

# Tree-Structured Indexes

## Chapter 10

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 1

---

---

---

---

---

---

---

---

# Index Design Space

**Organization Structure for k\***

- **Hash-based**
  - + Equality search
- **Tree-based**
  - + Range, equality search
    - B+Tree (dynamic)
    - ISAM (static)

→ Data Entry (k\*) Contents

1. Actual Data record
  - index = file!
2. <k, rid>
  - actual records in a different file
3. <k, list of rids>

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 2

---

---

---

---

---

---

---

---

# Motivation

- Range and equality searches very common
- Can scan heap file, but this is expensive
- **Goal:** *Create a dynamic index structure that allows for efficient evaluation and equality and range queries*

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 3

---

---

---

---

---

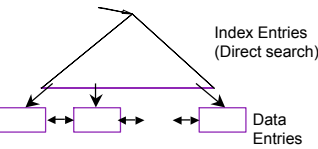
---

---

---

## B+ Tree

- Height-balanced (dynamic) tree structure
- 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.
- Supports equality and range-searches efficiently.



Index Entries (Direct search)

**Index Entries**  
Entries in the index (i.e. non-leaf) pages:  
(search key value, pageid)

Data Entries

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 4

---

---

---

---

---

---

---

---

---

---

## Example B+ Tree

- Search: Starting from root, examine index entries in non-leaf nodes, and traverse down the tree until a leaf node is reached.
  - Non-leaf nodes can be searched using a binary or a linear search.
- Search for 5\*, 15\*, all data entries  $\geq 24$ \*

Root

13	17	24	30
----	----	----	----

Height = 1

2*	3*	5*	7*	14*	16*	19*	20*	22*	24*	27*	29*	33*	34*	38*	39*
----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 5

---

---

---

---

---

---

---

---

---

---

## B+-tree Page Format

**Non-leaf Page**

index entries	
$P_1$	$K_1$
$P_2$	$K_2$
$P_3$	$\dots$
$P_m$	$K_m$
$P_{m+1}$	

↓

Pointer to a page with Values <  $K_1$

↓

Pointer to a page with values s.t.  $K_1 \leq \text{Values} < K_2$

↓

Pointer to a page with values s.t.  $K_2 \leq \text{Values} < K_3$

↓

Pointer to a page with values  $\geq K_m$

**Leaf Page**

data entries (Alternative 2)	
$P_0$	$R_1$
$K_1$	$R_2$
$\dots$	$\dots$
$R_n$	$K_n$
$P_{n+1}$	

←

Prev Page Pointer

↓

record 1

↓

record 2

↓

record m

↓

Next Page Pointer

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 6

---

---

---

---

---

---

---

---

---

---

## B+ Trees

- What is the height of a B+ tree?
  - Fanout  $F$  (average number of children for non-leaf node)
  - $N$  total leaf pages

$\log_F N$

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 7

---

---

---

---

---

---

---

---

## B+ Trees in Practice

- Typical order 100. Typical fill-factor 67%.
  - Average fanout = 133
- Typical capacity:
  - Height = 1: 133 pages of data entries (leaf pages)
  - Height = 2:  $133^2$  pages of data entries
  - Height = 3:  $133^3$  (> 2 million) pages of data entries
  - Height = 4:  $133^4$  (> 300 million) pages of data entries
- Can often keep top levels of index in buffer pool
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBytes

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 8

---

---

---

---

---

---

---

---

## Exercise

- You are given a file of 10 million records
- Suppose you can store 10 data entries per leaf page
- You build a B+ Tree with order 100, 67% average fill-factor
  - Fanout = 133
- What is the height of your B+ Tree?

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 9

---

---

---

---

---

---

---

---

## A Note on Order

- *Order (d)* concept replaced by physical space criterion in practice (*at least half-full*).
  - Index (i.e. non-leaf) pages can typically hold many more entries than leaf pages.
    - Leaf pages could have actual data records
  - Variable sized records and search keys mean different nodes will contain different numbers of entries.
  - Even with fixed length fields, multiple records with the same search key value (*duplicates*) can lead to variable-sized data entries (if we use Alternative (3)).

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 10

---

---

---

---

---

---

---

---

## B+ Tree Operations

- Search
  - Equality
  - Range
- **Insert data entry**
- Delete data entry
- Bulk load

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 11

---

---

---

---

---

---

---

---

## B+-Tree: Inserting a Data Entry

- Find correct leaf *L*.
- Put data entry onto *L*.
  - If *L* has enough space, *done!*
  - Else, must *split L* (into *L* and a new node *L2*)
    - Redistribute entries evenly, **copy up** middle key.
    - Insert index entry pointing to *L2* into parent of *L*.
- This can happen recursively
  - To **split index node**, redistribute entries evenly, but **push up** middle key. (Contrast with leaf splits.)
- Splits "grow" tree; root split increases height.
  - Tree growth: gets *wider* or *one level taller at top*.

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 12

---

---

---

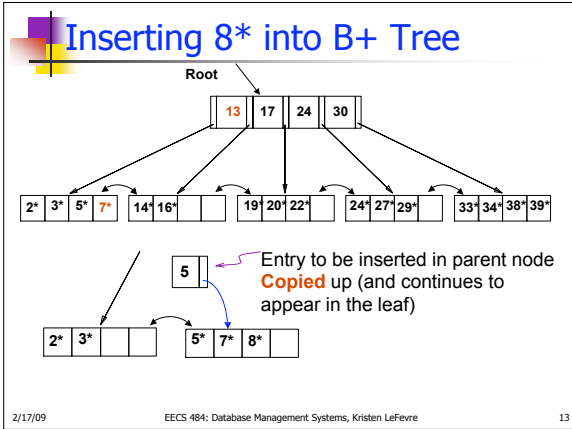
---

---

---

---

---




---

---

---

---

---

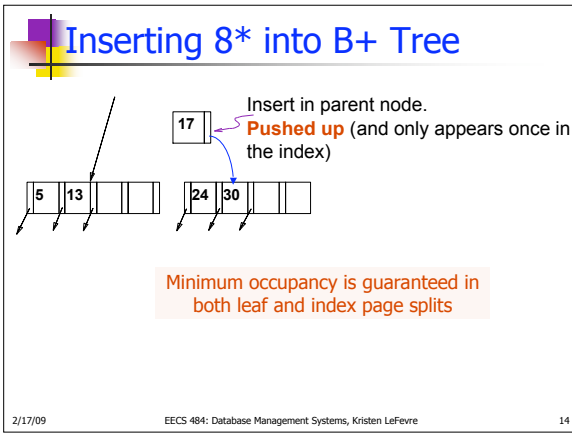
---

---

---

---

---




---

---

---

---

---

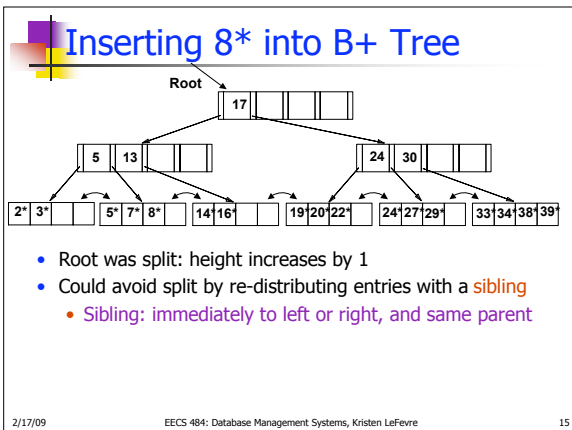
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

## Inserting 8\* into B+ Tree

Root

8 17 24 30

2\* 3\* 5\* 7\* 8\* 14\* 16\* 19 20 22\* 24 27 29\* 33\* 34\* 38\* 39\*

- Re-distributing entries with a **sibling**
  - Improves page occupancy
  - Usually not used for non-leaf node splits. Why?
    - Increases I/O, especially if we check both siblings
    - Better if split propagates up the tree (rare)
    - Use only for leaf level entries as we have to set pointers

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 16

---

---

---

---

---

---

---

---

---

---

## B+ Tree Operations

- Search
  - Equality
  - Range
- Insert data entry
- **Delete data entry**
- Bulk load

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 17

---

---

---

---

---

---

---

---

---

---

## B+-Tree: Deleting a Data Entry

- Start at root, find leaf  $L$  where entry belongs.
- Remove the entry.
  - If  $L$  is at least half-full, *done!*
  - If  $L$  has only  $d-1$  entries,
    - Try to **re-distribute**, borrowing from *sibling* (adjacent node with same parent as  $L$ ).
    - 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.

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 18

---

---

---

---

---

---

---

---

---

---

## Deleting 22\* and 20\*

- Deleting 22\* is easy.
- Deleting 20\* is done with re-distribution. Notice how middle key is *copied up*.

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 19

---

---

---

---

---

---

---

---

## ... And Then Deleting 24\*

- Must merge.
- In the non-leaf node, *toss the index entry with key value = 27*

Can this merge?

■ Pull down of index entry

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 20

---

---

---

---

---

---

---

---

## Non-leaf Re-distribution

- Tree *during deletion* of 24\*.
- Can re-distribute entry from left child of root to right child.

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 21

---

---

---

---

---

---

---

---

## After Re-distribution

- Rotate through the parent node
- It suffices to re-distribute index entry with key 20; For illustration 17 also re-distributed

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 22

---

---

---

---

---

---

---

---

## B+-Tree Deletion

- Try redistribution with **all** siblings first, then merge. Why?
  - Good chance that redistribution is possible (large fanout!)
  - Only need to propagate changes to parent node
  - Files typically grow not shrink!

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 23

---

---

---

---

---

---

---

---

## B+ Tree Operations

- Search
  - Equality
  - Range
- Insert data entry
- Delete data entry
- **Bulk load**

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 24

---

---

---

---

---

---

---

---

## Bulk Loading

- Option 1: multiple inserts.
  - Slow. Repeated Re-organization.
  - Does not give sequential storage of leaves.
- Option 2: *Bulk Loading*
  - Fewer I/Os during build.
  - Leaves will be stored sequentially (and linked, of course).
  - Can control "fill factor" on pages.

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 25

---

---

---

---

---

---

---

---

## Bulk Loading of a B+ Tree

- Initialization:* Sort all data entries, insert pointer to first (leaf) page in a new (root) page.

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 26

---

---

---

---

---

---

---

---

## Bulk Loading (Contd.)

- Index entries for leaf pages always entered into right-most index page just above leaf level. When this fills up, it splits. (Split may go up right-most path to the root.)
- Much faster than repeated inserts!

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 27

---

---

---

---

---

---

---

---

## Summary

- Tree-structured indexes are ideal for range-searches, also good for equality searches.
- B+ tree is a dynamic height-balanced index structure.
  - Inserts/deletes/search costs  $O(\log_F N)$ .
  - High fanout (**F**) means depth rarely more than 3 or 4.
  - Most widely used index in database management systems because of its versatility. One of the most optimized components of a DBMS.

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 28

---

---

---

---

---

---

---

---

## Announcements

- Optional Exercises: 10.1, 10.5, 10.7
- Reminder: Please fill out midterm course feedback online :)

2/17/09 EECS 484: Database Management Systems, Kristen LeFevre 29

---

---

---

---

---

---

---

---