
DBMS
data
database
distributed
parallel
shared
system
architecture
processing
network
DBMSs
Distributed
DDBMS
memory
processors
access
systems
centralized
multiple
performance
sites
site
applications
computers
require
physically
processor
nothing
Parallel
single
one
may
fragments
typically
major
sometimes
local
provide
number
disk
computer
stored
logically
decentralized
large
technology
Shared
application
bottleneck
interconnection
Figure
nodes
definition
development
logical
distinction
improve
approach
scalability
control
Database
like
machines
users
within
across
reliability
eliminates
fact
distribution
communication
computing
split
order
scalable
architectures
disks
islands
resources
operational
referred
make
replicated
also
many
discuss
often
Management
makes
user
dis
per
consider
incompatible
structure
collection
provides
global
way
associated
used
least
partitioned
consists
administered
organizational
transparent
management
mode
capable
must
System
databases
directly
com
Date
based
controlled
defined
remote
solely

Database System Concept (CSE 3103)

Lecture 06-Day 01

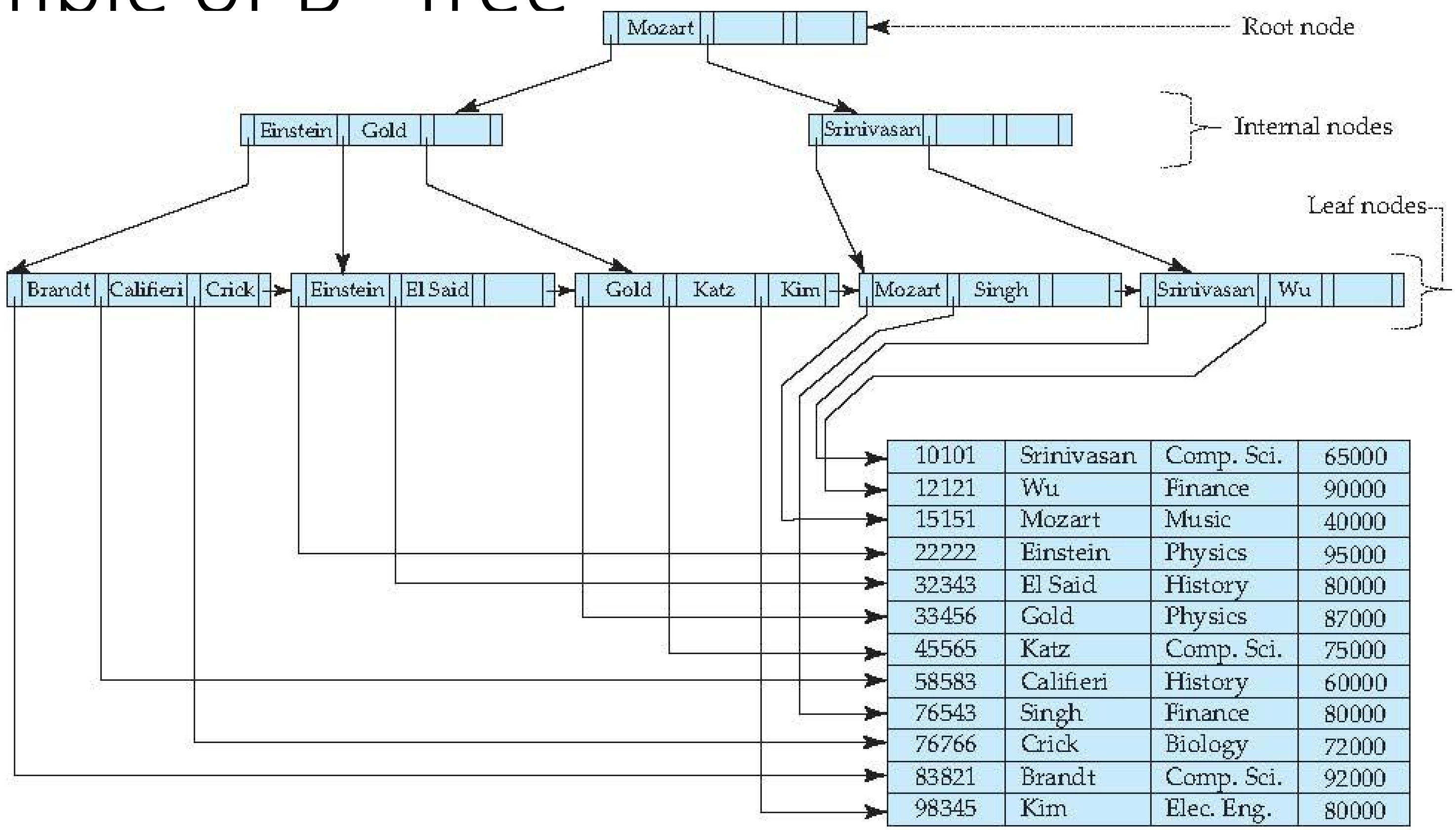
Nazmus Sakib, Assistant Professor, Dept. of CSE, AUST

B^+-Tree Index Files

B^+-tree indices are an alternative to indexed-sequential files.

- Disadvantage of indexed-sequential files
  - performance degrades as file grows, since many overflow blocks get created.
  - Periodic reorganization of entire file is required.
- Advantage of B^+-tree index files:
  - automatically reorganizes itself with small, local, changes, in the face of insertions and deletions.
  - Reorganization of entire file is not required to maintain performance.
- (Minor) disadvantage of B^+-trees:
  - extra insertion and deletion overhead, space overhead.
- Advantages of B^+-trees outweigh disadvantages
  - B^+-trees are used extensively

Example of B+-Tree



10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

B^+-Tree Index Files (Cont.)

A B^+-tree is a rooted tree satisfying the following properties:

- All paths from root to leaf are of the same length
- Each node that is not a root or a leaf has between $\lceil n/2 \rceil$ and n children.
- A leaf node has between $\lceil (n-1)/2 \rceil$ and $n-1$ values
- Special cases:
  - If the root is not a leaf, it has at least 2 children.
  - If the root is a leaf (that is, there are no other nodes in the tree), it can have between 0 and $(n-1)$ values.

B^+-Tree Node Structure

- Typical node

P_1	K_1	P_2	$\dots$	P_{n-1}	K_{n-1}	P_n

  - K_i are the search-key values
  - P_i are pointers to children (for non-leaf nodes) or pointers to records or buckets of records (for leaf nodes).
- The search-keys in a node are ordered

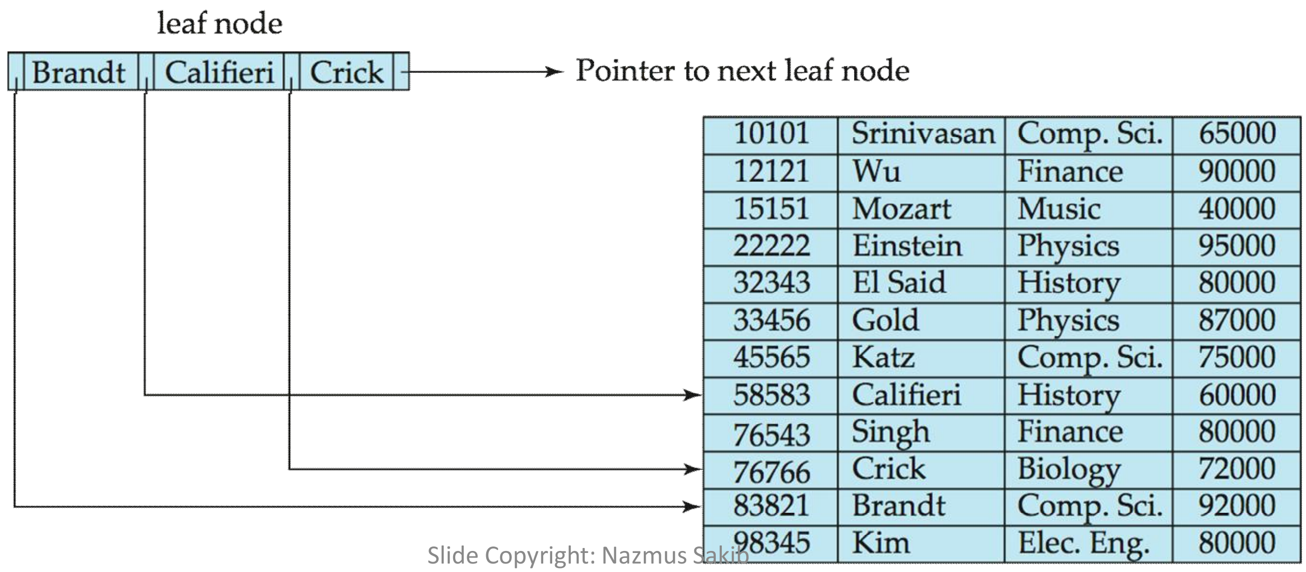
$$K_1 < K_2 < K_3 < \dots < K_{n-1}$$

(Initially assume no duplicate keys, address duplicates later)

Leaf Nodes in B^+-Trees

Properties of a leaf node:

- For $i = 1, 2, \ldots, n-1$, pointer P_i points to a file record with search-key value K_i,
- If L_i, L_j are leaf nodes and $i < j$, L_i's search-key values are less than or equal to L_j's search-key values
- P_n points to next leaf node in search-key order



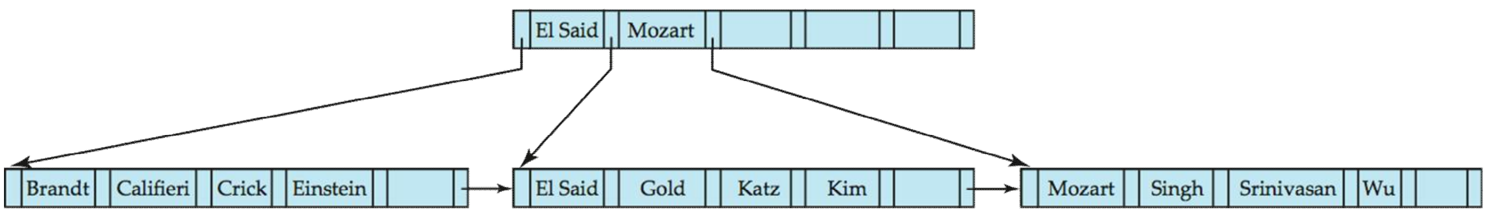
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	80000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	60000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Non-Leaf Nodes in B$^+$-Trees

- Non leaf nodes form a multi-level sparse index on the leaf nodes. For a non-leaf node with *m* pointers:
  - All the search-keys in the subtree to which P_1 points are less than K_1
  - For $2 \le i \le n - 1$, all the search-keys in the subtree to which P_i points have values greater than or equal to K_{i-1} and less than K_i
  - All the search-keys in the subtree to which P_n points have values greater than or equal to K_{n-1}

P_1	K_1	P_2	…	P_{n-1}	K_{n-1}	P_n

Example of B$^+$-tree



B$^+$-tree for *instructor* file ($n = 6$)

- Leaf nodes must have between 3 and 5 values ($\lceil (n-1)/2 \rceil$ and $n - 1$, with $n = 6$).
- Non-leaf nodes other than root must have between 3 and 6 children ($\lceil (n/2 \rceil$ and n with $n = 6$).
- Root must have at least 2 children.

Observations about B$^+$-trees

- Since the inter-node connections are done by pointers, "logically" close blocks need not be "physically" close.
- The non-leaf levels of the B$^+$-tree form a hierarchy of sparse indices.
- The B$^+$-tree contains a relatively small number of levels
    - Level below root has at least $2*\lceil n/2 \rceil$ values
    - Next level has at least $2*\lceil n/2 \rceil * \lceil n/2 \rceil$ values
    - .. etc.
  - If there are K search-key values in the file, the tree height is no more than $\lceil \log_{\lceil n/2 \rceil}(K) \rceil$
  - thus searches can be conducted efficiently.
- Insertions and deletions to the main file can be handled efficiently, as the index can be restructured in logarithmic time (as we shall see).