Tutorial 2

1. We have 12 pages of memory and we want to join tables R and S whith $[R] = 100$ and $[S] = 50$, where $[R]$ and $[S]$ are the numbers of pages needed to store relations R and S, respectively.

   (a) How many disk reads are needed to perform Chunk Nested Loops Join?

   (b) How many disk reads are needed to perform Hash-Merge Join? (Assume that no recursive partitioning is needed.)

2. Assume the relation R contains 10,000 tuples, each tuple taking up 80 bytes, and that the page size is 4 kilobytes. Further assume that we have 408 kilobytes of RAM available for storing data while executing a join algorithm. Given these assumptions, how much faster is the chunk nested loops algorithm going to be compared with the simple nested loops algorithm?

3. Alice and Bob are discussing the chunk nested loops algorithm for computing joins. Alice is claiming that allocating an output buffer the size of two pages, rather than the size of one page, is going to make the algorithm more efficient. Bob is claiming that it is going to make no difference. Is Alice right? Is Bob right? Justify your claim.

4. Alice and Bob are discussing the chunk nested loops algorithm for computing joins. Alice is claiming that the algorithm is going to run faster if we allocate an extra page-size chunk of RAM for reading the outer relation at the expense of an output buffer. Bob is claiming that it is going to slow the algorithm down. Is Alice right? Is Bob right? Justify your claim.

5. Give a scenario (tables, their schemas, their size, and a query) where an equijoin needs to be computed, and where the hash-join algorithm could not be used to process the query.

6. Suppose that tables R and S are such that both of them are sorted and, moreover, R entirely fits into RAM. Which algorithm is going to be the fastest for computing an equijoin of R and S? Justify your answer.

7. You have a slotted page with 80 bytes of free space, and it costs 1 byte to store an interger in a directory entry.

   (a) Whats the size of the largest record you can insert?

   (b) At most, how many 4-byte records can you insert?

8. Give 3 reasons why a DBMS cannot rely on the memory/file management of the OS?

9. What is sequential flooding? What problem does it cause? What are the ways to mitigate the problem?

10. When would you prefer a tree-based over a hash-based index and vice versa?

11. We are using a B+ tree storing our data records in leaf pages. The table contains one billion records. Each records is 200 bytes, each disk page has 16kB (16,384 Bytes) and should be at most 67% full.

    (a) How many leaf pages are required?

    (b) Assume each index entry takes 32 bytes. What is the fanout of the index?

    (c) What is the height (# levels of non-leaf nodes) of the tree? How many I/O operations are required to insert a new record (assuming there is enough space in the leaf page)?

    (d) How many pages are required to store the non-leaf nodes?

12. You have decided to develop a new deals website that pushes nearby deals to user's mobile phones based on their age group. As you are expanding you realize that your service is getting slower, probably a result of the 2 million users in your database. Assume that each user

entry is 2KB in size and that you are mainly performing range queries based on a user's age. Assume the page size is 16KB.

(a) You are storing all your data in a heap file. In the worst case, how many I/O operations are necessary to find all users in a certain age range?

(b) You have decided to create a clustered B+-Tree on the age field. The tree has a fanout of 200 and a height of 3. Assume that you are on average returning 50,000 users per query. On average, how many I/Os are perfomed by such a query?

(c) Assume your B+ tree is unclustered. In the worst case, how many I/Os do you need now?