Comp 115 - Spring 2017 - Class Project Part A

Due April 8th (for Part A)

Class Project Introduction and Goal

For your class project, you will implement a few components of a simple database management system (DBMS), starting by the buffer management component. This experience will give you a deeper understanding of database system internals.

Timeline

You will complete the class project in two stages. This will help to keep the workload manageable, and it will allow you to check in periodically with the instructor and TAs to make sure that your implementation is on track. At the end of the semester, you will have several components that interact with one another and form the basis of a working DBMS.

Part A of the class project is due on: April 8th

Your complete class project submission is due on: May 10th (K Block Schedule)

Late Policy

Students may submit the assignment one day late with a 5% penalty, two days late with a 10% penalty, and up to four days late with a 25% penalty.

After communication with the instructor, a submission may still be accepted beyond the four-day period, with a penalty of 25%.

How To Submit

Submit your assignment with this Provide command:

```
provide comp115 115_projectA <code archive>.tar <report document>.pdf
```

Part A: The Buffer Manager

Part A of the class project involves implementing the buffer manager for your DBMS. The buffer manager interacts with the file system to create and delete files, open and close files, and allocate new blocks when needed. The buffer manager is also responsible for maintaining the most useful parts of the files in memory. It reads in pages from storage, maintains them in the buffer pool, and writes them back to storage when the buffer is full.

Looking ahead, Part B will involve adding a heap file layer to your DBMS. Part C is optional and can be completed for extra credit; it will involve adding an index file layer to the DBMS. More details on these parts will be released before the Part A due date.

Key Concepts and Necessary Background

Most of the information that you need to complete the class project is found chapters 8 and 9 of the course textbook, *Database Management Systems* (Ramakrishnan & Gehrke). For Part A, pay particular attention to sections 9.3 and 9.4.

We also covered this information in class. Slides that are relevant to Part A can be found in the File Organization and Storage Layer lecture slides.

The project also requires basic familiarity with the C programming language. In particular, you should be comfortable with file I/O and memory allocation. There are many free online resources that you can use to brush up on these topics; here are just a few:

W3 Schools C Tutorial (look at the sections on pointers and file I/O)
http://www.w3schools.in/c-tutorial

Fresh 2 Refresh C Tutorial (look at the sections on dynamic memory allocation and file handling)
http://fresh2refresh.com/c-programming/c-file-handling/

The Buffer Manager API

To help you get started, we will provide you with a high-level API for the buffer manager. The API includes a signature for each buffer manager function, along with a description of what the function should do. Your job is to fill in the bodies of these functions, and then write a test suite to demonstrate that these functions work as expected.

While we are providing you with an initial framework, you will have to make several choices about the design of the buffer manager. Here are some aspects of the implementation to consider:

- **Page Organization**: The buffer manager moves one unit of a file at a time between an external storage device and main memory. Each unit is called a page and has a fixed size. The buffer manager itself doesn't deal with the internal organization of pages, but higher-level components of the DBMS expect pages to have a particular structure, so you will need to choose an organization strategy for your pages (which will be necessary for the next part of the project). It's useful to think of a page as a collection of slots, each of which is either empty or contains a record. There are several different ways to organize slots on a page; section 9.6 of the textbook describes some of them, and we will discuss a standard approach called *N-ary storage model* (NSM) in class. NSM is a good choice because you will be familiar with it from class. For extra credit, you can implement a more advanced strategy called *Partition Attributes Across* (PAX) that we will also discuss in class.

- **File Organization**: A file is a collection of pages. There are several different approaches to organizing records within a file; you are free to choose the approach that you like best. The

simplest option is to implement heap files--unordered collections of records. The lecture slides on File Organization describe two ways of implementing heap files: one involves using a list, and the other involves using a page directory. Any of the two strategies can be used.

- **File Metadata**: Several of the API functions handle interactions with the file system, such as creating and opening database files. The buffer manager should store some metadata about each database file, such as the file's location and the number of blocks that it contains. One way to do this is to use a hash table in which each key is a file descriptor and each value is the metadata for a given file. However, you are free to use any data structure that gets the job done.

- **Buffer Pool Structure**: You might find it helpful to think of the buffer pool as being like a page at a higher level of organization: just like a page is a collection of slots that contain records, a buffer pool is a collection of slots that contain pages. Your buffer pool should be a data structure (e.g., a list) with a fixed number of slots. Each slot should be the same size as your system's page size, and it should be initially empty. When the buffer manager reads a page from external storage into memory, it should store the page in an empty slot (frame) in the buffer pool.

- **Buffer Pool Replacement Policy**: When the buffer manager reads a page from external storage into memory and the buffer pool is full, it has to decide which page in the buffer pool to replace with the new page. The method that the buffer manager uses to make this decision is called its replacement policy. We discuss three common replacement policies called *least recently used* (LRU), *most recently used* (MRU) and *clock* in class; section 9.4.1 of the textbook describes these policies and a few others. You will need to decide which replacement policy your buffer manager will use, and implement it.

Here is the buffer manager API that you will implement:

```
#define FRAMESIZE 4096

typedef fileDesc int; /* The key for accessing a file's metadata, like its physical location, # blocks, ...*/
typedef errCode int;
typedef struct _block{
        int isPinned;
        void* data; // void data[FRAMESIZE];
        fileDesc fd;
        int blockID; // possible other ways to do the same thing
        /*….more metadata might be needed */
} block;
```

```
void BM_init();
```

- Perform necessary initializations for the buffer layer. This probably involves creating a buffer pool with a specified size and frame size (you can make these parameters to the function or define them as constants in your code), and initializing bookkeeping variables for each frame in the pool.

```
errCode BM_create_file( const *char filename );
```

- Create a new DB file on disk. You can use a C library function to help you with this. Return 0 if the operation succeeds or an error code if it fails.

fileDesc BM_open_file(const *char filename);
- Open an existing file and return a file descriptor for it. (If you need a refresher on what a file descriptor is, see the tutorials listed above). A C library function will help you open the file.

errCode BM_close_file(fileDesc fd);
- Close an opened file and return 0 if the operation succeeds or an error code if it fails. Again, you can use a C library function for this.

errCode BM_get_first_block(fileDesc fd, block* blockPtr);
- Make blockPtr point to the buffer pool location of the file's first block. If the block does not exist in the buffer pool yet, read it in from the file. This might involve replacing a block in the buffer pool if all frames are full. Return 0 if the operation succeeds or an error code if it fails.

errCode BM_get_next_block(fileDesc fd, block* blockPtr);
- Make blockPtr point to the buffer pool location of the file's next block. Again, this might involve replacing a block in the buffer pool if all frames are full. Return 0 if the operation succeeds or an error code if it fails.

errCode BM_get_this_block(fileDesc, int blockID, block* blockPtr);
- Make blockPtr point to the buffer pool location of the block with ID blockID. Again, this might involve replacing a block in the buffer pool if all frames are full. Return 0 if the operation succeeds or an error code if it fails.

errCode BM_alloc_block(fileDesc fd);
- Add a new block to an open file and save the new block's ID. This means both allocating an extra on-disk block for the file and updating the metadata for that file. Return 0 if the operation succeeds or an error code if it fails.

errCode BM_dispose_block(fileDesc, int blockID);
- Remove the block with ID blockID from an open file. This means both freeing disk space and updating the file's metadata. Return 0 if the operation succeeds or an error code if it fails.

errCode BM_unpin_block(block* blockPtr);
- Mark the block that blockPtr points to as not in use, so that it can be removed when the buffer pool gets full and a block has to be replaced. When a block is unpinned, the buffer manager should write it to disk immediately so that it doesn't have to keep track of which blocks are dirty (this makes bookkeeping simpler). Return 0 if the operation succeeds or an error code if it fails.

void BM_print_error(errCode ec);
- Print the error message that corresponds to the error code.

We have created a header file that contains these signatures; you can use it as the starting point for your implementation. The header file will be available on the project website.