A1 Part II: Relational data layout on disk

Liliya Aliyeva

For this part of the assignment, all the functions were executed on my local machine and I discuss the results based on the data I gathered. As a result, sizeof(int) on my machine is 4 bytes, which means that the minimum page size for a record of 1000B is 1005B. All the tests were conducted on csv file containing 1000 records.

Page Sizes Used in terms of records stored per page: [1, 2, 5, 10, 100, 1000, 5000, 10000] records per page

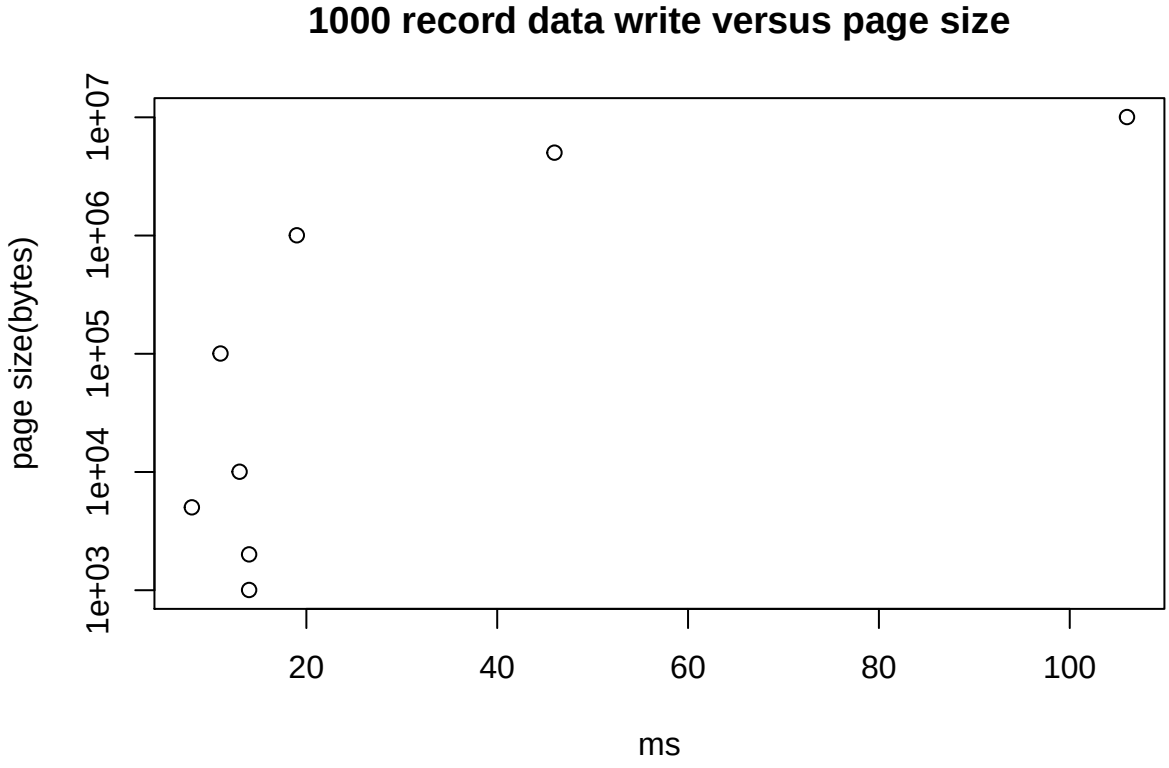
1. Record Serialization

1. The size of fixed length serialization of records in the table For fixed length records with 100 attributes and each attribute value being 10 bytes each, each record size serialized is 1000 bytes.
2. fixed_len_sizeof() uses a similar calculation. However, due to the constraints with time the record size was hardcoded. The fixed_len_sizeof() function is not used and instead, for row and column store heaps, the record size is passed as an argument.

2. Page Layout

1. Plot the performance (records/seconds) versus page size for write and read.

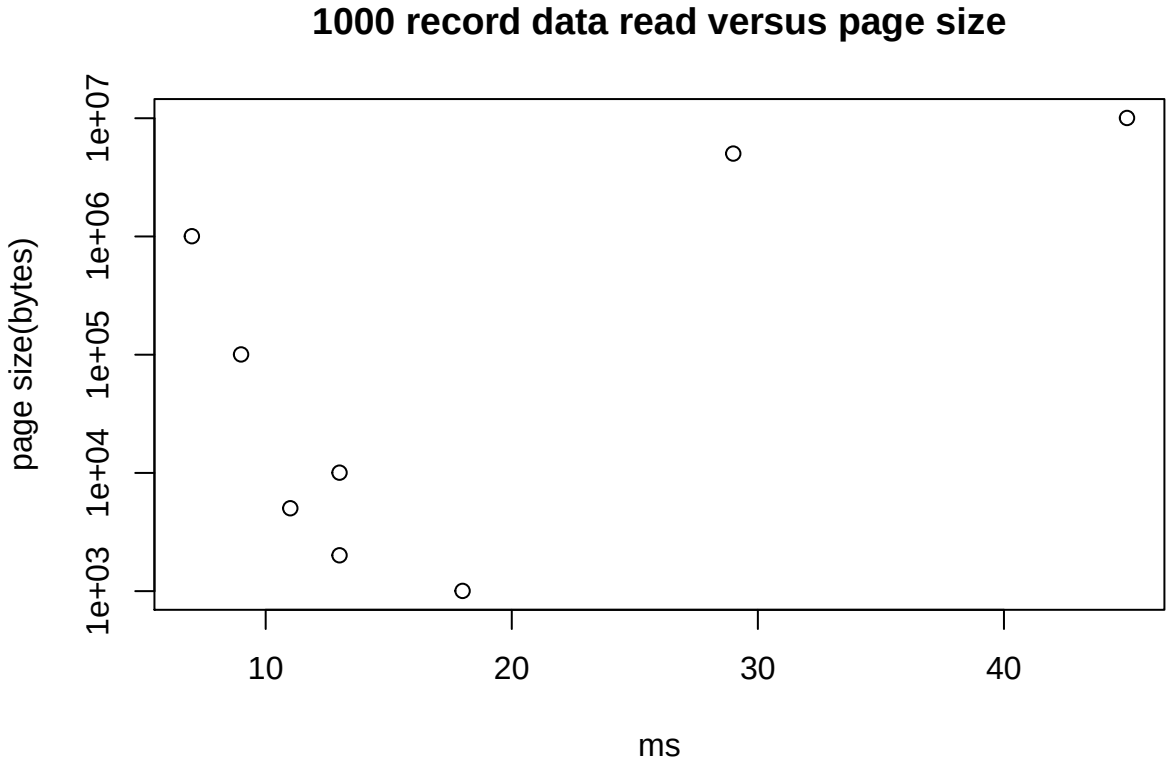
The number of Bytes is represented on a logarithmic scale to make the results more visible.



For 1000 records, although there seems to be a small advantage to storing 5 records per page, there is not much of a difference between storing 1, 2, 5, 10, 100 per page. The performance seems to be around the same.

However, as we try to store more records per page, it becomes expensive. For example to store all records on one page took about 4 times longer and the time keeps increasing with increased sized.

Please note that for reading, the following results were gathered without counting the time it took to print the records to terminal. The code submitted includes the time it takes to print records to terminal in the final time estimate.



Record reading is more efficient that record writing. It practially makes no difference the file size that's chosen to read the records in. However, as the ratio of records/ block size is decreased (i.e. as we increase the page size significantly), it takes more time to read the pages.

2. Advantages

One of the major advantages for storing records in pages is organization. Each page stores the capacity - the number of records per page - and which slots are occupied or free. So although the advantages are not apparent at this stage, we can already surmise that maintaining the files will be a lot easier that it would have been had we simply dumped all of the information without any order.

Additionally, it is particuarly efficient at reading records in.

3. Shortcomings

One major drawback to the way pages are organized is there could be a lot of space wasted because records are large and are stored on a page in full. For instance, if we wanted to store records in a page of 2000, then only one record would fit in that page as we need 1000 bytes to store the record, 4 bytes to store the capacity and 1 byte for the directory. That means that there's now 995 bytes that are left unused. Therefore, space is not being utilized efficiently.

3. Heap File

Performance of the *select* query versus page size.

select performance is relatively smooth regardless of whether it is one record page page or 100s of records per page. However, I could not get my code to be versatile enough to accept any page size possible regardless of the page size which which the heapfile was created. So when testing, the page size had to match the page size with which the heapfile was created. This means that the query performed better. This is because the code never had to make extra calculations that might be involved when there's a disrepancy between page size at creation of the file vs. page size when reading.

However, the differencein page size makes a difference as it is much faster to search records from pages that are already loaded versus loading a page at a time and running the query. Meaning that increasing the page size makes the query more efficient.

There does not seem to be any effect on the range of start and end on the performance of the query.

4. Column Store

1. Based on the performance of csv2colstore it takes significantly longer to build all the heap files.

Unlike the perfomance of csv2heapfile, the performance of csv2colstore is significantly slower. And as the size of the pages increased, so did the amount of time it took to build all the files. The test was run on 1000 records.

2. Compare the performance of *select*2 with that of select in the previous section. Comment on the difference.

It's not hard to see (although I couldn't because of bugs in my code), that the performance of select2 on a particular attribute is a lot more efficient than that of select.

3. Compare the performance of *select*3 with that of *select* and *select*2. Comment on the difference especially with respect to different selection ranges (different values of start and end).

select3 is slower than select2 in virtue of having to pull double the size of pages than that of select2. The selection of ranges would not affect performance any more significantly that it would for select or select2 since only one attribute is being tested.