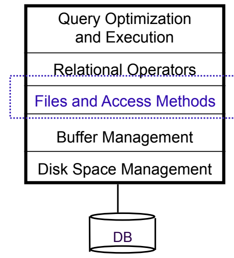
File Organizations and Indexing

"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



Goal for This Deck

- Big picture overheads for data access
  - We'll simplify things to get focused
  - Still, a bit of discipline:
    - Clearly identify assumptions up front
    - Then estimate cost in a principled way
- Foundation for query optimization
  - Can't choose the fastest scheme without an estimate of speed!

Multiple 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

- **B:** The number of data blocks
- **R:** Number of records per block
- **D:** (Average) time to read or write disk block
- *Average-case* analyses for *uniform random* workloads
- We will ignore:
  - Sequential vs. Random I/O
  - Pre-fetching
  - Any in-memory costs

☛ ***Good enough to show the overall trends!***

More Assumptions

- Single record insert and delete.
- Equality selection - exactly one match
- For Heap Files:
  - Insert always appends to end of file.
- For Sorted Files:
  - Files compacted after deletions.
  - Selections on search key.
- Question all these assumptions and rework
  - As an exercise to study for tests, generate ideas

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	$((\log_2 B)+B)D$	
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	$((\log_2 B)+B)D$	

Indexes

- Allow record retrieval *by value* in ≥1 field, e.g.,
  - Find all students in the "CS" department
  - Find students with a gpa > 3
  - Find students with firstname "Bob", lastname "Nob"
- *Index* : disk-based data structure for fast lookup by value
  - *Search key:* any subset of columns in the relation.
  - *Search key* need not be a *key* of the relation
    - I.e. There can be multiple items matching a search key
- Index contains a collection of *data entries*
  - (***k***, {items})
  - Items associated with each search key value ***k***
  - Data entries come in various forms, as we'll see

1st Question to Ask About Indexes

- What kinds of selections (lookups) do they support?
  - Selection: <key> <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 radii ("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
- See http://en.wikipedia.org/wiki/GiST for more

Categorizing Typical Indexes

- 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
- Index structure: Tree-based, hash-based, other

Alternatives for Data Entry **k*** in Index

- Three alternatives:
  1. Actual data record (with key value **k**)
  2. <**k**, rid of matching data record>
  3. <**k**, list of rids of matching data records>
- Choice is orthogonal to the indexing technique.
  - B+ trees, hash-based structures, R trees, GiSTs, …
- 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***)

- Index as a file organization for records
  - Alongside Heap files or sorted files
- At most one Alt. 1 index per relation
- No "pointer lookups" to get data records

Alternatives for Data Entries (Contd.)

Alternative 2
<**k**, rid of matching data record>
and Alternative 3
<**k**, list of rids of matching data records>

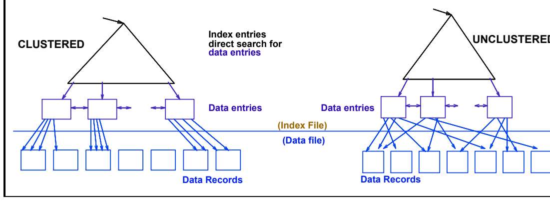
- Alts. 2 or 3 required to support multiple indexes per relation!
- Alt. 3 more compact than Alt. 2, but *variable sized data entries*
  - even if search keys are of fixed length.
- For large rid lists, data entry spans multiple blocks!

Clustered vs. Unclustered Index

- In a clustered index:
  - index data entries are stored in (approximate) order by value of search keys in data records
  - 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 => clustered
    - but not vice-versa!
- Note: there is another definition of "clustering"
  - Data Mining/AI: grouping similar items in n-space

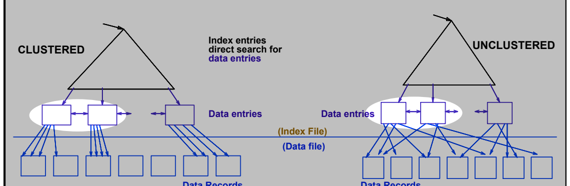
Clustered vs. Unclustered Index

- Alternative (2) data entries, data records 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.



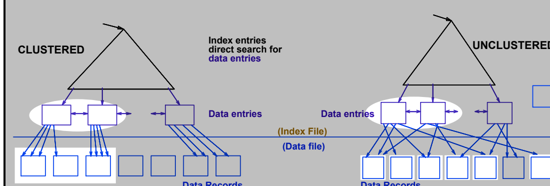
Clustered vs. Unclustered Index

- Alternative (2) data entries, data records 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.



Clustered vs. Unclustered Index

- Alternative (2) data entries, data records 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.



Unclustered vs. Clustered Indexes

- Clustered Pros
  - Efficient for range searches
  - Potential locality benefits
    - Disk scheduling, prefetching, etc.
  - Support certain types of compression
    - More soon on this topic
- Clustered Cons
  - More expensive to maintain
    - on the fly or "sloppily" via reorgs
    - Heap file usually only packed to 2/3 to accommodate inserts

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	$(\log_2 B) * D$	$(\log_F 1.5B+1) * D$
Range Search	BD	$[(\log_2 B) +$ #match pg$]*D$	$[(\log_F 1.5B) +$ *#match pg*$]*D$
Insert	2D	$((\log_2 B)+B)D$	$((\log_F 1.5B)+2) * D$
Delete	0.5BD + D	$((\log_2 B)+B)D$ *(because R,W 0.5)*	$((\log_F 1.5B)+2) * D$

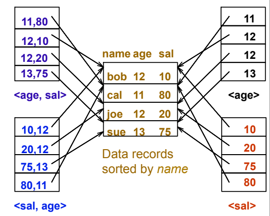
Composite Search Keys

- Search on a combo of fields.
  - E.g. Index on <age, sal>
  - Equality query:
    - age = 20
    - age = 20 and sal = 75
  - Range query:
    - age > 20
    - age = 20 and sal > 10
- Data entries in index can be sorted by search key to support range queries.
  - Lexicographic order
  - Like the dictionary, but on fields, not letters

Composite Search Keys

- Search on a combo of fields.
  - E.g. Index on <age, sal>
  - Equality query:
    - age = 20
    - age = 20 and sal = 75
  - Range query:
    - age > 20
    - age = 20 and sal > 10
- Data entries in index can be 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/page formats: fixed vs. variable-length.
  - Free space management
  - Slotted page format: var-len records, movable!
- Many file orgs, with tradeoffs
  - Understand the cost analyses
- For selection queries, sort/index
  - Tree-based indexes support equality, range search
- Index: two things
  - a collection of data entries
  - a way to quickly find entries with given key values

Summary (Contd.)

- Data entries: 1 of 3 alternatives:
  1. actual data records
  2. <key, rid> pairs, or
  3. <key, rid-list> pairs.
  - Choice orthogonal to *indexing structure (i.e., tree, hash, etc.).*
- Often multiple indexes per file of data records
  - each with a different search key.
- Indexes can be classified as *clustered* vs. *unclustered*
  - Difs have important consequences for utility/performance.
- Catalog stores info about relations, indexes and views.
  - Catalog is itself a set of relations