CIS 560 – Database System Concepts

Lecture 23

Indexes and B+ Trees

October 28, 2013

Credits for slides: Chang, Ullman, Whitehead.

Outline

Last:

- Timestamps 18.8
- Indexes

Today:

- Indexes and B-trees 14.1-14.2

Next:

- Query execution 15.1-15.6
- Query optimization 16

2

Index Classification

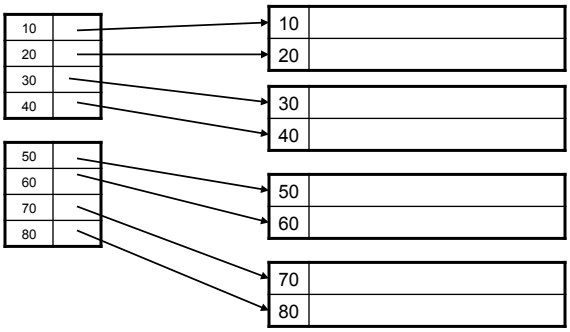
- Clustered/unclustered
  - Clustered = records close in index are close in data
  - Unclustered = records close in index might be far in data
- Primary/secondary:
  - Primary = on primary key
  - Secondary = on any other key
- Dense/sparse
  - Dense = each record has an entry in the index
  - Sparse = only some records have
- Organization: B+ tree or Hashtable

Clustered/Unclustered

- Clustered
  - Index determines the location of indexed records
  - Typically, a clustered index is one where values are data records (but not necessary)
- Unclustered
  - Index does not determine data location
  - In these indexes: value = pointer to data record

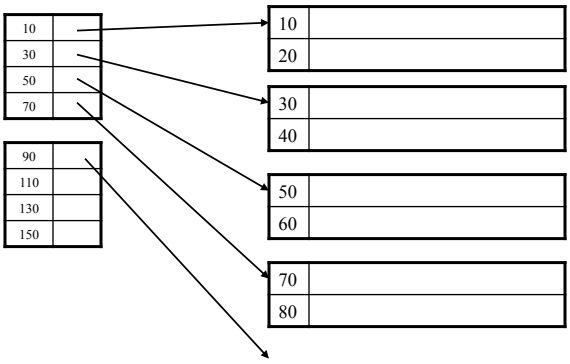
Clustered, Dense Index

- File is sorted on the index attribute
- Only one per table



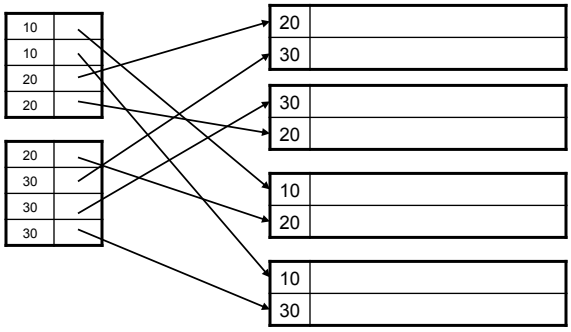
Clustered, Sparse Index

- *Sparse* index: one key per data block

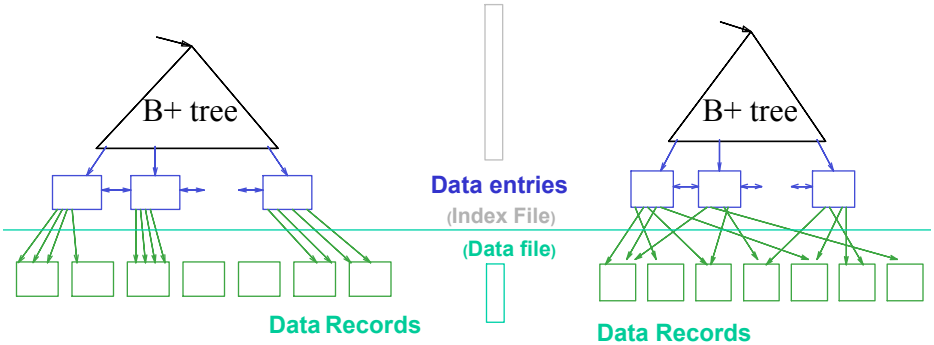


Unclustered Index

- Several per table
- Often for indexing other attributes than primary key
- Always dense (why?)



Clustered vs. Unclustered Index

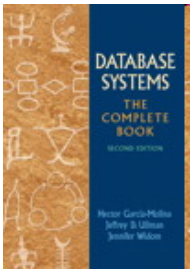


CLUSTERED

UNCLUSTERED

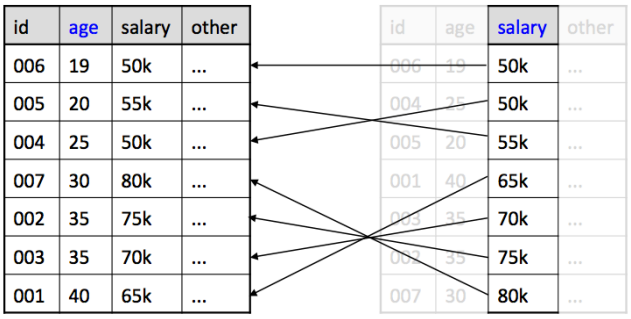
In a clustered B+ tree index,
data entries can be data records

Q: Our textbook as example: Indexes?



- How many indexes? Where?
- What are keys? What are records?
- Clustered?
- Dense?
- Primary?

Example



id	age	salary	other
006	19	50k	...
005	20	55k	...
004	25	50k	...
007	30	80k	...
002	35	75k	...
003	35	70k	...
001	40	65k	...

id	age	salary	other
006	19	50k	...
004	25	50k	...
005	20	55k	...
001	40	65k	...
002	35	70k	...
003	35	75k	...
007	30	80k	...

data file = index file
clustered, dense index

separate index file
unclustered, dense index

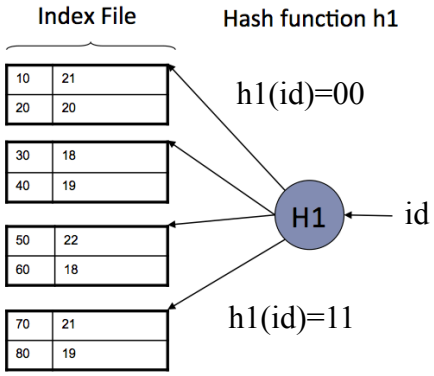
Hash-Based Index Example 1

Hash-based index on id

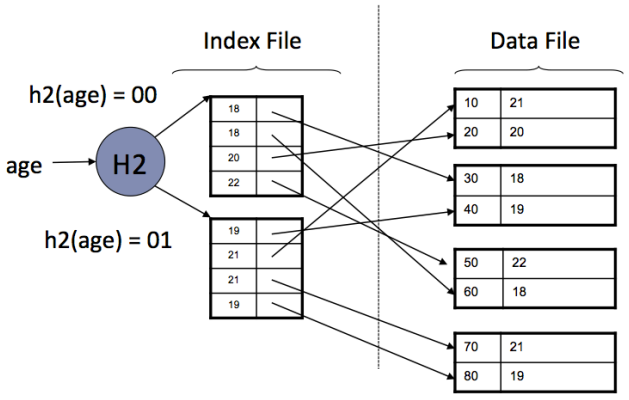
This is a primary index

In this case, data entries in the index are actual data records - there is no separate data file

This index is also clustered because it determines the order of indexed records



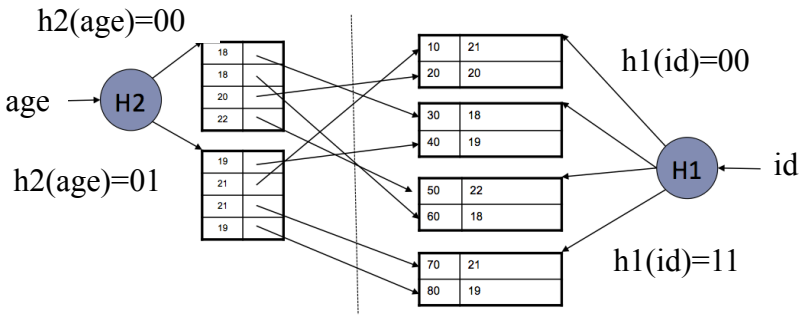
Hash-Based Index Example 2



Secondary index
Data entries in index are (key, recordID) pairs

Unclustered index

Hash-Based Indexes



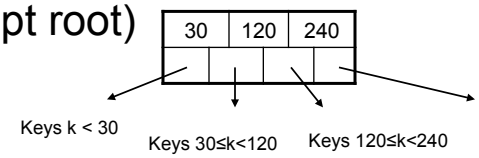
Good for point queries but not range queries

B+ Trees

- Search trees
- Idea in B Trees:
  - make 1 node = 1 block
  - keep tree balanced in height
- Idea in B+ Trees:
  - make leaves into a linked list to facilitate range queries

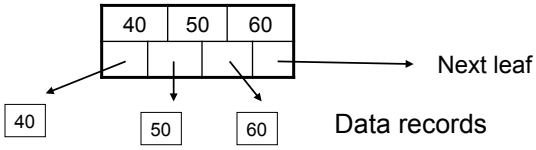
B+ Trees Basics

- Parameter d = the *degree*
- Each internal node has d≤m≤2d keys (except root)



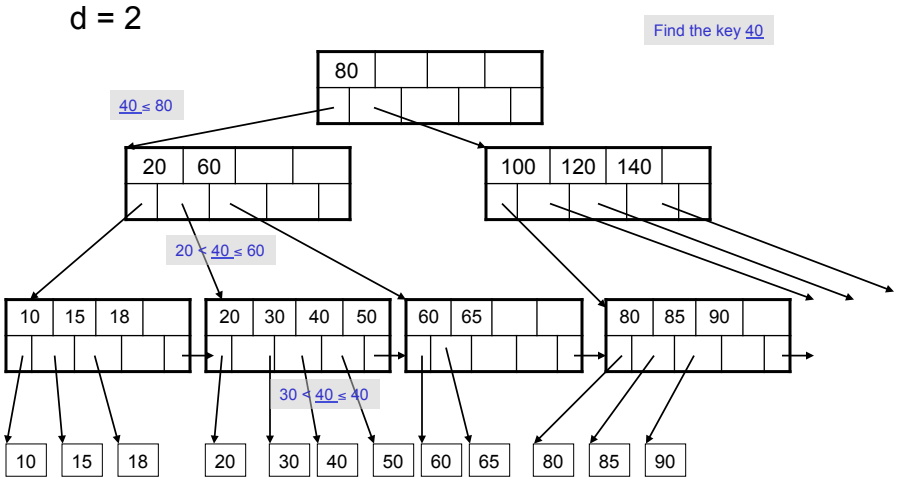
Each node also has **m+1 pointers**

- Each leaf has d≤m≤2d keys:



B+ Tree Example

d = 2



Using a B+ Tree

- Exact key values:
  - Start at the root
  - Proceed down, to the leaf
- Range queries:
  - As above
  - Then sequential traversal

Index on People(age)

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

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

Which queries can use this index?

Index on People(name, zipcode)

```
Select *
From People
Where name = 'Smith'
   and zipcode = 12345
```

```
Select *
From People
Where name = 'Smith'
```

```
Select *
From People
Where zipcode = 12345
```

B+ Tree Design

- How large d?
- Example:
  - Key size = 4 bytes
  - Pointer size = 8 bytes
  - Block size = 4096 byes

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

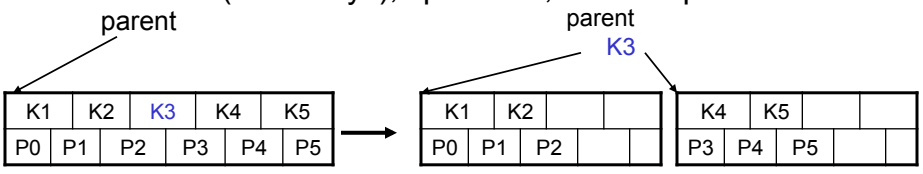
B+ Trees in Practice

- Average degree (fanout) = 133. Typical order (max number of children): 100. Typical fill-factor: 67%.
- 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)

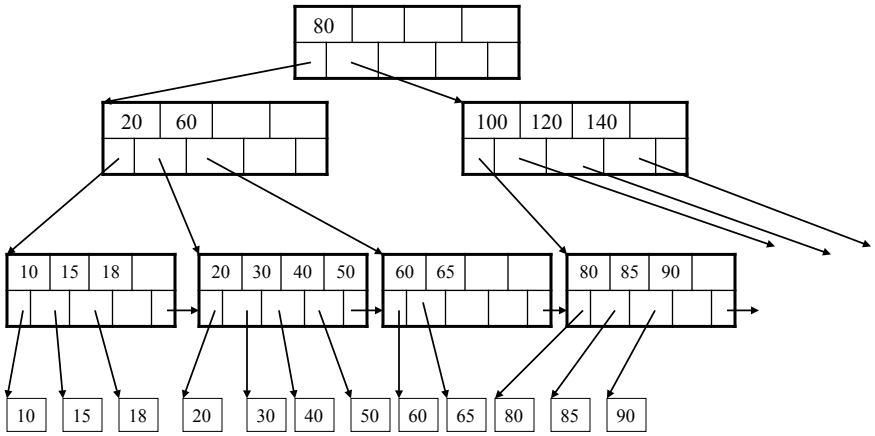
- Find leaf where K belongs, insert
- If no overflow (2d keys or less), halt
- If overflow (2d+1 keys), split node, insert in parent:



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

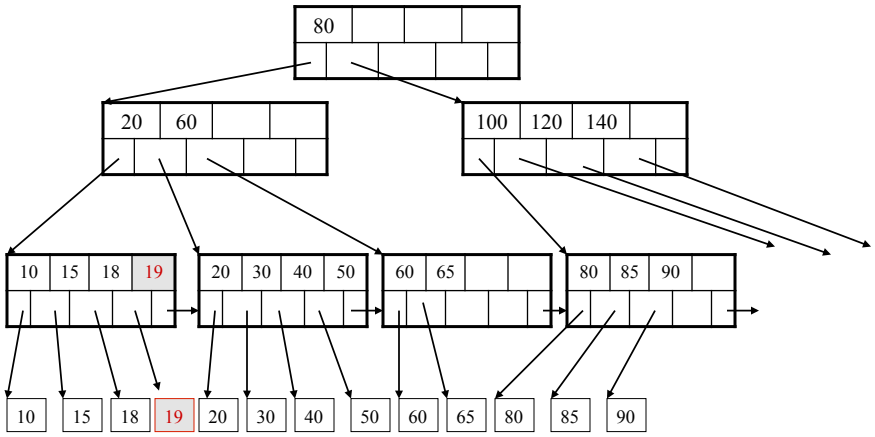
Insertion in a B+ Tree

Insert K=19



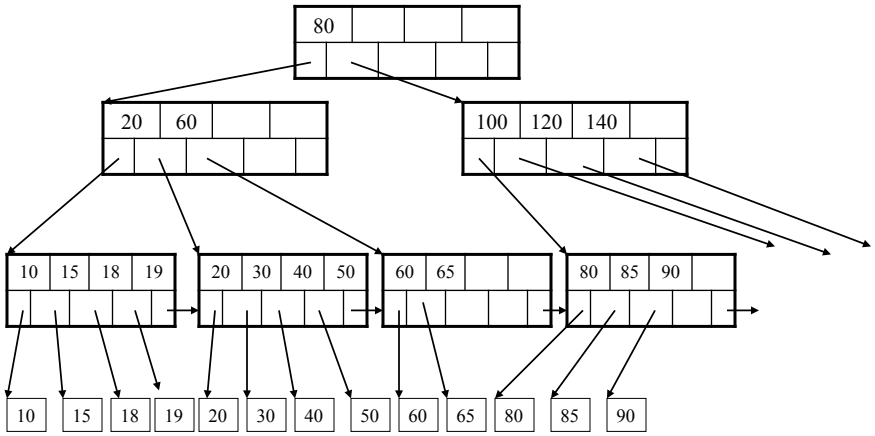
Insertion in a B+ Tree

After insertion



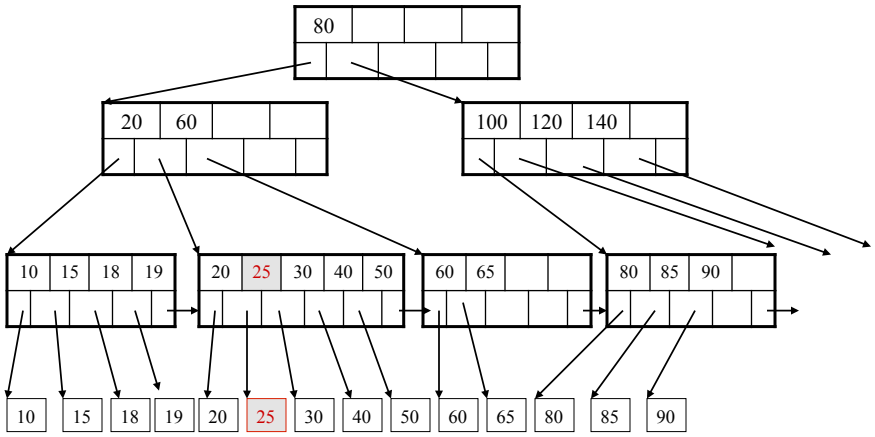
Insertion in a B+ Tree

Now insert 25



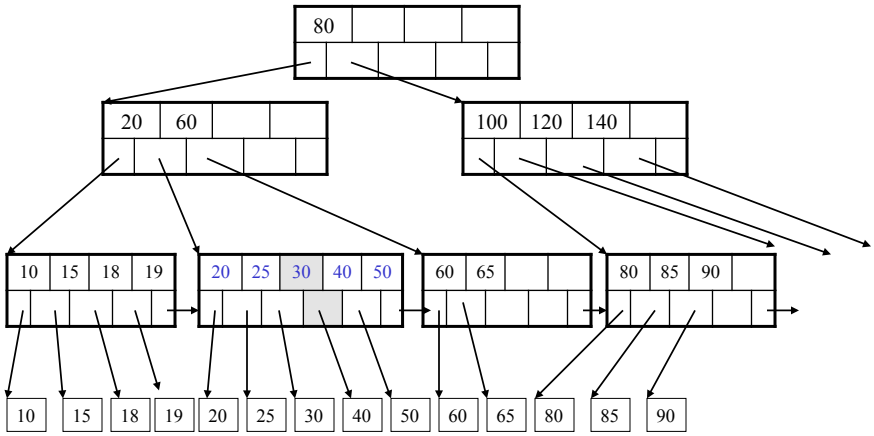
Insertion in a B+ Tree

After insertion



Insertion in a B+ Tree

But now have to split!



Insertion in a B+ Tree

After the split