CS4432 D-Term 2014

Project 1

Implementing a Better Buffer Manager for SimpleDB Database System

Release Date: March 25, 2014
Due Date: April 10, 2014 (11:59PM)
Total Points: 120 Points

About SimpleDB System

This semester we will work with the SimpleDB database system - a multi-user transactional database server written in Java, which interacts with Java client programs via JDBC. SimpleDB is a stripped-down open-source system developed by E. Sciore for pedagogical use. We have chosen this as foundation for the CS4432 projects to give you a chance to gain experience with database systems internals – while attempting to avoid the inherent complexity (aka steep learning curve) that would come with any industrial-strength system. Instead, the code for SimpleDB is clean and compact. Thus, the learning curve is small. Everything about SimpleDB is intentionally bare bone. It implements only a small fraction of SQL and JDBC, and does little or no error checking.

Task 1: Warm-up and Setup [10 Points]

As first step towards this project, it is essential that you get SimpleDB installed and become comfortable with its use. Hence, task 1 is to download SimpleDB and to play with it. The instructions for downloading the source and installing it on your machine can be found on the course website. You should use Java 1.6 or latter, otherwise you may get error messages.
After you have installed the SimpleDB server on your machine, then you are to create a small application schema (2 tables is sufficient) and design a few SQL queries. For this you need to utilize the JDBC interface provided with SimpleDB. So write a JDBC program to create and populate the two tables, and to search for some objects in this table and print their values based on some condition.
Please note that SimpleDB only supports a small subset of SQL! So it if does not work for a given command, it is probably not supported. Trace through the code and run some tests to get your bearings.

We strongly recommend using a Java IDE such as Eclipse (www.eclipse.org) to manage your code. IDE' such as eclipse allow you to compile, debug and execute within the same environment thus speeding up your programming time.

Task 2: Buffer Management.

The Current Buffer Management: Next, study the SimpleDB code in depth. In particular, familiarize yourself with the SimpleDB file and buffer manager. You can think of the current implementation of the buffer pool as fixed size array. When a (disk) block is requested by a transaction, SimpleDB reads it from disk and stores it in a buffer. We then say that the buffer is pinned by the transaction. Once the transaction is done, this buffer will be unpinned.

You will quickly observe that the SimpleDB buffer manager is very simplistic and thus inefficient. Accesses of the buffer pool are inefficient due to the reasons indicated below.

Every time the DBMS needs to find a free buffer to store a disk block or to check whether a block is in the buffer or not, it sequentially scans the buffer pool. In the worst case, this could be a complete scan through the entire buffer pool. Clearly, it would be more effective if instead more intelligent policies were used.

Project 1 Objective: The key objective of project 1 is now to improve the buffer manager of SimpleDB so to fix the above listed problems and thus make it more efficient. You are free to inherit and then override any method that you would like, as long as you maintain the method interfaces to be backwards compatible (so not to break simpleDB) . Think about what makes sense to do, and clearly document in your project report your design solution. As suggestion, it would make sense to work in the package simpledb.buffer and to start by modifying the following classes: *Buffer*, *BasicBufferMgr*, and *BufferMgr*.

Task 2.1) Efficient finding of Empty Frame [20 Points]

Your first task is to design an efficient technique that enables the Buffer Manager to find an empty frame fast (in constant time). You may use the methods suggested in class, or you may come up with other methods. Just make sure that this operation can be done in constant time.

Task 2.2) Efficient Search for a Given Disk Block [20 Points]

Your second task is to design an efficient technique that enables the Buffer Manager to figure out whether or not a given disk page, say D, exist in the buffer pool. If exist, then you should be able to return the frame# that contain D efficiently. You may use the methods suggested in class, or you may come up with other methods.

Task 2.3) Efficient Replacement Policy [20 Points]

You're their task is to implement buffer replacement policies to be used when the buffer is full and we need to choose a frame to evict from the pool. You are required to implement two replacement policies, which are LRU & Clock policies covered in class. These policies may require you to store and maintain more metadata information for each frame in the buffer pool. In your design, try to make it easy to switch between which policy is used, e.g., you may use a configuration parameter that specifies which policy is active (used).

Task 2.4) Other Basic Functionalities [10 Points]

The Buffer Manager has to also maintain metadata information indicating whether or not a given frame is ***pinned*** (implement that as a pin counter not a Boolean flag). Another information is whether or not a given allocated frame is ***dirty***. An evicted memory page from the buffer pool should be written back to disk if (and only if) it is dirty (has been modified).

Task 2.5) Reporting Functions [10 Points]

Implement or extend the *toString* methods for *Buffer*, *BufferMgr*, and *BasicBufferMgr*. The purpose of these methods is to display the contents of the buffer for testing purposes. The method for *Buffer* should show the buffer's id, the block it is allocated to, and whether the buffer is pinned. The method for the other two classes should show the information for each buffer. You will likely need this to do your testing, plus you can then use these printouts subsequently to produce output required as deliverable for this project 1 to show us your system is functioning correctly

Task 2.6) Testing Program [20 Points]

You must write a test program along with a variety of testing scenarios. Explain these scenarios in your report. You need to show us that you did good testing and your system is working in all scenarios and has not broken SimpleDB. For instance, it would be excellent if you could continue to again run the example SQL queries you had set up for part 1 above, and assure that they still function properly. Your test program should demonstrate (and print out) that the buffers are really allocated using the chosen policy, e.g., LRU or Clock, by doing judicious pinning and unpinning. Note that the call to *SimpleDB.init* causes the buffers to get used, so your test program won't be able to start from a clean slate.

Task 2.7) Code Documentation [10 Points]

Write comments on each and every function that you add or modify in SimpleDB. Your comments should always start with string “CS4432-Project1:” and then a description of what you are adding or changing. The TAs may look for these comments to understand what you did and to make sure you documented your code properly.

Deliverables

1. A *README.txt* file including precise installation instructions for the CS4432 staff to be able to run your modified SimpleDB. We will blindly follow your step-by-step instructions! In addition, we may also set up a one-to-one meeting with your team in which you will discuss your findings related to project1.

   Important: This file should start with the names of the team members

2. A *Examples.sql* file descripting your small SQL application schema and chosen queries, as well output produced (Task1). This should explain what subset of queries you are using to ensure that the system is working before and after your modifications.
3. A *Design.txt* file containing the design description in English of your effort in designing and developing a clean extension of the SimpleDB buffer manager, including the design of your two replacement policies, all associated data structures and metadata information, and the key places in SimpleDB that require changes to realize this buffer extension.
4. A *ExtendedSimpleDB.zip* file containing a fully working and self-contained code, carefully supported with inline comments as described in Task 2.7. You will be graded not only on the correctness of your code, but also its cleanness and the quality of your software documentation.
5. A *Testing.txt* file containing your testing harness for testing correctness, along with a description of the strategy you have chosen to test your enhanced system (Task 2.6). Your tests should not only test the new functionalities, but also make sure the original SimpleDB is not broken. The file should describe all testing scenarios that you have designed, a rationale for each test case, what it tests and how, and how you determined that your code indeed is correct.

   Important: This file should include printout the shows the output from each test and you should describe how this output shows correct execution. For example, the output should display the status of the buffer pages before and after some actions and indicate to us why this matches the expected behavior.

6. A *Bugs.txt* file that contains a list of bugs and components not working in your system. Make sure you report all aspects of your system that are not properly working. If your code does not work or at least has serious omissions and you have not properly documented this shortcoming, you will be graded more harshly.

What to Submit

A single file .zip from each team containing the 6 files mentioned above.

Where to Submit

Only through the Blackboard System.