CSE 510 - Database Management System Implementation

Spring 2020

Phase III

Due Date: Midnight, April 22nd

1 Goal

The version of the MiniBase I have distributed to you implements various modules of a relational database management system. Our goal this semester is to use these modules of MiniBase as building blocks for implementing a Bigtable-like DBMS.

2 Project Description

The following is a list of tasks that you need to perform for this final phase of the project:

- Implement a command-line program getCounts. Given the command line invocation

  getCounts NUMBUF

  the program will return the numbers of maps, distinct row labels, and distinct column labels. If these were not already computed during the batch insertion, then at most NUMBUF many buffers need to be used to compute them.

- We Know that BigTable is a distributed database which means that a single table is distributed across different locations as chunks. These parts should be always in sync and work together to address user queries or updates on the DB. We will be simulating these chunks of DB on a single system. We will assume the data persisted into DB is distributed across 5 different heap files and all these heap files should be in sync to address user actions. To do this task follow the belo mentioned steps.
  - Modify the bigDB class such that the constructor does not take type as input. In other words, in the same bigDB database there may be data stored according to different storage types.
  - Modify the bigT class such that the bigt method does not take type as input. In other words, in the same bigT table there may be maps stored according to different storage types.
  - • Modify the storage type definitions as follows:

    – Type 1: No index

    – Type 2: one btree to index row labels

    – Type 3: one btree to index column labels

– Type 4: one btree to index column label and row label (combined key) and

– Type 5: one btree to index row label and value (combined key)

Note: 5 different heapFiles are needed to be created for each storage type.

- Modify the batchInsert program from the previous phase as follows: Given the command line invocation

  batchinsert DATAFILENAME TYPE BIGTABLENAME NUMBUF

  where DATAFILENAME and BIGTABLENAME are strings and TYPE is an integer (between 1 and 5), the program will store all the maps in a bigtable. The name of the bigtable that will be created in the database will be BIGTABLENAME. If this bigtable already exists in the database, the tuples will be (logically) inserted into the existing bigtable.

  Note:

  – in the same bigT table, different maps may be stored according to different storage types, based on the batch in which they were inserted;

  – the insertMap method ensures that there are at most three maps with the same row and column labels, but different timestamps, in the entire bigtable – irrespective of which batch they were inserted in and which storage type they are subjected to(heapFile);

  – the openStream methods creates a unified stream by pulling maps (potentially stored using different storage types) and ordering them according to the orderType parameter.

  As before, at the end of the batch insertion process, the program should also output the number of disk pages that were read and written (separately).

- Implement a mapInsert program, which given the command line invocation

  mapinsert RL CL VAL TS TYPE BIGTABLENAME NUMBUF

  where BIGTABLENAME are strings and TYPE is an integer (between 1 and 5), the program will store the map woth row label RL, column label CL, value VAL, and timestamp TS in a bigtable. The name of the bigtable that will be created in the database will be BIGTABLENAME. If this bigtable already exists in the database, the tuples will be (logically) inserted into the existing bigtable.

- Implement a program query. Given the command line invocation

query BIGTABLENAME ORDERTYPE ROWFILTER COLUMNFILTER VALUEFILTER NUMBUF

  the program will access the database and printout the matching maps in the requested order. Each filter can be

– “*”: meaning unspecified (need to be returned),

– a single value, or
– a range specified by two column seperated values in square brackets (e.g. “[56, 78]”)

Note: you can see that the TYPE parameter does not exist for this query which means that it should fetch records from all the storageTypes.

Minibase will use at most NUMBUF buffer pages to run the query (see the Class BufMgr).

At the end of the query, the program should also output the number of disk pages that were read and written (separately).

- Implement a BigT join operator

  – RowJoin(int amt of mem, Stream leftStream, java.lang.String RightBigTName, attrString ColumnName, java.lang.String JoinType)

  amt_of_mem - IN PAGES
  leftStream - a stream for the left data source
  RightBigTName - name of the BigTable at the right side of the join ColumnName - condition to match the column labels

  JoinType - Need to implement two joins of your choice

  The output is a stream of maps corresponding to a BigT consisting of the maps of the matching rows based on the given conditions, such that

  ∗ Two rows match only if they have the same column and the most recent values for the two columns are the same.

  ∗ The resulting rowlabel is the concatenation of the two input rowlabels, seperated with a “:”

  ∗ The resulting row has all the columnlabels of the two input rows, except for the joined column which occurs only once in the bigtable – and only with the most recent three values.

- Implement a command-line program join. Given the command line invocation

rowjoin BTNAME1 BTNAME2 OUTBTNAME COLUMNFILTER JOINTYPE NUMBUF

the program will access the database to rowjoin the two bigtables and create a new type 1 big table with the given table name. Minibase will use at most NUMBUF buffer pages to run the query (see the Class BufMgr). At the end of the query, the program should also output the number of disk pages that were read and written.

- Implement an external rowSort operator that results in a type 1 BigT in which the rows are sorted (in non decreasing order) according to the most recent values for the given column label:

rowSort(Stream inStream, java.lang.String ColumnName, int n_pages)

Like its Minibase counterpart, Sort needs to support a getNext() method that returns maps of the resulting rows in the specified order.

- Implement a command-line program rowsort. Given the command line invocation

```
rowsort INBTNAME OUTBTNAME COLUMNNAME NUMBUF
```

The program will sort the big table. Minibase will use at most NUMBUF buffer pages to run the query (see the Class BufMgr).

IMPORTANT: If you need to process large amounts of data (for example to sort a file), do not use the memory. Do everything on the disk using the tools and methods provided by minibase.

- Implement a command-line program which creates on demand indexes on provided storageType

```
createindex BTNAME TYPE
```

At the end of the query, the program should also output the number of disk pages that were read and written.

3 Deliverables

You will be provided with a sample data set. You have to submit the following

- Your source code properly commented, tared or zipped.
- The output of your program with the provided test data and the driver.
- A report. The report should contain a short description of the operators.
- The report should clearly state who did what. This will be taken very seriously! So, be honest. Be prepared to explain on demand (not only your part) but the entire set of modifications. See the report specifications.
- The report should clearly state the comparisons of two join a that were implemented
- A confidential document (individually submitted by each group member) which rates group members' contributions out of 10 (10 best; 0 worst). Please provide a brief explanation for each group member.