Project 1: File Management and B+ Trees

Logistics

Due date: Tuesday 10/11/16, 11:59:59 PM

Grading

This project is worth **15%** of your overall course grade. Your project grade will be out of 100 points:

- 90 points for passing all of the tests provided for you. All tests are in `src/test/java/edu/berkeley/cs186/database`, so you can run them as you write code and inspect the tests to debug. Our testing provides extensive unit testing and some integration (end-to-end) testing.
- 10 points for writing your own, valid tests. The tests **must** pass both your implementation and the staff solution to be considered valid tests.

Extra Credit

You can earn up to 10 points in extra credit:

- 5 points for submitting **one week** before the due date (Tuesday 10/04/16, 11:59:59 PM).
- Up to 5 points for how effective your tests are. The way we will determine the effectiveness of your tests is based on how many edge cases you identify that the rest of the class did not identify. You **will not** be graded on how many other students' tests you pass.

Background

The goal of these projects is to give you experience implementing (simple versions of) the actual algorithms that you are learning in the course. In this project, you will be dealing with managing the way records are stored on pages and the way B+ Tree indices are structured.

In later projects you will be building on this code and adding other functionality. In Project 2, you will implement a more efficient query execution than the one provided in this project. In Project 3, you will make your database capable of supporting multiple concurrent transactions. But more on this later. For now, we will focus on the lowest levels of our database.

In order to make these projects more tractable, we aren't implementing all the functionality that you might find in a fully-fledged relational database. There are a number of restrictions we've imposed so as not to ruin your semester:

- We will only be implementing fixed-length records. If you remember from class, this means that whenever you specify the schema of the table, you have to specify the length of any potential variable-length fields.
- We've limited the kinds of data types you will be dealing with to integers, floats, strings, and booleans.
- We are not implementing the full SQL-92 specification. You will find more details below in the Query Generation section.
- B+ Trees indices will not be able to delete keys, therefore won't be able to merge/underflow nodes.

Additionally, we've written a bunch of code already to get you started! You'll find more technical details below, but the high-level idea is that we aren't asking you to deal with file or page allocation. We're doing all of that work for you so that you can focus on understanding exactly how to manage pages and records. That **doesn't** mean that those other things we've done for you aren't important aspects of building a database. It just means that we've decided not to focus on those things for this project.

Lastly, we've set up a function-oriented query interface for you. This current implementation is **intentionally inefficient**. At every stage of the query processing pipeline, we're fully materializing every record in memory. As you'll learn in the next couple weeks, that's a really bad idea. We're using it for this project to let you write end-to-end tests (and to enable our own), but you'll be making it significantly more efficient in the next project! We'll explain more about how to write your own tests using this query interface down below.

Getting Started

As usual, you can get the assignment from the Github course repository (assuming you've set things up as we asked you to in homework 0) by running `git pull course master`. This will get you all of the starter code we've provided, this project spec, and all of the tests that we've written for you.

We've also generated all of the API documentation of the code as webpages viewable here

Java

All of the projects in this course will involve programming with Java. For these projects you should have Java 7 (aka 1.7) installed on your machine, not Java 8 (aka 1.8). You can download Java 7 from Oracle's website.

NOTE: If you're running Windows, make sure that the JDK directory where you installed Java (e.g. `C:\Program Files\Java\jdk1.7.0_79`) is added to both your `PATH` and `JAVA_HOME` environment variables. If you're developing on Windows you should know how to do this, otherwise it's good practice to learn.

Maven

For the projects, we'll be using Maven as our build manager. If you don't have it installed and use a package manager like Homebrew or apt-get you can run `brew install maven` or `sudo apt-get install maven3`, respectively to install Maven. Otherwise, you can find more information about downloading and installing on their website.

NOTE: If you're running Windows, make sure that the `\bin` directory of the unzipped Maven folder (e.g. `C:\Program Files\Maven\bin`) is added to your `PATH` environment variable.

To compile your code, you can run `mvn clean compile`. To run all of the tests that we've provided for you, you can run `mvn test`. To run a single test file, you can run `mvn test -Dtest=TEST_NAME` (e.g. `mvn test -Dtest=TestTable`). You should run these commands from the `projects` directory (the directory that contains this `Project1Spec.md` file). The first time you run these commands, Maven will download a bunch of dependencies and may take a while. **NOTE**: `mvn test` is what we're going to be using to run your tests!

Right off the bat, you should be passing a number of the tests. In particular, any tests corresponding to `DataType`'s and any of the `io` code should be passing. Please ensure that they are before moving forward. If those components are not working properly on your system, then it's likely that none of the rest of the code will either!

Writing Code

You're more than welcome to write code using any text editor you'd like. However, in our experience, using IntelliJ, an IDE from JetBrains, makes developing in Java a lot easier. IntelliJ provides you a bunch of great tools like:

- Maven integration for compilation and testing
- A debugger that allows you to insert breakpoints and inspect data structure internals visually
- Automatic syntax checking to catch bugs before compilation
- Autocomplete that lets you use methods in other files without having to look them up

... and a ton of other stuff!

Starter Code

Package Overview

Like we said earlier, there's a bunch of code that we've provided for you. We strongly suggest that you spend some time looking through it before starting to write your own code. There's a bunch of stuff that you're going to find really useful. In this document, you'll find a high level overview of the kinds of things we've implemented for you. You can find more detailed documentation in the JavaDocs inline with the code.

- At the top level, all code is contained within the `database` package. The `Database` class is simply the integration of table, index, and query package methods that ties everything together. Those packages are described below.
- The `datatypes` package provides a `DataType` interface that encapsulates data values. We've provided implementations for the four data types we'll be dealing with in this project (`int`, `float`, `boolean`, and `String`). These data containers know how to serialize and deserialize themselves for storing as byte arrays. **You should not need to change any code in this package.**
- The `io` package provides you a fully-fledged paging system and buffer manager. The two interfaces you will want to understand are the `Page` interface and the `PageAllocator` interface. The `Page` interface allows you to manipulate bytes in a particular `Page`. `PageAllocator` will allow you to allocate new pages and fetch pages in a file. Don't stress if you don't understand everything that's happening in the code in these files. We've tried to document it as well as we can, but some things are pretty complicated. Just make sure you understand how to consume the existing interfaces! **Please don't modify any of the code in this directory.**
- The `table` package provides you the beginning of an implementation of relational database tables. We've set up some basics for you, like creating a `PageAllocator` for each table. We've also provided some helper methods like `writeBitToHeader` that you're probably going to want to use! Part 1 of the project will be finishing up this table implementation. **Please don't change the interfaces in any of these classes, or your tests won't pass.**
- Within the `table` package, you'll also find a `stats` package. This package gives you a simple set of statistics for each table (`numRecords`, `Histogram`'s, etc.). **You don't need to concern yourself with this package right now, but it'll be integral to the next project.**
- The `index` package provides the beginning of a B+ Tree implementation that you're going to finish up. Again, we've abstracted away most of the lower-level functionality for you. Make sure you don't duplicate our work, so take a look at the existing methods in classes like `LeafNode` and `InnerNode` before getting started. Part 2 of the project will be finishing up this B+ Tree implementation. **Please don't change the interfaces in any of these classes, or your tests won't pass.**
- The `query` package provides you a query processing implementation and query generation interface. The `QueryOperator` provides an interface for a bunch of different operators. In Project 2, you will be extending this to implement more efficient operators. For now, you can use the existing operators, which are implemented such

that they fully materialize all tuples in memory before processing them. For this project, you will mainly concern yourself with the `QueryPlan` interface. The methods in `QueryPlan` will allow you to easily generate queries. **You should not need to change any code in this package.**
- The `concurrency` package provides a lock manager for coordinating concurrent database transactions. **You don't need to concern yourself with this package right now, but it'll be integral to Project 3.**

Query Generation

The `QueryPlan` interface allows you to generate SQL-like queries without having to parse actual SQL queries. All your integration tests should start by creating a transaction (the `Database.Transaction` class). You can call `Database#beginTransaction` to start a new transaction. Make sure you call `Transaction#end` at the end of the test to clean up your transactions and release your locks! **You will not able to start another Transaction until the end of the active one until concurrent Transactions are implemented in Project 3. THE DATABASE WILL HANG!**

Once you have a `Transaction`, you can run *ad hoc* operations on the database by using the `Transaction#getRecord`, `Transaction#addRecord`, etc. methods. If you would like to run queries on the database, you can create a new `QueryPlan` by calling `Transaction#query` and passing the name of the base table for the query. You can then call the `QueryPlan#where`, `QueryPlan#join`, etc. methods in order to generate as simple or as complex a query as you would like. Finally, call `QueryPlan#execute` to execute the query and get a response of the form `Iterator<Record>`. You can also use the `Transaction#queryAs` methods to alias tables.

As a quick example, a simple query might look something like this:

```
// create a new transaction
Database.Transaction transaction = this.database.beginTransaction();

// alias both the Students and Enrollments tables
transaction.queryAs("Students", "S");
transaction.queryAs("Enrollments", "E");

// add a join and a where to the QueryPlan
QueryPlan query = transaction.query("Students");
query.join("Enrollment", "S.sid", "E.sid");
query.where("E.cid", PredicateOperator.EQUALS, "CS 186");

// execute the query and get the output
Iterator<Record> queryOutput = query.execute();
```

You can find more examples in the `QueryPlanTest`.

Your Assignment

Alright, now we can write some code! **NOTE**: Throughout this project, you're more than welcome to add any and all helper methods you'd like to write. However, it is very important that you **do not change any of the interfaces that we've given you**. It's also a good idea to always check the course repo for updates on the project.

Part 1: Tables

The first part of the project involves completing the implementations of `Schema` and `Table` that we've started for you. Like we said earlier, you will be implementing **fixed-length records** in this section. Booleans, integers, and floats are already of fixed-length, so this primarily affects our `StringDataType` implementation. Whenever you create a `StringDataType` field, you will need to specify the number of bytes that should be reserved for that string.

1.1 Schemas

You'll need to implement the `Schema#verify`, `Schema#encode`, and `Schema#decode` methods. The contracts provided by these methods are fully explained by the JavaDocs. For the `encode` and `decode` methods, you might find using the `java.nio.ByteBuffer` class useful.

1.2 Creating and Retrieving Records

Once you've finished, `Schema,` you should start by implementing `Table#addRecord` to add a new `Record` to the table.

However, before adding records to a page, you'll have to do some arithmetic in `Table#setEntryCounts` to figure out exactly how many slots are on a page. Pages in our system are a fixed size `Page.pageSize`, which is currently set to 4KB. Since each record has a fixed size, you should be able to pretty easily figure out the optimal number of records that can be stored on a page. Make sure you account for the slot header (one bit (not byte) per record!).

We then suggest you implement `Table#checkRecordIDValidity` as it will be useful for the rest of the `Table` class. The last part of this chunk is implementing `Table#getRecord`. This should allow you to pass some of the basic `Table` tests that we've provided.

1.3 Updating and Deleting Records

The next thing you'll need to do in this section is implement `Table#updateRecord` and `Table#deleteRecord`. Once you've implemented these methods, you should be passing all of the non-iterator `Table` tests.

1.4 Iterators

To finish up this section, you will need to implement the `TableIterator` subclass we've started for you within the `Table` class. To make your life easier, we've provided you with a `PageIterator` in `PageAllocator` that lets you iterate over all pages. All you have to do is return the valid records from each page. Remember that `Page 0` is reserved for the table header.

1.5 Testing

Once you've implemented all of these methods, you should be passing all of the `Table` and `Schema` tests, and *most* of the `QueryPlan` tests. These tests don't necessarily require you to have B+ Trees implemented yet. They'll pass as long as you return the correct data. We strongly recommend you start writing tests once you've wrapped your head around the code to try to catch some of the edge cases that you might have missed. It's generally a good idea to write your own tests as you go along due to the fact that the given tests don't cover all edge cases.

Part 2: B+ Trees

The second part of the project is focused on the B+ Tree implementation.

2.1 Inserting Keys

You will first implement B+ Tree key insertion. In order to support key insertion, you will first have to implement the methods `InnerNode#locateLeaf` and `LeafNode#locateLeaf`.

Once you've done that, you can implement the `InnerNode#splitNode` and `LeafNode#splitNode` methods. Remember that when dealing with B+ Trees, we copy keys up from the leaf node and push keys up from the inner nodes during splits.

2.2 Iterators

Similar to the first part of this project, you will implement the `BPlusIterator` subclass of `BPlusTree`. This iterator will be a little different from the `TableIterator`, however. You will need to support equality lookups, bounded range lookups, and full index scans with this iterator, which are respectively called from `BPlusTree#lookupKey`, `BPlusTree#sortedScanFrom`, and `BPlusTree#sortedScan`. Think about how to reuse your code for each case.

2.3 Testing

If you've completed all the sections up to this point, you should now be passing **all** of the tests that we've given you. Again, we strongly encourage you to write tests as you go to try to catch any relevant bugs in your implementation.

Part 3: Testing

We can't emphasize enough how important it is to test your code! Like we said earlier, writing valid tests that **test actual code** (i.e., don't write `assertEquals(true, true);`, or we'll be mad at you) is worth 10% of your project grade.

CS 186 is a design course, and validating the reasonability and the functionality of the code you've designed and implemented is very important. We suggest you try to find the trickiest edge cases you can and expose them with your tests. Testing that your code does exactly what you expect in the simplest case is good for sanity's sake, but it's often not going to be where the bugs are.

To encourage you to try to find interesting edge cases, we're going to give you some extra credit. If you find edge cases that most other students didn't find, you can get up to 5 points of extra credit. Keep in mind that if some other people find the same edge cases as you, that doesn't mean you won't get extra credit -- so don't treat this as a competition!

Writing Tests

In the `src/test` directory you'll notice we've included several tests for you already. You should take a look at these to get a sense of how to write tests. You should write your tests in one of the existing files according to the functionality you're trying to test.

All test methods you write should have both the `@Test` and `@Category(StudentTest.class)` annotations. We have included an example test in the `TestTable` class:

```
@Test @Category(StudentTest.class) public void testSample() { assertEquals(true, true); // Do not actually write a test like this! }
```

Then whenever you run `mvn test`, your test will be run as well.

Part 4: Feedback

We've been working really hard to give you guys the best experience possible with these new projects. We'd love to improve on them and make sure we're giving reasonable assignments that are helping you learn. In that vein, please fill out this Google Form to help us understand how we're doing!