CS127 Indexing Programming Assignment

Out: Oct 24th, 2019 - Due: Nov 13th, 2019

Disclaimer: We reserve the right to update this homework (within reason) while this homework is active. We won't be making massive changes and will probably be
bug fixes or added features. If we do so, we will send a Piazza post about it and you will be responsible for pulling these changes. To learn how to do it, please read the Git Setup section below.

Setup

Access the Github Classroom code at: https://classroom.github.com/a/mvG8z4w2

Machine to do your work:

1. We are providing access to a Linode cloud instance (the Linode machine). You
can do work there by SSH or SSHFS. You'll need to access the cloud instance
to submit your benchmarks and see how you are doing relative to the rest of
the class. To access the machine by SSH:

```
bash $ ssh B1234567890@96.126.110.231
```

where B1234567890 is your banner ID. Your password is your banner ID as well. Please change your password when you first log in:

```
bash $ passwd
```

which will prompt you in the following way:

```
bash (current) UNIX password:  Enter new UNIX password:  Retype
new UNIX password:  passwd:  password updated successfully
```

1. You can also do your work locally, you're going to need the Java JDK and JRE.
We're using Java 11 but I don't think you need this exact version. You'll also need Python3 and Pandas if you want to generate data other than the one
we included in the repo.

Git Setup

1. If you are reading this you must have setup Github and the Github classroom
   code. Great! You'll need to set this up in both your local and the Linode machine.

2. You'll make your first edit: enter your banner ID in `Main.java` and save Main.java.

```
class Main {

    // Enter your banner ID here
    public static String bannerid = "B0123456789";

...
```

1. You'll find your remote repos as below. Note that the remote repo below uses
   `https`. If you want to setup SSH, feel free. Also note that yourusername will be replaced with your Github user name.

```
$ git remote --v
origin  https://github.com/brown-cs127/indexing-youusername.git (fetch)
origin  https://github.com/brown-cs127/indexing-youusername.git (push)
```

Push the banner id change you made to your repo:

```
$ git add .
$ git commit -m "changing bannder id"
$ git push origin master
```

1. We want to add a remote repo so that you can get updates to the code that we
   make:

```
$ git remote add upstream https://github.com/brown-cs127/cs127-indexing.git
$ git remote --v
origin          https://github.com/brown-cs127/indexing-yourusername.git (fetch)
origin          https://github.com/brown-cs127/indexing-yourusername.git (push)
upstream        https://github.com/brown-cs127/cs127-indexing.git (fetch)
upstream        https://github.com/brown-cs127/cs127-indexing.git (push)
```

1. When you want to push and pull or to pull our changes:

```
$ git pull origin master     # pull from your repo
$ git push origin master     # push to your repo
$ git pull upstream master     # pull from changes from our repo
```

1. If you are working in the machine we provided, all the software required (Java and Python) are available. If you are working locally, you'll need to install the Java JDK and JRE and Python3. You'll also need to install Python's Pandas package.

Franco's going to have tech hours throughout the week to help with setup

Phases

We envision this project works out in 3 large phases. Each phase requires you to check in with a TA. We'll release sign-in sheets for when these check-ins are going to occur.

Phase #1: Code base and B+ Tree

There are two tasks to the first phase:

1. First, get familiar with the codebase. We are providing two recitations to walk through the codebase.
2. Second, write the B+ tree code. Want to know how a B+ tree works in a fun
   and interactive visualization? Check this out:

https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html

Phase #2: Bind B+ Tree to your table

You're going to need to bind your BPlusTree code into your Table and modify your
Table to use the B+ Tree. You're going to have to use this as a primary (clustered) index, and as a secondary index.

Phase #3: Investigate alternative indexing schemes

In this phase, you should explore alternatives to the B+ tree you build. Later in this document, we'll discuss what counts and doesnt. Test your changes agains the correctness tests and the benchmark. Feel free to add any necessary files to your alternative schemes into the repository.

When you are satisified (or about to run out of time), write a summary of what you did and why in `writeup.md`. You should confirm that there is some gain to your alternative indexing scheme by comparing benchmark results. If it's actually slower, then also explain why.

Submitting

Among the pushed commits in the repo, Github Classroom will take the last commit
within the deadline.

For phase 3's writeup, write it in `writeup/writeup.md`. To make it easier to read, use the markdown syntax. If you are reading this file in a text editor, you are currently reading this document in markdown syntax. Include this file in your commits.

Learning Goals and Tasks

In this programming assignment, students learn how to:

- implement a B+ tree index in a main-memory column-store
- understand the implications of using an index on the rest of the system
- understand the trade-offs of primary and secondary indices
- explore other indexing schemes for different workloads

Tasks

Students will accomplish the following tasks, in order of priority, along with a recommended timeline:

1. Implement a B+ tree (4 days)

2. Implement a primary and secondary indices using this B+ tree for a `Table` (1
   day)

3. Explore and implement one alternative avenue of indexing a database (3 days)

4. Write a one-page summary of optimizations performed on the B+ tree (if any)
   and results of item 3 above (<1 day)

The remainder of the time should be used to get your extra credit!

Grading

Students will be graded in the following way:

1. B: A baseline grade of for getting correct results

2. B+: Minimal working implementation of a B+ tree (pun intended)

3. A-: Implementing primary index for a `Table` and at least one secondary index

4. A: Exploration and implementation of one alternative avenue of indexing

5. Extra Credit for highly performant implementations based on the bench-marks
   described in the next subsection:

   (a) 40 extra points for the 50th percentile of the class

   (b) 50 extra points for the 25th percentile of the class

   (c) 60 extra points for the 10th percentile of the class

6. These extra points will be applied at the end of the semester to your overall grade.

An important note: If you submit code that doesnt pass the same tests that passed when you first got the code, we'll give you an **automatic C**.

Code

Summary of files

Although it seems like there's alot of code, most of it is testing code! The goal of the code base is to keep things small and streamlined. This allows you the student to go bazinga(s) over modifications and optimizations without fear that you shouldn't modify some file.

The files you can (and likely should modify) are the following:

- `Main.java`: contains the main entry point to the code

- `BPlusTree.java`: contains the B+ tree skeleton class. This compiles but doesn't work! You'll need to fill in the functions. We suggest some functions
  for you to make.

- `Column.java`: contains the Column class. You'll modify this to include your
  indexing mechanism after you've built your B+ tree.

- `Table.java`: contains the Table class. Defines the interfaces that allow you
  to perform range queries on the table class. You'll have to change the implementation of these methods to account for any changes you've made in the
  other files.

You can make any changes to the files above so long as you follow the instructions on what you are able to change and not. You can also make new Java
files that contain your own code.

The following files don't need to be changed but you should know what they are used for. It might be helpful to change some of them but it wouldnt be necessary.

- `Tuple.java`: contains the definition of a Tuple. This is the logical Tuple outside of the table. The Table takes an instance of a Tuple and stores the information in the Tuple, not necessarily using the instance of the Tuple.

- `TupleIDSet.java`: contains the definition of a set of Tuple IDS (Integers). This is what is returned from a filter predicate, for example, the Tuple IDs where attribute A is between some value x and y.

- `Filter.java`: contains the definition for a filter predicate. It's passed into the `Table.filter()` method so that the method can execute the code.

- `MaterializedResults.java`: contains the definition for the MaterializedResults class. An instance of this is returned when you want to get
the tuples from a set of tuple IDs.

The following files deal with tests. Do not edit these files!:

- `Benchmarks.java`: Contains benchmarking code where we measure the query
response time of your Table.

- `BPlusTreeTests.java`: Tests the correctness of your B+ tree. It puts and removes values from your B+ tree. It also checks the internal node structure of
your B+ tree using the `INodeValidator` class defined in this file.

- `TableTests.java`: Checks that the table's functions are return the right values.

What works and what doesn't?

We've written the table so that it works at a very fundamental level. You can confirm this by running `make testtable` as described below. To ensure correctness you should modify the code while ensuring the Table's tests stay correct.

We've given you skeleton code for the B+ tree and so this definitely doesn't work.

Building and Running

Unfortunately, the code is written in Java. As such, you need the Java JDK. We use version 11. I'm no Java programmer so I use a Makefile.

To compile the code:

```
make
```

We included some initial test data in your repo so you don't need to run this. However, if you need to generate the test data, you'll need Python3 and Pandas and run:

```
make data
```

To run the tests the code:

```
make testtable
```

```
make testtree
```

Expected testtable output can be found in `data_validation/expected`

To run the benchmarks:

```
make bench
```

To run register the benchmarks in our benchmark DB:

```
make register
```

You can query the Benchmark db using the file `benchmarks/query.sql`. You can
modify this file - the code in it is some starter or sample code:

```
make query
```

To do this benchmark, register, query all at once:

```
make bench register query
```

Registering to and querying from the benchmarks database requires you to be in our Linode machine!

To clean up class files:

```
make clean
```

Tests for Correctness

B+ Tree

You can see the tests for correctness in `BPlusTreeTests.java`. If you run this, these tests will fail. You'll need to define your B+ tree first! It does the following checks:

1. Inserts data into the tree
2. Most of the data from the tree
3. Inserts and removes data from the tree for some ratio
4. Checks internal nodes for children relationships. Note, that this is a very purposely loose check because there are many ways of writing a B+ tree.

If the tree performed correctly, you'll see something similar to the following:

```
$ make testtree
Seed for random generator: 0
Successfully inserted: 1000 items
Successfully got all 1000 expected values
Successfully deleted: 500 items with 500 items remaining
Successfully written 806 and deleted 194 and 1000 operations with writeRatio 80
Successfully validated tree
```

Table Tests

You can see the tests for correctness for the Table in the `TableTests.java`. To test for correctness, it reads the file `data` we gave you in the `data\_validation` directory. It writes results to the `data\_validation/results` directory. It then calls a python script to compare your results to the data in `data\_validation/expected` directory.

It checks to make sure your implementation is correct for the following table methods:

- `Table.insert`
- `Table.load`
- `Table.filter`
- `Table.update`
- `Table.delete`

In doing so, it also ensures that method for `Table.materialize` works correctly.

Benchmarking

Arguably equivalent in importance to correctness is performance. Indices are built to ensure performant databases. As such, we provide a benchmarking framework to estimate how fast your implementations run. You can run these benchmarks by running `make bench`

You can find the benchmarks in the file `benchmark_results.txt`. To register these benchmarks in the class's benchmarks, you'll need to run this in the Linode machine we provided.

An implementation overview

The code implements a simple column-oriented `Table` supporting only the Integer data type. This `Table` supports a few essential database operations: inserts, deletes, filters, and updates. In addition, the table supports a new operator you might not have seen called "materialize". These are discussed below.

Tuples

`Tuple.java` contains the code for a `Tuple`. You'll see that the `Tuple` class is simply a hashtable where the key is a String, and the value is an Integer. The key is the attribute, and the value is well, the value of that attribute. For example, the code below inserts an instance of a `Tuple` into the table `someTable`. This `Tuple` has two attributes: "A" = 1, and "B" = 2.

```
Tuple t = new Tuple();
t.put("A", 1);
t.put("B", 2);
some‘Table‘.insert(t1);
```

Columns

`Columns.java` contains the code for the *physical representation* for the *logical attribute* in a `Table`. Because the `Table` is a column-oriented data storage scheme, the set of values of a given attribute is stored as an array in contiguous memory. You'll see that this column is simply an `Vector` of Integers.

Tables

`Table.java` contains a bulk of the initial code. It has several internal variables:

- `String name` is the name of the `Table`. It's not used much but there anyways
  for future use.

- `Hashtable<String, Column> attributes` is the collection of columns and the
  attribute name corresponding to those columns.

- `Vector<Boolean> valid` is an array that indicates whether a tuple is valid or not (for example, when the tuple is deleted)

Internally, the column-oriented storage allows the `Table` to use the position of a tuple in its column as the tuple's id. In the code below, `Tuple` t1 will be referenced internally with id 0, t2 with id 1, and t3 with id 2. `Tuple` t2's value for attribute "A" can be accessed in the Column corresponding to attribute A equivalent to `columnA[0]`.

```
// Instantiate a new table with two attribtues "A" and "B".
HashSet<String> cols = new HashSet<String>();
cols.add("A");
cols.add("B");
Table someTable = new Table("test_table", cols);

// Insert tuples into the Table
Tuple t1 = new Tuple();
t1.put("A", 1);
t1.put("B", 2);
someTable.insert(t1);

Tuple t2 = new Tuple();
t2.put("A", 1);
t2.put("B", 3);
someTable.insert(t2);

Tuple t3 = new Tuple();
t3.put("A", 3);
t3.put("B", 1);
someTable.insert(t3);
```

The `Table` class provides several methods we describe below.

insert: Inserts a `Tuple` into the table by appending the values of the tuple to each of the columns. Internally marks that tuple as valid by also pushing `true` into the `valid` variable.

delete: Given a set of `Tuples`' IDs, marks the `Tuples`' as invalid by setting the `valid` variable for that id as `false`. *Note:* this extremely naive approach is done on purpose. At some point, there are performance implications to not garbage collecting or physically deleting these entries.

filter: Given a `Filter`, returns the `HashSet` of valid tuple IDs that qualify. See the description of a `Filter` later in the document.

update: Given a a set of `Tuples`' IDs, an attribute, and value, updates the `Tuples` attribute to the given value.

materializeResults: Given a set of attributes and a set of valid tuple ids (`Integers`), returns a `MaterializedResults` of the attributes and tuples requested. This is particularly necessary for column stores because a `Tuple`'s values are stored separately in columns. This operation is also quite expensive and is typically only used as the very last step in the query plan. This technique is called *late tuple materialization.* We describe a `MaterializedView` later in the document.

load: Given a list of `Tuples`, load all of these tuples into the Table.

Tip: the `load` method currently calls `insert` each time. This *might* be an issue when you are building an index…

Other stuff relating to Tuples, Columns, and Tables

Filter: A recursive data structure. At the bottom of the recursion, a Filter contains an `attribute`, a `low`, and a `high` value. If the low and high values are present, then it's a range filter `low <= x <= high`. Note that if `low = high`, then this is equivalent to an equality filter, it's an equality filter.
If `high` is null and `low` is present, then this is a query is equivalent to `x >= low`. If vice versa, then it's `x <= high`.

MaterializedResults: This is a very lightweight wrapper around `Vector<Tuple>`. To generate this list you'll need to iterate over the columns in the `Table` and put together the tuples as previously descibed.

Indexes and B+ Tree

The B+ tree code in `BPlusTree.java` currently stands alone and is unused anywhere else besides the tests. After completing the B+ tree code, you're going to need to write code to use the B+ tree as an index for columns in your table.

Other indexes At the end of the project, you're going to experiment with one of two things:

1. An significant optimization to your B+ tree. Note that this should fundamentally change how your B+ tree works. Minor optimizations such as
   changing branching factor doesn't count!

2. An alternative indexing scheme. We'll be discussing this as we progress in class. Some options include LSM Trees, Bitmap Indexes, HashMaps. In all cases, you DON'T need to implement these yourself - you can use anything provided in the Java standard library. You're simply going to need to write code on how to use these data structures as an index.

For example, if you use a HashMap, Java has a bunch of HashMap implementations. You can choose one and write code to use it as an index for your column or table.

Hardware Details:

```
$ lscpu # gives CPU information (alternatively: $cat /proc/cpuinfo)
$ free -m # gives RAM information (alternatively: $cat /proc/meminfo)
```