Indexing - revisited

CS 186, Fall 2012
R & G Chapter 8
Index Classification

1. Selections (lookups) supported
2. Representation of data entries in index
   – what kind of info is the index actually storing?
   – 3 alternatives here
3. Clustered vs. Unclustered Indexes
4. Single Key vs. Composite Indexes
5. Tree-based, hash-based, other
Indexes: Selections supported

field <op> constant

- **Equality** selections (<op> is =)
  - Either “tree” or “hash” indexes help here.
- **Range** selections (<op> is one of <, >, <=, >=, BETWEEN)
  - “Hash” indexes don’t work for these.

More exotic selections

- multi-dimensional ranges (“east of Berkeley and west of Truckee and North of Fresno and South of Eureka”)
- multi-dimensional **distances** (“within 2 miles of Soda Hall”)
- Ranking queries (“10 restaurants closest to Berkeley”)
- Regular expression matches, genome string matches, etc.
- Keyword/Web search - includes “importance” of words in documents, link structure, ...
Tree Index: Example

- **Index entries**: <search key value, page id> they direct search for data entries *in leaves.*

- In example: **Fanout** \((F) = 3\) (note: unrealistic!)
  
  — more typical: 16KB page, 67% full, 32Byte entries = approx 300.
What’s in a “Data Entry”? 

• **Question**: What is actually stored in the leaves of the index for key value “k”? (A data entry for key “k” is denoted “k*” in book and examples)

• Three alternatives:
  1. Actual data record(s) with key value k
  2. \{<k, rid of a matching data record>\}
  3. <k, \{rids of all matching data records\}>

• Choice is orthogonal to the indexing technique.
  – e.g., B+ trees, hash-based structures, R trees, ...
Alt 1= “Index-Organized File”

• Actual data records are stored in leaves.

• If this is used, index structure becomes a file organization for data records (e.g., a sorted file).

• 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.
Q: How many levels if B leaf blocks and a fanout of F?

A: $\log_F B$

<table>
<thead>
<tr>
<th># Leaf Blocks</th>
<th>Fanout</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,000</td>
<td>300</td>
<td>3</td>
</tr>
<tr>
<td>10,000</td>
<td>300</td>
<td>3</td>
</tr>
<tr>
<td>100,000</td>
<td>300</td>
<td>4</td>
</tr>
<tr>
<td>1,000,000</td>
<td>300</td>
<td>4</td>
</tr>
<tr>
<td>10,000,000</td>
<td>300</td>
<td>4</td>
</tr>
<tr>
<td>100,000,000</td>
<td>300</td>
<td>5</td>
</tr>
</tbody>
</table>

16KB pages, 67% full and 100 byte records = approx 100 recs/page.

so, can store 10B rows with 5 levels.

Note: All pages at all levels are: “Slotted Pages”
<table>
<thead>
<tr>
<th>Operation</th>
<th>Heap File</th>
<th>Sorted File (100% Occupancy)</th>
<th>Index-Organized File (67% Occupancy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan all records</td>
<td>BD</td>
<td>BD</td>
<td>1.5 BD (bcos 67% full)</td>
</tr>
<tr>
<td>Equality Search</td>
<td>0.5 BD</td>
<td>(log₂ B) * D</td>
<td>(logₐ 1.5B) * D</td>
</tr>
<tr>
<td>unique key</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Range Search</td>
<td>BD</td>
<td>[(log₂ B) + #match pg]*D</td>
<td>((logₐ 1.5B) + #match pg)*D</td>
</tr>
<tr>
<td>Insert</td>
<td>2D</td>
<td>((log₂B)+B)D</td>
<td>((logₐ 1.5B)+1)D</td>
</tr>
<tr>
<td>Delete</td>
<td>(0.5B+1)D</td>
<td>((log₂B)+B)D (because rd,wrt 0.5 file)</td>
<td>((logₐ 1.5B)+1)D</td>
</tr>
</tbody>
</table>

**Operation Cost**

- **B**: The size of the data (in pages)
- **R**: Number of records per page
- **D**: (Average) time to read or write disk page
Alternatives for Data Entries (Contd.)

Alternative 2

\{<k, rid of a matching data record>\}

and Alternative 3

\{<k, \{rids of all matching data records\}>\}

• Easier to maintain than Index-Organized.
  • On the other hand: Index-organized could be faster for reads.

• If more than one index is required on a given file, at most one index can use Alt 1; rest must use 2 or 3.

• Alt 3 more compact than Alt 2, but has \textit{variable sized} data entries even with fixed-length search keys

• Even worse, for large rid lists the data entry would have to span multiple blocks!
Clustered vs. Unclustered Index

“Clustered” Index: the order of data records is the same as, or `close to’, the order of index data entries.

• 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!

• Index-organized implies clustered but not vice-versa.
  • In other words, alt-1 is always clustered
  • alt 2 and alt 3 may or may not be clustered.
Example: Alt 2 index for a Heap File

For alts 2 or 3, typically two files – one for data records and one for the index.

For an unclustered index, the order of data records in the data file is unrelated to the order of the data entries in the leaf level of the index.
Example: Alt 2 index for a Heap File

For a clustered index:

• Sort the heap file on the search key column(s)
  – Leave some free space on pages for future inserts

• Build the index

• Use overflow pages in data file if necessary
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
  – Maintenance cost (pay on the fly or be lazy with reorganization)
  – Can only cluster according to a single order
**Operation Cost**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Unclustered Alt-2 Tree Idx</th>
<th>Clustered Alt-2 Tree Idx</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Index file: 67% occupancy)</td>
<td>(Index and Data files:</td>
</tr>
<tr>
<td></td>
<td>(Data file: 100% occupancy)</td>
<td></td>
</tr>
<tr>
<td>Scan all records</td>
<td>BD (ignore index)</td>
<td>1.5 BD (ignore index)</td>
</tr>
<tr>
<td>Equality Search</td>
<td>$(1 + \log_{\frac{B}{2}} B) \times D$</td>
<td>$(1 + \log_{\frac{B}{2}} B) \times D$</td>
</tr>
<tr>
<td>unique key</td>
<td>assume an index entry is 1/3 the size of a record so index leaf level = $0.33 \times 1.5B = 0.5B$</td>
<td></td>
</tr>
<tr>
<td>Range Search</td>
<td>$[\log_{\frac{B}{2}} B + \text{matching_leaf_pages} - 1 + \text{matches_records}] \times D$</td>
<td>$[\log_{\frac{B}{2}} B + \text{matching_pages}] \times D$</td>
</tr>
<tr>
<td>Insert</td>
<td>$((\log_{\frac{B}{2}} B) + 3)D$</td>
<td>$((\log_{\frac{B}{2}} B) + 3)D$</td>
</tr>
<tr>
<td>Delete</td>
<td>same as insert</td>
<td>same as insert</td>
</tr>
</tbody>
</table>

**Variables:**

- **B:** The size of the data (in pages)
- **D:** (Avg) time to read or write disk page
Composite Search Keys

- Search on a combination of fields.
  - Equality query: Every field value is equal to a constant value. E.g. wrt \(<\text{age}, \text{sal}\>\) 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!
Index Classification Revisited

1. Selections (lookups) supported
2. Representation of data entries in index
   – what kind of info is the index actually storing?
   – 3 alternatives here
3. Clustered vs. Unclustered Indexes
4. Single Key vs. Composite Indexes
5. Tree-based, hash-based, other