/jvm/java-11-openjdk-amd64/bin/java  -classpath .:.. tests.IndexTest
```

```
Running Index tests....

Replacer: Clock

------------------------ TEST 1 --------------------------
BTreeIndex created successfully.

BTreeIndex file created successfully.

Test1 -- Index Scan OK
------------------- TEST 1 completed ---------------------

------------------------ TEST 2 --------------------------
BTreeIndex opened successfully.

Test2 -- Index Scan OK
------------------- TEST 2 completed ---------------------

------------------------ TEST 3 --------------------------
BTreeIndex created successfully.

BTreeIndex file created successfully.

Test3 -- Index scan on int key OK

------------------- TEST 3 completed ---------------------
```

```
...Index tests
completely successfully
.



Index tests completed successfully
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java JoinTest.java
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
/usr/lib/jvm/java-11-openjdk-amd64/bin/java  -classpath .:.. tests.JoinTest
Replacer: Clock




Any resemblance of persons in this database to people living or dead
is purely coincidental. The contents of this database do not reflect
the views of the University, the Computer  Sciences Department or the
developers...

**********************Query1 strating *********************
Query: Find the names of sailors who have reserved boat number 1.
       and print out the date of reservation.

  SELECT S.sname, R.date
  FROM   Sailors S, Reserves R
  WHERE  S.sid = R.sid AND R.bid = 1
```

```
(Tests FileScan, Projection, and Sort-Merge Join)
[Mike Carey, 05/10/95]
[David Dewitt, 05/11/95]
[Jeff Naughton, 05/12/95]

Query1 completed successfully!
*******************Query1 finished!!!*****************




**********************Query2 strating *********************
Query: Find the names of sailors who have reserved a red boat
       and return them in alphabetical order.

  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
Plan used:
 Sort (Pi(sname) (Sigma(B.color='red')  |><|  Pi(sname, bid) (S  |><|  R)))

(Tests File scan, Index scan ,Projection,  index selection,
 sort and simple nested-loop join.)

After Building btree index on sailors.sid.

[David Dewitt]
```

```
[Mike Carey]
[Raghu Ramakrishnan]
[Yannis Ioannidis]

Query2 completed successfully!
******************Query2 finished!!!****************




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

  SELECT S.sname
  FROM   Sailors S, Reserves R
  WHERE  S.sid = R.sid

(Tests FileScan, Projection, and SortMerge Join.)

[Mike Carey]
[Mike Carey]
[Mike Carey]
[David Dewitt]
[David Dewitt]
[Jeff Naughton]
[Miron Livny]
[Yannis Ioannidis]
[Raghu Ramakrishnan]
[Raghu Ramakrishnan]
```

```
Query3 completed successfully!
******************Query3 finished!!!*****************



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

  SELECT DISTINCT S.sname
  FROM   Sailors S, Reserves R
  WHERE  S.sid = R.sid

(Tests FileScan, Projection, Sort-Merge Join and Duplication elimination.)

[David Dewitt]
[Jeff Naughton]
[Mike Carey]
[Miron Livny]
[Raghu Ramakrishnan]
[Yannis Ioannidis]

Query4 completed successfully!
******************Query4 finished!!!*****************
```

```
*********************Query5 strating ********************
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).

  SELECT S.sname, S.rating, S.age
  FROM   Sailors S, Reserves R
  WHERE  S.sid = R.sid and (S.age > 40 || S.rating < 7)

(Tests FileScan, Multiple Selection, Projection, and Sort-Merge Join.)

[Mike Carey, 9, 40.3]
[Mike Carey, 9, 40.3]
[Mike Carey, 9, 40.3]
[David Dewitt, 10, 47.2]
[David Dewitt, 10, 47.2]
[Jeff Naughton, 5, 35.0]
[Yannis Ioannidis, 8, 40.2]

Query5 completed successfully!
******************Query5 finished!!!****************


*********************Query6 strating ********************
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.
  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

Plan used:
 Sort(Pi(sname) (Sigma(B.color='red')  |><|  Pi(sname, bid) (Sigma(S.rating > 7)  |><|  R)))

(Tests FileScan, Multiple Selection, Projection,sort and nested-loop join.)

After nested loop join S.sid|><|R.sid.
After nested loop join R.bid|><|B.bid AND B.color=red.
After sorting the output tuples.
[David Dewitt]
[Mike Carey]
[Raghu Ramakrishnan]
[Yannis Ioannidis]

Query6 completed successfully!
*******************Query6 finished!!!*****************




Finished joins testing
join tests completed successfully
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java SortTest.java
/usr/lib/jvm/java-11-openjdk-amd64/bin/java  -classpath .:.. tests.SortTest
```

```
Running Sort tests....

Replacer: Clock

------------------------ TEST 1 --------------------------
Test1 -- Sorting OK
------------------- TEST 1 completed ---------------------

------------------------ TEST 2 --------------------------
Test2 -- Sorting OK
------------------- TEST 2 completed ---------------------

------------------------ TEST 3 --------------------------
 -- Sorting in ascending order on the int field --
Test3 -- Sorting of int field OK

 -- Sorting in descending order on the float field --
Test3 -- Sorting of float field OK

------------------- TEST 3 completed ---------------------

------------------------ TEST 4 --------------------------
Test4 -- Sorting OK
------------------- TEST 4 completed ---------------------


...Sort tests
```

```
completely successfully

.



Sorting tests completed successfully
/usr/lib/jvm/java-11-openjdk-amd64/bin/javac -classpath .:.. SM_JoinTest.java TestDriver.java
Note: SM_JoinTest.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
/usr/lib/jvm/java-11-openjdk-amd64/bin/java  -classpath .:.. tests.SM_JoinTest
Replacer: Clock



Any resemblance of persons in this database to people living or dead
is purely coincidental. The contents of this database do not reflect
the views of the University, the Computer  Sciences Department or th
```