File Organization

- file organization is about which tuples to place on each page, and how specific tuples can be inserted, retrieved, updated and deleted
- two simple file organizations

  "heap" files: any record can be placed on any page

  - fast, simple insertion
  - supports scan operations - retrieval of all tuples in a table
  - no efficient way to retrieve specific tuples

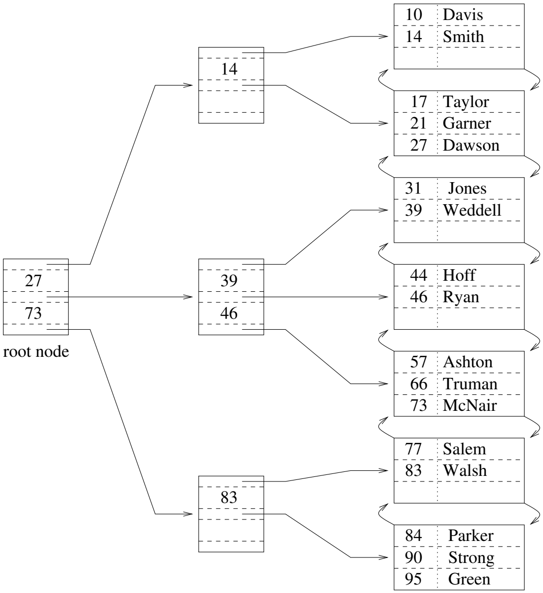
  sorted files: records are sorted according to the value of one (or more) attributes

  - can support table scan in sorted order
  - log time retrieval of specific tuples (binary search)
  - insertions may be expensive

Indexes

- an index is a data structure used for file organization
- indexes may be used to support
  - efficient retrieval of specific tuples, e.g., the record of an employee with a specified employee ID
  - efficient retrieval of ranges of tuples, e.g., the records of employees with start dates in 2010
  - ordered tuple scans, e.g., the records of all employees, ordered by surname
  - other operations, e.g., insertion, deletion, enforcement of integrity constraints
- there are many types of indexes, e.g., tree-structured, hash-based
- there may be more than one index on a single table

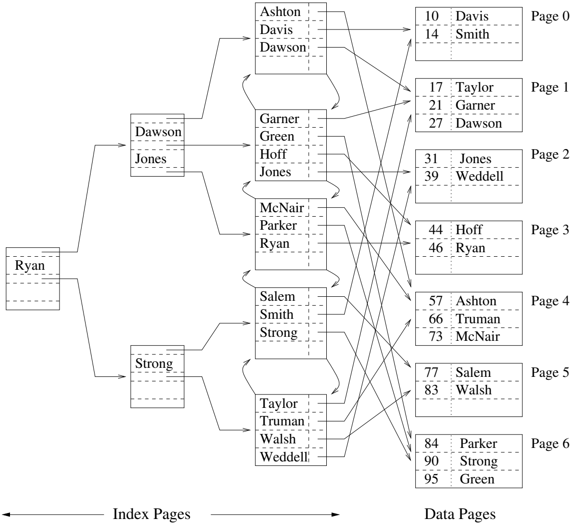
Example: A Clustered B+-tree



Clustering

- A relation whose tuples are grouped into blocks based on the value of attribute (field) *A* is said to be clustered on *A*. An index on attribute *A* is called a clustered index.
- Indexes that do not have this property are called unclustered indexes
- A relation that is sorted on *A* is clustered on *A*. However, a relation that is clustered on *A* need not be strictly sorted on *A*.

Example: An Unclustered B+-tree Index



Ryan

Dawson
Jones

Strong

Ashton
Davis
Dawson

Garner
Green
Hoff
Jones

McNair
Parker
Ryan

Salem
Smith
Strong

Taylor
Truman
Walsh
Weddell

10	Davis	Page 0
14	Smith	
17	Taylor	Page 1
21	Garner	
27	Dawson	
31	Jones	Page 2
39	Weddell	
44	Hoff	Page 3
46	Ryan	
57	Ashton	Page 4
66	Truman	
73	McNair	
77	Salem	Page 5
83	Walsh	
84	Parker	Page 6
90	Strong	
95	Green	

Index Pages

Data Pages

B-tree blocks

- Non-Leaf blocks
  - each block stores at most m values and $m + 1$ pointers
  - each block stores at least $\lfloor m/2 \rfloor$ values and $\lfloor m/2 \rfloor + 1$ pointers

$$\boxed{P_0 \mid V_1 \mid P_1 \mid V_2 \mid \cdots \mid V_m \mid P_m}$$

- Leaf blocks
  - may contain the tuples themselves (called Type 1 in the textbook)
    - illustrated on Slide 3
  - may contain key values plus tuple identifiers (called Type 2 and Type 3in the textbook)
    - illustrated on Slide 5
    - Type 2 and Type 3 are distinguished by the way they handle duplicate search keys

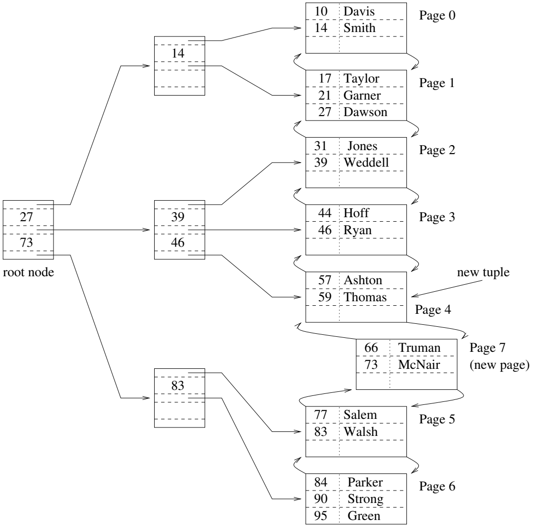
Terminology

- Type 1 Indexes
  - are clustered by definition (why?)
  - are sometimes called index-organized files
    - alternative to heap files and sorted files
- Type 2 and Type 3 Indexes
  - often unclustered, but may be clustered (how?)
- There can be at most one clustered index per relation (why?), and that index is sometimes called the primary index.
- A table can have multiple unclustered indexes, and they are sometimes called secondary indexes
- A dense index includes all search key values in its leaf nodes. A sparse index only includes one search key per data block.
  - a sparse index only makes sense if it is built on the relation's clustering attribute

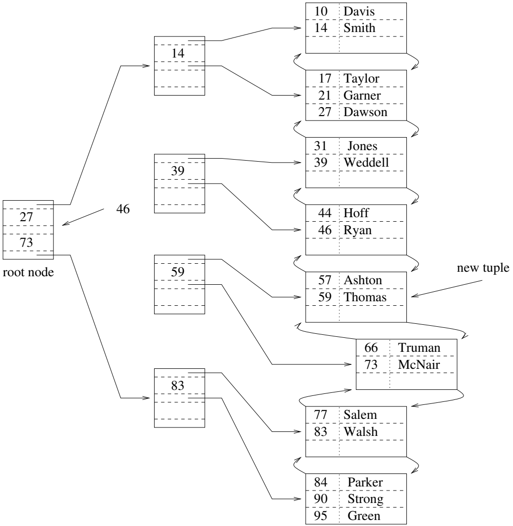
B-tree Insertions

1. Determine leaf block where new tuple belongs.
2. If there is room in the block, place the tuple in it.
3. If there is no room, find an empty block, and move half of the records into the new block. This is called **splitting**.
4. Add an entry for the new block in the parent index block.
5. If the index block is full, it may split. In this case, the middle pointer is promoted to the next higher index level.
6. Splitting may continue all the way to the root of the b-tree.

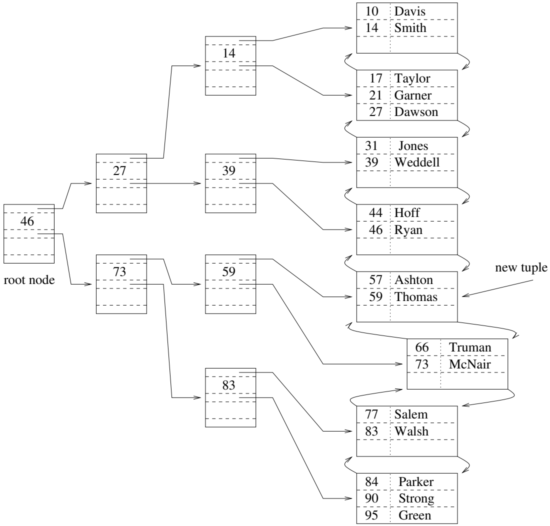
Insertion Example



Insertion Example (cont.)



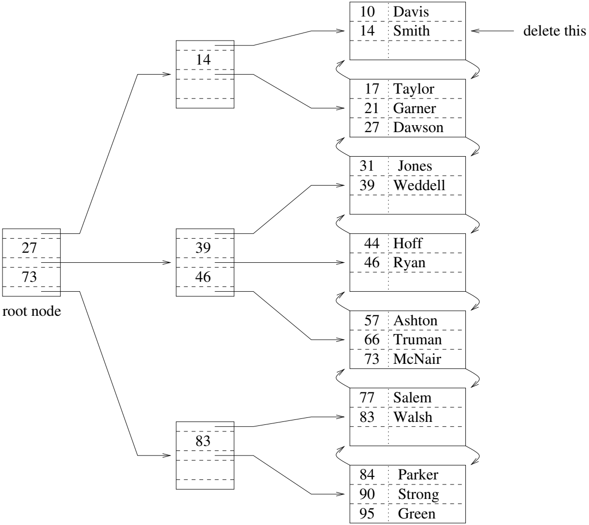
Insertion Example (cont.)



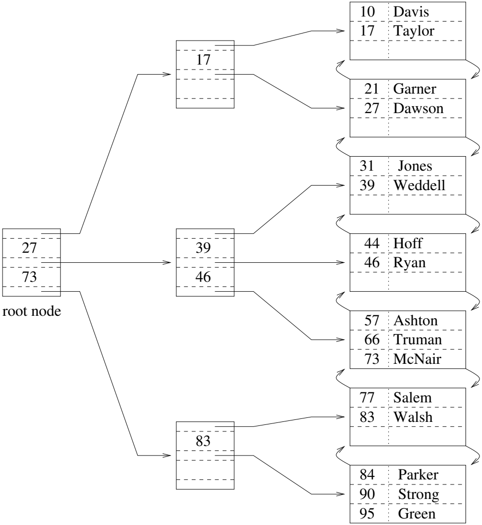
B-tree Deletions

1. Determine leaf block where tuple is located.
2. Remove the tuple from the leaf block.
3. If the block is less than half full, either:
    - distribute remaining tuples to the block's sibling, remove the block from the B-tree, and delete the block's pointer from the parent index node, or
    - steal some tuples from the block's siblings, and place them in the block
4. If a leaf block is removed, its pointer must be deleted from its parent's index node. Deletion of pointers may cascade all the way to the root.

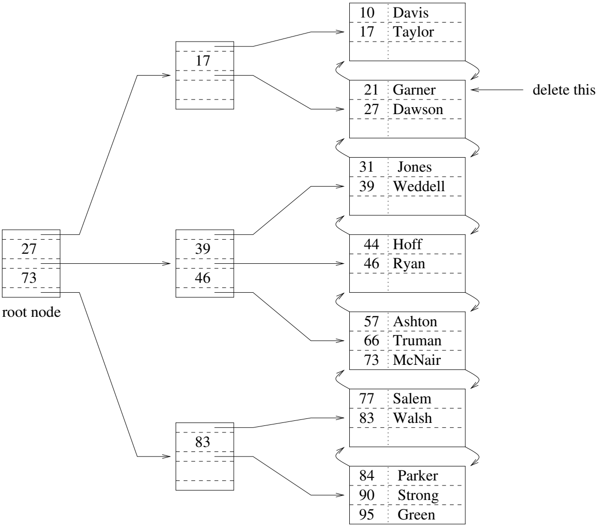
Deletion Example



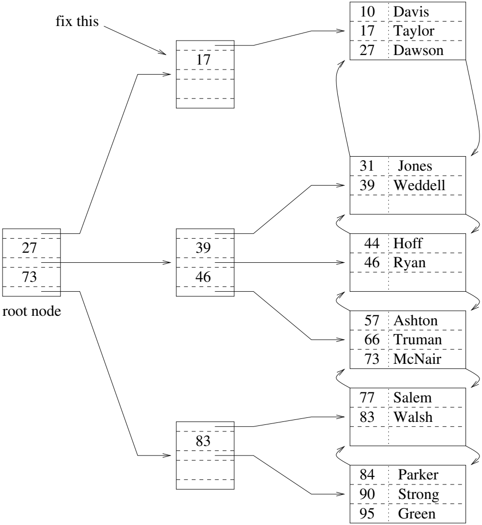
Deletion Example (cont'd)



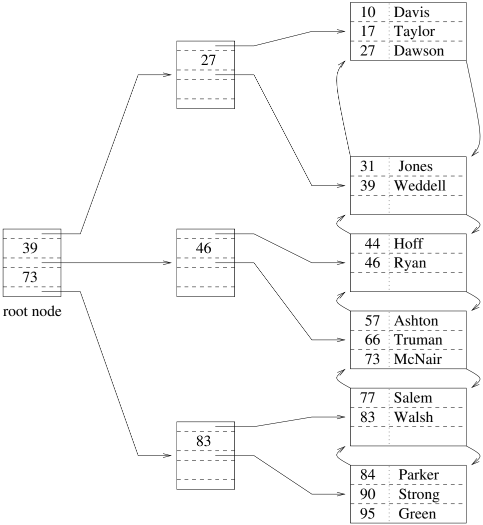
Deletion Example (cont'd)



Deletion Example (cont'd)



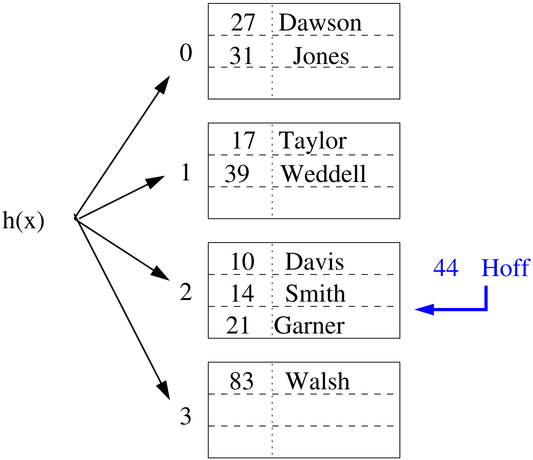
Deletion Example (cont'd)



Hash-based Indexes

- basic idea: use hashing to map attribute values to page numbers
- fast access to specific tuples, but no support for range retrievals or sorted scans
- retrieval performance of static hashing can deteriorate over time due to bucket overflows caused by tuple insertions
- extensible hashing tries to avoid this problem

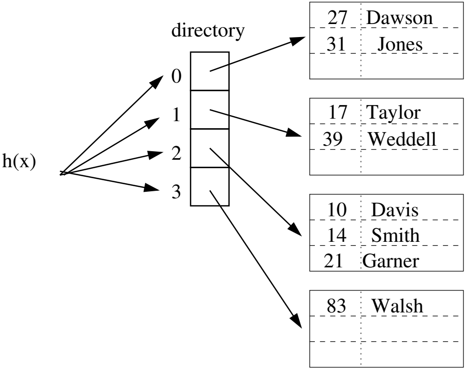
Static Hashing



Assume:

x	*h*(*x*)
10	2
14	2
17	1
21	2
27	0
31	0
39	1
44	2
83	3

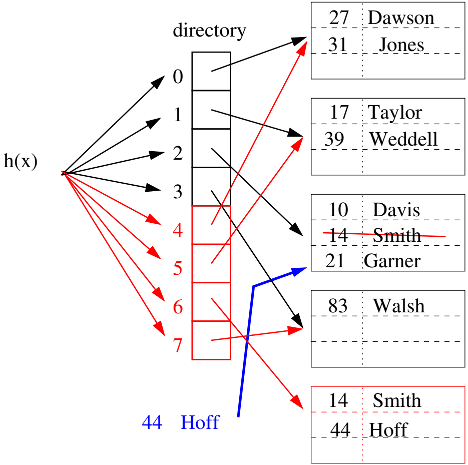
Extensible Hashing



Assume:

x	$h_4(x)$
10	2
14	2
17	1
21	2
27	0
31	0
39	1
83	3

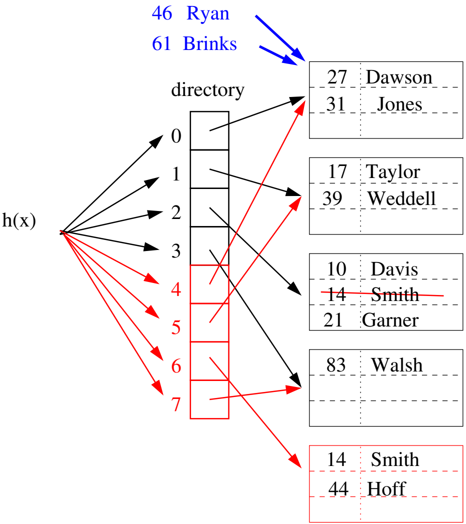
Extensible Hashing Directory Extension



Assume:

x	$h_4(x)$	$h_8(x)$
10	2	2
14	2	6
17	1	1
21	2	2
27	0	0
31	0	4
39	1	1
44	2	6
83	3	7

Extensible Hashing Block Allocation



Assume:

x	$h_8(x)$
10	2
14	6
17	1
21	2
27	0
31	4
39	1
44	6
46	0
61	4
83	7

Exploiting Access Methods

- DBMS may have multiple access methods for a single relation
- For each query, the DBMS query optimizer must select an access method for each relation involved in a query.
- The task of choosing which access methods are available falls to the database administrator (DBA), and is called physical design.

The Physical Schema

create index `LastnameIndex`
on `Employee(Lastname);`

drop index `LastnameIndex`

Effects of `LastnameIndex`:

- May speed up processing of queries involving `Lastname`
- May increase execution time for insertions/deletions/updates of tuples from `Employee`
- Increases the size of the database

Relevant Access Methods

```
Song(SongID,Title,ReleaseID,Duration,Format,Genre)
Release(ReleaseID,Title,ArtistID,ReleaseDate)

select  S.Genre, count(*), sum(S.Duration)
from  Song S, Release R
where  S.ReleaseID = R.ReleaseID
  and  S.Format = 'MP3'
  and  R.ReleaseDate > '1/1/2005'
group by  S.Genre
order by  S.Genre
```

Which access methods might be useful for this query?

Physical Design Advisors

```
% db2advis -d sample -s "select empno,lastname
from employee where workdept = 'xxxx'"
Found maximum set of [1] recommended indexes
total disk space needed for initial set [   0.005]
 [ 50.5219] timerons  (without indexes)
 [ 25.1521] timerons  (with current solution)
 [%50.22] improvement
-- ===========================
-- index[1],    0.005MB
   CREATE INDEX WIZ1517 ON "KMSALEM "."EMPLOYEE"
   ("WORKDEPT" ASC, "LASTNAME" ASC, "EMPNO" ASC) ;
-- ===========================
```

Design advisors can assist DBAs with physical design.