CPS 216-Spring 2003
Advanced Database Systems

Quiz One & Review Questions

Sample Solution

Instructor: Prof. Jun Yang
TA: Junyi Xie
Department of Computer Science
Duke University
{*junyang, junyi*}*@cs.duke.edu*

March 30, 2003

Problem 1 EXTERNAL SORTING

EXTERNAL SORTING
Suppose that you have a file with 10,000 pages and that you have three buffer pages. Answer the following questions for each of these scenarios, assuming that our most general external sorting algorithm is used: (a) A file with 10,000 pages and three available buffer pages. (b) A file with 20,000 pages and five available buffer pages. (c) A file with 2,000,000 pages and 17 available buffer pages.

- 1. How many runs will you produce in the first pass?
- 2. How many passes will it take to sort the file completely?
- 3. What is the total I/O cost of sorting the file?
- 4. How many buffer pages do you need to sort the file completely in just two passes?

Answer The answer to each question is given below.
1. In the first pass (Pass 0), $\lceil \frac{N}{B} \rceil$ runs of B pages each are produced, where N is the number of file pages and B is the number of available buffer pages:
(a) $\lceil \frac{10000}{3} \rceil$ = 3334 sorted runs.
(b)$\lceil \frac{20000}{5} \rceil$ = 4000 sorted runs.
(c)$\lceil \frac{2{,}000{,}000}{7} \rceil$ = 117648 sorted runs.
2. The number of passes required to sort the file completely, including the initial sorting pass, is $1 + \lceil \log_{B-1} K \rceil$ where $K = \lceil \frac{N}{B} \rceil$ is the number of runs produced by Pass 0.
(a) 13 passes. (b) 7 passes. (c) 6 passes.
3. Since each page is read and written once per pass, the total number of page IOs for sorting the file is $2 \times N \times (NumberOfPasses)$
(a) $2 * 10000 * 13 = 260000$. (b)$2 * 20000 * 7 = 280000$. (c)$2 * 2000000 * 6 = 24000000$.
4. In Pass 0, $\lceil \frac{N}{B} \rceil$ runs are produced. In Pass 1, we must be able to merge this many runs; i.e., $B - 1 \geq \lceil \frac{N}{B} \rceil$. This implies that B must at least be large enough to satisfy $B - 1 \geq \lceil \frac{N}{B} \rceil$; this can be used to guess at B, and the guess must

be validated by checking the first inequality. Thus:
(a) With 10000 pages in the file, B = 101 satisfies both inequalities, B = 100 does not, so we need 101 buffer pages. (b)142. (c) 1415.

Problem 2 INDEX

Consider a relation R(a,b,c,d,e) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with dense secondary indexes. Assume that R:a is a candidate key for R, with values lying in the range 0 to 4,999,999, and that R is stored in R:a order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:

- Access the sorted file for R directly.
- Use a (clustered) B+ tree index on attribute R.a.
- Use a linear hashed index on attribute R.a.

Queries

- $\sigma_{a<50000}(R)$
- $\sigma_{a=50000}(R)$
- $\sigma_{a>50000 \wedge a<50010}(R)$
- $\sigma_{a\neq 50000}(R)$

Answer The answer to each question is given below.
1. $\sigma_{a<50000}(R)$ - For this selection, the choice of accessing the sorted file is slightly superior in cost to using the clustered B+ tree index simply because of the lookup cost required on the B+ tree. 2.$\sigma_{a=50000}(R)$ - A linear hashed index should be cheapest here. 3. $\sigma_{50000a<50010}(R)$ A B+ tree should be the cheapest of the three. 4. $\sigma_{\neq 50000}(R)$ - Since the selection will require a scan of the available entries, and we're starting at the beginning of the sorted index, the sorted file should be slightly more cost-effective, again because of the lookup time.

Review Questions

Briefly answer the following questions:

- 1. Consider the three basic techniques, iteration, indexing, and partitioning, and the relational algebra operators selection, projection, and join. For each technique operator pair, describe an algorithm based on the technique for evaluating the operator.
- 2. Define the term most selective access path for a query.
- 3. Describe conjunctive normal form, and explain why it is important in the context of relational query evaluation.
- 4. When does a general selection condition match an index? What is a primary term in a selection condition with respect to a given index?
- 5. How does hybrid hash join improve upon the basic hash join algorithm?
- 6. Discuss the pros and cons of hash join, sort-merge join, and block nested loops join.
- 7. If the join condition is not equality, can you use sort-merge join? Can you use hash join? Can you use index nested loops join? Can you use block nested loops join?
- 8. Describe how to evaluate a grouping query with aggregation operator MAX using a sorting-based approach.
- 9. Suppose that you are building a DBMS and want to add a new aggregate operator called SECOND LARGEST, which is a variation of the MAX operator. Describe how you would implement it.

- 10. Give an example of how buffer replacement policies can affect the performance of a join algorithm.

Answer

1. (a) iterationselection Scan the entire collection, checking the condition on each tuple, and adding the tuple to the result if the condition is satisfied.
(b) indexingselection If the selection is equality and a B+ or hash index exists on the field condition, we can retrieve relevant tuples by finding them in the index and then locating them on disk.
(c) partitioning - selection Do a binary search on sorted data to find the first tuple that matches the condition. To retrieve the remaining entries, we simple scan the collection starting at the first tuple we found.
(d) iterationprojection Scan the entire relation, and eliminate unwanted at- tributes in the result.
(e) indexingprojection If a multi-attribute B+ tree index exists for all of the projection attributes, then one needs to only look at the leaves of the B+.
(f) partitioningprojection To eliminate duplicates when doing a projection, one can simply project out the unwanted attributes and hash a combination of the remaining attributes so duplicates can be easily detected.
(g) iterationjoin To join two relations, one takes the first attribute in the first relation and scans the entire second relation to find tuples that match the join condition. Once the first attribute has compared to all tuples in the second relation, the second attribute from the first relation is compared to all tuples in the second relation, and so on.
(h) indexingjoin When an index is available, joining two relations can be more e?ient. Say there are two relations A and B, and there is a secondary index on the join condition over relation A. The join works as follows: for each tuple in B, we lookup the join attribute in the index over relation A to see if there is a match. If there is a match, we store the tuple, otherwise we move to the next tuple in relation B.
(i) partitioningjoin One can join using partitioning by using hash join variant or a sort-merge join. For example, if there is a sort merge join, we sort both relations on the the join condition. Next, we scan both relations and identify matches. After sorting, this requires only a single scan over each relation.

2. The most selective access path is the query access path that retrieves the fewest pages during query evaluation. This is the most e?ient way to gather the query's results.

3. Conjunctive normal form is important in query evaluation because often indexes exist over some subset of conjuncts in a CNF expression. Since conjunct order does not matter in CNF expressions, often indexes can be used to increase the selectivity of operators by doing a selection over two, three, or more conjuncts using a single multi-attribute index.

4. An index matches a selection condition if the index can be used to retrieve just the tuples that satisfy the condition. A primary term in a selection condition is a conjunct that matches an index (i.e. can be used by the index).

5. Hybrid hash join improves performance by comparing the first hash buckets during the partitioning phase rather than saving it for the probing phase. This saves us the cost of writing and reading the first partition to disk.

6. Hash join provides excellent performance for equality joins, and can be tuned to require very few extra disk accesses beyond a one-time scan (provided enough memory is available). However, hash join is worthless for non-equality joins. Sort-merge joins are suitable when there is either an equality or non-equality based join condition. Sort-merge also leaves the results sorted which is often a desired property. Sort-merge join has extra costs when you have to use external sorting (there is not enough memory to do the sort in-memory). Block nested loops is e?ient when one of the relations will fit in memory and you are using an MRU replacement strategy. However, if an index is available, there are better strategies available (but often indexes are not available).

7. If the join condition is not equality, you can use sort-merge join, index nested loops (if you have a range style index such as a B+ tree index or ISAM index), or block nested loops join. Hash joining works best for equality joins and is not suitable otherwise.

8. First we sort all of the tuples based on the GROUP BY attribute. Next we re-sort each group by sorting all elements on the MAX attribute, taking case not to re-sort beyond the group boundaries.

9. The operator SECOND LARGEST can be implemented using sorting. For each group (if there is a GROUP BY clause), we sort the tuples and return the second largest value for the desired attribute. The cost here is the cost of sorting.

10. One example where the buffer replacement strategy affects join performance is the use of LRU and MRU in an simple nested loops join. If the relations don't fit in main memory, then the buffer strategy is critical. Say there are M buffer pages and N are filled by the first relation, and the second relation is of size M-N+P, meaning all of the second relation will fit in the the buffer except P pages. Since we must do repeated scans of the second relation, the replacement policy comes into play. With LRU, whenever we need to find a page it will have been paged out so every page request requires a disk IO. On the other hand, with MRU, we will only need to reread P-1 of the pages in the second relation, since the others will remain in memory.