Files and Their Organization

Data Hierarchy

Every file contains data which can be organized in a hierarchy to present a systematic organization. The data hierarchy includes data items such as fields, records, files, and database.

- A **data field** is an elementary unit that stores a single fact. A data field is usually characterized by its type and size.

- A **record** is a collection of related data fields which is seen as a single unit from the application point of view.

- A **file** is a collection of related records.

- A **directory** stores information of related files. A directory organizes information so that users can find it easily.

File Attributes

Every file in a computer system is stored in a directory. Each file has a list of attributes associated with it that gives the operating system and the application software information about the file and how it is intended to be used.

File name: It is a string of characters that stores the name of a file. File naming conventions vary from one operating system to the other.

File Position: It is a pointer that points to the position at which the next read/write operation will be performed.

File structure: It indicates whether the file is a text file or a binary file. In the text file, the numbers (interger or floating point) are stored as a string of characters. A binary file, stores numbers in the same way as they are represented in the main memory.

File Access Method: It indicates whether the records in a file can be accessed sequentially or randomly. In sequential access mode, records are read one by one. In random access, records can be accessed in any order.

Attributes Flag: A file can have six additional attributes attached to it. These attributes are usually stored in a single byte, with each bit representing a specific attribute. If a particular bit is set to '1' then this means that the corresponding attribute is turned on. Note that the directory is treated as a special file in the operating system. So, all these attributes are applicable to files as well as to directories.

Read-only: A file marked as read-only cannot be deleted or modified.

Hidden: A file marked as hidden is not displayed in the directory listing.

System: A file marked as a system file indicates that it is an important file used by the operating system and should not be altered or removed from the disk.

Volume Label: Every disk volume is assigned a label for identification. The label can be assigned at the time of formatting the disk or later through various tools.

Directory: In directory listing, the files and sub-directories of the current directory are differentiated by a directory-bit. This means that the files that have the directory-bit turned on are actually sub-directories containing one or more files.

Archieve: The archieve bit is used as a communication link between programs that modify files and those that are used for backing up files. Most backup programs allow the user to do an incremental backup. Incremental backup selects only those files for backup which have been modified since the last backup.

When the backup program takes the backup of a file, or in other words, when the program archieves the file, it clears the archieve bit (set it to zero). Seubsequently, if any program modifies the file, it turns on the archieve bit (sets to 1). Thus, whever the back program is run, it checks the archive bit to know whether the file has been modified since its last run. The backup program will archive only those file which were modified.

Text and Binary Files

A **text file**, also known as a flat file or an ASCII file, is structured as a sequence of lines of alphabet, numerals, special characters, etc. However, the data in a text file is often denoted by placing a special character, called an end-of-file marker, after the last line in the text file.

A **binary file** contains any type of data encoded in binary form for computer storage and processing purposes. A binary file can contain text that is not broken up into lines. A binary file stores data in a format that is similar to the format in which the data is stored in the main memory.

Basic File Operations

Creating a File: A file is created by specifying its name and mode. Then the file is opened for writing records that are read from an input device. Once all the records have been written into the file, the file is closed. The file is now available for future read/write operations by any program that has been designed to use it in some way or the other.

Updating a File: Updating a file means the content of the file to reflect a current picture of reality. A file can be updated in the following ways:

- Inserting a new record in the file.
- Deleting an existing record.
- Modifying an existing record.

Retrieving from a File: It means extracting useful data from a given file. Information can be retrieved from a file either from an inquiry or for report generation. An inquiry for some data retrieves low volume of data, while report generation may retrieve a large volume of data from the file.

Maintaining a File: It involves restructuring or re-organizing the file to improve the performance of the programs that access the file. Restructuring a file keeps the file organization unchanged and changes only the structural aspects of the file. File re-organization may involve changing the entire organization of the file.

File Organization

The main inssue in file management is the way in which the records are organized inside the file because it has a significant effect on the system performance. Organization of records means the logical arrangement of records in the file and not the physical layout of the file as stored on a storage media. The following considerations should be kept in mind before selecting an appropriate file organiation method:

- Rapid access to one or more records
- Ease of inserting/updating/deleting one or more records without disrupting the speed of accessing record(s)
- Efficient storage of records
- Using redundancy to ensure data integrity

Sequential Organization

A sequentially organized file stores the records in the order in which they were entered. That is, the first record that was entered is written as the first record in the file, the second record entered is written as the second record in the file, and so on. As a result, new records are added only at the end of the file.

Sequential files can be read only sequentially, starting with the first record in the file. Sequential file organization is the most basic way to organize a large collection of records in a file.

Once we store the records in a file, we cannot make any changes to the record. We cannot even delete the records from a sequential file. In case we need to delete or update one or more records, we have to replace the records by creating a new file.

In sequential file organization, all records have the same size and the same field format, and every field has a fixed size. The records are sorted based on the value of one field or a combination of two or more fields. This field is known as the **key**. Each key uniquely identifies a record in a file. Thus every record has a different value for the key field.

Relative File Organization

Relative file organization provides an effective way to access individual records directly. In a relative file organization, records are ordered by their **relative key**. It means the record number represents the location of the record relative to the beginning of the file. The record numbers range from 0 to $n-1$, where n is the number of records in the file. The record in a relative file are of fixed length.

In relative files, records are organized in ascending **relative record number**. A relative file can be thought of as a single dimension table stored on a disk, in which the relative record number is the index into the table. Relative files can be used for both random as well as sequential access. For sequential access, records are simply read one after another.

Relative files procide support for only one key, that is the relative record number. This key must be numeric and must take a value between 0 and the current highest relative record number -1. This means that enough space must be allocated for the file to contain the records with relative record numbers between 0 and the highest record number -1.

Relative file organization provides random access by directly jumping to the record which has to be accessed. If the records are fixed length and we know the base address of the file and the length of the record, then any record i can be accessed using the following formula:

$$\texttt{Address of } i^{th} \texttt{ record = base\_address + } (i-1) \texttt{ * record\_length}$$

Note that the base address of the file refers to the starting address of the file.

Indexed Sequential File Organization

Indexed sequential file organization stores data for fast retrieval. The records in an index sequential file are of fixed length and every record is uniquely identified by a key field. We maintain a table known as the **index table** which stores the record number and the address of all the records. That is for every file, we hve an index table. This type of file organization is called index sequential file organization because physically the records may be stored anywhere, but the index table stores the address of those records.

The ith entry in the index table points to the ith record of the file. Initially, when the file is created, each entry in the index table contains `NULL`. When the ith record of the file is written, free space is obtained from the free space manager and its address is stored in the ith location of the index table.

An indexed sequential file uses the concept of both sequential as well as relative files. While the index table is read sequentially to find the address of the desired record, a direct access is made to the address of the specified record in order to access it randomly.

Indexed sequential files perform well in situations where sequential access as well as random access is made to the data. Indexed sequential files can be stored only on devices that support random access.

Indexing

For a particular situation at hand, we analyze the indexing technique based on factors such as access type, access time, insertion time, deletion time, and space overhead involved. There are two kinds of indices:

- **Ordered indices** that are sorted based on one or more key values
- **Hash indices** that are based on the values generated by applying a hash function.

Ordered Indices

Indices are used to provide fast random access to records. A file may have multiple indices based on different key fields. An index of a file may be a primary index or a secondary index.

Primary Index

In a sequentailly ordered file, the index whose search key specifies the sequential order of the file is determined as the primary index. Indexed sequentil files are a common example where a primary index is associated with the file.

Secondary Index

An index whose search key specifies an order different from the sequential order of the file is called as the secondary index. Secondary indices are used to improve the performance of queries on non-primary keys.

Dense and Sparse Indices

In a dense index, the index table stores the address of every record in the file. However, in a sparse index, the index table stores the address of only some of the records in the file. Although sparse indices are easy to fit in the main memory, a dense index would be more efficient to use than a sparse index if it fits in memory.

Note that the records need not be stored in consecutive memory locations. The pointer to the next record stores the address of the next record.

By looking at the dense index, it can be concluded directly whether the record exists in the file or not. That is not the case in a sparse index. In a sparse index, to locate a record, we first find an entry in the index table with the largest search key value that is either less than or equal to the search key value of the desired record. Then, we start at that record pointed to by that entry in the index table and then proceed searching the record using the sequential pointers in the file, until the desired record is obtained.

We see that sparse index takes more time to find a record with the given key. Dense indices are faster to use, while sparse indices require less space and impose less maintence for insertions and deletions.

Cylinder Surface Indexing

Cylinder surface indexing is a very simple technique used only for the primary key index of a sequentially ordered file. In a sequentially ordered file, the records are stored sequentially in the increasing order of the primary key. The index file will contain two fields—cylinder index and several surface indices. Generally, there are multiple cylinders, and each cyliner has multiple surfaces. If the file needs m cylinders for storage then the cylinder index will contain m entries. Each cylinder will have an entry corresponding to the largest key value into that cylinder. If the disk has n usable surfaces, then each of the surface indices will have n entries. Therefore, the ith entry in the surface index for cylinder j is the largest key value on the jth track of the ith surface. Hence, the total number of surface index entries is $m \times n$

When a record with a particular key value has to be searched, then the following steps are performed:

- First the cylinder index of the file is read into memory.

- Second, the cylinder index is searched to detemine which cylinder holds the desired record. For this, either the binary search technique can be used or the cylinder index can be made to store an array of pointers to the starting of individual key values. In either case the search will take $O(\log m)$ time.

- After the index is searched, appropriate cylinder is determined.

- Depending on the cylinder, the surface index corresponding to the cylinder is then retrieved from the disk.

- Since the number of surfaces on a disk is very small, the corresponding track is read and searched for the record with the desired key.

- Once the cylinder and the surface are determined, the corresponding track is read and searched for the record with the desired key.

The total number if disk access is three—first, for accessing the cylinder index, second for accessing the surface index, and third for getting the track address. However, if track sizes are very large then it may not be a good idea to read the whole track at once. In such situations, we can include sector addresses. But this would add an extra level of indexing and, therefore, the number of accesses needed to retrieve a record will then become four. In addition to this, when the file extends over several disks, a dsik index will also be added.

The cylinder surface indexing method of maintaining a file and index is referred to as Indexed Sequential Acess Method (ISAM). This technique is the most popular and simplest file organization in use for single key values. But with files that contain multiple keys, it is not possible to use this index organization for the remaining keys.

Multi-level Indices

In real-world applications, we have very large files that may contain millions of records. For such files, a simple indexing technique will not suffice. In such a situation, we use multi-level indices.

Inverted Indices

Inverted files are commonly used in document retreval systems for large textual databases. An inverted file reorganizes the structure of an existing data file in order to provide fast access to all records having one field falling within the set limits.

There are two main variants of inverted indices:

- A record-level inverted index (also known as **inverted file** or **inverted list**) stores a list of references to documents for each word.

- A word-level index (also known as **full inverted** or **inverted list**) in addition to a list of reference to documents for each word also contains the positions of each word within a document. Although this technique needs more time and space, it offers more functionality (like phrase searches).

The inverted file system consists of an index file in addition to a document file (also known as **text file**). It is this index file that contains all the key words which may be used as search terms. For each keyword, an address or reference to each location in the document where that word occurs is stored. There is no restriction on the number of pinters associated with each word.

For efficietly retrieving a word from the index file, the keyswords are sorted in a specific order (usually alphabetically).

However, the main drawback of this structure is that when new words are added to the documents or text files, the whole file must be reorganized. Therefore, a better alternative is to use B-trees.

B-Tree Indices

A databse is defined as a collection of data organized in a fashion that facilitates updating, retrieving, and managing the data (that may include any item, such as names, address, pictures, and numbers).

For a databse to be useful, it must support fast retriebal and storage of data. Since it is impractical to maintain the enire databse in memory, B-trees are used to index the daya in order to provide fast acess.

Searching a value in an un-index and unsorted database containing n key values my take a running time of $O(n)$ in the worst case, but if the same database is index with a B-tree, the search operation wil run in $O(\log n)$ time.

Majority of the database management systems use the B-tree index technique as the default indexing method. This technqiue supersedes other techniques of creating indices, mainly due to its retrieval speed, ease of maintenance, and simplicity.

If a table has a column that has many unique values, then the selectivity of that column is said to be high, B-tree indices are most suitable for highly selective columns, but it causes a sharp increase in the size when the indices contain concatenation of multiple columns.

The B-tree structure has the following advantages

- Since the leaf nodes of a B-tree are at the same depth, retrieval of any record from anywhere in the index takes apporximately the same time.

- B-trees improve the performance of a wide range of queries that either search a value having an exact match or for a value within a specified range.

- B-trees provide fast an efficent algorithms to insert, update, and delete records that maintain the key order.

- B-trees perform as well as large tables. Their performance does not degrade as the size of a table grows.

- B-trees optimize costly disk access.

Hashed Indices

Choosing a good hash function is critical to the success of this technique. By a good hash function, we mean two things. First, a good hash function, irrespective of the number of search keys, gives an average-case lookup that is a small constant. Second, the function distributes records uniformly and randonly amonf the buckets, where a bucket is defines as a unit of one or more records (typically a disk block). Correspondingly, the worst hash function is one that maps all the keys to the same bucket.

However, the drawback of using hashed indices includes:

- Though the number of buckets is fixed, the number of files may grow with time.
- If the number of bucjets is too large, storage space is wasted.
- If the number of buckets is too smal, there may be too many collisions.

It is recommended to set the number of buckets to twice the number of search key values in the file.

A hased file organization uses hased undices. Hashing is used to calculate the address of disk block where the desired record is stored. If K is the set of all search key values and B is the set of all bucket addresses, then a has function H maps K to B.

We can perform the following operations in a hashed file organization.

Insertion

To insert a record that has k_i as its search value, use the hash function $h(k_1)$ to compute the address of the bucket for that record. If the bucket is free, store the record else use chaining to store the record.

Search

To search a record having the key value k_1, use $h(k_1)$ to compute the address of the bucket where the record is stored. The bucket may contain one or several records, so check for every record in the bucket (by comparing k_i with the key of every record) to finally retrieve the desired record with the given key value.

Deletion

To delete a record with key value k_i, use $h(k_i)$ to compute the address of the bucket where the record is stored. The bucket may contain one or several records so check for every record in the bucket (by comparing k_i with the key of every record). Then delete the record as we delete a node from a linear linked list.

Note that in a hashed file organization, the secondary indices need to be organized using hashing.

Points to Remeber

• A file is a block of useful information which is available to a computer program and is usually stored on a persisten storage medium.

• Every file contains data. This data can be organized in a hierarchy to present a systematic organization. The data hierarchy includes data items such as fieldsm records, files, and database.

• A data field is an elementary unit that stores a single fact. A record is a collection of related data fields which is seen as a single unit from the application point of view. A file is a collection related files.

• A database is defined as a collection of data organized in a fashion that facilitates updating, retrieving, and managing the data.

• There are two types of computer files—text files and binary files. A text file is structured as a sequence of lines of alphabet, numbers, special characters, etc. However, the data in a text file is stored using its corresponding ASCII code. Whereas in binary files, the data is stored in binary form, i.e., in the format it is stored in the memory.

• Each file has a list of attributes associated with it which can have one of two states—`on` or `off`. These attributes are: read-only, hidden, system, volume label, archive, and directory.

• A file marked as read-only cannot be deleted or modified.

• A hidden file is not displayed in the directory listing.

• A system file is used by the system and should not be altered or removed from the disk.

• The archive bit is useful for communication between programs that modify files and programs that are used for backing up files.

• A gile that has the directory bit turned on is actually a sub-directory containing one or more files.

• File organiation means the logical arrangement of records in the file. Files can be organized as sequential, relative, or index sequential.

• A sequentially organized file stores records in the order in which they were entered.

• In relative file organization, records in a file are ordered by their relative key. Relative files can be used for both random access as well as sequential access of data.

• In an indexed sequential file, every record is uniquely identifed by a key field. We maintain a table known as the index table that stores record number and the address of the record in the file.

• There are several indexing techniques, and each technique works well for a particular application.

• In a dense index, index table stores the address of every record in the file. However, in a sparse index, index tables stores address of only some of the records in the file.

• Cylinder surface indexing is a very simple technique which is used only for the primary key index of a sequentially ordered file.

• In multi-level indexing, we can create an index to the index itself. The original index is called the first-level index and the index to the index is called the second-level index.

• Inverted files are frequently used indexing technique in document retrieval systems for large textual databases. An inverted file reogranizes the structure of an existing data file in order to provide fast access to all the records having one field falling within set limits.

• Majority of the database management systems use B-tree indexing technique. The index consists of a hierarchical structure with upper blocks containing indices pointing to the lower blocks and lowest leve blocks containing pointers to the data records.

• Hashed file organization uses hashed indices. Hashing is used to calculate the address of disk block where the desired record is stored. If K is the set of all search key values and B is the set of bucket addresses, then a has function H maps K to B.