**MULTIKEY FILE ORGANISATION**

we will introduce a family of file organisation schemes that allow records to be accessed by more than one key field. Until this point, we have considered only single-key file organisation. Sequential, by a given key; direct access by a particular key; and indexed sequential giving both direct and sequential access by a single key. Now we enlarge our base to include those file organisations that enable a single data file to support multiple access paths, each by a different key. These file organization techniques are at the heart of database implementation. The ability to search on many keys is enabled by building multiple index files (multikey file organisation) “on top of” the data file. The physical database then consists of one or more data files and many index files, and each data file contains either one or several record types. Each index file supports access by a particular field or group of fields. There are numerous techniques that have been used to implement multikey file organisation. Most of the approaches are based on building indexes to

provide direct access by key value. In this section, we will discuss two approaches for providing additional access paths into a file of data records.

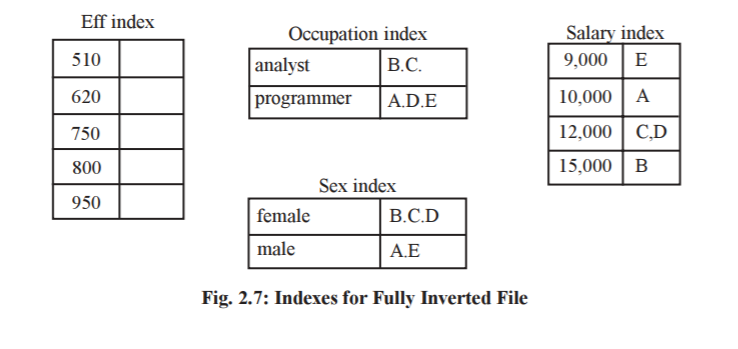
* Multilist file organisation
* Inverted file organisation

1. **Multilist File Organisation**

* The basic approach to providing the linkage between an index and the file of data records is called **multilist organisation**.
* A multilist file maintains an index for each secondary key. The index for secondary key contains, instead of a list of primary keys related to that secondary key, only one primary key value related to that secondary key. That record will be linked to other records containing the same secondary key in the data file.
* The multi-list organisation differs from inverted file in that while the entry in the inverted file index for a key value has a pointer to each data record with that key value, the entry in the multi-list index for a key value has just one pointer to the first data record with that key value.
* Linking records together in order of increasing primary key value facilitates easy insertion and deletion once the place at which the insertion or deletion to be made is known.
* Searching for a record with a given primary key value is difficult when no index is available, since the only search possible is a sequential search.
* To facilitate searching on the primary key as well as on secondary keys, it is customary to maintain several indexes, one for each key. Using an index in this way reduces the length of the lists and thus the search time.
* This idea is very easily generalized to allow for easy secondary key retrieval. We just set up indexes for each key and allow records to be in more than one list. This leads to the multilist structure for file representation.

1. **Inverted File Organisation**

In inverted file organisation, a linkage is provided between an index and the file of data records. A key’s inverted index contains all of the values that the key presently has in the records of the data file. Each key-value entry in the inverted index points to all of the data records that have the corresponding value. Inverted files represent one extreme of file organisation in which only the index structures are important. The records themselves may be stored in any way (sequentially ordered by primary key, random, linked ordered by primary key etc.).



Inverted files may also result in space saving compared with other file structures when record retrieval does not require retrieval of key fields. In this case, the key fields may be deleted from the records.

**Both inverted files and multilist files have:**

* An index for each secondary key.
* An index entry for each distinct value of the secondary key.
* The index may be tabular or tree-structured.
* The entries in an index may or may not be sorted.
* The pointers to data records may be direct or indirect.

**The indexes differ in that**

* An entry in an inverted index has a pointer to each data record with that value.
* An entry in a multilist index has a pointer to the first data record with that value. Thus, an inverted index may have variable-length entries whereas a multilist index has fixed-length entries.

**Some of the implications of these differences are the following:**

* Index management is easier in the multilist approach because entries are fixed in length.
* The inverted file approach tends to exhibit better inquiry performance. Many types of queries can be answered by accessing inversion indexes without necessitating access to data records, thereby reducing I/O-access requirements.
* Inversion of a file can be transparent to a programmer who accesses that file but does not use the inversion indexes, while a multilist structure affects the file’s record layout. The multilist pointers can be made transparent to a programmer if the data manager does not make them available for programmer use and stores them at the end of each record.