Lecture 20: Indexes

Monday, February 27, 2006

1

Outline

- Data storage
- Index structures (13.1, 13.2)
- B-trees (13.3)

2

Data Storage

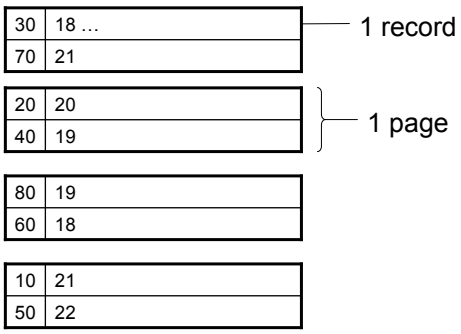
- For persistence, data is stored on disk
- Basic abstraction
  - *Collection of records* or *file*
  - Typically, 1 relation = 1 file
- A file consists of one or more *pages*
  - One page corresponds to one block
  - *Page is unit of info read/written to/from disk*
- How to organize records in file?

3

Heap File

File is not sorted on any attribute

`Student(sid: int, age: int, …)`



4

Heap File Example

- 10,000 students
- 10 student records per page
- Total number of pages: 1,000 pages
- Find student whose sid is 80
  - Must read on average 500 pages
- Find all students older than 20
  - Must read all 1,000 pages
- Can we do better?

5

Sequential File

File sorted on an attribute, usually on primary key
`Student(sid: int, age: int, …)`

10	21 …
20	20

30	18
40	19

50	22
60	18

70	21
80	19

6

Sequential File Example

- Total number of pages: 1,000 pages
- Find student whose sid is 80
  - Could do binary search, read $\log_2(1{,}000) \approx 10$ pages
- Find all students older than 20
  - Must still read all 1,000 pages
- Can we do even better?

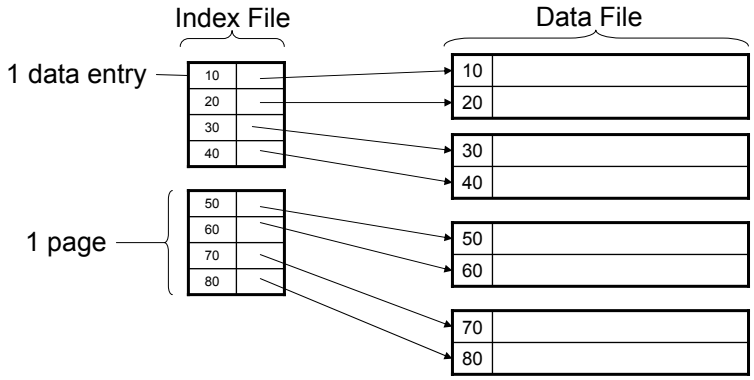
7

Indexes

- An *index* on a file speeds up selections on the *search key fields* for the index.
  - Any subset of the fields of a relation can be the search key for an index on the relation.
  - *Search key* is not the same as *key* (minimal set of fields that uniquely identify a record in a relation).
- An index contains a collection of *data entries*, and supports efficient retrieval of all data entries with a given key value **k**.

Primary Index

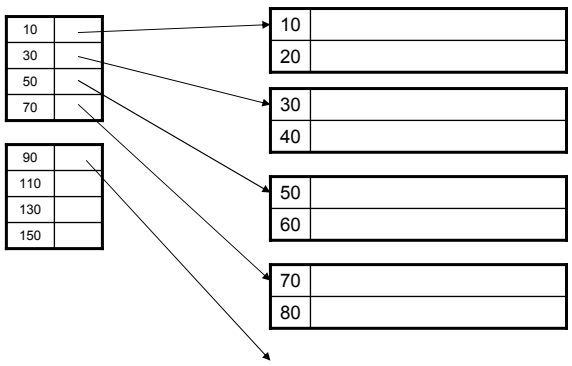
- File is sorted on the index attribute
- *Dense* index: sequence of (key,pointer) pairs



9

Primary Index

- *Sparse* index



10

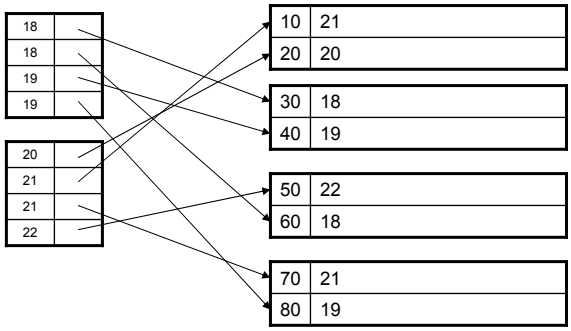
Primary Index Example

- Let's assume all pages of index fit in memory
- Find student whose sid is 80
  - Index (dense or sparse) points directly to the page
  - Only need to read 1 page from disk.
- Find all students older than 20
  - Must still read all 1,000 pages.
- How can we make *both* queries fast?

11

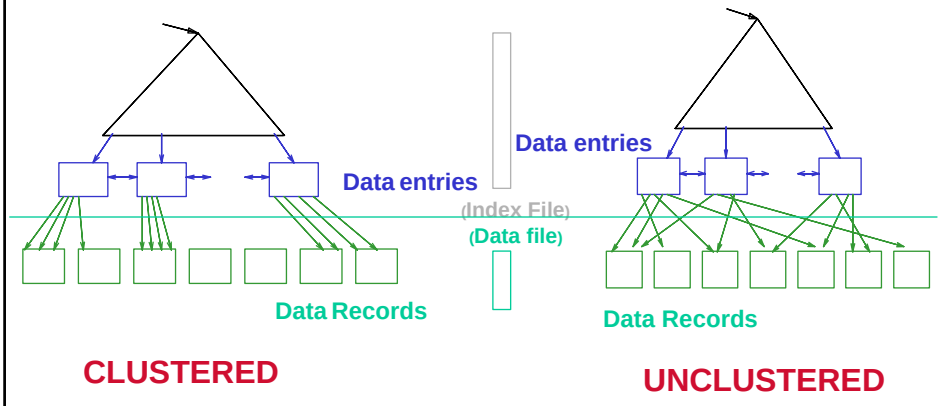
Secondary Indexes

- To index other attributes than primary key
- Always dense (why ?)



12

Clustered vs. Unclustered Index



Clustered/Unclustered

- Primary index = usually clustered
- Secondary indexes = usually unclustered

14

Secondary Indexes

- Applications
  - Index other attributes than primary key
  - Index unsorted files (heap files)
  - Index clustered data

15

Index Classification Summary

- Primary/secondary
  - Primary = may reorder data according to index
  - Secondary = cannot reorder data
- Dense/sparse
  - Dense = every key in the data appears in the index
  - Sparse = the index contains only some keys
- Clustered/unclustered
  - Clustered = records close in index are close in data
  - Unclustered = records close in index may be far in data
- B+ tree / Hash table / …

16

Large Indexes

- What if index does not fit in memory?
- Would like to index the index itself
- How many index levels do we need?
- Can we create them automatically?
  Yes!
- Can do something even more powerful!

17

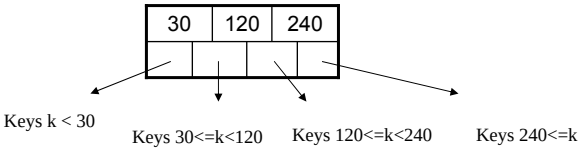
B+ Trees

- Search trees
- Idea in B Trees
  - Make 1 node = 1 page (= 1 block)
- Idea in B+ Trees
  - Make leaves into a linked list (range queries are easier)

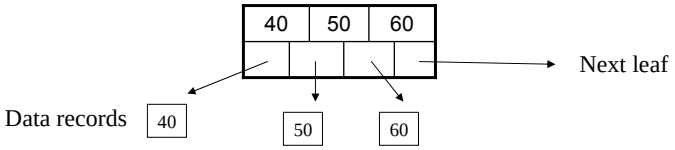
18

B+ Trees Basics

- Parameter d = the *degree*
- Each node has >= d and <= 2d keys (except root)



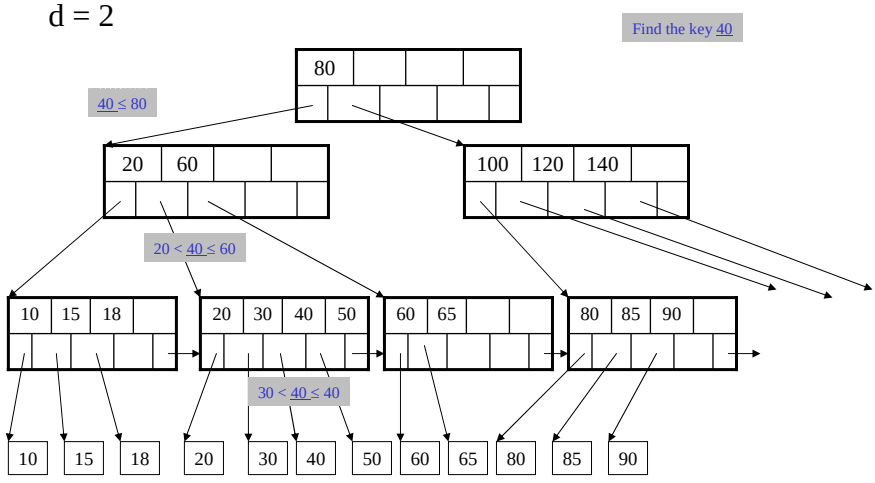
- Each leaf has >=d and <= 2d keys:



19

B+ Tree Example

d = 2



20

Searching a B+ Tree

- Exact key values:
  - Start at the root
  - Proceed down, to the leaf

```
Select name
From Student
Where age = 25
```

- Range queries:
  - Find lowest bound as above
  - Then sequential traversal

```
Select name
From Student
Where 20 <= age
  and  age <= 30
```

21

B+ Tree Design

- How large d ?
- Example:
  - Key size = 4 bytes
  - Pointer size = 8 bytes
  - Block size = 4096 bytes
- 2d x 4 + (2d+1) x 8 <= 4096
- d = 170

22

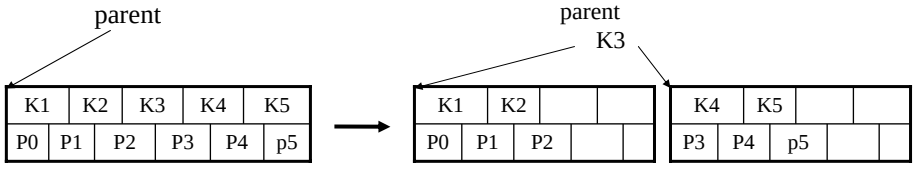
B+ Trees in Practice

- 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

Insertion in a B+ Tree

Insert (K, P)

- Find leaf where K belongs, insert
- If no overflow (2d keys or less), halt
- If overflow (2d+1 keys), split node, insert in parent:

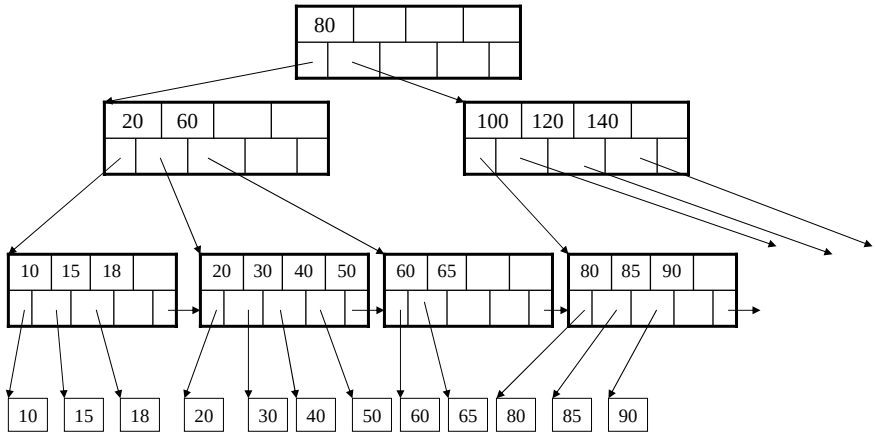


- If leaf, also keep K3 in right node
- When root splits, new root has 1 key only

24

Insertion in a B+ Tree

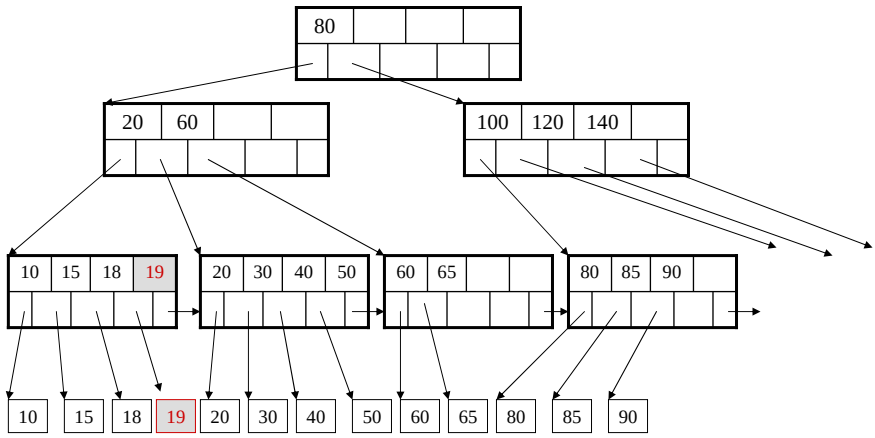
Insert K=19



25

Insertion in a B+ Tree

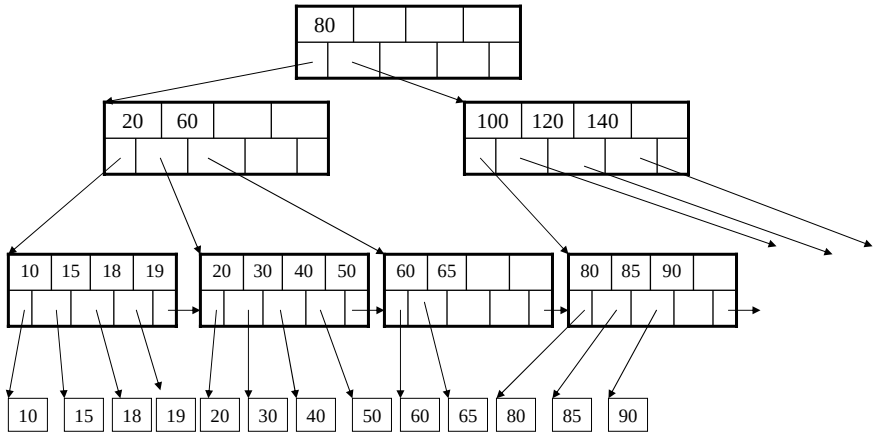
After insertion



26

Insertion in a B+ Tree

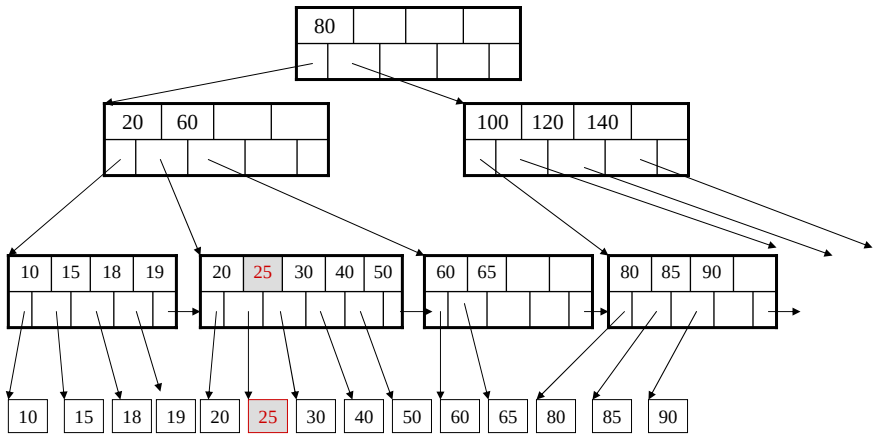
Now insert 25



27

Insertion in a B+ Tree

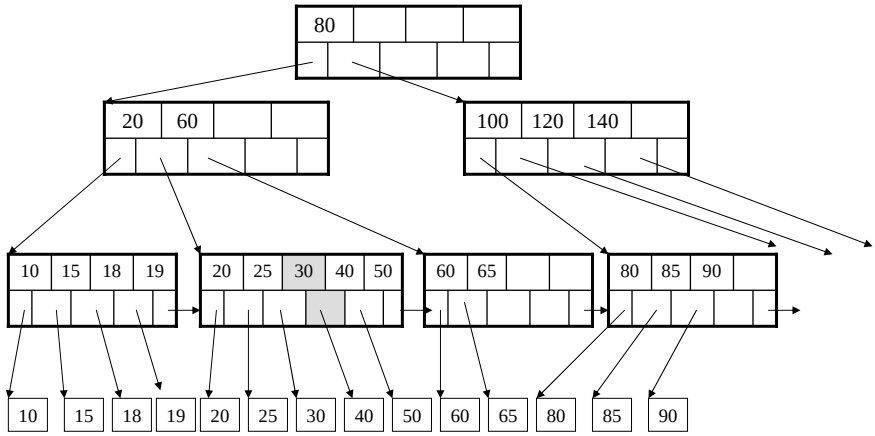
After insertion



28

Insertion in a B+ Tree

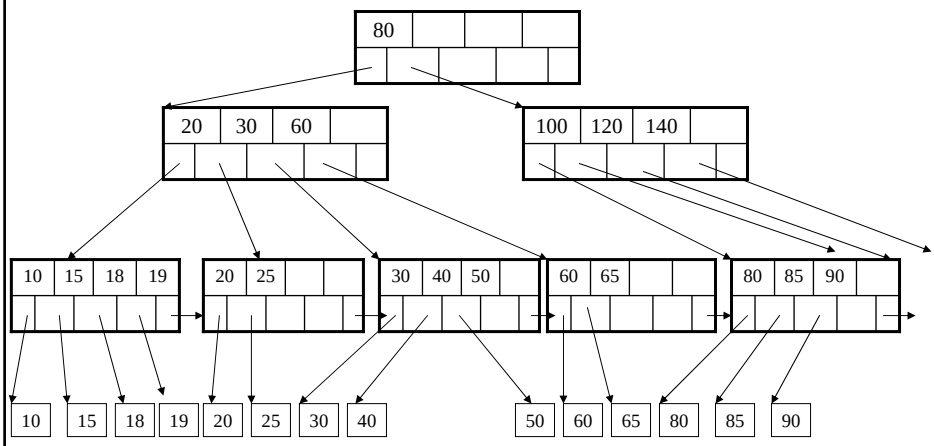
But now have to split !



29

Insertion in a B+ Tree

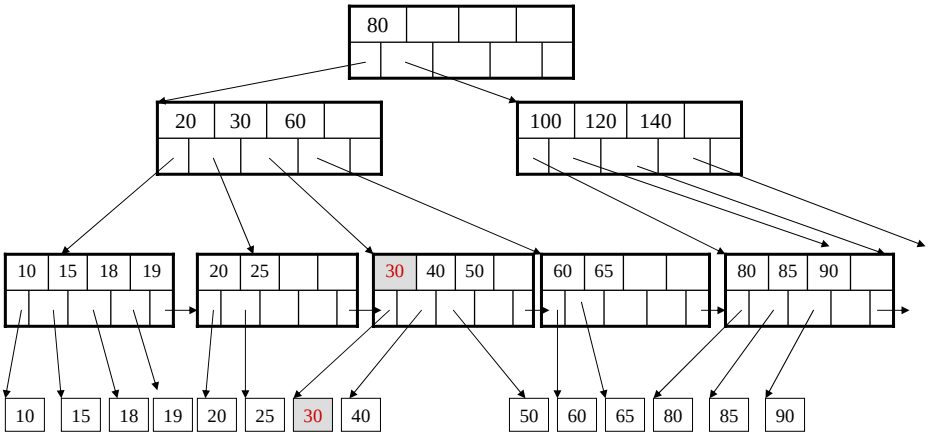
After the split



30

Deletion from a B+ Tree

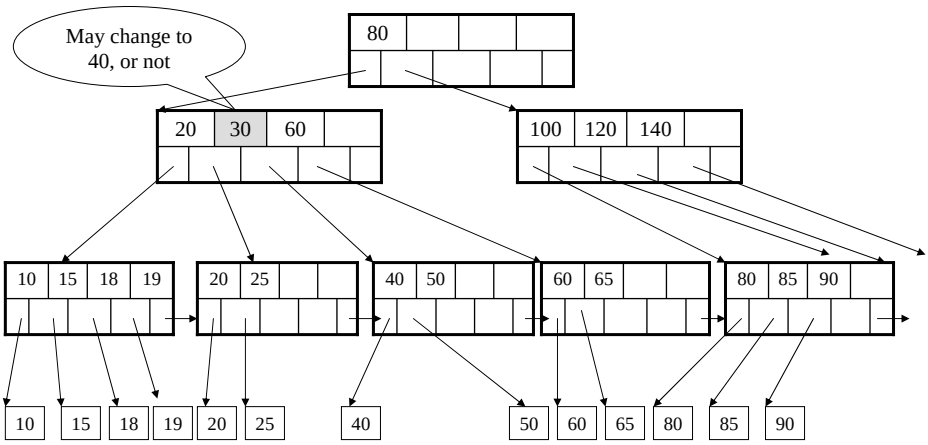
Delete 30



31

Deletion from a B+ Tree

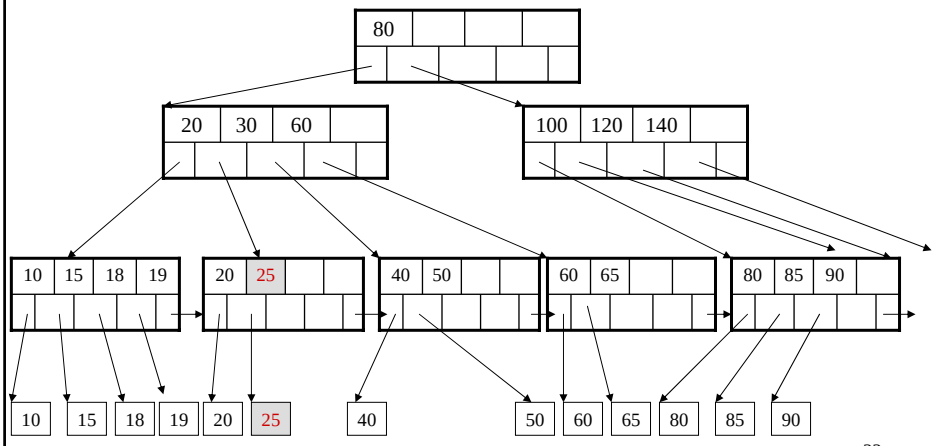
After deleting 30



32

Deletion from a B+ Tree

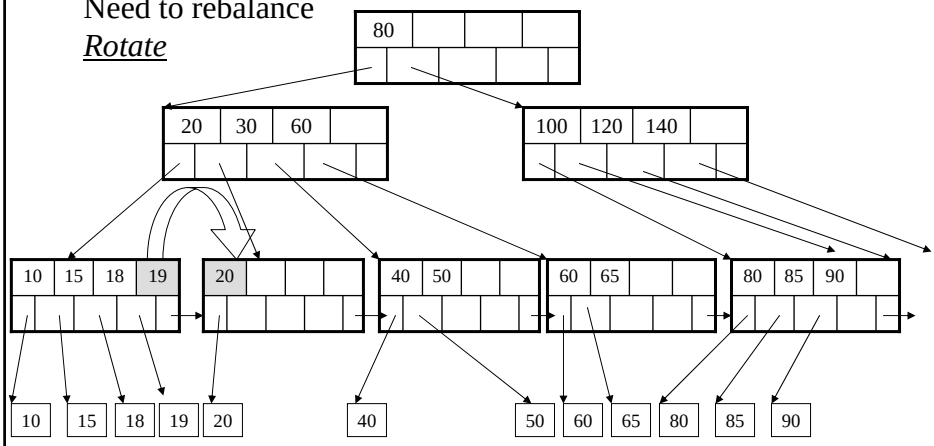
Now delete 25



33

Deletion from a B+ Tree

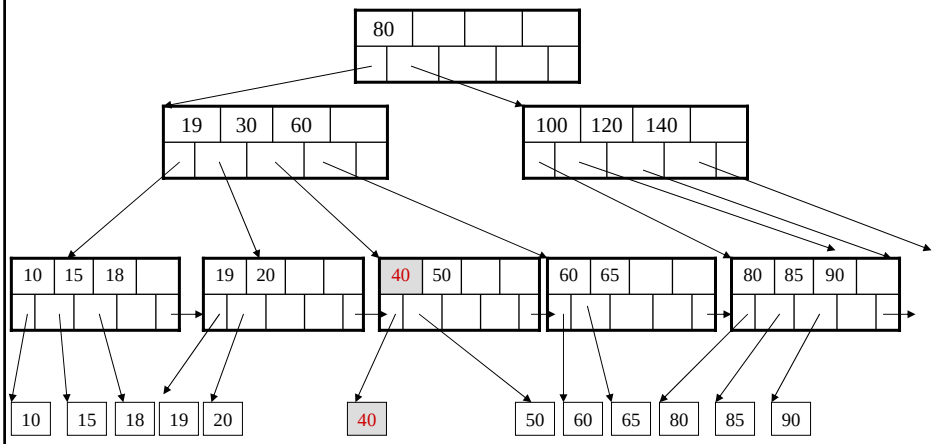
After deleting 25
Need to rebalance
Rotate



34

Deletion from a B+ Tree

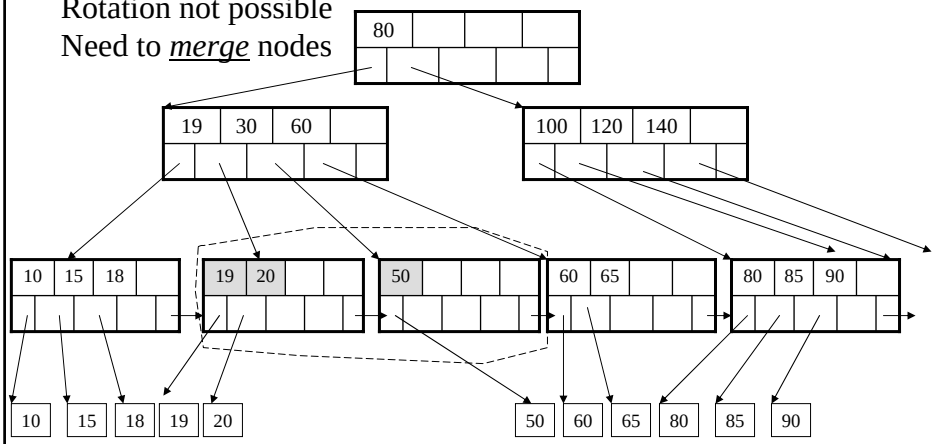
Now delete 40



35

Deletion from a B+ Tree

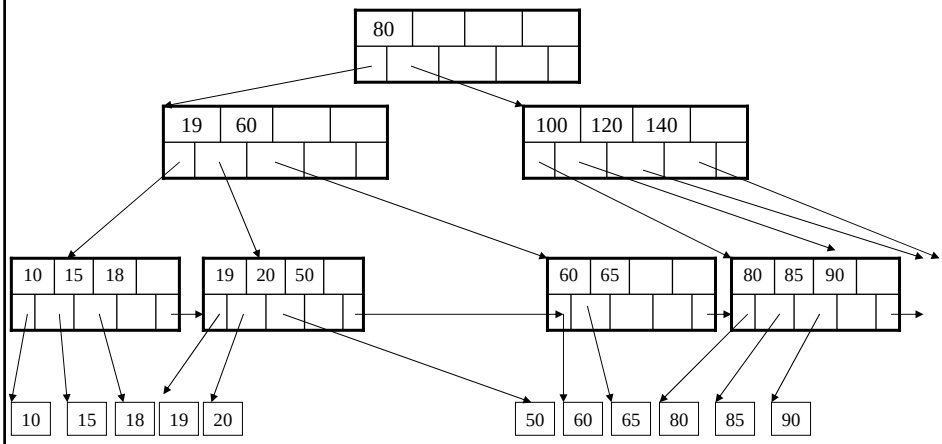
After deleting 40
Rotation not possible
Need to *merge* nodes



36

Deletion from a B+ Tree

Final tree



37

Summary on B+ Trees

- Default index structure on most DBMS
- Very effective at answering 'point' queries:
  productName = 'gizmo'
- Effective for range queries:
  50 < price AND price < 100
- Less effective for multirange:
  50 < price < 100 AND 2 < quant < 20

38