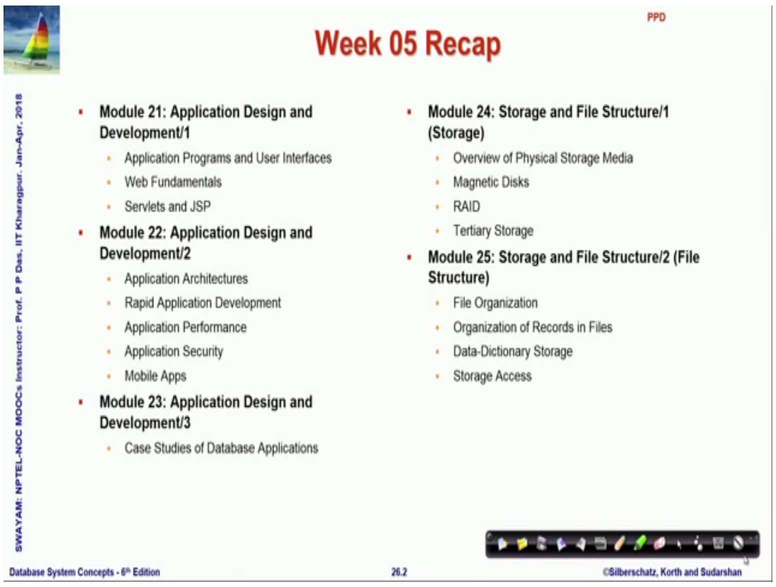
Database Management System
Prof. Partha Pratim Das
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture - 26
Indexing and Hashing/1: Indexing/1

Welcome to module 26 of Database Management Systems. In this module and the next 4; that is for the whole of this week we will talk about indexing and hashing.

(Refer Slide Time: 00:33)



PPD

Week 05 Recap

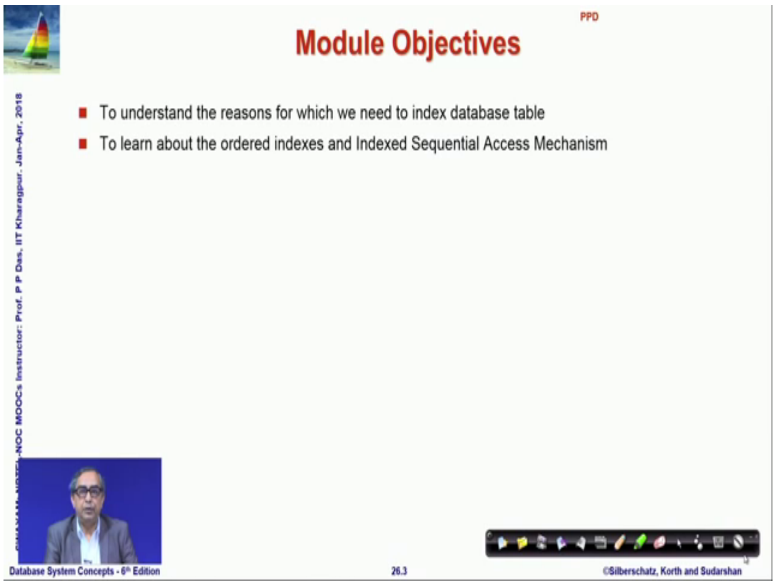
- Module 21: Application Design and Development/1
  - Application Programs and User Interfaces
  - Web Fundamentals
  - Servlets and JSP
- Module 22: Application Design and Development/2
  - Application Architectures
  - Rapid Application Development
  - Application Performance
  - Application Security
  - Mobile Apps
- Module 23: Application Design and Development/3
  - Case Studies of Database Applications
- Module 24: Storage and File Structure/1 (Storage)
  - Overview of Physical Storage Media
  - Magnetic Disks
  - RAID
  - Tertiary Storage
- Module 25: Storage and File Structure/2 (File Structure)
  - File Organization
  - Organization of Records in Files
  - Data-Dictionary Storage
  - Storage Access

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.2 ©Silberschatz, Korth and Sudarshan

So, this is a quick recap of what we did in the last week primarily talking about application development and then specifically; we focused on storage and file structure that is how a database file can be stored how it can be organized and in a manner so that, we have efficient representation for that now.

(Refer Slide Time: 01:01)


PPD

Module Objectives

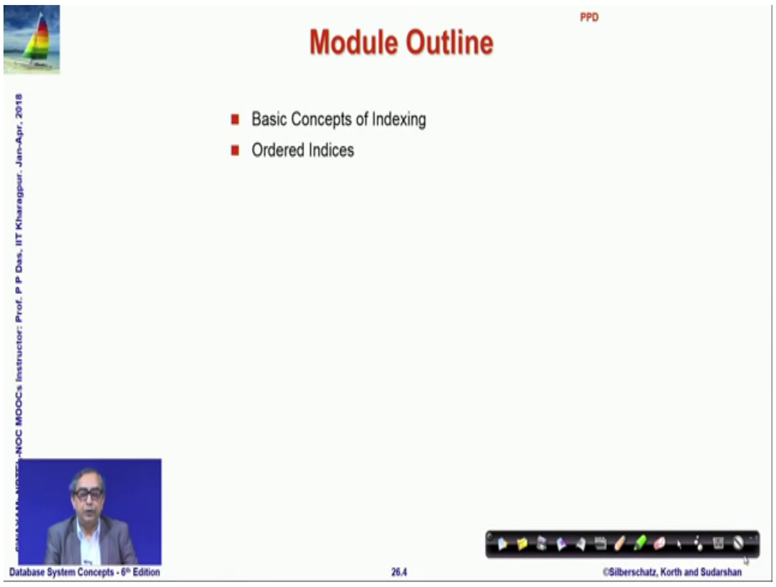
- To understand the reasons for which we need to index database table
- To learn about the ordered indexes and Indexed Sequential Access Mechanism

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.3 ©Silberschatz, Korth and Sudarshan

We will move on from there to and focus on the basic feature of a database system, which is the ability to find information in a very fast manner the ability to update in a very fast manner. And we will see the reasons for which we do something on the database tables called indexing and we will talk about ordered index in this module and about the index sequential access mechanism.

(Refer Slide Time: 01:31)


PPD

Module Outline

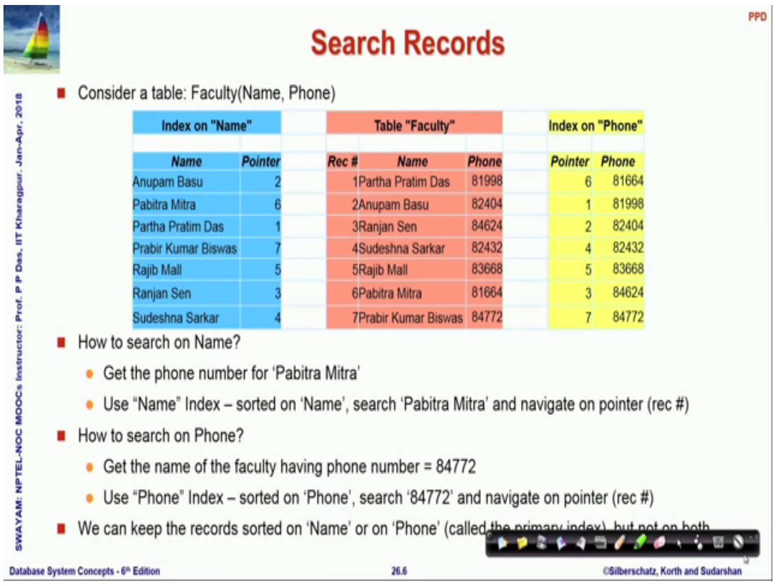
- Basic Concepts of Indexing
- Ordered Indices

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.4 ©Silberschatz, Korth and Sudarshan

So, introduction of the basic indexing concepts and the order indices and we start with the basic concepts.

(Refer Slide Time: 01:40)



PPD

Search Records

- Consider a table: Faculty(Name, Phone)

Index on "Name"			Table "Faculty"				Index on "Phone"	
Name	Pointer		Rec #	Name	Phone		Pointer	Phone
Anupam Basu	2		1	Partha Pratim Das	81998		6	81664
Pabitra Mitra	6		2	Anupam Basu	82404		1	81998
Partha Pratim Das	1		3	Ranjan Sen	84624		2	82404
Prabir Kumar Biswas	7		4	Sudeshna Sarkar	82432		4	82432
Rajib Mall	5		5	Rajib Mall	83668		5	83668
Ranjan Sen	3		6	Pabitra Mitra	81664		3	84624
Sudeshna Sarkar	4		7	Prabir Kumar Biswas	84772		7	84772

- How to search on Name?
  - Get the phone number for 'Pabitra Mitra'
  - Use "Name" Index – sorted on 'Name', search 'Pabitra Mitra' and navigate on pointer (rec #)
- How to search on Phone?
  - Get the name of the faculty having phone number = 84772
  - Use "Phone" Index – sorted on 'Phone', search '84772' and navigate on pointer (rec #)
- We can keep the records sorted on 'Name' or on 'Phone' (called the primary index), but not on both

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.6 ©Silberschatz, Korth and Sudarshan

So, consider a very simple example, let us consider an example where there is a relation faculty which has name and phone number and additionally. So, just focus on the middle part the pinkish part of the table which is table faculty besides the two attributes I have put a serial number for the records which kind of can be considered as internal numbers of the database to identify each record.

Now, let us say let us assume that we have to search on names suppose we have a query that get me the phone number of Pabitra Mitra. So, you can see that Pabitra Mitra the record name occurs at position 6. So, if we have to get the phone number, then naturally we have to look at all these entries from one end to the other till we come across Pabitra Mitra match the name Pabitra Mitra and we can access the phone number.

Now, similarly if we have to search for a phone number we will have similar situation. So, if we want to get the phone numbers of the faculty having phone number 84772. Again we will have to search on the phone number from one end to the other and find the name of the faculty. Now, certainly this will mean that if there are n records in the database then we will need or the order of about n by 2 comparisons to be done or accesses record accesses to be done before.

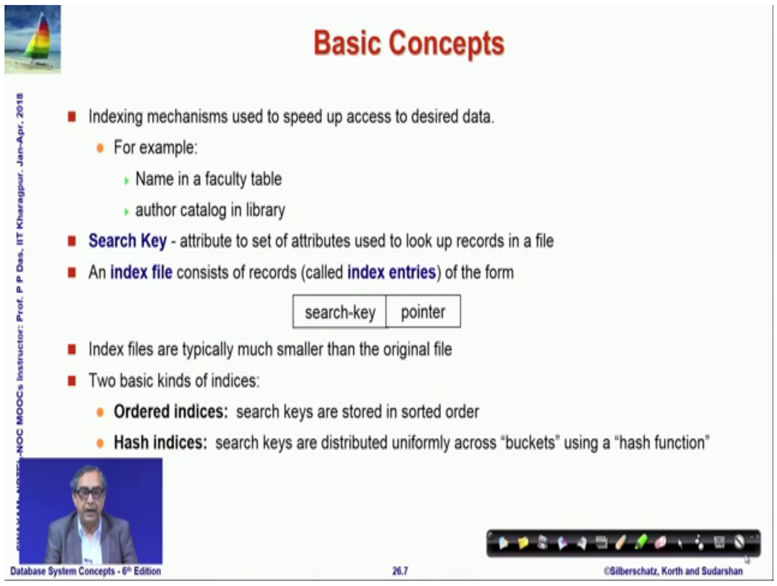
We can find the desired record which is certainly not a very efficient way of doing things and you know from your knowledge of basic algorithms that; if we have a set of data and we want to find a particular one then we can make it efficient by actually keeping the data in a sorted manner and doing some kind of a binary search that is one one possible mechanism through which you can do that.

So, we can use that same context same concept now and just look at the left side the blue part of the blue part tableware, what I have done have collected all the names of the faculty and I have sorted them in lexicographically increasing order and with that I have kept the record number as a pointer. Now, since this is a sorted array. So, here I can search for Pabitra Mitra for the first query in a binary search. So, at least in the log n order of comparisons I should be able to find professor Pabitra Mitra and get the fact; that it is record number is 6 navigate to the record number 6 in the table in the middle and take out the phone number.

Similarly, without disturbing the actual faculty table I can build a similar kind of supportive table on the right the yellow one where I collect all the phone numbers and I have sorted on the phone numbers I can again do binary search on the phone number 84772 and find the phone number find the record number where this phone number occurs and go and find the name of the faculty.

So, you can see that naturally this gives us a alternate way of finding out and certainly you would say that we could have kept the record sorted on name or on phone number, but certainly if we keep it sorted on name the first query we will get benefited the second query will still take a linear time if we get keep it sorted by phone number the first query will take a linear time. So, if we cannot actually keep the data sorted on both and therefore, this is just an alternate mechanism called the indexing mechanism through which even though the original data is not sorted by maintaining some auxiliary data we can actually make our search mechanism more efficient and that is the core idea of indexing.

(Refer Slide Time: 05:39)



Basic Concepts

- Indexing mechanisms used to speed up access to desired data.
  - For example:
    - Name in a faculty table
    - author catalog in library
- **Search Key** - attribute to set of attributes used to look up records in a file
- An **index file** consists of records (called **index entries**) of the form

search-key	pointer

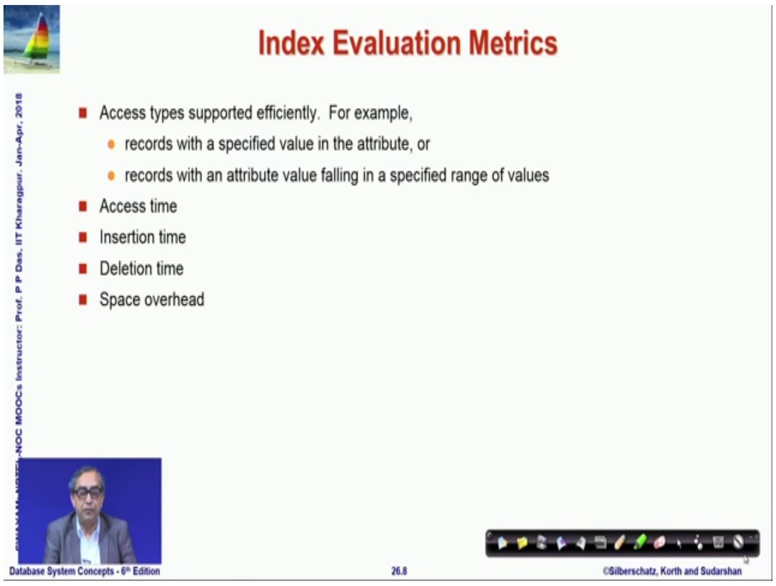
- Index files are typically much smaller than the original file
- Two basic kinds of indices:
  - **Ordered indices:** search keys are stored in sorted order
  - **Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function"

Database System Concepts - 6th Edition 26.7 ©Silberschatz, Korth and Sudarshan

So, indexing mechanism is used to speed up accesses to desired data. So, as as you have just seen. So, what we search on is called the search key and an index file consists of the index entries as you have just seen it has a search key and the pointer pointer is the record number or the internal you know address of the record where it occurs and certainly I have shown an example where there just two attributes in general there will be a large number of attributes. So, the entry of every index would be much smaller than the corresponding record.

So, the overall index file will be a much smaller one than the original and we can index it and there are two basic ways to index and; that is what we will discuss through these modules one is called ordered index where this search keys are stored in sorted order as you have just seen and the other is hash indices where the search keys are distributed randomly across different buckets. Using, concepts of hash function which you must have studied as a part of your data structure course.

(Refer Slide Time: 06:44)



Index Evaluation Metrics

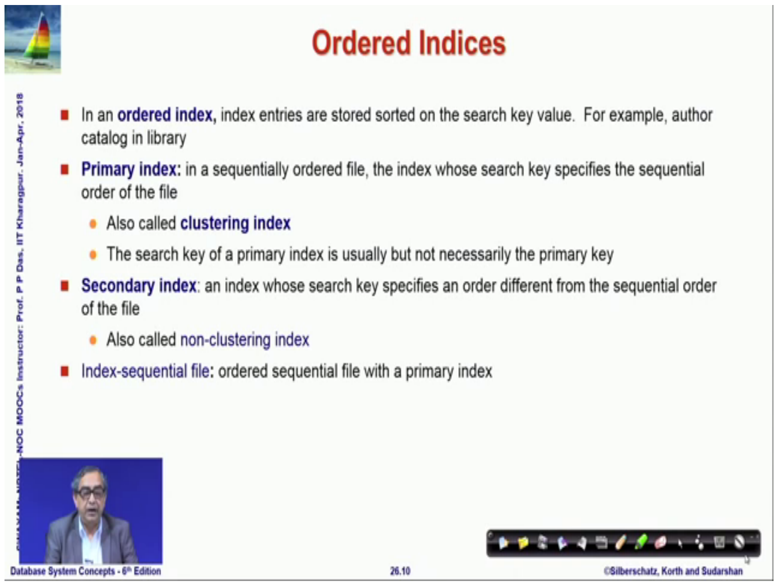
- Access types supported efficiently. For example,
  - records with a specified value in the attribute, or
  - records with an attribute value falling in a specified range of values
- Access time
- Insertion time
- Deletion time
- Space overhead

Database System Concepts - 6th Edition 26.8 ©Silberschatz, Korth and Sudarshan

Now, if we I mean why what should we index on the basic question is should we index on all attributes should we index on one attribute should we index on combination of attributes. So, access. So, there are certain measures to decide as to what is a good way to index. So, that will be that will be measured against the basic requirements of the database that is I should be able to index in a way.

So, that the access time the insertion time the deletion time all these should get as minimized as possible and as you have just seen indexing might mean maintaining additional structures. So, that overhead of space should also be minimal. So, with that with indexing we should be able to do at least certain kind of accesses where a particular value matches the attribute of a record or falls within a range of values in a record those kind of searches those kind of queries should become very efficient and these metrics will have to always keep in mind.

(Refer Slide Time: 07:51)



Ordered Indices

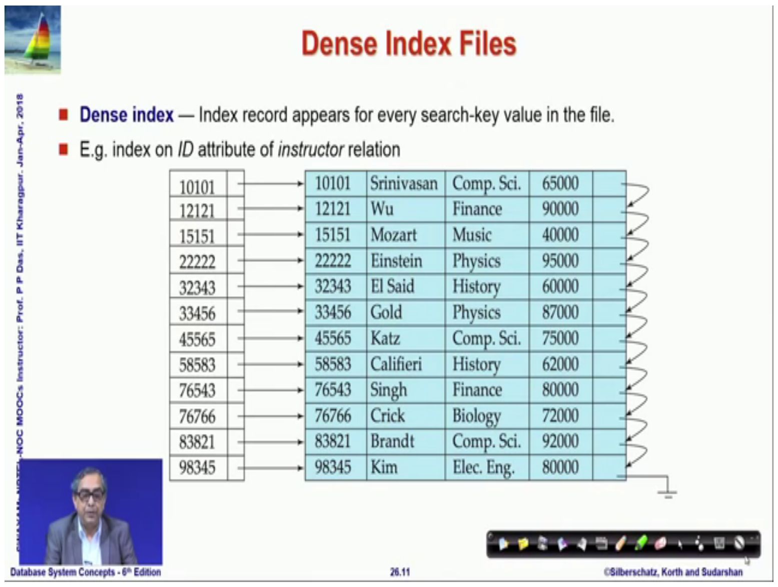
- In an **ordered index,** index entries are stored sorted on the search key value. For example, author catalog in library
- **Primary index:** in a sequentially ordered file, the index whose search key specifies the sequential order of the file
  - Also called **clustering index**
  - The search key of a primary index is usually but not necessarily the primary key
- **Secondary index**: an index whose search key specifies an order different from the sequential order of the file
  - Also called non-clustering index
- Index-sequential file: ordered sequential file with a primary index

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.10 ©Silberschatz, Korth and Sudarshan

So, let us look at the first concept of ordered index which is pretty much like what we have just sent. So, in a sequentially ordered file the index whose search key specifies the sequential order is known as the primary index. So, the sequential ordering is done based on that primary index it is called also called the clustering index and please keep in mind that the search key of a primary index is usually the primary key, but not necessarily the primary key it could be different from the primary key also and if I create an index who search key is different from the sequential order of the file then we say it is a secondary index and it is also called the non clustering index. So, index sequential file is ordered sequential file which is ordered on the primary index.

(Refer Slide Time: 08:44)



Dense Index Files

- **Dense index** — Index record appears for every search-key value in the file.
- E.g. index on *ID* attribute of *instructor* relation

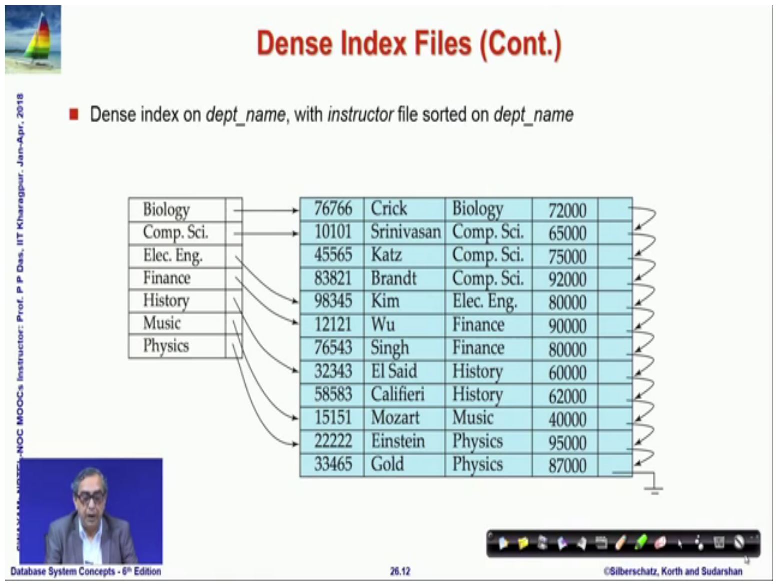
10101	10101	Srinivasan	Comp. Sci.	65000
12121	12121	Wu	Finance	90000
15151	15151	Mozart	Music	40000
22222	22222	Einstein	Physics	95000
32343	32343	El Said	History	60000
33456	33456	Gold	Physics	87000
45565	45565	Katz	Comp. Sci.	75000
58583	58583	Califieri	History	62000
76543	76543	Singh	Finance	80000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

NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.11 ©Silberschatz, Korth and Sudarshan

So, here I show an example and this is example of dense index file. So, you can see the blue part is actual table which has different fields the id the name of the faculty, the department, and the salary and this is as you can see that it is completely sorted on the id in the increasing value of id and on the left the white is basically just the ids and with every idea we have a pointer they record the number possibly of the record that it corresponds to here all the indices that feature in the table are also put on the index file and therefore, it is called a dense index you should also note on the right that the records are interlinked through a chain I mean kind of they are being formed in terms of a linked list whose purpose would be seen later on.

(Refer Slide Time: 09:40)



Dense Index Files (Cont.)

- Dense index on *dept_name*, with *instructor* file sorted on *dept_name*

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

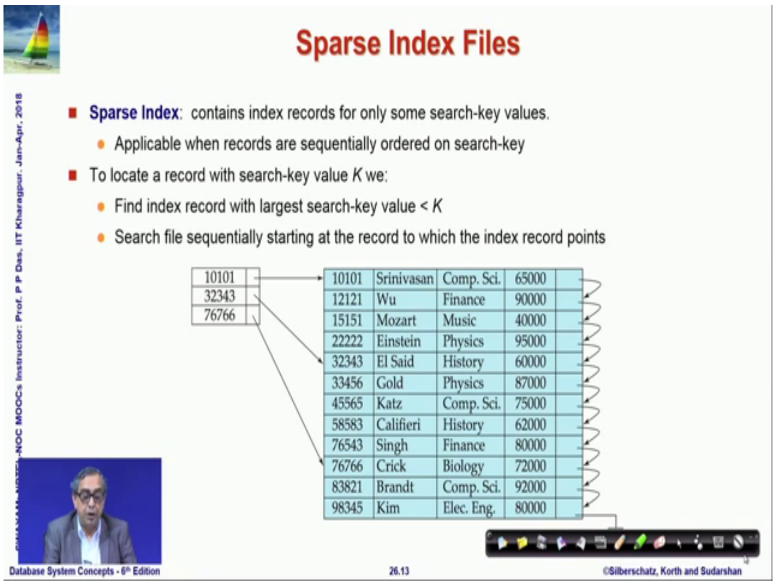
Database System Concepts - 6th Edition 26.12 ©Silberschatz, Korth and Sudarshan

Now, instead of doing id if I want to do a dense index on department name, then naturally the sequential file has to be indexed primarily on the department name and as you can note that unlike the id which is also the primary key and therefore, every id value was unique the department name is not a primary key it is a different attribute which allows for multiple value multiple records having the same attribute value and therefore, when we sort we have one instance of biology, but three of computer science two of finance and so, on.

So, when we make the index we made the index collect all the distinct values of the department names and for every department name we put a pointer we put the index marking the first record in the sequential file with that department name. So, you can see that your computer science points to the record starting with 1 0 1 0 1 and then the; it goes on to the next two records.

So, now you can understand why we need the link list; because if we are looking for the all those teachers all those faculty who are associated with department computer science, then I can find it on the index file with the department name, computer science traveled to the record first record in that which is of Srinivasan and then keep going on this list through the linked list that we have on write and find the record for Katz and record for Brandt and till we moved to the next record for Kim which does not match the department name anymore.

(Refer Slide Time: 11:26)



Sparse Index Files

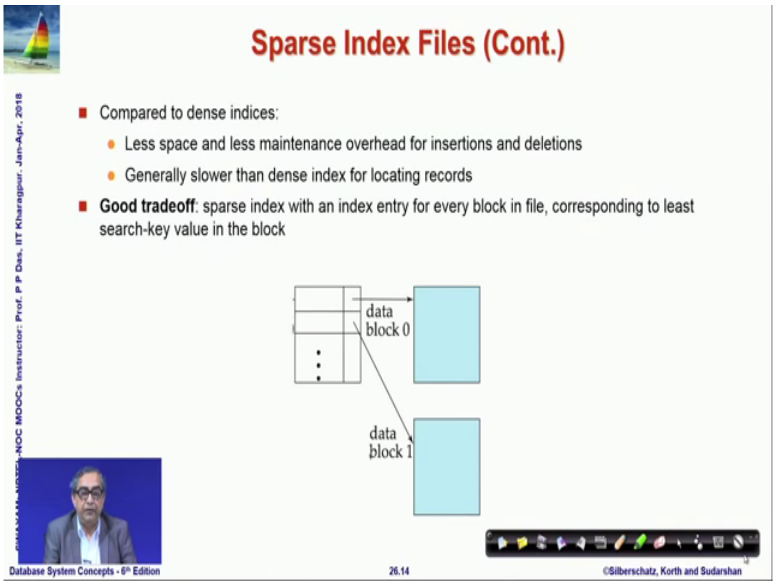
- **Sparse Index**: contains index records for only some search-key values.
  - Applicable when records are sequentially ordered on search-key
- To locate a record with search-key value *K* we:
  - Find index record with largest search-key value < *K*
  - Search file sequentially starting at the record to which the index record points

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

Now instead of dense index we could also do sparse index. Tweak parse index it means that instead of maintaining all the search key values that exist in the file we just take a subset of that and we maintain those in the index file. So, here again we are showing a index sequential structure with id being the primary index and we have chosen three of the ids and kept them in the index file. So, in the sorted order so, this if you want to say look at a record with search K value K we can find the index record with the largest key value largest search key value id value which is less than K and then search sequentially or onwards because these are all linked together in the sequential manner.

(Refer Slide Time: 12:18)



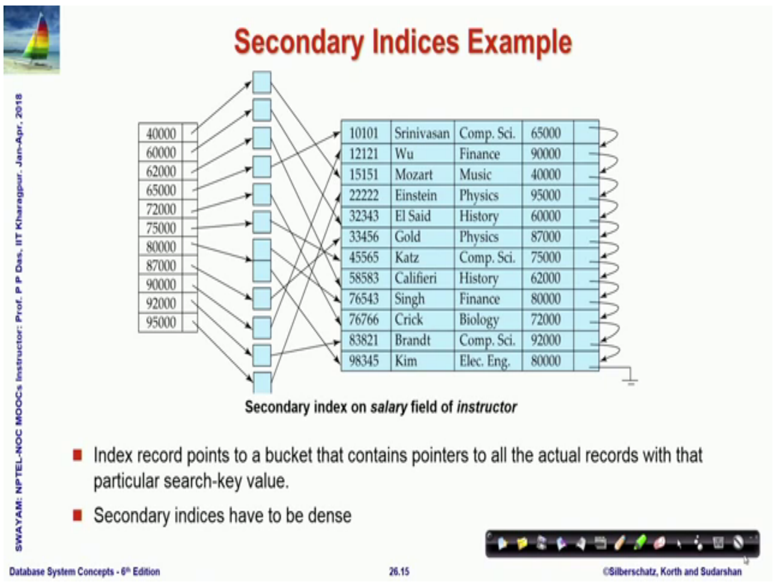
Sparse Index Files (Cont.)

- Compared to dense indices:
  - Less space and less maintenance overhead for insertions and deletions
  - Generally slower than dense index for locating records
- **Good tradeoff**: sparse index with an index entry for every block in file, corresponding to least search-key value in the block

data block 0
data block 1

So, naturally compared to the dense index of sparse index takes less space and therefore, less overhead for maintenance when we do insertion deletion you must have already noted that if I were dense index then every time I insert a value it will have to be the dense index file will also have to change changes will be required there for insertion as well as for deletion for sparse index it will not be required, because it will just be required when I am actually changing the record or creating a record which is existing on the sparse index.

(Refer Slide Time: 13:11).



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

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

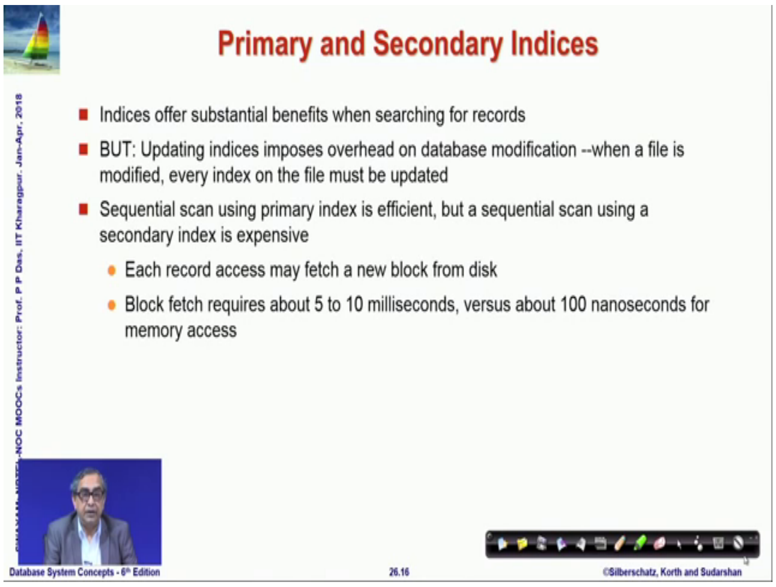
Database System Concepts - 6th Edition 26.15 ©Silberschatz, Korth and Sudarshan

So that way it is it is better than the dense index, but certainly in the dense index I can go to any record directly from indexing in the sparse index I need to first index and then go sequentially. So, it will be in general slower in terms of performance and will need to look at this tradeoff now it is also. So, these were the ways to do the primary indexing where we decide the order in which we actually keep the record sorted or the fields on which we do that and the index associated with it, but once since the data can be primarily indexed on one or couple of attributes and that gets fixed.

But we may want to search for other values also to make that efficient we create a secondary index. So, here we show an example where the primary index is id. So, the whole recall records are sorted in terms of that and we are creating an index on the salary. So, that we can quickly find the salary of and given the salary of an employee we can quickly find the employee or the employees who get that salary.

So, you can see that for secondary index now it is quite possible that there are multiple entries for the same value of salary for example, if you look at the salary eighty thousand that is received by Professor Singh as well as Professor Kim. So, if you look at the index file of the index sequence of the salary values you will find that against 80,000 we have two entries which marked to two different records corresponding to the faculty who are drawing that salary. So, secondary index naturally has to be dense and is created when we want we feel that there is a need to quickly search on that field or that set of fields and we will discuss more about those issues later on.

(Refer Slide Time: 14:46)



Primary and Secondary Indices

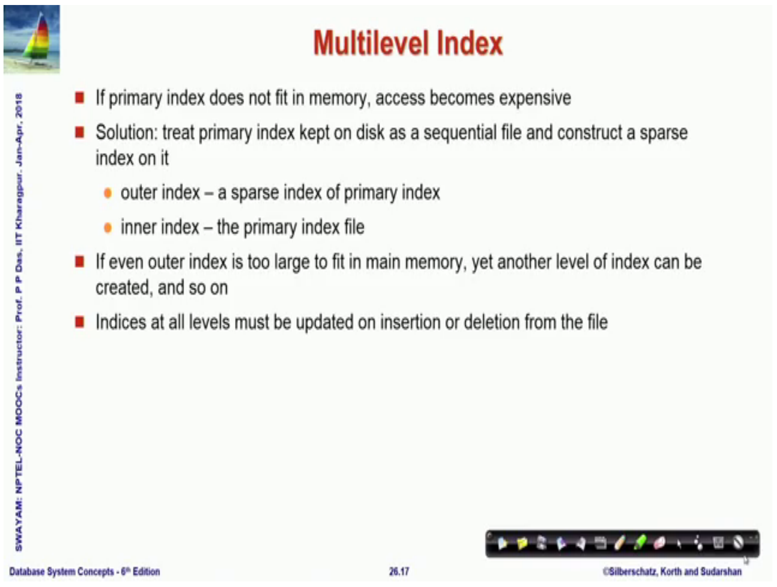
- Indices offer substantial benefits when searching for records
- BUT: Updating indices imposes overhead on database modification --when a file is modified, every index on the file must be updated
- Sequential scan using primary index is efficient, but a sequential scan using a secondary index is expensive
  - 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

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018
Database System Concepts - 6th Edition
26.16
©Silberschatz, Korth and Sudarshan

So, indices offer substantial benefits when searching for records, but updating index impose over it; because if you create an index whenever you insert a record or a delete a record or you change the value of a field which is indexed in a record then certainly all these indices will have to be also updated and therefore, while your access time significantly reduces, because of indexing your actual update insert delete update time will increase and therefore, the indexing will have to be done carefully.

So, sequential scan using primary index is efficient, but sequential scan using secondary index is expensive. So, you will have to bring them in blocks and that will require couple of millisecond versus the amount of time that you need in the memory access. So, these are the factors that we will have to take into consideration and we will talk more about those.

(Refer Slide Time: 15:44)



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

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.17 ©Silberschatz, Korth and Sudarshan

Now, if I have a primary index then naturally to be able to access the records I will have to first access the primary index and then do a search in that and then travers the point at to go to the actual record in the file. So, to access the primary record we would prefer that if the primary index can actually fit into the memory. So, that I can do a in memory search like we do the binary search in a in an array.

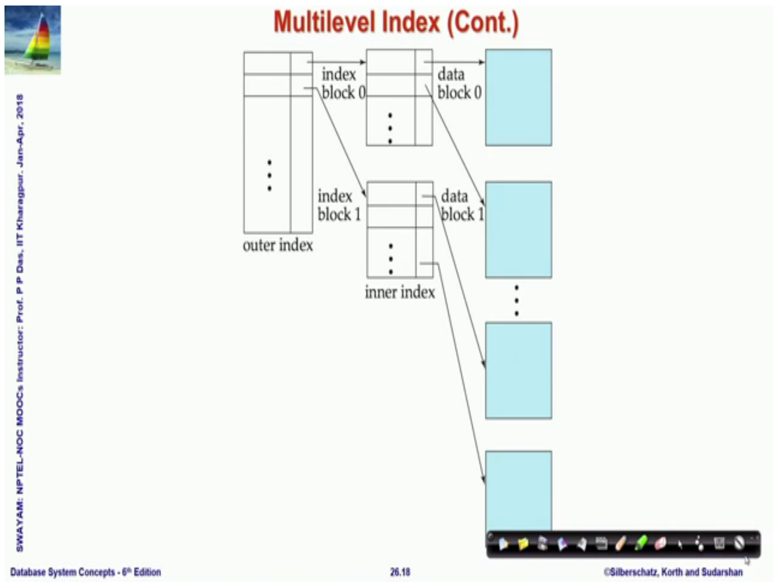
So, because if the primary index is large then that also has to exist in the disk and therefore, bringing that primary index into the memory and then searching will add two additional access cost of the disk and you have already seen in the earlier modules as. So, how these costs are expensive these accesses are expensive. So, primary index if it is not in the memory then we usually have a lot of disadvantage.

So, to take care of that if you have primary index actually is large enough. So, that it does not fit in memory then we simply apply the notion of indexing once again on the primary index file itself. So, we construct a say on the primary index we construct a past sparse index. So, which is called the outer index which is a sparse index of the primary index and in that index is the actual primary index file.

So, if now in turn the outer index the sparse index of the primary index also is too large to fit in memory then you need to do yet another level of indexing on it and. So, on till you come to a index of list which fits into a memory can be directly accessed. So, the cost for that so, this is what is called a multi level indexing.

Because, you are following multiple levels for indexing and the cost naturally of update insert delete increases; because now all of these multiple levels of indices will have to be updated.

(Refer Slide Time: 17:48)



Multilevel Index (Cont.)

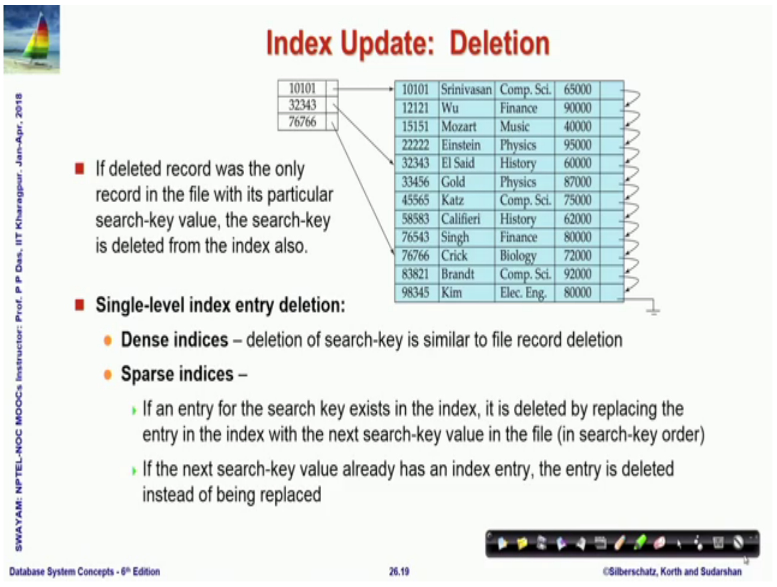
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.18 ©Silberschatz, Korth and Sudarshan

When you do an update so, this is what is a view of the multi level index you can see the outer index which is sparsely index and index those lead to different blocks of primary index of the of the inner index which is the primary index and then you traverse to the specific block where your record is expected. So, with this you since your outer index are in memory. So, you need to do one disk fetch for finding out the inner index block which should be one disk page or disk block one axis and then based on that to find another access to for the block in which the record exists.

So, with this you would be able to manage with to block accesses in this case and that is how the multiple this would not have been possible if in this case you would not have done the sparse outer index, because you would have required the different parts of the inner index of the primary index to be fetched repeatedly from the disk till you act could actually find the final result in that.

(Refer Slide Time: 18:59)



Index Update: Deletion

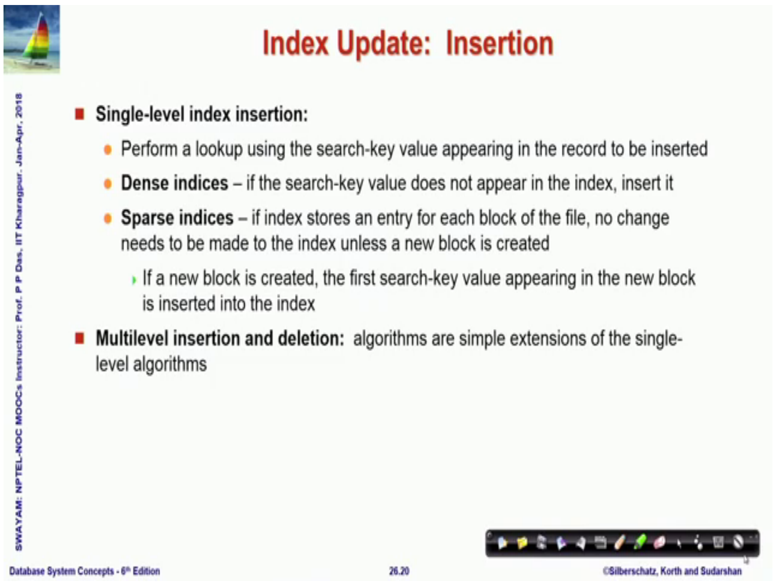
10101	
32343	
76766	

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

So, updating the index particularly if you do deletion then the if it is a dense index then the deletion of the search key is similar to deletion of the actual record in the file because it is dense if these parts, then naturally you will have to take care of some of the cases, because if it falls within a range then just deleting it would not matter, but if it falls on the boundary where it is actually sparsely indexed then that will have to be appropriately updated.

(Refer Slide Time: 19:32)

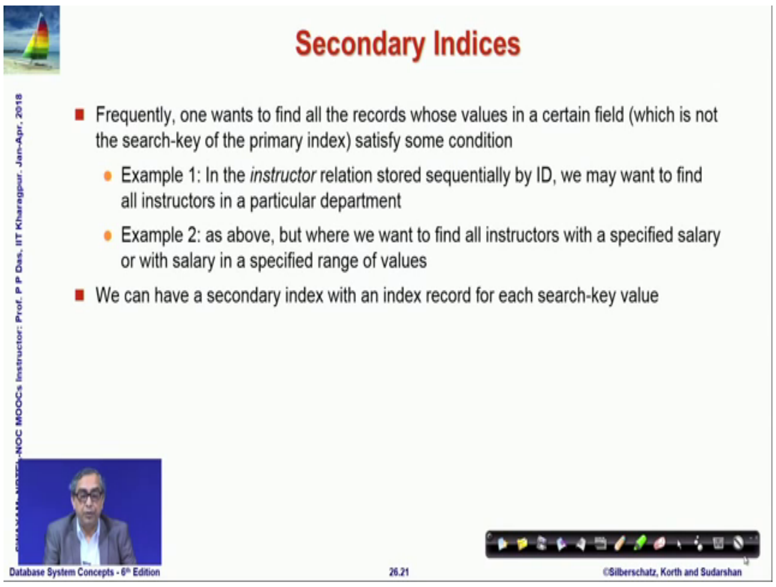


Index Update: Insertion

- **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

Similar thing will have to be taken care of in terms of insertion. So, first you will have to look up to find out to where the record needs to be inserted and then if it is a dense index if the search key does not appear in the index then you will have to insert it in case of sparse index you will have to do the additional care that whether it already belongs to the range and or whether if it will have to be a new block has to be created then it has to be also entered in terms of the sparse index and if you have multi level indexing then insertion deletion will be extensions of these basic algorithms.

(Refer Slide Time: 20:14)



Secondary Indices

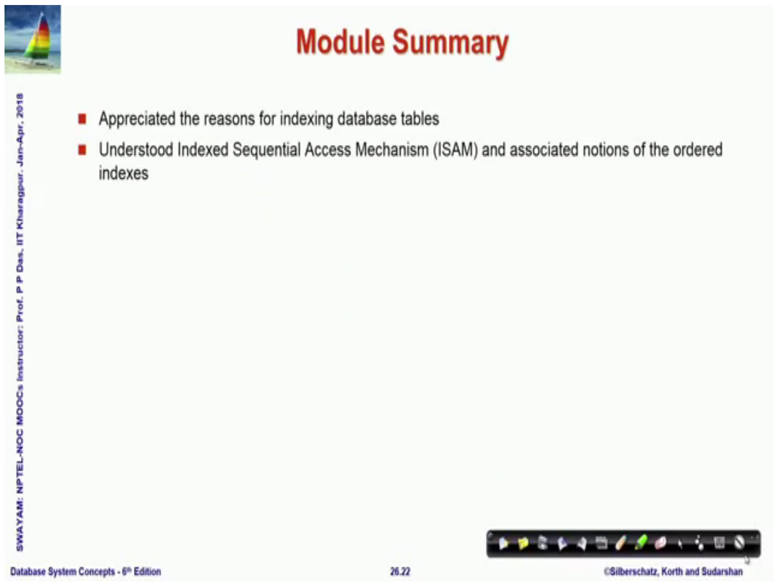
- Frequently, one wants to find all the records whose values in a certain field (which is not the search-key of the primary index) satisfy some condition
  - Example 1: In the *instructor* relation stored sequentially by ID, we may want to find all instructors in a particular department
  - Example 2: 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

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.21 ©Silberschatz, Korth and Sudarshan

For secondary indices frequently we want to find all records where certain value in a certain field which is not the search key or the; of the primary index satisfy some condition. And, we can have a secondary index with an index record for each of this search key values depending on what we expect what we would think or likely fields on which more search will be done.

(Refer Slide Time: 20:45)



Module Summary

- Appreciated the reasons for indexing database tables
- Understood Indexed Sequential Access Mechanism (ISAM) and associated notions of the ordered indexes

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur. Jan-Apr, 2018

Database System Concepts - 6th Edition 26.22 ©Silberschatz, Korth and Sudarshan

Or more range queries will be done. So, to summarize we have taken a brief look at the basic reasons for indexing database tables which is to speed up the process of access insert and delete and we have seen that primarily indexing primarily focuses on speeding up the access process.

So, in that maintenance of indexing whereas, insertion and deletion might have some additional overhead, but since insertion and deletion both inherently needs access to be done because you can insert only when you have tried to access and have not found exactly where the record should occur or for deletion; obviously, you need to first find the record to be able to delete it.

So, even though it is focused indexing is focused on improving the access time it actually improves the time for access insert delete all of that, but we will have to keep in mind that in the process there are certain overheads of index update for insert and delete that will have to be kept as a minimal and the additional storage requirement will also have to be kept as a least overhead. So, with this we will close this module we have taken the basic look at the index sequential access mechanism this is called index sequential access mechanism associated with different database index files.