CHAPTER 15

Query Processing

Practice Exercises

15.1 Assume (for simplicity in this exercise) that only one tuple fits in a block and memory holds at most three blocks. Show the runs created on each pass of the sort-merge algorithm when applied to sort the following tuples on the first attribute: (kangaroo, 17), (wallaby, 21), (emu, 1), (wombat, 13), (platypus, 3), (lion, 8), (warthog, 4), (zebra, 11), (meerkat, 6), (hyena, 9), (hornbill, 2), (baboon, 12).

15.2 Consider the bank database of Figure 15.14, where the primary keys are underlined, and the following SQL query:

select *T.branch_name*
from *branch T, branch S*
where *T.assets* > *S.assets* **and** *S.branch_city* = "Brooklyn"

Write an efficient relational-algebra expression that is equivalent to this query. Justify your choice.

---

branch(*branch_name*, *branch_city, assets*)
customer (*customer_name*, *customer_street, customer_city*)
loan (*loan_number*, *branch_name, amount*)
borrower (*customer_name*, *loan_number*)
account (*account_number*, *branch_name, balance*)
depositor (*customer_name*, *account_number*)

Figure 15.14 Bank database.

15.3 Let relations $r_1(A, B, C)$ and $r_2(C, D, E)$ have the following properties: r_1 has 20,000 tuples, r_2 has 45,000 tuples, 25 tuples of r_1 fit on one block, and 30 tuples of r_2 fit on one block. Estimate the number of block transfers and seeks required using each of the following join strategies for $r_1 \bowtie r_2$:

a. Nested-loop join.

b. Block nested-loop join.

c. Merge join.

d. Hash join.

15.4 The indexed nested-loop join algorithm described in Section 15.5.3 can be inefficient if the index is a secondary index and there are multiple tuples with the same value for the join attributes. Why is it inefficient? Describe a way, using sorting, to reduce the cost of retrieving tuples of the inner relation. Under what conditions would this algorithm be more efficient than hybrid merge join?

15.5 Let r and s be relations with no indices, and assume that the relations are not sorted. Assuming infinite memory, what is the lowest-cost way (in terms of I/O operations) to compute $r \bowtie s$? What is the amount of memory required for this algorithm?

15.6 Consider the bank database of Figure 15.14, where the primary keys are underlined. Suppose that a B^+-tree index on *branch_city* is available on relation *branch*, and that no other index is available. List different ways to handle the following selections that involve negation:

a. $\sigma_{\neg(branch\_city < \text{“Brooklyn”})}(branch)$

b. $\sigma_{\neg(branch\_city = \text{“Brooklyn”})}(branch)$

c. $\sigma_{\neg(branch\_city < \text{“Brooklyn”} \vee assets < 5000)}(branch)$

15.7 Write pseudocode for an iterator that implements indexed nested-loop join, where the outer relation is pipelined. Your pseudocode must define the standard iterator functions *open*( ), *next*( ), and *close*( ). Show what state information the iterator must maintain between calls.

15.8 Design sort-based and hash-based algorithms for computing the relational division operation (see Practice Exercise 2.9 for a definition of the division operation).

15.9 What is the effect on the cost of merging runs if the number of buffer blocks per run is increased while overall memory available for buffering runs remains fixed?

15.10 Consider the following extended relational-algebra operators. Describe how to implement each operation using sorting and using hashing.

a. **Semijoin** ($\ltimes_\theta$): The multiset semijoin operator $r \ltimes_\theta s$ is defined as follows: if a tuple r_i appears n times in r, it appears n times in the result of $r \ltimes_\theta$ if there is at least one tuple s_j such that r_i and s_j satisfy predicate θ; otherwise r_i does not appear in the result.

b. **Anti-semijoin** ($\overline{\ltimes}_\theta$): The multiset anti-semijoin operator $r \overline{\ltimes}_\theta s$ is defined as follows: if a tuple r_i appears n times in r, it appears n times in the result of $r \ltimes_\theta$ if there does not exist any tuple s_j in s such that r_i and s_j satisfy predicate θ; otherwise r_i does not appear in the result.

15.11 Suppose a query retrieves only the first K results of an operation and terminates after that. Which choice of demand-driven or producer-driven pipelining (with buffering) would be a good choice for such a query? Explain your answer.

15.12 Current generation CPUs include an *instruction cache*, which caches recently used instructions. A function call then has a significant overhead because the set of instructions being executed changes, resulting in cache misses on the instruction cache.

a. Explain why producer-driven pipelining with buffering is likely to result in a better instruction cache hit rate, as compared to demand-driven pipelining.

b. Explain why modifying demand-driven pipelining by generating multiple results on one call to *next*(), and returning them together, can improve the instruction cache hit rate.

15.13 Suppose you want to find documents that contain at least k of a given set of n keywords. Suppose also you have a keyword index that gives you a (sorted) list of identifiers of documents that contain a specified keyword. Give an efficient algorithm to find the desired set of documents.

15.14 Suggest how a document containing a word (such as "leopard") can be indexed such that it is efficiently retrieved by queries using a more general concept (such as "carnivore" or "mammal"). You can assume that the concept hierarchy is not very deep, so each concept has only a few generalizations (a concept can, however, have a large number of specializations). You can also assume that you are provided with a function that returns the concept for each word in a document. Also suggest how a query using a specialized concept can retrieve documents using a more general concept.

15.15 Explain why the nested-loops join algorithm (see Section 15.5.1) would work poorly on a database stored in a column-oriented manner. Describe an alternative algorithm that would work better, and explain why your solution is better.

15.16 Consider the following queries. For each query, indicate if column-oriented storage is likely to be beneficial or not, and explain why.

a. Fetch ID, *name* and *dept_name* of the student with ID 12345.

b. Group the *takes* relation by *year* and *course_id*, and find the total number of students for each (*year*, *course_id*) combination.