Unit 3

FILE ORGANIZATION AND INDEXES

Data on External Storage

Disks: Can retrieve random page at fixed cost	Tapes: Can only read pages in sequence
▪ But reading several consecutive pages is much cheaper than reading them in random order	▪ Cheaper than disks; used for archival storage
File organization: Method of arranging a file of records on external storage.	

File Organizations

- Heap (random order) files: Suitable when typical access is a file scan retrieving all records.
- Sorted Files: Best if records must be retrieved in some order, or only a `range' of records is needed.
- Indexes: Data structures to organize records via trees or hashing.

COST MODEL

- **B:** The number of data pages
- **R:** Number of records per page
- **D:** (Average) time to read or write disk page
- Measuring number of page I/O's ignores gains of pre-fetching a sequence of pages; thus, even I/O cost is only approximated.
- Average-case analysis; based on several simplistic assumptions.

Typical values today are $D = 15$ milliseconds, C and $H = 100$ nanoseconds;

COMPARISON OF THREE FILES ORGANIZATIONS

Operations to Compare

- Scan: Fetch all records from disk
- Equality search
- Range selection
- Insert a record
- Delete a record

Cost of Operations

	(a) Scan	(b) Equality	(c) Range	(d) Insert	(e) Delete
(1) Heap					
(2) Sorted					
(3) Clustered					
(4) Unclustered Tree index					
(5) Unclustered Hash index					

	(a) Scan	(b) Equality	(c) Range	(d) Insert	(e) Delete
(1) Heap	BD	0.5BD	BD	2D	Search +D
(2) Sorted	BD	Dlog 2B	Dlog 2 B + # matches	Search + BD	Search +BD
(3) Clustered	1.5BD	Dlog F 1.5B	Dlog F 1.5B + # matches	Search + D	Search +D
(4) Unclustered Tree index	BD(R+0.15)	D(1 + log F 0.15B)	Dlog F 0.15B + # matches	D(3 + log F 0.15B)	Search + 2D
(5) Unclustered Hash index	BD(R+0.125)	2D	BD	4D	Search + 2D

OVERVIEW OF INDEXES

- An *index* on a file speeds up selections on the *search key fields* for the index.
- An index contains a collection of *data entries*, and supports efficient retrieval of all data entries k* with a given key value k.

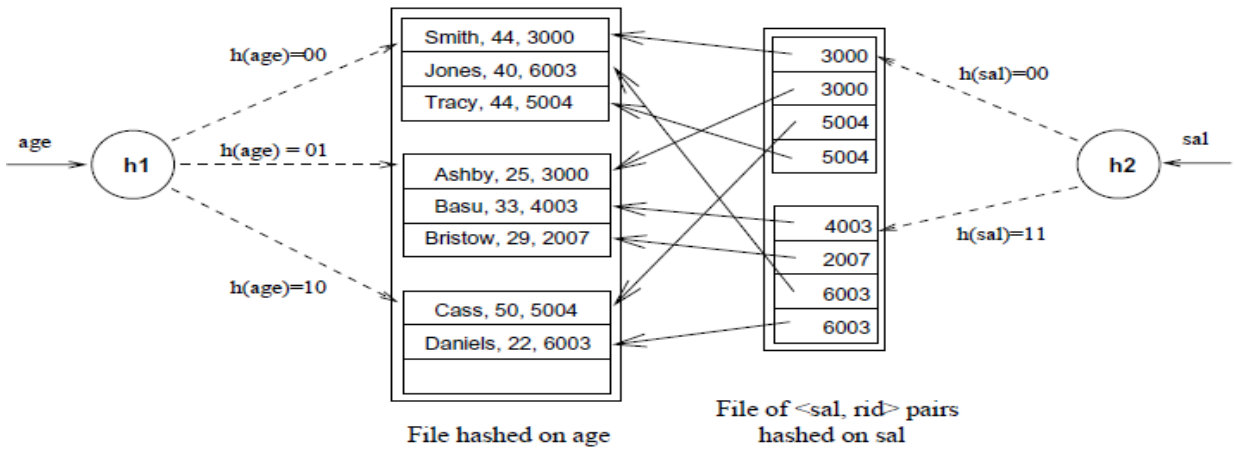


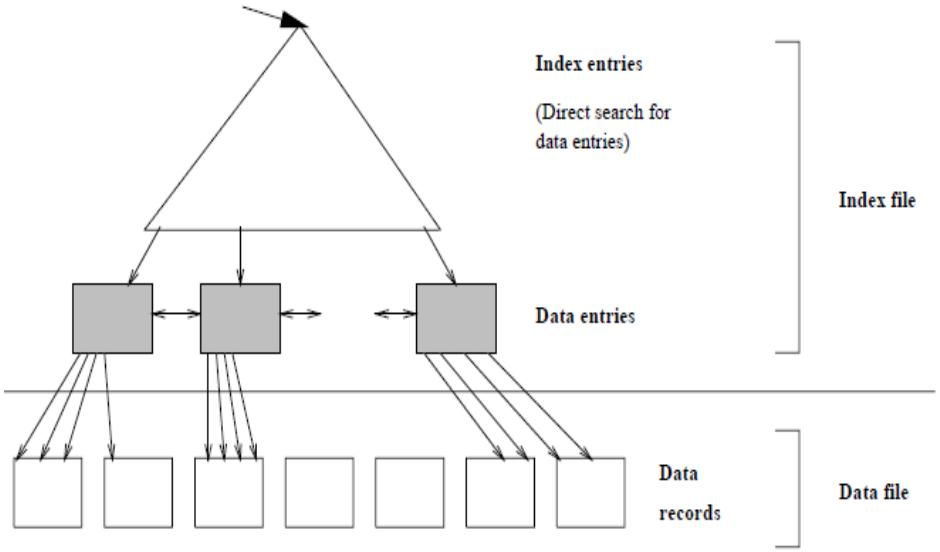
Figure 8.2 File Hashed on *age*, with Index on *sal*

Alternatives for Data Entries in an Index

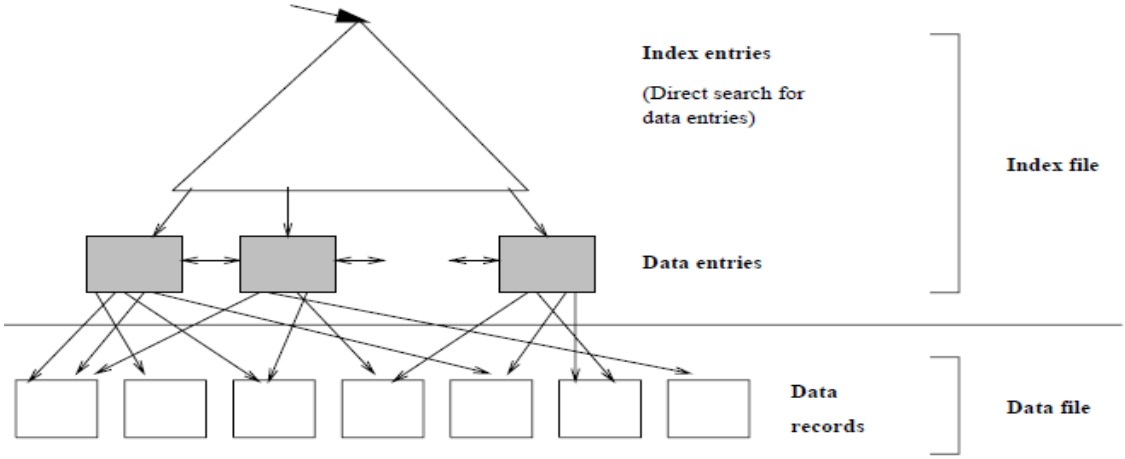
1. A data entry $k*$ is an actual data record (with search key value k).
2. A data entry is a $\langle k, rid \rangle$ pair, where *rid* is the record id of a data record with search key value k.
3. A data entry is a $\langle k, rid\text{-}list \rangle$ pair, where *rid-list* is a list of record ids of data records with search key value k.

Properties of indexes

- Clustered and un-clustered index



When a file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index, we say that the index is **clustered.**



- Dense and sparse indexes

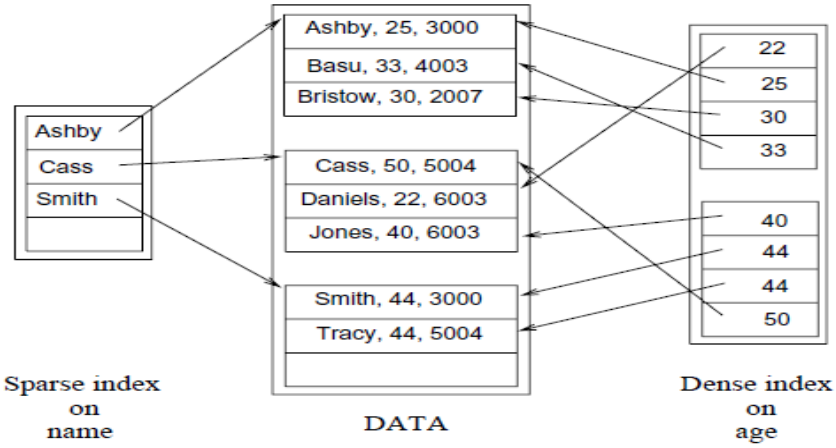


Figure 8.5 Sparse versus Dense Indexes

The first index is a sparse, clustered index
name. Notice how the order of data entries in the index corresponds to the order records in the data file. There is one data entry per page of data records. The second index is a dense, unclustered index on the *age* field. Notice that the order of data entries in the index differs from the order of data records.

- Primary and secondary index

An index on a set of fields that includes the *primary key* is called a **primary index**. An index that is not a primary index is called a **secondary** index. (The terms *primary index* and *secondary index* are sometimes used with a different meaning: An index that uses Alternative (1) is called a primary index, and one that uses Alternatives (2) or (3) is called a secondary index.

- Indexes using composite search keys

The search key for an index can contain several fields; such keys are called **composite search keys** or **concatenated keys**.

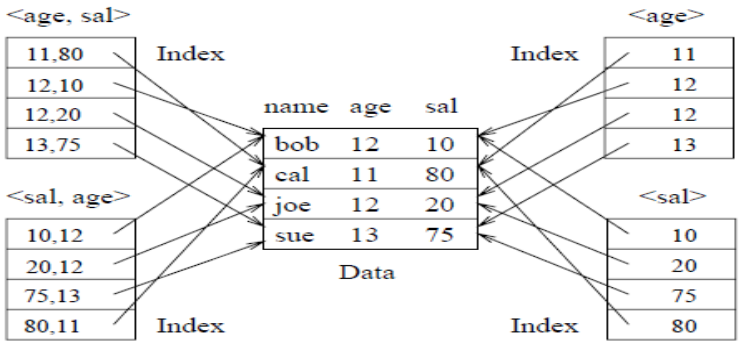


Figure 8.6 Composite Key Indexes

TREE STRUCTURED INDEDEXING

- Two index data structures, called ISAM and B+ trees, based on tree organizations

  ❖ Tree-structured indexing techniques support both *range searches* and *equality searches.*

- **ISAM**

  An ISAM tree is a static index structure that is effective when the file is not frequently updated, but it is unsuitable for files that grow and shrink a lot.

- **B+ Trees**

  It is a dynamic index structure

INDEXED SEQUENTIAL ACCESS METHOD (ISAM)

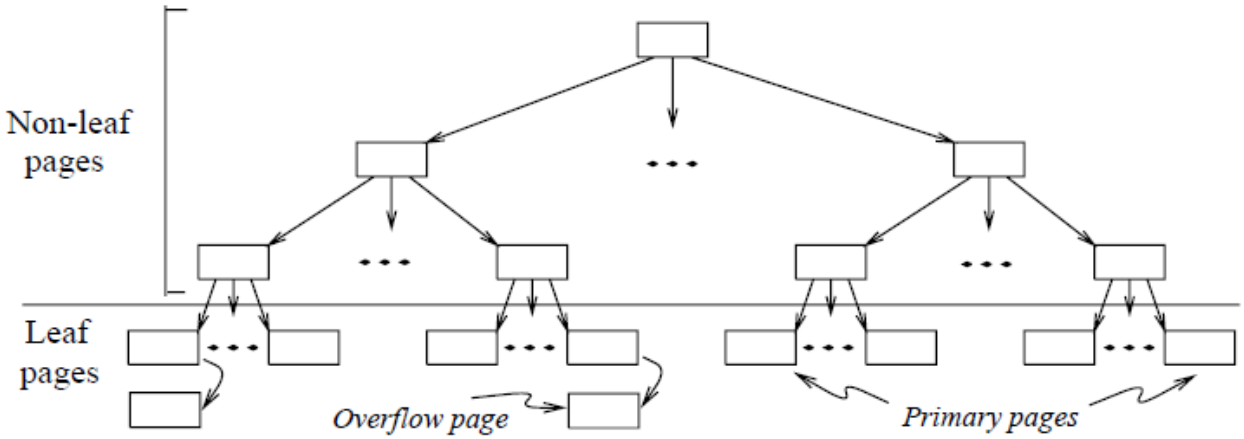


Figure 9.3 ISAM Index Structure

The data entries of the ISAM index are in the leaf pages of the tree and additional *overflow* pages that are chained to some leaf page.

The ISAM structure is completely static (except for the overflow pages, of which it is hoped, there will be few) and facilitates such low-level optimizations.

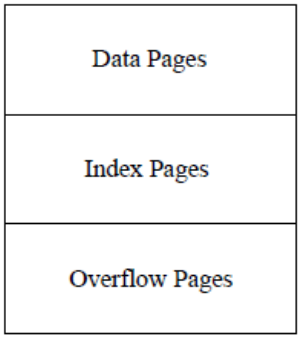


Figure 9.4 Page Allocation in ISAM

The following example illustrates the ISAM index structure. Consider the tree shown in Figure 9.5.

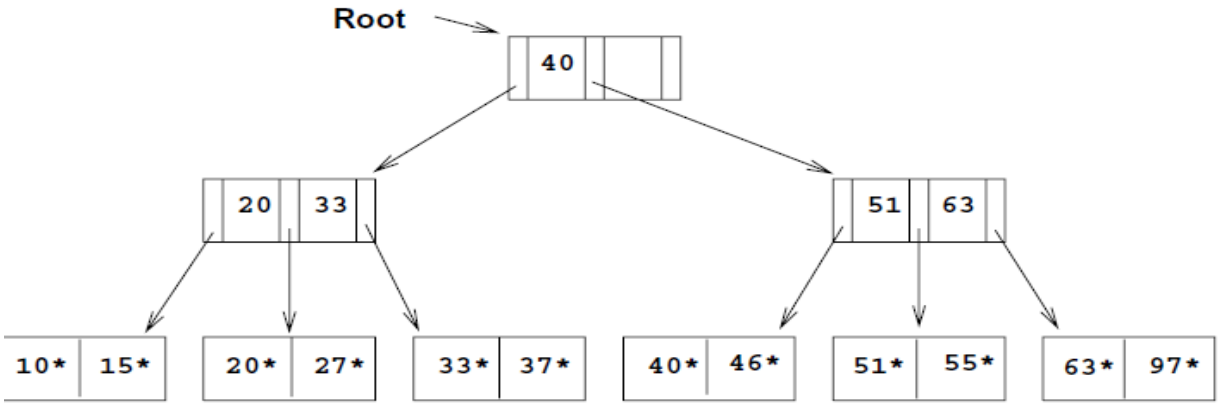


Figure 9.5 Sample ISAM Tree

All searches begin at the root. For example, to locate a record with the key value 27, we start at the root and follow the left pointer, since $27 < 40$. We then follow the middle pointer, since $20 <= 27 < 33$. For a range search, we find the first qualifying data entry as for an equality selection and then retrieve primary leaf pages sequentially (also retrieving overflow pages as needed by following pointers from the primary pages). The primary leaf pages are assumed to be allocated sequentially—this assumption is reasonable because the number of such pages is known when the tree is created and does not change subsequently under inserts and deletes—and so no 'next leaf page' pointers are needed.

We assume that each leaf page can contain two entries. If we now insert a record with key value 23, the entry 23* belongs in the second data page, which already contains 20* and 27* and has no more space. We deal with this situation by adding an *overflow* page and putting 23* in the overflow page. Chains of overflow pages can easily develop.

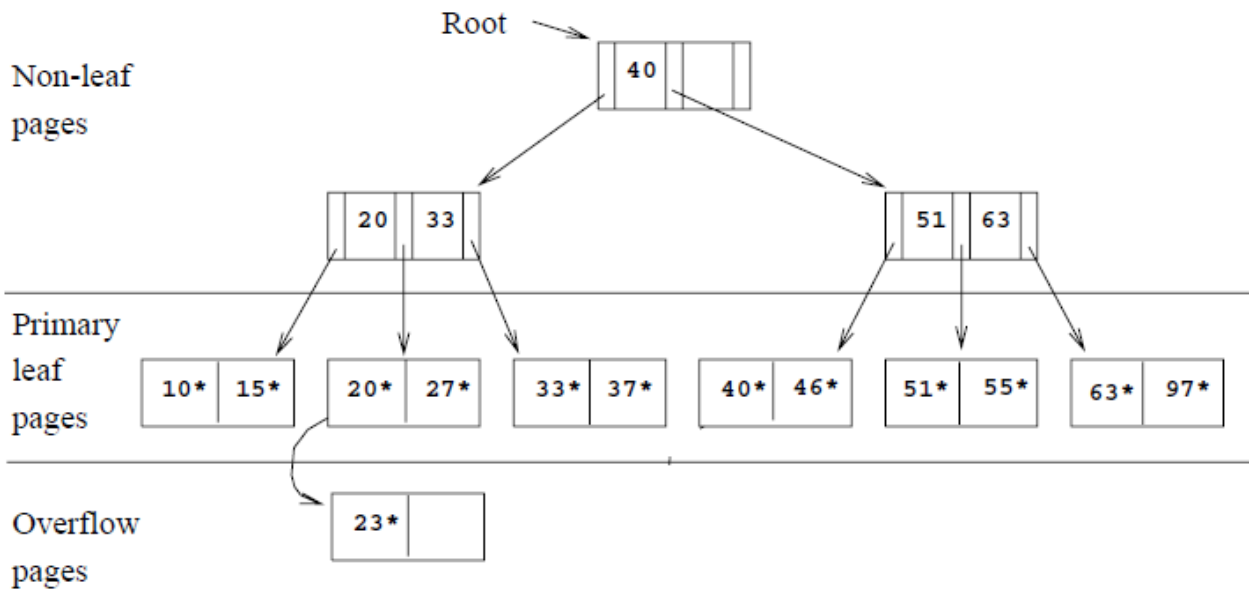


Figure 9.6 ISAM Tree after Inserts

For instance, inserting 48*, 41*, and 42* leads to an overflow chain of two pages. The tree of Figure 9.5 with all these insertions is shown in Figure 9.6.

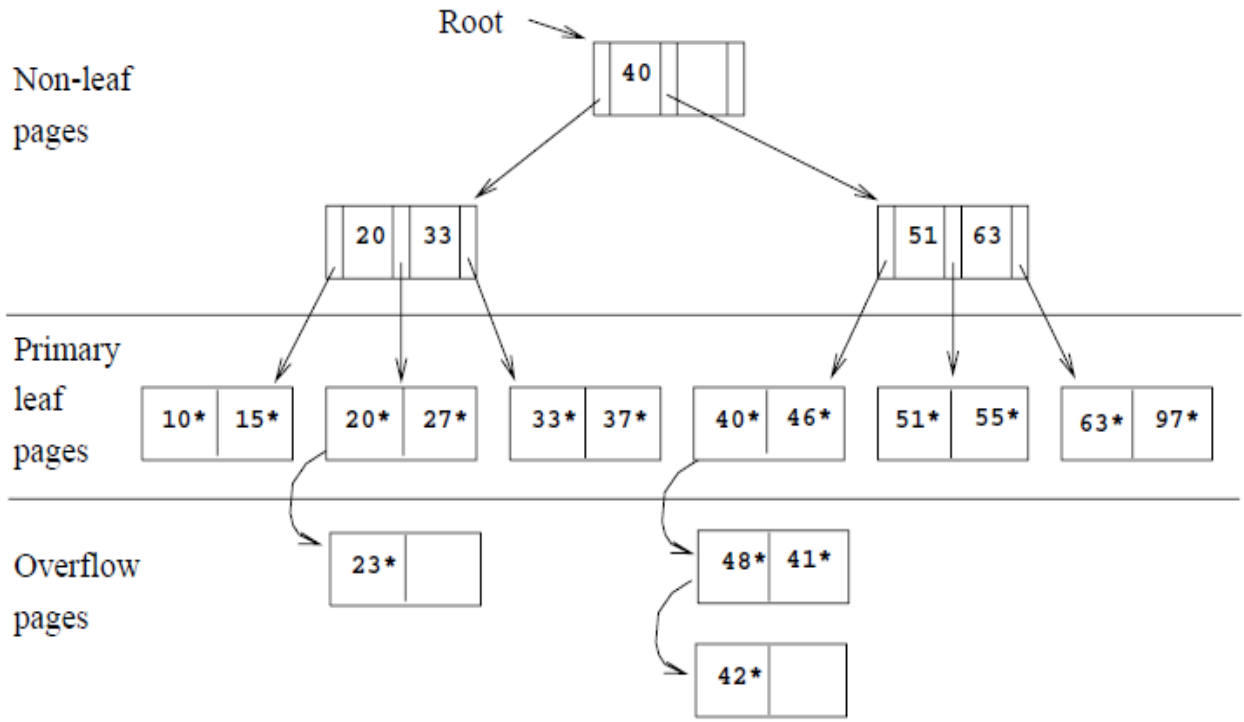
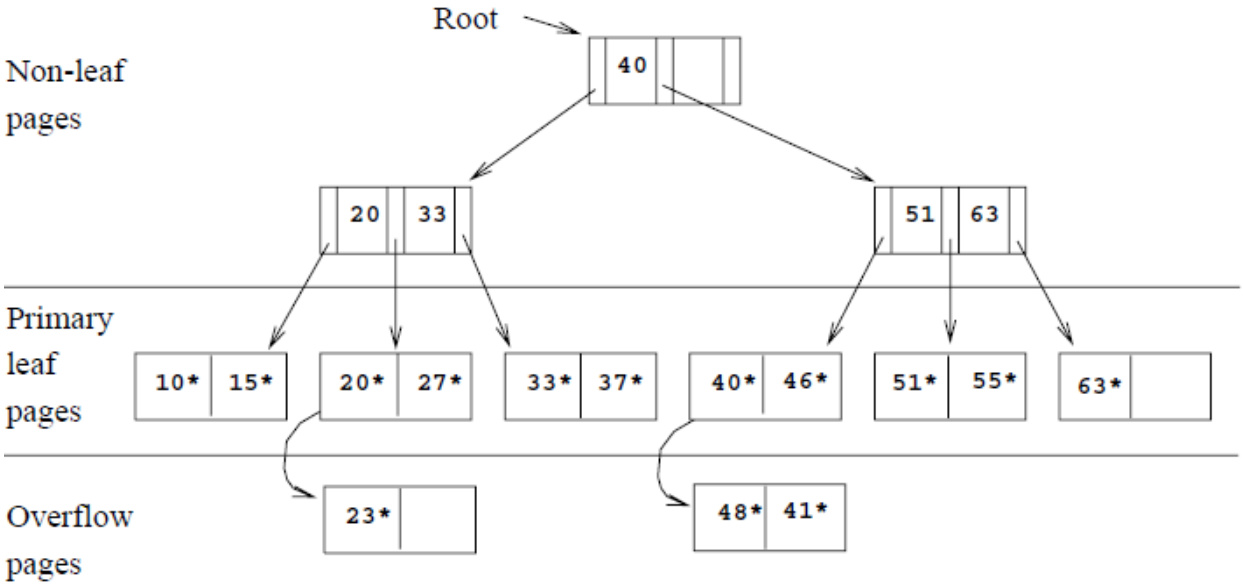


Figure 9.6 ISAM Tree after Inserts

tree of Figure 9.6 is shown in Figure 9.7 after deletion of the entries 42* and 97*.



ISAM Tree after Deletes