2

In exp2.cc, we generate a random 10\*100 record and calculate its size by function fixed\_len\_sizeof().

The output of exp2 is:

fixed\_len\_sizeof(record) returned 1000, expected 100\*10=1000

So function fixed\_len\_sizeof() agrees with our calculation.

3

The two figures below are performance of writing and read fixed length pages.

After comparing the two figures, we can say that read and write have similar trends when page size changes. That is, both rates of read and write increase as page size increases. Note that reading data from CSV file using getline(csv,) in our code takes a lot of time, so we did not count the time used to read from CSV.

Page based format is superior to CSV because CSV operations are very costly. We cannot locate a record directly in CSV file, we have to read it one by one. And it also takes more space.

The shortcoming is that when we want to insert something, it can only be added at last or we have to scan the whole pagefile to find an empty spot.

4

1) The figure below is performance of csv2heap.

As we can see, the time spent to convert the CSV file to a heap file decreases as page size increases at first. But when page size is too large, we do not need a whole page, so part of the page is unused, and the time spend increases.

2) The figure below is performance of selecting records from heapfile.

As we can see, the time spent decreases from page size 2000 to 1e07, as larger page size indicates less pages, make the operation faster. But when page size is too large, after there is only one page totally, increase of page size does not make the number of pages smaller. So the time spent increases as page size keeps increasing.

5

1)

As we can see, the time spent to convert the CSV file to a column heap file increases as page size increases. Comparing with the performance in row based CSV2HeapFile, 1KB page size seems to be the optimal performance for CSV2ColStore, but CSV2HeapFile optimal performance seems to much larger than CSV2ColStore. Similar to CSV2HeapFile when CSV2Colstores’s page size is too large, we do not need a whole page, so part of the page is unused, and the time spend increases.

2)

In Select2 with ColStore, the time spent increases as the page size increases. Different from select with HeapFile, Select2 with ColStore does not decreases as Page Size increases, and 1KB also seems to be the optimal page size for selecting and projecting with ColStore HeapFile. Comparing with Select with HeapFile, Select2 with ColStore HeapFile has a much lesser overall time spent when page size is smaller than 1MB. Unlike select with Row based Heap File, the Column based HeapFile only contains that certain attribute and its slot size is much smaller than a row based HeapFile. On the other hand, Select with HeapFile takes much longer time to trasverse through because of the larger dataset contained within a row.

3)

Like select2 with ColStore, the time spent in Select3 increases as the page size increases, but the overall timespent in Select3 is much larger than the time spent in Select2, since select3 is reading records from 2 different ColStore Heap File at the same time. Different from select with HeapFile, Select3 with ColStore does not decreases as Page Size increases, and 1KB also seems to be the optimal page size. Similar to Select2 with Column based HeapFile, Select 3 has a much lesser overall time spent when page size is smaller than 1MB, when it compares with Select with HeapFile. And it takes slightly longer overall time spent than Select2, because of the extra HeapFile to search for in Select3.