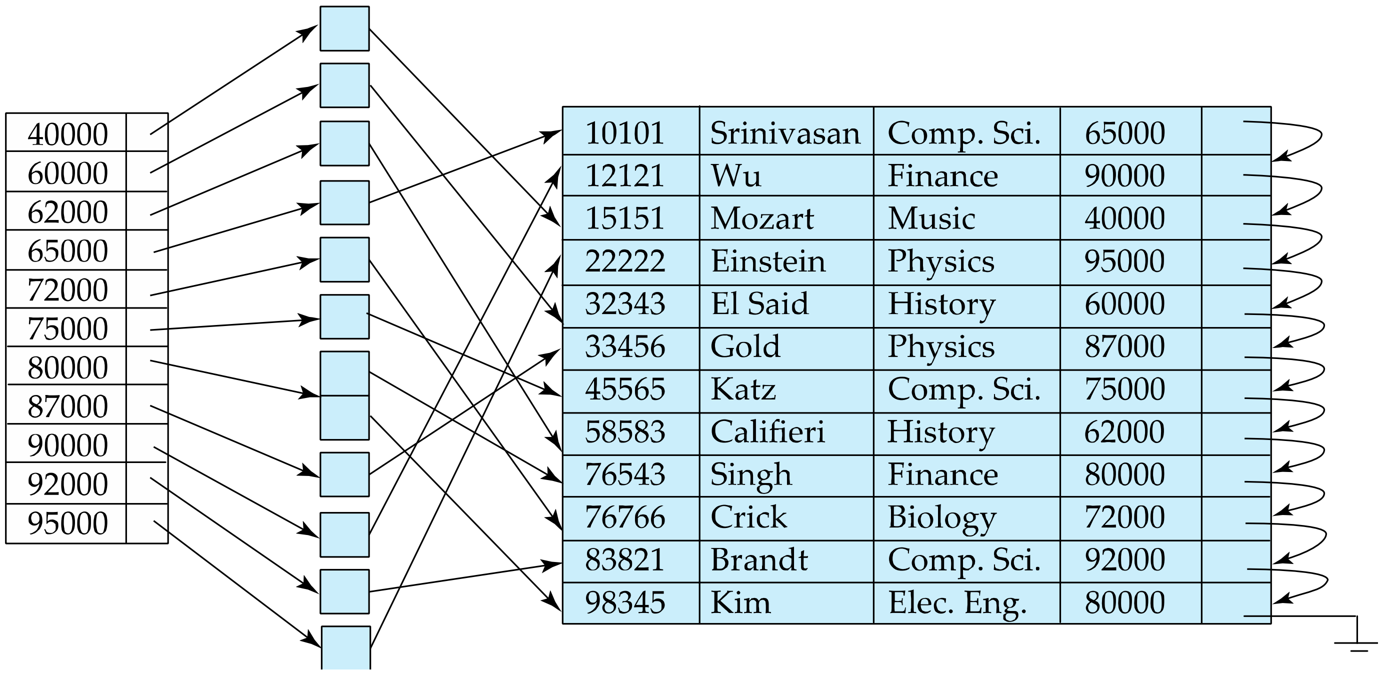
data
distributed
database
DBMS
shared
parallel
architecture
system
processing
network
DBMSs
systems
Distributed
access
centralized
performance
processors
multiple
memory
DDBMS
site
sites
single
applications
computer
disk
stored
nothing
number
require
physically
Parallel
processor
fragments
may
provide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

Database System Concept (CSE 3103)

Lecture 05-Day 02

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

Secondary Indices Example



40000
60000
62000
65000
72000
75000
80000
87000
90000
92000
95000

10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Secondary index on *salary* field of *instructor*

- Index record points to a bucket that contains pointers to all the actual records with that particular search-key value.
- Secondary indices have to be dense

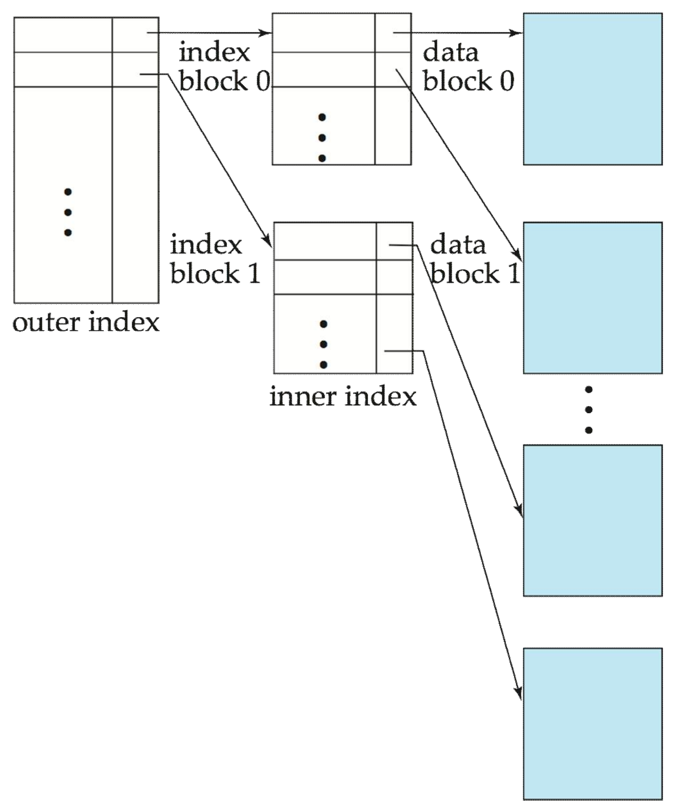
Primary and Secondary Indices

- Indices offer substantial benefits when searching for records.
- BUT: Updating indices imposes overhead on database modification --when a file is modified, every index on the file must be updated,
- Sequential scan using primary index is efficient, but a sequential scan using a secondary index is expensive
  - Each record access may fetch a new block from disk
  - Block fetch requires about 5 to 10 milliseconds, versus about 100 nanoseconds for memory access

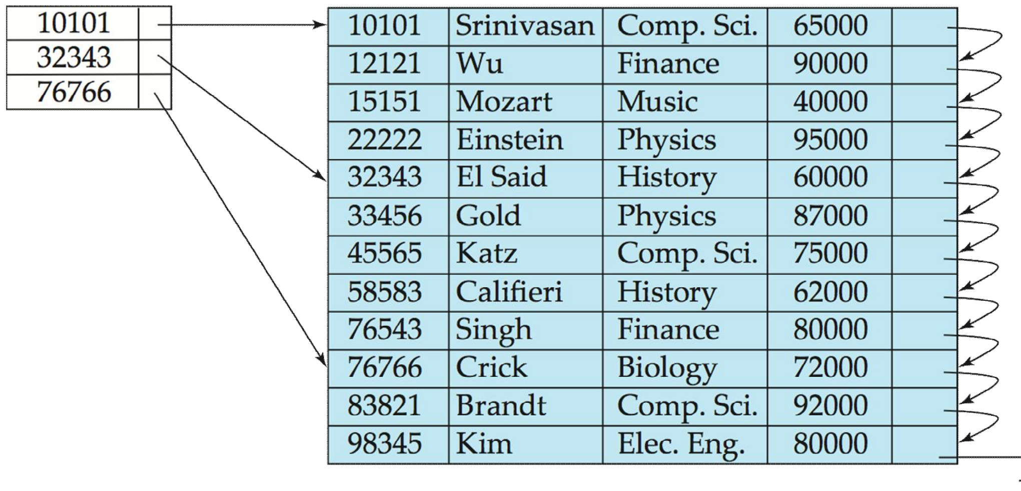
Multilevel Index

- If primary index does not fit in memory, access becomes expensive.
- Solution: treat primary index kept on disk as a sequential file and construct a sparse index on it.
    - outer index – a sparse index of primary index
    - inner index – the primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.

Multilevel Index (Cont.)



Index Update: Deletion



10101	
32343	
76766	

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

- If deleted record was the only record in the file with its particular search-key value, the search-key is deleted from the index also.

- **Single-level index entry deletion:**
  - **Dense indices** – deletion of search-key is similar to file record deletion.
  - **Sparse indices** –
    - if an entry for the search key exists in the index, it is deleted by replacing the entry in the index with the next search-key value in the file (in search-key order).
    - If the next search-key value already has an index entry, the entry is deleted instead of being replaced.

Index Update: Insertion

- **Single-level index insertion:**
  - Perform a lookup using the search-key value appearing in the record to be inserted.
  - **Dense indices** – if the search-key value does not appear in the index, insert it.
  - **Sparse indices** – if index stores an entry for each block of the file, no change needs to be made to the index unless a new block is created.
    - If a new block is created, the first search-key value appearing in the new block is inserted into the index.
- **Multilevel insertion and deletion:** algorithms are simple extensions of the single-level algorithms

Secondary Indices

- Frequently, one wants to find all the records whose values in a certain field (which is not the search-key of the primary index) satisfy some condition.
    - Example 1: In the *instructor* relation stored sequentially by ID, we may want to find all instructors in a particular department
    - Example 2: as above, but where we want to find all instructors with a specified salary or with salary in a specified range of values
- We can have a secondary index with an index record for each search-key value