

Project 1
Database Management Systems (DM556)



**UNIVERSITY OF
SOUTHERN DENMARK**

Mark Jervelund (Mjerv15) Troels Petersen (trpet15)
IMADA

February 28, 2017

Overall Status

The overall status of of project is that we

Division of Labor

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
isnt in the bufferpool already.

Unpinpage should unpin a page, flush it to disk if its 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.

Design

Flushpages was implementing using Flushpage as they're almost doing the same and this reduces the
amount of duplicate code.

Implementation

Freepage First checks if the page is pinned. If its not, it then deallocates the page from disk.

```

1      public void pinPage(PageId pageno, Page page, boolean skipRead) {
2          if(debugvalue){
3              System.out.println("pinpage_called_with_pageid_"+pageno.pid+" _
4                  ↳ Skipread_"+skipRead+"and_page_"+ page.toString());
5          }
6          //first check if the page is already pinned
7              FrameDesc fdesc = pagemap.get(pageno.pid);
8          if (fdesc != null) {
9
10             //Validate the pin method
11             if (skipRead == PIN_MEMCPY && fdesc.pincnt > 0)
12                 ↳ throw new IllegalArgumentException(
13                     "Page_pinned;_PIN_MEMCPY_not_allowed"
14             );
15             //increment pin count, notify the replacer, and wrap the buffer
16                 ↳ .
17             fdesc.pincnt++;
18             replacer.pinPage(fdesc);
19             page.setPage(bufpool[fdesc.index]);
20             return;
21         } // if in pool
22
23         // select an available frame
24         int frameNo = replacer.pickVictim();
25         if (frameNo < 0){
26             throw new IllegalStateException("All_pages_pinned."
27                 ↳ );
28         }
29         // System.out.println(frameNo);

```

```

//      System.out.println("skipread = " + skipRead);
//fdesc.pageno.pid = frameNo;
//Minibase.BufferManager.frametab[frameNo] = fdesc;

30      fdesc = Minibase.BufferManager.frametab[frameNo];

      if( fdesc.pageno.pid != INVALID_PAGEID) {
          pagemap.remove(fdesc.pageno.pid);
          if(fdesc.dirty) {
35              Minibase.DiskManager.write_page(
                  ↪ fdesc.pageno, bufpool[frameNo]
                  ↪ );
          }
      }
      //read in the page if requested, and wrap the buffer
      if(skipRead == PIN_MEMCPY) {
40          bufpool[frameNo].copyPage(page);
      } else {
          Minibase.DiskManager.read_page(pageno, bufpool[
                  ↪ frameNo]);
      }
      page.setPage(bufpool[frameNo]);
45 //      if (debugvalue) {System.out.println("Pageno = " + pageno.pid);}
      //update the frame descriptor

          fdesc.pageno.pid = pageno.pid;
          fdesc.pincnt = 1;
50          fdesc.dirty = false;

          pagemap.put(pageno.pid, fdesc);
          replacer.pinPage(fdesc);

55      }

```

```

1      public void freePage(PageId pageno) throws IllegalArgumentException
          ↪ {
      FrameDesc fdesc = pagemap.get(pageno.pid);

          if(fdesc.pincnt > 0){
5          throw new IllegalArgumentException("The_page_is_pinned.");
      }

          Minibase.DiskManager.deallocate_page(pageno);
      }

10     /**
      * Pins a disk page into the buffer pool. If the page is already
          ↪ pinned,

```

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.

```

1      @Override
      public int pickVictim() {

```

```

// TODO Auto-generated method stub
for (int i = 0; i < Minibase.BufferManager.frametab.length;
    ↪ i++) {
5   if (Minibase.BufferManager.frametab[i].pincnt == 0) {
        return i;
    }
    }
10   return -1;
}
}

```

Testing

From the testing we've done the programs gets into a neverending loop pin/unpin loop at SystemCatalog = new Catalog(false) in Minibase.java line 79 (my file with some print statements for debugging)

Conclusion

Appendix

Pickvictim

```

1   @Override
   public int pickVictim() {
       // TODO Auto-generated method stub
       for (int i = 0; i < Minibase.BufferManager.frametab.length;
           ↪ i++) {
5       if (Minibase.BufferManager.frametab[i].pincnt == 0) {
           return i;
       }
       }
10      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;

private boolean debugvalue = false;

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

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

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

35 /**
    * Constructs a buffer manager with the given settings.
    *
    * @param numbufs: number of pages in the buffer pool
    */
40 public BufMgr(int numbufs) {
    // initialize the buffer pool and frame table
    bufpool = new Page[numbufs];
    frametab = new FrameDesc[numbufs];
45 for (int i = 0; i < numbufs; i++) {
    bufpool[i] = new Page();
    frametab[i] = new FrameDesc(i);
    }

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

55 /**
    * Allocates a set of new pages, and pins the first one in an
    ↪ appropriate
    * frame in the buffer pool.
    *
    * @param firstpg
    * holds the contents of the first page
60 * @param run_size
    * number of new pages to allocate
    * @return page id of the first new page
    * @throws IllegalArgumentException
    * if PIN_MEMCPY and the page is pinned
65 * @throws IllegalStateException
    * if all pages are pinned (i.e. pool exceeded)
    */
public PageId newPage(Page firstpg, int run_size) {
70 // allocate the run
    PageId firstid = Minibase.DiskManager.allocate_page(
    ↪ run_size);

    // try to pin the first page

```

```

75      System.out.println("trying_to_pin_the_first_page");
      try {pinPage(firstid, firstpg, PIN_MEMCPY);}
          catch (RuntimeException exc) {
              System.out.println("failed_to_pin_the_first_page.");
              // roll back because pin failed
              for (int i = 0; i < run_size; i++) {
80                  firstid.pid += 1;
                  Minibase.DiskManager.deallocate_page(firstid);
              }
              // re-throw the exception
              throw exc;
85          }
          // notify the replacer and return the first new page id
          replacer.newPage(pagemap.get(firstid.pid));
          return firstid;
      }
90
      /**
       * Deallocates a single page from disk, freeing it from the pool if
       *   ↪ needed.
       * Call Minibase.DiskManager.deallocate_page(pageno) to deallocate
       *   ↪ the page before return.
       *
95      * @param pageno
       *             identifies the page to remove
       * @throws IllegalArgumentException
       *             if the page is pinned
       */
100     public void freePage(PageId pageno) throws IllegalArgumentException
       ↪ {
        FrameDesc fdesc = pagemap.get(pageno.pid);

        if(fdesc.pincnt > 0){
105            throw new IllegalArgumentException("The_page_is_pinned.");
        }

        Minibase.DiskManager.deallocate_page(pageno);
    }

    /**
110     * 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
       * it is dirty.
       *
115     * (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. )
       *
120     * @param pageno
       *             identifies the page to pin
       * @param page
    
```

```

*          if skipread == PIN_MEMCPY, works as 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
125 * @param skipRead
*          PIN_MEMCPY(true) (copy the input page to the buffer
*          ↪ pool); PIN_DISKIO(false) (read the page from disk)
* @throws IllegalArgumentException
*          if PIN_MEMCPY and the page is pinned
* @throws IllegalStateException
130 *          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());
135 }
//first check if the page is already pinned
    FrameDesc fdesc = pagemap.get(pageno.pid);
if (fdesc != null) {
140         //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
        ↪ .
145         fdesc.pincnt++;
        replacer.pinPage(fdesc);
        page.setPage(bufpool[fdesc.index]);
        return;
    } // if in pool
150
    // select an available frame
    int frameNo = replacer.pickVictim();
    if (frameNo < 0){
        throw new IllegalStateException("All_pages_pinned."
            ↪ );
155 }
//    System.out.println(frameNo);
//    System.out.println("skipread = " +skipRead);
//fdesc.pageno.pid = frameNo;
//Minibase.BufferManager.frametab[frameNo] = fdesc;
160
        fdesc = Minibase.BufferManager.frametab[frameNo];

        if( fdesc.pageno.pid != INVALID_PAGEID) {
            pagemap.remove(fdesc.pageno.pid);
165             if(fdesc.dirty) {
                Minibase.DiskManager.write_page(
                    ↪ fdesc.pageno, bufpool[frameNo]
                    ↪ );
            }
        }
        //read in the page if requested, and wrap the buffer
170         if(skipRead == PIN_MEMCPY) {
            bufpool[frameNo].copyPage(page);

```

```

        } else {
            Minibase.DiskManager.read_page(pageno, bufpool[
                ↪ frameNo]);
        }
175     page.setPage(bufpool[frameNo]);
//     if (debugvalue) {System.out.println("Pageno = " + pageno.pid);}
        //update the frame descriptor

        fdesc.pageno.pid = pageno.pid;
180     fdesc.pincnt = 1;
        fdesc.dirty = false;

        pagemap.put(pageno.pid, fdesc);
        replacer.pinPage(fdesc);
185
    }

    /**
190     * Unpins a disk page from the buffer pool, decreasing its pin
        ↪ count.
    *
    * @param pageno
    *         identifies the page to unpin
    * @param dirty
195     *         UNPIN_DIRTY if the page was modified, UNPIN_CLEAN
        ↪ otherwise
    * @throws IllegalArgumentException
    *         if the page is not present or not pinned
    */
    public void unpinPage(PageId pageno, boolean dirty) throws
        ↪ IllegalArgumentException {
200     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
205     FrameDesc fdesc = pagemap.get(pageno.pid);

    if (fdesc == null) throw new IllegalArgumentException(
        "Page_not_pinned;"
    );
210     if (dirty){
        flushPage(pageno);
        fdesc.dirty = false;
    }
    fdesc.pincnt--;
215     pagemap.put(pageno.pid, fdesc);
    replacer.pinPage(fdesc);
    //unpin page.

    return;
220
}

/**
    * Immediately writes a page in the buffer pool to disk, if dirty.

```



```

225      */
      public void flushPage(PageId pageno) {
          FrameDesc fdesc = pagemap.get(pageno.pid);
          if (fdesc == null) {return;}
230      if (debugvalue) {
          System.out.println("fdesc_=" + fdesc.index);
      }

      if( fdesc.pageno.pid != INVALID_PAGEID) {
          pagemap.remove(fdesc.pageno.pid);
235          if(fdesc.dirty) {
              Minibase.DiskManager.write_page(fdesc.pageno, bufpool[fdesc
                  ↪ .index]);
          }
      }
      }
240

      /**
       * Immediately writes all dirty pages in the buffer pool to disk.
       */
      public void flushAllPages() {
245          for (int i = 0 ; i < Minibase.BufferManager.frametab.length; i
                  ↪ ++ ){
              flushPage(Minibase.BufferManager.frametab[i].pageno);
          }
      }

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.
       */
260      public int getNumUnpinned() {
          int j = 0;
          for (int i = 0 ; i < Minibase.BufferManager.frametab.length; i++ ){
              if (0 != Minibase.BufferManager.frametab[i].state){ j++;}
          }
265      return j;
      }

} // public class BufMgr implements GlobalConst

```