CHAPTER 14

Indexing

Practice Exercises

14.1 Indices speed query processing, but it is usually a bad idea to create indices on every attribute, and every combination of attributes, that are potential search keys. Explain why.

14.2 Is it possible in general to have two clustering indices on the same relation for different search keys? Explain your answer.

14.3 Construct a B^+-tree for the following set of key values:

(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)

Assume that the tree is initially empty and values are added in ascending order. Construct B^+-trees for the cases where the number of pointers that will fit in one node is as follows:

a. Four

b. Six

c. Eight

14.4 For each B^+-tree of Exercise 14.3, show the form of the tree after each of the following series of operations:

a. Insert 9.

b. Insert 10.

c. Insert 8.

d. Delete 23.

e. Delete 19.

14.5 Consider the modified redistribution scheme for B$^+$-trees described on page 651. What is the expected height of the tree as a function of n?

14.6 Give pseudocode for a B$^+$-tree function findRangeIterator(), which is like the function findRange(), except that it returns an iterator object, as described in Section 14.3.2. Also give pseudocode for the iterator class, including the variables in the iterator object, and the next() method.

14.7 What would the occupancy of each leaf node of a B$^+$-tree be if index entries were inserted in sorted order? Explain why.

14.8 Suppose you have a relation r with n_r tuples on which a secondary B$^+$-tree is to be constructed.

a. Give a formula for the cost of building the B$^+$-tree index by inserting one record at a time. Assume each block will hold an average of f entries and that all levels of the tree above the leaf are in memory.

b. Assuming a random disk access takes 10 milliseconds, what is the cost of index construction on a relation with 10 million records?

c. Write pseudocode for bottom-up construction of a B$^+$-tree, which was outlined in Section 14.4.4. You can assume that a function to efficiently sort a large file is available.

14.9 The leaf nodes of a B$^+$-tree file organization may lose sequentiality after a sequence of inserts.

a. Explain why sequentiality may be lost.

b. To minimize the number of seeks in a sequential scan, many databases allocate leaf pages in extents of n blocks, for some reasonably large n. When the first leaf of a B$^+$-tree is allocated, only one block of an n-block unit is used, and the remaining pages are free. If a page splits, and its n-block unit has a free page, that space is used for the new page. If the n-block unit is full, another n-block unit is allocated, and the first $n/2$ leaf pages are placed in one n-block unit and the remaining one in the second n-block unit. For simplicity, assume that there are no delete operations.

i. What is the worst-case occupancy of allocated space, assuming no delete operations, after the first n-block unit is full?

ii. Is it possible that leaf nodes allocated to an n-node block unit are not consecutive, that is, is it possible that two leaf nodes are allocated to one n-node block, but another leaf node in between the two is allocated to a different n-node block?

iii. Under the reasonable assumption that buffer space is sufficient to store an n-page block, how many seeks would be required for a leaf-level scan of the B$^+$-tree, in the worst case? Compare this number with the worst case if leaf pages are allocated a block at a time.

iv. The technique of redistributing values to siblings to improve space utilization is likely to be more efficient when used with the preceding allocation scheme for leaf blocks. Explain why.

14.10 Suppose you are given a database schema and some queries that are executed frequently. How would you use the above information to decide what indices to create?

14.11 In write-optimized trees such as the LSM tree or the stepped-merge index, entries in one level are merged into the next level only when the level is full. Suggest how this policy can be changed to improve read performance during periods when there are many reads but no updates.

14.12 What trade offs do buffer trees pose as compared to LSM trees?

14.13 Consider the *instructor* relation shown in Figure 14.1.

a. Construct a bitmap index on the attribute *salary*, dividing *salary* values into four ranges: below 50,000, 50,000 to below 60,000, 60,000 to below 70,000, and 70,000 and above.

b. Consider a query that requests all instructors in the Finance department with salary of 80,000 or more. Outline the steps in answering the query, and show the final and intermediate bitmaps constructed to answer the query.

14.14 Suppose you have a relation containing the x, y coordinates and names of restaurants. Suppose also that the only queries that will be asked are of the following form: The query specifies a point and asks if there is a restaurant exactly at that point. Which type of index would be preferable, R-tree or B-tree? Why?

14.15 Suppose you have a spatial database that supports region queries with circular regions, but not nearest-neighbor queries. Describe an algorithm to find the nearest neighbor by making use of multiple region queries.