DBMS Models and implementation (Section 001)
Instructor: Sharma Chakravarthy
Project I: Implement buffer replacement policies and Buffer Hit Ratios

Made available on:	**8/29/2022**
Due on:	**9/22/2022 (11:59Pm) (milestone is not enforced)**
Submit by:	**Canvas (1 zipped folder containing all the files/sub-folders)**
Weight:	**15% of total**
Total Points:	**100**

This project implements page replacement policies and buffer hit ratios for those policies of the buffer manager layer and of the database management system MINIBASE. These hit ratios indicate the effectiveness of buffer replacement policies.

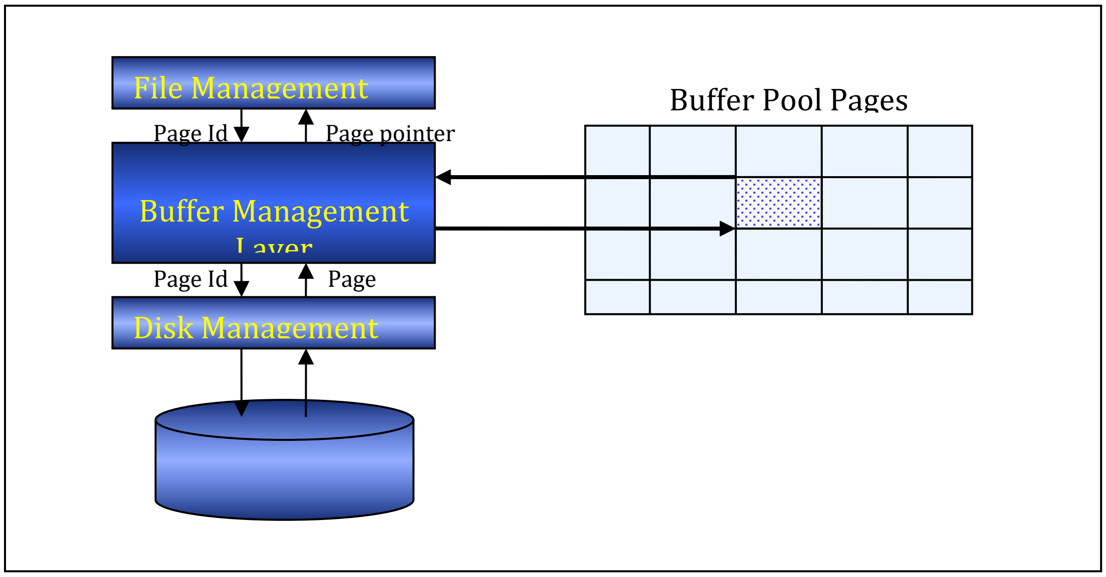
Hit ratio is defined as the ratio of the hits to the pages already in the buffer manager to the total number of page requests. You can compute it using unique as well as total hits and requests (see details later). Concurrency control and recovery are included in the actual MINIBASE implementation of the buffer manager layer but is not included as part of this project.

I. Problem Statement:

In this project you will implement an integral part of the bottom layers of a typical DBMS, without support for concurrency control and recovery. The underlying Disk Manager and other framework classes are provided along with the skeleton code. Before jumping into this project, make sure you thoroughly understand each of the buffer replacement policies LRU, MRU, clock, Random, and FIFO from the textbook.

The buffer manager provides a uniform interface for allocating and de-allocating pages on disk, bringing disk pages into the buffer pool and pinning and unpinning them in the buffer pool. You are given the buffer management code. You will be asked to implement 2 (two) of the replacement policies (each as a separate class that extends the `Replacer` class) as described in the textbook. Each team will be assigned 2 policies to implement in the parameters file.

The disk space manager and buffer manager has been **implemented and is provided.**



Overview of the DBMS layers showing the position of the buffer management layer

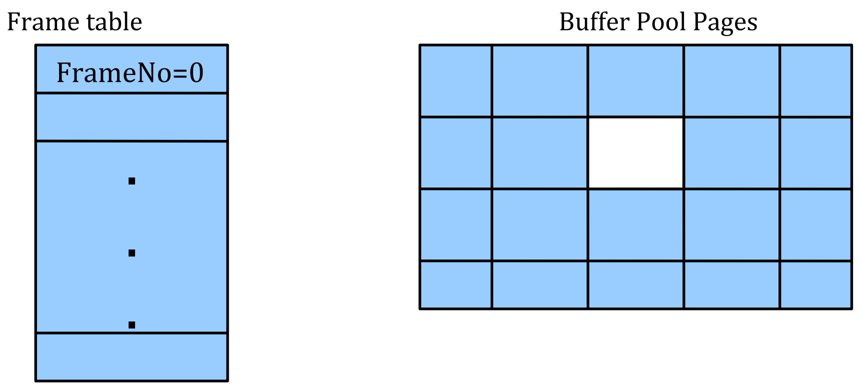
Use the following links (on the course web site) for additional information on the project and Minibase, respectively:

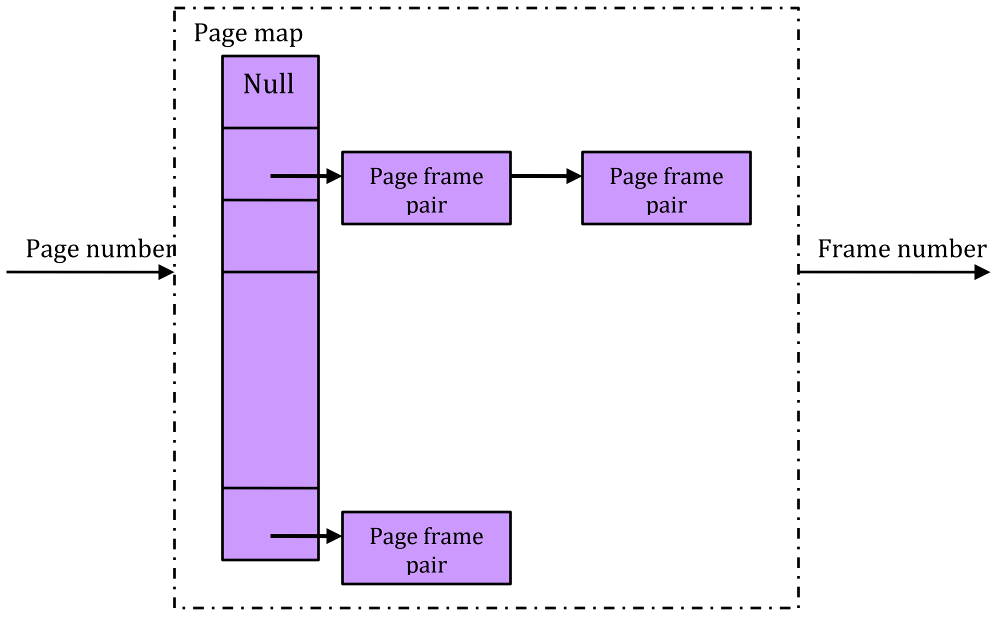
1) http://itlab.uta.edu/courses/cse5331And4331/javadocs/index.html
2) http://itlab.uta.edu/courses/cse5331And4331/mini_doc-2.0/minibase.html

II. **Internally, the buffer manager should consist of (at least) the following.**

i. Buffer Pool
   The actual array of Page objects (can be viewed as an array of byte arrays).
ii. Frame Table
   Array of frame descriptors, each containing the pin count and dirty status.
iii. Page Map
   Maps current page numbers to frame numbers; used for efficient lookups.
iv. Replacement Policy
   Note that policy class name has to be changed in the constructor of the BufMgr using the name of the class you have implemented!
v. Make sure you understand the following methods in the BufMgr class clearly as you will be inserting hit ratio computation code into some of them
   a. Public BufMgr(int numbufs)
      Initialize your buffer manager layer with buffer pool, frame table and page map and replacer policy
      Buffer Pool: The actual array of page objects
      Frame table: Array of frame descriptors, each containing the pin count and dirty status
      Page map: Maps current page numbers to frame numbers

b. newPage(Page firstpg, int run_size)
c. pinPage(PageId pageno, Page page, boolean skipRead)
d. unpinPage (PageId pageno, boolean dirty)
e. flushPage(pageId pageno)





III. Some useful hints and tips:

- For extensibility, avoid putting your entire solution into a single class.
  (i.e. you might want to separate the replacement policy code from BufMgr)
- 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. Implement the two assigned replacement policy class given to you separately (as Lru.java, Mru.java, Clock.java, Random.java, Fifo.java) that extends the Replacer class for implementing the replacement policies. Use the 2 buffer sizes given for comparative analysis. You need to change buffer size in GlobalConst.java, recompile and execute!
2. You will implement and execute each policy separately on the same test cases. You will have to use the correct replacement policy file for this. Keep them separate by creating separate directories for each policy to avoid confusion. Upload all directories.
3. Calculate the Buffer Hit Ratio (BHR) as follows. Buffer hit ratio can be computed by keeping track of hits (already in the buffer pool) to the number of times the page is loaded to the buffer pool. Higher the BHR, better is the replacement policy
4. Also, keep track of the loads and hits for each page and output the top 5/10 pages with respect to hits
5. BHR value is time dependent. Initially, the BHR value will be low. It will improve (hopefully and based on the policy and access pattern) as the number of pages accessed increases
6. Do an analysis of the hit ratios based on the test cases for the two buffer sizes given

7. For each replacement policy you implement, print out some additional information, as specified below, to get some insights into how these policies function for the same sequence of page accesses. Make sure you output other relevant information, such as policy being used, number of buffers being used, etc.
    a. For LRU, MRU, FIFO, and Random, print the number of times the requested page has been referenced when you pick that page for replacement (victim page)
    b. For Clock, print the number of pages that have used their second chance i.e., the number of page whose second chance bit was flipped before reaching the victim page.

8. Print BHR after k*b page accesses where b is the buffer size and k is from 1 to whatever number based on the test case. The hit ratios for each access as well as the policy specific print tasks must be written to an output file and included in the submission.

9. 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.

   NOTE: First 8 pages are created by the system. So, count the pages actually pinned from 9 onwards **(i.e., pageno.pid > 8)**

V. Getting Started

Copy the code-given.zip file and unzip it in your project directory. You need to set the SRCPATH and BINPATH to the directory in which you have unzipped the files. It is much easier to use makefile and the make command for compiling and executing your code and testing it.

Code given to you will compile and run on Omega without any problem (as long your environment is set correctly). As much as possible you should use omega for your project and demonstration.

Windows 10 provides a ubuntu terminal by installing WSL (Windows subsystem for Linux). You can install and try using this link. https://ubuntu.com/tutorials/install-ubuntu-on-wsl2-on-windows-10#1-overview But we will not be able to provide support for that.

Similarly, if you are using windows, you can install Cygwin to get into a linux-like command prompt. You may want to install Cygwin if you are going to use windows environment for this project. You also need the Java compiler and the runtime environment. Again, we will Not provide any support for this.

After you copy the code, the code should compile without any errors when you execute the command. Do not delete any files you copy!!

```
>make bufmgr
```

//should compile without any errors once you fix the variable name in BufMgr.java

You can execute the test code by typing

```
>make bmtest
```

//will start with creating a database … and give an Unrecoverable error as the Policy class is not implemented. When you implement the policy class, it should proceed and should give all tests completed successfully if your code is correct.

You may get an error after a couple of print statements as the Replacement policy is not complete. This is what you need to implement as part of this project.

If you want to use an IDE (e.g., Eclipse, Netbeans) for doing the project, you need to figure out ON YOUR OWN 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.{ 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 for determining partial credit.)
- **File Descriptions**
  all code used and developed for the project. 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 modified bufmgr package. We will ignore source code in any other directories.
- **You will turn in one zipped file containing you source code as well as the report. Your source code must include all implemented policy files and the buffer manager code given to you in different directories. Name the outer most directories using the policy names.**
- All of the above directories should be placed in a single zipped folder named as - ‘proj1_firstname_lastname_Section_final’. **Only one zipped folder per team should be uploaded using canvas.**
- 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!**
- Five days after the due date, the submission upload will be closed. Penalty for each day is 20% of the earned grade. No partial penalties per day!

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 1st Policy (code+output)	15
2.	Correctness of the 2nd Policy (code+output)	15
3.	Correct computation of BHR and top 5/10:	20 (10 + 10)
4.	Report on analysis of the policies wrt to hit ratios	10
5.	Report on analysis of buffer sizes on different policies	10
6.	Reporting and fixing 3 to 5 logical errors in the program	5
7.	Answering questions during demo	25
	Total	100

Your java program must be executable (<u>without any modification by us</u>) 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.

X. Demo

You will be asked to demo your code for grading. During demo, you should compile the version submitted on Canvas (timestamp will be checked), execute it to show the output produced. NO DEBUGGING will be entertained during the demo. Also, be prepared to answer questions on your implementation to ascertain you/team have done the project. For teams, BOTH are required to be present during the demo (whether virtually or in person)

XI. FYI, when your code runs correctly, the output from BMTest.java should look like (we may provide new test cases)

```
Running buffer manager tests...

 Test 1 does a simple test of normal buffer manager operations:
 - Allocate a bunch of new pages
 - Write something on each one
 - Read 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
 --> Failed as expected

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

 - Try to unpin 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.

All buffer manager tests completed successfully!
```