In this project, you will have a chance to study and modify the PostgreSQL source code, with a focus on one of the core modules – the *buffer manager*. Specifically, you are required to implement the Least Recently Used (LRU) buffer replacement policy by understanding and modifying the current code provided in PostgreSQL version 8.2.19, which comes with an implementation of the *clock* buffer replacement policy. You may find that the amount of code to write in this project is minimal, but you have to understand the codebase first before you know where and how to write your code.

2. Environment

You have to finish this project based on the PostgreSQL engine you built up in project 1. Obviously, you need to recompile and restart PostgreSQL every time you modify the code. You need to write your code in C.

The source code that is related to this project is located in the following files:

~/src/backend/storage/buffer/buf_init.c

~/src/backend/storage/buffer/bufmgr.c

~/src/backend/storage/buffer/freelist.c

and

~/src/include/storage/buf_internals.h

where ~ is the root directory of your PostgreSQL installation. Depending on how you design your data structure and algorithm, you may need to change the code in some or all of the files mentioned above. Rarely you will have to touch other files, but feel free to do so if you think it is necessary.

The file freelist.c maintains a list of buffers that the system can use for replacement. In the current clock algorithm in PostgreSQL 8.2.19, this list is used in a more complicated way than it will be if we are to implement LRU. For the purpose of doing this project, you can simply maintain a FIFO queue (linked list) of those buffers with a refcount of 0. When a new buffer becomes available, it is added to the tail of the list (e.g., in function StrategyFreeBuffer) and we always retrieve the one on the head of the list to be replaced (e.g., in function StrategyGetBuffer).

Pay attention to the BufferDescriptors data structure. This is the run-time data structure to store information of all buffer pages I mentioned in class. You have to understand how it is initialized in buf_init.c and used to maintain the buffer-related information. Depending on how you design the queue, you may or may not need to modify its definition (in buf_internals.h) and initialization.

To test the implementation of the buffer manager, as a requirement of this project, please add the following lines of code in your implementation:

```
elog(LOG, "Add buf %d\n", buf->buf_id);
```

whenever you add a buffer (pointed to by the pointer variable buf) into the list, and

```
elog(LOG, "Get buf %d\n", buf->buf_id);
```

whenever you obtain a buffer page from the list. For example, you can add them to the functions StrategyFreeBuffer and StrategyGetBuffer, respectively. By adding these two lines of code, the database server will output some messages every time it manipulates the free buffer queue. Recall that in project 1, you probably have done the following to start the server:

~/bin/postgres –D ~/mydir/data > logfile 2>&1

By this, the output message from the elog function calls will be written into a file named ‘logfile’. In addition to that, you can also control the total number of buffers in starting the server. For example,

~/bin/postgres –B 100 –D ~/mydir/data > logfile 2>&1

will run PostgreSQL with only 100 buffer pages.

You need to modify relevant functions (e.g., UnpinBuffer) in bugmgr.c to put a page back to freelist when the page is unpinned and reaches a refcount of 0.

Since understanding code is the main component of this project, I tend not to provide more detailed instructions on the PostgreSQL buffer manager code. Feel free to send me emails if you have questions about code organization in the buffer manager.

3. Testing

The ideal approach to test your code would be to run the PostgreSQL server and send a bunch of queries to it. You will need to pay attention to two things: 1) the server does not crash; 2) the patterns/orders of the buffer pages’ being released (added to the linked list) and reclaimed (dequeued from the list) are correct. For your convenience in testing and debugging, a sample log file from running some queries resulted from my implementation of LRU will be posted.

4. Submission and Grading

Please submit all source files you modified and a README file in plain text format with a description of your algorithm/data structure design and anything you want the grader to know about your project. In the submitted source file, please clearly mark those chunks of code you added/modified using comments. Attention: Please do NOT submit the whole postgres directory, only the modified source files and the README are needed!

You should compress all relevant files into a package named proj3-xxx.tar or proj3-xxx.zip where xxx is your NetID and submit via the assignment link in Canvas. This project is due 11:30pm, October 29, 2016. Your grade will be based on the correctness (shown in the log file trace) of your implementation and the description you write in the README file.

5. Other Issues/Tips:

- You have to write your own code! Copying from any sources (e.g., other students, the Internet, other PostgreSQL implementations …) will be regarded as cheating.
- Make sure you understand what you are going to implement: read the relevant chapters in the textbook about the difference between LRU, MRU, and the clock sweep algorithms.
- Understanding the current code is the key. Do not jump into writing code at the very beginning. You can print out the code in a compact format by

enscript –P printer_name -2Gr –DDuplex:true file_names

- Try to use the current data structures and function interfaces in PostgreSQL 8.2.19 as much as you can. These interfaces are defined in h. Even if you see some of the interfaces have parameters that do not make sense in LRU, leaving them untouched will save your time in implementing LRU.
- Stop the postgres server before you restart it. To shutdown postgres, you can type

~/bin/pg_ctl –D ~/mydir/data stop

- Do not wait till the last minute, you only have 2 weeks and reading source code may not be easy for most of you. You may start slowly by reading the code and adding the elog function calls to the right place, but start working asap!