CIS560 – Database System Concepts **Name:_______________**

Homework Assignment 9 [20 points] – due November 15th (in class)

Note 1: Please remember that you are allowed to discuss the assigned exercises, but you should write your own solution. Identical solutions will receive 0 points.

Note 2: For full credit, show your work (not only the final answers).

Exercise I (Operator algorithms) [6 points]

1. Explain how two relations R and S can be joined together using a two-pass partitioned hash join algorithm. In your explanation, use the following facts: R has 90 pages; S has 80 pages; there are 11 pages of memory. Please provide a detailed explanation that includes (1) the goal of each step, (2) how many pages are allocated as input buffer(s), (3) how they are used, (4) how many pages are allocated as output buffer(s), (5) how they are used, etc. You can assume that hash tables have no overhead (a hash table for a relation that has 9 pages will require 9 pages of memory). You can assume a uniform data distribution.

First, we split R into partitions:

- Allocate one page for the input buffer. Allocate 10 pages for the output buffers: one page per partition.
- Partition R: Read in R one page at the time. Hash into 10 buckets. As the pages of the different buckets fill-up, write them to disk. Once we process all of R, write remaining incomplete pages to disk. At the end of this step, we have 10 partitions of R on disk. Assuming uniform data distribution, each partition comprises 9 pages.
- Partition S: Split S into 10 partitions the same way we split R (must use same hash function). At the end of this step, we have 10 partitions of S on disk. Assuming uniform data distribution, each partition comprises 8 pages.
- Perform the join: For each pair of partitions that match
  - Allocate one page for input buffer
  - Allocate one page for the output buffer.
  - Read one 9-page partition of R into memory. Create a hash table for it using a different hash function than above.
  - Read corresponding S partition into memory one page at the time. Probe the hash table and output matches.

2. Consider joining two relations R(x, y) and S(x, z) on their common attribute x using a sort-merge join algorithm. The size of relation R is 1000 blocks and the size of relation S is 500 blocks. Suppose the memory buffer has 101 blocks. Assume that attribute x of relation R has two distinct values (x1 and x2) and the values are evenly distributed in R. Similarly, attribute x of relation S also has the same two values (x1 and x2) and the values are evenly distributed in S. Describe the sort-merge join algorithm and compute the total number of disk I/Os that are needed for the algorithm.

   Remember that the two-pass sort-merge join algorithm that we discussed and analyzed in class requires that all tuples with a common value for the join attribute fit in the memory. If this condition is not satisfied, the sort-merge join algorithm needs additional disk I/O's.

   We effectively have to perform two nested-loop joins of 500 and 250 blocks, respectively, using 101 blocks of memory. Such a join takes 250 + 500*250/100 = 1500 disk I/O's, so two of them take 3000. To this number, we must add the cost of sorting the two relations, which takes four disk I/O's per block of the relations, or another 6000. The total disk I/O cost is thus 9000.

Exercise II (Selection cost) [4 points]

Consider a relation R(a, b, c, d) that has a clustering index on a and non-clustering indexes on each of the other attributes. The relevant parameters are:

B(R) = 1000, T (R) = 5000, V (R, a) = 20, V (R, b) = 1000, V (R, c) = 5000, V (R, d) = 500.

Give the best query plan and the disk I/O cost for each of the following selection queries:

(1) $\sigma_{(a=1)AND(b=2)AND(c=3)}(R)$

Use an index-scan using the nonclustering index on c. Since V(R,c) = 5000, only one tuple should be retrieved. Filter the retrieved tuple for a=1 and b=3. The expected disk I/O cost is 1.

(2) $\sigma_{(a=1)AND(b=2)AND(c<3)}(R)$

Use an index-scan using the nonclustering index on b. Since V(R,b) = 1000, 5 blocks should be retrieved. Filter the retrieved blocks for a=1 and c<3. The expected disk I/O cost is 5.

Exercise III (Physical query plans) [5 points]

Consider two relations R(a, b, c) and S(x, y, z) that have the following characteristics:

B(R) = 600	B(S) = 800
T(R) = 3000	T(S) = 4000
V(R, a) = 300	V(S, x) = 100
V(R, b) = 100	V(S, y) = 400
V(R, c) = 50	V(S, z) = 40

We also have M = 1000 (number of memory blocks).

Relation R has a clustered index on attribute a and an unclustered index on attribute c. Relation S has a clustered index on attribute x and an unclustered index on attribute z. All indexes are B+ trees.

Specify and justify a good physical plan for performing the join

$R \bowtie_{a=z} S$ (i.e., join R and S using the condition R.a = S.z)

Your answer should specify the physical join operator used (hash, nested loop, sort-merge, or other) and the access methods used to read relations R and S (sequential scan, index, etc.). Be sure to give essential details: i.e., if you use a hash join, which relations(s) are included in hash tables; if you use nested loops, which relation(s) are accessed in the inner and outer loops, etc.

Give the estimated cost of your solution in terms of number of disk I/O operations needed (you should ignore CPU time and the cost to read any index blocks).

You should give a brief justification why this is the best (cheapest) way to implement the join. You do not need to exhaustively analyze all the other possible join implementations and access methods, but you should give a brief discussion of why your solution is the preferred one compared to the other possibilities.

A simple hash join where both R and S are scanned sequentially will be as good as anything. The cost of that is B(R)+B(S). We have enough memory to store either R or S in a hash table, so there is no particular advantage to picking either one if we view this query in isolation. From the system's standpoint, it would be better to hash R and then read blocks of S to perform the join. R is smaller, so storing it in the hash table will tie up less memory.

There is no advantage to using indices to read the blocks of the relations compared to a simple scan, and there can be a significant cost. We can scan the blocks of either relation at a cost of B(R) or B(S). If we look for tuples during the join using an index the costs will be higher, even with the clustered index, since we need to find matching tuples in one relation depending on the attribute values of the current tuple in the other, and those may be in any order.

A nested loop join is at least as expensive. We could hold all of one relation in memory and scan it repeatedly as we scan the other a block at a time, but the I/O cost will still be at least B(R)+B(S) and we will be repeatedly scanning the relation stored in memory. A naïve loop join where we read the blocks of the inner relation for each block of the outer one would cost B(R) + B(R)*B(S) or B(S) + B(S)*B(R), depending on which relation we scan in the outer loop, which would be more expensive.

A standard sort-merge join would be worse since we will need to sort one relation, write that to disk, sort the other, then read the first relation back a block at a time to join it to the sorted tuples of the other one. If we sort R first to minimize the number of blocks in the temporary file, the total cost of a sort-merge join would be 3B(R) + B(S). A naïve sort-merge join that writes both sorted relations to temporary files would be even worse, with a cost of 5B(R) + 5B(S).

If we were clever, we could do a sort-merge join where we sort S in memory at the cost B(S), then read the corresponding tuples of R using the clustered index on a at a cost of B(R) for a total cost of B(S) + B(R). But this has the same I/O costs as the hash join, and requires an extra sort operation.

[Grading notes: Answers were not expected to be this detailed or long-winded, but did need to pick the right algorithm, analyze the cost, and give at least a general argument about why this was the best choice.]

Exercise IV (Logical query plans) [5 points]

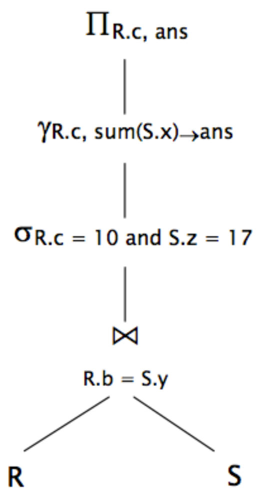
Consider again the two relations R(a, b, c) and S(x, y, z) that we used in Exercise III, with the same characteristics:

B(R) = 600	B(S) = 800
T(R) = 3000	T(S) = 4000

V(R, a) = 300	V(S, x) = 100
V(R, b) = 100	V(S, y) = 400
V(R, c) = 50	V(S, z) = 40

We also have M = 1000 (number of memory blocks).

As in III, relation R has a clustered index on attribute a and an unclustered index on attribute c. Relation S has a clustered index on attribute x and an unclustered index on attribute z. All indices are B+ trees. Now, consider the following logical query plan for a query involving these two relations.

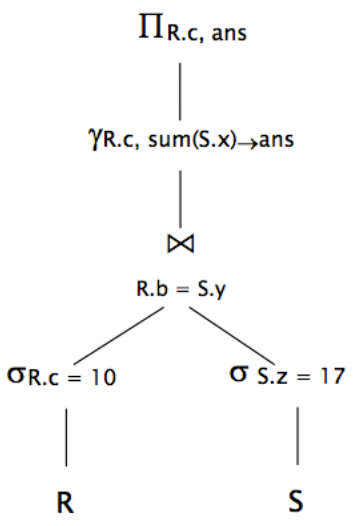


$\Pi_{R.c,\ ans}$
$\gamma_{R.c,\ sum(S.x) \rightarrow ans}$
$\sigma_{R.c = 10 \text{ and } S.z = 17}$
$\bowtie$
R.b = S.y
R
S

Change or rearrange the original logical query plan to produce one that is equivalent (has the same final results), but which is estimated to be significantly faster, if possible. Recall that logical query optimization does not consider the final physical operators used to execute the query, but only things at the logical level, such as the sizes of relations and estimated sizes of intermediate results.
You should include a brief but specific explanation of how much you expect your changes to improve the speed of the query and why.

In the original plan, the estimated size of the join result would be T(R)T(S)/max(V(R,b), V(S, y)) = 30,000 tuples, which then have to be passed through the select operation.

The most useful change would be to push the select operations down below the join. If that is done, the estimated result size of $\sigma_{R.c=10}$(R) is T(R)/V(R,c) = 60 and of $\sigma_{S.z=17}$(S) is T(S)/V(S,z) = 100. The estimated size of the join is then 60*100/400 or 15. The amount of work done by the group and project operations would remain the same as in the original plan.



$\Pi_{R.c,\ ans}$
$\gamma_{R.c,\ sum(S.x) \rightarrow ans}$
$\bowtie$
R.b = S.y
$\sigma_{R.c = 10}$
$\sigma_{S.z = 17}$
R
S