Index: helps to retrieve data quicker

Select * FROM Emp WHERE salary = 1,000,000;
B+-Tree

- Give up “sequentiality” of index
- Still multi-level, but always achieve balance of “tree”
- Predictable performance under updates
- Automate restructuring under updates
B+Tree Example  

n=3
B+ Trees in Practice

- Typical order: 100. Typical fill-factor: 67%.
  - average fanout = 133
- Typical capacities:
  - Height 4: $133^4 = 312,900,700$ records
  - Height 3: $133^3 = 2,352,637$ records
- Can often hold top levels in buffer pool:
  - Level 1 = 1 page = 8 Kbytes
  - Level 2 = 133 pages = 1 Mbyte
  - Level 3 = 17,689 pages = 133 MBytes
Insert Data into B+ Tree

- Find correct leaf \( L \).
- Put data entry onto \( L \).
  - If \( L \) has enough space, \textit{done}!
  - Else, must \textit{split} \( L \) (into \( L \) and a new node \( L_2 \))
    - Redistribute entries evenly, \textit{copy up} middle key.
    - Insert index entry pointing to \( L_2 \) into parent of \( L \).

- This can happen recursively
  - To split index node, redistribute entries evenly, but \textit{push up} middle key. (Contrast with leaf splits.)

- Splits “grow” tree; root split increases height.
  - Tree growth: gets \textit{wider} or \textit{one level taller at top}. 
Insert into B+tree

(a) simple case
   - space available in leaf
(b) leaf overflow
(c) non-leaf overflow
(d) new root
(a) Insert key = 32

```
  100
   /\   \\
 30  /   \\
 /     \\
3 5 11
```

```
  30
 /  \   \\
30  /   \\
/     \\
30 31 32
```
(a) Insert key = 7

\[ n=3 \]
(c) Insert key = 160
(d) New root, insert 45

\[
\begin{array}{c}
1 & 2 & 3 \\
10 & 12 \\
20 & 25 \\
30 & 32 & 40 \\
40 & 45
\end{array}
\]
Delete Data from B+ Tree

- 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.
Deletion from B+tree

(a) Simple case
(b) Coalesce with neighbor (sibling)
(c) Re-distribute keys
(d) Cases (b) or (c) at non-leaf
(a) Delete key = 11

```
3 5 11
30 31
```

```
30
```
```
100
```
(b) Coalesce with sibling
   – Delete 50
(c) Redistribute keys

- Delete 50

\[ n=4 \]
(d) Non-leaf coalesce
– Delete 37

new root

n=4
B+tree deletions in practice

– Often, coalescing is not implemented
  – Too hard and not worth it!