

File Organizations and Indexing

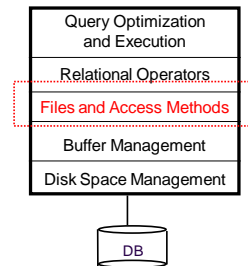
R&G Chapter 8

"If you don't find it in the index, look very carefully through the entire catalogue."

-- Sears, Roebuck, and Co.,
Consumer's Guide, 1897



Context



Alternative File Organizations

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

- Heap files: Suitable when typical access is a file scan retrieving all records.
- Sorted Files: Best for retrieval in *search key* order, or only a 'range' of records is needed.
- Clustered Files (with Indexes): Coming soon...



Cost Model for Analysis

We ignore CPU costs, for simplicity:

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

☒ *Good enough to show the overall trends!*



Some Assumptions in the Analysis

- Single record insert and delete.
- Equality selection - exactly one match (what if more or less???)
- Heap Files:
 - Insert always appends to end of file.
- Sorted Files:
 - Files compacted after deletions.
 - Selections on search key.



Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

| | Heap File | Sorted File | Clustered File |
|------------------|-----------|-------------|----------------|
| Scan all records | | | |
| Equality Search | | | |
| Range Search | | | |
| Insert | | | |
| Delete | | | |



Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

| | Heap File | Sorted File | Clustered File |
|------------------|-----------|-------------|----------------|
| Scan all records | BD | BD | |
| Equality Search | | | |
| Range Search | | | |
| Insert | | | |
| Delete | | | |



Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

| | Heap File | Sorted File | Clustered File |
|------------------|-----------|------------------|----------------|
| Scan all records | BD | BD | |
| Equality Search | 0.5 BD | $(\log_2 B) * D$ | |
| Range Search | | | |
| Insert | | | |
| Delete | | | |



Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

| | Heap File | Sorted File | Clustered File |
|------------------|-----------|----------------------------------|----------------|
| Scan all records | BD | BD | |
| Equality Search | 0.5 BD | $(\log_2 B) * D$ | |
| Range Search | BD | $[(\log_2 B) + \#match\ pg] * D$ | |
| Insert | | | |
| Delete | | | |



Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

| | Heap File | Sorted File | Clustered File |
|------------------|-----------|--|----------------|
| Scan all records | BD | BD | |
| Equality Search | 0.5 BD | $(\log_2 B) * D$ | |
| Range Search | BD | $[(\log_2 B) + \#match\ pg] * D$ | |
| Insert | 2D | $((\log_2 B) + B)D$ (because $R, W 0.5$) | |
| Delete | | | |



Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

| | Heap File | Sorted File | Clustered File |
|------------------|-----------|--|----------------|
| Scan all records | BD | BD | |
| Equality Search | 0.5 BD | $(\log_2 B) * D$ | |
| Range Search | BD | $[(\log_2 B) + \#match\ pg] * D$ | |
| Insert | 2D | $((\log_2 B) + B)D$ | |
| Delete | 0.5BD + D | $((\log_2 B) + B)D$ (because $R, W 0.5$) | |



Indexes

- Sometimes, we want to retrieve records by specifying the *values in one or more fields*, e.g.,
 - Find all students in the "CS" department
 - Find all students with a gpa > 3
- An *index* on a file is a disk-based data structure that 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* (e.g. doesn't have to be unique ID).
- An index contains a collection of *data entries*, and supports efficient retrieval of all records with a given search key value **k**.



First Question to Ask About Indexes

- What kinds of selections do they support?
 - Selections of form field <op> constant
 - Equality selections (op is =)
 - Range selections (op is one of <, >, <=, >=, BETWEEN)
 - More exotic selections:
 - 2-dimensional ranges ("east of Berkeley and west of Truckee and North of Fresno and South of Eureka")
 - Or n-dimensional
 - 2-dimensional distances ("within 2 miles of Soda Hall")
 - Or n-dimensional
 - Ranking queries ("10 restaurants closest to Berkeley")
 - Regular expression matches, genome string matches, etc.
 - One common n-dimensional index: R-tree
 - Supported in Oracle and Informix
 - See <http://gist.cs.berkeley.edu> for research on this topic



Index Breakdown

- What selections does the index support
- Representation of data entries in index
 - i.e., what kind of info is the index actually storing?
 - 3 alternatives here
- Clustered vs. Unclustered Indexes
- Single Key vs. Composite Indexes
- Tree-based, hash-based, other



Alternatives for Data Entry k^* in Index

- Three alternatives:
 - Actual data record (with key value k)
 - $\langle k, \text{rid of matching data record} \rangle$
 - $\langle k, \text{list of rids of matching data records} \rangle$
- Choice is orthogonal to the indexing technique.
 - Examples of indexing techniques: B+ trees, hash-based structures, R trees, GiSTs, ...
 - Typically, index contains auxiliary information that directs searches to the desired data entries
- Can have multiple (different) indexes per file.
 - E.g. file sorted by *age*, with a hash index on *salary* and a B+tree index on *name*.



Alternatives for Data Entries (Contd.)

- Alternative 1:
 - Actual data record (with key value k)
 - 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.
 - This alternative saves pointer lookups but can be expensive to maintain with insertions and deletions.



Alternatives for Data Entries (Contd.)

Alternative 2

$\langle k, \text{rid of matching data record} \rangle$

and Alternative 3

$\langle k, \text{list of rids of matching data records} \rangle$

- Easier to maintain than Alt 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.
- Even worse, for large rid lists the data entry would have to span multiple blocks!

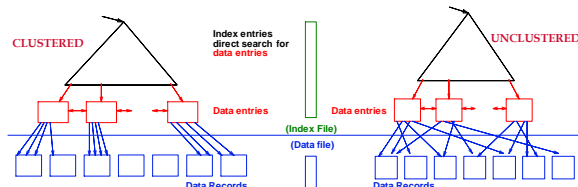


Index Classification

- *Clustered vs. unclustered*: If order of *data records* is the same as, or 'close to', order of *index data entries*, then called *clustered index*.
 - 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!
 - Alternative 1 implies clustered, *but not vice-versa*.

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 block for future inserts).
 - Overflow blocks may be needed for inserts. (Thus, order of data recs is 'close to', but not identical to, the sort order.)



Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

| | Heap File | Sorted File | Clustered File |
|------------------|-----------|---|------------------------------------|
| Scan all records | BD | BD | 1.5 BD |
| Equality Search | 0.5 BD | $(\log_2 B) * D$ | $(\log_F 1.5B) * D$ |
| Range Search | BD | $[(\log_2 B) + \#match pg] * D$ | $[(\log_F 1.5B) + \#match pg] * D$ |
| Insert | 2D | $((\log_2 B) + B)D$ | $((\log_F 1.5B) + 1) * D$ |
| Delete | 0.5BD + D | $((\log_2 B) + B)D$ (because R, W 0.5) | $((\log_F 1.5B) + 1) * D$ |

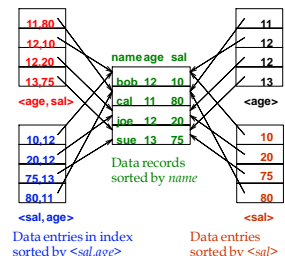
Unclustered vs. Clustered Indexes

- What are the tradeoffs????
- Clustered Pros
 - Efficient for range searches
 - May be able to do some types of compression
 - Possible locality benefits (related data?)
 - ???
- Clustered Cons
 - Expensive to maintain (on the fly or sloppy with reorganization)
 - Pages tend to be only 2/3 full!

Composite Search Keys

- Search on a combination of fields.
 - Equality query: Every field value is equal to a constant value. E.g. wrt <age,sal> index:
 - age=20 and sal =75
 - Range query: Some field value is not a constant. E.g.:
 - age > 20; or age=20 and sal > 10
- Data entries in index sorted by search key to support range queries.
 - Lexicographic order
 - Like the dictionary, but on fields, not letters!

Examples of composite key indexes using lexicographic order.



Summary

- File Layer manages access to records in pages.
 - Record and page formats depend on fixed vs. variable-length.
 - Free space management an important issue.
 - Slotted page format supports variable length records and allows records to move on page.
- 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 in index can be *actual data records*, <key, rid> pairs, or <key, rid-list> pairs.
 - Choice orthogonal to *indexing structure* (i.e., tree, hash, etc.).
- Usually have several indexes on a given file of data records, each with a different search key.
- Indexes can be classified as *clustered* vs. *unclustered*
- Differences have important consequences for utility/performance.
- Catalog relations store information about relations, indexes and views.