Practice Exercises

10.2 Flash storage: 10

a. How is the flash translation table, which is used to map logical page numbers to physical page numbers, created in memory?

b. Suppose you have a 64 gigabyte flash storage system, with a 4096 byte page size. How big would the flash translation table be, assuming each page has a 32 bit address, and the table is stored as an array.

c. Suggest how to reduce the size of the translation table if very often long ranges of consecutive logical page numbers are mapped to consecutive physical page numbers.

Disk 1	Disk 2	Disk 3	Disk 4
B_1	B_2	B_3	B_4
P_1	B_5	B_6	B_7
B_8	P_2	B_9	B_{10}
⋮	⋮	⋮	⋮

Figure 10.18 Data and parity block arrangement.

10.3 A power failure that occurs while a disk block is being written could result in the block being only partially written. Assume that partially written blocks can be detected. An atomic block write is one where either the disk block is fully written or nothing is written (i.e., there are no partial writes). Suggest schemes for getting the effect of atomic block writes with the following RAID schemes. Your schemes should involve work on recovery from failure.

10

a. RAID level 1 (mirroring)

b. RAID level 5 (block interleaved, distributed parity)

10.6 Consider the relations *section* and *takes*. Give an example instance of these two relations, with three sections, each of which has five students. Give a file structure of these relations that uses multitable clustering. 5

10.7 Consider the following bitmap technique for tracking free space in a file. For each block in the file, two bits are maintained in the bitmap. If the block is between 0 and 30 percent full the bits are 00, between 30 and 60 percent the bits are 01, between 60 and 90 percent the bits are 10, and above 90 percent the bits are 11. Such bitmaps can be kept in memory even for quite large files. 5

a. Describe how to keep the bitmap up to date on record insertions and deletions.

b. Outline the benefit of the bitmap technique over free lists in searching for free space and in updating free space information.

10.8 It is important to be able to quickly find out if a block is present in the buffer, and if so where in the buffer it resides. Given that database buffer sizes are very large, what (in-memory) data structure would you use for the above task? 5

10.9 Give an example of a relational-algebra expression and a query-processing strategy in each of the following situations: 5

a. MRU is preferable to LRU.

b. LRU is preferable to MRU.

Exercises

10.10 List the physical storage media available on the computers you use routinely. Give the speed with which data can be accessed on each medium.

10.11 How does the remapping of bad sectors by disk controllers affect data-retrieval rates? 5

10.15 Explain why the allocation of records to blocks affects database-system performance significantly.

10.16 If possible, determine the bu5er-management strategy used by the operating system running on your local computer system and what mechanisms it provides to control replacement of pages. Discuss how the control on replacement that it provides would be useful for the implementation of database systems.

10.17 List two advantages 5 and two disadvantages of each of the following strategies for storing a relational database:

a. Store each relation in one file. 5

b. Store multiple relations (perhaps even the entire database) in one file.

10.18 In the sequential file organization, why is an overflow *block* used even if there is, at the moment, only one overflow record? 5

10.19 Give a normalized version of the *Index_metadata* relation, and explain why using the normalized version would result in worse performance. 5

- We can also use hashing to create secondary indices; such indices are called *hash indices*. For notational convenience, we assume hash file organizations have an implicit hash index on the search key used for hashing.
- Ordered indices such as B$^+$-trees and hash indices can be used for selections based on equality conditions involving single attributes. When multiple attributes are involved in a selection condition, we can intersect record identifiers retrieved from multiple indices.
- Bitmap indices provide a very compact representation for indexing attributes with very few distinct values. Intersection operations are extremely fast on bitmaps, making them ideal for supporting queries on multiple attributes.

Review Terms

- Access types
- Access time
- Insertion time
- Deletion time
- Space overhead
- Ordered index
- Clustering index
- Primary index
- Nonclustering index
- Secondary index
- Index-sequential file
- Index entry/record
- Dense index
- Sparse index
- Multilevel index
- Composite key
- Sequential scan
- B$^+$-tree index
- Leaf node
- Nonleaf node
- Balanced tree
- Range query
- Node split
- Node coalesce
- Nonunique search key
- B$^+$-tree file organization
- Bulk load
- Bottom-up B$^+$-tree construction
- B-tree index
- Static hashing
- Hash file organization
- Hash index
- Bucket
- Hash function
- Bucket overflow
- Skew
- Closed hashing
- Dynamic hashing
- Extendable hashing
- Multiple-key access
- Indices on multiple keys
- Bitmap index
- Bitmap operations
    - Intersection
    - Union
    - Complement
    - Existence bitmap

Practice Exercises

11.1 Indices speed query processing, but it is usually a bad idea to create indices on every attribute, and every combinations of attributes, that is a potential search keys. Explain why. 10

11.2 Is it possible in general to have two clustering indices on the same relation for different search keys? Explain your answer. 10

11.3 Construct a B$^+$-tree for the following set of key values: 10

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

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

11.4 For each B$^+$-tree of Practice Exercise 11.3, show the form of the tree after each of the following series of operations: 10

a. Insert 9.

b. Insert 10.

c. Insert 8.

d. Delete 23.

e. Delete 19.

11.5 Consider the modified redistribution scheme for B$^+$-trees described on page 501. What is the expected height of the tree as a function of n? 10

11.6 Suppose that we are using extendable hashing on a file that contains records with the following search-key values: 10

$$2, 3, 5, 7, 11, 17, 19, 23, 29, 31$$

Show the extendable hash structure for this file if the hash function is $h(x) = x \bmod 8$ and buckets can hold three records.

11.7 Show how the extendable hash structure of Practice Exercise 11.6 changes as the result of each of the following steps: 5

a. Delete 11.

b. Delete 31.

c. Insert 1.

d. Insert 15.

11.12 What would the occupancy of each leaf node of a B$^+$-tree be, if index entries are inserted in sorted order? Explain why. 5

11.13 Suppose you have a relation r with n_r tuples on which a secondary B$^+$-tree is to be constructed. 10

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 11.4.4. You can assume that a function to efficiently sort a large file is available.

Exercises

11.15 When is it preferable to use a dense index rather than a sparse index? Explain your answer. 5

11.16 What is the difference between a clustering index and a secondary index? 5

11.20 What are the causes of bucket overflow in a hash file organization? What can be done to reduce the occurrence of bucket overflows? 5

11.21 Why is a hash structure not the best choice for a search key on which range queries are likely? 5

11.22 Suppose there is a relation $r(A, B, C)$, with a B$^+$-tree index with search key (A, B). 10

a. What is the worst-case cost of finding records satisfying $10 < A < 50$ using this index, in terms of the number of records retrieved n_1 and the height h of the tree?

b. What is the worst-case cost of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$ using this index, in terms of the number of records n_2 that satisfy this selection, as well as n_1 and h defined above?

c. Under what conditions on n_1 and n_2 would the index be an efficient way of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$?

11.23 Suppose you have to create a B$^+$-tree index on a large number of names, where the maximum size of a name may be quite large (say 40 characters) and the average name is itself large (say 10 characters). Explain how prefix compression can be used to maximize the average fanout of nonleaf nodes. 10

11.24 Suppose a relation is stored in a B$^+$-tree file organization. Suppose secondary indices stored record identifiers that are pointers to records on disk. 10

a. What would be the effect on the secondary indices if a node split happens in the file organization?

b. What would be the cost of updating all affected records in a secondary index?

c. How does using the search key of the file organization as a logical record identifier solve this problem?

d. What is the extra cost due to the use of such logical record identifiers?

11.25 Show how to compute existence bitmaps from other bitmaps. Make sure that your technique works even in the presence of null values, by using a bitmap for the value *null*. 10

11.26 How does data encryption affect index schemes? In particular, how might it affect schemes that attempt to store data in sorted order? 10