CS3223: Database Management Systems
Tutorial 4
(Week 6, February 2022)

1. Suppose you have 2 relations, R(A,B,C) and S(B,C,D,E). You have a clustered unique (no duplicate keys) B+-tree index on attribute A for relation R. Assume this index is kept entirely in memory (i.e., you do not need to read it from disk). For relation S, you have two indexes: (a) a non-clustered non-unique B+-tree index for attribute B, and (b) a clustered non-unique B+-tree index for attribute C. Furthermore, assume that these two indexes are kept in memory (i.e. you do not need to read them from disk). Moreover, we assume that all R.B values are also found in column S.B, and all R.C values are also found in the S.C column of relation S. Also, assume that all of the tuples of S that agree on attribute C are stored in sequentially adjacent blocks on disk (that is, if more than one block is needed to store all of the tuples with some value of C, then these blocks will be sequentially located on the disk). Other relevant data: (a) 500 tuples of R are stored per block on disk; (b) R has 500,000 tuples; (c) 100 tuples of S are stored per block on disk; (d) S has 100,000 tuples; (e) S.B has 40,000 distinct values; (f) S.C has 40 distinct values.

   You want to execute the following query:

   ```
   SELECT *
   FROM R, S
   WHERE (R.B=S.B) AND (R.C=S.C)
   ```

   using the following strategy:

   ```
   For every block B of R, retrieve using the clustered index on A for R
       For every tuple r of B
           Use the index on B for S to retrieve all the tuples s of S such that s.B = r.B
               For each of these tuples s
                   If s.C = r.C, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
   ```

   Analyze this strategy carefully in terms of its behavior regarding accesses to disk. Your analysis should consider the behavior both in terms of the number of I/Os, as well as the total access time. Be sure to include in your analysis what accesses to disk are sequential accesses and which ones are random accesses. The total access time is measured in terms of number of random I/Os and sequential I/Os, e.g., X random I/Os and Y sequential I/Os.

2. Referring to the previous question. You are given an alternative strategy:

   ```
   For every block B of R, retrieve using the clustered index on A for R
       For every tuple r of B
           Use the index on C for S to retrieve all the tuples s of S such that s.C = r.C
               For each of these tuples s
                   If s.B = r.B, output r.A, r.B, r.C, s.B, s.C, s.D, s.E
   ```

   Analyze this strategy carefully in terms of their behavior regarding accesses to disk. Your analysis should consider the behavior both in terms of the number of I/Os, as well as the total access time.

Explain which of the two plans (this and the one from the previous question) is therefore better under what circumstances.

3. Consider the join operation R $\bowtie_{R.a = S.b}$ S given the following information about the relations to be joined. The cost metric is the number of page I/Os, and the cost of writing out the result *should be ignored*. Where indexes are used, you may assume that the B+-tree indexes have three levels (format 2).
   - R contains 1,000,000 tuples and has 20 tuples per page.
   - S contains 2,000,000 tuples and has 40 tuples per page.
   - S.b is the primary key for S.
   - R.a is a foreign key that references S.b
   - 100 buffer pages are available (inclusive of input/output buffers)

   a) If neither relation has any indexes built on it,
      I. What is the cost of joining R and S using a block nested-loops join algorithm?
      II. What is the cost of joining R and S using a sort-merge join algorithm? What would the cost be if both relations are already sorted on the join attributes?
      III. What is the cost of joining R and S using a hash join algorithm?

   b) If a clustered B+-tree exists on S.b, would nested-index algorithm be preferred? What are the worst and best case scenarios?

4. The sort-merge join can be refined by combining the merging phases in the sorting of R and S with the merging required for the join. In other words, when sorting R (and S), we do not merge the runs into one single sorted run; instead, the join is performed on a set of sorted runs of R and S, rather than on a single sorted run of R and S. Describe and discuss a sort-merge join with this refinement.

Generate sorted runs of R. Instead of merging the runs into a single sorted run
Stop the merging when the number of runs is < FLR((B-1)/2) where B is the number of buffer pages.
Repeat the above step for S.

Allocate 1 buffer page for each run of R and S.
Allocate 1 buffer page for join output. As tuples of the sorted R and S are produced, they can be checked whether the join predicate is satisfied.

Savings: Cost of reading and writing a single sorted run of R and S