CS 587
Creators: Haomin He, Zicheng Ren

First of all, we build our project based on instructions from DbImplem-Heap File Assignment-Winter-2018.pdf. We implemented the constructor HeapFile() first. If the input name is given we need to call Minibase.DiskManager.get_file_entry(name) to get the first page of the directory for this heap file. If name is null, which means that file is not in the library, we create a new empty heapfile.

finalize() is implemented next. If heap file is temporary (isTemp==true) we call deleteFile() to delete it.

deleteFile() is called by finalize(). We need to delete heap file by scanning total entries and counting occupied entries and freeing each occupied page that resides in the entry. If there exists a heap file in the system we need to call Minibase.DiskManager.delete_file_entry(fileName) to remove it.

insertRecord() is used to insert a new record into the file and returns its RID. We first search available space in directory page, if the record length does not exceed the MAX_TUPSIZE we call getAvailPage() to get page id and call insertRecord from HFPage class to insert the record into the data page. We then call getFreeSpace(), updateDirEntry() to update entry info.

selectRecord() reads a record from the file according to its record id (rid). We created a new DataPage to hold the content of the page that record resides in. rid has page id info. We pin the page to get the content of the page, then call selectRecord() from HFPage class to keep track of the record.

updateRecord() updates the specified record in the heap file. Similar to the implementation of selectRecord(), we need to find out the page id to which record id (rid) points, then call updateRecord from HFPage class. If we encounter an invalid rid or newRecord throw illegalargumentexception.

deleteRecord() deletes the specified record from the heap file. Pin the data page that we need to delete record from. Call deleteRecord() from HFPage class to delete the record. We call getFreeSpace() from HFPage class to keep track of the free space and call updateDirEntry() to update the record page.

getRecCnt() gets the number of records in the file. We traverse through the directory pages, count the entries for each directory page and count the records within each page. We need to call getEntryCnt() and getRecCnt() from DirPage class.

getAvailPage() searches the first data page with enough space in the directory to store a record of the given size. While we are traversing through the directory pages we collect free space info for each entry in each directory page. We return the available page id once we found a suitable page whose free space is greater than the record length plus the SLOT_SIZE. We create a new data page by calling insertPage() if no such page found.

findDirEntry() is implemented for finding directory entries of data pages. We first need to locate the head directory page given headId. We need to traverse through the directory pages and pin the current direcotry page in the pool and loop through all the entries within each directory page to locate the page id.

updateDirEntry() updates the directory entry for the given data page. We create a new PageId() to locate directory page id and a new DirPage() to hold the directory page content. Calling findDirEntry() returns the directory entry of the given page id. We look up the record counts for the entry and update the counts adjusted by deltaRec. If the adjusted record counts is 0 we call deletePage() to remove it.

insertPage() inserts a new empty data page and its directory into the heap file. We first locate the head directory page of the heapfile and create a new DirPage() to hold the content of the page. We then traverse through directory page and check the entry count for each page. If the entry count is less than MAX_ENTRIES (125) we break the loop and create a new page. If the next directory page id is invalid we pin the new directory page in the pool and chain the newly inserted directory page with the current page and previous page, and then we start working on the newly created page by setting up the parameters (page id, reccnt, freecnt, entrycnt) in dirpage entry.

deletePage() deletes the given data page and its directory entry from the heap file by calling freePage(pageno) from BufferManager class. We call compact() from DirPage class to compact the dirpage to make sure no empty slots in between entries. If the dirpage is header dirpage or the entryCnt is greater than 1 we simply decrement the entryCnt by 1 and set the entryCnt. If the entryCnt is 0 we need to rechain the previous

dirpage with the next dirpage after deleting the target dirpage. There are two cases in the rechaining process: case 1 is that the previous page id is valid, we chain the previous dirpage with the next dirpage; case 2 is that the next dirpage is valid, we chain the next dirpage with the previous dirpage.

The teaching assistant Shraddha Bhanudas Zingade has given us lots of help on debugging. The common mistakes we have made are: pinning a page, but forget to unpin it at the end of the function; pinning a page outside of a loop, but put unpinning statement inside of the loop, which leads to unpin the same page for multiple times; sometimes we mistakenly use current page as next page, which leads to pointing a page that does not exist in the buffer pool; we changed our while loop to while(true) so that we can make sure the program will always execute the loop as we wanted; in deletePage() we made sure the previous page and delete page are valid and chain them correctly, and ect.

References:

DbImplem-Heap File Assignment-Winter-2018.pdf
Ramakrishnan, Gehrke - Database Management Systems
https://github.com/munafahad/
https://github.com/Elessawy/
https://github.com/AhmedGad/Database/blob/master/decompiled/heap
https://github.com/mouhamadIk/miniSGBD
https://github.com/leong1016/cs6530project