## CS 4432 – Project 2 Report

### Douglas Lally – [dlally@wpi.edu](mailto:dlally@wpi.edu)

### Nate Miller – [nwmiller@wpi.edu](mailto:nwmiller@wpi.edu)

The following document contains each of the sections described in the deliverables section of the Project 2 Specification. For simplicity, we have included deliverables 1 through 4 in this report. This includes the following deliverables: installation instructions, testing scenario discussion, project design description, and a bug report.

Installation Instructions:

Testing Scenarios Discussion:

Task 3: Evaluation of Index-based Query Performance (Selection & Join Queries):

The testing of this task could not be completed due to underlying issues with the SimpleDB application, which prevented creation of any type of index.

Task 5: Evaluation of modified sort-merge-join operator:

To test the performance of the modified sort-merge-join operator (Task 4) we created a test program (src/testPrograms/SortMergeJoinTest.java) that was adapted from the DataGenerator Tool provided for the project. This program follows the project specification description of how to perform evaluation. Namely, two tables are created and populated with randomized data. Then two queries are executed over the tables, structured in such a way as to join the two tables and utilize the sort-merge-join operator. This test program requires that the ExploitSortQueryPlanner (src/simpledb/planner/ExploitSortQueryPlanner.java) be enabled in the SimpleDB configuration (src/simpledb/server/SimpleDB.java).

The run time of each of then queries is measured by measuring the system time in nano seconds before and after the queries are each executed. From this, the elapsed time is calculated to give a measure of how long each query execution took. In doing so, we were able to evaluate the performance of the modified sort-merge-join operator. No I/O cost was measured as the project specification for Task 5 did not specify the need to do so for this testing scenario.

Design Description:

Task 1 & 2:

Task 4:

In order to carry out the modifications for task 4 a number of files were looked at and updated. In particular, the TableInfo class (/src/simpledb/record/TableInfo.java) was modified to hold the sorted status flag for each table to support the checking of this status during the sort-merge-join operation. In addition, the SortPlan (src/simpledb/materialize/SortPlan.java) and MergeJoinPlan (src/simpledb/materialize/MergeJoinPlan.java) classes were modified. As the plan classes is where the heart of the operation exists, these files were the focus of modifications. By checking the status of the plans’ table sort statuses before creating sort plans in the merge join plan we were able to only utilize sort plans when the tables have not been sorted. Otherwise, the unmodified plan is utilized ensuring that no unnecessary sorting takes place when scans are made. Additionally, the TableScan class (src/simpledb/query/TableScan.java) was modified slightly to ensure that any UpdateScans cause a plans’ table sorted status to be reset back to false.

Results and Analysis:

Task 3 Results:

The testing of this task could not be completed due to underlying issues with the SimpleDB application, which prevented creation of any type of index.

Task 5 Results:

The evaluation test program created to test Task 4, as described above in the Testing Scenarios Discussion section, was executed multiple times in order to gather results. The number of records in each test table was varied between runs. The program was executed with test table record sizes of 500, 1000, 5000, and 10000 per test table. The database was completely deleted in between each execution of the test program. The execution time of each of the two queries during the test program execution were recorded and analyzed based on expected results.

We saw that the execution time of the second query was lower than that of the initial query. This trend was held across all observed executions of the test program. These results matched what we expected in that the second query should execute faster due to the fact that the sorting phase of the sort-merge-join operator is skipped.

A file has been included in this submission, sort-merge-join\_test\_log.log, which contains selected results of executions of the test program from Task 5. This file contains results of two test program runs for each test table record sizes (500, 1000, 5000, 10000).

Bug Report:

1. An underlying issue exists in the SimpleDB application itself in regards to the index functionality. Specifically, the built-in simple hash index functionality is flawed in some manner. As a result, we were unable to create any indices on tables. Trying to create an index would throw a NullPointerException. We were unable to find a solution to this problem as it is an issue with SimpleDB itself and not any modifications we made. This was verified by trying to create indices using a fresh and completely unmodified version of SimpleDB. As a direct result of this problem, we were unable to perform the testing required by Task 3 of the assignment.
2. When testing Task 4, we ran into issues where SimpleDB was throwing BufferAbortException (<http://dpuadweb.depauw.edu/bhoward_web/csc480/SimpleDB/apidocs/simpledb/buffer/BufferAbortException.html> shows the documentation for the exception itself). Through debugging via our IDE we were able to trace this error to the SimpleDB buffer manager not being able to fulfill a request. We further pinpointed the problem to the buffer pool being full, which was causing transactions to fail as the buffer manager could not pin new buffers. In order to alleviate this problem, we modified the default buffer size of SimpleDB. In the SimpleDB class (src/simpledb/server/SimpleDB.java) we increased the BUFFER\_SIZE. We increase the max buffers from 8 to 1000. We did not carry out detailed testing with different buffer numbers to find a minimal number to avoid the exception. However, we found that this increase in max buffers did prevent the exception.