CPSC 421
Database Management Systems

Lecture 13:
More on Indexing

* Some material adapted from R. Ramakrishnan, L. Delcambre, and B. Ludaescher

Agenda

• Quiz
• Assignment 5
• B+ Tree update
• More on indexes
**B+ Tree: Most widely used index**

- Ensures the tree stays balanced
  - Insert, delete, search are $O(\log_f n)$
  - Where $f$ denotes the “fanout”
- Minimum **50% occupancy** (except for root)
  - Each node contains $d \leq m \leq 2d$ entries
  - The parameter $d$ is called the order of the tree
- Maintains a **doubly linked list** of data-entry pages
- Supports equality and range searches efficiently
Searching a B+ Tree

- How many page I/Os are required to find
  - Entry 24*
  - Entry 30*?
  - All data entries > 15* and < 30*?

Updates to a B+ Tree

- Insert/modify/delete require finding data entry in leaf
- When inserting if the page is full it is SPLIT
  - One entry is added to parent
  - Changes “bubble” up the tree (if parent is full)
  - Root case is special
  - This maintains the tree balance
- Combine (merge) pages on delete to maintain 50% full constraint
  - Changes also “bubble” up the tree
**Inserting into a B+ tree**

Insert $k^*$ into B+ Tree

- Find correct leaf $L$ for $k^*$
- Add $k^*$ to $L$
  - If $L$ has enough space then done!
  - Else, split $L$ into $L$ and $L_{new}$
    - Redistribute entries evenly and copy up middle key
    - Insert index entry pointing to $L_{new}$ into parent of $L$

- This can happen recursively
  - If index node (parent of $L$) is full
    - Split index node, redistribute entries evenly, and push up middle key

- Splits “grow” the tree, root split increases height
  - Tree gets wider or one level taller at top

---

**Inserting 10$^*$ into a B+ tree**

Leaf is full

Index node has space

Copy up

Split leaf
Deleting from a B+ tree

Delete \( k^* \) from B+ Tree

• Find leaf \( L \) with \( k^* \)
• Remove \( k^* \) from \( L \)
  – If \( L \) is at least 50% full, then done!
  – Else, \( L \) has only \( d-1 \) entries
    • Try to redistribute, i.e., borrow entries from sibling leaves
    • If re-distribution fails, merge \( L \) and sibling

• If merge occurred, must delete entry (pointing to \( L \) or sibling) from parent of \( L \)
• Merge could propagate to root, decreasing height

Deleting 7* into a B+ tree

Leaf is 50% full

Leaf < 50% full
Merge siblings
Remove index entry 13

Keep going, recursively on index entries
Hash Indexes (Hashing)

• Again, 3 alternatives for data entry $k^*$
  – Data record with key value $k$
  – $<k, \text{rid of record having } k>$
  – $<k, \text{list of rids of records having } k>$

• Hash-based indexes are best for equality selections
  – Cannot support range searches

• Static and dynamic hashing techniques
  – Similar trade-offs for ISAM vs. B+ trees

Static Hashing

• Number of primary pages fixed
  – Allocated sequentially
  – Never de-allocated
  – Overflow pages if needed

• A hash function $h : K \rightarrow \text{Int}$
  – $h(k) \mod M = \text{bucket data entry for key } k$
    – ($M = \# \text{ of buckets}$)

\[
\begin{array}{c}
| h(0) | h(1) | h(2) | \ldots | h(M-1) |\\
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>k1^*</td>
<td>k2^*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\end{array}
\]

\text{Primary bucket pages} \quad \text{Overflow pages (buckets)}

key $\rightarrow h$
Static Hashing

- Buckets contain \textit{data entries}
- Hash function on search key field of records
  - Distributes values over range $0 \ldots M - 1$
- Overflow chains can develop & degrade performance
  - \textit{Extendible and Linear hashing}: dynamic techniques to fix this

Note: No guarantee about distribution of data entries (keys) across buckets
More Terminology …

“**Dense Index**”

– One index entry for each data record

Search key is “Name”

```
Ashby
Basu
Bristow
Cass
Daniels
Jones
Smith
Tracy
```

Index

More Terminology …

“**Clustered Index**”

– Records are sorted based on search key

```
<Ashby, 25, 3000>
<Basu, 44, 4003>
<Bristow, 30, 2007>
<Cass, 50, 5004>
<Daniels, 22, 6003>
<Jones, 40, 6003>
<Smith, 44, 3000>
<Tracy, 33, 5004>
```

Data File
More Terminology ...

For example, we can create a *dense clustered index* — one entry per search key, records sorted on search key

Search key is “Name”

```
<table>
<thead>
<tr>
<th>Index</th>
<th>Data File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ashby</td>
<td>&lt;Ashby, 25, 3000&gt;</td>
</tr>
<tr>
<td>Basu</td>
<td>&lt;Basu, 44, 4003&gt;</td>
</tr>
<tr>
<td>Bristow</td>
<td>&lt;Bristow, 30, 2007&gt;</td>
</tr>
<tr>
<td>Cass</td>
<td>&lt;Cass, 50, 5004&gt;</td>
</tr>
<tr>
<td>Daniels</td>
<td>&lt;Daniels, 22, 6003&gt;</td>
</tr>
<tr>
<td>Jones</td>
<td>&lt;Jones, 40, 6003&gt;</td>
</tr>
<tr>
<td>Smith</td>
<td>&lt;Smith, 44, 3000&gt;</td>
</tr>
<tr>
<td>Tracy</td>
<td>&lt;Tracy, 33, 5004&gt;</td>
</tr>
</tbody>
</table>
```

More Terminology ...

“*Sparse Index*”

— one index entry per page of data

Search key is “Name”

```
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ashby</td>
</tr>
<tr>
<td>Cass</td>
</tr>
<tr>
<td>Smith</td>
</tr>
</tbody>
</table>
```
More Terminology ...

For example, we can create a **sparse clustered** index
- One entry per data page, records sorted on search key

<table>
<thead>
<tr>
<th>Search key is “Name”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ashby, 25, 3000</td>
</tr>
<tr>
<td>Basu, 44, 4003</td>
</tr>
<tr>
<td>Bristow, 30, 2007</td>
</tr>
<tr>
<td>Cass, 50, 5004</td>
</tr>
<tr>
<td>Daniels, 22, 6003</td>
</tr>
<tr>
<td>Jones, 40, 6003</td>
</tr>
<tr>
<td>Smith, 44, 3000</td>
</tr>
<tr>
<td>Tracy, 33, 5004</td>
</tr>
</tbody>
</table>

Data File

More Terminology ...

**“Unclustered Index”**
- Records NOT sorted on search key

If the search key is “Age” ...

This becomes an unclustered index for Age

<table>
<thead>
<tr>
<th>If the search key is “Age”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ashby, 25, 3000</td>
</tr>
<tr>
<td>Basu, 44, 4003</td>
</tr>
<tr>
<td>Bristow, 30, 2007</td>
</tr>
<tr>
<td>Cass, 50, 5004</td>
</tr>
<tr>
<td>Daniels, 22, 6003</td>
</tr>
<tr>
<td>Jones, 40, 6003</td>
</tr>
<tr>
<td>Smith, 44, 3000</td>
</tr>
<tr>
<td>Tracy, 33, 5004</td>
</tr>
</tbody>
</table>

Data File
Combinations …

Can we define

- A **sparse unclustered** index?
  - NO!
  - This only works if the index is clustered (i.e., sorted on the search key)

- A **dense unclustered** index?
  - YES!
  - This induces a sort over the file on the search key
  - The sort “lives” in the index
  - Like an “old fashioned” card catalog
  - Sometimes referred to as a “secondary index” (also refers to an index created on one or more non-key attributes)

More Terminology …

For example, we can create a dense unclustered index

- *One entry per search key, records NOT sorted on search key*
I/O cost of using dense unclustered index

- For a range query:
  - Every data record costs I/O
  - You may have to re-read some pages
  - Cost to scan file in sort order equal to number of records in the file!
  - (this is really bad, which is why we'll also talk about external sorting)

Using Composite Search Keys

Which indexes should be used to answer these queries? Why?

1. Age = 18
2. Age = 18 and Sal = 22
3. Age = 18 and Sal > 15
4. Age > 17 and Sal > 23

Using an index is (generally) cheaper if the answer forms a contiguous block of search keys in the leaf nodes

index + block < block scan
Indexes (almost) final words

• A DBMS often creates **clustered indexes** on primary keys
  – i.e., the file is sorted on the declared keys
  – Since primary keys must be the values used in foreign keys
  … these indexes used in joins (Employee.cid = Company.id)
• Only one clustered index is create per table … Why?
• As a DB designer/administrator, you must determine whether you need additional (unclustered) indexes
  – Including composite indexes
  – You have to consider trade-offs between query & update
  – This is one reason why DBAs get paid so well!

For Thursday

• Reading
  – Ch. 8
  – Ch. 10: Intro, 10.1-10.6

• Homework
  – Should be working on project part 3
  – Start on assignment 5 (form groups)