COMP 530 Assignment 2: Record/Page Management

Due Monday, February 12th at 11:55 PM.

1. The Task

In Assignment 2, your task is to put database records together into pages, and to then put such pages together into files. I have implemented much of the Record infrastructure for you. I'll provide you with around 1,000 lines of nasty C++ code that (1) can read and write records from a string of bytes, (2) can evaluate functions over records, (3) can manage table schemas and create records from schemas, and (4) can save meta-data information (such as schemas) to and retrieve information from the catalog file, among other things.

The thing that I have not provided for you is code that reads and writes records from pages, and then uses the buffer manager to store and retrieve those pages. That is what you need to build.

Basically, you need to provided implementations of the two classes defined in the files `MyDB_PageReaderWriter.h` and `MyDB_TableReaderWriter.h`. For the most part, the interfaces are not too complicated. The former class is layered on top of the buffer manager, and allows us to view a buffer page as a sequence of records that we can modify or access. The latter class, in turn, is layered on top of the former, and allows us to view the sequence of pages in a file is a huge sequence of records, or a database table.

2. Details, Details, Details

2.1 Records

One of the most complicated aspects of any database system is the code for interpretation and manipulation of individual records. It is tricky because it requires the use of a schema to interpret the bits that make up the record in a meaningful way. Fortunately for you, I have provided code for all of this. You'll need to look over this new code, but at the highest level, the way that this works is that we associate a schema with each table. That schema, in turn, can be used to produce `MyDB_Record` objects. Those objects can do things like read their data contents from text files and binary strings, write data contents to text streams and to binary strings. Importantly, records can also build and return functions (lambdas) over their contents. These lambdas can perform computations on the contents of the record (performing math over attribute values, comparing attribute values, and so on) and will be vital later on when we try to actually execute queries.

An important thing to realize is that it is relatively expensive to create records and to create functions over them, and we want to be able to process database tables that

have millions of records. Hence, the idea is not to create a new `MyDB_Record` object for each record physically stored in a database table. Instead, the idea is to create a single `MyDB_Record` object (or maybe a couple of them) and then use those to process the entire table by repeatedly reading chunks of data into the same record objects. This is efficient and easy to do, and the assumption that this is the way that `MyDB_Record` objects are going to be used is built into the interface of the `MyDB_PageReaderWriter` and `MyDB_TableReaderWriter` classes.

That is why, if you look at the specification for `MyDB_PageReaderWriter` and `MyDB_TableReaderWriter`, you will see that the methods over those objects that return iterators take as an argument a smart pointer to a record. The idea is that the returned iterator will then repeatedly read contents from the page/file into the *same* record object. Thus, the record object is recycled and not recreated for each record that is physically stored in the file.

2.2 The `MyDB_RecordIterator` Class

Iteration is accomplished via the `MyDB_RecordIterator` class. When asked to create an iterator, both `MyDB_PageReaderWriter` and `MyDB_TableReaderWriter` return a smart pointer to an object of this type; the way that we'll subsequently write code to scan all of the records in a file or in a page is to call a method to create one of these iterators, and then use the iterator to process all of the file's/page's records in sequence.

Note that `MyDB_RecordIterator` is a virtual class, so you will need to supply one or more concrete classes that implement it; your `MyDB_PageReaderWriter.getIterator ()` and `MyDB_TableReaderWriter.getIterator ()` will then return smart pointers to instances of those concrete class/es. (I would recommend that you have one concrete class returned by `MyDB_PageReaderWriter.getIterator ()` and another returned by `MyDB_TableReaderWriter.getIterator ()`). How you define those concrete class/es is totally up to you. As is standard when implementing an iterator in an object-oriented language, the parent object who is asked to create the iterator will typically "inject" a reference/pointer to itself or its contents into the iterator when it constructs the iterator, so that the iterator can subsequently access the data in the parent. The iterator will also typically have some state that allows it to sequentially access all of the data in the parent.

Note that, as described above, for performance reasons, these iterators work by repeatedly filling up the **same**, client-specified `MyDB_Record` object with the next data stored in the file/page; they do not create new `MyDB_Record` objects when iterating.

2.3 My Code, Your Code

Note that the `.zip` file that I am supplying you with contains my complete implementation of A1. You can either use my A1 code (no warranty!!), or you can replace my A1 code with your own. If you are happy with your code, it is recommended

that you use yours; after all, it is much cooler to have a system that is running your code rather than mine. But it is fine to use my code.

If you do replace my code with your own buffer manager code, please realize that there is one tiny little change in the buffer manager between A1 and A2. Specifically, I had to add a `getPageSize ()` method to `MyDB_BufferManager` that returns the page size used by the buffer manager. Since you undoubtedly have this information stored in your buffer manager, it should take you all of 60 seconds to implement this method!

Note that I have also extended the interface of some of the A1 classes that I suppled previously (`MyDB_Table.h` now has a lot more methods than it had in A1) but these changes are all backwards compatible in the sense that there is now additional functionality (no functionality has been removed or changed) so it should not affect your ability to just drop your A1 code into A2.

2.4 My Code, Your Code, and the Honor Code

Next, let me say something about how the honor code applies here, since I'm giving out my A1 solution. Simply stated, you are not allowed to distribute my code to **anyone** who is not taking the class right now… **ever**! Doing this constitutes a violation of the honor code. I'm particularly concerned about my code falling into the hands of students who might take the class in the future. Thus, a hard and fast prohibition on distributing my source code.

2.5 Testing And Grading

Again, we're using `Qunit`. Again, I have written a few `Qunit` tests. When you build your project using the `SCons` build tool these tests will be complied and an executable will be produced.

Again, if you can make it through my test cases, it is likely that you've worked through the majority of the bugs in your buffer manager, and you've got most of the functionality done. However, you'll probably want to create a few additional test cases of your own—some simpler ones that you can use early on as you develop your code, and some additional, nasty ones just to make sure that everything is working.

When you turn in your code, and it's time for us to grade, we'll run the test cases I've suppled, as well as several others that won't be made public until after the turnin. You'll be graded on your code's success in passing all of the test cases, though we revere the right to browse through your code and take off additional points if it appears you are missing some functionality or have somehow hacked something in a sketchy sort of way. You won't be graded on style and comments. However, I strongly encourage you to take this opportunity to put your best software engineering practices to use.

2.6 Project Difficulty

From talking with students, I think that many people found A1 to be a bit difficult. A2 is easier IMO, though I wouldn't call it easy! Here is a listing of all of my code, to let you know how much code you can expect to write:

```
cmj4$ wc -l DatabaseTable/*/*.cc DatabaseTable/*/*.h
      52 DatabaseTable/source/MyDB_PageReaderWriter.cc
      28 DatabaseTable/source/MyDB_PageRecIterator.cc
     119 DatabaseTable/source/MyDB_TableReaderWriter.cc
      34 DatabaseTable/source/MyDB_TableRecIterator.cc
      43 DatabaseTable/headers/MyDB_PageReaderWriter.h
      32 DatabaseTable/headers/MyDB_PageRecIterator.h
      32 DatabaseTable/headers/MyDB_RecordIterator.h
      59 DatabaseTable/headers/MyDB_TableReaderWriter.h
      37 DatabaseTable/headers/MyDB_TableRecIterator.h
     436 total
```

So my A2 solution includes around 200 lines of source, plus you'll need to write/extend several header files.

3. Getting Started

The instructions this time around for getting started are much the same as the instructions last time around, so I won't reproduce them here.

4. Turnin

Simply zip up all of your source code and then turn it in on Canvas (make sure to archive into the zip format, and not some other archiving format. If you choose to use something else, we'll take off a few points!). Please name your archive A2.zip. Please do not change the original directory structure, except for perhaps adding some new files. The root should be a directory called A2, with two subdirectories Build and Main. And so on.

And remember, **to get *any* credit on A2, your code must compile and run on Clear**. That way, we have a common environment for grading and we don't have to spend time getting your code to compile.

Finally, and this is important: **include a README file in the root of your project with any important information**, including the names of the one or two people who worked on the project.

And also: if you work with a partner, **only turn in one copy of your source**. Otherwise, we'll possibly grade your submission twice, and end up getting quite annoyed.