Phase #1: Test and Document the internals of the MiniBase

CSE 510: Database Management Systems Implementation

Arizona State University - Spring 2022

Author – Sharanya Banerjee (*sbaner67@asu.edu*)

GROUP 10:

Sharanya Banerjee
Aishwarya Balaji Rao
Manisha Nandkumar Phadke
Lalit Kumar Tiwari
Manoj Mysore Srinath
Vivek Keshava

ABSTRACT: This document serves as the report to the first phase of the project. In this phase, we run a few tests (from the given test-bench) on MiniBase. The tests focus on the key aspects of the MiniBase architecture, such as the Buffer Manager, Heap, Disk Space Allocator, and the B-tree structure. The codebase is in Java and has been tested on a Linux based system. The typescript generated from running the tests has been included in the Appendix.

KEYWORDS: Minibase, Heap, B-Tree, Disk-Space Allocator, Buffer Manager, Merge test, Sort test.

INTRODUCTION

The first phase of the project is intended to showcase an understanding of Minibase internals and architecture. We were provided with a Minjava (*tar.gz*) file which was to be extracted; following which a few dependencies (E.g.: *JAVA_PATH, LIB_PATH*) were to be set (as is in the local machine) in the Makefile. $*make db* was then used to build the database and $*make test* was executed to run the test bench. Several tests executed automatically, with a few requiring user interactions from a menu-driven process. The code included an interface that allowed a user to interact with the MiniBase B-tree. All the outcomes from the tests were recorded on a typescript. Majority of the tests focused on verifying that the expected functionality was achieved as well as ensuring that the correct exception was thrown under edge case execution. In addition, the file provided to us included documentation pertaining to the general class layout of the Mini Base implementation. The documentation includes all method exceptions as well as any method the class might implement. This showcased an idea about the internal architecture of Minibase as well as some important boundary conditions.

DESCRIPTION OF TESTS

Buffer Manager:

The initial set of tests focused entirely on the buffer manager's operation where pages can be referenced or pinned to the buffer, under a clock which runs indefinitely until all pages have been unpinned and none have been referenced. If a page is not pinned or referenced at any moment, it is made available.

First Test – The main class JavabaseBM performs the first test to simulate working of the system during normal DB operations. Initially, some space is allocated in the buffer in the form of new pages and write operations are performed on each one. Proceeding that, read operations are performed by referencing the pages and then the pages are unpinned, and the buffer memory is made available again. Since the number of pages pinned is less than or equal to the number of frames available, the test executes successfully.

Second Test – This test tries an edge case where the number of pages being pinned is more than the number of frames available. This test is performed to ensure the system handles exceptions properly. While running this test, the buffer pool is filled to the brim. To indicate this, the getNumUnpinnedBuffers() function returns 0. Under this condition, the buffer pool returns an exception *BufferPoolExceededException*. The method pinPage() function is additionally used to prevent a page from being pinned twice and then freed without an exception being thrown. When a doubly pinned page is released, an exception in the class *ChainException* is thrown. Along with this, the test tries to unpin a page that is in not pinned by checking *pid=lastPid*. All the exceptions are caught, and the tests fail as expected.

Third Test – The third test is used to interact with the buffer internals. Some pages (based on integers assigned) are allocated onto the buffer and write operations are performed on them one by one, to make them dirty. After this, some are left pinned while the others are released. Then the pinned pages are read from to ensure that the written data is accurate. This test concludes successfully.

Buffer Manager Test outcome – Buffer Manager tests completed successfully.

Disk Space Management:

This segment of tests is performed to understand how Minibase performs disk space management under a clock.

First Test – This test creates a new database and adds some file entries into it. It then allocates a few pages from the pool and performs a write operation onto them. Consequently, it deallocates the empty pages. The first test concludes successfully.

Second Test – This test accesses the aforementioned DB and deletes a few of the file entries from it. It then searches the remnant pages and reads from them to retrieve the data we wrote in the first test. The second test concludes successfully.

Third Test – Test three set is used to simulate error conditions. When a file entry is deleted, the *Pgid* is set to Null. In this test, first we try to lookup an already deleted file and then we try to delete the file. In either case, the system fails to locate that file and throws the *FileEntryNotFoundException*. Also, when the test tries to lookup and/or delete a non-existent file, the same exception is thrown. When a test attempts to insert a duplicate file entry (same name), the system throws the *DuplicateEntryException* and for inserting a file entry with a name longer than stipulated, it throws the *FileNameTooLongException.* The run allocates and de-allocates (for reverse entry) option additionally ensures that the run used is always a positive integer and within the stipulated boundary size. When the test violates these clauses, the *InvalidRunSizeException* is thrown. All over the edge cases tests hence failed, as expected.

Fourth Test – The last set of tests work the system on certain edge cases to push the Disk Manager to its limit. While the disk space was full and the number of unpinned pins were 0, the test attempts to pin a new page wherein the system threw an *OutOfSpaceException*. The test then de-allocates multiple run of pages, proceeding which it tries to allocate a particular deallocated page. The Disk Manager class has exceptions in place to ensure that these pages are being re-allocated in the right order. The next test tries to allocate and de-allocate collections of pages in the same manner. The final check aims to allocate more pages than should be available to check overflow condition. The last two pages are then de-allocated which tests the space map boundary condition. All exceptions were invoked and test 4 also failed as expected.

Disk Manager Test outcome – Disk Manager tests completed successfully.

Heap File Tests:

The Heap File class maintains an unordered set of records with their respective unique *Record_ID*. We can perform insertion, deletion, open, scan and close operations on any of these files. The test in this segment aims to work on this heap.

First Test – This test inserts 100 records (int, float, length, name) into a newly created heap file and scans all the inserted files to ensure that the files have been correctly inserted with desired attributes while none have been pinned to the buffer pre-scan. Post scan, all records are unpinned. The first test passes successfully.

Second Test – This test deletes half the records previously inserted (in odd positions). Error handlers ensure all records are unpinned after deletion. Like the first test, a final scan of the remaining records (in even positions) ensures the outcome of the delete operation. This test also passes successfully.

Third Test – The penultimate test aims to update some fixed-sized records. First the records were scanned and the new content was stored in tuples; after which the updateRecord() function sets the updates in place. Invalid file updates are handled by the *InvalidUpdateException*. A final scan verifies this, and the test concludes with flying colors.

Fourth Test – The final test works with record length to simulate edge cases. When the test attempts to lengthen or shrink a record by 1, the *InvalidTupleSizeException* is thrown, caught, and handled. To change the size of a record, it must be deleted and replaced by one of the updated sizes. Similarly, any insert operation on records of size greater than stipulated size are blocked. All these tests fail as expected.

Heap File Test outcome – Heap File tests completed successfully.

B Tree

This segment of tests allows the user to interact with the B-Tree structure of Minibase through a menu driven interface. There are 20 tests pertaining to B-Tree functioning, numbered [0] to [19].

Test 0 – Performs a naïve delete on a new file.
Test 1 – Performs a full delete on a new file.
Test 2 – Prints the B+ Tree structure while displaying the current page ID being referenced.
Test 3 – Displays all the pages currently in the B-Tree. For an empty tree, it displays the corresponding message.
Test 4 - Gives the user an option to select a page to print from the B-Tree. For a missing page or invalid input, it displays the corresponding message.
Test 5 – Inserts a record into the B-Tree, based on user input.
Test 6 – Deletes a record currently in the B-Tree. For empty tree, leaves it unaltered.
Test 7 – Takes integer input n from the user and inserts records in order, from 0 to n.
Test 8 – Takes integer input n from the user and inserts records in reverse order, from n to 0.
Test 9 – Takes integer input n from the user and inserts n records in random order.
Test 10 – Takes integer inputs from the user as n and m. It then inserts n records randomly into the tree and deleted m records randomly from the tree.
Test 11 – Deletes some records from the tree in random order.
Test 12 – Starts a scan on the B-Tree.
Test 13 – Scans the next record as scan(pid=current_pid+1).
Test 14 – Deletes the record scanned above, if present. Since the record to be deleted is not present, the system throws the *btree.ScanDeleteException*.
Test 15 – Takes integer inputs from the user as n and m. It then inserts n records randomly into the tree and deleted m records randomly from the tree.
Test 16 – Closes the file.
Test 17 – Opens a file with filename <integer> input by the user.
Test 18 – Destroys a file with filename <integer> input by the user.
Test 19 – Exits the menu and proceedes with Index tests.

B-Tree Test outcome – B-Tree tests completed successfully.

Index Tests:

This segment of testing deals with indexing of the file system.

Test 1 – At the onset, a new record and heap file is created. Once this is cleared, a scan is started using the BTreeFile class. Indexing is performed using a key based ordering scheme which enables pruning to make the system more efficient. These tests pass and a status ok message is returned.

Test 2 – The second test opens and scans the newly indexed B-Tree to verify its accuracy. A control statement prevents the scan to go beyond the stipulated boundary and return a NULL value.

Test 3 – This test performs a range scan on the newly opened and indexed file to reflect any changes made. This scan also verifies that the records are stored in a sorted manner, as it should be in a B-Tree. This test too passes with flying colors.

Index Test outcome – Index tests completed successfully.

Join Tests:

Join Tests performs query executions (application-level interactions) to identify how Minibase performs join operations on tables stored in the DB.

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

Output –
[Mike Carey, 05/10/95]
[David Dewitt, 05/11/95]
[Jeff Naughton, 05/12/95]

This query finds the names of sailors who have booked boat 1 and displays their reservation dates. The query joins the Sailors table with Reserves table using *sid* parameter and checks for objects with boat number *R.bid=1*.

Query 2 – 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

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

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

Query 2 aims to execute a nested-join on 3 tables: Sailors, Reserves and Boats. Consequently, an inner join is performed with the resultant subset of tables and Boats table. Post that, the names are returned in sorted order.

Query 3 – SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid

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

This query returns the names (with repetition) of the all sailors who have reserved a boat by performing a join on Sailors and Reserves table, based on their *sid*.

Query 4 – SELECT DISTINCT S.sname
FROM Sailors S, Reserves R
WHERE S.sid = R.sid

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

This query returns the names (without repetition) of the all sailors who have reserved a boat by performing a join on Sailors and Reserves table, based on their *sid*. The *DISTINCT* keyword eliminates repetition.

Query 5 – SELECT S.sname, S.rating, S.age
FROM Sailors S, Reserves R
WHERE S.sid = R.sid and (S.age > 40 || S.rating < 7)

Output –
[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]

Query 5 again performs a join on Sailors and Reserves table based on *sid*, but now it also checks for all those sailors who have reserved boats whose ages are greater than 40, or ratings are less than 7; maybe in order to enable variable pricing on rentals based on lessee's demography.

Query 6 - 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

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

Output –
[David Dewitt]
[Mike Carey]
[Raghu Ramakrishnan]
[Yannis Ioannidis]
```

The final query finds the names of all sailors who having a rating greater than 7, have reserved a red boat and displays them in a sorted manner. This is done by performing 3 join operations based on *sid* and *bid* and *color* attributes, followed by an ORDER BY operation which sorts the output. This concludes all join tests.

Join Test outcome – Join tests completed successfully.

Sort Tests:

The final test segment works with the Sort class, which works to sort unordered heap files by passing necessary arguments to the constructors. The *SortException* class handles caught exceptions while performing sort operations and get_next() function is used to access following tuples.

Test 1 & 2 – These tests take two unsorted files and perform sort operations on them. The sorted output is then scanned to verify that the entries are in order. These tests pass as expected.

Test 3 – This is the final test in the test bench. First this test aims to sort tuples in ascending order on the <int> field and then attempts to sort them in descending order on the <float> field, independently. Post sorting, the entries are scanned for verification.

The test here all revolve around the Sort class. The exception handlers are all tested here to make sure all errors are caught and handled. This class essentially sorts a file. All necessary information is passed as arguments to the constructor. Then the user can call get_next to tuples back in sorted order. The first two test this class by using it on two unsorted files, the output is then verified using an iterator, which is obtained from running a scan operation to ensure all entries are in the right order.

Sort Test outcome – Sort tests completed successfully.

Sort-Merge Tests:

Essentially this segment of tests is very similar to the Join test segment with one key difference. In Join tests, first the query plan joins the two tables based on some criteria and then sort the resultant tuples (O(log(n)) complexity). So, this is a more optimized QEP because the system has to sort lesser tuples. Unlike this, the Sort-Merge tests have a QEP to sort individual tuples first ($O(n^2)$ complexity) and then merge the tuples. This would essentially take more time to execute because the system must sort 2 tables instead of one as in Join test. However, the resultant outputs would still be the same regardless (as can be seen from Query 1,3,4,5 of Join test segment; Query 2,6 is omitted). This test passes successfully and concludes ***Phase #1 – Testing*** of our project.

Sort-Merge Test outcome – Sort-Merge tests completed successfully.

CONCLUSION

The test displayed all the functionality of the internal architecture of MiniBase. All the functionality was examined, along with many aspects of the operation. Additionally, all potential entries were checked to ensure that edge cases were handled correctly and efficiently. Furthermore, the detailed documentation contributes to a tidy overall design. This phase of the project allowed the team to familiarize themselves with Minibase by understanding what was tested and the reason behind it.

BIBLIOGRAPHY

1. Source file: ../../users/sharanyabanerjee/Desktop/Assignments/CSE510 DBMSI/minjava/javaminibase/src/javadoc/overview-frame.html (Hosted on local system).
2. https://research.cs.wisc.edu/coral/minibase/minibase.html
3. https://research.cs.wisc.edu/coral/minibase/project.html
4. https://en.wikipedia.org/wiki/Sort-merge_join
5. https://www.sqlshack.com/sql-server-query-execution-plan-beginners-types-options/
6. https://www.tutorialspoint.com/unix_commands/make.html

APPENDIX

Typescript:

```
 Script started on Sat Jan 29 20:31:31 2022
[1m[7m%[27m[1m[0m


[0m[27m[24m[J(base) sharanyabanerjee@Sharanyas-MacBook-Pro src % [K[?2004hmmake test[?2004l

cd tests; make bmtest dbtest; whoami; make hftest bttest indextest jointest sorttest sortmerge
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.jdk/Contents/Home/bin/javac -classpath .:.. TestDriver.java
BMTest.java
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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.

/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.jdk/Contents/Home/bin/javac -classpath .:.. TestDriver.java
DBTest.java
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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.

sharanyabanerjee
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.jdk/Contents/Home/bin/javac -classpath .:.. TestDriver.java
HFTest.java
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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.

/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.jdk/Contents/Home/bin/javac -classpath .:.. TestDriver.java
BTTest.java
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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 :2
The Tree is Empty!!!
------------------------- MENU ------------------



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

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

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

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

[12]  Initialize a Scan
```

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

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

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

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

[19]  Quit!
Hi, make your choice :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 :4
Please input the page number:
```

```
3
Sorry!!! This page is neither Index nor Leaf page.
------------------------- MENU -----------------


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

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

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

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

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

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

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

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

[19]  Quit!
Hi, make your choice :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 :6
Please input the integer key to delete:
2
------------------------- 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:
4
 ********** The file name is: AAA3  *********
------------------------- MENU -----------------


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

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

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

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

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

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

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

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

[19]  Quit!
Hi, make your choice :8
Please input the number of keys to insert:
2 5
 ********** The file name is: AAA4  *********
```

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


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

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

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

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

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

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

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

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

[19]  Quit!
Hi, make your choice :9
Please input the number of keys to insert:
4
 ********** The file name is: AAA5  **********
------------------------- MENU ------------------


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

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

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

[5]   Insert a Record
```

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

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

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

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

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

[19]  Quit!
Hi, make your choice :10
Please input the number of keys to insert:
3
Please input the number of keys to delete:
4
 ********** The file name is: AAA6  **********
------------------------- MENU ------------------


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

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

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

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

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

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

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

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

[19]  Quit!
Hi, make your choice :11
Please input the LOWER integer key(>=0):
2
Please input the HIGHER integer key(>=0)
5
java.lang.ArrayIndexOutOfBoundsException: Index -1 out of bounds for length 20
  at bufmgr.BufHashTbl.insert(BufMgr.java:143)
  at bufmgr.BufMgr.pinPage(BufMgr.java:524)
  at btree.BTreeFile.pinPage(BTreeFile.java:92)
  at btree.BTreeFile._Delete(BTreeFile.java:1332)
  at btree.BTreeFile.FullDelete(BTreeFile.java:1290)
  at btree.BTreeFile.Delete(BTreeFile.java:991)
  at tests.BTDriver.runAllTests(BTTest.java:249)
  at tests.BTDriver.runTests(BTTest.java:80)
  at tests.BTTest.main(BTTest.java:648)
btree.PinPageException:
java.lang.ArrayIndexOutOfBoundsException: Index -1 out of bounds for length 20
  at bufmgr.BufHashTbl.insert(BufMgr.java:143)
  at bufmgr.BufMgr.pinPage(BufMgr.java:524)
  at btree.BTreeFile.pinPage(BTreeFile.java:92)
  at btree.BTreeFile._Delete(BTreeFile.java:1332)
  at btree.BTreeFile.FullDelete(BTreeFile.java:1290)
  at btree.BTreeFile.Delete(BTreeFile.java:991)
  at tests.BTDriver.runAllTests(BTTest.java:249)
  at tests.BTDriver.runTests(BTTest.java:80)
  at tests.BTTest.main(BTTest.java:648)
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      !!         Something is wrong                    !!
      !!     Is your DB full? then exit. rerun it!     !!
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
------------------------- MENU -----------------


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

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

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

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

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

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

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

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

[19]  Quit!
Hi, make your choice :12
Please input the LOWER integer key (null if -3):
1
Please input the HIGHER integer key (null if -2):
5
------------------------- MENU ------------------


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

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

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

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

[12]  Initialize a Scan
[13]  Scan the next Record
```

```
[14]  Delete the just-scanned record

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

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

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

[19]  Quit!
Hi, make your choice :13
AT THE END OF SCAN!
------------------------- MENU ------------------


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

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

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

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

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

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

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

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

[19]  Quit!
Hi, make your choice :14
No Record to delete!
btree.ScanDeleteException
```

```
  at btree.BTFileScan.delete_current(BTFileScan.java:105)
  at tests.BTDriver.runAllTests(BTTest.java:278)
  at tests.BTDriver.runTests(BTTest.java:80)
  at tests.BTTest.main(BTTest.java:648)
btree.ScanDeleteException
  at btree.BTFileScan.delete_current(BTFileScan.java:121)
  at tests.BTDriver.runAllTests(BTTest.java:278)
  at tests.BTDriver.runTests(BTTest.java:80)
  at tests.BTTest.main(BTTest.java:648)
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      !!         Something is wrong                    !!
      !!     Is your DB full? then exit. rerun it!     !!
      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
------------------------- MENU ------------------


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

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

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

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

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

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

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

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

[19]  Quit!
Hi, make your choice :15
Please input the number of keys to insert:
"5"
Please input the number of keys 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 :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
3
 ********** You open the file: 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 :18
4
 ********** You destroy 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 :19

... Finished .


/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.jdk/Contents/Home/bin/javac -classpath .:.. TestDriver.java
IndexTest.java
```

```
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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.
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.jdk/Contents/Home/bin/javac -classpath .:.. TestDriver.java
SortTest.java
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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.
/opt/homebrew/Cellar/openjdk/17.0.1_1/libexec/openjdk.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
[1m[7m%[27m[1m[0m


[0m[27m[24m[J(base) sharanyabanerjee@Sharanyas-MacBook-Pro src % [K[?2004h
```