## CPS216 Advanced Database Systems (Data-Intensive Computing Systems, Fall 2009), Assignment 3

- Due date: Tuesday, Nov. 17, 2009, 2.50 PM. Late submissions will not be accepted.
- Submission: In class, or email solution in pdf or plain text to shivnath@cs.duke.edu.
- Do not forget to indicate your name on your submission.
- State all assumptions. For questions where descriptive solutions are required, you will be graded both on the correctness and clarity of your reasoning.
- Email questions to shivnath@cs.duke.edu.
- Total points $=100$.


## Question 1

Points 20
The following information is available about relations $R$ and $S$ :

- Relation $R$ is clustered and the blocks of $R$ are laid out contiguously on disk. $B(R)=1000$ and $T(R)=10,000$.
- Relation $S$ is clustered and the blocks of $S$ are laid out contiguously on disk. $B(S)=500$ and $T(S)=5000$.
- $\mathrm{M}=101$ blocks.
- For simplicity, we will assume that a random access can be done on average in time $t_{r}=20$ ms , and a sequential access can be done on average in time $t_{s}=1 \mathrm{~ms}$. For example, scanning five contiguous blocks on disk, assuming the first access is random, incurs a cost $t_{r}+4 t_{s}$.

1. [6 Points] How will you extend the "Efficient" Sort-Merge Join algorithm (that we learned in class) to minimize cost when our cost model distinguishes between random accesses and sequential accesses? (Note that in class we did not distinguish between random and sequential accesses.) Compute the cost of your algorithm.
2. [6 Points] Design an algorithm for the block nested-loop join of relations $R$ and $S$ which has minimum cost when we distinguish between random and sequential disk accesses. Compute this minimum cost using the parameter values specified above.
3. [8 Points] How does your answer to (2) change if blocks of $R$ are not laid out contiguously on disk? All other assumptions and parameters remain the same as specified above. Compute the minimum cost possible for block nested-loop join in this case.

## Question 2

Points $20=5+3 * 5$
Suppose you have two clustered relations $R(A, X, Y)$ and $S(B, C, Z)$. You have the following indexes on S .

- A non-clustering B-tree index on attribute B for S .
- A clustering B-tree index on attribute C for S .

Assume that both indexes are kept entirely in memory always (i.e., you do not need to read them from disk). Also, assume that all of the tuples of $S$ that have the same value of attribute $C$ are stored in sequentially adjacent (i.e., contiguous) 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 located sequentially on the disk.
You have the following information about $R$ and $S$ :

- 100 tuples of R are stored per block on disk. Assume that blocks of R are laid out contiguously on disk.
- $T(R)=360,000$ (number of tuples of $R$ ). The values of attribute A in R range from 1 to 360,000 . Assume that A is a key of R , so each tuple in R has a unique value of A in $[1, \ldots, 360,000]$.
- 5 tuples of S are stored per block on disk.
- $T(S)=1,200,000$ (number of tuples of $S$ ).
- $\mathrm{V}(\mathrm{S}, \mathrm{B})=1200$, i.e., there are 1200 distinct values of attribute B in S. Assume that these values are distributed uniformly in $S$, so each value of $B$ occurs $T(S) / V(S, B)=1000$ times in $S$. Furthermore, assume that these values range from 1 to 1200 . That is, for each value v in $[1, \ldots, 1200]$, there are 1000 tuples in $S$ with $S . B=v$.
- $\mathrm{V}(\mathrm{S}, \mathrm{C})=120,000$, i.e., there are 120,000 distinct values of attribute C in S. Assume that these values are distributed uniformly in $S$, so each value of $C$ occurs $T(S) / V(S, C)=10$ times in $S$. Furthermore, assume that these values range from 1 to 120,000 . That is, for each value v in $[1, \ldots, 120,000]$, there are 10 tuples in $S$ with $S . C=v$.

You want to execute the following query:
SELECT *
FROM R, S
WHERE R.A $=$ S.B AND R.A $=$ S.C
We present you with two indexed-nested-loop-join plans:
Plan 1:
For every block BLK of $R$, retrieved using a scan of $R$
For every tuple r of BLK
Use the index on $B$ for $S$ to retrieve all of the tuples $s$ of $S$ such that s. $B=r . A$
For each of these tuples s, if s.C=r.A, output r.A, r.X, r.Y, s.B, s.C, s.Z
Plan 2:
For every block BLK of $R$, retrieved using a scan of $R$
For every tuple r of BLK
Use the index on $C$ for $S$ to retrieve all of the tuples $s$ of $S$ such that s. $C=r$.A
For each of these tuples s, if s.B=r.A, output r.A, r.X, r.Y, s.B, s.C, s.Z
Note that both plans read $R$ one block at a time, and retrieve all $S$ tuples that join with tuples in the current block of R (using one of the indexes on S ) before reading the next block of R .
a. Analyze each of these plans in terms of their behavior regarding accesses to disk. For each plan compute the number of sequential accesses and the number of random accesses to blocks on disk. Given that random accesses are at least an order of magnitude costlier than sequential accesses, which of the plans performs better?
b. Assume all statistics remain the same except for the number of tuples of S stored per block on disk, which now reduces to 2 (from 5). How does this change your answer to (a)?
c. Let the variable $X$ represent the number of tuples of $S$ stored per block on disk. Assuming all other statistics remain the same as before, what values of X in $[1, \ldots, 10,000]$ will make the worse plan of (a) perform better than the other?
d. Which plan is better if both indexes are non-clustering, and everything else remains as specified originally in the question? Note that now tuples of $S$ that have the same value of attribute $C$ are not stored in contiguous blocks on disk.
e. Which plan is better if both indexes are non-clustering, and $\mathrm{V}(\mathrm{S}, \mathrm{B})=180,000$ ? There are 180,000 distinct values of attribute B in S. Assume that these values range from 1 to 180,000 and are distributed uniformly in $\mathrm{S} . \mathrm{V}(\mathrm{S}, \mathrm{C})=120,000$ as before.
f. Suppose everything remains as specified originally in the question except that values of attribute B come from the domain 1-3,600,000. (That is, the domain is positive integers $1,2,3$ and so on up to 3.6 million.) Assume that the values of attribute B in S are distributed uniformly in this domain, and $\mathrm{V}(\mathrm{S}, \mathrm{B})=1,200,000$. Which plan is better in this scenario?

## Question 3

Points 10
A set of indexes is called a covering index set for a query if the query can be evaluated using these indexes only (i.e, without fetching any data records). For queries Q1 and Q2 below:
(a) Give a minimal covering index set
(b) Give an efficient technique (need not be a query plan; an explanation will suffice) to evaluate the query using your minimal covering index set from (a)
(c) Compute the number of disk blocks read by your technique from (b)

Queries Q1 and Q2 are as follows:

```
Q1: SELECT R.a
    FROM R, S
    WHERE R.a = S.a
Q2: SELECT DISTINCT R.a
    FROM R, S, T
    WHERE R.a > S.a AND S.a >= T.b
```

Note that SQL's DISTINCT operator used in Q2 will eliminate duplicates from Q2's result. DISTINCT is the duplicate-eliminating projection that we considered in a previous homework. DISTINCT is also discussed in Section 6.4.1 of the textbook.

Make the following assumptions about relations $R(a, b), S(a, b)$, and $T(a, b)$ (Note: you may not need all this information to compute the number of disk blocks accessed):

- R.a is the primary key of R, S.a is the primary key of S, and T.a is the primary key of T.
- All relations are clustered.
- $\mathrm{B}(\mathrm{R})=1000, \mathrm{~B}(\mathrm{~S})=10,000$, and $\mathrm{B}(\mathrm{T})=100,000$
- $T(R)=10,000, T(S)=50,000$, and $T(T)=300,000 .(T(T)$ denotes the number of tuples in relation T.)
- There are clustering B-tree indexes on R.a, S.a, and T.a. There are non-clustering B-tree indexes on R.b, S.b, and T.b.
- For simplicity of computation, assume that all indexes contain two levels, with the root node in the first level and some number of leaf nodes in the second level. The indexes on R.a and R.b contain 25 leaf nodes each; the indexes on S.a and S.b contain 250 leaf nodes each; and the indexes on T.a and T.b contain 2500 leaf nodes each.
- Assume that root nodes of all indexes are always in memory so that access to a root node never incurs an I/O.


## Question 4

Points 10
Consider the join of four relations $R 1 \bowtie R 2 \bowtie R 3 \bowtie R 4$. We have not shown the join predicates since they are not relevant to this problem. Consider two plans for joining these relations: one using a left-deep join tree (Figure 1) and one using a right-deep join tree (Figure 2). $X 1, X 2, X 3, X 4$ represent various intermediate relations produced in the plans. All the join operators are tuplebased, nested loop joins. The plans are fully pipelined. Only 4 blocks of memory are available. We have $B(R 1)=B(R 2)=B(R 3)=B(R 4)=1000$ blocks, and $T(R 1)=T(R 2)=T(R 3)=T(R 4)=$ $T(X 1)=T(X 2)=T(X 3)=T(X 4)=10000$ tuples. What is the number of disk I/Os for the left-deep plan and the right-deep plan?


Figure 1: Left-deep plan


Figure 2: Right-deep plan

## Question 5

Points 15
Consider the following query over relations $R_{1}-R_{4}$ :

$$
R_{1} \bowtie R_{2} \bowtie R_{3} \bowtie R_{4}
$$

Suppose there are three possible access methods for each $R_{i}$ and two possible join methods for each join. Assume that all combinations of access and join methods are feasible, and that both join methods are asymmetric (e.g., the two join methods could be Nested-Loop join and Hash join, both of which are asymmetric).

1. [4 Points] How many different left-deep plans are there for this query?
2. [5 Points] How many different bushy plans are there for this query? Note that a plan that is not left-deep or right-deep is bushy.
3. [6 Points] How would your answer to (1) change if there is only one join method, but this join method is symmetric (e.g., the join method could be Sort-Merge join, which is symmetric)? Compute the number of different left-deep plans in this case.

## Question 6

Points 10
(a) Justify or refute the argument made regarding hash joins in the footnote on Page 13 of the paper by Yannis Ioannidis (a copy of the paper was handed out in class).
(b) Consider the left-deep join tree (Figure 1) and right-deep join tree (Figure 2). Suppose all joins in these trees are hybrid hash joins, and the joins will execute in a pipelined fashion as much as possible. Under what conditions do you expect the left-deep join tree to be better (i.e., with lower cost) than the right-deep join tree, and vice versa. Assume that M blocks of memory are available for each join operator.

## Question 7

Points 5
As per the paper by Yannis Ioannidis, what are the advantages of randomized algorithms for query optimization over the Selinger algorithm? What would be some disadvantages of a randomized algorithm?

## Question 8

Points 10
Suppose the values of an attribute A in a table R are the following: $19,17,7,17,18,19,2,20,18,4$, $1,18,15,18,19,3,9,17,17,19,1,20,18,5,17$
(a) Draw a equi-width histogram with 5 buckets for attribute A.
(a) Draw a equi-height histogram with 5 buckets for attribute A.
(a) Draw a serial histogram with 5 buckets for attribute A.
(a) Draw an end-biased histogram with 5 buckets for attribute A.

