# DBMSI Minibase Project - Phase 1


## Group Members

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


**Abstract** — The trend in the growth of the data size in the past few years is exponential. The demand for decisive methods to establish data retrieval and storage have become critical. This assimilation can be done through Database Management Systems. In this first phase of the project, Minibase database management system is set up in a UNIX environment. We also delve into various peripherals that creates this storage engine. With these means we get a peek of the assorted crucial pieces that establishes a database management system. In this phase we test different functionalities of every component through a streak of different tests (bilateral and non-bilateral). These tests are executed after a successful build of the Minibase java project.

# INTRODUCTION

A database management system or DBMS is a software designed to assist in maintaining and utilizing large collections of data[1]. Minibase is a miniature implementation of such a database management system. It belongs to RDBMS which is Relational Database Management System[2]. RDBMS is based on a relational model. It means the management of data is done using a structure and language that is consistent of first order logic. The Minibase has a parser, optimizer, buffer pool manager, storage mechanisms (heap files, secondary indexes based on B+ Trees), and a disk space management system. The goal is not just to have a functional DBMS, but to have a DBMS where the individual components can be studied and implemented by students. A large number of RDBMS uses a SQL like syntax for queries and data definitions.

# Terminology

**Database:** A database is an organized collection of data[3].
**Database Management System:** The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data[3].
**SQL:** SQL stands for Structured Query Language. It is a domain specific language which is used for managing data in RDBMS (Relational Database Management System)[4].
**Data Model:** A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities[5].
**Relational Calculus:** The Relational calculus consists of two calculi, the tuple relational calculus and the domain relational calculus, that are part of the relational model for databases and provide a declarative way to specify database queries[6].
**Relational Algebra:** Relational Algebra is procedural query language, which takes Relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL[7].
**Disk Space Manager:** The disk space manager is the component of the system that takes care of the allocation and de-allocation of pages among the different files of a database. It also performs reads and writes of pages to and from disk[8].
**Buffer Manager:** Buffer manager intelligently shuffles data from main memory to disk: It is transparent to higher levels of DBMS operation[9].
**Parser:** The Parser checks the SQL statement and ensures its validity. The statements are verified both syntactically and semantically. During parsing, the SQL statement is transformed into a database-internal representation, called the parse tree[10].

**Optimizer:** For a given parse tree, the database must decide the most efficient data fetching algorithm. Optimizer evaluates multiple data traversing and based on their overall cost; it chooses the one requiring the least amount of time to execute[10].
**Executor:** The Executor simply runs the execution plan which is like a runtime-generated program that tells the Executor how to fetch the data the client requires[10].
**B+ Trees:** A B+ tree is an N-ary tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves[13].
**Database Index:** A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed[12].

# Goal Description

The goal of this first phase of the project is to get a glimpse of Minibase project, the setup of the Minibase in a UNIX system. Understand how various parts of DBMS work with each other. Use the Makefiles to build the project in the respective environment. Perform all the tests and discuss what each test does.

# Test case Description

**Buffer Management Tests** - In this three tests of functionality, the first test is for checking the basic operations of buffer management. The second test checks for illegal operations like pinning too many pages and unpinning double pinned page and the last test is for checking the internals of the buffer management system.

1. Test1
    - First obtain the unpinned buffer frames, then allocate these pages.
    - Loop through these and pin these pages. Write data to these pages, the data written is page id + 99999 correspondingly. Unpin these pages.
    - Once the previous steps are complete, Now pin back these pages and check for the data written which is for validating the content in these pages. Once this is complete, all these pages are unpinned.
    - Now free all the pages which were allocated.
2. Test2
    - The first part includes pinning more pages than the frames in the buffer, this is carried out similar to the above test and it's ensured that no more pages can be pinned, and then we try to pin the page and it fails as expected.

- In the second part, the first page is pinned twice and then this doubly pinned page is tried to be freed and it fails as expected.
- In the last part, unpinning a page which is not there in the buffer is tried and it fails as expected. To do this the pages are freed similar to the test above. Once it is ensured that the page is not present, the page is attempted to unpin, and it fails.

3. Test3
   - 60 pages are allocated and 99999 is added to page id of the page, but if the page id %20 is not 12 then unpin the page.
   - Now pin all pages, when this is done, some of the pages are doubly pinned. After this is done, loop and read the contents and validate.
   - The pages having page id % 20 is 12 are unpinned.

**Disk Management Tests** - These tests are carried out to ensure that there is proper functionality in the disk management. The first test checks for the standard operations of the disk manager. The second test is a continuation of the first test which performs some more tests. The third test checks for illegal operations and the fourth test checks for edge cases or boundary conditions.

1. Test1
   - 6 pages are allocated, and a file entry is added for each of these pages.
   - Now 30 pages are allocated, and data A followed by a number is written for the first 20 pages in it.
   - The ten remaining empty pages are then deallocated.
2. Test2
   - It performs tests enduring on test1. The 3 files which were created before are deleted.
   - For the remaining 3 files, file lookup is done and validated.
   - The contents of pages, which were written in first test are checked and verified.
   - Once the verification is complete, deallocation is done for all the pages.
3. Test3
   - This illegal operation involves the lookup of a file entry which was deleted. The entry called file1 which was deleted in the previous task is looked up and it fails
   - The next operation involves deletion of a file which was already deleted. The same file, file1 which was used in previous task is tried to delete, this was already deleted and hence the failure.

- Now the next task involves the deletion of a file entry which was not existent. A random file name is chosen, and that name is used for deletion operation and it failed as expected.
- The fourth operation constitutes a lookup for a non-existing file, the same file name which was used before which was random is used to do a lookup, since it's not present it fails.
- Next a file entry is added but a file with the same name already exists, file3 is used here and it fails because file3 was already existing.
- Now a new file entry is attempted to be added which has the file name length 5 characters more than the maximum permissible limit which is 50. Since the length is more the operation fails.
- The seventh operation constitutes a run of pages to be allocated which is very long. In this case 9000 pages are allocated, and the operation fails because its too long.
- Next step involves allocation of a negative number of pages, in this case -10 is passed and the operation fails.
- The ninth step involves a negative run used for deallocating the pages, -10 is passed again to deallocate the pages, and it fails which was expected.

4. Test4
   - Check the number of unpinned buffer frames and the total number of buffer frames to confirm that no pages are unpinned.
   - Create a database to hold up to two pages in its space map, now a run is done for allocating slightly less than the database size. Now this will account to database overhead.
   - Try allocating one more page, and because there is an overhead the operation fails.
   - Free the pages 3 to 9 and 33 to 40 and pages from 33 to 40 are allocated again.
   - Two runs of pages are done from 11-17 and 18-28 which are freed and now these pages are allocated again.
   - The file entries are added in directory to exceed a page. For this directory try to allocate more pages than what's available. This fails as anticipated.
   - By this time all the pages have to be claimed, so try allocating one more page and this operation fails.
   - In the last step, deallocate the last 2 pages in the space map to test the boundary condition.

**Heap Tests** - These involve creating a heap and scanning them. Opening files and doing some operations like deletions, modifications and some illegal operations.

1. Test1

- Create file_1 which is a heap file.
- In this file, hundred records are inserted and each of these have name, ivalue and fvalue as their attributes, all of these are given.
- Check buffer to ensure that no page is left pinned. Now check for consistency by scanning these files sequentially.
- All the attributes mentioned earlier are checked during the above scan.

2. Test2
   - Use the same file as created in the first test above.
   - Now delete half of the records by deleting the alternating records which are odd numbered ones.
   - Check the buffer again to ensure that these records are not pinned.
   - Now to check for consistency scan the remaining records and the test is completed successfully.
3. Test3
   - Use the same file again as used in the first test.
   - The fvalues of all these records are updated and this is done by scanning all the records in the heap.
   - Check buffer to ensure that there are no records which are pinned.
   - Now to check for the consistency check the records by scanning all the values and the tests executed successfully.
4. Test4
   - The first erroneous procedure is carried out by trying to shorten the length of the record.
   - As done in the earlier tests use file_1 heap file. Read this and now reduce the length of the first record by 1. This errors out as expected.
   - The next part is to increase the length and test, to do this, use the same heap file and increase the length of the first record and this operation fails
   - Now the last part consists of testing the insertion of a large record. To carry out this use the MINIBASE_PAGESIZE and add a value to this, in this case its 4, so the total becomes 1028. Now this big record is now inserted and since it's bigger than the maximum size it errors out.

**B+ Tree tests** - The B+ tree tests are interactive with the user and a menu is displayed for each course of action. The user has to select an option given in the menu and each option performs a task. There are 20 options given to the user. These tests play around the queries to files using this structure. In these tests volatile files are created, it's named in the format as AAA<number> and <number> is a positive integer.

0. Create a new B+ Tree file with naive delete option. This file is prefixed with AAA as described above. Once this is created, the context is switched to this file. The

redistribution and page merging when the entries fall below threshold will not be done when Naïve delete is used[11].

1. Create a B+ tree with full delete option. Once this is done, context is switched to the new file.
2. Print the B+ tree structure. On first line prints 1 and then the ID of root and remaining pages of the tree, if the tree is not there, it prints the tree is empty.
3. Print all leaf edges. It prints the contents of all the leaf node in <key, rid> format. It also gives the indication of the current page id.
4. Choose a page to print. An input is taken which is the page id and only the contents of that page is printed. If you give any invalid page id, it tells that it is invalid.
5. Inserting record to the tree. This will insert a record to the B+ tree. Negative integers cannot be added to the tree.
6. The next item is for deleting an element in the tree. Balance is maintained by restructuring, if there are same element values then only one is deleted.
7. Inserting n records in order. It creates a new file. It takes the value of n and it adds from 0 to n-1 in increasing order. Records can be viewed by using the 3rd option.
8. This is similar to 7, but it inserts the values in descending order. The values start at n and the last element of insertion is 1.
9. This is similar to 7. It inserts n records randomly in the B+ tree. But it inserts records in random order.
10. This is similar to 9 and in addition to that it also deletes m records in random order.
11. Delete some records. This is option is nothing but range deletion. This is carried out by specifying a range namely lower and higher range.
12. This option is used to check the scan initialization. Similar to the previous task you specify a lower and upper limits. This is used in later stages.
13. Within the B+ tree, this will scan the next element and the initialization is done in the previous option. If there are no records it tells that its empty.
14. Delete the just scanned record. After scanning in 12th option and reading it in 13th option, this option is used to delete the record from the B+ tree which was just scanned before. If there is nothing to delete error is displayed.
15. Inserting n records randomly and deletion of m records randomly. This is same as 10. The only difference is that the records keys are string here.
16. Close the file. The current file can be closed using this option. Insertion and deletion will not work because the file is closed.
17. Open the file. The file which was closed previously can be opened again using this option. The input to this is a number which is present in the file name. Once the number is entered the file is opened again. It will error out if a wrong number is entered.
18. Destroy a file. This is used for deleting a file, the input is similar to the previous steps. If a wrong number is given here, it will error out as expected.

19. Quit. This is the last option and is used for quitting the test.

**Index tests** – Index tests are used to validate the indexes used by the DBMS. These tests include creation of indexes, scanning them and also evaluating them.

1. Test1
    - This involves in creation of index. Test1.in filename is used here, and the file is created.
    - From the data1 array add the tuples to the heap file, once this is complete, scan this heap file and add these to the BTreeIndex which is a B+ tree file.
    - Once this is done, scan the index and check for the data consistency. The tests execute as expected.
2. Test2
    - For this test, use the same index file BTreeIndex and use the same heap file test1.in created during the previous step.
    - Dsilva is used for the selection and the index is scanned to see if there is a match with dsilva. Validate and check for if its valid.
    - The next step is to do a range search. Set the range between dsilva and yuc and repeat the similar process and validate as before.
3. Test3
    - Create a new heap file test3.in for this test.
    - To this heap insert tuples randomly which have integer, float and string.
    - Create the index BTIndex on the integer fields of records present in test3.in.
    - Perform a range search between 100 and 900 and validate as before, cleanup in the end.

**Join Tests** - These tests are for testing different joins, projections, file scans and index scans.

1. Query1
    - Find the names of sailors who have reserved boat number 1 and print out the date of reservation. This does filescan, projection and sort merge join.
    - Query: SELECT S.sname, R.date FROM Sailors S, Reserves R WHERE S.sid = R.sid AND R.bid = 1
    - To perform this query, see the condition R.bid=1 and construct the condition. Use sailors.in heap file and obtain the scan pointer. Repeat the same for reserves.in heap file.
    - Get the records that satisfy these conditions using sort and merge join operations. Once this is done, now project the name and date as per the query.
2. Query2

- Find the names of sailors who have reserved a red boat and return them in alphabetical order. This tests file scan, index scan, projection, index selection, sort and simple nested loop join.
- Query: SELECT S.sname FROM Sailors S, Boats B, Reserves R WHERE S.sid = R.sid AND R.bid = B.bid AND B.color = 'red' ORDER BY S.sname
- To do this first scan the heap file sailors.in and now create an index file BTreeIndex with field sid. Use this file to scan reserves.in heap file and do a nested loop join on sid. The result of this operation is a projection of sname and bid
- Use boats.in heap file to select records with condition on color is red.
- Join the results of the previous two steps on sid and project sname. Finally sort the results in increasing order as per the query.

3. Query3
   - Find the names of sailors who have reserved a boat. This does file scan, projection and sort merge join.
   - Query: SELECT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
   - As before use heap files sailors.in and reserves.in and get the scan pointers.
   - Now sort and merge and get the records that are in match with the condition given and then project sname according to the given query.
4. Query4
   - Find the names of sailors who have reserved a boat and print each name once. It does file scan, projection, sort merge and duplicate elimination.
   - Query: SELECT DISTINCT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
   - As before use heap files sailors.in and reserves.in and get the scan pointers.
   - Now sort and merge and get the records that are in match with the condition given and then project sname according to the given query. Once this is done the duplicates are removed and then shown in the output.
5. Query5
   - Find the names of old sailors or sailors with a rating less than 7, who have reserved a boat. It does file scan, multiple selection, projection and sort merge join.
   - SELECT S.sname, S.rating, S.age FROM Sailors S, Reserves R WHERE S.sid = R.sid and (S.age $>$ 40 || S.rating $<$ 7)
   - To do this query use the given conditions in order to construct the query condition.
   - As before use heap files sailors.in and reserves.in and get the scan pointers.

- Now sort and merge join is used and obtain records for the given conditions. Once this is done sname, rating and age are projected according to the given query.

6. Query6
   - Find the names of sailors with a rating greater than 7 who have reserved a red boat and print them out in sorted order. It does file scan, multiple selection, projection, sort and nested loop join.
   - Query: SELECT S.sname FROM Sailors S, Boats B, Reserves R WHERE S.sid = R.sid AND S.rating > 7 AND R.bid = B.bid AND B.color = 'red' ORDER BY S.name
   - To do this query, use the heap file sailors.in and get the records according to the constraint given where rating is greater than 7. Get the records from reserves.in and join these two using sid and project sname and bid
   - Use heap file boats.in and get the ones where color is red. Now join these records with the ones obtained in the step before using sid. Project sname in as per the query in ascending order.

**Sort tests** – These tests are used to test the sort functionality and evaluates if it works properly.

1. Test1
   - Take two arrays where one is sorted, and one is not sorted. Elements of unsorted array are placed in test1.in heap file.
   - This file is scanned, and the elements are sorted. Once this is complete read the elements and check from the second array and see if matches. Tests work as expected.
2. Test2
   - To do this test, it follows the same procedure as the previous test with the only difference is that the sort order is reversed.
3. Test3
   - To perform this test, create a heap file test3.in, 1k records are added to this and this record has 3 types of components integer, float and string.
   - First sort this based on integer field in increasing order and validate them.
   - Finally sort these again in decreasing order of float values and validate again.
4. Test4
   - As the test1 have two arrays where one is sorted, and one is not. The unsorted one is put into two files test4-1.in and test4-2.in
   - The first file is sorted in increasing order and validate the data and the order and do the same for the second file with the order of sort being decreasing order.

**Sort Merge Join tests** – These tests are for testing the sort merge and join operations.

1. Query1
    - Find the names of sailors who have reserved boat number 1 and print out the date of reservation. This tests file scan, projection and sort merge join.
    - Query: SELECT S.sname, R.date FROM Sailors S, Reserves R WHERE S.sid = R.sid AND R.bid = 1
    - To do this query construct the query condition which is R.bid is 1 and obtain pointers as described in the previous tests for sailors.in and reserves.in
    - Use sort and merge join and get the records which will satisfy the condition and now project sname and data.
2. Query3
    - Find the names of sailors who have reserved a boat. This tests file scan, projection and sort merge join.
    - Query: SELECT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
    - In order to perform this query first as described above obtain the scan pointers on sailors.in and reserves.in
    - Do the sort and merge join and get the records that satisfies the conditions and project sname as per the query.
3. Query4
    - Find the names of sailors who have reserved a boat and print each name once. This tests file scan, projection, sort merge join and duplication elimination.
    - Query: SELECT DISTINCT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
    - To perform this query, do the same thing as the previous steps of getting the scans pointers of files sailors.in and reserves.in
    - Do sort and merge and join the results according to the condition given and then project sname as per the given query and now eliminate the duplicates and this will be the result of the query.
4. Query5
    - Find the names of old sailors or sailors with a rating less than 7, who have reserved a boat. This tests file scan, multiple selection, projection and sort merge join.
    - Query: SELECT S.sname, S.rating, S.age FROM Sailors S, Reserves R WHERE S.sid = R.sid and (S.age $>$ 40 || S.rating $<$ 7)
    - In order to do this query, first construct the given query condition which is S.age greater than 40 or S.rating is less than 7.

- Obtain the scan pointers as discussed before for sailors.in and resrves.in
- Do the sort merge and join operation and get all records that match the condition. Now project sname, age and rating as per the query.

# Conclusions

In this phase we obtained a peek of DMBS. The setup was done with ease. The test cases evaluated a specific functionality of DBMS. These tests included the interactive ones and the non-interactive ones. All these tests completed successfully, since the original code was not modified it was not a bewilderment. These tests provide a very good insight on how the parts of DBMS interact with each other, we also got to know how projections sorting and indexing works in DBMS. Performing and understanding these tests would help a lot in the future phases of the project and it would be helpful in writing the test cases for other operations in future phases of the project.

# BIBLIOGRAPHY

[1] Raghu Ramakrishnan and Johannes Gehrke. 2000. Database Management Systems (2nd. ed.). McGraw-Hill, Inc., USA.
[2] https://research.cs.wisc.edu/coral/minibase/minibase.html
[3] https://en.wikipedia.org/wiki/Database
[4] https://www.atmosera.com/resources/glossary/sql/
[5] https://en.wikipedia.org/wiki/Data_model
[6] https://en.wikipedia.org/wiki/Relational_calculus
[7] https://www.geeksforgeeks.org/introduction-of-relational-algebra-in-dbms/
[8] http://pages.cs.wisc.edu/~dbbook/openAccess/Minibase/spaceMgr/olddsm.html
[9] https://web.stanford.edu/class/cs346/2015/notes/Lecture_One.pdf
[10] https://vladmihalcea.com/relational-database-sql-prepared-statements/
[11] http://mll.csie.ntu.edu.tw/course/database_f09/assignment/hw5.pdf
[12] https://en.wikipedia.org/wiki/Database_index
[13] https://en.wikipedia.org/wiki/B%2B_tree

# APPENDIX

## Typescript output

```
Script started on Sat Feb  1 18:16:22 2020
⚡⋘ ωɬzαʔə ⋙ ⚡ make test

cd tests; make bmtest dbtest; whoami; make hftest bttest indextest jointest
sorttest sortmerge
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/javac -
classpath .:.. TestDriver.java BMTest.java
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/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.

/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/javac -
classpath .:.. TestDriver.java DBTest.java
```

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/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.

rakeshr
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/javac -
classpath .:.. TestDriver.java HFTest.java
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/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.

/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/javac -
classpath .:.. TestDriver.java BTTest.java
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/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 :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:
3
------------------------- 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     5
--------------- 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: 5
Left Link      : -1
Right Link     : -1
0 (key, [pageNo, slotNo]):   (3,  [ 3 3 ] )
************** 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 :6
Please input the integer key to delete:
3
------------------------- 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 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:
101
 ***************** 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 :2



---------------The B+ Tree Structure---------------
1     8
2            6
2            7
2            9
```

```
-------------- 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: 6
Left Link      : -1
Right Link     : 7
0 (key, [pageNo, slotNo]):   (0,  [ 0 0 ] )
1 (key, [pageNo, slotNo]):   (1,  [ 1 1 ] )
2 (key, [pageNo, slotNo]):   (2,  [ 2 2 ] )
3 (key, [pageNo, slotNo]):   (3,  [ 3 3 ] )
4 (key, [pageNo, slotNo]):   (4,  [ 4 4 ] )
```

```
5 (key, [pageNo, slotNo]):   (5,  [ 5 5 ] )
6 (key, [pageNo, slotNo]):   (6,  [ 6 6 ] )
7 (key, [pageNo, slotNo]):   (7,  [ 7 7 ] )
8 (key, [pageNo, slotNo]):   (8,  [ 8 8 ] )
9 (key, [pageNo, slotNo]):   (9,  [ 9 9 ] )
10 (key, [pageNo, slotNo]):   (10,  [ 10 10 ] )
11 (key, [pageNo, slotNo]):   (11,  [ 11 11 ] )
12 (key, [pageNo, slotNo]):   (12,  [ 12 12 ] )
13 (key, [pageNo, slotNo]):   (13,  [ 13 13 ] )
14 (key, [pageNo, slotNo]):   (14,  [ 14 14 ] )
15 (key, [pageNo, slotNo]):   (15,  [ 15 15 ] )
16 (key, [pageNo, slotNo]):   (16,  [ 16 16 ] )
17 (key, [pageNo, slotNo]):   (17,  [ 17 17 ] )
18 (key, [pageNo, slotNo]):   (18,  [ 18 18 ] )
19 (key, [pageNo, slotNo]):   (19,  [ 19 19 ] )
20 (key, [pageNo, slotNo]):   (20,  [ 20 20 ] )
21 (key, [pageNo, slotNo]):   (21,  [ 21 21 ] )
22 (key, [pageNo, slotNo]):   (22,  [ 22 22 ] )
23 (key, [pageNo, slotNo]):   (23,  [ 23 23 ] )
24 (key, [pageNo, slotNo]):   (24,  [ 24 24 ] )
25 (key, [pageNo, slotNo]):   (25,  [ 25 25 ] )
26 (key, [pageNo, slotNo]):   (26,  [ 26 26 ] )
27 (key, [pageNo, slotNo]):   (27,  [ 27 27 ] )
28 (key, [pageNo, slotNo]):   (28,  [ 28 28 ] )
29 (key, [pageNo, slotNo]):   (29,  [ 29 29 ] )
30 (key, [pageNo, slotNo]):   (30,  [ 30 30 ] )
************** END ********


**************To Print an Leaf Page ********
Current Page ID: 7
Left Link      : 6
Right Link     : 9
0 (key, [pageNo, slotNo]):   (31,  [ 31 31 ] )
1 (key, [pageNo, slotNo]):   (32,  [ 32 32 ] )
2 (key, [pageNo, slotNo]):   (33,  [ 33 33 ] )
3 (key, [pageNo, slotNo]):   (34,  [ 34 34 ] )
4 (key, [pageNo, slotNo]):   (35,  [ 35 35 ] )
5 (key, [pageNo, slotNo]):   (36,  [ 36 36 ] )
6 (key, [pageNo, slotNo]):   (37,  [ 37 37 ] )
7 (key, [pageNo, slotNo]):   (38,  [ 38 38 ] )
8 (key, [pageNo, slotNo]):   (39,  [ 39 39 ] )
9 (key, [pageNo, slotNo]):   (40,  [ 40 40 ] )
10 (key, [pageNo, slotNo]):   (41,  [ 41 41 ] )
11 (key, [pageNo, slotNo]):   (42,  [ 42 42 ] )
12 (key, [pageNo, slotNo]):   (43,  [ 43 43 ] )
13 (key, [pageNo, slotNo]):   (44,  [ 44 44 ] )
14 (key, [pageNo, slotNo]):   (45,  [ 45 45 ] )
15 (key, [pageNo, slotNo]):   (46,  [ 46 46 ] )
16 (key, [pageNo, slotNo]):   (47,  [ 47 47 ] )
17 (key, [pageNo, slotNo]):   (48,  [ 48 48 ] )
18 (key, [pageNo, slotNo]):   (49,  [ 49 49 ] )
19 (key, [pageNo, slotNo]):   (50,  [ 50 50 ] )
```

```
20 (key, [pageNo, slotNo]):   (51,  [ 51 51 ] )
21 (key, [pageNo, slotNo]):   (52,  [ 52 52 ] )
22 (key, [pageNo, slotNo]):   (53,  [ 53 53 ] )
23 (key, [pageNo, slotNo]):   (54,  [ 54 54 ] )
24 (key, [pageNo, slotNo]):   (55,  [ 55 55 ] )
25 (key, [pageNo, slotNo]):   (56,  [ 56 56 ] )
26 (key, [pageNo, slotNo]):   (57,  [ 57 57 ] )
27 (key, [pageNo, slotNo]):   (58,  [ 58 58 ] )
28 (key, [pageNo, slotNo]):   (59,  [ 59 59 ] )
29 (key, [pageNo, slotNo]):   (60,  [ 60 60 ] )
30 (key, [pageNo, slotNo]):   (61,  [ 61 61 ] )
************** END ********


**************To Print an Leaf Page ********
Current Page ID: 9
Left Link      : 7
Right Link     : -1
0 (key, [pageNo, slotNo]):   (62,  [ 62 62 ] )
1 (key, [pageNo, slotNo]):   (63,  [ 63 63 ] )
2 (key, [pageNo, slotNo]):   (64,  [ 64 64 ] )
3 (key, [pageNo, slotNo]):   (65,  [ 65 65 ] )
4 (key, [pageNo, slotNo]):   (66,  [ 66 66 ] )
5 (key, [pageNo, slotNo]):   (67,  [ 67 67 ] )
6 (key, [pageNo, slotNo]):   (68,  [ 68 68 ] )
7 (key, [pageNo, slotNo]):   (69,  [ 69 69 ] )
8 (key, [pageNo, slotNo]):   (70,  [ 70 70 ] )
9 (key, [pageNo, slotNo]):   (71,  [ 71 71 ] )
10 (key, [pageNo, slotNo]):   (72,  [ 72 72 ] )
11 (key, [pageNo, slotNo]):   (73,  [ 73 73 ] )
12 (key, [pageNo, slotNo]):   (74,  [ 74 74 ] )
13 (key, [pageNo, slotNo]):   (75,  [ 75 75 ] )
14 (key, [pageNo, slotNo]):   (76,  [ 76 76 ] )
15 (key, [pageNo, slotNo]):   (77,  [ 77 77 ] )
16 (key, [pageNo, slotNo]):   (78,  [ 78 78 ] )
17 (key, [pageNo, slotNo]):   (79,  [ 79 79 ] )
18 (key, [pageNo, slotNo]):   (80,  [ 80 80 ] )
19 (key, [pageNo, slotNo]):   (81,  [ 81 81 ] )
20 (key, [pageNo, slotNo]):   (82,  [ 82 82 ] )
21 (key, [pageNo, slotNo]):   (83,  [ 83 83 ] )
22 (key, [pageNo, slotNo]):   (84,  [ 84 84 ] )
23 (key, [pageNo, slotNo]):   (85,  [ 85 85 ] )
24 (key, [pageNo, slotNo]):   (86,  [ 86 86 ] )
25 (key, [pageNo, slotNo]):   (87,  [ 87 87 ] )
26 (key, [pageNo, slotNo]):   (88,  [ 88 88 ] )
27 (key, [pageNo, slotNo]):   (89,  [ 89 89 ] )
28 (key, [pageNo, slotNo]):   (90,  [ 90 90 ] )
29 (key, [pageNo, slotNo]):   (91,  [ 91 91 ] )
30 (key, [pageNo, slotNo]):   (92,  [ 92 92 ] )
31 (key, [pageNo, slotNo]):   (93,  [ 93 93 ] )
32 (key, [pageNo, slotNo]):   (94,  [ 94 94 ] )
33 (key, [pageNo, slotNo]):   (95,  [ 95 95 ] )
34 (key, [pageNo, slotNo]):   (96,  [ 96 96 ] )
```

```
35 (key, [pageNo, slotNo]):   (97,  [ 97 97 ] )
36 (key, [pageNo, slotNo]):   (98,  [ 98 98 ] )
37 (key, [pageNo, slotNo]):   (99,  [ 99 99 ] )
38 (key, [pageNo, slotNo]):   (100,  [ 100 100 ] )
************** 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:
9

**************To Print an Leaf Page ********
Current Page ID: 9
Left Link      : 7
```

```
Right Link     : -1
0 (key, [pageNo, slotNo]):   (62,  [ 62 62 ] )
1 (key, [pageNo, slotNo]):   (63,  [ 63 63 ] )
2 (key, [pageNo, slotNo]):   (64,  [ 64 64 ] )
3 (key, [pageNo, slotNo]):   (65,  [ 65 65 ] )
4 (key, [pageNo, slotNo]):   (66,  [ 66 66 ] )
5 (key, [pageNo, slotNo]):   (67,  [ 67 67 ] )
6 (key, [pageNo, slotNo]):   (68,  [ 68 68 ] )
7 (key, [pageNo, slotNo]):   (69,  [ 69 69 ] )
8 (key, [pageNo, slotNo]):   (70,  [ 70 70 ] )
9 (key, [pageNo, slotNo]):   (71,  [ 71 71 ] )
10 (key, [pageNo, slotNo]):   (72,  [ 72 72 ] )
11 (key, [pageNo, slotNo]):   (73,  [ 73 73 ] )
12 (key, [pageNo, slotNo]):   (74,  [ 74 74 ] )
13 (key, [pageNo, slotNo]):   (75,  [ 75 75 ] )
14 (key, [pageNo, slotNo]):   (76,  [ 76 76 ] )
15 (key, [pageNo, slotNo]):   (77,  [ 77 77 ] )
16 (key, [pageNo, slotNo]):   (78,  [ 78 78 ] )
17 (key, [pageNo, slotNo]):   (79,  [ 79 79 ] )
18 (key, [pageNo, slotNo]):   (80,  [ 80 80 ] )
19 (key, [pageNo, slotNo]):   (81,  [ 81 81 ] )
20 (key, [pageNo, slotNo]):   (82,  [ 82 82 ] )
21 (key, [pageNo, slotNo]):   (83,  [ 83 83 ] )
22 (key, [pageNo, slotNo]):   (84,  [ 84 84 ] )
23 (key, [pageNo, slotNo]):   (85,  [ 85 85 ] )
24 (key, [pageNo, slotNo]):   (86,  [ 86 86 ] )
25 (key, [pageNo, slotNo]):   (87,  [ 87 87 ] )
26 (key, [pageNo, slotNo]):   (88,  [ 88 88 ] )
27 (key, [pageNo, slotNo]):   (89,  [ 89 89 ] )
28 (key, [pageNo, slotNo]):   (90,  [ 90 90 ] )
29 (key, [pageNo, slotNo]):   (91,  [ 91 91 ] )
30 (key, [pageNo, slotNo]):   (92,  [ 92 92 ] )
31 (key, [pageNo, slotNo]):   (93,  [ 93 93 ] )
32 (key, [pageNo, slotNo]):   (94,  [ 94 94 ] )
33 (key, [pageNo, slotNo]):   (95,  [ 95 95 ] )
34 (key, [pageNo, slotNo]):   (96,  [ 96 96 ] )
35 (key, [pageNo, slotNo]):   (97,  [ 97 97 ] )
36 (key, [pageNo, slotNo]):   (98,  [ 98 98 ] )
37 (key, [pageNo, slotNo]):   (99,  [ 99 99 ] )
38 (key, [pageNo, slotNo]):   (100,  [ 100 100 ] )
************** 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:
50
 ***************** 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 B+ Tree Structure---------------
1     11
--------------- 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:
105
 ***************** 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 :2



---------------The B+ Tree Structure---------------
1     15
2            13
2            14
--------------- 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 :10
Please input the number of keys to insert:
106
Please input the number of keys to delete:
26
 ***************** 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 :2



---------------The B+ Tree Structure---------------
1     19
2            17
2            18
--------------- 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 :11
Please input the LOWER integer key(>=0):
10□ □1
Please input the HIGHER integer key(>=0)
21
------------------------- 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     19
2            17
2            18
--------------- 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: 17
Left Link      : -1
Right Link     : 18
0 (key, [pageNo, slotNo]):   (0,  [ 0 0 ] )
1 (key, [pageNo, slotNo]):   (1,  [ 1 1 ] )
2 (key, [pageNo, slotNo]):   (2,  [ 2 2 ] )
3 (key, [pageNo, slotNo]):   (3,  [ 3 3 ] )
4 (key, [pageNo, slotNo]):   (4,  [ 4 4 ] )
5 (key, [pageNo, slotNo]):   (5,  [ 5 5 ] )
6 (key, [pageNo, slotNo]):   (6,  [ 6 6 ] )
7 (key, [pageNo, slotNo]):   (8,  [ 8 8 ] )
8 (key, [pageNo, slotNo]):   (9,  [ 9 9 ] )
9 (key, [pageNo, slotNo]):   (23,  [ 23 23 ] )
10 (key, [pageNo, slotNo]):   (24,  [ 24 24 ] )
11 (key, [pageNo, slotNo]):   (25,  [ 25 25 ] )
12 (key, [pageNo, slotNo]):   (26,  [ 26 26 ] )
13 (key, [pageNo, slotNo]):   (27,  [ 27 27 ] )
14 (key, [pageNo, slotNo]):   (29,  [ 29 29 ] )
15 (key, [pageNo, slotNo]):   (30,  [ 30 30 ] )
16 (key, [pageNo, slotNo]):   (34,  [ 34 34 ] )
17 (key, [pageNo, slotNo]):   (35,  [ 35 35 ] )
18 (key, [pageNo, slotNo]):   (38,  [ 38 38 ] )
19 (key, [pageNo, slotNo]):   (39,  [ 39 39 ] )
20 (key, [pageNo, slotNo]):   (41,  [ 41 41 ] )
21 (key, [pageNo, slotNo]):   (43,  [ 43 43 ] )
22 (key, [pageNo, slotNo]):   (44,  [ 44 44 ] )
23 (key, [pageNo, slotNo]):   (45,  [ 45 45 ] )
24 (key, [pageNo, slotNo]):   (46,  [ 46 46 ] )
25 (key, [pageNo, slotNo]):   (47,  [ 47 47 ] )
26 (key, [pageNo, slotNo]):   (48,  [ 48 48 ] )
27 (key, [pageNo, slotNo]):   (50,  [ 50 50 ] )
28 (key, [pageNo, slotNo]):   (51,  [ 51 51 ] )
29 (key, [pageNo, slotNo]):   (53,  [ 53 53 ] )
30 (key, [pageNo, slotNo]):   (54,  [ 54 54 ] )
31 (key, [pageNo, slotNo]):   (55,  [ 55 55 ] )
************** END ********


**************To Print an Leaf Page ********
Current Page ID: 18
Left Link      : 17
Right Link     : -1
0 (key, [pageNo, slotNo]):   (56,  [ 56 56 ] )
1 (key, [pageNo, slotNo]):   (57,  [ 57 57 ] )
2 (key, [pageNo, slotNo]):   (58,  [ 58 58 ] )
3 (key, [pageNo, slotNo]):   (60,  [ 60 60 ] )
4 (key, [pageNo, slotNo]):   (62,  [ 62 62 ] )
5 (key, [pageNo, slotNo]):   (63,  [ 63 63 ] )
6 (key, [pageNo, slotNo]):   (64,  [ 64 64 ] )
7 (key, [pageNo, slotNo]):   (65,  [ 65 65 ] )
8 (key, [pageNo, slotNo]):   (66,  [ 66 66 ] )
9 (key, [pageNo, slotNo]):   (67,  [ 67 67 ] )
```

```
10 (key, [pageNo, slotNo]):   (68,  [ 68 68 ] )
11 (key, [pageNo, slotNo]):   (70,  [ 70 70 ] )
12 (key, [pageNo, slotNo]):   (71,  [ 71 71 ] )
13 (key, [pageNo, slotNo]):   (72,  [ 72 72 ] )
14 (key, [pageNo, slotNo]):   (74,  [ 74 74 ] )
15 (key, [pageNo, slotNo]):   (75,  [ 75 75 ] )
16 (key, [pageNo, slotNo]):   (76,  [ 76 76 ] )
17 (key, [pageNo, slotNo]):   (77,  [ 77 77 ] )
18 (key, [pageNo, slotNo]):   (78,  [ 78 78 ] )
19 (key, [pageNo, slotNo]):   (80,  [ 80 80 ] )
20 (key, [pageNo, slotNo]):   (83,  [ 83 83 ] )
21 (key, [pageNo, slotNo]):   (84,  [ 84 84 ] )
22 (key, [pageNo, slotNo]):   (85,  [ 85 85 ] )
23 (key, [pageNo, slotNo]):   (86,  [ 86 86 ] )
24 (key, [pageNo, slotNo]):   (87,  [ 87 87 ] )
25 (key, [pageNo, slotNo]):   (88,  [ 88 88 ] )
26 (key, [pageNo, slotNo]):   (91,  [ 91 91 ] )
27 (key, [pageNo, slotNo]):   (93,  [ 93 93 ] )
28 (key, [pageNo, slotNo]):   (94,  [ 94 94 ] )
29 (key, [pageNo, slotNo]):   (95,  [ 95 95 ] )
30 (key, [pageNo, slotNo]):   (97,  [ 97 97 ] )
31 (key, [pageNo, slotNo]):   (100,  [ 100 100 ] )
32 (key, [pageNo, slotNo]):   (101,  [ 101 101 ] )
33 (key, [pageNo, slotNo]):   (102,  [ 102 102 ] )
34 (key, [pageNo, slotNo]):   (103,  [ 103 103 ] )
35 (key, [pageNo, slotNo]):   (104,  [ 104 104 ] )
36 (key, [pageNo, slotNo]):   (105,  [ 105 105 ] )
************** 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 :12
Please input the LOWER integer key (null if -3):
10
Please input the HIGHER integer key (null if -2):
30
-------------------------- 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
SCAN RESULT: 23 [ 23 23 ]
------------------------- 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
------------------------- 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:
5□ □75
Please input the number of keys to delete:
10
 ***************** 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 :2



---------------The B+ Tree Structure---------------
1     23
2            21
2            22
--------------- 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 :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
7
 ***************** You open 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 :18
7
 ***************** You destroy 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 :19

... Finished .


/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/javac -
classpath .:.. TestDriver.java IndexTest.java
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/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
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/javac -
classpath .:.. TestDriver.java JoinTest.java
Note: JoinTest.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/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
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/javac -
classpath .:.. TestDriver.java SortTest.java
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/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
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/bin/javac -
classpath .:.. SM_JoinTest.java TestDriver.java
Note: SM_JoinTest.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
/Library/Java/JavaVirtualMachines/jdk1.8.0_221.jdk/Contents/Home/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 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

⚡≪ ωɬzαʔ∂ ≫ ⚡ exit


Script done on Sat Feb  1 22:01:34 2020
```