Study Notes on File Organization

File organization is a crucial aspect of database systems that significantly impacts data storage and retrieval efficiency. By selecting appropriate file organization techniques and indexing structures, database administrators and developers can optimize data access, minimize disk I/O operations, and enhance overall system performance. Understanding the characteristics and trade-offs of different file organization methods is essential for designing efficient database systems.

Here we have provided study notes on File Organisation to help you in the GATE Computer Science Engineering preparation.

File Organisation

The database is stored as a collection of files. Each file is a sequence of records. A record is a sequence of fields. Data is usually stored in the form of records. Records usually describe entities and their attributes. e.g., an employee record represents an employee entity and each field value in the record specifies some attributes of that employee, such as Name, Birth-date, Salary or Supervisor.

If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records. If different records in the file have different sizes, the file is said to be made up of variable length records.

Spanned versus Unspanned Records

- The records of a file must be allocated to disk blocks because a block is the unit of data transfer between disk and memory. When the block size is larger than the record size, each block will contain numerous records, although some of the files may have unusually large records that cannot fit in one block.
- Suppose that block size is B bytes. For a file of fixed length records of size R bytes, with B ≥ R, then we can fit $\text{bfr} = \left\lfloor \frac{B}{R} \right\rfloor$ records per block. The value of bfr is called the blocking factor.
- In general, R may not divide B exactly, so we have some unused space in each block equal to B – (bfr * R) bytes.

- To utilize this unused space, we can store part of a record on one block and the rest on another. A pointer at the end of the first block points to the block containing the remainder of the record. This organisation is called spanned.
- If records are not allowed to cross block boundaries, the organisation is called unspanned.

Allocating File Blocks on Disk: There are several standard techniques for allocating the blocks of a file on disk

- **Contiguous Allocation:** The file blocks are allocated to consecutive disk blocks. This makes reading the whole file very fast.
- **Linked Allocation:** In this, each file contains a pointer to the next file block.
- **Indexed Allocation:** Where one or more index blocks contain pointers to the actual file blocks.

Files of Unordered Records (Heap Files)

In the simplest type of organization records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. Such an organisation is called a heap or pile file.

This organisation is often used with additional access paths, such as the secondary indexes.

In this type of organisation, inserting a new record is very efficient. Linear search is used to search a record.

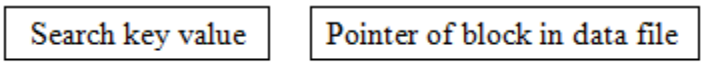
Files of Ordered Records (Sorted Files)

We can physically order the records of a file on disk based on the values of one of their fields called the ordering field. This leads to an ordered or sequential file. If the ordering field is also a key field of the file, a field guaranteed to have a unique value in each record, then the field is called the ordering key for the file. Binary searching is used to search a record.

Indexing Structures for Files

Indexing mechanisms are used to optimize certain accesses to data (records) managed in files. e.g., the author catalog in a library is a type of index. Search key (definition) attribute or combination of attributes used to look-up records in a file.

An index file consists of records (called index entries) of the form.



Indexing structures for files

Index files are typically much smaller than the original file because only the values for search key and pointer are stored. The most prevalent types of indexes are based on ordered files (single-level indexes) and tree data structures (multilevel indexes).

Types of Single Level Ordered Indexes

In an ordered index file, index entries are stored sorted by the search key value. There are several types of ordered Indexes

Primary Index

A primary index is an ordered file whose records are of fixed length with two fields. The first field is of the same data type as the ordering key field called the primary key of the data file and the second field is a pointer to a disk block (a block address).

- There is one index entry in the index file for each block in the data file.
- Indexes can also be characterized as dense or sparse.
- **Dense index** A dense index has an index entry for every search key value in the data file.
- **Sparse index** A sparse index (non-dense), on the other hand has index entries for only some of the search values, specifically one for each block.
- A primary index is a non-dense (sparse) index, since it includes an entry for each disk block of the data file rather than for every search value.
- It is built over the ordered and key field of the given table.

Clustering Index

If file records are physically ordered on a non-key field which does not have a distinct value for each record that field is called the clustering field. We can create a different type of

index, called a clustering index, to speed up retrieval of records that have the same value for the clustering field.

- A clustering index is also an ordered file with two fields. The first field is of the same type as the clustering field of the data file.
- The record field in the clustering index is a block pointer.
- A clustering index is another example of a non-dense index.
- It is possible to have only one index for a file either primary index or clustering index.

Secondary Index

A secondary index provides a secondary means of accessing a file for which some primary access already exists. The secondary index may be on a field which is a candidate key and has a unique value in every record or a non-key with duplicate values. The index is an ordered file with two fields. The first field is of the same data type as some non-ordering field of the data file that is an indexing field. The second field is either a block pointer or a record pointer. A secondary index usually needs more storage space and longer search time than does a primary index.

Multilevel Indexes

The idea behind a multilevel index is to reduce the part of the index. A multilevel index considers the index file, which will be referred now as the first (or base) level of a multilevel index. Therefore, we can create a primary index for the first level; this index to the first level is called the second level of the multilevel index and so on.

The basic idea is to create the index file for the previously created index file so that cost of indexing can be further reduced and memory can be used efficiently.

Dynamic Multilevel Indexes Using B-Trees and B^+ -Trees

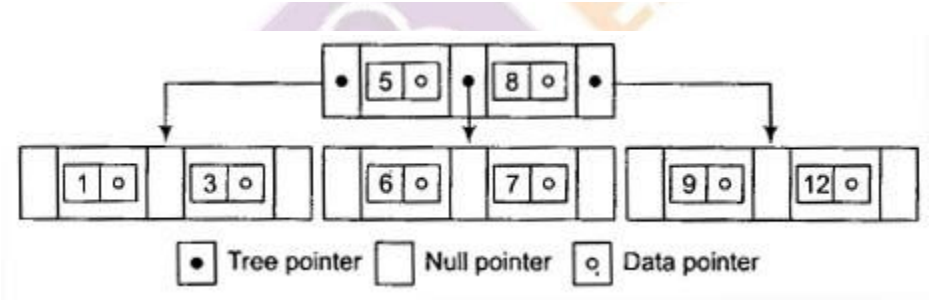
There are two multilevel indexes:

B-Trees

- When data volume is large and does not fit in memory, an extension of the binary search tree to disk based environment is the B-tree.

- In fact, since the B-tree is always balanced (all leaf nodes appear at the same level), it is an extension of the balanced binary search tree.
- The problem which the B-tree aims to solve is given a large collection of objects, each having a key and a value, design a disk based index structure which efficiently supports query and update.
- A B-tree of order p, when used as an access structure on a key field to search for records in a data file, can be defined as follows
    1. Each internal node in the B-tree is of the form

       $<P_1, <K_1, P_{r_1}>, P_2, <K_2, P_{r_2}>, \ldots. <K_{q-1}, P_{r_{q-1}}> P_q>$ where, q ≤ p Each P_i is a tree pointer to another node in the B-tree. Each P_{r_i} is a data pointer to the record whose search key field value is equal to K_j.
    2. Within each node, $K_1 < K_2 < \ldots. < K_{q-1}$
    3. Each node has at most p tree pointers.
    4. Each node, except the root and leaf nodes, has at least [(p/2)] tree pointers.
    5. A node within q tree pointers q ≤ p, has q – 1 search key field values (and hence has q –1 data pointers). e.g., A B-tree of order p = 3. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

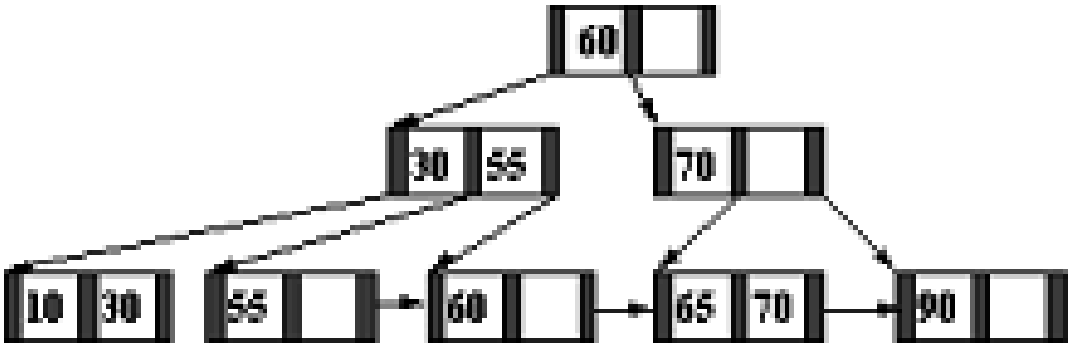


B⁺ Trees

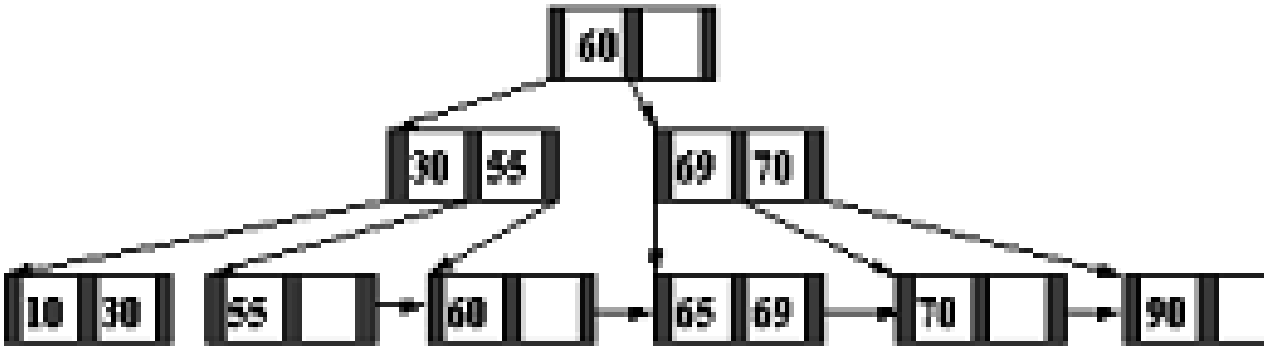
- It is the variation of the B-tree data structure.
- In a B-tree, every value of the search field appears once at some level in the tree, along with a data pointer. In a B⁺-tree, data pointers are stored only at the leaf nodes of the tree. Hence, the structure of the leaf nodes differs from the structure of internal nodes.
- The pointers in the internal nodes are tree pointers to blocks that are tree nodes whereas the pointers in leaf nodes are data pointers.
- B⁺ **Tree's Structure:** The structure of the B⁺-tree of order p is as follows

1. Each internal node is of the form < P_I, K_1, P_2, K_2, …. , P_{q-1}, K_{q-1}, P_q> where, q ≤ P and each P_i is a tree pointer.
2. Within each internal node, $K_1 < K_2 < K_3 …. < K_{q-1}$.
3. Each internal node has at most p tree pointers and except root, has atleast [(p/ 2)] tree pointers.
4. The root node has atleast two tree pointers, if it is an internal node.
5. Each leaf node is of the form:

   $<<K_1 P_{r_1}>, <K_2, P_{r_2}>, ..., <K_{q-1}, P_{r_{q-1}}>, P_{next}>$ where, q ≤ p, each P_{r_i} is a data pointer and P_{next} points to the next leaf node of the B^+-trees.



Add 10



Add 69

Difference between B/B^+ tree

- The structure of the leaf node is different.
- In the B+ tree, all the keys are present at the leaf level, but in the B tree, Some of the keys are present at leaf level.
- B+ tree is preferred for accessing sequential records or sequential data.
- B tree is preferred for random access of data items.

- Bulk Loading is allowed with the B+ tree since B+ tree allows all the keys to appear at leaf level.

Thanks!

BYJU'S Exam Prep - The Most Comprehensive App!