Lecture 18: Indexes

Wednesday, November 13, 2002

Outline

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

Administrative:

- HW4 due on Monday, 11/18

Indexes

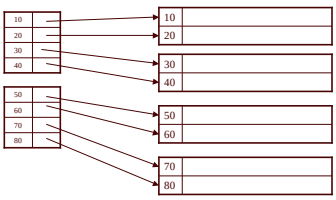
- An *index* on a file speeds up selections on the *search key fields* for the index.
  - Any subset of the fields of a relation can be the search key for an index on the relation.
  - *Search key* is not the same as *key* (minimal set of fields that uniquely identify a record in a relation).
- An index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

Index Classification

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

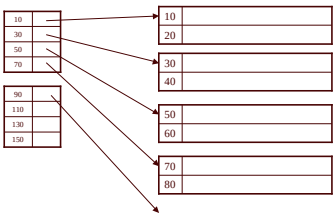
Primary Index

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



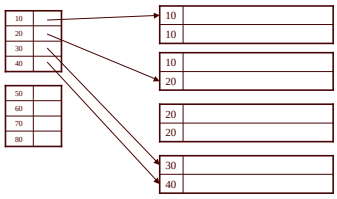
Primary Index

- *Sparse* index



Primary Index with Duplicate Keys

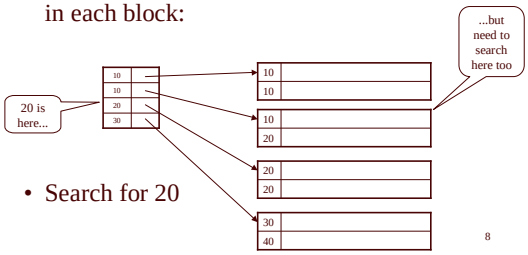
- Dense index:



7

Primary Index with Duplicate Keys

- Sparse index: pointer to lowest search key in each block:

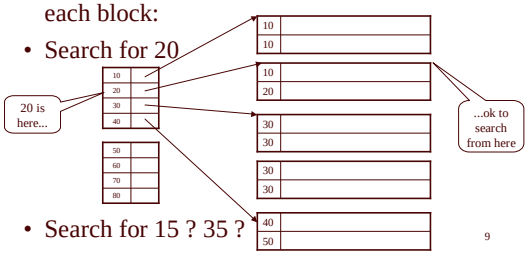


- Search for 20

8

Primary Index with Duplicate Keys

- Better: pointer to ***lowest new search key*** in each block:
- Search for 20

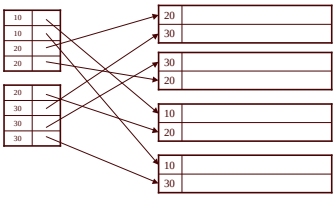


- Search for 15 ? 35 ?

9

Secondary Indexes

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



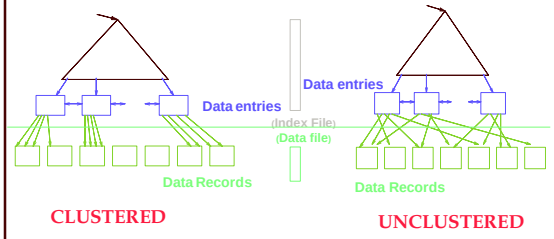
10

Clustered/Unclustered

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

11

Clustered vs. Unclustered Index



Secondary Indexes

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

13

Applications of Secondary Indexes

- Secondary indexes needed for ***heap files***
- Also for ***Clustered data:***

Company(name, city), Product(pid, maker)

```
Select city
From Company, Product
Where name=maker
   and pid="p045"
```

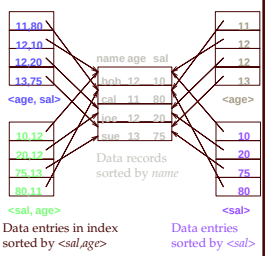
```
Select pid
From Company, Product
Where name=maker
   and city="Seattle"
```

14

Composite Search Keys

- *Composite Search Keys*: Search on a combination of fields.
  - Equality query: Every field value is equal to a constant value. E.g. wrt <sal,age> index:
    - age=20 and sal =75
  - Range query: Some field value is not a constant. E.g.:
    - age =20; or age=20 and sal > 10

Examples of composite key indexes using lexicographic order.



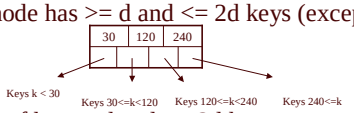
B+ Trees

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

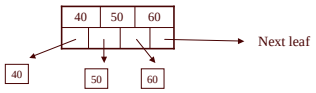
16

B+ Trees Basics

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



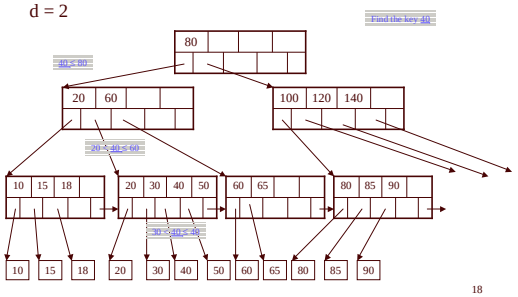
- Each leaf has >=d and <= 2d keys:



17

B+ Tree Example

d = 2



18

B+ Tree Design

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

19

Searching a B+ Tree

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

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

- Range queries:
  - As above
  - Then sequential traversal

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

20

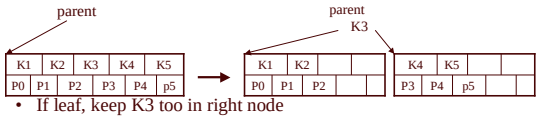
B+ Trees in Practice

- 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:

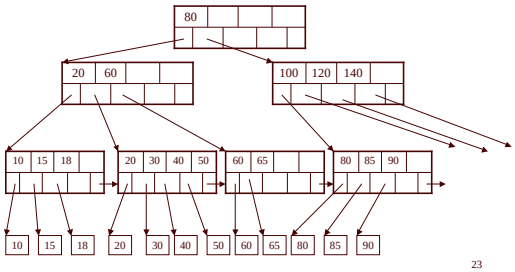


- If leaf, keep K3 too in right node
- When root splits, new root has 1 key only

22

Insertion in a B+ Tree

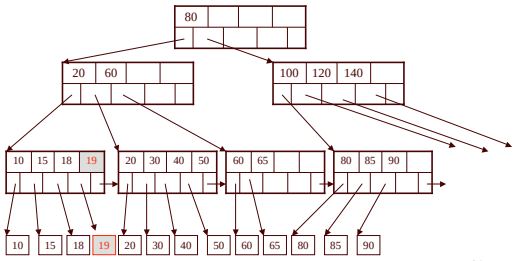
Insert K=19



23

Insertion in a B+ Tree

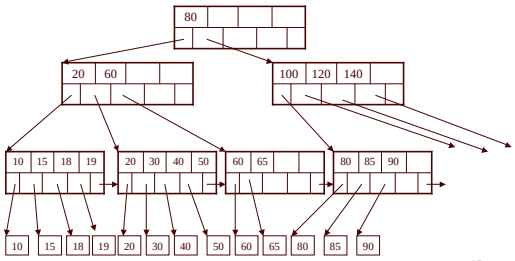
After insertion



24

Insertion in a B+ Tree

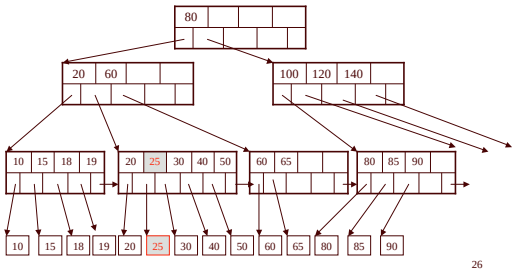
Now insert 25



25

Insertion in a B+ Tree

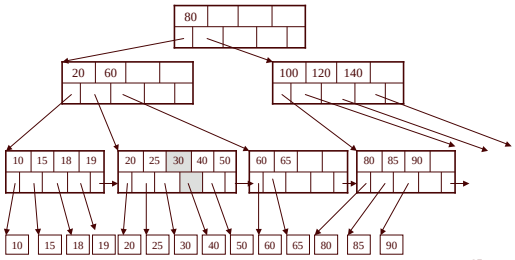
After insertion



26

Insertion in a B+ Tree

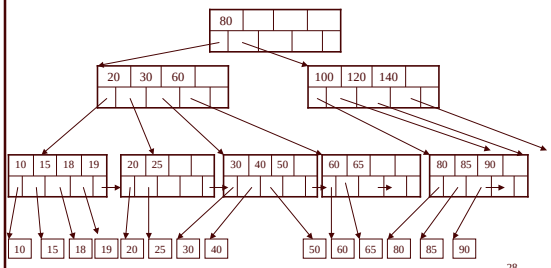
But now have to split !



27

Insertion in a B+ Tree

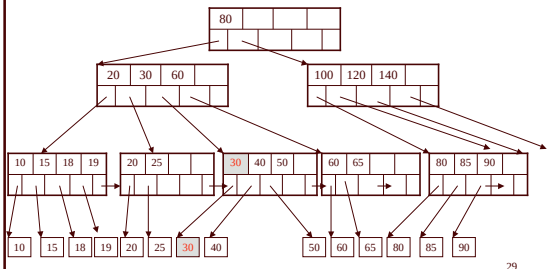
After the split



28

Deletion from a B+ Tree

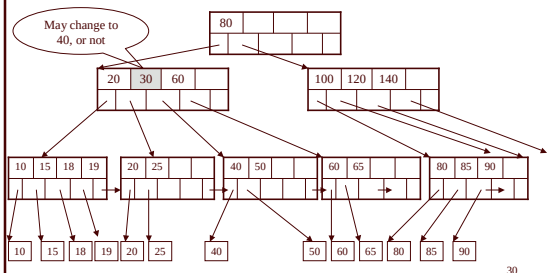
Delete 30



29

Deletion from a B+ Tree

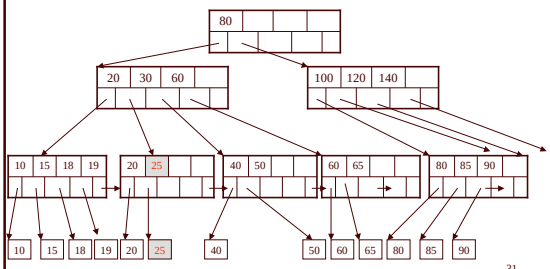
After deleting 30



30

Deletion from a B+ Tree

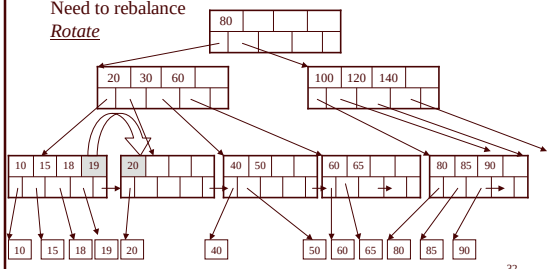
Now delete 25



31

Deletion from a B+ Tree

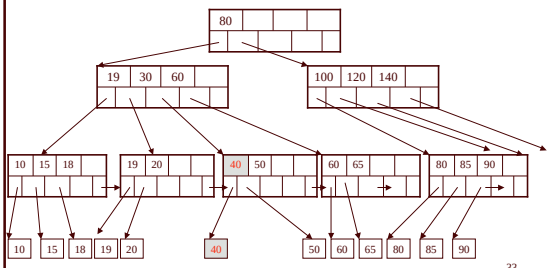
After deleting 25
Need to rebalance
Rotate



32

Deletion from a B+ Tree

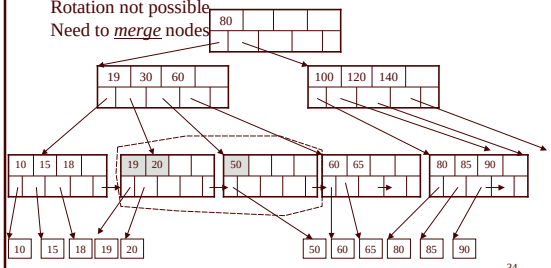
Now delete 40



33

Deletion from a B+ Tree

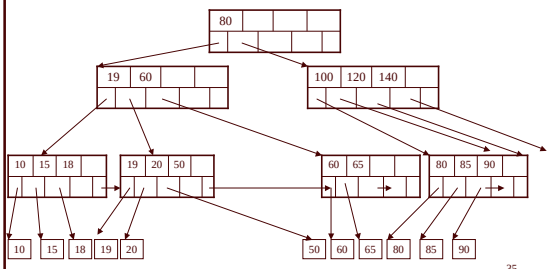
After deleting 40
Rotation not possible
Need to *merge* nodes



34

Deletion from a B+ Tree

Final tree



35