

Project 1
Database Management Systems (DM556)



**UNIVERSITY OF
SOUTHERN DENMARK**

Group 2
Mark Jervelund (Mjerv15) Troels Petersen (trpet15)
IMADA

March 29, 2017

Overall Status

While there was some bugs in the original hand-in, all of those have been fixed in this re-assignment. Therefore this hand-in passes all the tests in `BMTest.java`.

Division of Labor

We worked on the project either sitting together at the university or at home remotely working together and splitting tasks when possible. We spent a lot of time trying to make `fix pinPage`, whereas we found the rest of the functions relative to `pinPage`, very simple to implement. The work was very evenly divided.

Specification

We were tasked with implementing the following functions for the `bufmgr.java`
`freePage`, `pinPage`, `unpinPage`, `flushPage`, `flushAllPages`, `getNumBuffers`, `getNumUnpinned` and `pick victim`.

`freePage` should deallocate a page from disk.

`Pinpage` should pin a page by incrementing the `pinCnt` by 1, or by loading it into the bufferpool if it isn't in the bufferpool already.

`Unpinpage` should unpin a page, flush it to disk if it's dirty and reduce the `pincount` by 1.

`Flushpage` should save a page to disk if dirty.

`Flushpages` should write all pages to disk if they're dirty.

`getNumBuffers` gets the amount of buffers.

`getNumUnpinned` gets the number of unpinned pages.

`Pickvictim` gets the index for the first unpinned page, and returns -1 if all pages in the pool are pinned.

Implementation

`FreePage` First checks if the page is pinned. If it's not, it then deallocates the page from disk.

```

103      * @throws IllegalArgumentException if the page is pinned
      */
105      public void freePage(PageId pageno) throws IllegalArgumentException {
          FrameDesc fdesc = pagemap.get(pageno.pid);
          if (debugvalue){
              System.out.println("freeing_page_with_id_"+pageno.pid);
          }
110      if (fdesc != null){

```

`pinpage`

```

129      * shouldn't be in the buffer pool. Throw an IllegalArgumentException
      ↪ if so.)

```

First we check if the page is already in the bufferpool if it is we increment the pin counter

```

133      *          if skipread == PIN_DISKIO, works as an output param,
      ↪ holding the contents of the pinned page read from the disk
      * @param skipRead PIN_MEMCPY(true) (copy the input page to the buffer
      ↪ pool); PIN_DISKIO(false) (read the page from disk)
135      * @throws IllegalArgumentException if PIN_MEMCPY and the page is
      ↪ pinned
      * @throws IllegalStateException if all pages are pinned (i.e. pool
      ↪ exceeded)
      */
      public void pinPage(PageId pageno, Page page, boolean skipRead) {

```

```

    if (debugvalue) System.out.println("pinpage_called_with_pageid_" +
    ↪ pageno.pid + "_Skipread_" + skipRead + "and_page_" + page.
    ↪ toString());
140
    // First check if the page is already pinned
    FrameDesc fdesc = pagemap.get(pageno.pid);
    if (fdesc != null) {
145
        // Validate the pin method

```

If it isnt we pick a victim, and if there isnt any victims we throw an IllegalStateException.

```

147
        "Page_pinned;_PIN_MEMCPY_not_allowed"
    );
    // Increment pin count, notify the replacer, and wrap the
    ↪ buffer.
150
        fdesc.pincnt++;
        replacer.pinPage(fdesc);
        page.setPage(bufpool[fdesc.index]);

```

If we have a non pinned frame we write this page to disk if its dirty.

```

154
        } // If in pool
155
        // Select an available frame
        int frameNo = replacer.pickVictim();
    // If no pages are unpinned, then throw an exception telling that.
        if (frameNo < 0){
160
            throw new IllegalStateException("All_pages_pinned."
            ↪ );

```

And if it isnt dirty we copy or read the new page into the bufferpool and update the pagemap.

```

164
        fdesc = Minibase.BufferManager.frametab[frameNo];
165
        // If the frame was in use and dirty, it should write it to
        ↪ the disk.
        if( fdesc.pageno.pid != INVALID_PAGEID) {
            flushPage(fdesc.pageno);
            pagemap.remove(fdesc.pageno.pid);
        }
170
        //read in the page if requested, and wrap the buffer
        if(skipRead == PIN_MEMCPY) {
            bufpool[frameNo].copyPage(page);
        } else {
175
            Minibase.DiskManager.read_page(pageno, bufpool[
            ↪ frameNo]);
        }
        page.setPage(bufpool[frameNo]);

        //update the frame descriptor

```

unpinpage

```

189
    /**

```

First we check if the page is pinned. if its not we throw an exception.

```

193
        * @param dirty UNPIN_DIRTY if the page was modified, UNPIN_CLEAN
        ↪ otherwise

```

```

195      * @throws IllegalArgumentException if the page is not present or not
      ↪ pinned
      */
      public void unpinPage(PageId pageno, boolean dirty) throws
      ↪ IllegalArgumentException {

```

If its in the buffpool we decrement the pagecounter by 1, and update the pagemap with the new information.

```

198      // Checks if page is dirty.
      // First check if the page is unpinned
200      FrameDesc fdesc = pagemap.get(pageno.pid);
      if (fdesc == null || fdesc.pincnt == 0) throw new
      ↪ IllegalArgumentException(
          "Page_not_pinned;"
      );
      // If dirty, it should write the the page to the disk and then tell
      ↪ that the page is not dirty anymore.
205      if (dirty == UNPIN_DIRTY) {

```

flushpage

Pickvictim is implemented to return the index for the first element with pincnt 0. and if all elements are in use, it returns -1 to indicate this.

```

37      @Override
      public int pickVictim() {
      // Finds the first element in the frametab array, where pin count
      ↪ is equal to zero and returns it.
40          for (int i = 0; i < Minibase.BufferManager.frametab.length;
              ↪ i++) {
              if (Minibase.BufferManager.frametab[i].pincnt == 0) {
                  return i;
              }
          }
      // If no pages has zero pins, then it returns -1.
45      return -1;

```

Testing

Testing this time around was very successful, assuming it passes in the tests where it returns "failed as expected". It reports that test1, test2 and test3 completed successfully.

Appendix

Pickvictim

```

37      @Override
      public int pickVictim() {
      // Finds the first element in the frametab array, where pin count
      ↪ is equal to zero and returns it.
40          for (int i = 0; i < Minibase.BufferManager.frametab.length;
              ↪ i++) {
              if (Minibase.BufferManager.frametab[i].pincnt == 0) {

```

```

        return i;
    }
}
// If no pages has zero pins, then it returns -1.
return -1;

```

bufmgr.java

```

1 package bufmgr;

import java.util.HashMap;

5 import global.GlobalConst;
import global.Minibase;
import global.Page;
import global.PageId;

10 /**
 * <h3>Minibase Buffer Manager</h3> The buffer manager reads disk pages
 *   ↪ into a
 * main memory page as needed. The collection of main memory pages (called
 * frames) used by the buffer manager for this purpose is called the buffer
 * pool. This is just an array of Page objects. The buffer manager is used
 *   ↪ by
15 * access methods, heap files, and relational operators to read, write,
 * allocate, and de-allocate pages.
 */
@SuppressWarnings("unused")
public class BufMgr implements GlobalConst {

20     /**
 * Actual pool of pages (can be viewed as an array of byte arrays).
 */
    protected Page[] bufpool;

25     private boolean debugvalue = false;

    /**
 * Array of descriptors, each containing the pin count, dirty status,
 *   ↪ etc.
30     */
    protected FrameDesc[] frametab;

    /**
 * Maps current page numbers to frames; used for efficient lookups.
35     */
    protected HashMap<Integer, FrameDesc> pagemap;

    /**
 * The replacement policy to use.
40     */
    protected Replacer replacer;

    /**
 * Constructs a buffer manager with the given settings.
45     *
 * @param numbufs: number of pages in the buffer pool
 */

```

```

50  public BufMgr(int numbufs) {
    // initialize the buffer pool and frame table
    bufpool = new Page[numbufs];
    frametab = new FrameDesc[numbufs];
    for (int i = 0; i < numbufs; i++) {
55      bufpool[i] = new Page();
      frametab[i] = new FrameDesc(i);
    }

    // initialize the specialized page map and replacer
    pagemap = new HashMap<Integer, FrameDesc>(numbufs);
60    replacer = new Clock(this);
  }

  /**
   * Allocates a set of new pages, and pins the first one in an
   *   ↪ appropriate
65   * frame in the buffer pool.
   *
   * @param firstpg holds the contents of the first page
   * @param run_size number of new pages to allocate
   * @return page id of the first new page
70   * @throws IllegalArgumentException if PIN_MEMCPY and the page is
   *   ↪ pinned
   * @throws IllegalStateException if all pages are pinned (i.e. pool
   *   ↪ exceeded)
   */
  public PageId newPage(Page firstpg, int run_size) {
    // allocate the run
75    PageId firstid = Minibase.DiskManager.allocate_page(run_size);

    // try to pin the first page
    if (debugvalue) {
      System.out.println("trying_to_pin_the_first_page");
80    }
    try {
      pinPage(firstid, firstpg, PIN_MEMCPY);
    } catch (RuntimeException exc) {
      System.out.println("failed_to_pin_the_first_page.");
85      // roll back because pin failed
      for (int i = 0; i < run_size; i++) {
        firstid.pid += 1;
        Minibase.DiskManager.deallocate_page(firstid);
      }
90      // re-throw the exception
      throw exc;
    }
    // notify the replacer and return the first new page id
    replacer.newPage(pagemap.get(firstid.pid));
95    return firstid;
  }

  /**
   * Deallocates a single page from disk, freeing it from the pool if
   *   ↪ needed.
100  * Call Minibase.DiskManager.deallocate_page(pageno) to deallocate the
   *   ↪ page before return.
   *

```

```

    * @param pageno identifies the page to remove
    * @throws IllegalArgumentException if the page is pinned
    */
105 public void freePage(PageId pageno) throws IllegalArgumentException {
    FrameDesc fdesc = pagemap.get(pageno.pid);
    if (debugvalue){
        System.out.println("freeing_page_with_id_"+pageno.pid);
    }
110    if (fdesc != null){
        if (fdesc.pincnt != 0) {
            throw new IllegalArgumentException("The_page_is_pinned.");
        }
        return;
115        //throw new IllegalArgumentException( "page does not excists");
    }

    Minibase.DiskManager.deallocate_page(pageno);
120 }

/**
 * Pins a disk page into the buffer pool. If the page is already pinned
 * → ,
 * this simply increments the pin count. Otherwise, this selects
 * → another
 * page in the pool to replace, flushing the replaced page to disk if
125 * it is dirty.
 * <p>
 * (If one needs to copy the page from the memory instead of reading
 * → from
 * the disk, one should set skipRead to PIN_MEMCPY. In this case, the
 * → page
 * shouldn't be in the buffer pool. Throw an IllegalArgumentException
 * → if so. )
130 *
 * @param pageno identifies the page to pin
 * @param page if skipread == PIN_MEMCPY, works as an input
 * → param, holding the contents to be read into the buffer pool
 * if skipread == PIN_DISKIO, works as an output param,
 * → holding the contents of the pinned page read from the disk
 * @param skipRead PIN_MEMCPY(true) (copy the input page to the buffer
 * → pool); PIN_DISKIO(false) (read the page from disk)
135 * @throws IllegalArgumentException if PIN_MEMCPY and the page is
 * → pinned
 * @throws IllegalStateException if all pages are pinned (i.e. pool
 * → exceeded)
 */
public void pinPage(PageId pageno, Page page, boolean skipRead) {
    if (debugvalue) System.out.println("pinpage_called_with_pageid_" +
    → pageno.pid + "_Skipread_" + skipRead + "and_page_" + page.
    → toString());
140
    // First check if the page is already pinned
    FrameDesc fdesc = pagemap.get(pageno.pid);
    if (fdesc != null) {
145
        // Validate the pin method
        if (skipRead == PIN_MEMCPY && fdesc.pincnt > 0)
            → throw new IllegalArgumentException(

```

```

        "Page_pinned;_PIN_MEMCPY_not_allowed"
    );
    // Increment pin count, notify the replacer, and wrap the
    ↪ buffer.
150         fdesc.pincnt++;
        replacer.pinPage(fdesc);
        page.setPage(bufpool[fdesc.index]);
        return;
    } // If in pool
155
        // Select an available frame
        int frameNo = replacer.pickVictim();
    // If no pages are unpinned, then throw an exception telling that.
        if (frameNo < 0){
160             throw new IllegalStateException("All_pages_pinned."
                ↪ );
        }

    // Pick the frame that is not pinned.
        fdesc = Minibase.BufferManager.frametab[frameNo];
165        // If the frame was in use and dirty, it should write it to
        ↪ the disk.
        if( fdesc.pageno.pid != INVALID_PAGEID) {
            flushPage(fdesc.pageno);
            pagemap.remove(fdesc.pageno.pid);
        }
170

        //read in the page if requested, and wrap the buffer
        if(skipRead == PIN_MEMCPY) {
            bufpool[frameNo].copyPage(page);
        } else {
175            Minibase.DiskManager.read_page(pageno, bufpool[
                ↪ frameNo]);
        }
        page.setPage(bufpool[frameNo]);

        //update the frame descriptor
180        fdesc.pageno.pid = pageno.pid;
        fdesc.pincnt = 1;
        fdesc.dirty = false;

    // Pin the page and put the updated page in the pagemap.
185    pagemap.put(pageno.pid, fdesc);
    replacer.pinPage(fdesc);
}

/**
190 * Unpins a disk page from the buffer pool, decreasing its pin count.
 *
 * @param pageno identifies the page to unpin
 * @param dirty UNPIN_DIRTY if the page was modified, UNPIN_CLEAN
    ↪ otherwise
 * @throws IllegalArgumentException if the page is not present or not
    ↪ pinned
195 */
public void unpinPage(PageId pageno, boolean dirty) throws
    ↪ IllegalArgumentException {
    if (debugvalue) System.out.println("unpin_page_called_with_pageid"

```



```

    ↪ + pageno.pid + "_Dirty_status_" + dirty);
    // Checks if page is dirty.
    // First check if the page is unpinned
200   FrameDesc fdesc = pagemap.get(pageno.pid);
    if (fdesc == null || fdesc.pincnt == 0) throw new
        ↪ IllegalArgumentException(
            "Page_not_pinned;"
        );
    // If dirty, it should write the the page to the disk and then tell
    ↪ that the page is not dirty anymore.
205   if (dirty == UNPIN_DIRTY) {
        fdesc.dirty = dirty;
    }
    // Decrement the pin count, since the page is pinned by one less.
    ↪ Also unpin the page and update the page in the
    // pagemap.
210   fdesc.pincnt--;
    replacer.unpinPage(fdesc);
    //unpin page.

    return;
215 }

/**
 * Immediately writes a page in the buffer pool to disk, if dirty.
220 */
public void flushPage(PageId pageno) {
    // Check if page is unpinned
    FrameDesc fdesc = pagemap.get(pageno.pid);
    if (fdesc.dirty == true) {
225     // Writes page to disk and sets the dirty-state to false, since
        ↪ it has not been modified when comparing it
        // to the same page on the disk.
        Minibase.DiskManager.write_page(fdesc.pageno, bufpool[fdesc.
            ↪ index]);
        fdesc.dirty = false;
        pagemap.put(pageno.pid, fdesc);
230     }
}

/**
 * Immediately writes all dirty pages in the buffer pool to disk.
235 */
public void flushAllPages() {
    for (int i = 0 ; i < Minibase.BufferManager.frametab.length; i
        ↪ ++ ){
        if (debugvalue) {
            System.out.println("flushing_page_" + Minibase.
                ↪ BufferManager.frametab[i].pageno.pid);
240         }
        if (Minibase.BufferManager.frametab[i].pageno.pid > 0) {
            flushPage(Minibase.BufferManager.frametab[i].pageno);
        }
    }
245 }

/**

```

```
250      * Gets the total number of buffer frames.
      */
      public int getNumBuffers() {
          return Minibase.BufferManager.bufpool.length;
      }

255      /**
      * Gets the total number of unpinned buffer frames.
      */
      public int getNumUnpinned() {
          // Using a loop, this checks the state of each frame. Each time
          //      ↪ an unpinned frame is found, "j" is incremented.
          // In the end "j" is returned, as that must be the total amount of
          //      ↪ unpinned buffer frames.
260          int j = 0;
          for (int i = 0 ; i < Minibase.BufferManager.frametab.length; i++ )
              ↪ {
                  if (0 == Minibase.BufferManager.frametab[i].pincnt) j++;
              }
          //      System.out.println("there is "+j+" unpinned frames out of "+
          //      ↪ Minibase.BufferManager.frametab.length);
265          return j;
      }

} // public class BufMgr implements GlobalConst
```