DBMS Models and implementation (Section 005)
Instructor: Sharma Chakravarthy
Project I: B+ Tree (full deletion)

Made available on:	**9/3/2015**
Due on:	**10/1/2015(11:55Pm)**
Submit by:	**Blackboard (1 zipped folder containing all the files/sub-folders) https://elearn.uta.edu/**
Weight:	**15% of total**
Total Points:	**100**

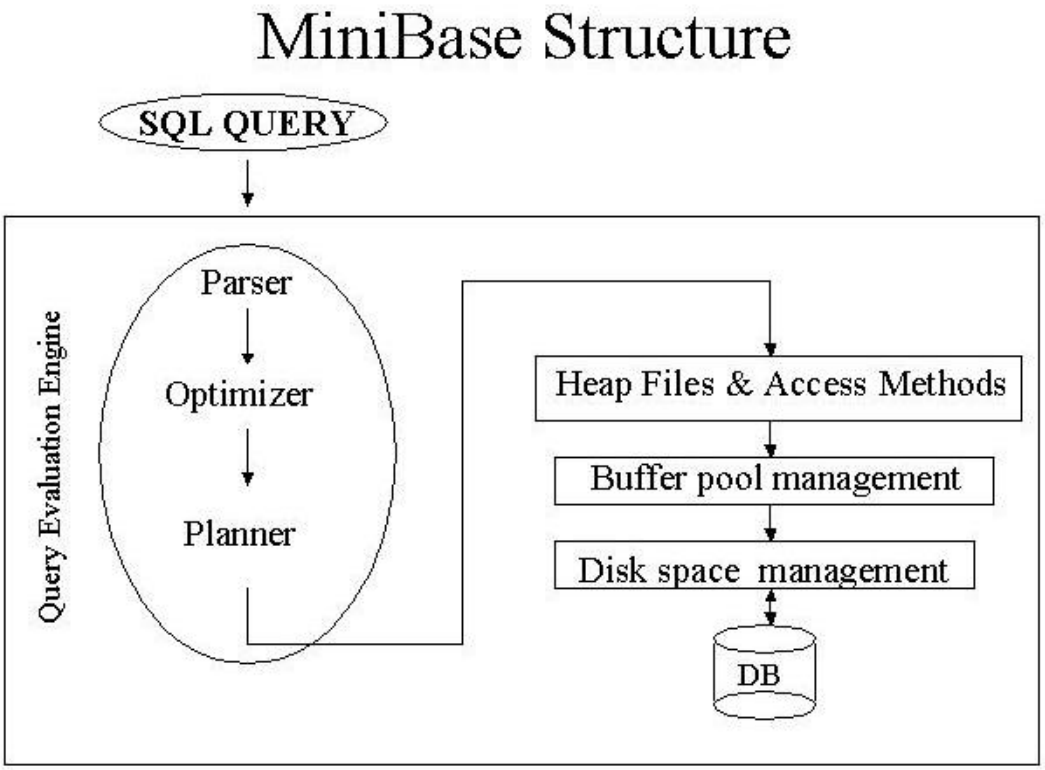
This project is on the implementation of the storage management and efficient access of records corresponding to the Files and Access Methods layer of the database management system MINIBASE. You will use some methods of Buffer Manager Class which have already been implemented.

I. Problem Statement:

In this project, you will implement a B+ tree in which leaf level pages contain entries of the form <key, rid of a data record> (Alternative 2 for data entries, in terms of the textbook on page). You must implement the full deletion algorithms as discussed in class. For this project, you need to implement delete i.e. remove the record and perform any merging or redistribution as required. Before jumping into this project, make sure you thoroughly understand pages 344 to 363 of the textbook for the algorithms.

You will be given, among all other necessary packages and classes, *HFPage* and *BTSortedPage*. *BTSortedPage* is derived from *HFPage*, and it augments the `insertRecord` method of *HFPage* by storing records on the *HFPage* in sorted order by a specified <key> value. The key value must be included as the initial part of each record, to enable easy comparison of the key value of a new record with the key values of existing records on a page. The documentation available in the java code documentation is sufficient to understand what operation each function performs (please see the javadoc URL for detailed descriptions).

You need to use two page-level classes, *BTIndexPage* and *BTLeafPage*, both of which extend *BTSortedPage*. These page classes are used to build the B+ tree index; The methods you will need to code like create, destroy, open and close a B+ tree index, and to open scans that return all data entries (from the leaf pages) which satisfy some range selection on the keys are been provided.



Overview of the Minibase along with all the layers

Use the following links for additional information on the project and Minibase, respectively:

1) https://wweb.uta.edu/faculty/sharmac/javadocs/index.html
2) https://wweb.uta.edu/faculty/sharmac/mini_doc-2.0/minibase.html

II. Design Overview:

1. A key point to remember

- You should note that key values are passed to functions using `KeyClass` objects (an abstract class). The contents of a key should be interpreted using the `AttrType` variable. The key can be either a string (`attrString`) or an integer (`attrInteger`), as per the definition of `AttrType` in `global` package. We just implement these two kinds of keys in this assignment. If the key is a string, its value is stored in a `StringKey` class which extends the `KeyClass`. Likewise, if the key is an integer, its value is stored in an `IntegerKey` class that also extends the `KeyClass`.

- Although using the above structure, keys can be of (the more general enumerated) type `AttrType`, you can return an error message if the keys are not of type `attrString` or `attrInteger`.

- The `BTSortedPage` class, which augments the `insertRecord` method of `HFPage` by storing records on a page in sorted order according to a specified `key` value, assumes that the key value is included as the initial part of each record, to enable easy comparison of the key value of a new record with the key values of existing records on a page.
  (Note. For this project, the key will be an integer (attrInteger) for simplicity)

2. Methods that need to be implemented

- **BTreeFile**

  The methods you need to be implemented for this project is BTreeFile class. BTreeFile is a derived class of the IndexFile class, which means a BTreeFile is a kind of IndexFile.

  The methods to be implemented are described under the Btree package of the given java documentation. Specifically, you will be responsible to provide code for the following method:

  - **Delete method**

    The Delete method simply removes the entry from the appropriate BTLeafPage. You need to implement redistribution or page merging when the number of entries falls below threshold but duplicate values are allowed. This method is given and is set to call _Delete() method by default.

    - _Delete() method

      In _Delete() you need to remove the data entry <key, rid> from an index and perform any merging or redistribution if the leaf page is at least half full after the deletion. Use available_space() method to determine the capacity of a page.

      Method required to compare the key is:

      ```
      int keyCompare(KeyClass key1, KeyClass key2)
      ```

      This method returns an integer if key1>key2 returns positive value, if key1<key2 it will return a negative value and return 0 if equal.

      The method required to search is already given. It returns the leaf page at the left most of the tree and then search for the key to be deleted as leaf pages are organized as a doubly link list. You can see the search algorithm in the book for more clarity.

      ```
      BTLeafPage findRunStart(key, curRid);
      ```

3. B+ Tree page level classes

- HFPage:

  This is the base class, you can look at `heap` package to get more details.

- BTSortedPage:

This class is derived from the class HFPage. It's only function is to maintain records on a HFPage in a sorted order. Only the slot directory is re-arranged. The data records remain in the same positions on the page.

- BTIndexPage:

This class is derived from BTSortedPage. It inserts records of the type < key, pageNo> on the BTSortedPage. The records are sorted by the key.

- BTLeafPage:

This class is derived from BTSortedPage. It inserts records of the type <key, dataRid> on the BTSortedPage. *dataRid* is the rid of the data record. The records are sorted by the key. Further, leaf pages must be maintained in a doubly-linked list.

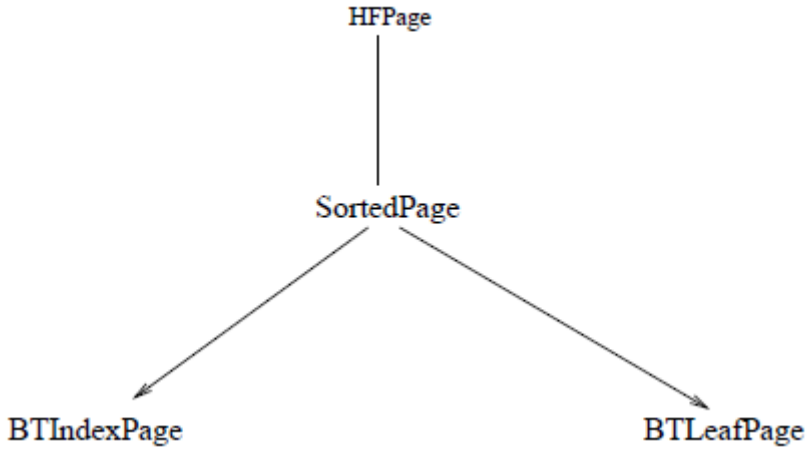


Figure 1: The classes used for the B+ Tree pages

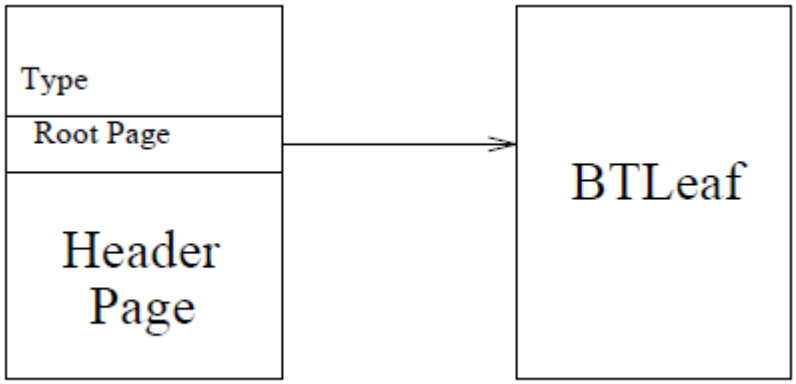


Figure 2: B+ tree with one leaf page

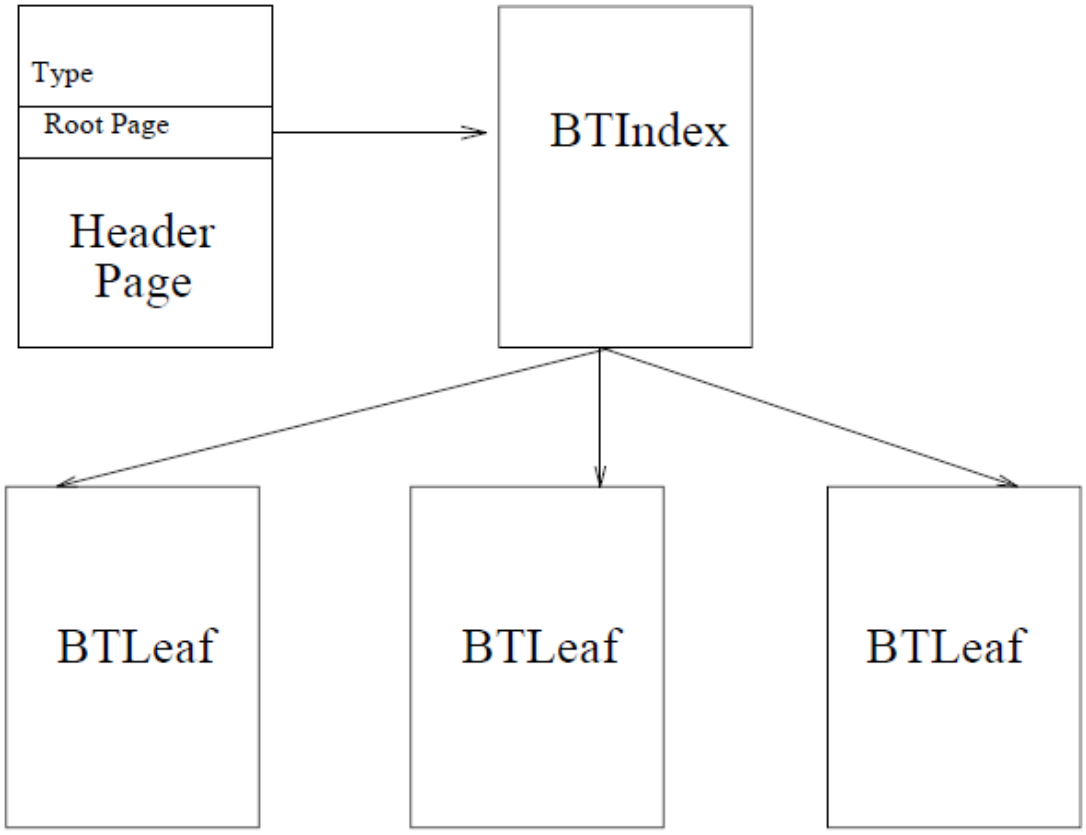


Figure 3: B+ tree with more than one leaf page

4. Other B+ tree classes:

Figure 2 shows what a BTreeFile with only one BTLeafPage looks like; the single leaf page is also the root. Note that there is *no* BTIndexPage in this case. Figure 3 shows a tree with a few BTLeafPages, and this can easily be extended to contain multiple levels of BTIndexPages as well.

- **IndexFile and IndexFileScan :**

  A BTree is one particular type of index. There are other types, for example a Hash index. However, all index types have some basic functionality in common. This basic index functionality is taken and an abstract class called IndexFile is created. However, any class derived from an IndexFile should support `IndexFile()`, `Delete()`, and `insert()`.

  Likewise, an IndexFileScan is an abstract class that contains the basic functionality all index file scans should support.

- **BTIndexPage class**

  See the explanation in previous sections.

- **BTLeafPage class**

  See the explanation in previous sections.

Note that the java documentation provided to you for methods in `BTreeFile`, `BTIndexPage`, and `BTLeafPage`, do not specify whether they throw any exceptions; or rather, what exceptions that they throw. You should follow the error protocol defined as before to implement your methods with reasonable exceptions thrown.

III. Some useful hints and tips:

- Remove the UnsupportedOperationException lines as you implement your code.
  (i.e. these are simply place holders so that the skeleton code would compile)

- Follow the example of good Java programming style set by the skeleton code.
  (i.e. you will be docked if your code isn't well-commented and/or formatted)

IV. What you are asked to do:

1. Complete the implementation of the `BTreeFile.java` skeleton class given to you

2. Implement the method `_Delete()`

3. Compile and run using the test code given to you. We may add/extend the test cases for evaluation. If you add additional test cases, please do so after the current ones and keep it to the same style.

V. Getting Started

Please copy all the files from "PATH/…/…/" into your own local directory. The files are:

- Makefile:

  A sample Makefile for you to compile your project. You will have to set up any dependencies by editing this file. You can also design your own Makefile. Whatever you do, please make sure that you use the classpath provided in the original Makefile.

- Some compiled classes, BT.class, BTSortedPage.class, etc.

  The java documentations of those classes can be found at http://path to the file. Read the classes and methods descriptions carefully, for it will help you understand the program structure.

  You can find other useful utilities in the java documentation.

If you want to use an IDE (e.g., Netbeans, Eclipse) for doing the project, you need to figure out how to create a project with the given files. Please make sure you do not delete the .class files in the bin directory as they are needed for running your code (the rest of the system is provided as .class files)

VI. Project Report

Please include (at least) the following sections in a **REPORT.{txt, pdf, doc}** file that you will turn in with your code:

- **Overall Status**
  Give a *brief* overview of how you implemented the major components. If you were unable to finish any portion of the project, please give details about what is completed and your understanding of what is not. (This information is useful when determining partial credit.)
- **File Descriptions**
  List any new files you have created and *briefly* explain their major functions and/or data structures. If you have added additional test cases, please summarize them.
- **Division of Labor**
  Describe how you divided the work, i.e. which group member did what. Please also include how much time each of you spent on this project. (This has no impact on your grade whatsoever; we will only use this as feedback in planning future projects -- so be honest!)
- **Logical errors and how you handled them**
  List at least 3 logical errors you encountered during the implementation of the project. Pick those that challenged you. This will provide us some insights into how we can improve the description and forewarn students for future assignments.

VII. What to Submit

- After you are satisfied that your code does exactly what the project requires, you may turn it in for grading. Please submit your project report and the BTree package. We will ignore source code in any other directories.
- You will turn in one zipped file containing your source code as well as the report
- All of the above files should be placed in a single zipped folder named as 'proj1_firstname_lastname_Section_final'. **Only one zipped folder should be uploaded using blackboard.**
- You can submit your zip file at most 3 times. The latest one (based on timestamp) will be used for grading. So, be careful in what you turn in and when!
- **Only one person per group should turn in the zip file!**

VIII. Coding Style:

Be sure to observe the following standard Java naming conventions and style. These will be used across all projects for this course; hence it is necessary that you understand and follow them correctly. You can look this up on the web. Remember the following:

a. Class names begin with an upper-case letter, as do any subsequent words in the class name.
b. Method names begin with a lower-case letter, and any subsequent words in the method name begin with an upper-case letter.
c. Class, instance and local variables begin with a lower-case letter, and any subsequent words in the name of that variable begin with an upper-case letter.
d. No hardwiring of constants. Constants should be declared using all upper case identifiers with _ as separators.
e. All user prompts (if any) must be clear and understandable
f. Give meaningful names for classes, methods, and variables even if they seem to be long. The point is that the names should be easy to understand for a new person looking at your code
g. Your program is properly indented to make it understandable. Proper matching of if … then … else and other control structures is important and should be easily understandable
h. Do not put multiple statements in a single line

In addition, ensure that your code is properly documented in terms of comments and other forms of documentation for generating meaningful javadoc.

IX. Grading Scheme:

The project will be graded using the following scheme:

1.	Correctness of the `_Delete` code:	70
2.	Reporting and fixing 3 to 5 logical errors in the program:	10
3.	Documentation (Javadoc) and commenting of the code: (Including coding style)	10
4.	Report:	10

Three days after the due date, the submission will be closed. Penalty for each day is 8% of the earned grade (NO partial penalty!!)

Your java program must be executable (without any modification by us) from the Linux command prompt or Cygwin command prompt (on UTA's Omega server or a PC). So, please test it for that before submitting it. However, the source code files can be created and/or edited on any editor that produces an ASCII text file. As I mentioned in the class, an IDE is not necessary for this and subsequent projects. If you decide to use it, please learn it on your own and make sure your code compiles and executes with appropriate package information.

FYI, when your code runs correctly, the output from BTTest.java should look like:

```
Replacer: Clock


Running  tests....

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

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

           ---Integer Key (for choices [3]-[5]) ---

[3]   Insert a Record
[4]   Delete a Record (Naive Delete)
[5]   Delete some records (Naive Delete)

[6]  Quit!
Hi, make your choice :3
Initially 100 values are inserted from 1 to 100.
------------------------- MENU -----------------

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

           ---Integer Key (for choices [3]-[5]) ---

[3]   Insert a Record
[4]   Delete a Record (Naive Delete)
[5]   Delete some records (Naive Delete)

[6]  Quit!
Hi, make your choice :0



---------------The B+ Tree Structure---------------
1     5
2            3
2            4
2            6
--------------- End ---------------

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

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

           ---Integer Key (for choices [3]-[5]) ---

[3]   Insert a Record
```

```
[4]   Delete a Record (Naive Delete)
[5]   Delete some records (Naive Delete)

[6]  Quit!
Hi, make your choice :2
Please input the page number:
5

**************To Print an Index Page ********
Current Page ID: 5
Left Link      : 3
0 (key, pageId):   (32,  4 )
1 (key, pageId):   (63,  6 )
************** END ********

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

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

           ---Integer Key (for choices [3]-[5]) ---

[3]   Insert a Record
[4]   Delete a Record (Naive Delete)
[5]   Delete some records (Naive Delete)

[6]  Quit!
Hi, make your choice :2
Please input the page number:
3

**************To Print an Leaf Page ********
Current Page ID: 3
Left Link      : -1
Right Link     : 4
0 (key, [pageNo, slotNo]):   (1,  [ 1 1 ] )
1 (key, [pageNo, slotNo]):   (2,  [ 2 2 ] )
2 (key, [pageNo, slotNo]):   (3,  [ 3 3 ] )
3 (key, [pageNo, slotNo]):   (4,  [ 4 4 ] )
4 (key, [pageNo, slotNo]):   (5,  [ 5 5 ] )
5 (key, [pageNo, slotNo]):   (6,  [ 6 6 ] )
6 (key, [pageNo, slotNo]):   (7,  [ 7 7 ] )
7 (key, [pageNo, slotNo]):   (8,  [ 8 8 ] )
8 (key, [pageNo, slotNo]):   (9,  [ 9 9 ] )
9 (key, [pageNo, slotNo]):   (10,  [ 10 10 ] )
10 (key, [pageNo, slotNo]):   (11,  [ 11 11 ] )
11 (key, [pageNo, slotNo]):   (12,  [ 12 12 ] )
12 (key, [pageNo, slotNo]):   (13,  [ 13 13 ] )
13 (key, [pageNo, slotNo]):   (14,  [ 14 14 ] )
14 (key, [pageNo, slotNo]):   (15,  [ 15 15 ] )
15 (key, [pageNo, slotNo]):   (16,  [ 16 16 ] )
16 (key, [pageNo, slotNo]):   (17,  [ 17 17 ] )
17 (key, [pageNo, slotNo]):   (18,  [ 18 18 ] )
18 (key, [pageNo, slotNo]):   (19,  [ 19 19 ] )
19 (key, [pageNo, slotNo]):   (20,  [ 20 20 ] )
```

```
20 (key, [pageNo, slotNo]):   (21,  [ 21 21 ] )
21 (key, [pageNo, slotNo]):   (22,  [ 22 22 ] )
22 (key, [pageNo, slotNo]):   (23,  [ 23 23 ] )
23 (key, [pageNo, slotNo]):   (24,  [ 24 24 ] )
24 (key, [pageNo, slotNo]):   (25,  [ 25 25 ] )
25 (key, [pageNo, slotNo]):   (26,  [ 26 26 ] )
26 (key, [pageNo, slotNo]):   (27,  [ 27 27 ] )
27 (key, [pageNo, slotNo]):   (28,  [ 28 28 ] )
28 (key, [pageNo, slotNo]):   (29,  [ 29 29 ] )
29 (key, [pageNo, slotNo]):   (30,  [ 30 30 ] )
30 (key, [pageNo, slotNo]):   (31,  [ 31 31 ] )
************** END ********
```