# 6.830 Lab 5: Rollback and Recovery

Assigned: Nov 29 2010
Due: Dec 9 2010

## 0. Introduction

In this lab you will implement log-based rollback for aborts and log-based crash recovery. We supply you with the code that defines the log format and appends records to a log file at appropriate times during transactions. You will implement rollback and recovery using the contents of the log file.

The logging code we provide generates records intended for physical whole-page undo and redo. When a page is first read in, our code remembers the original content of the page as a before-image. When a transaction updates a page, the corresponding log record contains that remembered before-image as well as the content of the page after modification as an after-image. You'll use the before-image to roll back during aborts and to undo loser transactions during recovery, and the after-image to redo winners during recovery.

We are able to get away with doing whole-page physical UNDO (while ARIES must do logical UNDO) because we are doing page level locking and because we have no indices which may have a different structure at UNDO time than when the log was initially written. The reason page-level locking simplifies things is that if a transaction modified a page, it must have had an exclusive lock on it, which means no other transaction was concurrently modifying it, so we can UNDO changes to it by just overwriting the whole page.

Your BufferPool already implements abort by deleting dirty pages, and pretends to implement atomic commit by forcing dirty pages to disk only at commit time. Logging allows more flexible buffer management (STEAL and NO-FORCE), and our test code calls BufferPool.flushAllPages() at certain points in order to exercise that flexibility.

## 1. Find bugs, be patient, earn candy bars

You may find bugs, inconsistencies, and bad, outdated, or incorrect documentation, etc. We ask you to do this lab with an adventurous mind-set. If you run into problems, try to figure them out, or send us a friendly email. We promise to help. If you find a bug in our code, we'll give you a candy bar.

## 2. Getting started

You should begin with the code you submitted for Lab 4 (if you did not submit code for Lab 4, or your solution didn't work properly, contact us to discuss options.)

You'll need to modify some of your existing source with a patch, and install a few new files. Here's what to do:

- Make a backup of your Lab 4 solution by typing :

```
$ tar -cvzf 6.830-lab4-submitted.tar.gz 6.830-lab4
```

- Change to the directory that contains your top-level simpledb code:

```
$ cd 6.830-lab4
```

- Download the new tests and skeleton code for Lab 5 from http://db.csail.mit.edu/6.830/6.830-lab5-supplement.tar.gz:

```
$ wget http://db.csail.mit.edu/6.830/6.830-lab5-supplement.tar.gz
```

- Extract the new files for Lab 5 by typing:

```
$ tar -xvzf 6.830-lab5-supplement.tar.gz
test/simpledb/systemtest/LogTest.java
src/simpledb/LogFile.java
```

- You will also need to apply some patches, which you can fetch from http://db.csail.mit.edu/6.830/lab5.patch:

```
$ wget http://db.csail.mit.edu/6.830/lab5.patch
```

- To apply the patches:

```
$ patch -p5 < lab5.patch
patching file src/simpledb/BufferPool.java
patching file src/simpledb/Database.java
patching file src/simpledb/HeapPage.java
patching file src/simpledb/HeapPageId.java
patching file src/simpledb/Page.java
patching file src/simpledb/Transaction.java
```

- Some or all of the patches will fail. You can tell that a patch succeeded because it will print something like the above, or it may specify some additional detail, such as:

```
patching file src/simpledb/HeapPage.java
Hunk #1 succeeded at 21 with fuzz 2.
Hunk #2 succeeded at 62 with fuzz 1 (offset 2 lines).
Hunk #3 succeeded at 88 (offset -2 lines).
```

  When it fails it will print something like:

```
patching file src/simpledb/BufferPool.java
Hunk #1 FAILED at 235.
Hunk #2 FAILED at 420.
2 out of 2 hunks FAILED -- saving rejects to file src/simpledb/BufferPool.java.rej
```

You will need to manually apply any patch that failed; you can also manually apply all of the patches if you'd rather do that (see the next item.) Even if a patch succeeds, you should read through the changes below to understand what we are adding to your code.

- To apply patches manually:
  1. Insert the following lines into BufferPool.flushPage() before your call to writePage(p), first ensuring that "p" is a reference to the page being written:

     ```
             // append an update record to the log, with
             // a before-image and after-image.
             TransactionId dirtier = p.isDirty();
             if (dirtier != null){
               Database.getLogFile().logWrite(dirtier, p.getBeforeImage(), p);
               Database.getLogFile().force();
             }
     ```

     This causes the logging system to write an update to the log. We force the log to ensure the log record is on disk before the page is written to disk.

  2. Your BufferPool.transactionComplete() calls flushPage() for each page that a committed transaction wrote. For each such page, add a call to p.setBeforeImage(), as follows:

     ```
             // use current page contents as the before-image
             // for the next transaction that modifies this page.
             p.setBeforeImage();
     ```

     After an update is committed, a page's before-image needs to be updated so that later transactions that abort rollback to this committed version of the page. (Don't just call setBeforeImage() in flushPage(), since flushPage() might be called even if a transaction isn't committing; if you implemented transactionComplete () by calling flushPages() you may need to add an additional argument to flushPages() to tell you whether the flush is being done for a committing transaction or not.)

  3. Add these lines to the start of `class Database` in Database.java:

     ```
         private final static String LOGFILENAME = "log";
         private LogFile _logfile;
     ```

  4. Add this code to the constructor for Database:

     ```
             try {
                 _logfile = new LogFile(new File(LOGFILENAME));
     ```

```
        } catch(IOException e) {
            _logfile = null;
            e.printStackTrace();
            System.exit(1);
        }
```

5. Add this function to `class Database`:

```
    public static LogFile getLogFile() {
        return _instance._logfile;
    }
```

   These three changes simply create a LogFile, ensure it exists at database startup, and provide a way to access the log file.

6. Delete the definition of getBeforeImage() from HeapPage.java.
7. Add these to `class HeapPage` in HeapPage.java:

```
    byte[] oldData;

    public HeapPage getBeforeImage(){
        try {
            return new HeapPage(pid,oldData);
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
        return null;
    }

    public void setBeforeImage() {
        oldData = getPageData().clone();
    }
```

   These methods are used to keep track of the before-image of a page, which will be needed to UNDO changes.

8. Add this line to the end of the HeapPage constructor:

```
        setBeforeImage();
```

   This ensures that the before image is properly set when a HeapPage is initialized.

9. Change the definition of serialize in HeapPageId.java to this:

```
    public int[] serialize() {
        int data[] = new int[2];

        data[0] = tableId;
        data[1] = pgNo;

        return data;
    }
```

10. Add this to `class Page` in Page.java:

```
    /*
     * a transaction that wrote this page just committed it.
     * copy current content to the before image.
     */
    public void setBeforeImage();
```

    This extends the Page interface with setBeforeImage.

11. Add this line to `class Transaction` in Transaction.java:

```
    boolean started = false;
```

12. Change Transaction.start() to:

```
    public void start() {
        started = true;
        try {
            Database.getLogFile().logXactionBegin(tid);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

    This instructs logging system to write a BEGIN record when a transaction starts.

13. Change Transaction.transactionComplete() to:

```
    public void transactionComplete(boolean abort) throws IOException {
        if (started) {
            //write commit / abort records
            if (abort) {
                Database.getLogFile().logAbort(tid); //does rollback too
            } else {
                //write all the dirty pages for this transaction out
                Database.getBufferPool().flushPages(tid);
                Database.getLogFile().logCommit(tid);
            }

            try {
                // release locks
                Database.getBufferPool().transactionComplete(tid, !abort);
            } catch (IOException e) {
                e.printStackTrace();
            }

            started = false;
        }
    }
```

This writes COMMIT or ABORT records to the log when a transaction ends.

- After you have applied the patches, do a clean build (`ant clean` from the command line, or a "Clean" from the "Project" menu in Eclipse.)
- At this point your code should pass the first three sub-tests of the LogTest systemtest, and fail the rest:

```
% ant runsystest -Dtest=LogTest
    ...
    [junit] Running simpledb.systemtest.LogTest
    [junit] Testsuite: simpledb.systemtest.LogTest
    [junit] Tests run: 10, Failures: 0, Errors: 7, Time elapsed: 0.42 sec
    [junit] Tests run: 10, Failures: 0, Errors: 7, Time elapsed: 0.42 sec
    [junit]
    [junit] Testcase: PatchTest took 0.057 sec
    [junit] Testcase: TestFlushAll took 0.022 sec
    [junit] Testcase: TestCommitCrash took 0.018 sec
    [junit] Testcase: TestAbort took 0.03 sec
    [junit]     Caused an ERROR
    [junit] LogTest: tuple present but shouldn't be
    ...
```

- If you don't see the above output from `ant runsystest -Dtest=LogTest`, something has gone wrong with the patch or with the un-tarring of `6.830-lab5-supplement.tar.gz`. You should figure out and fix the problem before proceeding; ask us for help if necessary.

## 3. Rollback

Read the comments in LogFile.java for a description of the log file format. You should see in LogFile.java a set of functions, such as logCommit(), that generate each kind of log record and append it to the log.

Your first job is to implement the rollback() function in LogFile.java. This function is called when a transaction aborts, before the transaction releases its locks. Its job is to un-do any changes the transaction may have made to the database.

Your rollback() should read the log file, find all update records associated with the aborting transaction, extract the before-image from each, and write the before-image to the table file. Use raf.seek() to move around in the log file, and raf.readInt() etc. to examine it. Use readPageData() to read each of the before- and after-images. You can use the map tidToFirstLogRecord (which maps from a transaction id to an offset in the heap file) to determine where to start reading the log file for a particular transaction. You will need to make sure that you discard any page from the buffer pool whose before-image you write back to the table file.

As you develop your code, you may find the Logfile.print() method useful for displaying the current contents of the log.

Exercise 1: LogFile.rollback()

Implement LogFile.rollback().

After completing this exercise, you should be able to pass the TestAbort and TestAbortCommitInterleaved sub-tests of the LogTest system test.

## 3. Recovery

If the database crashes and then reboots, LogFile.recover() will be called before any new transactions start. Your implementation should:

1. Read the last checkpoint, if any.
2. Scan forward from the checkpoint (or start of log file, if no checkpoint) to build the set of loser transactions. Re-do updates during this pass. You can safely start re-do at the checkpoint because LogFile.logCheckpoint() flushes all dirty buffers to disk.
3. Un-do the updates of loser transactions.

Exercise 2: LogFile.recover()

Implement LogFile.recover().

After completing this exercise, you should be able to pass all of the LogTest system test.

## 3. Logistics

You must submit your code (see below) as well as a short (1 page, maximum) writeup describing your approach. This writeup should:

- Describe any design decisions you made, including anything that was difficult or unexpected.
- Discuss and justify any changes you made outside of LogFile.java.

### 3.1. Collaboration

This lab should be manageable for a single person, but if you prefer to work with a partner, this is also OK. Larger groups are not allowed. Please indicate clearly who you worked with, if anyone, on your writeup.

### 3.2. Submitting your assignment

To submit your code, please create a `6.830-lab5.tar.gz` tarball (such that, untarred, it creates a `6.830-lab5/src/simpledb` directory with your code) and submit it.
You may submit your code multiple times; we will use the latest version you submit that arrives before the deadline (before 11:59pm on the due date). If applicable, please indicate your partner in your writeup. Please also submit your individual writeup as a PDF or plain text file (.txt). Please do not submit a .doc or .docx.

### 3.3. Submitting a bug

Please submit (friendly!) bug reports. When you do, please try to include:

- A description of the bug.
- A `.java` file we can drop in the `src/simpledb/test` directory, compile, and run.
- A `.txt` file with the data that reproduces the bug. We should be able to convert it to a `.dat` file using `PageEncoder`.

If you are the first person to report a particular bug in the code, we will give you a candy bar!

## 3.4 Grading

50% of your grade will be based on whether or not your code passes the system test suite we will run over it. These tests will be a superset of the tests we have provided; the tests we provide are to help guide your implementation, but they do not define correctness. Before handing in your code, you should make sure produces no errors (passes all of the tests) from both `ant test` and `ant systemtest`.

`Important`: before testing, we will replace your `build.xml` and the entire contents of the `test/` directory with our version of these files. You should therefore be careful changing our APIs. In other words, we will untar your tar file, replace the files mentioned above, compile it, and then grade it. It will look roughly like this:

```
$ gunzip 6.830-lab5.tar.gz
$ tar xvf 6.830-lab5.tar
$ cd 6.830-lab5
[replace build.xml and test]
$ ant test
$ ant systemtest
[additional tests]
```

If any of these commands fail, we'll be unhappy, and, therefore, so will your grade.

An additional 50% of your grade will be based on the quality of your writeup and our subjective evaluation of your code.

We've had a lot of fun designing this assignment, and we hope you enjoy hacking on it!