Indexes

Auxiliary structures that speed up operations that are not supported efficiently by the basic file organization

A Simple Index Example

<table>
<thead>
<tr>
<th>Index blocks</th>
<th>Data blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
</tr>
</tbody>
</table>

About Indexes

- Index entry
  - <key, rid>
  - <key, list of rid>
  - Data record
- The majority of database indexes are designed to reduce disk access

Organization of Index Entries

- Tree-structured
  - B-tree, R-tree, Quad-tree, kd-tree, ...
- Hash-based
  - Static, dynamic
- Other
  - Bitmap, VA-file, ...

From BST to BBST to B

- Binary Search Tree
  - Worst case??
- Balance Binary Search Tree
  - E.g. AVL, Red-Black
- B-tree
  - Why not use BBST in databases??
B-tree Properties

- Each node occupies one block
- Order $n$
  - $n$ keys, $n+1$ pointers
- Nodes (except root) must be at least half full
  - Internal node: $\lceil (n+1)/2 \rceil$ pointers
  - Leaf node: $\lfloor (n+1)/2 \rfloor$ pointers
- All leaf nodes are on the same level

B-tree Operations

- Search
- Insert
- Delete

B-tree Insert

- Find the appropriate leaf
- Insert into the leaf
  - there's room → we're done
  - no room
    - split leaf node into two
    - insert a new <key,pointer> pair into leaf's parent node
- Recursively apply previous step if necessary
  - A split of current ROOT leads to a new ROOT

B-tree Insert Examples

- (a) simple case
  - space available in leaf
- (b) leaf overflow
- (c) non-leaf overflow
- (d) new root

(a) Insert key = 32

HGM Notes
(b) Insert key = 7

(c) Insert key = 160

(d) New root, insert 45

B-tree Delete

Find the appropriate leaf
Delete from the leaf
- still at least half full → we’re done
- below half full → coalescing
  • borrow a <key,pointer> from one sibling node, or
  • merge with a sibling node, and delete from a parent node
Recursively apply previous step if necessary

B-tree Delete in Practice

Coalescing is usually not implemented because it’s too hard and not worth it

Static Hash Index
Hash Function

A commonly used hash function: \( \text{hash} = k \% b \)
- \( k \) is the key value
- \( b \) is the number of buckets

Static Hash Index Example ...

- 4 buckets
- Hash function: \( \text{key} \% 4 \)
- 2 index entries per bucket block

... Static Hash Index Example

- Insert the records with the following keys: 4, 3, 7, 17, 22, 10, 25, 33

Dynamic Hashing

- Problem of static hashing??
- Dynamic hashing
  - Extendable Hash Index

Extendable Hash Index ...

- Maximum \( 2^m \) buckets
  - \( m \) is maximum depth of index
- Multiple buckets can share the same block
- Inserting a new entry to a block that is already full would cause the block to split

... Extendable Hash Index

- Each block has a local depth \( l_r \), which means that the hash values of the records in the block has the same rightmost \( l_r \) bit
- The bucket directory keeps a global depth \( d_l \), which is the highest local depth
Extendable Hash Index Example

- $M = 4$ (i.e. could have at most 16 buckets)
- Hash function: $\text{key} \% 2^4$
- 2 index entries per block
- Insert 8, 11, 4, 14

Extendable Hashing (I)

Bucket directory

Bucket blocks

- $d = 0$
- Insert 8 (i.e. 1000)
- Insert 11 (i.e. 1011)
- Insert 4 (i.e. 0100)

Extendable Hashing (II)

Bucket directory

Bucket blocks

- $d = 0$
- Insert 8 (i.e. 1000)
- Insert 11 (i.e. 1011)
- Insert 4 (i.e. 0100)

Bucket directory

Bucket blocks

- $d = 1$
- Insert 14 (i.e. 1110)

Readings

- Textbook Chapter 21.1 – 21.4