COMP 530 Assignment 7: Putting It All Together

Due Thursday, May 3rd at 5:00 PM

1. The Task

Your task, very simply, is to put together all of the pieces we've built this semester together into a single, final product. Basically, when your program is given an SQL query, you will compile and type check it using the A5 compiler, translate the output of the A5 compiler into relational algebra, optimize the relational algebra, and then execute the relational algebra, print the results, and then clean up any garbage files that you have created.

Note that in `A7.zip`, I've included my solution to A6, as well as the skeleton code I provided for A5. Note that this A5 code maintains a `MyDB_TableReaderWriter` and `MyDB_Table` object for each of the tables currently defined in the system (it also maintains a `MyDB_BPlusTreeReaderWriter` object for each B+-Tree file). It also has some functionality for loading those tables from text (you can try this; from MyDB the command line, create the supplier table, and then type "`LOAD supplier FROM supplier.tbl`"). Also note that it is possible to create a table using a B+-Tree, by simply adding "`AS BPLUSTREE ON some_attribute`" at the end of the `CREATE TABLE` statement. Note that I did not include my own type checking code in `A7.zip`, because all of the queries in the test suite are semantically correct. Also note that you should be able to easily merge any A5 code that you find useful for A7 into my MyDB `main.cc`; the changes compared to the A5 `main.cc` are minor.

2 Creating the TPC-H Data

For the remainder of A7, we'll be using the TPC-H benchmark database, which is the one whose schema you already have already used for A5. You can either create this database yourself (see `http://www.tpc.org/tpch/`) or you can download a version that I've posted onto Dropbox, at:

`https://www.dropbox.com/sh/o5qytjw7qncglem/gtNx3EklIT`

Note that we are using the default "scale factor one" database, which is about one GB in size.

3 The Path to Finishing the Assignment

The first thing you'll need to do is to design some data structures that can hold a relational algebra expression. Then you'll write code to take the output of the parser and create a simple relational algebra expression with a single selection over a single table. You should just ignore projections and grouping/aggregation at this point. Then write some code that can actually execute that expression, using the A6 code.

Once you've got that, you might want to work on just being able to run a simple relational algebra expression that has aggregation but no joins, and no optimization; that is, build and run a query plan that simply has an aggregate, followed by a selection to do the final computation over the results of the aggregation (see, for example, query 4, which requires a grouping and aggregation, plus a final computation over the results of the aggregation to compute the `“order priority was " + o.o_orderpriority` part of the query). This will get you a passing grade on the assignment, since several of the queries I'll give you don't even have any joins.

One you've actually been able to run something and you have a few points guaranteed on the assignment, you should work on being able to run a query with joins, but without any optimization—it just picks an arbitrary ordering (perhaps based on the ordering of the tables in the query) and executes the query. A good heuristic is to join the tables from smallest to largest (where the size is defined in terms of the number of tuples). The, push down selections as far as they will go. This will get most of the queries running. Your executions will be quite fast if you do things such as project away unnecessary attributes as early as possible.

If you can get all of this, work on a full-blown implementation, with optimization. The easiest way to do this is to write some code that constructs all possible join orderings (as discussed in class about a month ago). For a particular join ordering, push your selections and projections down as far as they can go, and then "cost" the resulting plan, by predicting the number of tuples that the plan will create. Note that there is now functionality within the `MyDB_Table` class that allows you to get the number of distinct values counts for all of the attributes in a table, as well as the number of tuples in the table. Choose the best plan that you can construct. At that point, you have your optimized relational algebra expression. If there is any aggregation/grouping, add that to the plan at the very end of the best plan. Then execute it.

Thats it!

Well, not really. For extra credit you can add the ability to choose an appropriate relational algebra implementation. Specifically, for up to 10% extra credit, make it so that you can use a B+-Tree, if appropriate (you can demonstrate this by organizing `lineitem` as a B+-Tree via the `l_shipdate` attribute, and trying to run queries 1 and 5 using this).

You can also experiment with choosing the `ScanJoin` versus the `SortMergeJoin`, depending on which is more appropriate for the situation. The `ScanJoin`, for example, may be faster for joins including a large table—though you might want to experiment yourself—but it won’t work when the smaller table won’t fit into RAM (note however, that all of our queries, if a good plan is chosen, can be run using only the `ScanJoin`; `SortMergeJoin` is needed only in the case of a bad plan, where a very large intermediate output is created.

4 Testing, Grading, and Turnin

In the end, when the user issues a query, your program should evaluate it, and then print at most the first 30 records from the output file to the screen, the number of records resulting from running the query, as well as the number of seconds taken to run the query.

Included with the assignment is a suite of ten test queries. When you are ready to prepare your turnin doc, you should fire up your program and run those queries (in order) in single session. Skip any ones that you can't run. To get 65% of the credit on the assignment, you need to be able to run three of those correctly. That should be relatively easy, since three of the queries don't even have a join and can be run without even having a functioning optimizer. After that, running each of the remaining six queries gets you an additional 5%, for a total of 100%. Include a B+-Tree, and you get to 110%.

For turnin, you need to include a PDF file that includes all of the results you get from running those 10 queries, with all team member names, along with all of your source code as a zip archive. Please turn in two different files: one PDF, and one zip.

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