CSC540 – Programming Team Project 2
Systems Project - Buffer & Log Management
Due by Nov 30th, 2015, 11.59p.m.

Background

A buffer pool contains a set of buffers and each buffer can hold a page. Data is stored on disk as blocks and after being read into main memory they are represented as pages. 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. Each page can be held in a buffer and the collection of buffers is called the buffer pool. The buffer manager is responsible for bringing blocks from disk to the buffer pool when they are needed and writing blocks back to the disk when they have been updated. The buffer manager keeps a pin count and dirty flag for each buffer 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 are appropriate to use under different conditions.

SimpleDB Buffer Manager (from Dr.EdwardSciore's notes)

The SimpleDB buffer manager is grossly inefficient in two ways:

- When looking for a buffer to replace, it uses the first unpinned buffer it finds, instead of using some intelligent replacement policy.
- When checking to see if a block is already in a buffer, it does a sequential scan of the buffers, instead of keeping a data structure (such as a map) to more quickly locate the buffer.

Project Tasks Description

Below is a brief description of the tasks that you are required to implement and some additional guidelines that will help you with the implementation:

- **Task 1: Use a Map data structure to keep track of the buffer pool**
  Keep a Map of allocated buffers, keyed on the block they contain. (A buffer is allocated when its contents is not null, and may be pinned or unpinned. A buffer starts out unallocated; it becomes allocated when it is first assigned to a block, and stays allocated forever after.) Use this map to determine if a block is currently in a buffer. When a buffer is replaced, you must change the map -- The mapping for the old block must be removed, and the mapping for the new block must be added. For our convenience, we will be using "**bufferPoolMap**" as the name of the Map.

- **Task 2: Most Recently Modified (MRM) Buffer Replacement Policy**

This suggests a page replacement strategy that chooses the modified page with the lowest LSN. Implement this strategy.

- **Task 3**: Extend the class(es) associated with buffer management to include code for maintaining useful statistics about buffer usage. E.g. number of times buffers are read and written. Add a new method getStatistics() which returns the number of times each buffer in the buffer pool was read, and the number of times each buffer in the buffer pool was written into.

How to Start

- **You may retain the same teams from project 1.**
- Follow the instructions to setup and test your SimpleDB installation (previously posted).
- Go through the java files in the *simpledb.buffer* folder – *Buffer, BufferMgr, and BasicBufferMgr.*Understand the current replacement policy used and see how the buffer pool is maintained.
- Before starting implementation, it would be useful to trace the existing methods and identify the modifications that will be required to implement the tasks.
- Start thinking about how to test your code. Below we provide some guidance and some test scenarios that will be used for grading. **However, do not rely solely only on these.** Part of designing algorithms to meet requirements is thinking about corner cases and testing for them. So there are a few test cases that we will perform that are not captured here. However, a general description of the scope of testing to be done is.

General Testing guidelines

We will be running several test cases to evaluate the proper implementation of the three tasks. Below are some example scenarios that will be used to ensure that the buffers are allocated using MRM and that the Map is updated appropriately, using judicious pinning and unpinning:

• Basic Buffer manager functionality – pin / unpin / pinNew.
• Test for appropriate changes in availability of buffers after pinning.
• Are all buffers allocated before re-allocating any?
• Pin more blocks than the number of buffers available (Expected: BufferAbortException)
• Test if the contents of the Buffer pool Map is up to date after pinning / unpinning
• Basic MRM algorithm - Pin / unpin blocks in a random order to test MRM.

Below are methods that will be used for testing and some example scenarios that will be used to test your buffer management implementation.

Code snippets _

Below, we provide some methods that we will use to test your implementation of the Map for the Buffer pool. Copy them into the respective java files. For our convenience, we have used "**bufferPoolMap**" as the name of the Map.

BasicBufferMgr.java

```
/**
* Determines whether the map has a mapping from
* the block to some buffer.
* @paramblk the block to use as a key
* @return true if there is a mapping; false otherwise
*/
boolean containsMapping(Block blk) {
return bufferPoolMap.containsKey(blk);
}
/**
* Returns the buffer that the map maps the specified block to.
* @paramblk the block to use as a key
* @return the buffer mapped to if there is a mapping; null otherwise
*/
Buffer getMapping(Block blk) {
return bufferPoolMap.get(blk);
}
```

BufferMgr.java

```
/**
* Determines whether the map has a mapping from
* the block to some buffer.
* @paramblk the block to use as a key
* @return true if there is a mapping; false otherwise
*/
public boolean containsMapping(Block blk) {
return bufferMgr.containsMapping(blk);
}
```

```
/**
* Returns the buffer that the map maps the specified block to.
* @paramblk the block to use as a key
* @return the buffer mapped to if there is a mapping; null otherwise
*/
public Buffer getMapping(Block blk) {
return bufferMgr.getMapping(blk);
}
```

Note: These tasks require you to change files only in the package *simpledb.buffer*. Though there may be several ways to achieve the above mentioned tasks, however you are likely only required to make modifications to the *Buffer*, *BasicBufferMgr*,*BufferMgr*, *LogRecordIterator* and *RecoveryManager* classes and the *LogRecord*Interface .