Competition Overview

Overview

Welcome to the CS186 contest where you can optimize RookieDB over a set of queries! You're free to make any change to your database but certain assumptions and rules are in place. Your goal is to minimize the total I/O count across all X queries.

Assumptions

1. The queries are run sequentially (i.e., the lock manager and recovery managers are not tested).
2. None of the queries we'll be using are INSERT or CREATE SQL statements (We will still run them to import data but the I/Os associated with them won't be counted). Your query optimizer may choose to materialize intermediate outputs.
3. RookieDB traditionally uses 4KB pages. However, we will be using 16KB pages.
4. Your database will have access to 256 MB of Java memory. To be precise, the Java VM will run with up to 256MB of memory — this will include memory for everything to do with running the program, including your buffer pool.

Rules

You may modify a subset of the files in RookieDB (listed later in this doc) in any way as you please except:

1. We will replace your buffer and disk manager with our implementation to ensure consistency when counting I/Os, but you may change the buffer manager's eviction policy.
2. You may not add any intermediate tables by hand or create your own tables.
3. You are allowed to create indexes over any column and table. Keep in mind that RookieDB only supports generating indexes over one column.
4. We expect your query result to match the staff solution.

TPC-H

The Transaction Processing Performance Council (TPC) has a number of benchmarks for datasets; the tables we'll be looking at are from TPC-H, a decision support benchmark. The tables are meant to simulate real business data and the queries aim to help inform business decisions.

Tables:

- Table customer - c_custkey (int), c_name (string, 25), c_address (string, 40), c_nationkey (int), c_phone (string, 15), c_acctbal (float), c_mktsegment (string, 10), c_comment (string, 117)
- Table lineitem - l_orderkey (int), l_partkey (int), l_suppkey (int), l_linenumber (int), l_quantity (float), l_extendedprice (float), l_discount (float), l_tax (float), l_returnflag (string, 1), l_linestatus (string, 1), l_shipdate (date), l_commitdate (date), l_receiptdate (date), l_shipinstruct (string, 25), l_shipmode (string, 10), l_comment (string, 44)
- Table nation - n_nationkey (int), n_name (string, 25), n_regionkey (int), n_comment (string, 152)
- Table orders - o_orderkey (int), o_custkey (int), o_orderstatus (string, 1), o_totalprice (float), o_orderdate (date), o_orderpriority (string, 15), o_clerk (string, 15), o_shippriority (int), o_comment (string, 79)
- Table part - p_partkey (int), p_name (string, 55), p_mfgr (string, 25), p_brand (string, 10), p_type (string, 25), p_size (int), p_container (string, 10), p_retailprice (float), p_comment (string, 23)
- Table partsupp (parts supplemental) - ps_partkey (int), ps_suppkey (int), ps_availqty (int), ps_supplycost (float), ps_comment (string, 199)
- Table region - r_regionkey (int), r_name (string, 25), r_comment (string, 152)
- Table supplier - s_suppkey (int), s_name (string, 25), s_address (string, 40), s_nationkey (int), s_phone (string, 15), s_acctbal (float), s_comment (string, 101)

Additionally, there are 4 sizes of table data. The size listed is for all tables combined.

- Tiny: 32 KB total
- Small: 1MB total
- Medium: 3MB total
- Large: 5MB total

The tables and queries can be found under `resources/contest/tables` and `resources/contest/queries`, respectively. The queries, called “Workloads” in the tests, are run exactly once in the order shown. Feel free to take a closer look at both the tables and queries to get a better sense of what's happening.

Code

`contest/ContestSetup.java` is one of the main files that you should try to modify. In addition, you may modify any of the files from Projects 2 or 3, namely:

- src/main/java/edu/berkeley/cs186/database/index/InnerNode.java
- src/main/java/edu/berkeley/cs186/database/index/LeafNode.java
- src/main/java/edu/berkeley/cs186/database/query/QueryPlan.java
- src/main/java/edu/berkeley/cs186/database/query/SortOperator.java
- src/main/java/edu/berkeley/cs186/database/query/join/BNLJOperator.java
- src/main/java/edu/berkeley/cs186/database/query/join/GHJOperator.java
- src/main/java/edu/berkeley/cs186/database/query/join/SortMergeOperator.java

`contest/ContestRunner.java` is provided as a convenience when testing out your changes. The final contest is actually run through JUnit in `test/contest/TestContest.java`. **Your final score is equal to the number of I/Os generated by the `runLargeContest` test.**

Building an Index

If you would like to build an index over a particular column, go into `contest/ContestSetup.java` and add the table and column in `INDICES_TO_BUILD`. For example, if I wanted to add an index over `customer::c_custkey`, I would add `{"customer", "c_custkey"}` as an array element. The contest runner will build the requested indexes before running any queries. **Do note that building the index will count as part of your total IO cost, as we do not bulk load. Furthermore, remember that our default implementation of B+ Trees have some design limitations, such as not allowing duplicates (you are allowed to modify that though, as mentioned).**

Eviction Policy

To change the eviction policy, replace the variable `EVICTION_POLICY` in `contest/ContestSetup.java` with your new eviction policy.

`ContestRunner.java`

This is a mock contest runner that you can use and modify. There are 4 static variables which you can change as you would like.

-

- `PRINT_N_ROWS`: Print n rows of the query output.
- `EXPORT_ROWS`: If you would like to serialize query output and schema, set this to true.
- `WORKLOAD_SIZE_TO_RUN`: Corresponds to Tiny, Small, Medium, and Large database sizes. Change this to run the mock contest on a different size.
- `EXPORT_PATH`: If you want to export the serialized output and schema, a file path you want to serialize the results to.

Some Potential Ideas

- Build indices in the ContestSetup, this is the quickest way to cut I/Os
- Modify the optimizer to consider other types of joins
- Implement better join algorithms (example)
- Materialize the intermediate results and store them in the buffer pool
- Store data in column major order
- Modify/replace the B+Tree index with a more efficient/capable index structure

Previous
Getting Started

Next
Submitting the Assignment

Last updated 4 months ago