Database Systems
(CS 355 / CE 373)

Dr. Umer Tariq

Assistant Professor,

Dhanani School of Science & Engineering,

Habib University

Acknowledgements

- Many slides have been borrowed from the official lecture slides accompanying the textbook:

    Database System Concepts, (2019), Seventh Edition,

    Avi Silberschatz, Henry F. Korth, S. Sudarshan

    McGraw-Hill, ISBN 9780078022159

    The original lecture slides are available at:

    https://www.db-book.com/

- Some of the slides have been borrowed from the lectures by Dr. Immanuel Trummer (Cornell University). Available at: (www.itrummer.org)
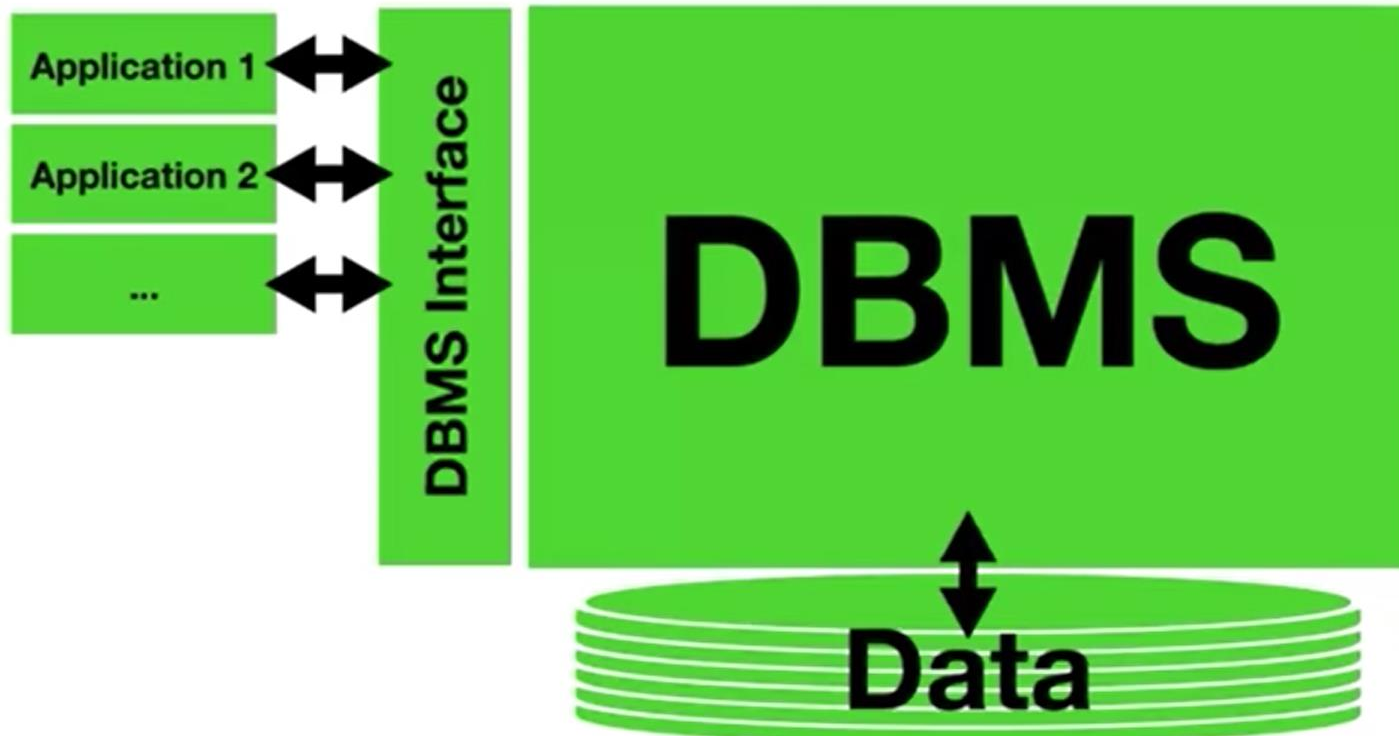
Outline: Week 14

- Introduction to the Concept of Database Indexing

- Various Types of Database Indices

- Tradeoffs between Various Types of Database Indices
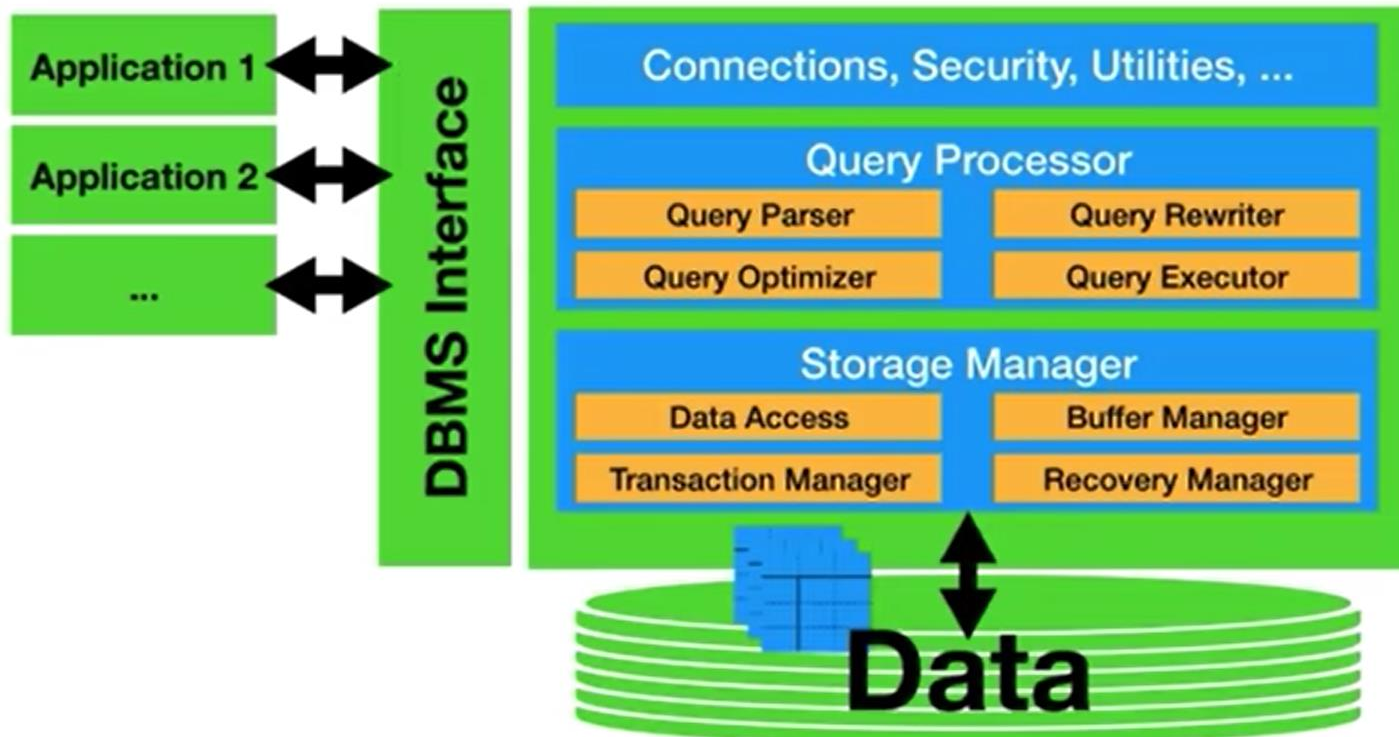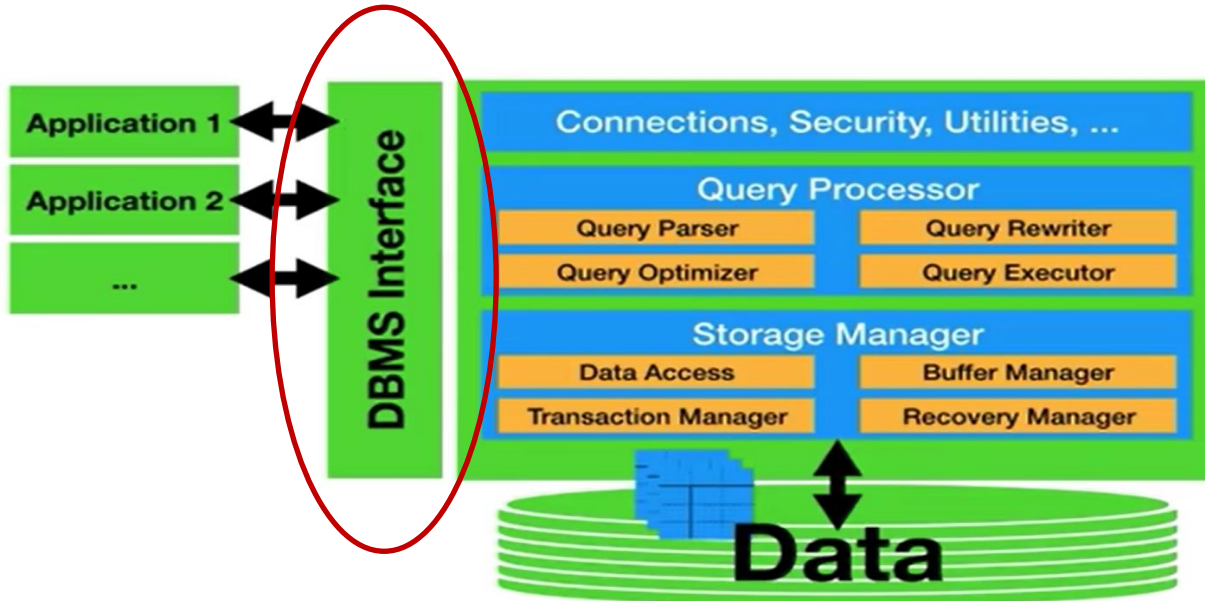
File-Based Approach
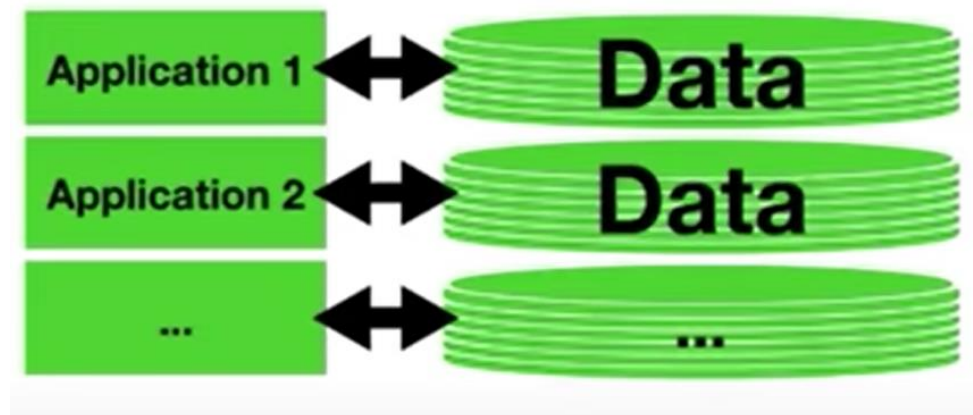
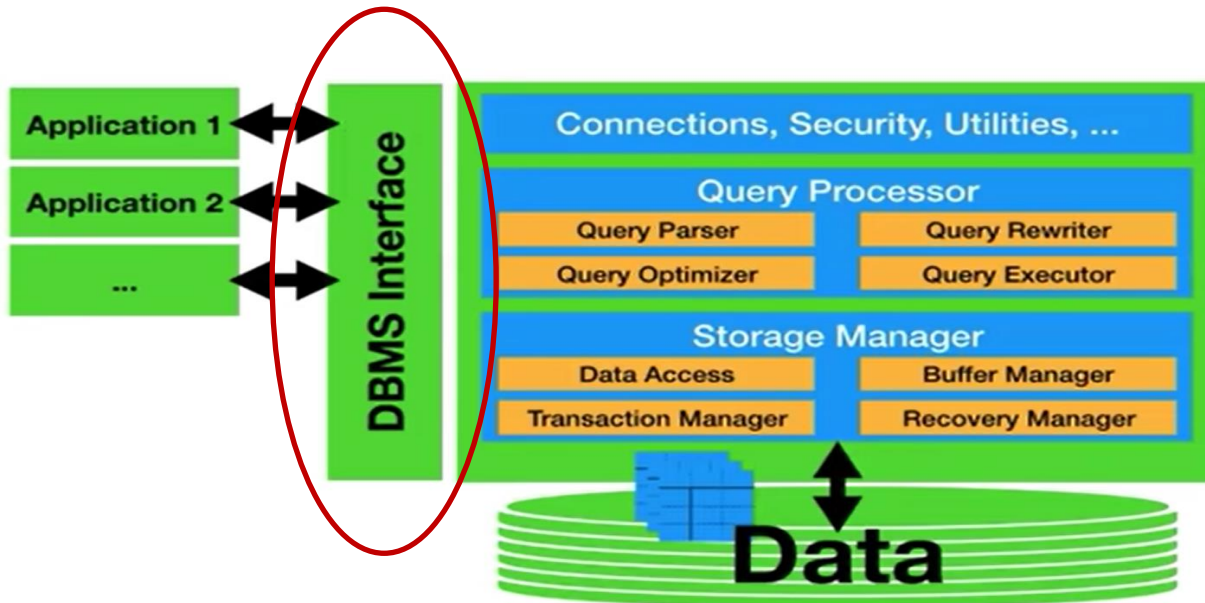# Database Management System (DBMS)

Database Management System (DBMS)

What should be the DBMS Interface?

What should be the DBMS Interface?

- Data Model
  - A collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

The Relational Model (TABLE-based Data Model)

- The relational model uses <u>a collection of tables</u> to represent both data and the relationships among those data.

- Its conceptual simplicity has led to its widespread adoption; a vast majority of database products are based on the relational model.

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

(a) The *instructor* table

dept_name	building	budget
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

9

File Storage: Background

Instructor.db

| Disk location |
| Disk location |
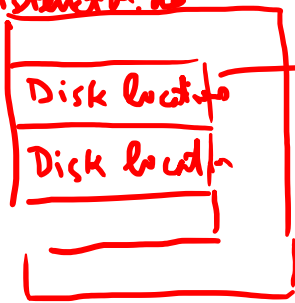
→ Block No. 10 + Offset: 16

og Block   16

Offset

Processor

↕

Main Memory
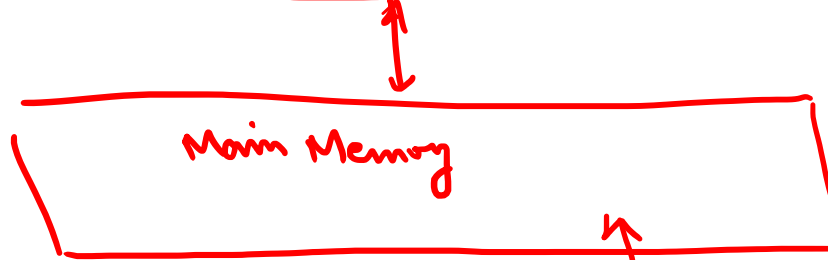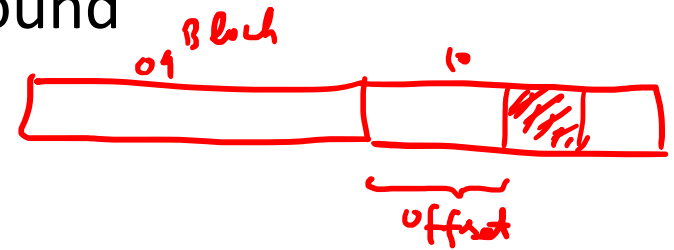
"Disk Block"

instructor.db

File with instructor table →
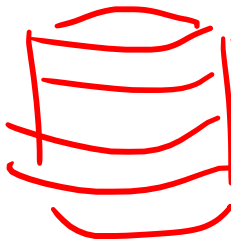
Hard Disk
(Non-Volatile Storage)

Block :   1 - 10,000

Database Indexing: Motivation

- Consider the following query:
  - Find all instructors in the Physics department

Instructor Table (Database File)

	ID	Name	Dept_Name	Salary	
1.	10101	Srinivasan	Comp. Sci.	65000	
2.	12121	Wu	Finance	90000	
3.	15151	Mozart	Music	40000	
4.	22222	Einstein	Physics	95000	
5.	32343	El Said	History	60000	
6.	33456	Gold	Physics	87000	
7.	45565	Katz	Comp. Sci.	75000	
8.	58583	Califieri	History	62000	
9.	76543	Singh	Finance	80000	
10	76766	Crick	Biology	72000	
11	83821	Brandt	Comp. Sci.	92000	
12	98345	Kim	Elec. Eng.	80000	

Sequential file for *instructor* records

Database Indexing: Motivation

- Consider the following query:
  - " Find all instructors in the Physics department "

Index file

Bio	10
CS	1,7,11
EE	12
	-
PHYSICS	4,6
	:

Instructor Table (Database File)

	ID	Name	Dept-Name	Salary
1-	10101	Srinivasan	Comp. Sci.	65000
2.	12121	Wu	Finance	90000
3.	15151	Mozart	Music	40000
4.	22222	Einstein	Physics -	95000
5.	32343	El Said	History	60000
6.	33456	Gold	Physics	87000
7-	45565	Katz	Comp. Sci.	75000
8.	58583	Califieri	History	62000
9.	76543	Singh	Finance	80000
10	76766	Crick	Biology	72000
11	83821	Brandt	Comp. Sci.	92000
12	98345	Kim	Elec. Eng.	80000

Sequential file for *instructor* records
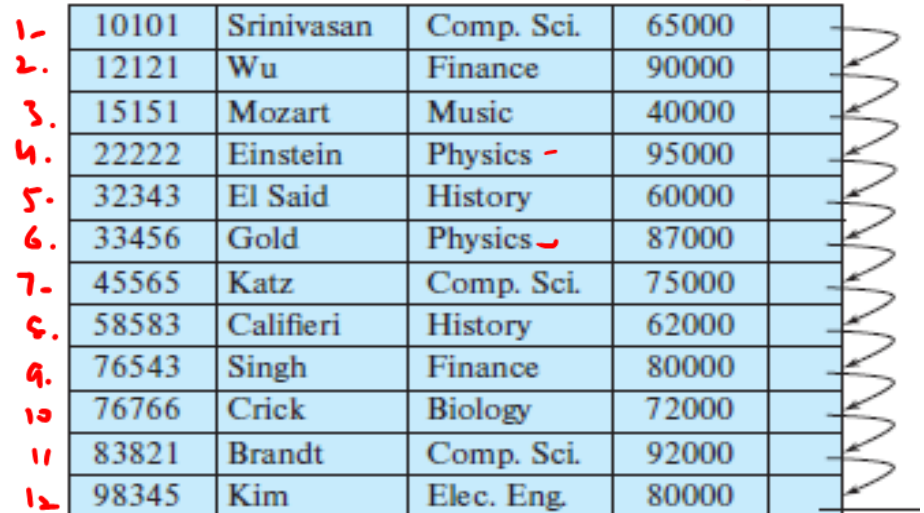
Database Indexing: Motivation

- Consider the following query:
  - Find all instructors in the Physics department

- It is inefficient for the system to read every tuple in the *instructor* relation to check if *dept_name* value is Physics

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	

Sequential file for *instructor* records

Database Indexing: Motivation

- Consider the following query:
  - Find all instructors in the Physics department

- It is inefficient for the system to read every tuple in the *instructor* relation to check if *dept_name* value is Physics
  - Let's consider an alternative, inspired by the idea of index typically at the end of a book.

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	

Sequential file for *instructor* records

Database Indexing: More about Analogy with Textbook Index

- Note the following about a textbook's index:
  - The words in the index are in sorted order, making it easy to find the word we want.
  - The index is much smaller than the textbook, further reducing the search effort needed.

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	

Sequential file for *instructor* records

Database Indexing

- An index for a file in a database system works in much the same way as the index in a textbook.
  - For example, to retrieve a student record given an ID, the database system would look up an index to find on which disk block the corresponding record resides.
  - The disk block is then fetched, to get the appropriate student record.

- Search Key
  - The attribute (or set of attributes) whose values are used to look up records in a table/file

- Index File
  - An index file consists of records (index entries) of the form:

    | search-key value | pointer to the record on disk |

    The pointer consists of an identifier of a disk block and an offset within the disk block to identify the record.
  - Index files are typically much smaller than the original database files.

INDEX	TABLE		
E00127	Tyler	Bennett	E10297
E01234	John	Rappl	E21437
E03033	George	Woltman	E00127
E04242	Adam	Smith	E63535
E10001	David	McClellan	E04242
E10297	Rich	Holcomb	E01234
E16398	Nathan	Adams	E41298
E21437	Richard	Potter	E43128
E27002	David	Motsinger	E27002
E41298	Tim	Sampair	E03033
E43128	Kim	Arlich	E10001
E63535	Timothy	Grove	E16398

16

Multiple Indices on a Database File

- A database file may have several indices on different search keys.

DB Index # 2

ID

ID	
10101	1
12121	2
15151	3

DB Index # 1

Dept-name

Dept-name	
Bio	10
CS.	1,7,11
EE	12
Finance	2,4
⋮	

	ID	Name	Dept-Name	Salary	
1.	10101	Srinivasan	Comp. Sci.	65000	
2.	12121	Wu	Finance	90000	
3.	15151	Mozart	Music	40000	
4.	22222	Einstein	Physics	95000	
5	32343	El Said	History	60000	
6	33456	Gold	Physics	87000	
7	45565	Katz	Comp. Sci.	75000	
8	58583	Califieri	History	62000	
9	76543	Singh	Finance	80000	
10	76766	Crick	Biology	72000	
11	83821	Brandt	Comp. Sci.	92000	
12	98345	Kim	Elec. Eng.	80000	

Sequential file for *instructor* records

Not All Indices are Created Equal!!

- Consider implementing an index on a very large student relation by keeping a sorted list of student IDs
  - It would not work very well because:
  - i) the index would itself be very big
  - ii) even though keeping the index sorted reduces the search time, finding the student can still be rather time-consuming
  - iii) updating a sorted list as students are added or removed from the database can be quite expensive

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

Figure 4.1 The *student* relation.

Index Evaluation Metrics

- We shall consider several techniques of indexing.
  - No one technique is best.
  - Each technique is best suited to particular database applications.
  - These techniques can be evaluated based on the following factors:

1) - **Access types**: The types of access that are supported efficiently. Access types can include finding records with a specified attribute value and finding records whose attribute values fall in a specified range.

2) - **Access time**: The time it takes to find a particular data item, or set of items, using the technique in question.

3) - **Insertion time**: The time it takes to insert a new data item. This value includes the time it takes to find the correct place to insert the new data item, as well as the time it takes to update the index structure.

4) - **Deletion time**: The time it takes to delete a data item. This value includes the time it takes to find the item to be deleted, as well as the time it takes to update the index structure.

5) - **Space overhead**: The additional space occupied by an index structure. Provided that the amount of additional space is moderate, it is usually worthwhile to sacrifice the space to achieve improved performance.

Handwritten annotations (top):

$y = h(x)$

$1 = h(\text{PHYSICS})$

$2 = h(\text{BIO})$

$3 = h(\text{MUSIC})$ $h(\text{EE}) = 3$

Types of Indices

There are two basic kinds of indices:

- **Ordered indices**
  - Based on a sorted ordering of the key values.

- **Hash indices**
  - Based on a uniform distribution of key values across a range of buckets. The bucket to which a value is assigned is determined by a function, called a *hash function*.

Handwritten labels: HASH INDEX, ORDERED Index, Dept-Name

Handwritten: $h(\text{PHY})$ $= 1$, $h($

Hash buckets: (PHY, 4) (PHY, 6); (BIO, 10); (EE, 12) 3 (MUSIC)

Ordered index entries: BIO, CS, EE 12, PHYSICS 4,6

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	

Sequential file for *instructor* records

20

Ordered Index

- An ordered index stores the values of the search keys in sorted order and associates with each search-key value the records that contain that value.

- The records in the indexed file may themselves be stored in some sorted order (not necessarily the order of the search key in the index structure).

INDEX	TABLE		
E00127	Tyler	Bennett	E10297
E01234	John	Rappl	E21437
E03033	George	Woltman	E00127
E04242	Adam	Smith	E63535
E10001	David	McClellan	E04242
E10297	Rich	Holcomb	E01234
E16398	Nathan	Adams	E41298
E21437	Richard	Potter	E43128
E27002	David	Motsinger	E27002
E41298	Tim	Sampair	E03033
E43128	Kim	Arlich	E10001
E63535	Timothy	Grove	E16398
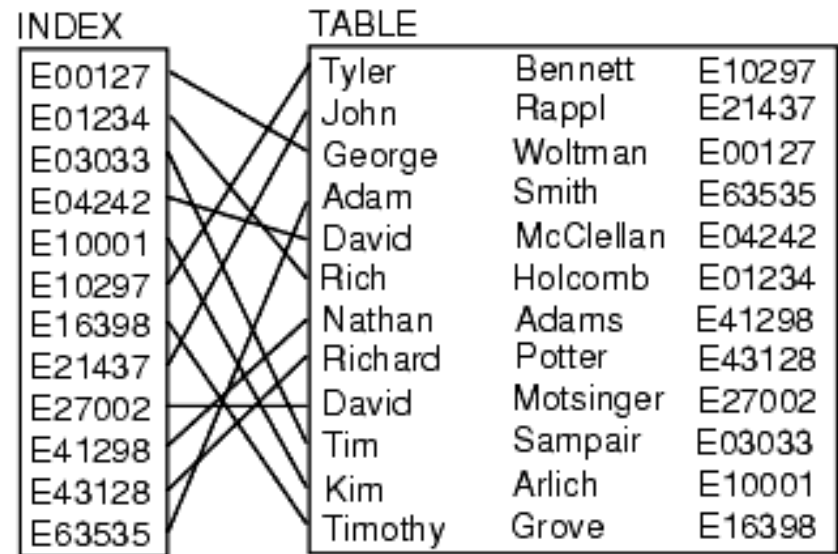
Types of Ordered Indices: Clustering vs Non-Clustering

- **Clustering Index (Clustered Index)**
  - If the database file containing the records/tuples is sequentially ordered, a clustering index is an index whose search key also defines the sequential order of the file.
  - Also known as Primary Index.

- **Non-Clustering Index (Non-Clustered Index)**
  - A non-clustering index is an index whose search key does not define the sequential order of the database file.
  - Also known as Secondary Index.

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

Sequential file for *instructor* records

Types of Ordered Indices: Dense vs Sparse

- **Dense Index**
  - In a dense index, an index entry appears in the index for every search-key value in the database file.

- **Sparse Index**
  - In a sparse index, an index entry appears in the index for only some of the search-key value in the database file.
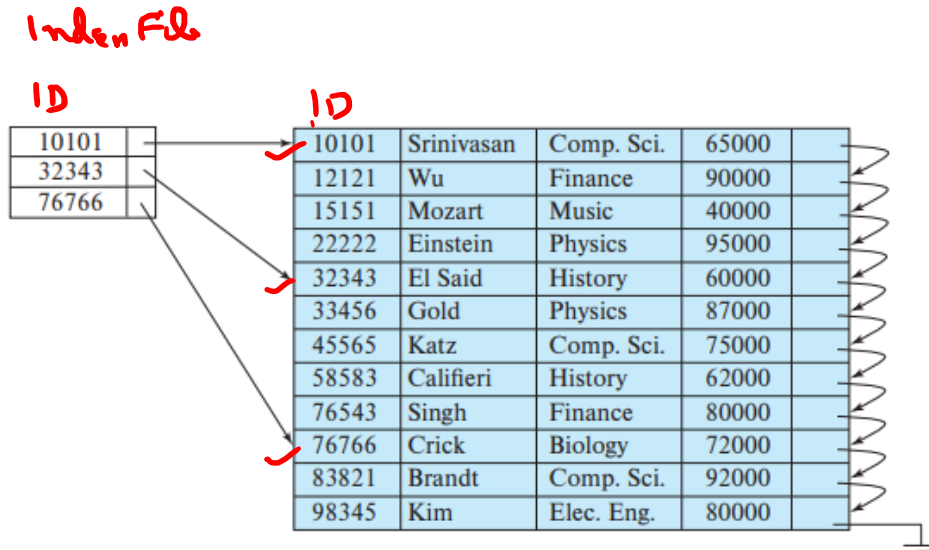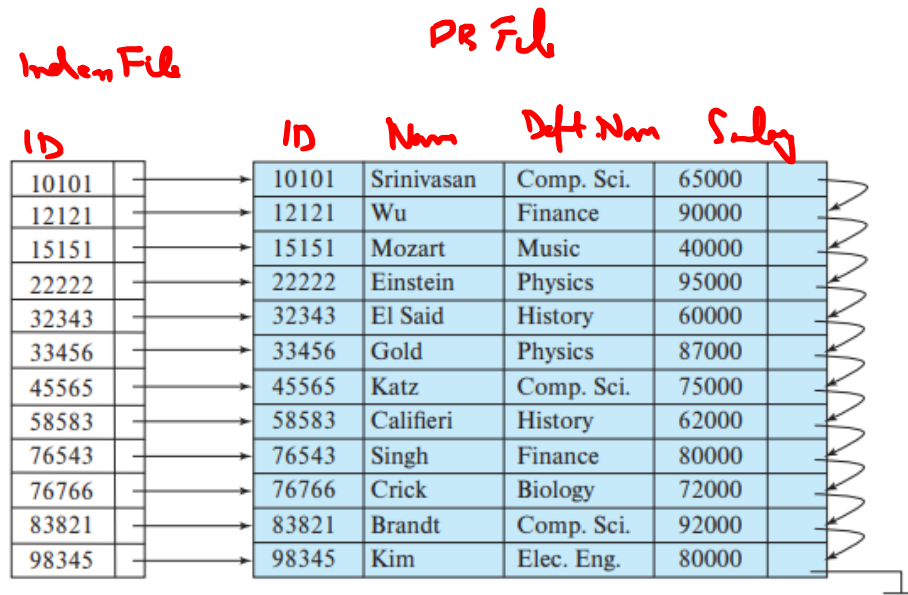
Index File

ID

ID			
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

Index File entries: 10101, 32343, 76766

Figure 14.3 Sparse index.

DB File

Index File

ID

ID	Name	Dept Name	Salary
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

Index File entries: 10101, 12121, 15151, 22222, 32343, 33456, 45565, 58583, 76543, 76766, 83821, 98345

Figure 14.2 Dense index.

Dense Clustering Index vs Dense Non-Clustering Index

- **Dense Clustering Index**
  - In a <u>dense clustering index</u>, the index record contains the search-key value and a pointer to the first data record with that search-key value. The rest of the data records with the same search-key value would be stored sequentially after the first record.

- **Dense Non-Clustering Index**
  - In a <u>dense non-clustering index</u>, the index must store a list of pointers to all records with the same search-key value.

Index File

Database File

Dept-name
Biology
Comp. Sci.
Elec. Eng.
Finance
History
Music
Physics

ID	Name	Dept-name	Salry
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

Dense Clustering

Index File

DB File

Dept-Name
Biology
Comp. Sci.
Elec. Eng.
Finance
History
Music
Physics

ID	Name	Dept-Name	Salry
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

Dense Clustering Index vs Dense Non-Clustering Index

- **Dense Clustering Index**
  - In a <u>dense clustering index</u>, the index record contains the search-key value and a pointer to the first data record with that search-key value. The rest of the data records with the same search-key value would be stored sequentially after the first record.

- **Dense Non-Clustering Index**
  - In a <u>dense non-clustering index</u>, the index must store a list of pointers to all records with the same search-key value.
  - As an alternative, an index record can point to a bucket that contains pointers to all the actual records with that particular search-key value.
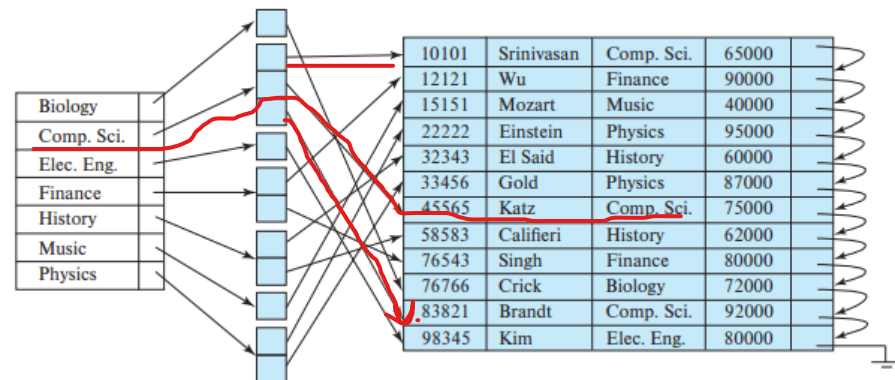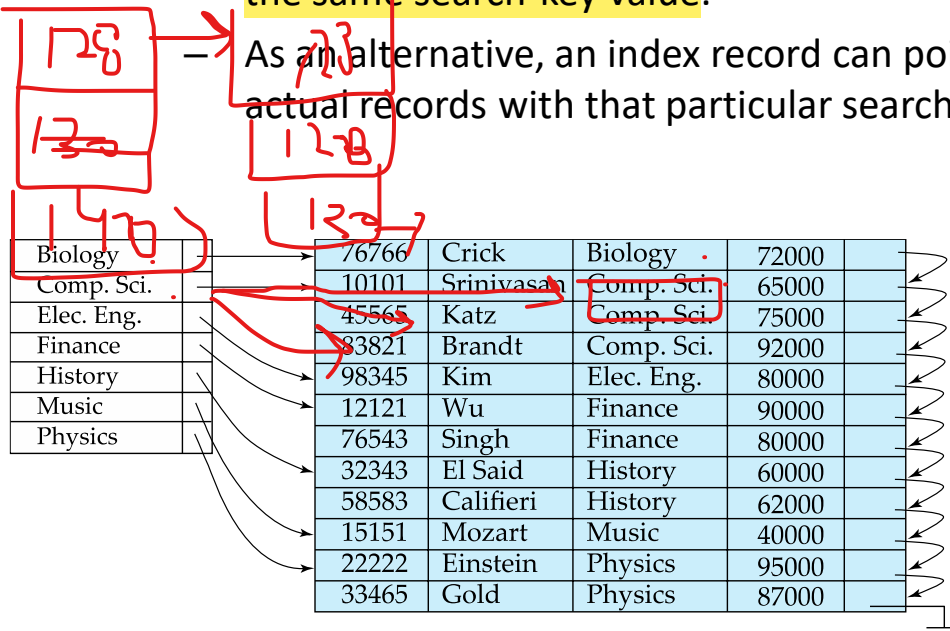
Sparse Clustering Index

- For a sparse clustering index
  - Each index entry contains a search-key value and a pointer to the first data record with that search-key value. To locate a record, we find the index entry with the largest search-key value that is less than or equal to the search-key value for which we are looking. We start at the record pointed to by that index entry and follow the pointers in the file until we find the desired record.
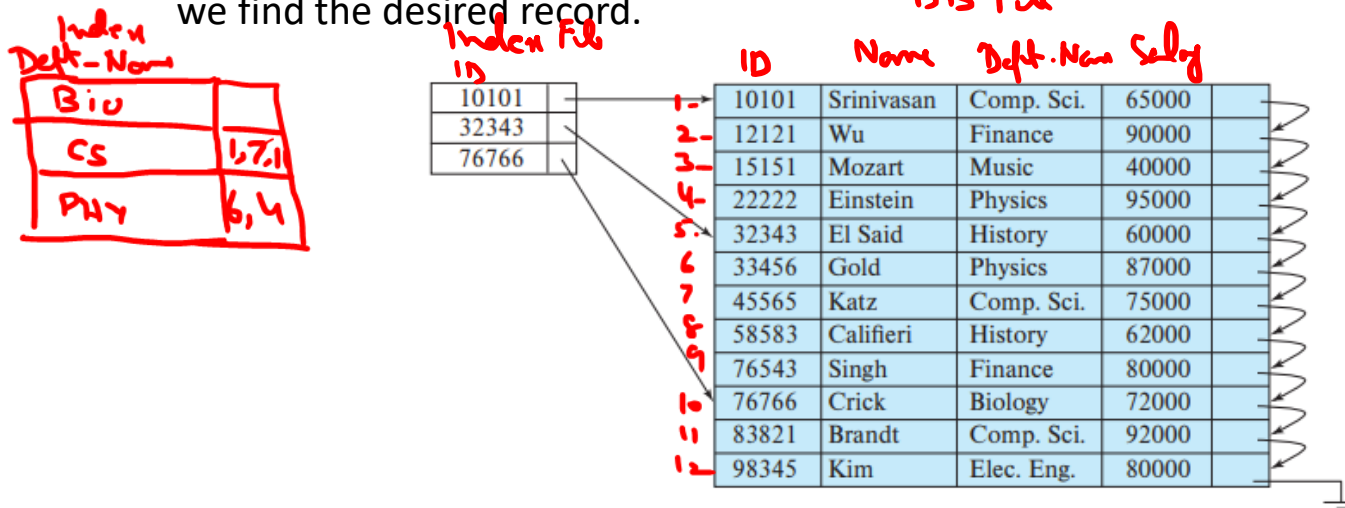

Figure 14.3 Sparse index.

- Sparse Non-Clustering Index?

Dense vs Sparse Indices: Tradeoff between Access Time and Space Overhead

- It is generally faster to locate a record if we have a dense index, rather than a sparse index.

- However, sparse indices have advantages over dense indices in that they require less space and they impose less maintenance overhead for insertions and deletions.

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

Figure 14.3 Sparse index.

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

Figure 14.2 Dense index.

Dense vs Sparse Indices: Tradeoff between Access Time and Space Overhead

- A good compromise
  - A sparse index with one index entry per disk block
  - Reason: The dominant cost in processing a database request is the time that it takes to bring a block from disk into main memory. Once we have brought in the block, the time to scan the entire block is negligible.
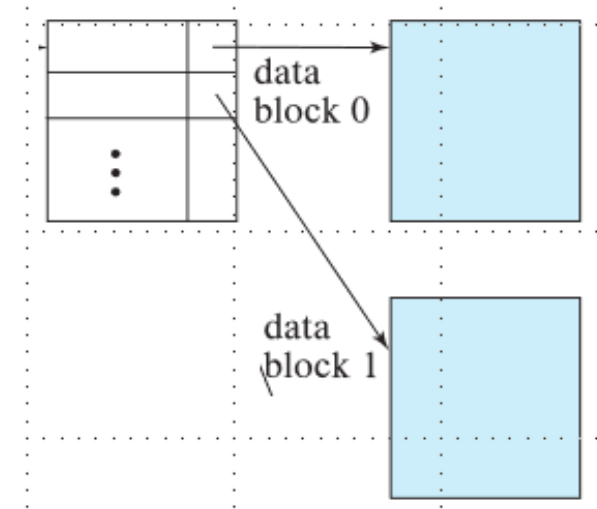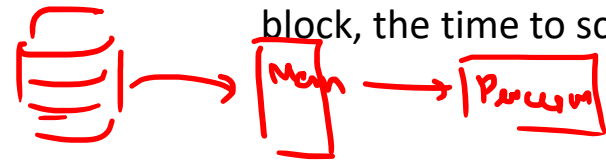
Figure 14.3 Sparse index.

Figure 14.2 Dense index.

Multi-Level Indices: Motivation

- If an index is small enough to be kept entirely in main memory, the search time to find an entry is low.

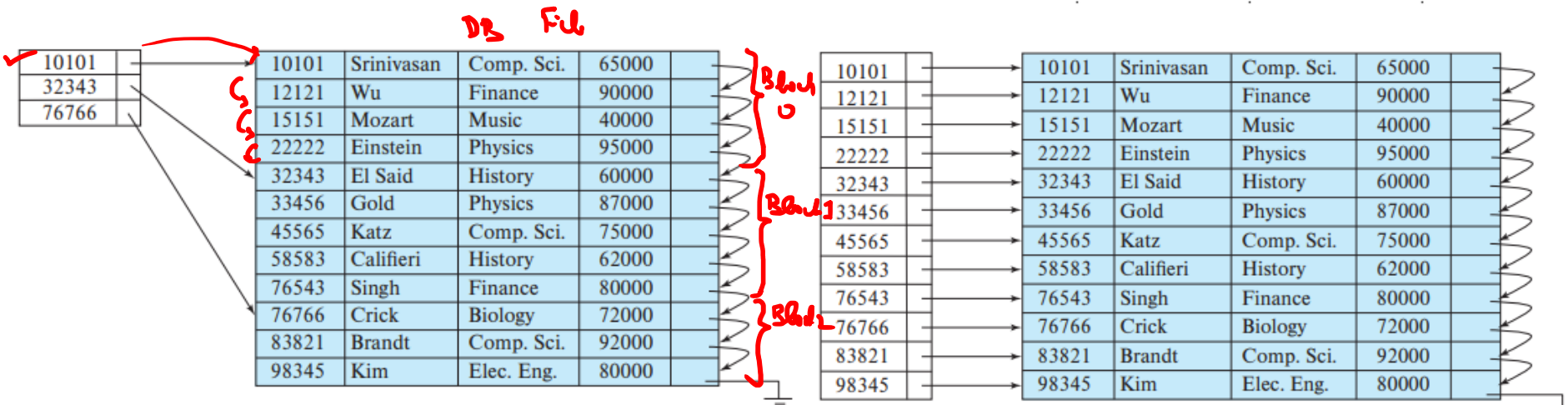- However, if the index is so large that not all of it can be kept in memory, index blocks must be fetched from disk when required.

HardDisk

Main Mem ← → Processor

IndexFile   DB_File

Block0

Block1

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

Figure 14.2 Dense index.

Multi-Level Indices

- If index does not fit in memory, access becomes expensive.
  - Solution: treat index kept on disk as a sequential file and construct a sparse index on it.

- Outer index
  - a sparse index of the basic index

- inner index
  - the basic 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.
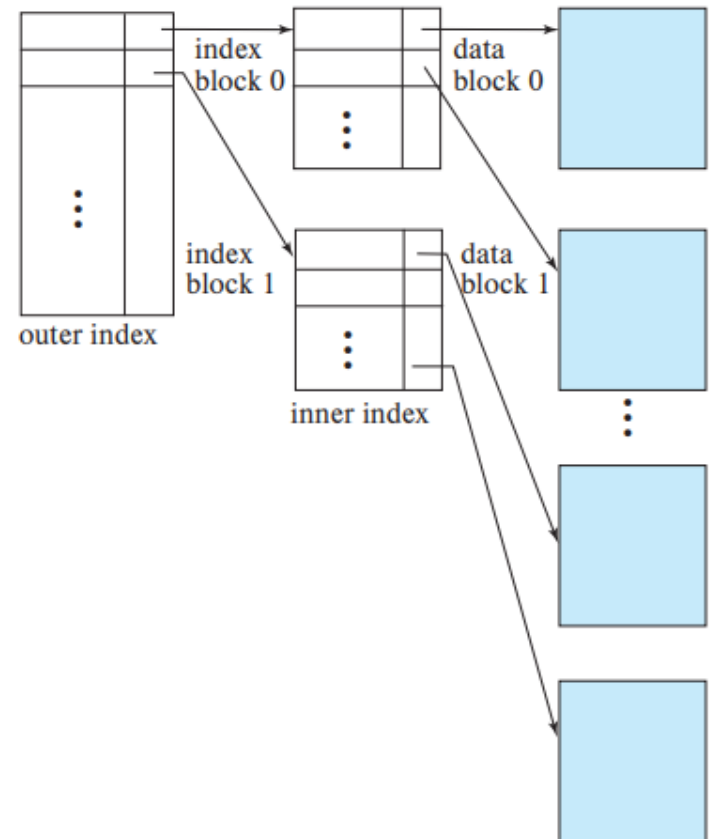
Figure 14.5 Two-level sparse index.

Index Update: Insertion

Biology			
Comp. Sci.			
Elec. Eng.			
Finance			
History			
Music			
Physics			

76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

Dense Clustery

Case 1 :- New Entry for "Physics"

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

Figure 14.3 Sparse index.

Case 2:- New Entry for "Economics"

Index Update: Deletion

76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

Biology, Comp. Sci., Elec. Eng., Finance, History, Music, Physics

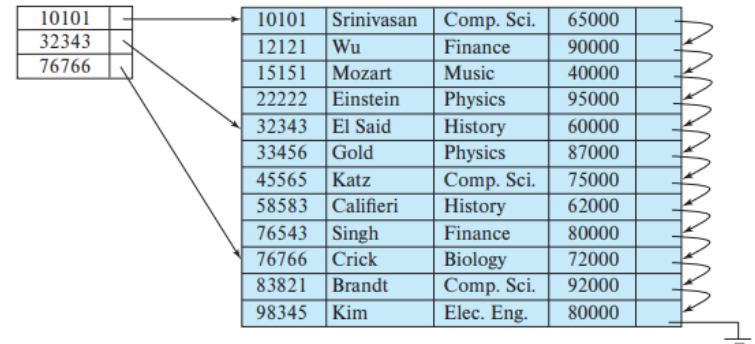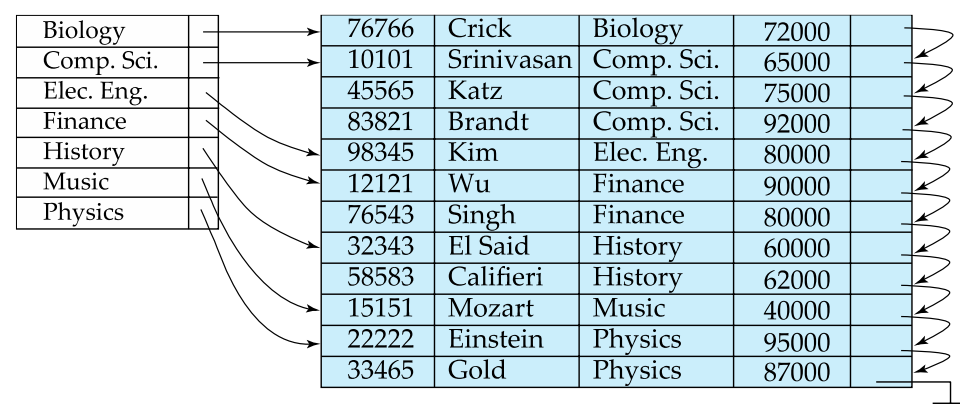|			
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.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

10101, 32343, 76766

Figure 14.3 Sparse index.

Case 1 :- Delete 22222

Case 2 : Delete 33465