**CSC540 – Implementing RDBMS Lower-Level Layers – Project 1**

**Due date:10/04/09 11:59p.m.**

**Note:** This programming assignment (and subsequent programming assignments) should be carried out in groups of two. Choose a partner as soon as possible; people who do not yet have partners should post a message on the course project message board and monitor it to find available students

This project and those to come will be based on Minibase, a small relational DBMS, structured into several layers. An overview of Minibase can be found [**here.**](http://pages.cs.wisc.edu/~dbbook/openAccess/Minibase/intro/single_user.html) In this project, you are required to implement parts of the lower-level layers: BM - the buffer manager layer

*Acknowledgements: Assignment description was adapted from UC-Berkeley’s CS186 course (Introduction to Database Systems) in Spring 2007 by Mary Roth & Eben Haber*

**General Guidelines**

Reading and writing pages from main memory to disk is an important task of a database system.  Main memory is partitioned into collections of pages, and the collection of pages is called the *buffer pool.* The buffer pool is organized into *frames*, and each frame in the buffer pool can hold a page that is brought in from the disk. The *buffer manager* is responsible for bringing pages from disk to the buffer pool when they are needed, and writing pages back to the disk when they have been updated.  The buffer keeps a *pin* count and *dirty* flag for each frame in the buffer pool.  The pin count records the number of times a page has been requested but not released, and the dirty flag records whether the page has been updated or not.  As the buffer pool fills, some pages may need to be removed in order to make room for new pages. The buffer manager uses a *replacement policy* to choose pages to be flushed from the buffer pool.  The strategy used can greatly affect the performance of the system. *LRU* (least recently used), *MRU* (most recently used) and *Clock* are different policies that appropriate to use under different conditions.  
You can read more about buffer management in chapter 9 of the text book.

#### Background

The components of Minibase that you need to know about for this assignment are described below.  The javadoc for these classes can be found in [here](http://courses.ncsu.edu/csc540/lec/001/hw/proj/javadoc/packages.html).    
  
The [global](http://courses.ncsu.edu/csc540/lec/001/hw/proj/javadoc/Package-global.html) package contains definitions that will be used throughout the project.  The most important class to become familiar with for this assignment is Systemdefs.java.  Systemdefs maintains the state of the minibase database, including the database on disk (JavabaseDB) and the buffer pool (JavabaseBM).  The source is available [here](http://inst.eecs.berkeley.edu/%7Ecs186/sp07/homework/hw1/global-source.zip).  
  
The [diskmgr](http://courses.ncsu.edu/csc540/lec/001/hw/proj/javadoc/Package-diskmgr.html) package manages the database on disk. The disk manager is already implemented for you; you will not need to make any modifications. Its two most important classes are Page.java, which implements a disk page, and DB.java, which provides an API to create and delete a database of pages (on disk) and read and write pages to that database.  Your buffer manager will make use of this package, and in particular, the DB class. The source is available [here.](http://inst.eecs.berkeley.edu/%7Ecs186/sp07/homework/hw1/diskmgr-source.zip)  
  
The exceptions package provides the base definition of a minibase exception.  The [ChainException](http://courses.ncsu.edu/csc540/lec/001/hw/proj/javadoc/chainexception.ChainException.html) class maintains a stack of exceptions thrown from different components in minibase (such as disk manager and buffer manager).  Many exceptions have already been defined that you can use to indicate various error conditions that can occur.  You can also define additional subclasses of ChainException to handle specific error conditions if there is not already an existing exception.

#### Homework project description

Your job is to implement the buffer manager (in the [bufmgr](http://courses.ncsu.edu/csc540/lec/001/hw/proj/javadoc/Package-bufmgr.html) package) that will allow a higher-level minibase component to allocate and deallocate pages on a disk, as well as pin, read from and write to, and  unpin disk pages in the buffer pool. Your buffer manager should be able to create a buffer pool with frames for disk pages and support the *Clock* **and** *MRU* replacement policies.

**Getting Started.**

Download [BufMgr.zip](http://inst.eecs.berkeley.edu/%7Ecs186/sp07/homework/hw1/Homework1.zip) to get started. This is a complete eclipse project zipped into a single file, but you can also run everything from the command line.

The unzipped directory will contain a few things to help you get started:  
  
The *bufmgr* directory.  This directory contains class definitions that you will need to modify. It contains class definitions for the following classes:

* BufMgr.java: This is the class that implements the buffer manager.  An initial implementation is provided for you that keeps the buffer in main memory.   The class definition is provided, and the current implementation uses a hash table for pages and all of them reside in main memory -- each request for a new page simply adds to the hash table. You will need to fill in the details to pin, unpin, read and write pages to disk, and use a replacement policy to select pages to remove from the buffer pool when it is full.
* BufMgrFrameDesc.java:  The buffer manager maintains an array of these frames to describe each page in the buffer pool.
* BufMgrReplacer.java:  This is an abstract class that defines the strategy to replace pages in the buffer pool if it is full.  You will need to extend this with 2 subclasses to implement different replacement policies: Clock and MRU.
* Clock.java:  This class extends BufMgrReplacer.java to implement a Clock strategy.  The class definition is provided, but you will need to fill in the details.
* MRU.java:  This class extends BufMgrReplacer.java to implement an MRU strategy.  The class definition is provided, but you will need to fill in the details.

The *tests* directory.  This directory contains a driver program that you can use to test your buffer manager.  It contains a single class:

* BMDriver.java.  BMDriver extends the class TestDriver that is found in the global package. TestDriver is a generic test harness that you can extend for each of your homework assignments to test your code.   The current implementation of BMDriver simply creates a buffer pool, runs any methods with the prefix 'test', and then exits.
* After you have unzipped the file, you can test that everything is working by running BMDriver as a Java application.  You should see the following output:

|  |
| --- |
| Running Buffer Manager tests....    Invoking test: test1    Test 1 is not implemented.     Invoking test: test2    Test 2 is not implemented.    ...Buffer Manager tests completed successfully.   Cleaning up ... Done. Exiting... |