



File Organizations and Indexing

Chapter 8

"How index-learning turns no student pale
Yet holds the eel of science by the tail."
-- Alexander Pope (1688-1744)



Alternative File Organizations

Many alternatives exist, *each ideal for some situation, and not so good in others:*

- **Heap 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.
- **Hashed Files:** Good for equality selections.
 - ◆ File is a collection of *buckets*. Bucket = *primary* page plus zero or more *overflow* pages.
 - ◆ *Hashing function h*: $h(r)$ = bucket in which record r belongs. h looks at only some of the fields of r , called the *search fields*.



Cost Model for Our Analysis

We ignore CPU costs, for simplicity:

- **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 blocks of pages; thus, even I/O cost is only approximated.
- Average-case analysis; based on several simplistic assumptions.

☛ *Good enough to show the overall trends!*



Assumptions in Our Analysis

- ◆ Single record insert and delete.
- ◆ Heap Files:
 - Equality selection on key; exactly one match.
 - Insert always at end of file.
- ◆ Sorted Files:
 - Files compacted after deletions.
 - Selections on sort field(s).
- ◆ Hashed Files:
 - No overflow buckets, 80% page occupancy.



Cost of Operations

	Heap File	Sorted File	Hashed File
Scan all recs			
Equality Search			
Range Search			
Insert			
Delete			

☛ *Several assumptions underlie these (rough) estimates!*



Cost of Operations

	Heap File	Sorted File	Hashed File
Scan all recs	BD	BD	1.25 BD
Equality Search	0.5 BD	$D \log_2 B$	D
Range Search	BD	$D (\log_2 B + \# \text{ of pages with matches})$	1.25 BD
Insert	2D	Search + BD	2D
Delete	Search + D	Search + BD	2D

☛ *Several assumptions underlie these (rough) estimates!*

Indexes

- ❖ An *index* on a file speeds up selections on the *search key fields* for the index.
 - Any subset of the fields of a relation can be the search key for an index on the relation.
 - *Search key* is not the same as *key* (minimal set of fields that uniquely identify a record in a relation).
- ❖ An index contains a collection of *data entries*, and supports efficient retrieval of all data entries k^* with a given key value k .

Alternatives for Data Entry k^* in Index

- ❖ Three alternatives:
 - ① Data record with key value k
 - ② $\langle k, \text{rid of data record with search key value } k \rangle$
 - ③ $\langle k, \text{list of rids of data records with search key } k \rangle$
- ❖ Choice of alternative for data entries is orthogonal to the indexing technique used to locate data entries with a given key value k .
 - Examples of indexing techniques: B+ trees, hash-based structures
 - Typically, index contains auxiliary information that directs searches to the desired data entries

Alternatives for Data Entries (Contd.)

- ❖ Alternative 1:
 - If this is used, index structure is a file organization for data records (like Heap files or sorted files).
 - At most one index on a given collection of data records can use Alternative 1. (Otherwise, data records duplicated, leading to redundant storage and potential inconsistency.)
 - If data records very large, # of pages containing data entries is high. Implies size of auxiliary information in the index is also large, typically.

Alternatives for Data Entries (Contd.)

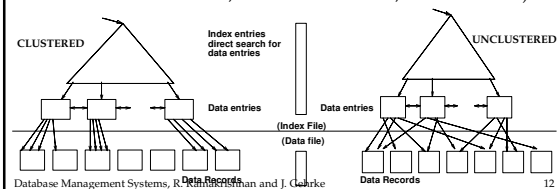
- ❖ Alternatives 2 and 3:
 - Data entries typically much smaller than data records. So, better than Alternative 1 with large data records, especially if search keys are small. (Portion of index structure used to direct search is much smaller than with Alternative 1.)
 - If more than one index is required on a given file, at most one index can use Alternative 1; rest must use Alternatives 2 or 3.
 - Alternative 3 more compact than Alternative 2, but leads to variable sized data entries even if search keys are of fixed length.

Index Classification

- ❖ *Primary* vs. *secondary*: If search key contains primary key, then called primary index.
 - *Unique* index: Search key contains a candidate key.
- ❖ *Clustered* vs. *unclustered*: If order of data records is the same as, or 'close to', order of data entries, then called clustered index.
 - Alternative 1 implies clustered, but not vice-versa.
 - A file can be clustered on at most one search key.
 - Cost of retrieving data records through index varies *greatly* based on whether index is clustered or not!

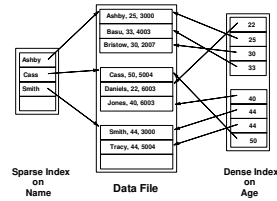
Clustered vs. Unclustered Index

- ❖ Suppose that Alternative (2) is used for data entries, and that the data records are stored in a Heap file.
 - To build clustered index, first sort the Heap file (with some free space on each page for future inserts).
 - Overflow pages may be needed for inserts. (Thus, order of data recs is 'close to', but not identical to, the sort order.)



Index Classification (Contd.)

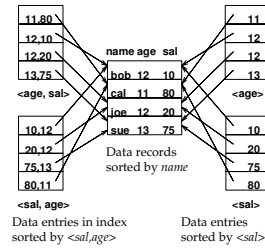
- ❖ **Dense vs. Sparse:** If there is at least one data entry per search key value (in some data record), then dense.
 - Alternative 1 always leads to dense index.
 - Every sparse index is clustered!
 - Sparse indexes are smaller; however, some useful optimizations are based on dense indexes.



Index Classification (Contd.)

- ❖ **Composite Search Keys:** Search on a combination of fields.
 - Equality query: Every field value is equal to a constant value. E.g. wrt $\langle \text{sal}, \text{age} \rangle$ index:
 - ♦ $\text{age}=20$ and $\text{sal}=75$
 - Range query: Some field value is not a constant. E.g.:
 - ♦ $\text{age}=20$; or $\text{age}=20$ and $\text{sal} > 10$
- ❖ **Data entries in index sorted by search key to support range queries.**
 - Lexicographic order, or
 - Spatial order.

Examples of composite key indexes using lexicographic order.



Summary

- ❖ Many alternative file organizations exist, each appropriate in some situation.
- ❖ If selection queries are frequent, sorting the file or building an *index* is important.
 - Hash-based indexes only good for equality search.
 - Sorted files and tree-based indexes best for range search; also good for equality search. (Files rarely kept sorted in practice; B+ tree index is better.)
- ❖ Index is a collection of data entries plus a way to quickly find entries with given key values.

Summary (Contd.)

- ❖ Data entries can be actual data records, $\langle \text{key}, \text{rid} \rangle$ pairs, or $\langle \text{key}, \text{rid-list} \rangle$ pairs.
 - Choice orthogonal to *indexing technique* used to locate data entries with a given key value.
- ❖ Can have several indexes on a given file of data records, each with a different search key.
- ❖ Indexes can be classified as clustered vs. unclustered, primary vs. secondary, and dense vs. sparse. Differences have important consequences for utility/performance.