DBMSI PROJECT - PHASE 1

GROUP MEMBERS - GROUP 3

- Ganesh Ashok Yallankar
- Sreenivasan Ramesh
- Rakesh Ramesh
- Sumukh Ashwin Kamath
- Baani Khurana
- Kanishk Bashyam

ABSTRACT

The world of data is constantly changing and evolving every second. This, in turn, has created a completely new dimension of growth and challenges for companies around the globe. By accurately recording data, updating and tracking them on an efficient and regular basis, companies can address their challenges on one hand and make use of the immense potential offered by this sector on the other hand, A database management system provides efficient techniques to solve the problems. In this phase of the project, we set up the minibase database management system and perform various test cases comprising of database operations. Here, we get an insight into the implementation of database management systems.

INTRODUCTION

Minibase is a database management system. It is based on Relational Database management systems. It has a parser, optimizer, buffer pool manager, storage mechanisms (heap files, secondary indexes based on B+ Trees), and a disk space management system. It is intended for educational use. A Relational Database management system (RDBMS) is a database management system (DBMS) that is based on the relational model. A Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables). A relation, also known as a table or file, is a subset of the Cartesian product of a list of domains characterized by a name. And within a table, each row represents a group of related data values. A row, or record, is also known as a tuple. The columns in a table is a field and are also referred to as an attribute. You can also think of it this way: an attribute is used to define the record and a record contains a set of attributes. We are testing various test case scenarios for the minibase system implementation.

Terminologies:
Database: It is a collection of related data [1].
Attribute: Each column in a Table. Attributes are the properties that define a relation[2].
Tables – In the Relational model, relations are saved in the table format. It is stored along with its entities. A table has two properties rows and columns. Rows represent records and columns represent attributes[2].
Tuple – It is nothing but a single row of a table, which contains a single record[2].
Heap Files: A heap file is an unordered set of records [3].
SQL: It is a standard language for storing, manipulating and retrieving data in databases [4].
Buffer Manager: The buffer manager reads disk pages into the main memory page as needed [5].
Parser: A parser is a software component that takes input data (frequently text) and builds a data structure – often some kind of parse tree, abstract syntax tree or other hierarchical structure, giving a structural representation of the input while checking for correct syntax [6].
Optimizer: The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans [7].

DESCRIPTION OF TESTS

Buffer Manager Tests

Test 1

This Test checks for some simple normal buffer manager operations.
Steps implemented in the test:

1. Ensure at least one page is written during this test.
2. Get the number of unpinned buffer frames.
3. Allocate new pages in the database based on the count of unpinned buffer frames.
4. Write data into every allocated page.
   - Pin allocated page.
   - Write the data onto the page (data = page_id + 99999).
   - Unpin the page.
   - Repeat the above steps for all the allocated pages.
5. Validate the data stored in the pages.
   - Pin the page.
   - Read the content of the page.
   - Validate the content of the page (data == page_id + 99999).
   - Unpin the page.
   - Repeat the above steps for all the allocated pages.
6. Free up all the pages.

Test 2

This performs some illegal buffer manager operations and tests whether the illegal operation can be caught.
Steps:

1. Try to pin more pages than the count of frames.
   - Set number of pages to be one more than the count of frames
     (num_of_pages = num_of_unpinned_buffer_frames + 1).
   - Allocate and pin the pages as per the above count.
   - Finally, Pinning the last allocated page fails as expected.
2. Try to free a doubly pinned page.
   - Pin the first page from step 1 for the second time.
   - Freeing the first page fails due to doubly pinning and the step fails as expected.
3. Try to unpin a page not present in the buffer pool.
   - Unpin the last allocated page from step1 which is already unpinned.
   - The operation fails as the page is not in the buffer pool.

Test 3

Performs some internals of buffer manager operations.
Steps involved are:

- Allocate some new pages (number of pages = 60) and dirty them(data = page_id + 99999). All the pages are pinned.
- Unpin the pages if the condition satisfies - "page_id % 20 != 12".
- Pin all the pages again, this results in some doubly pinned pages.
- All the pages data is read and validated.
- Unpin all the pages.
- Unpin the pages if the condition satisfies - "page_id % 20 == 12".

DB Tests

Test 1

This test creates a new database and does operations such as allocate page, add file entry, writing data, and deallocate at the end. This helps to test the normal disk space management mechanism.
Procedure:

- Allocate 6 pages and add file entries with file names as "file0" - "file5".
- Next, Allocate 30 pages and write data into 20 of them. Where data will be of the form "A0" - "A19".
- Deallocate the remaining 10 pages.

Test 2

This test opens the database created in test 1 and does additional tests namely - delete few file entries, lookup a few existing file entries and read and validate the data written in test 1.
Procedure:

- Delete the first 3 file entries (0,1,2) created in Test1.
- Look up the next 3 file entries (i.e., 3,4,5).
- Read and verify if the contents of 20 pages written in Test1 match with "A0" - "A19".
- Deallocate all remanining allocated pages.

Test 3

This step tests for error conditions by performing illegal operations.
Procedure:

1. Lookup for a deleted file entry.
   - Lookup for the "file1" entry which was deleted in the previous test - Test2. This fails and throws an error as expected and returns a Null value for page_id.

2. Delete a previously deleted entry.
    - Delete "file1" entry which was already deleted in Test2. This step fails and throws a "FileEntryNotFoundException" exception.
3. Delete a non-existent file entry.
    - Try to delete a file with name "blargle".
    - Fails with "FileEntryNotFoundException" exception as expected.
4. Lookup for non-existent file entry.
    - Lookup for file entry with name "blargle".
    - Returns page_id as Null since file entry does not exist.
5. Add a redundant file entry.
    - Try to add a file entry with the name "file3".
    - This throws the exception "DuplicateEntryException" as the file already exists (created in test1).
6. Add a file entry with a longer name.
    - Create a byte array that is big enough to fail the test. This will have the size MAXSIZE + 5 (=55).
    - Since the allowed size is MAXSIZE(=50), fails with "FileNameTooLongException" exception.
7. Try to allocate a run of pages that's too long.
    - Try to allocate a run of 9000 pages.
    - This fails with the "OutOfSpaceException" exception.
8. Try to allocate a negative run of pages.
    - Try to allocate a run of -10 pages.
    - This fails with "InvalidRunSizeException" exception.
9. Try to deallocate a negative run of pages.
    - Deallocate -10 pages.
    - This fails with "InvalidRunSizeException" exception.

Test 4

This test checks for some boundary conditions.

Procedure:

1. Make sure none of the pages are pinned by comparing the count of unpinned buffers and the total number of buffers.
2. Create a database big enough that requires 2 pages to hold its space map.
3. Allocate a run size slightly less than database size (i.e., DB size -3).
4. Try to allocate one more page, which fails as expected.
5. Free up few allocated pages - from 3 to 9 and from 33-40.
6. Allocate pages from 33-40 now.
7. Deallocate run of pages from 11-17 and 18-28.

8. Reallocate the previously freed pages.
9. Delete leftover entries that is 3,4,5.
10. Add file entries more than page count to surpass (20 + 1).
11. The above step fails to add the last entry.
12. Try to allocate more pages than available. (6+1).
13. The above step fails with the "OutOfSpaceException" exception.
14. Now try to allocate more pages, this step fails as all are already allocated.
15. Try to deallocate the last two pages, this step fails as they are pinned.

Heap File Tests

Test 1

This test involves Inserting and scanning fixed-size records in a heap file.

Steps:

- Create a heap file - "file_1".
- A record with fields of type integer, float, and string (ival, fval, and name) is created. 100 records of such type are inserted into the heap file - file_1.
- Ensure the number of unpinned buffers is the same as the total number of the buffer.
- Scan each record from the heap and validate all 3 fields in the record for consistency. Checks are made to catch errors.

Test 2

We delete a fixed size of records from the heap file in this test.

Steps:

- Open the heap file created in Test 1.
- Delete half the elements created in test 1. This step deletes every odd-numbered element in the heap (i % 2 == 1).
- Ensure all deleted records are unpinned by comparing unpinned buffer count with total buffer count.
- Scan and read every even-numbered record and check the stored data for consistency.

Test 3

We update the data in a fixed number of records.

Steps:

- Open the heap file created in Test 1.
- For every even-numbered record, Scan the record and modify the value of fval to 7*i where i is the record_id.

- Verify if all modified records are unpinned by comparing unpinned buffer count with total buffer count.
- Scan and validate the data of every modified record.

Test 4

This test involves performing error conditions such as illegal operations on the heap.

Steps:

- Open the heap file modified in test 3.
  1. Read the first record and try to shorten its length by 1.
     This step fails as expected with "heap.InvalidUpdateException".
  2. Read the first record and try to increase the length by 1.
     This step fails as expected with "heap.InvalidUpdateException".
  3. Inserting a record with a length more than the limit.
     - Create a new record of size "MINIBASE_PAGESIZE+4" (1024 + 4 bytes).
     - Try to insert the above-created record, which results in the “SpaceNotAvailableException" exception as expected.

B Tree Tests

Options:

Option 0:

This step creates a new B Tree file with naming "AAA" and an integer.
The file is created with naive delete preference.

Option 1:

This step creates a new B Tree file with naming "AAA" and an integer.
The file is created with full delete preference.

Option 2:

This option prints the whole B tree file.

- The first column represents the height of the tree.
- Subsequent columns represent the page id of the nodes at the levels.

Option 3:

- This option prints all leaf pages of the B tree.

Option 4:

- Reads an input for page id.

- Verify if entered page id is a valid non-negative number.
- Page id will be an internal or leaf node.
- Print the data of input page id.

Option 5:
- Reads an input Integer key to be inserted into the file.
- Verify if the input is a non-negative number.
- Generate record id and insert it into the file.

Option 6:
- This option deletes a key from the file.
- Reads an input key to delete.
- Verify if the input is a non-negative number.
- Generate record id and delete the key from the file.
- Handle all validations and exceptions.

Option 7:
- This step creates a new file and inserts the “n" number of records as per user input.
- Read the number of keys to be inserted in a new file.
- Verify if the input is a positive number.
- Close the existing file. Open the file with the name "AAA" and an integer suffix.
- For the range of 0 to "n", generate new record id and insert in order into a new file created in the previous step.
- Catch exceptions and errors.

Option 8:
- This step creates a new file and inserts the “n" number of records as per user input.
- Read the number of keys to be inserted in a new file.
- Verify if the input is a positive number.
- Close the existing file. Open the file with the name "AAA" and an integer suffix.
- For the range of 0 to "n", generate new record id and insert in reverse order into a new file created in the previous step.
- Catch exceptions and errors.

Option 9:
- This step creates a new file and inserts the “n" number of random records as per user input.
- Read the number of keys to be inserted in a new file.
- Verify if the input is a positive number.

- Close the existing file. Open a file with the name "AAA" and an integer suffix.
- For "n" random numbers, generate new record id and insert in order into a new file created in the previous step.
- Catch exceptions and errors.

Option 10:

- This step creates a new file and inserts the “n" number of random records as per user input and deletes "m" records in random order.
- Read the number of keys to be inserted in a new file.
- Verify if input "n" is a positive number.
- Read the number of keys to be deleted from the file.
- Verify if input "m" is a non-negative number.
- Close the existing file. Open the file with the name "AAA" and an integer suffix.
- For "n" random numbers, generate new record id and insert in order into a new file created in the previous step.
- Remove "m" records from the file in random order.
- Catch exceptions and errors.

Option 11:

- This step reads the range from the user and deletes records inside the range.
- Read the lower key and higher key of range from the user.
- Verify if the numbers are non-negative.
- Iterate the range between the lower key and higher key, delete the corresponding record in the file.

Option 12:

- This step creates a scan for a given input range.
- Read the lower key and higher key of range from the user.
- Verify if the inputs are non-negative.
- Scan and retrieve all records in the given range.

Option 13:

- This step verifies the scan from the previous option.
- Prints the next entry key and data in the scan.
- If the scan is at last element, this prints an output message - "AT THE END OF SCAN!"

Option 14:

- This step verifies the deletion on the B tree file.
- Deletes a current record in the scan from option 12.

- If there are no records to delete, it'll display an error message.

Option 15:

- This step creates a new file and inserts the "n" number of random records as per user input and deletes "m" records in random order.
- Read the number of keys to be inserted in a new file.
- Verify if input "n" is a positive number.
- Read the number of keys to be deleted from the file.
- Verify if input "m" is a non-negative number.
- Close the existing file. Open the file with the name "AAA" and an integer suffix.
- For "n" random numbers, generate a string key, generate new record id and insert in order into a new file created in the previous step.
- Remove "m" records from the file in random order. Here keys have string values.
- Catch exceptions and errors.

Option 16:

- This step validates the closing of the file.
- Currently, the open file is closed and appropriate error handling is done.

Option 17:

- This step is closing the previous file and creating a new B tree file.
- Read input from the user say "n".
- Close the existing file, and Opens the file with the name "AAA" with "n" as the suffix.
- Catch exceptions and errors.

Option 18:

- This step closes the previous file and deletes the B tree file.
- Read input from the user say "n".
- Close the existing open file. Delete the file with name - "AAA" and "n" as the suffix.
- Handle errors and exceptions.

Option 19:

- This option exits from the interactive user interface.

Index Tests

Test 1

Here, we create and scan a B tree index file.

Steps:

- Create a heap file - "test1.in".
- Tuples are generated using the array "data1" and inserted into the heap file.
- Create a B tree index file.
- Create a scan on the heap file and insert the record ID into the B tree for each record in the heap file.
- Now scan the B tree by index and verify the order and data consistency.

Test 2

This test involves verifying index scanning operation on a B tree index file.

Steps:

- Identity search is done for the data - "dsilva" in the B tree index to match the given name. The B tree index created in the previous test is used here.
- Verification for the consistency of the data is done.
- Range scan is done for names from "dsilva" to "yuc".
- A new index scan for the above range is being done and validated based on actual "data1" array for consistency.

Test 3

We create an index file and perform scanning on the index file.

Steps:

- Create a new heap file - "test3.in".
- Create a tuple of data - string (data1[i%NUM_RECORDS), integer (inum%1000), and float (0). Insert into the heap file.
- Create a new B tree index on the integer field of the heap file.
- Now a range index scan from 100 to 900 is performed on the B tree index and validates the data for consistency.

Join Tests

Query 1

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

SQL Equivalent: 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.

Steps:

- Create a query condition with boat number 1 (R.bid = 1).
- Create a file scan for "sailors.in" as "am" and for "reserves.in" as "am2".
- Use "SortMerge" for sort and merge operations to get all the records matching query condition.
- Project the columns "sname" and "date" from "sm" (SortMerge) of the previous step.

Query 2

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

SQL Equivalent: 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;

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

Steps:

- Create Btree index on "sailors.sid" from heap file "sailers.in".
- Nested loop joins on heap file "reserves.in" and Btree index from the step above.
- Nested loop join on heap file "boats.in" on condition color = 'red'.
- The results from previous steps are joined based on "sid" and "sname" and sorted in ascending order.

Query 3

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

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

Tests FileScan, Projection, and SortMerge Join.

Steps:

- Create a file scan for "sailors.in" as "am" and for "reserves.in" as "am2".
- Use "SortMerge" for sort and merge operations to get all sailors who have reserved boats.
- Project the column "sname" from "sm" (SortMerge).

Query 4

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

SQL Equivalent: SELECT DISTINCT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid;

Tests FileScan, Projection, Sort-Merge Join, and duplication elimination.

Steps:

- Create a file scan for "sailors.in" as "am" and for "reserves.in" as "am2".
- Use "SortMerge" for sort and merge operations to get all the records matching the query conditions.
- Project the column "sname". Redundant names are removed from the projected names.

Query 5

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

SQL Equivalent: 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.

Steps:

- Create a query condition - (S.age > 40 || S.rating < 7).
- Create a file scan for "sailors.in" as "am" and for "reserves.in" as "am2".
- Use "SortMerge" for sort and merge operations to get all the records matching the query condition.
- Project the colums "sname", "age" and "rating" from "sm" (SortMerge).

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 Equivalent: 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;

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

Steps:

- Create a query condition - (S.rating > 7) and create a file scan for "sailors.in" as "am".
- Project field sname and bid from nested loop join based above query condition, file scan and "reserves.in" heap file on field sid.
- Create a query condition - (color=red) and create a file scan for "boats.in" as am2. The result from the query is joined with the above result and "sname" is printed in ascending order.

Sort Tests

Test 1

This Test verifies creation and sorting data in ascending order.

Steps:

- Create an unsorted heap file - "test1.in"
- Create a tuple of size 94 and store the data in the heap file.
- Create a file scan for the heap file and sort the file in ascending order.
- Validate if the result is sorted in ascending order.

Test 2

This Test verifies creation and sorting data in descending order.

Steps:

- Create an unsorted heap file - "test2.in"
- Create a tuple of size 94 and store the data in the heap file.
- Create a file scan for the heap file and sort the file in descending order.
- Validate if the result is sorted in descending order.

Test 3

This test verifies the sort operations in different orders for different fields.

Steps:

- Create an unsorted heap file - "test3.in".
- Create a tuple of 1000 records consisting of fields - Integer (inum), Float(fnum), and String (data1[i%NUM_RECORDS]) for all 1000 records.
- A file scan is created and the heap file is sorted based on the Integer field in ascending order and validated.
- The heap file is then sorted in descending order based on the Float field and validated.

Test 4

This test verifies the sort operations in ascending order and descending order.

Steps:

- Create two heap files - "test4-1.in" and "test4-2.in".
- Create a tuple from unsorted data and store it in the above-created files.
- Heap file "test4-1.in" is sorted in ascending order and data is validated.
- Heap file "test4-2.in" is sorted in descending order and data is validated.

Sort Merge Tests

Query 1

Query: Find the names of sailors who have reserved boat number 1 and print out the date of the reservation.
SQL Equivalent: 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.
Steps:

- Create a query condition with boat number 1.
- A file scan is created for "sailors.in" as a variable "am" and for "reserves.in" as variable "am2".
- "SortMerge" is used for sort and merge operations to get all the records matching the query conditions.
- Project the columns "sname" and "date" from the above step.

Query 3

Query: Find the names of sailors who have reserved a boat.
SQL Equivalent: SELECT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid.
Tests FileScan, Projection, and SortMerge Join.
Steps:

- Create a file scan for "sailors.in" as variable "am", also for "reserves.in" as variable "am2".
- Sort and merge operations are performed to get all sailors who have reserved boats.
- From the results of the previous step, the column "sname" is projected.

Query 4

Query: Find the names of sailors who have reserved a boat and print each name once.
SQL Equivalent: SELECT DISTINCT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid.
Tests FileScan, Projection, Sort-Merge Join and Duplication elimination.
Steps:

- Create a file scan for "sailors.in" as variable "am", also for "reserves.in" as variable "am2".
- Sort and merge operations are performed to get all sailors who have reserved boats.
- Project the column "sname" from the above result. Redundant names are removed from the projected names.

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 Equivalent: 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.

Steps:

- Query condition - (S.age > 40 || S.rating < 7) is created.
- A file scan is created for "sailors.in" as variable "am" and for "reserves.in" as variable "am2".
- Use "SortMerge" for sort and merge operations to get all the records matching the query conditions.
- Project the columns "sname", "age" and "rating" from the results of the previous step.

CONCLUSION

Relational Database management systems implementation consists of various components. Using Minibase, we were able to understand some of the implementations and functionalities of RDBMS. Various test cases aided in understanding internal mechanisms and test cases for multiple conditions. Also, we were able to understand the querying and join, sort implementations in a relational database management system.

BIBLIOGRAPHY

1. https://www.tutorialspoint.com/dbms/index.htm
2. https://www.geeksforgeeks.org/relational-model-in-dbms/
3. https://www.cs.purdue.edu/homes/clifton/cs541/project3/javadocs/heap/HeapFile.html
4. https://www.w3schools.com/sql/
5. https://research.cs.wisc.edu/coral/mini_doc/bufMgr/bufMgr.html
6. https://www.wikiwand.com/en/Parsing#/Parser
7. https://www.wikiwand.com/en/Query_optimization

APPENDIX

Typescript output:

```
Script started on 2020-02-02 16:24:01-08:00 [TERM="xterm-256color" TTY="/dev/pts/0"
COLUMNS="181" LINES="47"]
]0;ganesh@ubuntu: ~/dbms/src[01;32mganesh@ubuntu[00m:[01;34m~/dbms/src[00m$ make
test
cd tests; make bmtest dbtest; whoami; make hftest bttest indextest jointest
sorttest sortmerge
make[1]: Entering directory '/home/ganesh/dbms/src/tests'
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/javac -classpath /home/ganesh/dbms/src
TestDriver.java BMTest.java
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java  -classpath /home/ganesh/dbms/src
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-1.8.0-openjdk-amd64/bin/javac -classpath /home/ganesh/dbms/src
TestDriver.java DBTest.java
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java  -classpath /home/ganesh/dbms/src
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 '/home/ganesh/dbms/src/tests'
ganesh
make[1]: Entering directory '/home/ganesh/dbms/src/tests'
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/javac -classpath /home/ganesh/dbms/src
TestDriver.java HFTest.java
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java  -classpath /home/ganesh/dbms/src
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-1.8.0-openjdk-amd64/bin/javac -classpath /home/ganesh/dbms/src
TestDriver.java BTTest.java
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java  -classpath /home/ganesh/dbms/src
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 :1 0
 ***************** 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 :1
 **************** 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 :5
Please input the integer key to insert:
```

```
1
-------------------------- 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:
5
 ***************** 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 :8
Please input the number of keys to insert:
9
 ***************** 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 :9
Please input the number of keys to insert:
5
 ***************** The file name is: AAA5  **********
-------------------------- 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:
3
Please input the number of keys to delete:
2
 ***************** The file name is: AAA6  **********
-------------------------- 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 :11
Please input the LOWER integer key(>=0):
4
Please input the HIGHER integer key(>=0)
6
-------------------------- 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      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 :3
```

```
---------------The B+ Tree Leaf Pages---------------

**************To Print an Leaf Page ********
Current Page ID: 13
Left Link     : -1
Right Link    : -1
0 (key, [pageNo, slotNo]):   (0,  [ 0 0 ] )
************** END ********



------------- All Leaf Pages Have Been Printed --------


-------------------------- 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 :4
Please input the page number:
3
Sorry!!! This page is neither Index nor Leaf page.
-------------------------- 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 :6
Please input the integer key to delete:
5
-------------------------- 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):
4
Please input the HIGHER integer key (null if -2):
8
-------------------------- 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 :13
AT THE END OF SCAN!
-------------------------- 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 :14
No Record to delete!
btree.ScanDeleteException
    at btree.BTFileScan.delete_current(BTFileScan.java:105)
    at tests.BTDriver.runAllTests(BTTest.java:278)
    at tests.BTDriver.runTests(BTTest.java:80)
    at tests.BTTest.main(BTTest.java:648)
btree.ScanDeleteException
    at btree.BTFileScan.delete_current(BTFileScan.java:121)
    at tests.BTDriver.runAllTests(BTTest.java:278)
    at tests.BTDriver.runTests(BTTest.java:80)
    at tests.BTTest.main(BTTest.java:648)
       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
       !!            Something is wrong                    !!
       !!      Is your DB full? then exit. rerun it!   !!
       !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-------------------------- 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 :15
Please input the number of keys to insert:
3
Please input the number of keys to delete:
1
 ***************** The file name is: AAA7  **********
-------------------------- 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 :16
 ***************** You close the file: AAA7  **********
-------------------------- 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 :17
1
 ***************** You open the file: 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 :18
6
 ***************** You destroy the file: AAA6  **********
-------------------------- 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-1.8.0-openjdk-amd64/bin/javac -classpath /home/ganesh/dbms/src
TestDriver.java IndexTest.java
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java  -classpath /home/ganesh/dbms/src
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-1.8.0-openjdk-amd64/bin/javac -classpath /home/ganesh/dbms/src
TestDriver.java JoinTest.java
Note: JoinTest.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java  -classpath /home/ganesh/dbms/src
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-1.8.0-openjdk-amd64/bin/javac -classpath /home/ganesh/dbms/src
TestDriver.java SortTest.java
/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin/java  -classpath /home/ganesh/dbms/src
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-1.8.0-openjdk-amd64/bin/javac -classpath /home/ganesh/dbms/src
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-1.8.0-openjdk-amd64/bin/java  -classpath /home/ganesh/dbms/src
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 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!!!*****************



**********************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!!!*****************



Finished joins testing
join tests completed successfully
make[1]: Leaving directory '/home/ganesh/dbms/src/tests'
]0;ganesh@ubuntu: ~/dbms/src[01;32mganesh@ubuntu[00m:[01;34m~/dbms/src[00m$ exit

Script done on 2020-02-02 16:27:27-08:00 [COMMAND_EXIT_CODE="0"]
```