# DBMSI Minibase Project - Phase 1

## GROUP MEMBERS – GROUP 3


- Aditya Chayapathy (1213050538)
- Soumya Adhya (1213191315)
- Kurra Dixith Kumar (1211262790)
- Archana Ramanathan Seshakrishnan (1211112666)
- Ejaz Saifudeen (1213262763)
- Jithin Perumpral John (1213175858)


## ABSTRACT

With the exponential growth of data over the past few years, the need for efficient techniques to organize data for storage and retrieval has become crucial. Database management systems provide an interface that let us achieve this. In this phase of the project, we set up minibase database management system in a UNIX environment and explore the various components that constitute this storage engine. With this, we get a glimpse of the various crucial component that constitutes a database management system. This involves testing various functionalities of each component through a series of interactive and non-interactive tests, which execute upon the successful building of the minibase java package.

# INTRODUCTION

A database management system (DBMS) is an application that helps manage the database structure and controls access to the data stored in the database. Minibase is a database management system intended for educational use [1]. It consists of a parser, optimizer, buffer manager, heap files, secondary indexes based on B+ Trees and a disk management system. It belongs to a class of database management systems called RDBMS. A relational database management system is based on the relational model, which is an approach to managing data using a structure and language consistent with the first order predicate logic [2]. The data is represented in terms of tuples, grouped into relations. Most relational databases use the SQL data definition and query language. We have configured the systems for minibase to run and various interactive and non-interactive functionalities present in the minibase engine were explored.

## Terminology

**Database**: A database is an organized collection of data [3].
**Database Management System**: A database-management system (DBMS) is a computer-software application that interacts with end-users, other applications, and the database itself to capture and analyze data [3].
**Data Model**: A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to properties of the real-world entities [4].
**Data Schema**: The database schema of a database system is its structure described in a formal language supported by the database management system (DBMS) [5].
**Relational Algebra**: Relational algebra is a family of algebras with a well-founded semantics used for modelling the data stored in relational databases and defining queries on it [6].
**Relational Calculus**: 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 [7].
**SQL**: SQL (Structured Query Language) is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS) [8].
**Buffer Manager**: The buffer manager reads disk pages into a main memory page as needed. The collection of main memory pages (called frames) used by the buffer manager for this purpose is called the buffer pool. This is just an array of Page objects. The buffer manager is used by (the code for) access methods, heap files, and relational operators to read/write/allocate/de-allocate pages [9].
**Parser**: A parser is a compiler or interpreter component that breaks data into smaller elements for easy translation into another language. A parser takes input in the form of a sequence of tokens or program instructions and usually builds a data structure in the form of a parse tree or an abstract syntax tree [10].
**Optimizer**: A query optimizer is a critical database management system (DBMS) component that analyzes Structured Query Language (SQL) queries and determines efficient execution mechanisms. A query optimizer generates one or more query plans for each query, each of which may be a mechanism used to run a query [11].

**Heap Files**: A heap file is an unordered set of records [12].
**Disk Space Manager**: The disk space manager (DSM) is the component of Minibase that takes care of the allocation and deallocation of pages within a database. It also performs reads and writes of pages to and from disk and provides a logical file layer within the context of a database management system [13].
**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. The root may be either a leaf or a node with two or more children [14].

## Goal Description

The goal of phase one of the project is to set up the minibase storage engine in a UNIX environment and to understand how the various components of the DBMS operate and interact with each other. The steps involved are to extract the source code from blackboard, make necessary changes to the "Makefile"s, building the project to generate the class files and running the tests on the source code to check for consistency. Finally, the results of the interactive and non-interactive test cases are observed and described in this report.

## Assumptions

1. The minibase storage engine is running in a UNIX environment.
2. The "Makefile"s are modified to reflect the user's working environment.
3. Keys entered for insertion into B+ Tree are positive integers.
4. For the interactive test cases in B+ Trees, the values to be entered are between 0 and 19.
5. For input values for the number of records 'n' and the file number 'k', only valid integers and no strings or other characters are entered.

# DESCRIPTIONS OF THE TESTS

## Buffer Management Tests

- Test 1:
    1. This test verifies if the basic operations of the buffer manager function as expected.
    2. These are the steps involved:
        a. Identify the number of the unpinned buffer frames.
        b. Allocate pages based on the count obtained in the previous step.
        c. On each of the newly allocated page, data (page id + 99999) is written into it. This step involves pinning the page, writing the data to the page and finally unpinning the page.
        d. Having written the data to the newly allocated pages, this test then goes onto reading the contents of the pages and validates the contents of the pages. This step involves pinning the page, reading the contents of the page, validating the content of the page and then finally unpinning the page.
        e. As a final step, the test frees all the pages.
- Test 2:
    1. This test involves performing illegal buffer manager operations and observing its behavior.
    2. The first illegal operation involves pinning more pages than there are frames in the buffer. First, the pages are allocated and then pinned to ensure no more pages can be pinned. Finally, the last of the allocated pages is attempted to being pinned upon which it fails as expected.
    3. The second illegal operation involves attempting to free a doubly pinned page. This involves pinning the first page obtained in step 1 the second time. Next, this page is attempted to be freed upon which it fails as expected.
    4. The third illegal operation involves attempting to unpin a page not in the buffer pool. For this, the last of the allocated pages obtained in step 1 is used. We know that this page is not in the buffer pool. Upon attempting it to unpin this page, the operation fails as expected.
- Test 3:
    1. This test involves performing some internal buffer manager operations.
    2. These are the steps involved:
        a. Allocate 60 pages. All these pages are pinned by default.
        b. Copy the data (page id + 99999) to each of these pages.
        c. If “page id % 20 != 12”, unpin the pages.

d. Pin all the pages. This leaves some of the pages to be doubly pinned. Next, the contents of the page are read and validated. Finally, all the pages are unpinned.
e. The pages with "page id % 20 == 12" are unpinned again.

## Disk Space Management Tests

- Test 1:
  1. This test creates a database and verifies if all the operations of the disk space manager function as expected.
  2. This involves the following:
     a. Allocate 6 pages and adding a file entry for each of these pages.
     b. Next, 30 pages are allocated and to the first 20 of these pages, we write data to them ("A" followed by an integer).
     c. The remaining 10 pages are deallocated.
- Test 2:
  1. This test is a continuation of the previous test. Here, we perform some more of the disk manager operations and verify its functionality.
  2. This involves the following:
     a. The first 3 file entries created in Test 1 are deleted.
     b. A file entry lookup operation is performed on the remaining 3 file entries.
     c. The contents of the pages written in Test 1 are read and validated.
     d. Finally, all the pages are deallocated.
- Test 3:
  1. This test involves performing illegal disk manager operations and observing its behavior.
  2. The first illegal operation involves doing a lookup on a deleted file entry. For this, the file entry deleted in Task 2 ("file1") is used to do a lookup, upon which it fails as expected.
  3. The second illegal operation involves attempting to delete an already deleted file entry. For this, the file entry deleted in Task 2 ("file1") is used for deletion, upon which it fails as expected.
  4. The third illegal operation involves attempting to delete a non-existent file entry. A non-existent file entry ("blargle") is used for this test, upon which it fails as expected.
  5. The fourth illegal operation involves attempting to do a lookup on a non-existent file entry. A non-existent file entry ("blargle") is used for this test, upon which it fails as expected.

6. The fifth illegal operation involves attempting to add a file entry that already exists. For this, the file entry created in Task 1 ("file3") is used for adding a new file entry, upon which it fails as expected.
7. The sixth illegal operation involves attempting to add a file entry that is too long. For this, an effort is made to add a file entry whose length (55) is greater than the maximum allowed length (50). This operation fails as expected.
8. The seventh illegal operation involves attempting to allocate a run of pages that is too long. To achieve this, an effort is made to allocate a run of 9000 pages. This operation fails as expected.
9. The eighth illegal operation involves attempting to allocate a negative run of pages. To achieve this, an effort is made to allocate a run of -10 pages. This operation fails as expected.
10. The ninth illegal operation involves attempting to deallocate a negative run of pages. To achieve this, an effort is made to deallocate a run of -10 pages. This operation fails as expected.

- Test 4:
    1. This test involves checking some boundary conditions.
    2. The steps performed are as follows:
        a. Ensure none of the pages is pinned by verifying if the number of unpinned buffer frames equals the total number of buffer frames.
        b. To allocate all pages remaining after database overhead count is accounted for, we create a database that is big enough such that it holds up two pages for its space map. A run is then allocated with the run size slightly less than the database size, which accounts for the overhead.
        c. An attempt is made to allocate one more page. This operation fails as there is no room for one more page due to overhead.
        d. The pages 3-9 and 33-40 are freed.
        e. Pages 33-40 are allocated.
        f. Two continued runs of pages (i.e. 11-17 and 18-28) are freed.
        g. Next, the pages freed in the previous step are allocated.
        h. Enough file entries (6) are added to the directory to surpass a page.
        i. An attempt is made to allocate more pages than available (7) for the directory. Since this operation is not allowed, it fails as expected.
        j. The test then verifies if all the pages are claimed by allocating one more page. This results in failure as expected.
        k. Finally, the test de-allocates the last 2 pages in the space map.

## Heap File Tests

- Test 1:
    1. This test involves inserting and scanning fixed sized records in a heap file.
    2. The steps involved are as follows:
        a. Create a heap file ("file_1").
        b. Insert 100 records to the newly created heap file. Each of the 100 records consists of 3 components, namely, ivalue, fvalue and the name of the record.
        c. Checks are made to ensure that none of the pages is left pinned in the buffer.
        d. Having inserted the records into the heap file, the next operation is to scan these files sequentially and check for their consistency.
        e. All the 3 components of the record are checked for consistency during the sequential scan.
- Test 2:
    1. This test involves deleting fixed sized records from a heap file.
    2. The steps involved are as follows:
        a. Open the file created in Task 1.
        b. All the alternate records, starting from the $2^{nd}$ record, are deleted from the heap file.
        c. Checks are made to ensure that none of the pages is left pinned in the buffer.
        d. The remaining records are scanned and each of their components is checked for consistency.
- Test 3:
    1. This test involves updating fixed sixed records in the heap file.
    2. The steps involved are as follows:
        a. Open the file created in Task 1.
        b. Scan the records and update the fvalue of all the records present in the heap file.
        c. Checks are made to ensure that none of the pages is left pinned in the buffer.
        d. The remaining records are scanned and each of their components is checked for consistency.
- Test 4:
    1. This test involves performing illegal heap file operations and observing its behavior.
    2. The first illegal operation involves attempting to shorten the length of a record in the heap file. To achieve this, the heap file created in Test 1 is used. The first record in the heap file is read and an attempt is made to shorten its length by 1. This results in a failure as expected.
    3. The second illegal operation involves attempting to increase the length of a record in the heap file. To achieve this, the heap file created in Test 1 is used.

The first record in the heap file is read and an attempt is made to increase its length by 1. This results in a failure as expected.

4. The third illegal operation involves attempting to insert a record into the heap file that is too long. To achieve this, a record of size 1028 is created and attempted to be inserted into the heap file. Since the maximum size of the record is 1024, this operation fails as expected.

## B+ Tree Tests

- Test 0:
    1. This test creates a B+ Tree file with prefix "AAA" followed by an integer. The file is created with naïve delete option. The file context is then switched to this newly created file.
- Test 1:
    1. This test creates a B+ Tree file with prefix "AAA" followed by an integer. The file is created with full delete option. The file context is then switched to this newly created file.
- Test 2:
    1. This test is used to print the entire B+ Tree file.
    2. The first column represents the height of the tree.
    3. The remaining columns represent the page id of the nodes at their respective levels.
- Test 3:
    1. This test is used to print the contents of all the left pages present in the B+ tree file.
- Test 4:
    1. This test takes a page id (internal or leaf node) as input. It then prints the contents of the corresponding page.
- Test 5:
    1. This test is used to insert records into the B+ Tree file.
    2. If number of keys present in the node exceeds 62, the node is split, and the tree is re-structured accordingly to maintain balance.
    3. Only positive integers are inserted into the tree.
- Test 6:
    1. This test is used to delete records from the B+ Tree file.
    2. Upon deletion, the tree is re-structured accordingly to maintain balance.
    3. If duplicates exist, only one instance of the record is deleted.
- Test 7:
    1. This test is used to insert "n" records into the B+ Tree structure.

2. This test takes the value of "n" as input from the user. It then inserts "n" records into the B+ Tree file with the key values ranging from 0 to n-1 in ascending order.

- Test 8:
  1. This test is used to insert "n" records into the B+ Tree structure in reverse order.
  2. This test takes the value of "n" as input from the user. It then inserts "n" records into the B+ Tree file with the key values ranging from 1 to n in descending order.
- Test 9:
  1. This test is used to insert "n" records into the B+ Tree structure in random order.
  2. This test takes the value of "n" as input from the user. It then inserts "n" records into the B+ Tree file with the keys inserted in random order.
- Test 10:
  1. This test is used to insert "n" records into the B+ Tree file in random order, followed by deleting "m" records from the B+ Tree file in random order.
  2. This test takes the value of "n" and "m" as input from the user. It then inserts "n" records into the B+ Tree file with the keys inserted in random order. This is followed by deleting "m" records from the B+ Tree file in random order.
- Test 11:
  1. This test is used to delete a range of records from the B+ Tree file.
  2. This test takes the lower limit and the upper limit as the two inputs from the user.
  3. It then deletes all the records corresponding to the key values in the range [lower integer key, upper integer key]
- Test 12:
  1. This test is used to verify the initialization of scan on the B+ Tree files.
  2. This test takes the lower limit and the upper limit as the two inputs from the user.
  3. All the records in the range [lower integer key, upper integer key] are retrieved and can be used for Test 13 and Test 14.
- Test 13:
  1. This test is used to verify the scanning operation on the B+ Tree files.
  2. The scan is based on the range as initialized by the user in Test 12. The test returns the record pointed to by the scan pointer and advances the pointer to the next record.
  3. When the scan is initialized to the last element, it displays "AT THE END OF SCAN"
- Test 14:
  1. This test is used to verify the deletion operation on the B+ Tree files.

2. The scan is based on the range as initialized by the user in Test 12 and is a continuation of Test 13.
3. The record which is scanned is deleted and the scan is set to its next record before deletion. In case of no more records to delete, an error message is displayed.

- Test 15:
  1. This test is used to insert "n" records into the B+ Tree file in random order, followed by deleting "m" records in the B+ Tree file in random order. This test is same as Test 10 except string values are used as keys instead of integer values.
  2. This test takes the value of "n" and "m" as input from the user. It then inserts "n" records into the B+ Tree file with the keys inserted in random order. This is followed by deleting "m" records from the B+ Tree file in random order.
- Test 16:
  1. This test is used to verify the close operation of a file.
  2. The current file in use is closed and any insert/delete operations performed on this file will display an error message.
- Test 17:
  1. This test is used to verify the open operation of a file.
  2. This test takes an integer as input. The "AAAk" file is then open where "k" is the integer accepted as input.
  3. Any operation such as insertion or deletion will be performed on this opened file. If the file does not exist, then an error message is displayed.
- Test 18:
  1. The test is used to verify the deletion operation of a file.
  2. This test takes an integer as input. The "AAAk" file is then deleted where "k" is the integer accepted as input. If the file is not present in the database, then an error message is displayed.
- Test 19:
  1. This option is used to quit the interactive interface.

# Index Tests

- Test 1:
  1. This test involves creating and scanning an index file.
  2. The following are the steps involved:
     a. A heap file ("test1.in") is created.
     b. Tuples are created using the data present in array "data1" and are inserted into the heap file.

c. A heap file scan is initiated on the newly created heap file and the scanned records are inserted into a new B+ Tree file ("BTreeIndex").
d. This is then followed by initiating an index scan on the B+ Tree file. The read records are checked for data and order consistency.

- Test 2:
  1. This test involves verifying index scanning operation on an index file.
  2. The following are the steps involved:
     a. The heap file ("test1.in") and the index file ("BTreeIndex") created in Test 1 is used in this test.
     b. The entity selection is set to "dsilva" and an index scan is done on the opened B+ Tree file to find the match for the selected entity. Validation steps are added to ensure the consistency of the result returned.
     c. A range search is performed on the index file with the range set between "dsilva" and "yuc". Validation steps are added to ensure the consistency of the result returned.
- Test 3:
  1. This test involves creating an index file and verifying the scanning operation.
  2. The following are the steps involved:
     a. A new heap file is created ("test3.in").
     b. Tuples are inserted into the heap file in random order. Each tuple consists of 3 types of fields, namely, integer, float and string.
     c. A B+ Tree index file ("BTIndex") is created on the integer field of the records present in the heap file ("test3.in").
     d. A range search is performed on the index file with the range set between 100 and 900. Validation steps are added to ensure the consistency of the result returned.

# Join Tests

- Query 1:
  1. Query: Find the names of sailors who have reserved boat number 1 and print out the date of reservation.
  2. SQL Equivalent: SELECT S.sname, R.date FROM Sailors S, Reserves R WHERE S.sid = R.sid AND R.bid = 1
  3. The steps involved are as follows:
     a. Construct the query condition as per the requirement (R.bid = 1).
     b. Obtain the scan pointer on "sailors.in" heap file in variable "am".
     c. Obtain the scan pointer on "reserves.in" heap file in variable "am2".

    d. Using sort and merge join operation, obtain all the records that satisfy the constraints. The columns, “sname” and “data”, are projected as result of the query.

- Query 2:
    1. Query: Find the names of sailors who have reserved a red boat and return them in alphabetical order.
    2. 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
    3. The steps involved are as follows:
        a. Create a scan on the heap file (“sailors.in”). Using this scan, read the records from the heap file and create a B+ Tree index file on field “sid”.
        b. Using the index scan on the B+ Tree index and records present “reservers.in” heap file, perform a nested loop join on field “sid”. Two fields, namely, “sname” and “bid” are projected as result.
        c. Records are selected from “boats.in” heap file that satisfy the condition “color=red”.
        d. The results from steps “b” and “c” are joined on field “sid” and “sname” is projected as the result. This result is then sorted in ascending order and is printed on the console.
- Query 3:
    1. Query: Find the names of sailors who have reserved a boat.
    2. SQL Equivalent: SELECT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
    3. The steps involved are as follows:
        a. Obtain the scan pointer on “sailors.in” heap file in variable “am”.
        b. Obtain the scan pointer on “reserves.in” heap file in variable “am2”.
        c. Using sort and merge join operation, obtain all the records that satisfy the constraints. The “sname” column is projected as result of the query.
- Query 4:
    1. Query: Find the names of sailors who have reserved a boat and print each name once.
    2. SQL Equivalent: SELECT DISTINCT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
    3. The steps involved are as follows:
        a. Obtain the scan pointer on “sailors.in” heap file in variable “am”.
        b. Obtain the scan pointer on “reserves.in” heap file in variable “am2”.
        c. Using sort and merge join operation, obtain all the records that satisfy the constraints. The “sname” column is projected. This result is further investigated, and duplicates are eliminated. This is the result of the query.

- Query 5:
    1. Query: Find the names of old sailors or sailors with a rating less than 7, who have reserved a boat.
    2. 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)
    3. The steps involved are as follows:
        a. Construct the query condition as per the requirement (S.age > 40 || S.rating < 7).
        b. Obtain the scan pointer on “sailors.in” heap file in variable “am”.
        c. Obtain the scan pointer on “reserves.in” heap file in variable “am2”.
        d. Using sort and merge join operation, obtain all the records that satisfy the constraints. The columns, “sname”, “age” and “rating”, are projected as result of the query.
- Query 6:
    1. 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.
    2. 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
    3. The steps involved are as follows:
        a. The records present in heap file “sailors.in” with condition “rating>7” is retrieved using a file scan and is stored in “am”. This result is joined with the records in “reserves.in” based on field “sid”. The columns “sname” and “bid” are projected.
        b. The records present in heap file “boats.in” with condition “color=red” is retrieved using a file scan. This result is joined with the records obtain from the previous step based on field “sid”. Finally, “sname” is projected as the result in ascending order.

# Sort Tests

- Query 1:
    1. This test involves verifying the sort operations.
    2. It involves the following steps:
        a. We have two arrays of strings where the first array is not sorted while the second array is sorted.
        b. Tuples are created using the data present in unsorted array and are inserted into a heap file (“test1.in”).

c. A heap file scan is initiated on the newly created heap file and the heap file is sorted.
d. This is then followed by initiating a scan on the heap file. The read records are checked for data and order consistency against the sorted array.

- Query 2:
    1. This test verifies the sort operation and is the same as Query 1, except that the sort order is reversed.
- Query 3:
    1. This test involves verifying the sort operations.
    2. It involves the following steps:
        a. An unsorted heap file is created ("test3.in"). 1000 records are inserted into this heap file. Each record has 3 types of components, namely, integer, float and string.
        b. The heap file is sorted in ascending order based on the integer field. The result is then validated for consistency.
        c. The heap file is sorted in descending order based on the float field. The result is then validated for consistency.
- Query 4:
    3. This test involves verifying the sort operations.
    4. It involves the following steps:
        a. We have two arrays of strings where the first array is not sorted while the second array is sorted.
        b. Tuples are created using the data present in unsorted array and are inserted into two heap files ("test4-1.in" and "test4-2.in").
        c. The heap file "test4-1.in" is sorted in ascending order. The records are then checked for data and order consistency.
        d. The heap file "test4-2.in" is sorted in descending order. The records are then checked for data and order consistency.

## Sort Merge Join Tests

- Query 1:
    1. Query: Find the names of sailors who have reserved boat number 1 and print out the date of reservation.
    2. SQL Equivalent: SELECT S.sname, R.date FROM Sailors S, Reserves R WHERE S.sid = R.sid AND R.bid = 1
    3. The steps involved are as follows:
        a. Construct the query condition as per the requirement (R.bid = 1).
        b. Obtain the scan pointer on "sailors.in" heap file in variable "am".

c. Obtain the scan pointer on “reserves.in” heap file in variable “am2”.
d. Using sort and merge join operation, obtain all the records that satisfy the constraints. The columns, “sname” and “data”, are projected as result of the query.

- Query 3:
  1. Query: Find the names of sailors who have reserved a boat.
  2. SQL Equivalent: SELECT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
  3. The steps involved are as follows:
     a. Obtain the scan pointer on “sailors.in” heap file in variable “am”.
     b. Obtain the scan pointer on “reserves.in” heap file in variable “am2”.
     c. Using sort and merge join operation, obtain all the records that satisfy the constraints. The “sname” column is projected as result of the query.
- Query 4:
  1. Query: Find the names of sailors who have reserved a boat and print each name once.
  2. SQL Equivalent: SELECT DISTINCT S.sname FROM Sailors S, Reserves R WHERE S.sid = R.sid
  3. The steps involved are as follows:
     a. Obtain the scan pointer on “sailors.in” heap file in variable “am”.
     b. Obtain the scan pointer on “reserves.in” heap file in variable “am2”.
     c. Using sort and merge join operation, obtain all the records that satisfy the constraints. The “sname” column is projected. This result is further investigated, and duplicates are eliminated. This is the result of the query.
- Query 5:
  1. Query: Find the names of old sailors or sailors with a rating less than 7, who have reserved a boat.
  2. 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)
  3. The steps involved are as follows:
     a. Construct the query condition as per the requirement (S.age > 40 || S.rating < 7).
     b. Obtain the scan pointer on “sailors.in” heap file in variable “am”.
     c. Obtain the scan pointer on “reserves.in” heap file in variable “am2”.
     d. Using sort and merge join operation, obtain all the records that satisfy the constraints. The columns, “sname”, “age” and “rating”, are projected as result of the query.

# CONCLUSIONS

In this phase of the project, we got a glimpse of the various components that constitute a relational database engine. With the help of the interactive and the non-interactive test cases, we were able to understand the different functionalities of each component and how the individual components interact with each other to handle all the responsibilities of a database management system. Also, we were able to explore the various techniques used for joins in a relational database management system.

# BIBLIOGRAPHY

1. http://research.cs.wisc.edu/coral/mini_doc/minibase.html
2. https://en.wikipedia.org/wiki/Relational_model
3. https://en.wikipedia.org/wiki/Database
4. https://en.wikipedia.org/wiki/Data_model
5. https://en.wikipedia.org/wiki/Database_schema
6. https://en.wikipedia.org/wiki/Relational_algebra
7. https://en.wikipedia.org/wiki/Relational_calculus
8. https://en.wikipedia.org/wiki/SQL
9. http://research.cs.wisc.edu/coral/minibase/bufMgr/bufMgr.html
10. https://www.techopedia.com/definition/3854/parser
11. https://www.techopedia.com/definition/26224/query-optimizer
12. http://pages.cs.wisc.edu/~dbbook/openAccess/Minibase/spaceMgr/heap_file.html
13. http://pages.cs.wisc.edu/~dbbook/openAccess/Minibase/spaceMgr/dsm.html
14. https://en.wikipedia.org/wiki/B%2B_tree

# APPENDIX

1. Typescript file

```
Script started on Sun 04 Feb 2018 05:33:16 PM MST
adityac@aditya-ubuntu [32m~/Documents/DBMSI/minibase/javaminibase/src[33m (master)[00m
$ make test
cd tests; make bmtest dbtest; whoami; make hftest bttest indextest jointest sorttest
sortmerge
make[1]: Entering directory
'/home/adityac/Documents/DBMSI/minibase/javaminibase/src/tests'
/usr/lib/jvm/java-8-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java
BMTest.java
/usr/lib/jvm/java-8-openjdk-amd64/bin/java  -classpath .:.. tests.BMTest

Running Buffer Management tests....
Replacer: Clock


  Test 1 does a simple test of normal buffer manager operations:
  - Allocate a bunch of new pages
  - Write something on each one
  - Read that something back from each one
   (because we're buffering, this is where most of the writes happen)
  - Free the pages again
  Test 1 completed successfully.

  Test 2 exercises some illegal buffer manager operations:
  - Try to pin more pages than there are frames
*** Pinning too many pages
  --> Failed as expected

  - Try to free a doubly-pinned page
*** Freeing a pinned page
  --> Failed as expected

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

  Test 2 completed successfully.

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

...Buffer Management tests completely successfully.

/usr/lib/jvm/java-8-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java
DBTest.java
/usr/lib/jvm/java-8-openjdk-amd64/bin/java  -classpath .:.. tests.DBTest

Running Disk Space Management tests....

Replacer: Clock


  Test 1 creates a new database and does some tests of normal operations:
```

```
  - Add some file entries
  - Allocate a run of pages
  - Write something on some of them
  - Deallocate the rest of them
  Test 1 completed successfully.

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

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

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

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

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

  - Try to add a file entry that's already there
**** Adding a duplicate file entry
  --> Failed as expected

  - Try to add a file entry whose name is too long
**** Adding a file entry with too long a name
  --> Failed as expected

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

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

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

  Test 3 completed successfully.

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

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

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

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

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

...Disk Space Management tests completely successfully.

make[1]: Leaving directory
'/home/adityac/Documents/DBMSI/minibase/javaminibase/src/tests'
adityac
make[1]: Entering directory
'/home/adityac/Documents/DBMSI/minibase/javaminibase/src/tests'
/usr/lib/jvm/java-8-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java
HFTest.java
/usr/lib/jvm/java-8-openjdk-amd64/bin/java  -classpath .:.. tests.HFTest

Running Heap File tests....

Replacer: Clock


  Test 1: Insert and scan fixed-size records

  - Create a heap file

  - Add 100 records to the file

  - Scan the records just inserted

  Test 1 completed successfully.


  Test 2: Delete fixed-size records

  - Open the same heap file as test 1

  - Delete half the records

  - Scan the remaining records

  Test 2 completed successfully.


  Test 3: Update fixed-size records

  - Open the same heap file as tests 1 and 2

  - Change the records

  - Check that the updates are really there

  Test 3 completed successfully.
```

```
  Test 4: Test some error conditions

  - Try to change the size of a record

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

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

  - Try to insert a record that's too long

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

  Test 4 completed successfully.


...Heap File tests completely successfully.

/usr/lib/jvm/java-8-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java
BTTest.java
/usr/lib/jvm/java-8-openjdk-amd64/bin/java  -classpath .:.. tests.BTTest
Replacer: Clock


Running  tests....

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


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

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

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

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

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

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

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

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

```
[19]  Quit!
Hi, make your choice :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:
4
-------------------------- 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:
4
-------------------------- 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:
100
 ***************** 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 ] )
************** 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:
6

**************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 ********

------------------------- 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:
100
 ***************** 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:
100
Please input the number of keys to delete:
20
 ***************** 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
Please input the HIGHER integer key(>=0)
20
------------------------- 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 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 :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: 21 [ 21 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 :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:
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
Hi, make your choice :17
4
 ***************** You open the file: 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 :18
5
 ****** You destroy the file: 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 :19

... Finished .


/usr/lib/jvm/java-8-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java
IndexTest.java
/usr/lib/jvm/java-8-openjdk-amd64/bin/java  -classpath .:.. tests.IndexTest

Running Index tests....

Replacer: Clock

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

BTreeIndex file created successfully.

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

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

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

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

BTreeIndex file created successfully.

Test3 -- Index scan on int key OK

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


...Index tests
completely successfully
.


Index tests completed successfully
/usr/lib/jvm/java-8-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java
JoinTest.java
Note: JoinTest.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
/usr/lib/jvm/java-8-openjdk-amd64/bin/java  -classpath .:.. tests.JoinTest
Replacer: Clock



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

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

  SELECT S.sname, R.date
  FROM   Sailors S, Reserves R
```

```
  WHERE  S.sid = R.sid AND R.bid = 1


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

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



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

  SELECT   S.sname
  FROM     Sailors S, Boats B, Reserves R
  WHERE    S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
  ORDER BY S.sname
Plan used:
 Sort (Pi(sname) (Sigma(B.color='red')  |><|  Pi(sname, bid) (S  |><|  R)))

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

After Building btree index on sailors.sid.

[David Dewitt]
[Mike Carey]
[Raghu Ramakrishnan]
[Yannis Ioannidis]

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



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

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

(Tests FileScan, Projection, and SortMerge Join.)

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

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

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

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

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

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

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



**********************Query5 strating ********************
Query: Find the names of old sailors or sailors with a rating less
       than 7, who have reserved a boat, (perhaps to increase the
       amount they have to pay to make a reservation).

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

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

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

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



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

  SELECT   S.sname
  FROM     Sailors S, Boats B, Reserves R
  WHERE    S.sid = R.sid AND S.rating > 7 AND R.bid = B.bid
           AND B.color = 'red'
  ORDER BY S.name

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

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

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

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



Finished joins testing
join tests completed successfully
/usr/lib/jvm/java-8-openjdk-amd64/bin/javac -classpath .:.. TestDriver.java
SortTest.java
/usr/lib/jvm/java-8-openjdk-amd64/bin/java  -classpath .:.. tests.SortTest

Running Sort tests....

Replacer: Clock

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

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

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

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

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

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


...Sort tests
completely successfully
.


Sorting tests completed successfully
/usr/lib/jvm/java-8-openjdk-amd64/bin/javac -classpath .:.. SM_JoinTest.java
TestDriver.java
Note: SM_JoinTest.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
/usr/lib/jvm/java-8-openjdk-amd64/bin/java  -classpath .:.. tests.SM_JoinTest
Replacer: Clock



Any resemblance of persons in this database to people living or dead
is purely coincidental. The contents of this database do not reflect
the views of the University, the Computer  Sciences Department or 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/adityac/Documents/DBMSI/minibase/javaminibase/src/tests'
adityac@ubuntu [32m~/Documents/DBMSI/minibase/javaminibase/src[33m (master)[00m $ exit

Script done on Sun 04 Feb 2018 05:43:22 PM MST
```