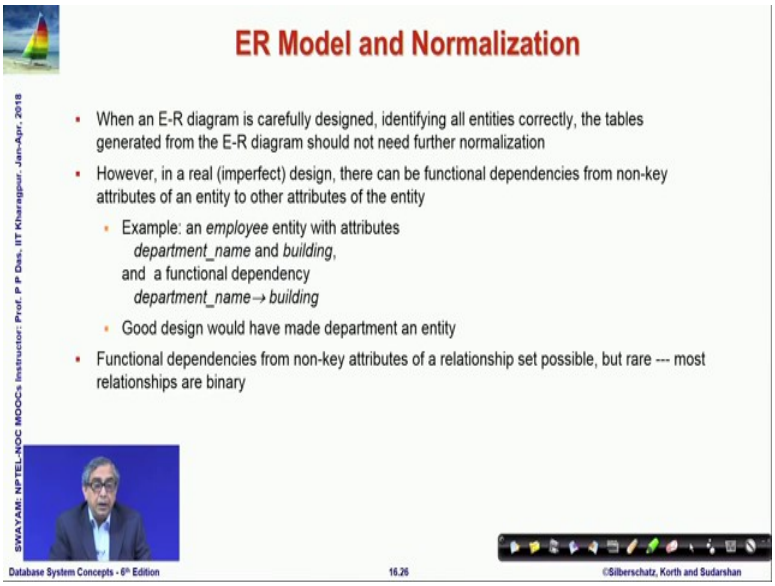
(Refer Slide Time: 28:33)

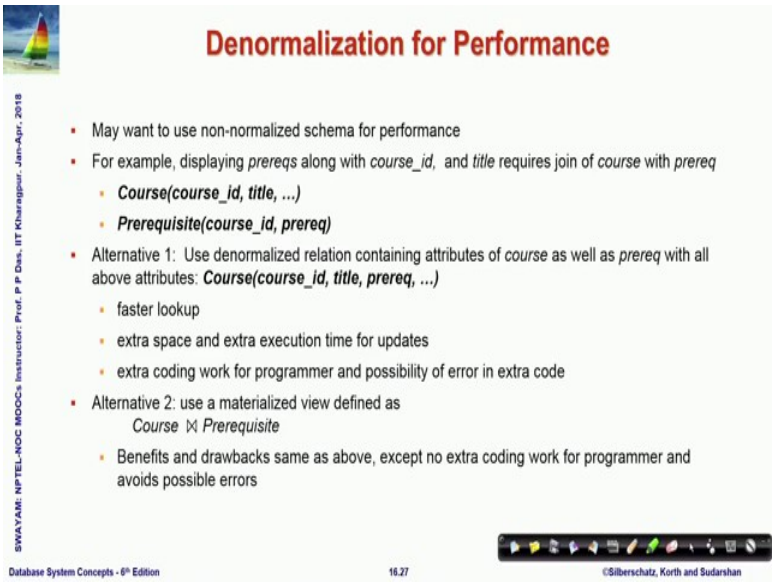


So, there are possible all different possible tracks that can happen. So, if we have taken the ER model track, then frankly speaking if the ER model is carefully designed, then every entity defined in that ER model will have only the dependency; which are the determining super key.

So, just recall the employee department building kind of situation we discussed earlier. So, an employee entity has attributes department name and building, and there is a functional dependency from department_name to building. So, what it means that in the entity relationship diagram itself we didn't do a good job. If we had done a good job then we would have identified that the department itself is an entity and therefore, would not feature as an attribute on the employee. So, it would have been I mean right there, we would have if we had called it as a separate entity, then that is equivalent of what we are doing now taking the relation and then breaking it down through decomposition.

So, functional dependencies from non-key attributes of a relationship are possible ah, but are rare. So, mostly the relationships are binary, and if you do a careful design of the ER model then many of these deep exercise of normalization you will not have to go through.

(Refer Slide Time: 29:55)



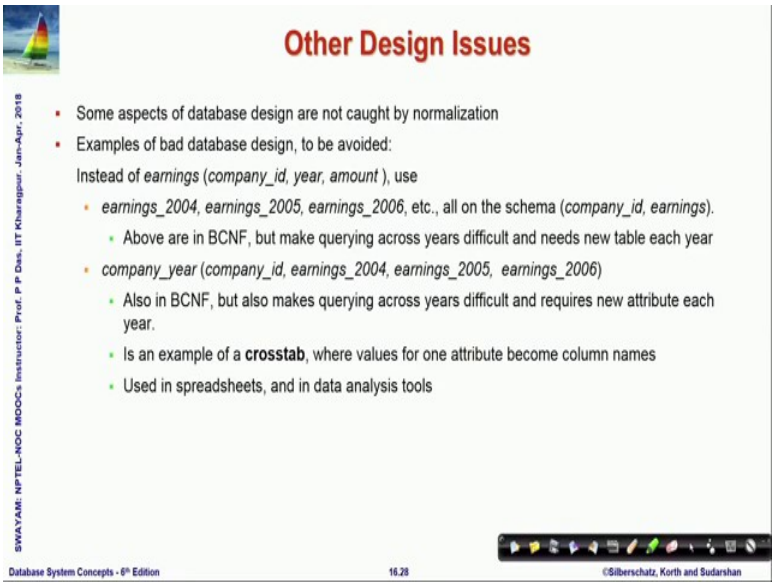
It should also be kept in your view, that at the times when you want to de normalize want to use denormalized relations, because if you have normalized and the only way to get back the original view is to perform join. So, if we of course, if you have prerequisites and if you want to say, view or print prerequisites with that title and course id naturally you will have to take a join with the course, which is expensive.

So, one option could be first alternative could be that, you use a denormalized relation, where the course prerequisite is actually included in the course and you know that will have you know violations of some of the normal forms. Because, there are there are functional dependencies between them, but that will certainly lead you to first a look up, because you have them in the same table you do not need to perform join. But you need extra space, exact extra execution time for update, because you have redundant data you have redundancy while programming on that coding on that, because of this redundancy there could be possibility of error, because any of these anomalies can happen and your code will have to now take care of that.

So, it does help in certain way in terms of getting a better efficiency, but it there is a there is a cost to pay in a different way also the other alternative could be you can have a materialized view, which is actually the join in course of prerequisite. In terms of performance it has a same benefit or the costs as you say, but only thing is you will not need to do that extra coding. So, it is better from that perspective.

So, always keep the issue of denormalization in view, and we do a careful design that if it is very frequent that, you will have to compute a join then, you might want to sacrifice some of the redundancy some of the you know possibilities of having anomaly, and still have a you know denormalized design in your database.

(Refer Slide Time: 31:59)



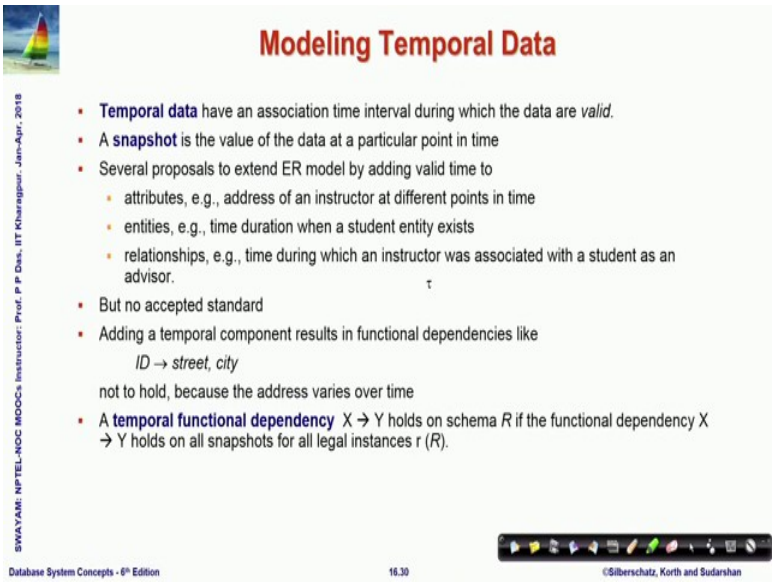
There are several other issues of design, which do not get captured in what we have designed. For example, let say very regularly we are we have returns, income tax returns to submit and we will be maintain income tax return tax your sales tax return and on all that, and you maintain your accounts book of transactions debit credit accounted and so on. Now naturally, these are all bound in terms of one-year effectivity.

So, when they come when in the next year comes, then you need a separate you know set of records to be done for that year. So, how do you. So, if you if you have such a table where you along with the company idea of year and amount and then how do you take care of this situation, because one way could be that you have all of these you take out year, from the attribute and you have separate table in every year. So, you will have to create new table and remember their name. So, if queries which run across year will be difficult to do.

The other way could be that, you every New Year you start renaming you know you do a year where your earnings from different years are shown on different columns. So, you are basically every year you have the result in terms of a different attribute. So, that also

is not a very good solution for a database it is something which is with the spreadsheets will often use, but in terms of data which has a certain format and needs to be you know redefined from scratch, at a at a different time frame in a different way, then you will come across these issues.

(Refer Slide Time: 33:56)



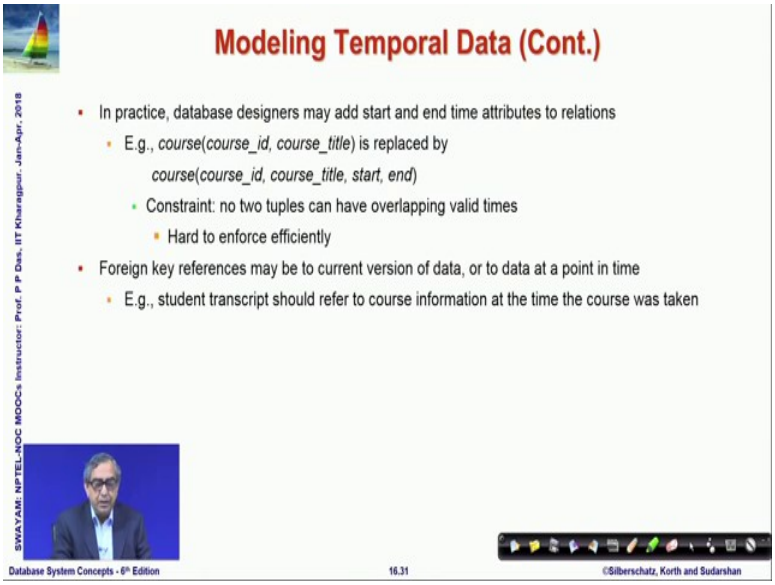
Ah let me close with just pointing out that, if we have a one kind of data that we have not looked at which are temporally in nature, that is all that we have said is the attributes and their values. So, if we put at value to an attribute then that value is taken to be the truth for now, and for the past and for the features. So, if that value changes, then you completely erase that in the database.

So, for example, today I stay at a certain address, tomorrow I may take up a different quarter my address has changed. So, in our design if there is against my employee id there is an address given, then once I change my quarter my address will change it will not be possible to recollect, what address I resided in say 2017.

So, temporal data of such kind, temporal data I am sorry just of such kind have an association with an interval. So, a snapshot often does not solve the problem. So, you have you have to decide, how you do that? Whether you can you would like to put some attributes, which specify the timestamp or you would like to I mean really have multivalued attributes, denoting the different time frames where they are they may have taken effect, there is no accepted standard. And the fact that, if you if you know that it

keeps on changing with time then your original dependencies might get affected they will they will change as well. So, these are the things that you will have to take care ah.

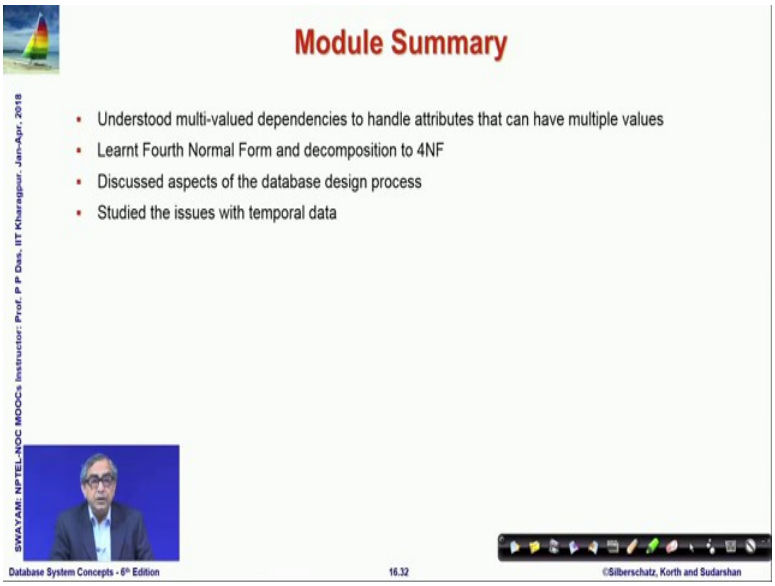
(Refer Slide Time: 35:30)



This is another style, that many a times when you have to say that ok. This course with this title existed from this semester, a different semester the title may have changed. So, you can put a start and end attribute with which specifies what is the time for which the remaining attributes made sense, a good design, but these also have issues because, if you do this kind of temporal intervals, then how do you make sure that between 2 records the intervals are not overlapped. So, you are not saying that at the same time this course had them X this of course, also had name Y. So, they have to be disjoint. So, how do you check for this ah this consistency of data, there is no easy way to do that.

So, the foreign key references and all those. So, handling of temporal data is another aspect which will have to be looked into very carefully, in the design and you will need to do some kind of design compromise and implementation has to take care of those issues.

(Refer Slide Time: 36:33)



So, to summarize we have ah taken a full look into the multivalued dependencies and tried to understand what happens, when your attributes get multiple values. Learn the 4th normal form for that and the decomposition into that, and most importantly we have tried to summarize the core database design process. That we have been discussing for the last four modules, this is the fifth one including this and we have understood that and we have talked a little bit about the temporal data.

And with this we close our discussions on the relational database design, and from the next module we will move on to other aspects of the database systems.

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

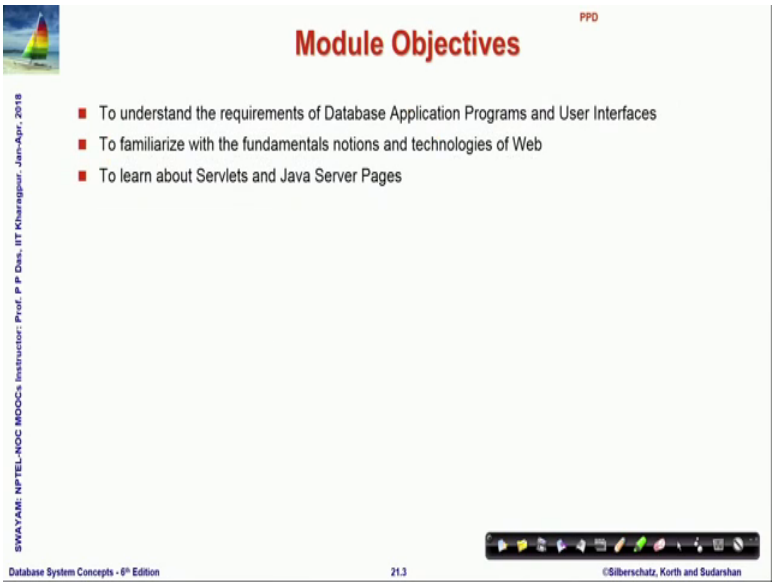
Lecture – 21
Application Design and Development

Welcome to module 21 of Database Management Systems in this module and the next too we will discuss about application design and development.

(Refer Slide Time: 00:31)

In the last week we have for the whole week in the five modules we have discussed about relational database design; in depth we have looked into what are the different aspects of that.

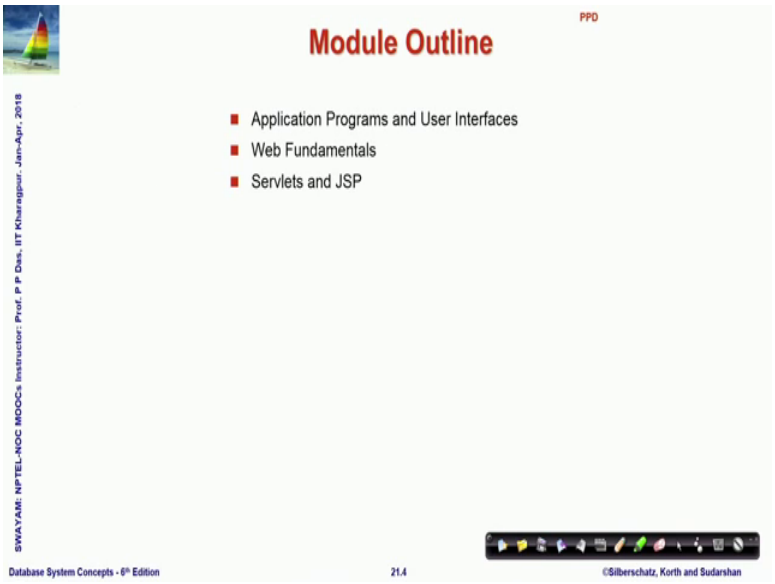
(Refer Slide Time: 00:43)



And now we get into the core issue of if I have a relational database design existing and that is populated with the data then based on that how do we develop, how do we create an application where the user can interact and actually get answers to the questions that the user has or the user can actually update the data create new data, remove old data and so, on.

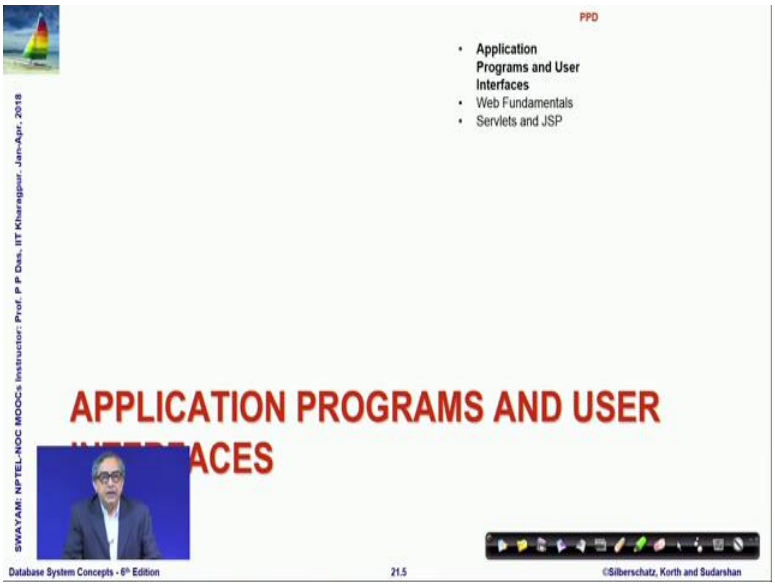
So, we would in that process like to familiarize with the fundamental notions of notions and technologies of web applications and specifically we would learned about servlets and Java server pages.

(Refer Slide Time: 01:30)



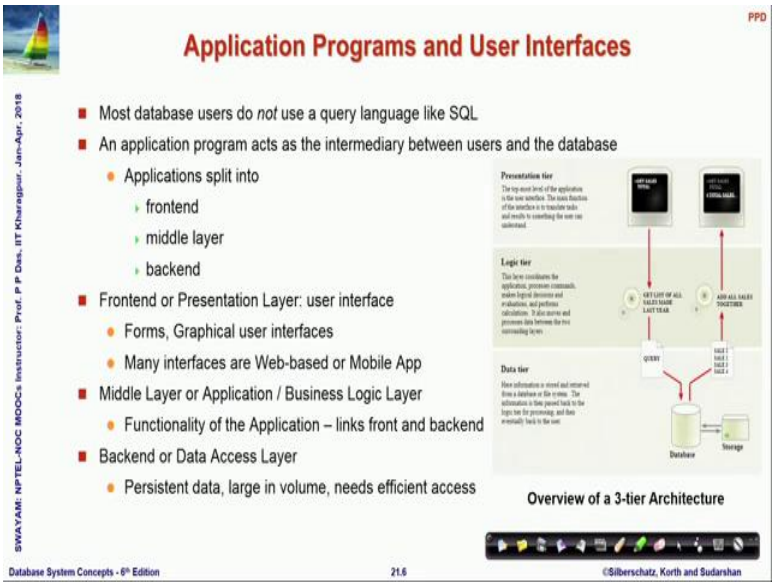
So, these are the free topics to be covered.

(Refer Slide Time: 01:34)



So, we first start with application programs and user interfaces.

(Refer Slide Time: 01:41)



So, the situation is the where we do have a relational database design it is populated with the required data, but how about the interaction with the user incidentally most of the you database users do not interact with the database or query the database using language like SQL because as you have seen it is not a very friendly language and it is not presentable in a way which I would we would always expect or we would like.

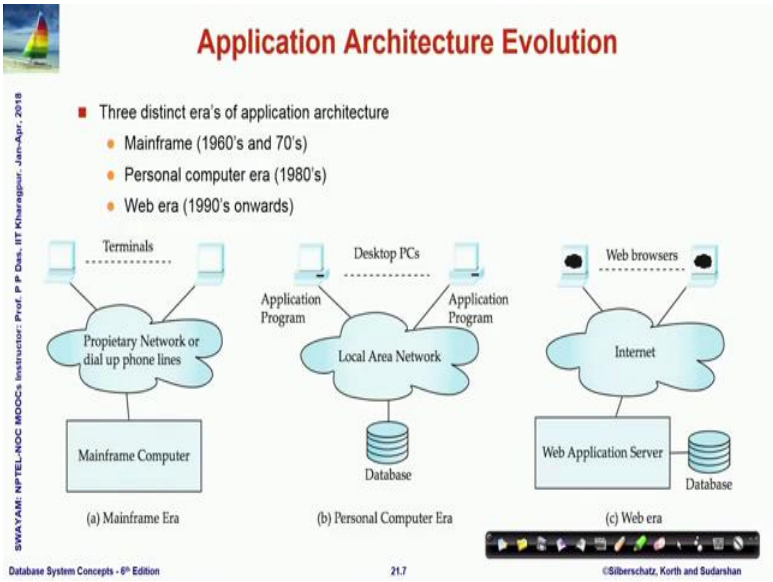
So, usually an application program acts as a as we say as an intermediately between the user and the database. And it is often split into three layers as we will say frontend middle layer and backend. So, on the right I have shown these three layers we will talk more about the this is just a representative diagram. So, the frontend is the user interface it is also called the presentation layer. So, it is the layer where it is the part of the application where there are forms GUIs and different ways to input as well as get output of the data.

Which is directly interacting with the user then we have a middle layer or its also called the business logic layer or the application layer where the functionality of the application the required operations of the or desired behavior of the application are coded and it is kind of acts as a link between the frontend and the backend and what is the backend?.

Backend is an actual data access layer or it is where the persistent data the database that we have created exist it is typically large in volume need sufficient access and so, on. So, in this module we will try to understand as to how such what are the different

requirements and what are the different technologies involved in creating such a layered application.

(Refer Slide Time: 03:55)



Now, if we historically look at; so, here we are just showing three phases initially 60s and 70s where the first database applications started then the interaction used to be takes based from the terminal.

And the those to directly connect to the main frame computer where the data existed through either a direct connection dial up phone or proprietary network. And as we move to the 80s; then we saw the advent of local area networks to application programs or desktop would interact to with the database through these local area networks. And beyond that we have the what we call the web era which is 1990 onwards we in the that is that is about roughly the last 30 years where typically the applications are now based on web browsers.

So, the frontend where we actually interact are web browsers and that connect to the web application server the database everything through an internet. And I must tell you at this point that when you say this is the architecture it does not necessarily mean that the cloud shown as internet will have to be the web, it could be an internet which is created with a set of systems within your organization which we typically call as an intranet or couple of organizations across.

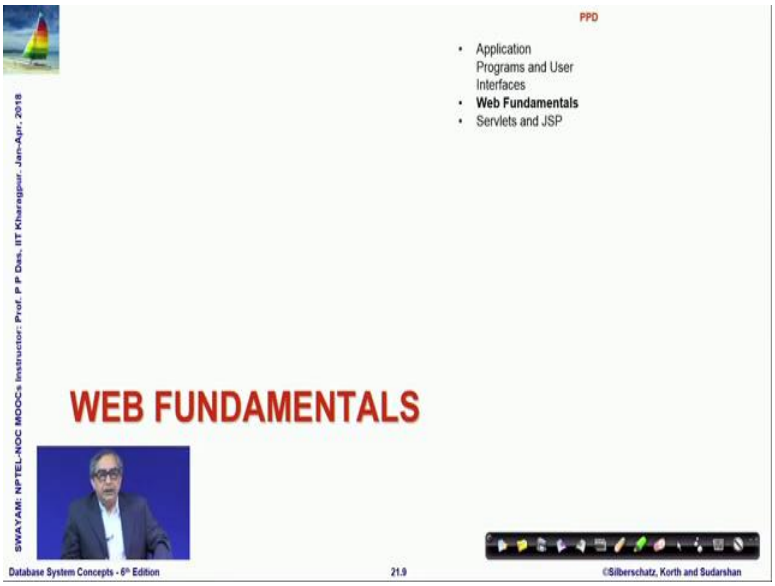
Which we say are extranet or it could even be a set of systems which are connected through the internet protocol within your lap or it could even be a single computer in which all these layers are integrated together, but by internet we mean it is a internet protocol and technologies will be used for doing this interaction.

(Refer Slide Time: 05:52)

So, web interface has become the de-facto standard which gives a very distributed access to the database enables large number of users to access together. And it avoids the requirement of downloading or installing specialized code into that all that we need is just a web browser. And we have seen we are living through a variety of applications which are of this kind the banking application, the airline and railway reservations car rental hotel booking or web mail systems we will check mail in Gmail or Yahoo those are all different web interfaces through which we actually access a the required set of databases.

And every even every enterprise operations the ERP are now web based and that is become a de facto standard. So, in the web interface along with a web interface what has been imagined of let of the last about 10 years are mobile interfaces that we are getting use to using such applications from our mobile phone or tablet. And these are similar in architecture and workflow as of the web application, but there are significant differences to and at a later point in the next module, we will discuss about the specific requirements of mobile apps in this context as well.

(Refer Slide Time: 07:27)



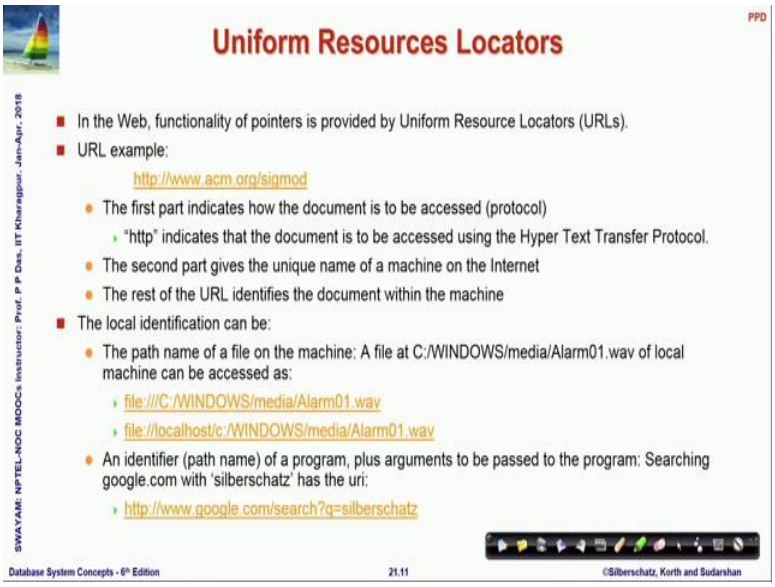
So, before we move forward in terms of the details of how to build these applications; we need to familiarize ourselves with the basic notions of the web as such. So, we call them as web fundamentals.

(Refer Slide Time: 07:43)

So, web is a distributed information system which is based on hyper text a hyper text is one where you have a part of the text available to you and then you have links we say our hyper links which can connect to the different documents contents.

Which are locate at other places at other servers and typically most web documents are hyper text documents which are formatted in terms of what we know as HTML Hyper Text Markup Language; you will be familiar with that I am sure. So, they contain text and along with that they can have a other components like images, video, the text as specifications for font colors style all that. And in addition there are forms which can be used to enter data and send them back to the web server.

(Refer Slide Time: 08:35)



Now, naturally when we operate on the web we need to have the functionality of pointing to different resources and this is done by what is known as URL or uniform resource locator. So, that is a URL is a procedure to which you can identify and point to a certain specific location of content. So, here I am showing an example of such a URL all of you be familiar with URLs.

But just to look into the different components the first component http : this http is actually a tells us the way the content would be accessed and this is typically called a protocol http its stands for hyper text transfer protocol which allows you different text to be accessed.

The second component in this URL which is between the two forward slash and the next slash www [dot] acm [dot] org identifies uniquely identifies a machine on the internet you will understand this is the symbolic name and the actual machine has what is known as an IP address.

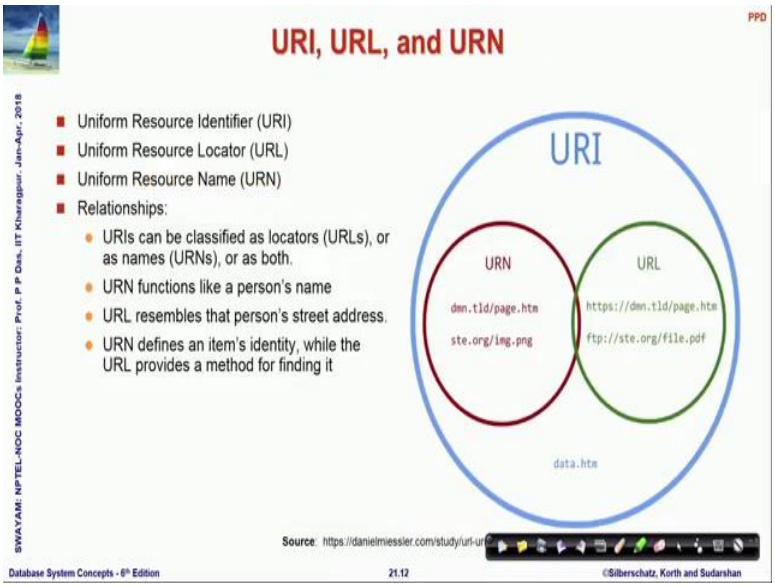
And we will not go into those details, but for us it is enough to understand that www [dot] acm [dot] org here is uniquely relatable to a particular machine on the internet. And the last part which is remaining sigmon and if there are more and more and more parts, then it identifies the document inside within that machine. So, URL can be used in a multiple other ways also.

For example I can use URL locator to specify a file in by machine for example, I have shown an example of an AVI file in my C drive in windows and that is done through a similar URL where the protocol is not http the protocol is file telling me that it is actually residing in my local machines.

So, I have shown two ways to look at that and you will see that between the two forms; if you look at the second form you can easily understand that the machine to be identify this is called the local host. And in the first form that part is missing because it is by default the machine where I am running this code and rest of it is same which is basically the identifying the document to be to be located in that machine.

Similarly, this such URL can also in the last example you can see that www [dot] Google [dot] com is the basic machine where I am putting the URL and then the rest of it is search.

(Refer Slide Time: 11:56)

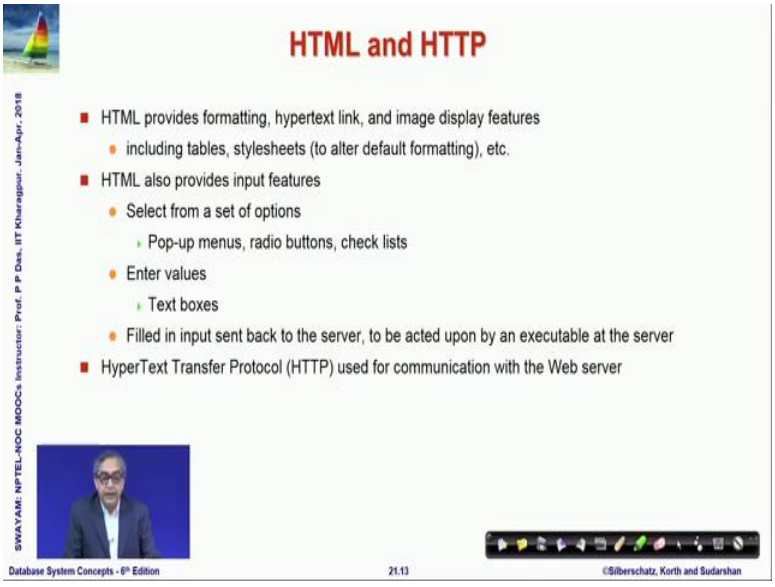


So, which actually takes it to a document where I tell the search to be performed and then there are parameter to this form the parameter is q is equal to silberschatz which is equivalent to same that I am asking Google [dot] com to search for contents which have silberschatz in it. So, this is the basic purpose of the uniform resource locator or URLl incidentally you may have hard the names like URI and URN in addition to URL.

So, they are related, but they mean little bit different things as this venn diagram shows. So, a URI can be either a URL or a URN or it could be both. So, URN functions like a person's name; so, you can conceive it that way universal resource name and URL resembles that of a person's street address. So, URN says what is the name of the content and URL says where that can be found. And URI in general could be either the name or the address or both of them.

In this context of our discussion we will continue to use the term URL only, but I just wanted you to be aware of the other two terms in case you come across them in the text.

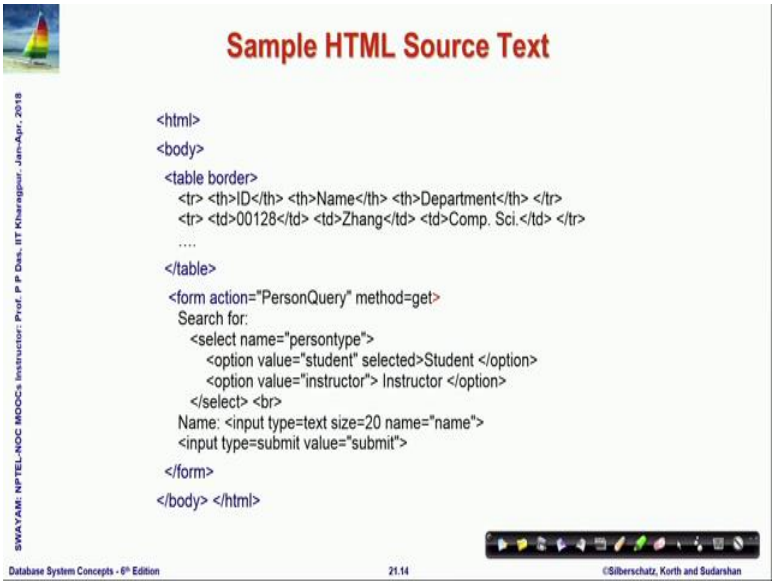
(Refer Slide Time: 12:53)



So, we know by now that http HTML provides the formatting the hyper text links images and so, on and http provides the protocol through which the contents are exchanged between different machines in the internet. So, you can select from a set of options in terms of a HTML popup menus, radio buttons, check boxes and so, on; you can enter values to text box and once a form has been filled up that form will be sent back to the

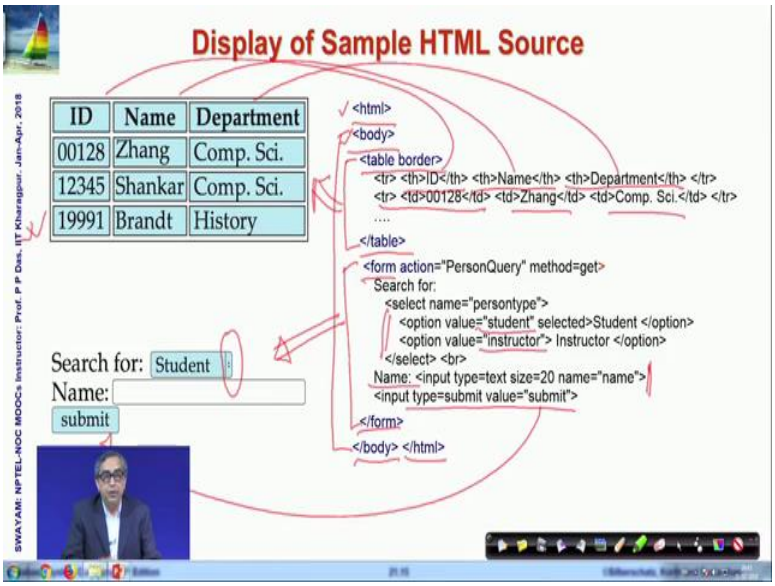
server from where it came and would be acted upon by the server http helps in that transfer mechanism.

(Refer Slide Time: 13:39)



So, the here is a sample HTML code let me show you the effect of this in the next slide.

(Refer Slide Time: 13:46)



So, if you look in here then you can see that these are this is what is known as a tag < > and < / > it kind of the tags are kind of given in the form of **( )** notation.

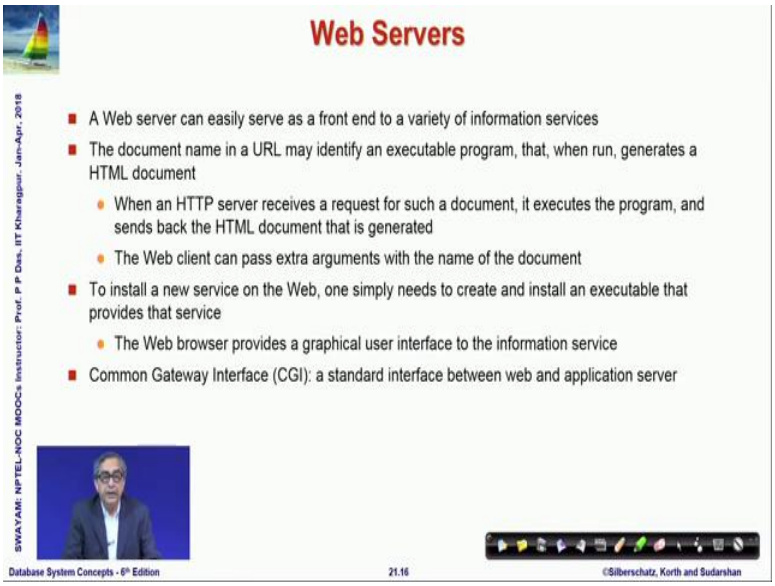
So, it has a opening and a closing and these have to have to actually match and you can see that the opening is written within corner brackets and the closing is write the same tag name, but you put a forward slash before the name. And then between the closing opening and the closing you can write more tabs in a nested manner.

So, here it says that I have HTML which has a body and the body expanse this much and then we are saying that there is a table. So, this is the table that you can get to see and then it also says that there is a form and this is the form that you get to see within the table you can see. So, it is saying that this is an ID there is a name; so, it is describing the first row the department. So, you can see each one of them the ID is here the name is here the department is here similarly this is a next row where it is saying it is 0 1; 00128 Zhang Computer Science.

So, this is how you can you actually in an HTML in a text form all these details will be given and when it is rendered by the web browser then you will see a table like this. Similarly here we are I am showing a an instance of a form which is use to input data. So, we are saying that here is a drop down and it is written out here in terms of options.

So, the first options student is visible here if you drop down you will actually see another option instructor here you will not see that because it is a frozen image. So, and then I have a qualifier name and there is a input text box where you can input any strain up to size 20. And once this has been done then you have an input which is submit, which is the submit button here which shows that you can now submit and then this form filled up form will be sent back to the web browser from where it originally came.

(Refer Slide Time: 16:23)

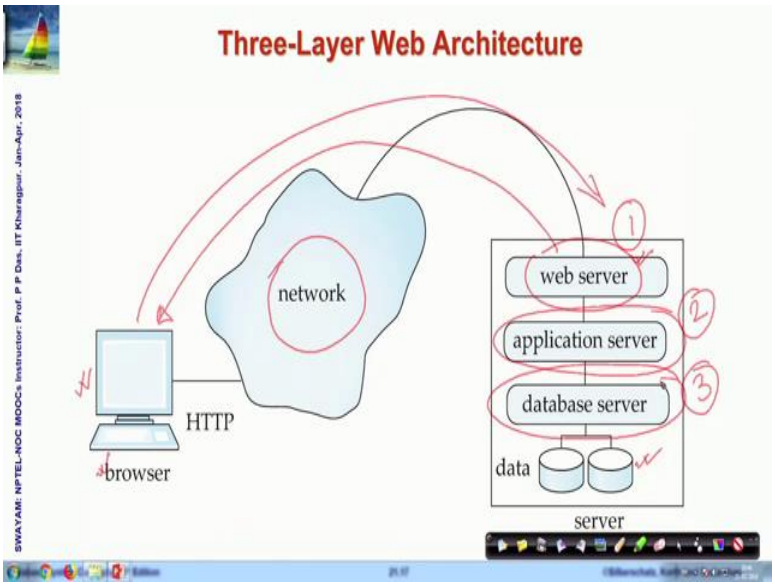


So, this is the basic mechanism of HTML you can learn little bit more about that and get familiar with it. So, a document name in a URL may identify a program that is written that generate. So, HTML could be either at the URL in the web server you can either have a HTML which we say is a static HTML or you could actually have a program which when you send the request it actually.

For example, when you are doing Google you said **www.google.com/search?q= silberschatz**. Then actually at that location there is no HTML currently existing which contains the search result, but instead there is a program which will be executed based on the submission of this form and when run that will generate a HTML document. And once that is generated then this will be passed back to the to your web browser. So, that is a basic mechanism.

So, if you want a new service on the web then you all that you simply need to do is to create and install a new program that will provide that service and through this process we will see how easily this can be done and how web browser provides a graphical interface to this information service. There is there has been another other mechanisms of doing similar things also which was particularly popularly are called the common gateway interface or CGI, but now we have various other ways of doing the same thing.

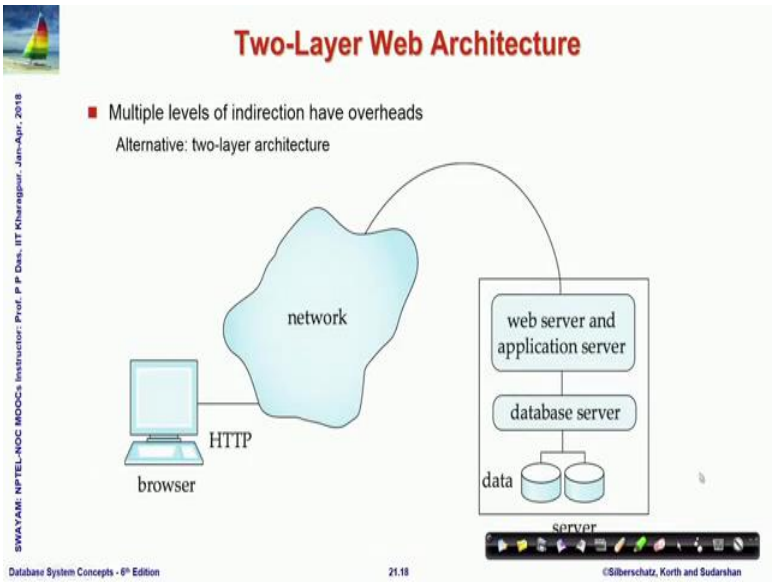
(Refer Slide Time: 18:03)



So, in this context then the basic three layers that we started discussing with are here. So, this is the most common way an application is. So, this is where your frontend is where you have the browser where you see that this is the network; we are just in general writing network it could be internet it could be intranet and a web server. So, when you send a request this is received by the server; web server somehow computes a result HTML and that send back to the browser and that is how the interaction keeps on happening.

So, the browser and the web server together often would be refer to as a frontend because that gives a presentation that presents the results the interaction to you. The next is the application layer which is the business logic where you write and then you have the data access layer or the database server and these are the actual disk where the data exist. So, this is a tier 1 this is tier 2 and this is tier 3 which is a very typical way a web application will be architected and these are the three layers or three tiers that we will usually find.

(Refer Slide Time: 19:22)

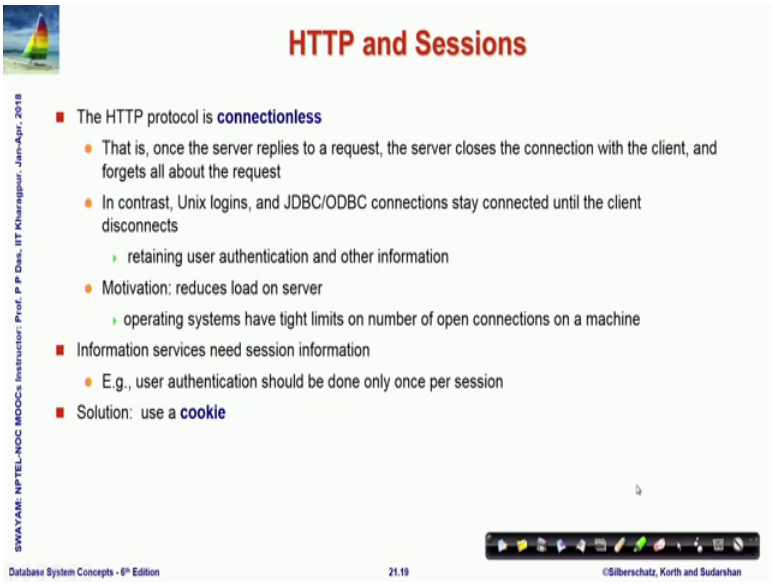


So, often actually three is not a very magical number in terms of tiers; it is possible that you could have more some applications have more tiers. And some applications may choose to have multiple functionality in the same layer for example, web server and application server functionality could be clubbed together and when this is done we say that we will then we have only two layers. So, the frontend and the middle layer are merge together and the backend or the database server becomes a second layer.

So, we would often might want to do that the reason we do that I will just take to back to the three layer view. So, if the question is naturally if we have the web server and we have these connections this is clear; now the question is what is this connection and what is this connection? Is it necessary that they will have to be on the same server physically or will they be on can be on different server and servers could be connected through a LAN or they themselves could be on different servers over the internet and may they may be connected through internet.

So, all of these are possibilities and the way we connect is the way we will write the application will be will not depend on the way these servers are connected between each other we will often assume that as if they are connected over a net and write it in a way so, that even when they may be connected over a LAN or even when they may actually be on the same machine things will work in the same way.

(Refer Slide Time: 20:57)



Now, one point that should be born in mind in terms of the http protocol is it is connectionless. So, this is a very very critical concept and it means that once I start the process I have an URL; I submit that and that goes to the server the server runs an application or it is a static page server picks up and returns me that HTML; the http loop is closed.

So, that is the all that happens and when I submit again something based on that. So, I have gone to Gmail **mail.google.com** I submit that and I get back a form which tells me to put my user ID and login I do that I submit and that when that goes then there is no memory about the earlier interaction that has happen.

So, when my login submission goes it is authenticated in the backend I am able to login and I am given back the first screen of my mail box which is the inbox screen. And as soon as I get that inbox the HTML containing the inbox on my browser that transaction has also been over. So, if I now want to look at a specific mail it has to be a new query and it is not remember anything from the previous query.

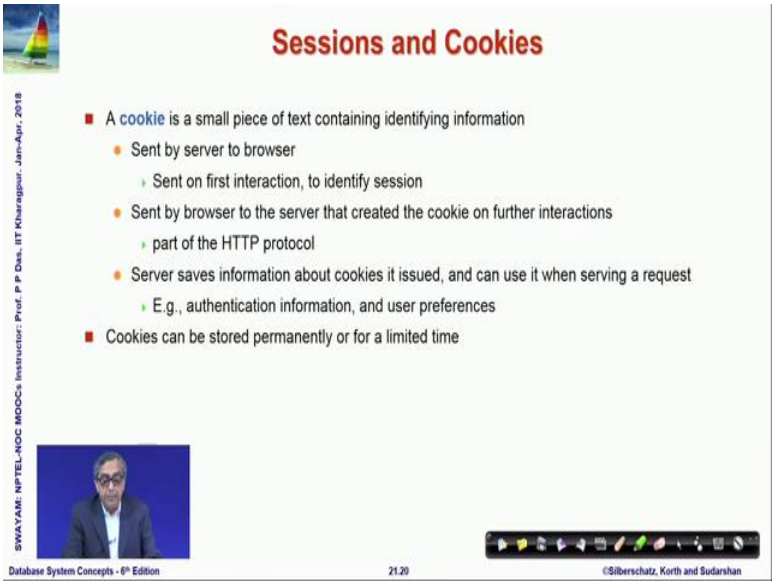
So, this connectionless property naturally makes it makes certain things more difficult; you will you will realize that many of the other connections that we do for example, if we login to UNIX system or to a window system if we use some database connections they are connected till the we disconnect them, but in http it is not. So, it is connectionless every time you do you have a separate session. So, naturally the question

this was I mean the there are there are reasons of why this is done this way this is to reduce load from the server and so, on.

But naturally the consequences therefore, we cannot remember information from one request response loop to the next. So, if I have logged in to my mail Gmail account and I have seen the I have got the inbox then when I want to check my first mail the system does not know any more that I am logged in because that session request response has is over and now I am making a new request that show me the first mail and I expect to see the whole body.

So, there is no information that is carried from one request to the other which makes http difficult to work with. So, the solution for that is something which is known as a cookie.

(Refer Slide Time: 23:46)

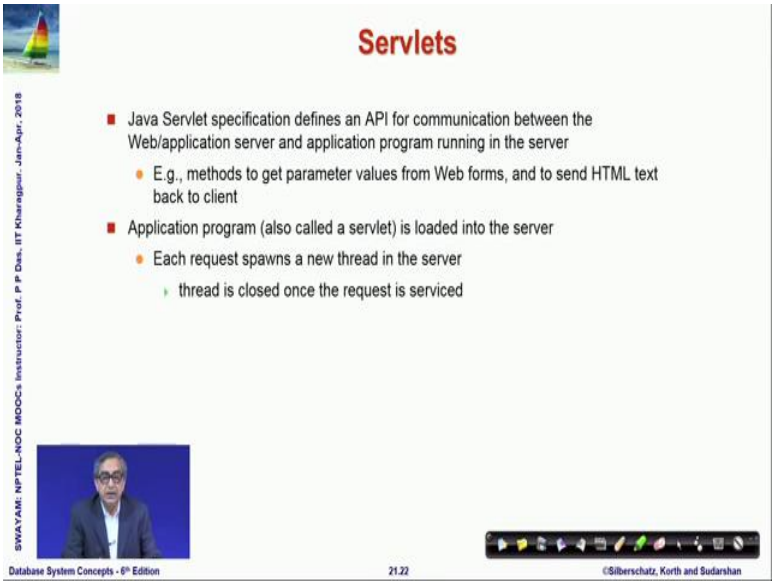


So, a cookie is a small piece of text which contain information which is identifying and which can go back and forth between the browser and the server. So, first it is sent by the sever to the browser and the what the browser does.

So, this happens the first time. So, when we logged in my browser has got a got some cookie from the mail Gmail server. Then the browser can send it back to the server when it is doing the next request so, that I can be identified as a logged in person and. So, the browser can keep it as a I mean locally in its memory or locally here and that is a part of the http protocol.

So, this keeps on this cookie keeps on going back and forth back and forth. So, every time I send a request the cookie actually has to go to tell the server that yes this is the Partha Pratim Das who is already logged in an authenticated himself for checking his mails. So, cookies are a big convenience and they are very important factor of the web applications. So, they can be stored permanently or for a limited period of time.

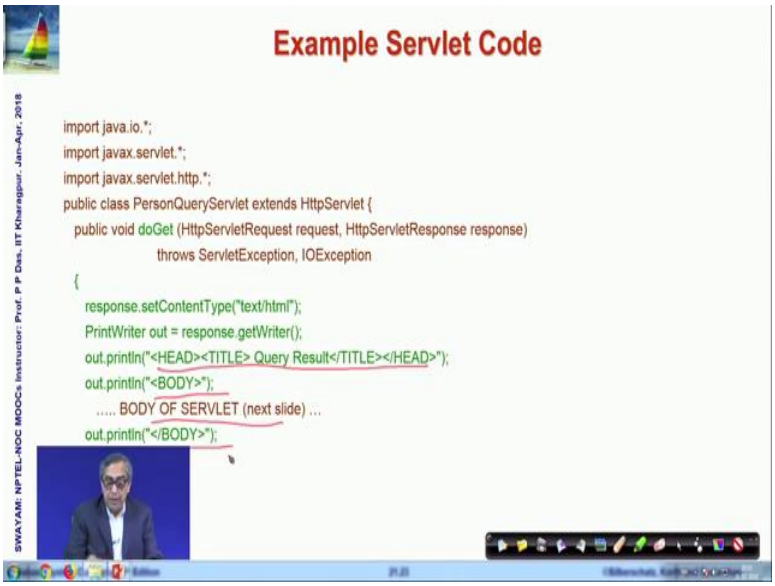
(Refer Slide Time: 25:04)



Next let us look into some of the core technologies that are involved the first technology is called a servlet. Servlet is nothing but as you can understand from the name itself is as you have book booklet booklet is a small very small book servlet is a very small server.

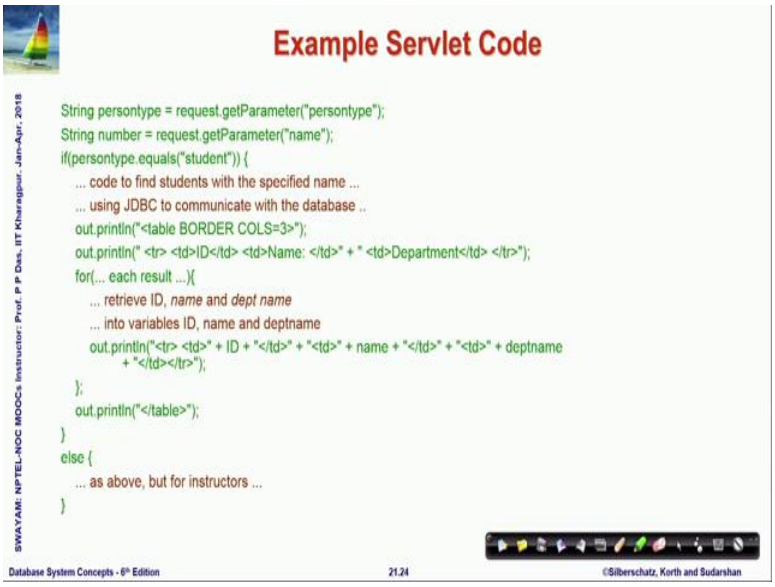
So, it is a Java application which can do certain tasks; so, it is an kind of an application program and every time we request then the server actually spawns a new thread and in that thread this servlet would be running and once the request is serviced the thread will be closed.

(Refer Slide Time: 25:42)



So, this is the typical server servlet view. So, which shows that in the servlet you are creating actually creating the requested I mean possible HTML response that you would like to have.

(Refer Slide Time: 25:58)

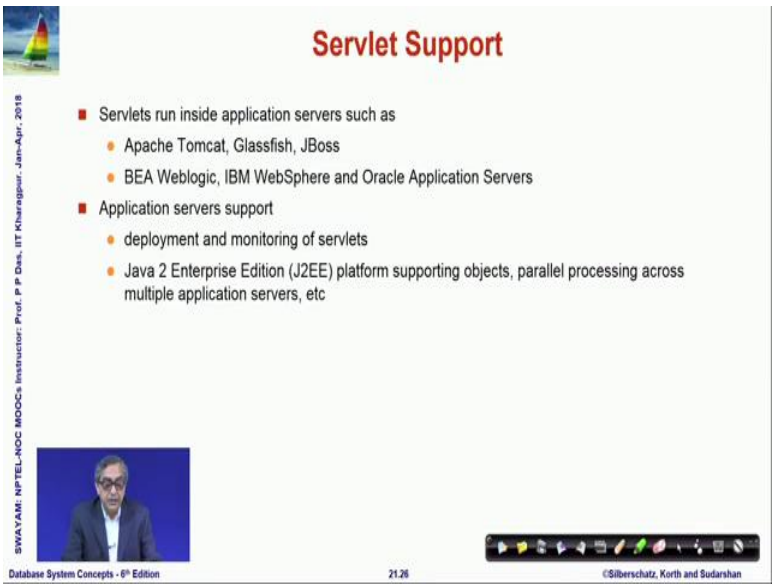


So, there are; so, there are different details you can read through that. So, this is the typical servlet code. So, it actually is a Java code which through print line will generate different lines of the HTML. Now naturally servlets maintain session the way we talked about.

(Refer Slide Time: 26:11)

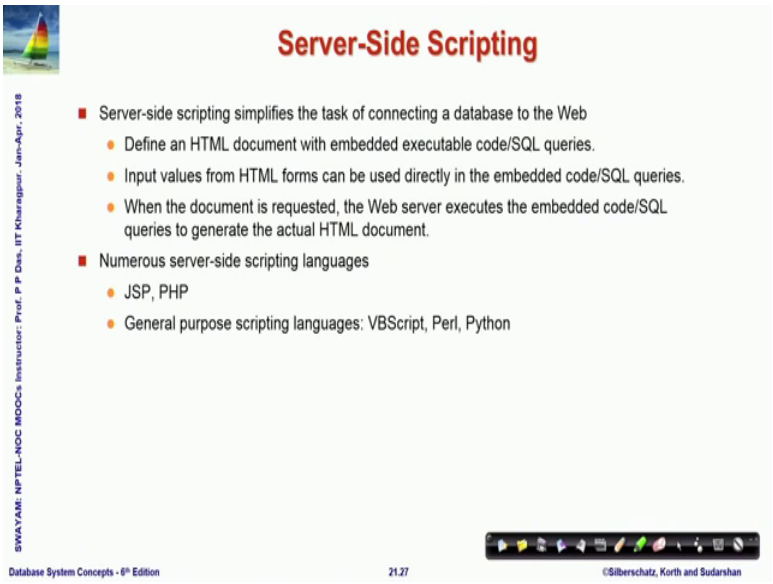
So, that through an interaction I can continued to be identified and the servlet can check whether the session is on or the session is already over. So, these are these are the different ways of doing that in terms of shaking the user ID and several web servers application servers have support for servlet apache tomcat is one of the very popular one.

(Refer Slide Time: 26:33)



So, which you must have heard the name of and there are, but there are several other servers as well.

(Refer Slide Time: 26:48)

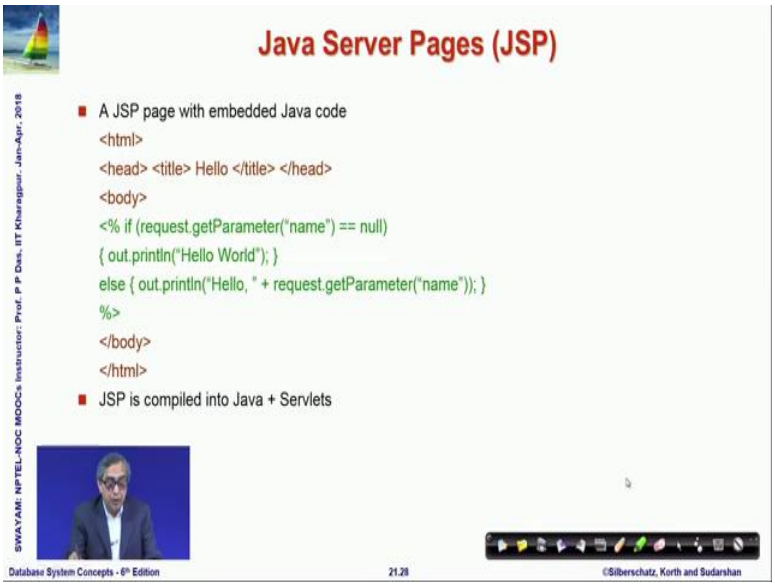


Now, along with the servlet there is another concept which is called server side scripting. server side scripting is a mechanism where you define an HTML document and to within that HTML document can be used. So, you may have some inputs to that and they can be used to directly fire embedded SQL queries.

So, we talked about a madding of SQL query in while we discuss about the basic mechanism of host language and query language. So, here the HTML kind of a language is a host and you can embed the query right in as a part of that. And so, that query goes to the database query server and you get the answer and that answer is placed where your original query was there. So, that you continue to get very easily a my complete HTML as a response.

So, this kind of a mechanism is makes it very easy because a it is quite easy to conceive of the HTML and fill in. So, if I have asked for say logged in to the my mail Gmail service then I have given the input as my user name password and when that got gets authenticated. Then I get a response which is select mail from different respective tables where my authentication is there the user name is PPD and so, on. So, it becomes quite easy to actually create the HTML and there several such scripting language is JSP and PHP are the most popular ones.

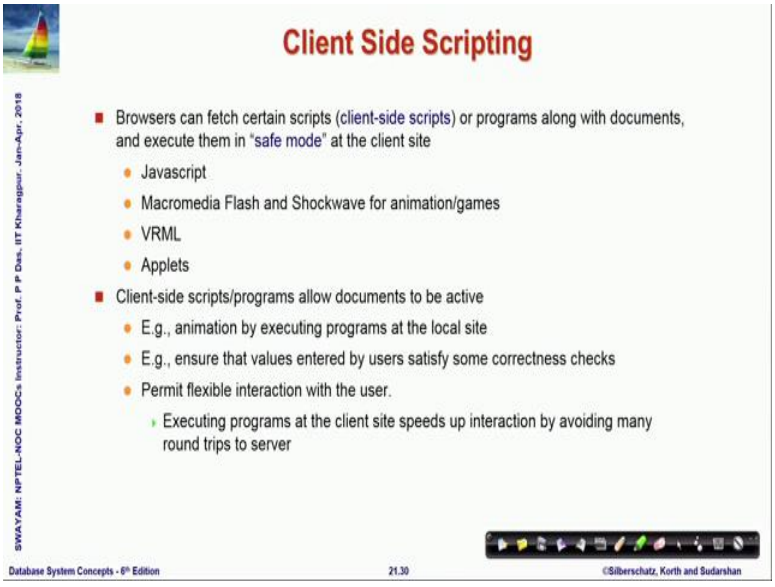
(Refer Slide Time: 28:35)



So, this is how a typical JSP will look like. So, you can see that this actually looks like HTML, but inside that you have a, you have some part of a Java code. So, what will happen the body will get replaced when the when the response has come for example, here the response is doing hello world.

So, when this is executed then whatever is a result will replace the body in the HTML here and the result in the HTML will get generated. There is another mechanism of scripting which is also very popular called PHP. So, this is how it is done.

(Refer Slide Time: 29:19)

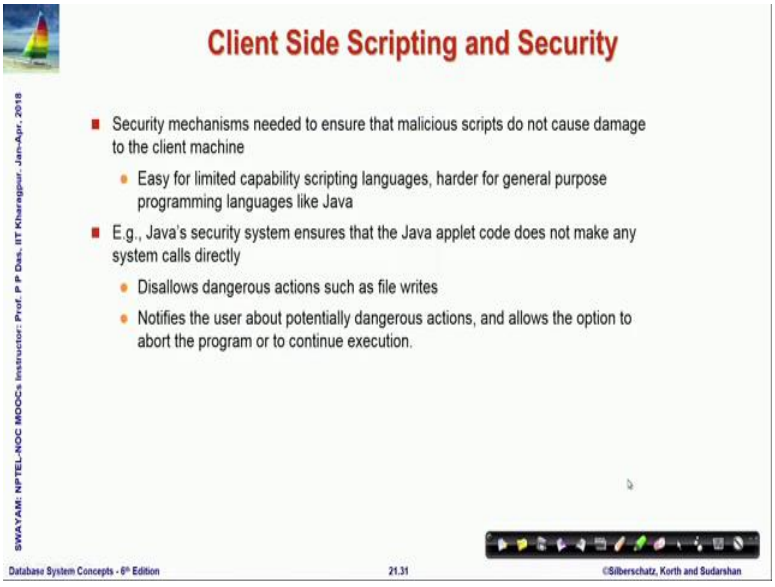


Similarly, you could have script an client side also for example, if you are entering data for say a month and in numeric and you happened to enter 14; then in most cases the page will immediately give a error saying that 14 cannot be a valid month; so, there is a validation involved.

So, in the client side in the browser there are some small script that can run a most typically it is a Java script which can too different authentication which is possible without actually accessing the data in the database. You cannot for example, validate a mail data based on the client side scripting sitting on the browser, but you can validate small things like valid data forms, range of data and so, on.

And it is it is very important because if you could not do that then all you required is you would have send that faulty month numbered 14 to the to the backend server and got an error and you would have come back and then have to correct it, but you can do this locally at the browser itself.

(Refer Slide Time: 30:26)

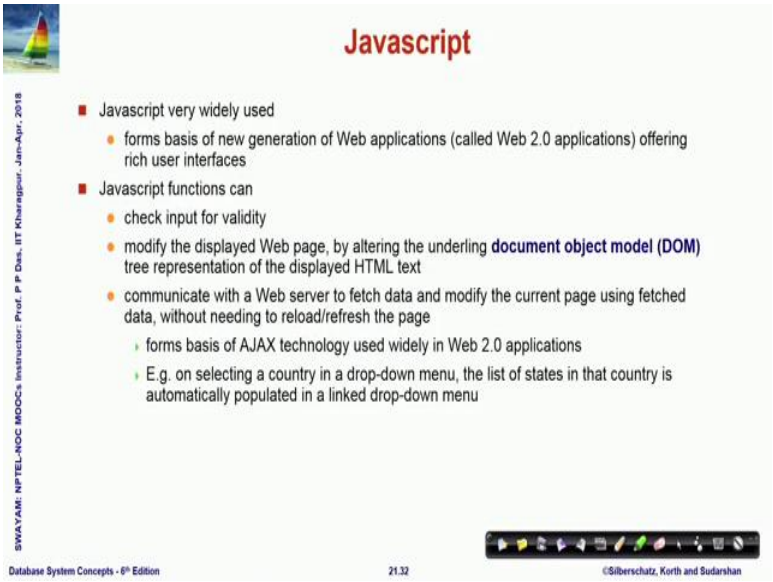


So, client side scripting has a lot of value, but you will have to have to remember that a if you are doing client side scripting then there are security issues.

Because it is quite possible that if you are doing things on the client side that is on the browser then we might also inadvertently or by a malicious intact actually make damages to the machine on which the browser is running. So, there are different kinds of care that

is to be taken for example, Java applet which is another way of doing client side computation disallows file writes and so, on.

(Refer Slide Time: 31:03)

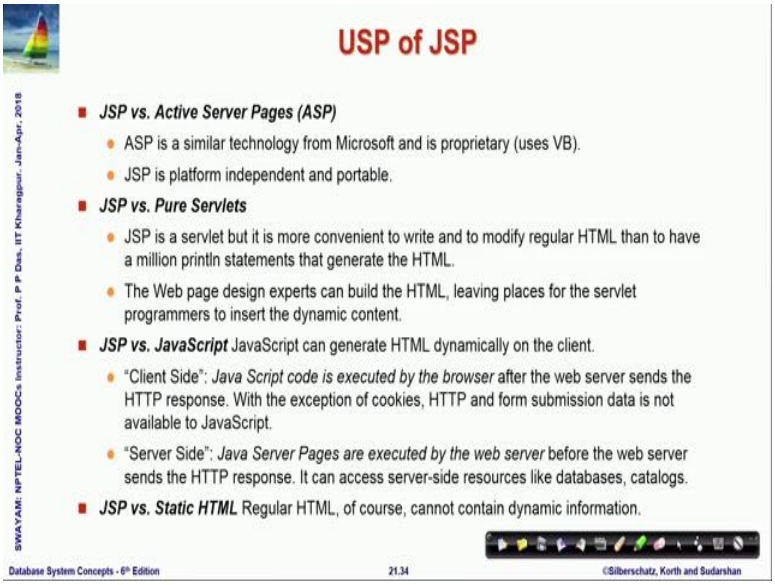


And Java script as I have said is widely used and it can function to check for input validity which I gave an example of it can modify the displayed web page, it can communicate with the web server to fetch data and so, on. And it is you should familiarize yourself with Java script more; it is a very powerful mechanism to do compute the sample things at the client side this is an example that I have given.

(Refer Slide Time: 31:20)

So, you can read through and you will be able to if you know Java you will be able to understand Java script very easily, you could get through that.

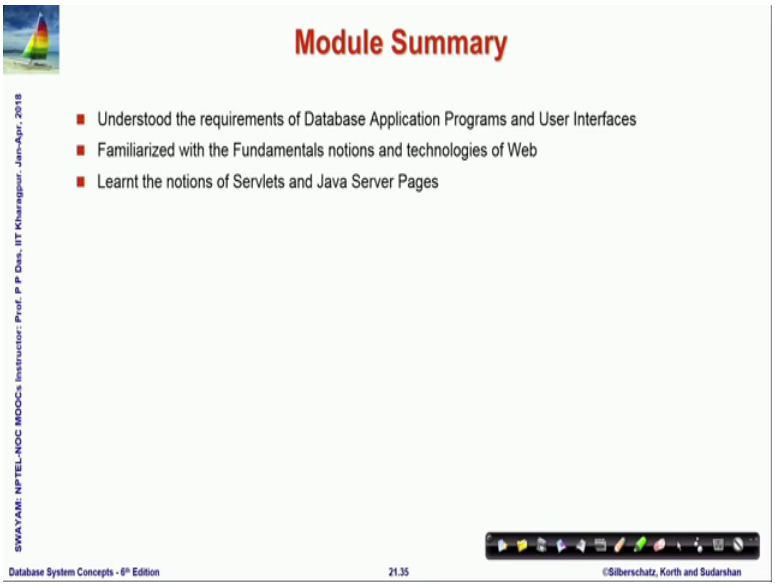
(Refer Slide Time: 31:33)



And so, there are multiple options of doing such things, but JSP has a has a unique position because it is it is very useful in many context wise a JSP has certain USP like active server page which is used in terms of the Microsoft platform is also another mechanism of doing server side scripting, but JSP is better because it is portable in with respect to pure servlets which we showed you in the beginning JSP performs is easier to use.

Because JSP has the structure of the HTML page whereas, in a pure servlet we will have to use print line to print every tag of the HTML which is cumbersome. JSP in contrast with Java script is certainly a different thing because Java script runs on the browser on the client side, JSP runs on the server side and certainly JSP is compared to static HTML is more powerful because it can handle dynamic information.

(Refer Slide Time: 32:39)



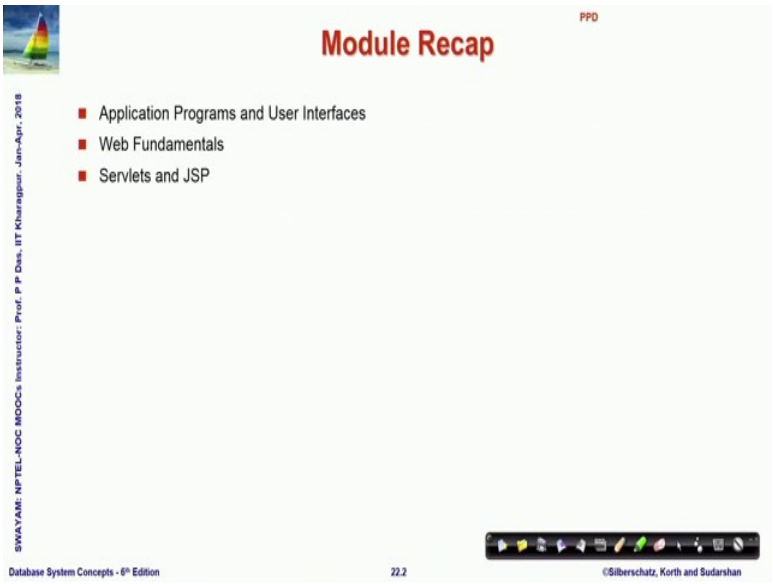
So, in this that brings us to the end of this current modules; so, what we have done we have understood the basic requirements of database application programs and user interfaces understood the basic terminology of the web and took a look into the core notion, core technologies of application development which is in terms of the servlets and Java server pages.

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

Lecture – 22
Application Design and Development (Contd.)

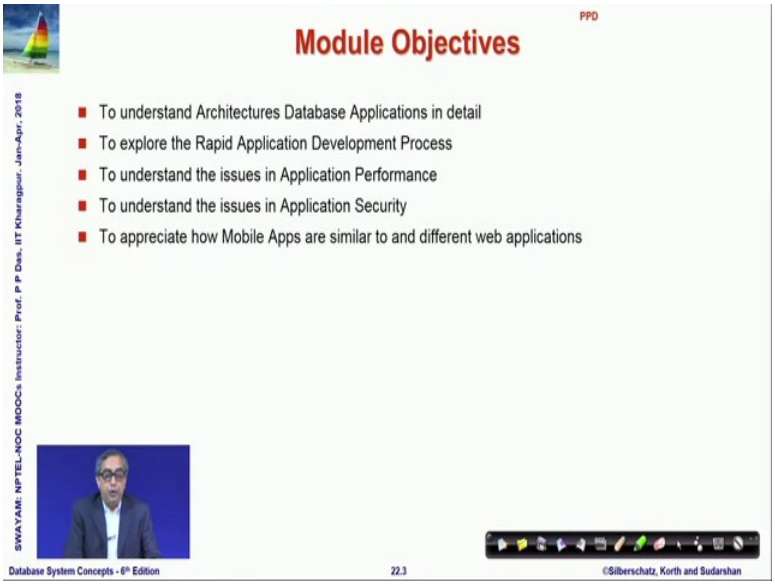
Welcome to module 22 of database management systems, we have been discussing application, design and development this is the second part of that discussion.

(Refer Slide Time: 00:27)



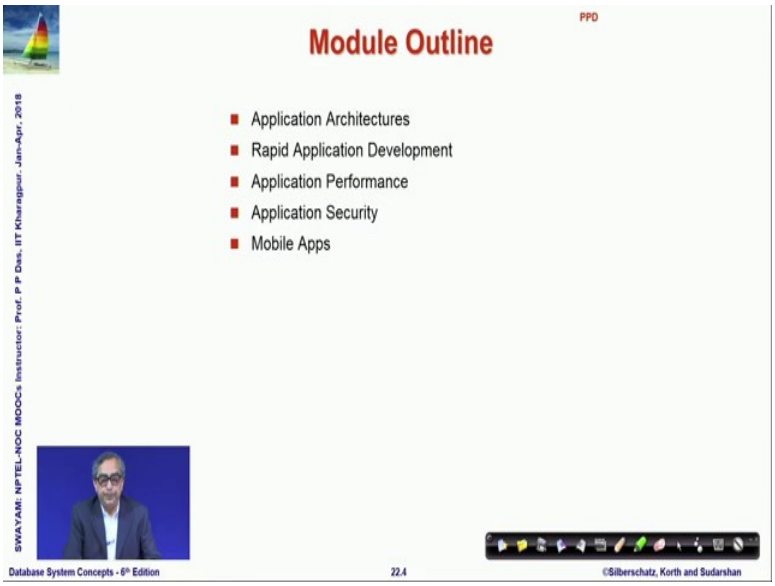
In the last module, we have taken a quick look at the application programs and the user interfaces, looked at the fundamental notions of web and specifically the servlets and JSP.

(Refer Slide Time: 00:38)



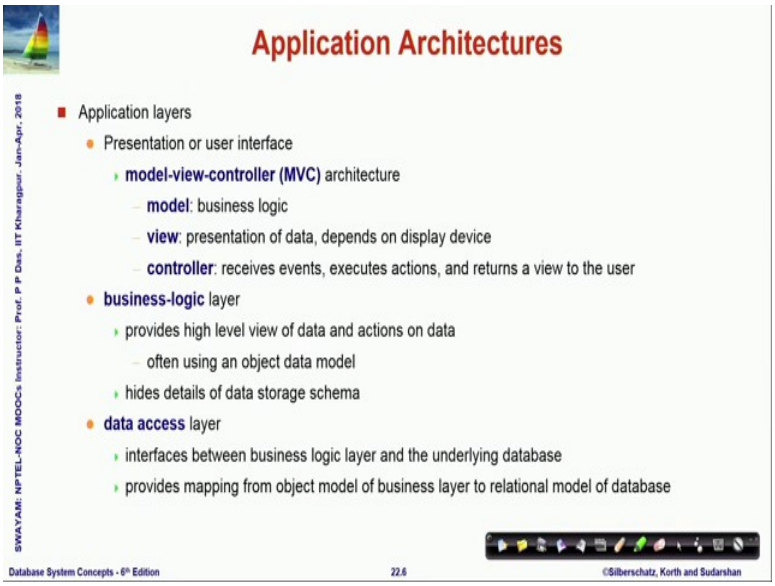
In the current module, we would like to understand the 3 tier architecture in little bit more detail, and explore quickly take a look into the rapid application development processes what kind of help is available, for quickly develop applications and then, we briefly we will look into the issues in terms of an applications performance and it is required security, and at the end we will discuss how a mobile app is similar to such web based database applications? And how they are different?

(Refer Slide Time: 01:14)



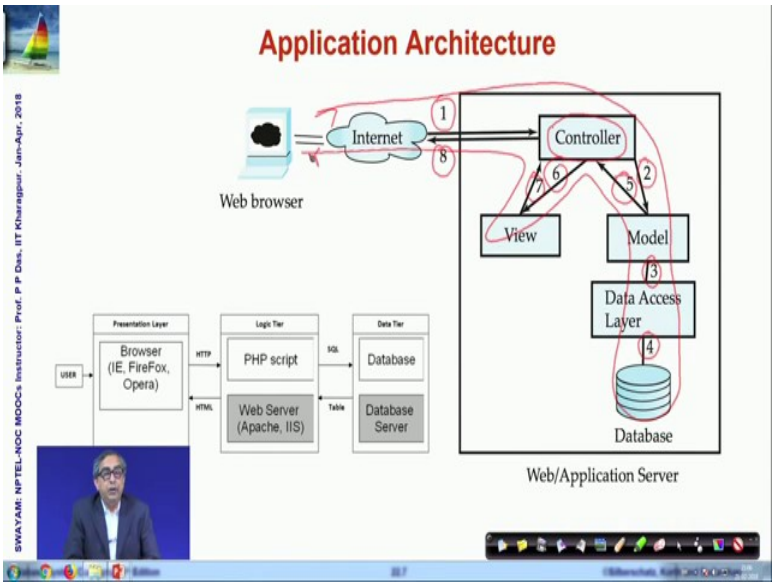
So, this is the outline the 5 parts.

(Refer Slide Time: 01:19)



So, in terms of the application architecture again the presentation layer, or the user interface, business logic layer, and the data access layer, the frontend, the middle layer and the backend. Now, in the presentation layer or the user interface it is typical that applications follow, what is now known as MVC architecture, model view control architecture where, the model is the the is kind of the business logic that, that is implemented in terms of the frontend information. View is the actual presentation of the data, that HTML and controller is one who, receives different events execute actions and so on and then, we will go into the other layers.

(Refer Slide Time: 02:09)



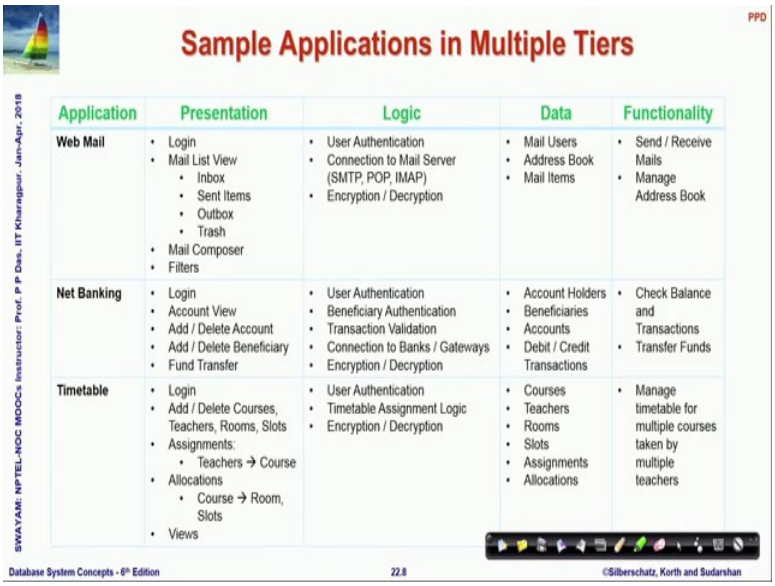
So, here let us try to understand this flow. So, there is a request in the web browser say, to the service. So, this is the sequential now, let us say we are trying to log in to Gmail. So, in one we send a form HTML form which, has the username and the password and possibly encrypted, that comes to the controller ah. So, which basically controls the different events.

So, the controller knows that, it has to now decide whether, this what actions are required in terms of this input data. So, it is sends it to the model which, is the business logic here. So, the business logic model knows now well. So, at this there is an application which, deciphers the business logic which says that, business logic required here is we have a password and we have a user names now, we have to decide whether this user is a valid user and whether, he or she can be allowed to login. So, the model has to check on the user data, the user id and password data and therefore, it has to come from a database. So, it passes on this request to the data access layer, the data access layer in turn access the database.

So, you can think of data access layer is something like a SQL query layer where, you have formed a query select etc., from etc., where, user id is equal to PPD, password is equal to XXX, the database depending on what is found in the database is back to the model, and the model then sends it to the control it is says ok, this is what I have found. So, this is the result of the data that result of the request that has been prepared. So, it is says that well this is have been found and therefore, we have extracted the mails in the inbox that existed, or it is says that the authentication is not possible. So, it plugs in a error message and sends it to the controller, controller now knows that a response has to be framed. So, the controller sends it to the view of the MVC, the view we prepare the HTML that needs to go back. So, view prepares the HTML and sends it back to the controller. So, controller now has the response which it is sends back to the web browser through the internet, and we get to see that well now, my inbox and mails are all here.

So, this is a complete flow of starting from here, going through this, coming back, going here, going back here, is the is the whole route of the request, response that goes over the HTTP in a typical web or application scenario, that is the there is a way this application architecture is expected to work.

(Refer Slide Time: 05:13)



So, here I have created a small table, showing you the different activity is that happens at the presentation logic and data layer of different common applications like, web mail like, Google. So, at the presentation layer you will do things like, log in, mail list, view inbox, sent item, outbox so on, mail composer. So, we can write mails filters of checking at different mails. So, all the all these happens.

So, for example, if you talk about filters they might often happen in the java script itself, that trans within the browser, the logic the business logic will do user authentication, connection to mail server because, a mails have to come from a different server, they are not they may not be setting typed in terms of the of the database itself.

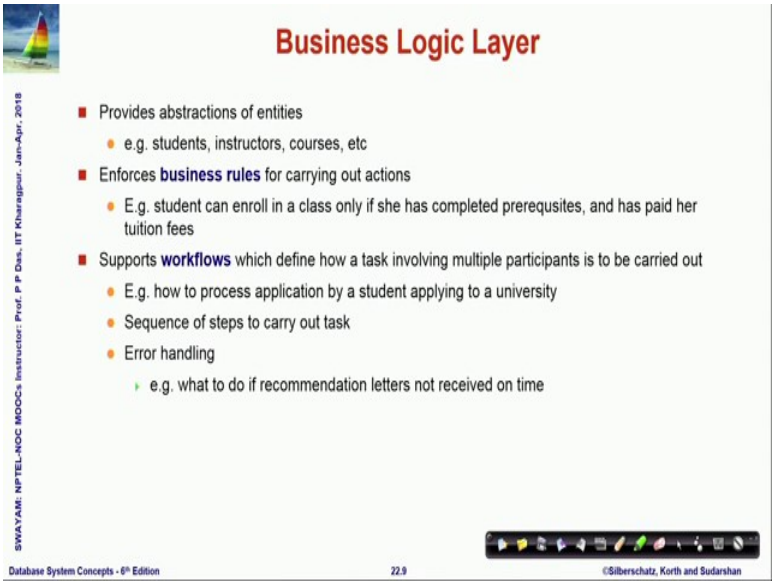
And then user encryption, decryption and the data side you will have different tables to represent the mail users, the users like you and me all who are users of the Gmail, the address book of each for each one of us the mail items and so on, and that will give us the functionality of send, receive mails, managing address book and so on. So, similarly I have listed an application for net banking which can where, we can check balance and do transactions transfer funds, or a timetable where you can manage time table for multiple courses taken by multiple teachers and so on.

So, you can if you think about an application then, you should be able to moderately map it is a functionality across the presentation logic and data layer. So, that you know what you want to do at the clients, and which is which is at the presentation layer, or what you

want at the absolute packet which is on the data, what tables and all the databases that you want to maintain, and the business logic of how actually this application will give you the result, how actually it will?

So, that is something where, you will have a whole lot of complex logic that might come in.

(Refer Slide Time: 07:18)

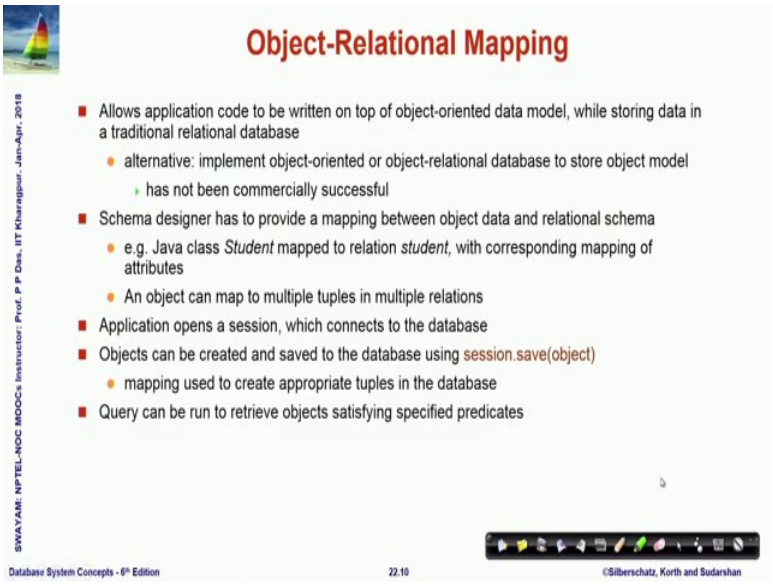


So, coming to the business logic layer specifically, that provides abstraction of that will provide abstraction of various entities, students, instructors, courses, mails, your accounts, balance and so on, and that will enforce business tools for carrying out this. So, a student can enroll in a class only if she has completed prerequisites, you can transfer funds from one account to the other, provided, you have the authority to transfer, provided you have enough funds in the account that is going to get debited and so on. So, those are the different business tools, which will be employed by the business logic layer, and it will support a work flow which defines how a task will be carried out in terms of the multiple participants, and remember that all participants may not actually be human beings, they could be some could be human being some could be other machines or other applications as well.

So, it gives you the work flow. So, you can any of the 3 application that I just mentioned, everywhere you can find that there is a work flow, if you are want to check a mail then, there is a steps that you need to do go to the inbox, chose the particular mail item, get the

body and then if you want to reply to that, select that, the submit that, you get a get a new form when you write the reply, and within that form you get the original copy of the original mail and so on. So, all these kind of work flows will be supported by the business logic layer.

(Refer Slide Time: 08:47)



Now, certainly you can understand that at the frontend, if we if we talk about what are different you know languages and models, that we are working within the frontend naturally our language is HTML tries for presentation, embedded with java script, at the backend in the data it is the database and the SQL query. So, it is a relational model, but what happens in between? What happens in the business layer which connects them? Now, naturally business layer you could write complex business tools.

For example, in the time table application we will have a complex algorithm, I know to find out what allocation of class rooms and slots, are feasible for assignments of teachers to courses availability of rooms and so on. So, often the business logic layer, the this tier would be convenient to write in some typical common high-level language like, C ++ or java, and naturally you would know from your experience of software engineering that, if you have a object based language then, that will be a very convenient to do that.

So, which means that, if you have say some entity as a student in your relational database then, most likely in your business logic, which is a java code you will have a class called student. So, the relation student is in the relational model, and your class student is in the

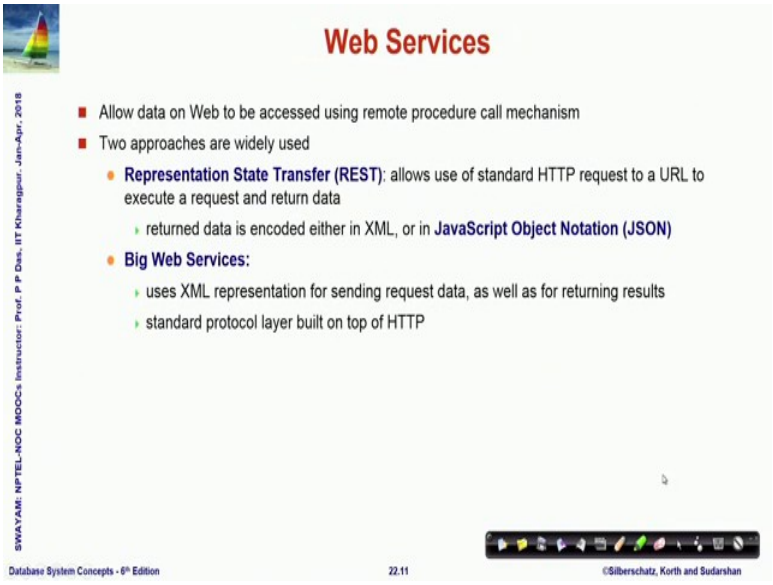
object based model, and you will need to define these in terms of certain mapping of the attributes, which we had shown when we talked about embedding of a languages, and this is what is commonly known as a object relational mapping.

So, that objects can map to multiple tuples, in multiple relations and can be viewed in a different way. So, so this mapping itself we could create a virtual view in the business logic layer, in the business logic language that you are looking at. Of course, they have been attempts to create a models, which are relational models which are also object oriented those are called object relational databases, some of them have been successful, but not really commercially successful.

So, we continue to work with SQL, and the relational database kind of things in the database level, and some kind of a high-level language like, C++, java and the object-based model in the middle tarred in the in the business logic, and continue to do the object relational mapping for solving the problems. So, you can here, I have given the points of what happens? How the objects get created? Application opens a session, which connects to the database because, we need to get the data from the database, they can be objects that can be created safe to the database.

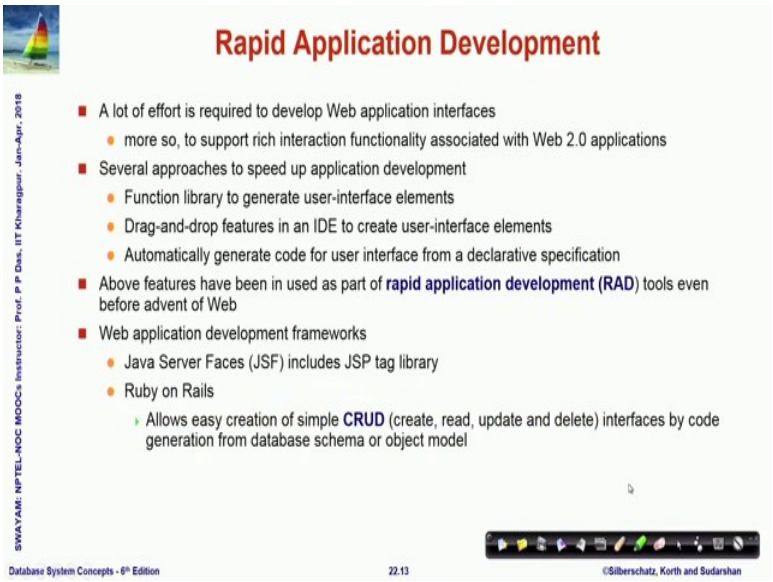
They can be extracted from the database; new objects can be created and mapping use to create a appropriate tuples in the database. So, it is a two-way traffic that will keep on happening where, the business logic layer will continue to see entities as objects whereas, the database layer will continue to see them, as tuple in the relational database.

(Refer Slide Time: 12:07)



So, there are also several web services, that can be used and a you may be getting familiar with that, we will not go deep into this, but I will just mention that, web services a mechanism through which, you can access a data from remote server using what is known as a remote procedure call, and today very common approach for this is called rest representation state transfer, which allow standard HTTP request to a URL to execute a request and return data, and a several of the web services are based on that, and are the are big web services which you must have heard of, but I just mentioned it at this point because it is contextual, but we will not get into those in this course really. Now, coming to how do you actually develop applications?

(Refer Slide Time: 13:05)

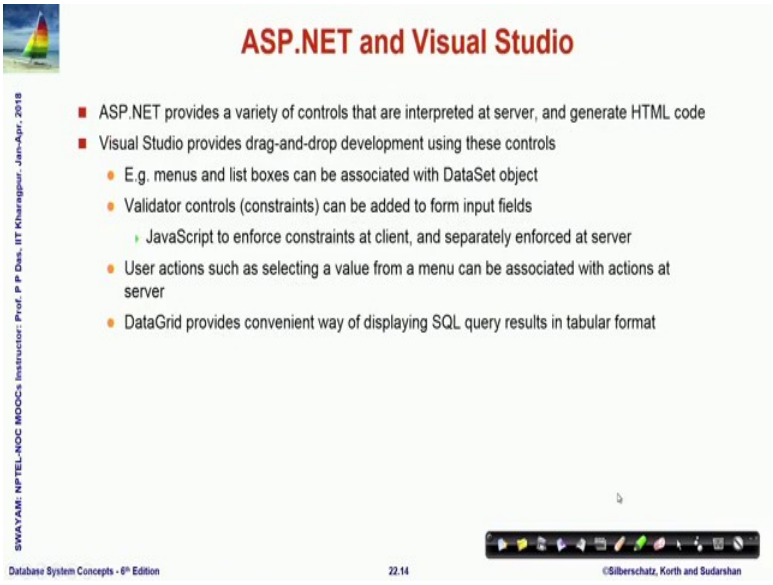


Now, it is not a easy process that is a lot of refer require to develop web applications, you need to support the functionality that of current day web. So, you need several approaches to speed up applications. So, you this is in parallel to if you think of how? What has been done to speed up development processes, development applications for different say C applications, or C ++, or java applications.

So, one approach naturally is to variety of function library, which can help you easily get user interface elements like buttons, checkboxes, radio drag and drop features in the IDE, IDE stands for integrated development environment like, visual studio, front page these kind of which can use a, which can create user interference elements easily you can automatically generate code for user interface, and these are all parts of rapid application development tools, and some frameworks are very popular, this is for primarily the java server faces or JSF is a framework where, you can rapidly develop fill in all the requirements of the different layers, in a web based database application in other very popular is ruby on rails, which allows easy creation of simple crud create, read, update, delete.

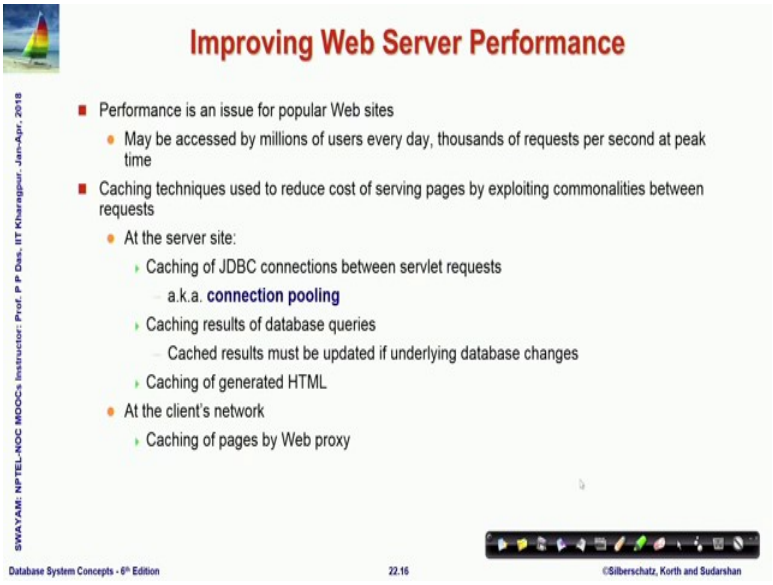
So, if you look at database applications and common applications will all applications will at least need to do this it lead to create data, read data, update data, delete data. So, you can do that quickly with ruby on rails. So, if you are locating into really the development of database applications, get yourself familiar with this rapid application development processes.

(Refer Slide Time: 14:53)



There are different other frameworks as well, another very popular and widely used framework is Microsoft specific, this is unfortunately not portable because, it is proprietary of Microsoft where, you can use the the [dot] net framework of Microsoft, which helps you with lot of an a resources and libraries which are already provided, and like we talked about JSP, we can use ASP[dot] net here active server page in the[dot] net framework, which provides a whole lot of controls and you have a very nice powerful id in terms of visual studio, high were being proprietary these need licenses and you need to pay for that. So, many a times, many developers may not be able to afford it or like it for that reason. Naturally, as you designed applications and create that, you will have to be careful about it is performance because certainly we all want very fast results.

(Refer Slide Time: 15:52)



So, if I log in to my Gmail application, and I would expect that as soon as I press the submit button with a couple of seconds, my inbox will be displayed on my browser, I am not ready to wait for 2 minutes, 3 minutes, 5 minutes for doing that.

So, while developing the application you will have to make an estimate of how often it will be used? How many users will use that every day and so on? how many, what is your expected heat rate? That is, when the maximum number of users are trying to use this then, what is the you know estimated number of request per second? That will come to your server. And to improve performance their different kinds of techniques can be used, the significant of them is called caching, caching is nothing but if you expect to request to be similar ah. So, that they are results should be similar then, after the first request besides sending the response back, you actually keep a local copy of that.

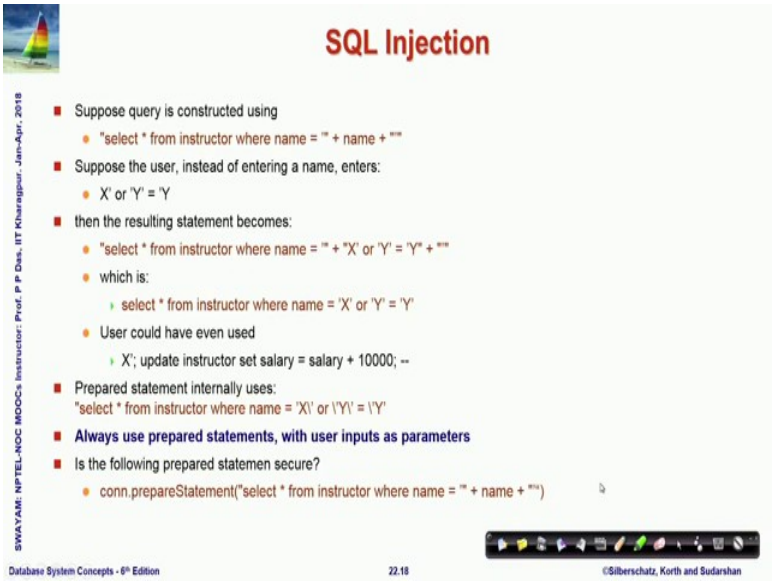
So, that if a similar request come in future, you can you may not re compute it, you can just send that cache copy. So, it can be done in terms of verity of JDBC connections called connection pooling, it can be done in terms of database queries, caching of generated, HTML and so on.

It can be done at the clients network side also by caching pages by web proxy; obviously, if you are using caching to improve performance, you will have to understand lot more of the web dynamics in depth because, certainly if you cache then there is a possibility, that somebody is asking is sending a request for which, the response would not be the same as what it was, when the last time the response was computed and cached. So, if

you send the cached information back then, you may be giving a dated information. So, possibly say for example, if you are checking at the net banking transaction, you have making a fund transfer and checking your account transactions after that one transfer, you would not expect a cached page whereas, if you are looking at the website of a say IIT Kharagpur then, it will be to cache that because, it is not expected to change very frequently. So, these are different factors. So, we do not have it in the scope to going to different issues of, how to improve performance? And what are the different challenges?

But I just want that you to be sensitive about these issues. Other very deep you know concerned, deep requirement about applications are the security of applications, there are this is a very involved topic, and there are several issues that are involved.

(Refer Slide Time: 18:48)

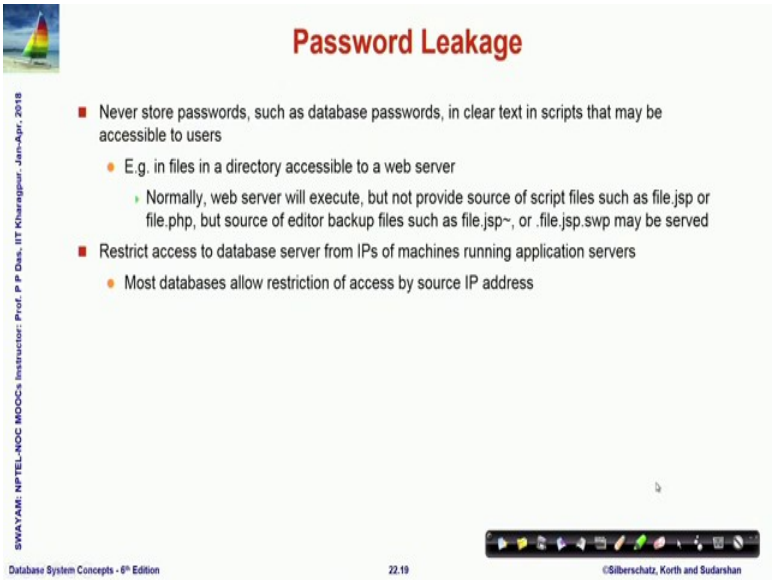


So, and I am talking about only a few of them because, the many of them require knowledge about several other fields particularly, in the field of security and an encryption and so on. So, one that is very common in terms of SQL query is known as a SQL injection where, based on I mean there is if you are expecting some inputs to come ah, and fill up certain parts of the SQL query then, you will have to be careful that, user should not be able to give such input say such strings.

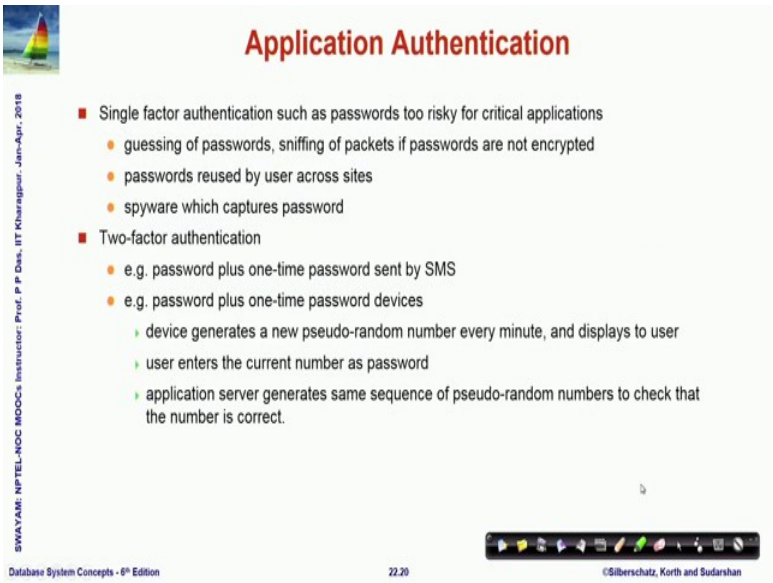
So, that the query actually mean something different, this query may be which was just a query to read something, may update something, or give some different result. So, here I have just briefly highlighted some of those issues, there are other security issues like,

leakage of password if there are important things which are based on a single password then, use the password gets compromised because, it is shared or it is broken in a middle by some hacker and so on then, you will have a lot of risks involved.

(Refer Slide Time: 19:37)



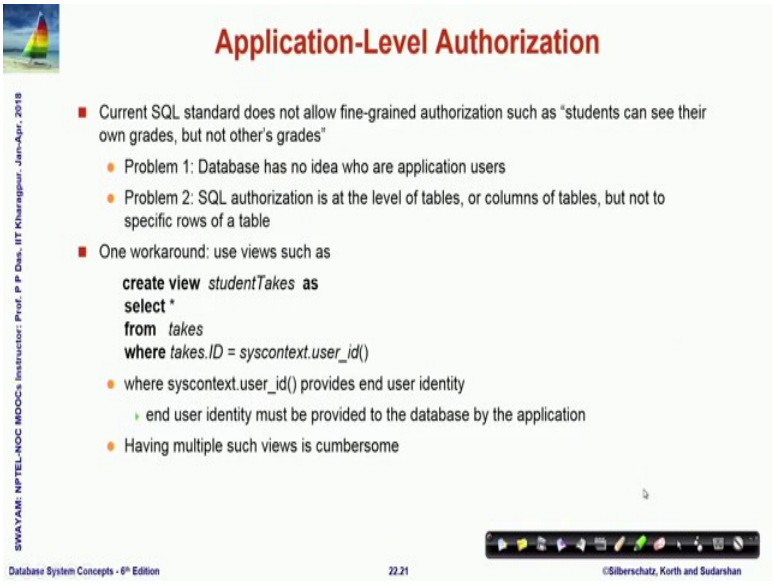
(Refer Slide Time: 19:59)



So, you should be you will need to make sure that, you do not store passwords you just store encrypted forms of that in which encrypted forms, you have must have observed for the last couple of years that, in many cases earlier you could just log in. So, giving just one password was enough, but now in many transactions for example, if you are logging

into a net banking application then, often you will be asked to provide some additional key information, or you will be asked to do a authentication by OTP one-time password and and so on. So, these are common because, you know it is risky to work with just a single authentication, and you will find that some net banking applications for example, if you are doing a fund transfer, then they actually require two additional password authentication, one is a special password for fund transfer, and then possibly we will be asked to go through an OTP. So, depending on the criticality of your application and the potential vulnerability of the application, your authentication mechanism will have to be appropriately designed.

(Refer Slide Time: 21:09)

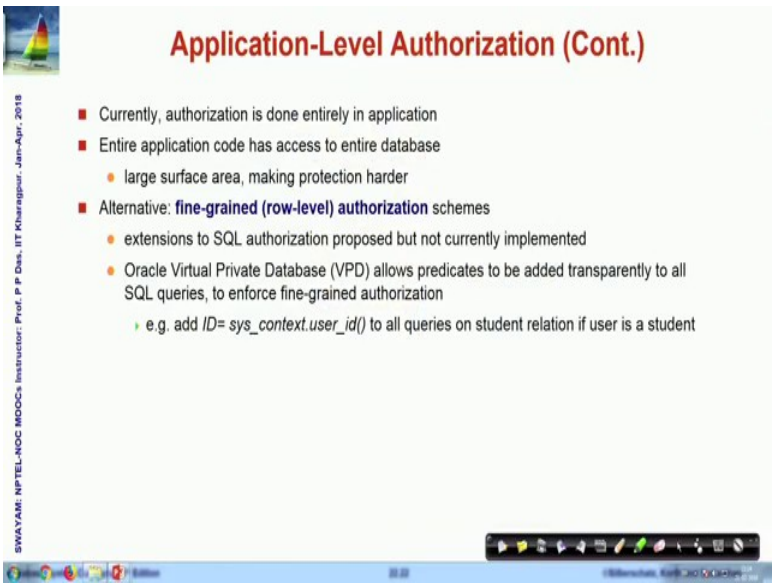


There are issues in terms of security, in terms of the application level also, for example, if you if you are looking at a at a student application where, you want to say that the student, every student can see his or her own grade, but they should not be able to see the grade of others. Now, how do you implement this? Now, two issues are one is the database cannot implement, this database cannot provide this as a part of access control because, database has no idea about who the application users are, and the second if you recall the way, we talked about authorization in SQL, that is in terms of tables or columns of the tables, but SQL has no mechanism to create authorization for rows of the table.

So, this will have to be handled, in terms of what is known as at the application layer. So, this is where you are. So, what this is saying that you are created a view where, you are created a view student text where, to do this you will have to read you are saying where, text [dot] id is equal to this function called is sys context [dot] user id. So, sys context is an object naturally, dot user I d this function called. So, what this function gives you call to the function gives you is an information about the end user identity, and that identity has to match, the identity that exist in the text ID for the result to be computed. So, this will ensure that depending on who is actually the current user, the same view will be evaluated for different results.

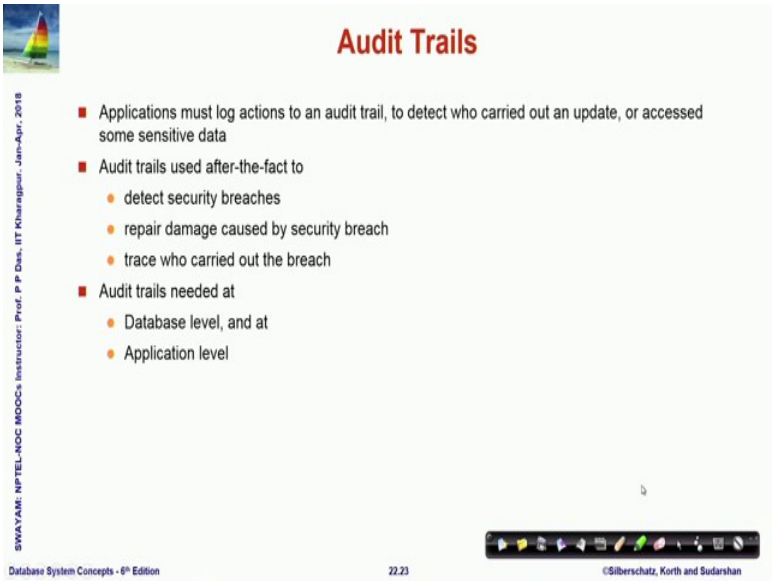
So, this is no not a not a very sound this is not a very comfortable situation because, here the authorization is not being implemented in the database level, but is being implemented at the application level, but that is way things a because, most of the fine grained authorization currently, is done entirely in the application level only a extension to SQL authorization, at for similar you know fine graining and so on has been proposed, but they have not been implemented because, there are several issues of implementing where there several issues of how do you represent? How do you module? And most importantly, these have potentials of slowing down the database, query process and significantly.

(Refer Slide Time: 23:08)



So, often this is not a preferred mechanism either. Another way to ensure security is to keep audit trail, trail must log actions into it.

(Refer Slide Time: 23:46)

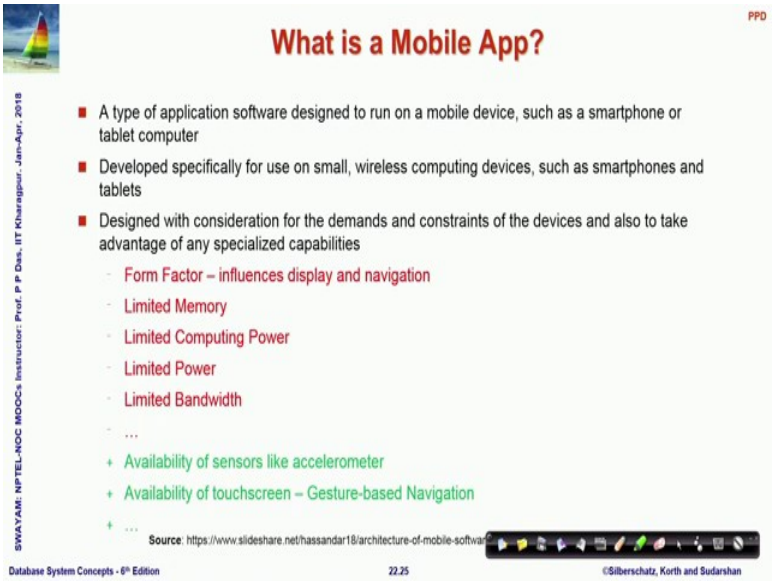


So, where you write down actions in terms of who carried out and update, or who access some sensitive data, or who did a delete and so on. So, audit trails can be used later on if the need arises to detect, if some security breach that happen, or if some damage has been caused by the security breach, that can be corrected and so on create a trace.

So, auditing is necessarily a very required, in a very required activity for any data-based application and audit trail had a good mechanism for that. So, when you develop applications you have to consider has to whether, you would like to support audit trail of course, if you support audit trail then, somewhat your application will get slowed down, it will require more disk to keep that trail because, every transaction every details you will get logged in to the audit trail, but it is very, very important for critical applications like, net banking where currently it is mandated every transaction that, you do every action that you do on your account, through the net banking is trailed in the banks end.

So, that if later there is a there is some dispute, there is some breach has found then, the same can be recovered and trace back to the actions, that you are actually taken ah. At the end of this module let me take a quick look into the mobile apps.

(Refer Slide Time: 25:32)

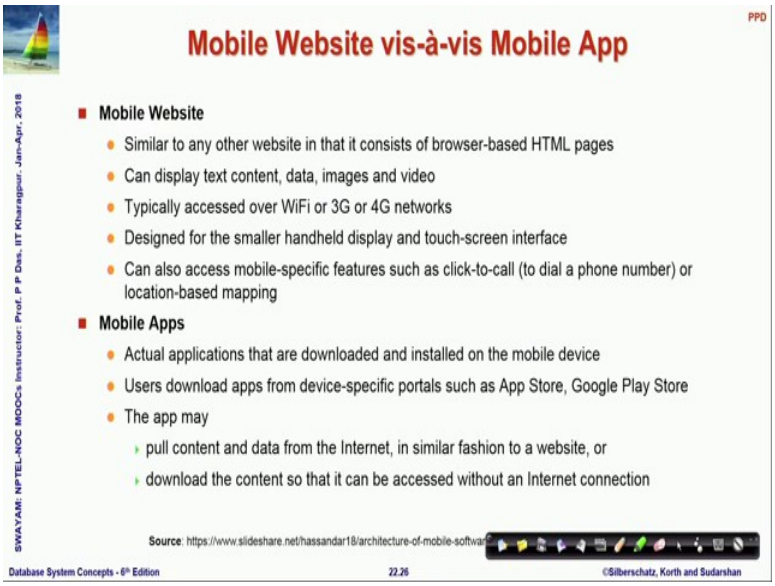


These are somewhat different, but it is very important to today to take a notion of the mobile app. So, it is a type of a application software, which is designed to run on a mobile devise. So, it this is not necessarily mean that, it has to be a smart phone, it could be a tablet or you know some small device, which typically is handled and carried around. So, it is typically specifically for use on small wireless computing devices, and it is designed with considerations for demands and coincident of the device. So, how are how is a different? If I want to use the application or want to have an application which does my task, but I want to do it as a mobile app, I want to do it for a small wireless handled device. Certainly, there are if we if we since the small device it has lot of constraints, there are lot of demands of the device, compare to if we were using a web browser in a desktop or a laptop where, you have lot of resources.

And also on the this is this is on the one kind of the restriction side and on the other side, these devices have certain capabilities, which your desktop or laptop may not have and therefore, it may be you may be able to do more interesting application using this mobile devices, and there could be really interesting mobile apps, and as you all must be using today use 10s of if not 100s of mobile apps for different applications, many of which actually support a database at the backend. So, these are in red are some of the negatives, which are restrictive in terms of a mobile application the first thing is form factor, which is the look the aspect ratio the size and the style of the device, which influences display and influences the way you navigate. In a desk of application you will typically use a mouse, or a roller and keyboard to navigate, but that is not possible or that is not easy in terms of a mobile device. So, you are navigation may happen in in a different way.

The presentation itself may happen in a different way, you have a very limited display area. So, you might want to stack multiple responses one after the other whereas, the in a in a web application running on a desktop, you would probably have shown them on different tabs side by side, or would have just shown them side by side on the display. Then, mobile devices typically have limited memory, they have limited computing power, very importantly most of them run on battery and therefore, they have one limited power available. So, you are applications will lead to be power optimized, which is not the case with the normal web based application, you have a wireless connection, so which may be limited in bandwidth based on your connectivity and that time, and so on. So, you while developing a mobile app corresponding to a possible web application, your considerations would be very different. So, if I say that a bank say, HDFC bank has a net banking application, which runs on the browser. And it is also having a mobile app, which runs on say the android phone then, the their requirements and their style of solutions will have to be very, very different. And at the same time if I talk about a mobile app then, the mobile devices often have features which desktop do not have for example, you have a number of senses available like, accelerometer and so on which you can make to advantage for example, we I mean in in many smart phones, if we just rotate the screen, rotate that device, then the display self-rotate automatically, which we use we use some some kind of an accelerometer inputs for that, you have a touch screen which can allow a wide range of gesture based navigation, which is typically not possible in desktop and most of the laptops.

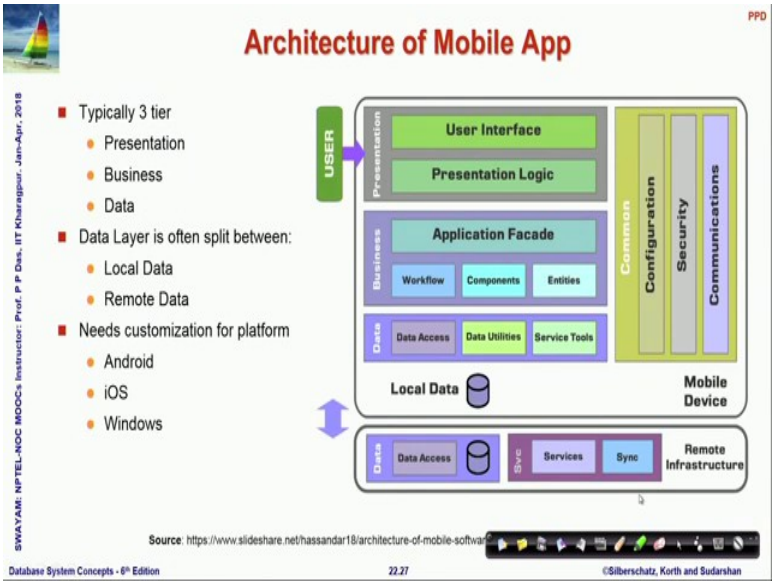
(Refer Slide Time: 29:48)



So, having said that, naturally there are there are 2 aspects of mobile apps as well. So, when we talking about mobile apps, we have talking about stand alone mobile apps, there are other ways of developing applications also for example, we can do a typical web-based application, but we can use a website which is specifically designed catering to the mobile devices. So, these send back pages, which are smaller in size are stack differently and so on. So, in contrast to that a mobile app are actually, applications that are downloaded and runs on the system.

So, these are all different possibilities and even though mobile website is also becoming popular, but certainly mobile apps are very, very common in terms of a majority of critical applications of data bases that we have.

(Refer Slide Time: 30:40)



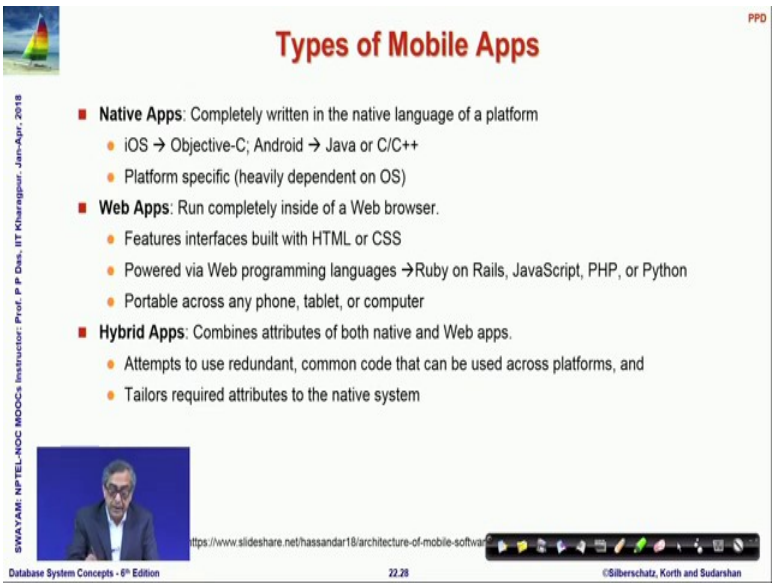
So, this is the typical architecture of a mobile app as you can see again here, that you have the three layers presentation, business and data the only difference being now, since you have a device, which can go anywhere and you have a connectivity.

So, this is this is what shows the connectivity, this is the device side and this is the pure backend or your service provider side. So, since your connectivity is not may not be very strong, you try to create all the layers of a presentation, business, as well as data on the mobile phone itself, but naturally all the data you will not have on your device ah, you are checking mails on the Gmail naturally, you will not have all the data on your Gmail.

So, what you do is you use the connection to connect to the remote database provided by the service, but primarily most of this layer of this presentation, business and a small part of the data layer are all realized within the phone itself, or within the mobile device itself. So, the data layer is split in this case, which is very different from how you do the web-based applications? And specifically, it might need customizations based on the kind of platform, you are using whether you have using android, iOS and windows, and these are all custom solutions for that and you could also note that, there are different types of mobile apps one large class is known as native application where, which is completely written in the native language of the platform.

So, for if you are doing an iOS mobile app then you will write it in object C, objective C if you are doing an android one you will write it in C or C++ or java is platform specific whereas, there is another class of mobile apps, which are known as web apps which run completely inside the web browser, so much like the way java scripts work.

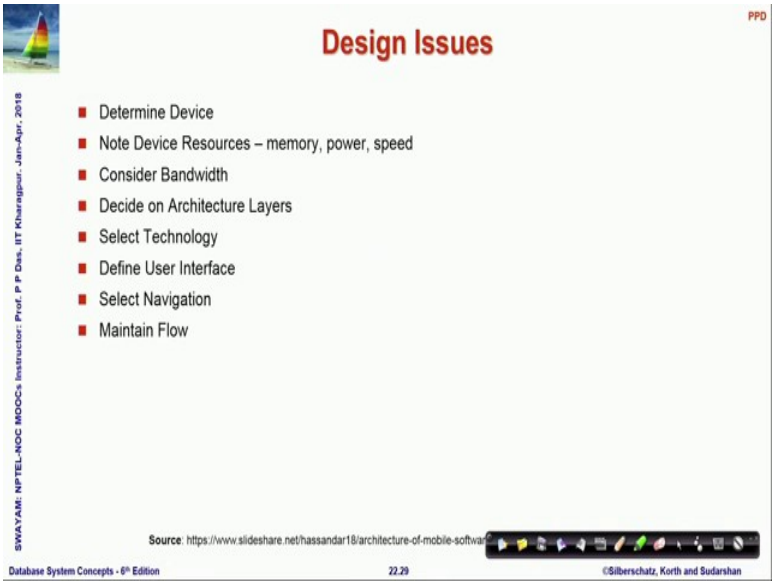
(Refer Slide Time: 32:12)



So, they feature interfaces built with HTML and the style shades, and they have powered by different web programming languages like, ruby on rails java script to PHP and so on. And certainly there is a third kind, when you combine the attributes of both native and wave application and you try to you know. So, you can very easily understand, that if your application is a native one then it is not portable across devices, this is not an iOS application is not run on android and so on.

If it is a web app app, then it is portable because it is portable across different phone tablet or computer because, you are using generic technologies. So, you could do a hybrid app also where, you could use redundant or common code, which is usable across platform and add some tailor functionality in terms of the native system. So, these are the typical kinds of mobile apps that.

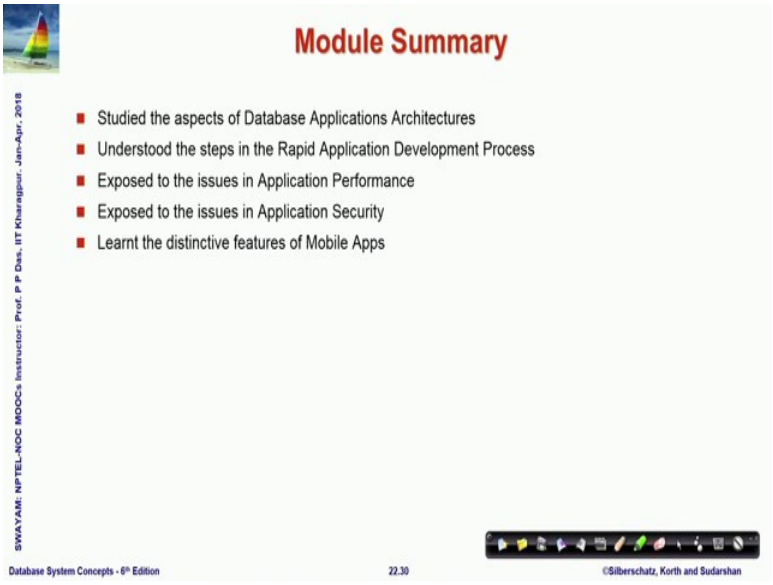
(Refer Slide Time: 33:50)



You might be developing and there a several factors to considered in the design issue, you have to first decide on the device, you have to take specific note of the typical resources, that you will your device will support memory, power, speed. You have to consider what should be the band width should it be 2 G, 3 G, 4 G or wireless you know LAN, what kind of connections you would expect? Decide on the layers in the architecture, select the technology based on the device choice and the other factors define the user interface, navigation and maintain the work flow.

So, these are very variety I will not go into details of this, but just wanted to give a glimpse of the fact, that in today's time while you are talking about database applications then it is a very reality, that you will create that as a mobile app.

(Refer Slide Time: 34:32)



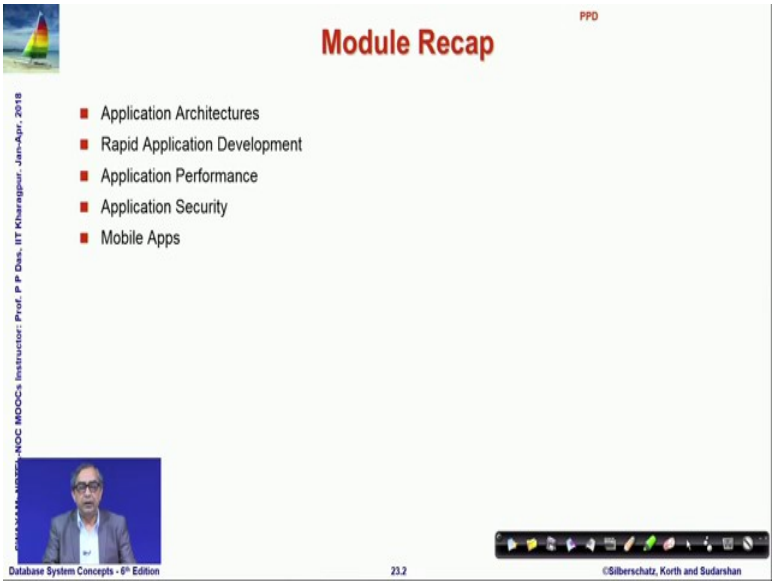
So, in this module to summarize your study aspects of database application architecture, understood the steps of rapid development, and took a quick look into the issues of application performance security, and what it takes to? What are the distinctive features of a mobile app for database applications?

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

Lecture – 23
Application Design and Development (Contd.)

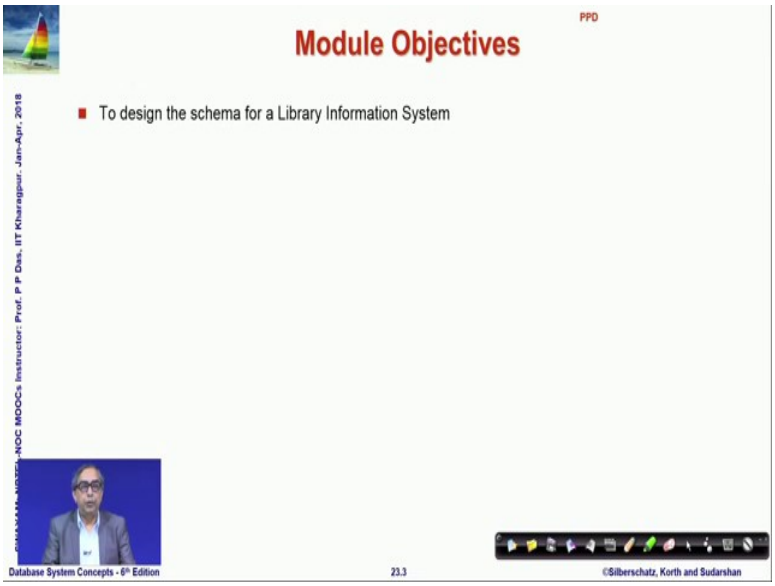
Welcome to module 23 of Database Management Systems. We have been discussing about application design and development and this is the third and concluding module in that regard.

(Refer Slide Time: 00:27)



In the last module we have discussed about different aspects of application architecture rapid development process issues and performance and security and took a glimpse in terms of, what is required for doing a mobile app.

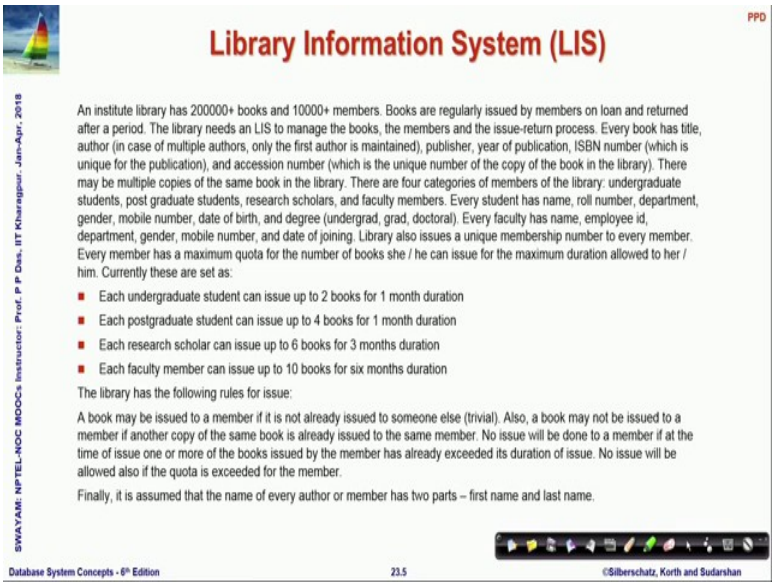
(Refer Slide Time: 00:44)



In this module, we will take care case study in terms of a library information system. We will try to design the schema for that as you have seen that for a database application design there are frontend designs there are middle tier business logic design and there will be issues in terms of the database design.

Since we are in the DBMS course, I will not focus on the whole aspects of application design, but we will focus specifically on the database aspect. So, we will start with a basic requirement specification for a library information system and then from the starting from that we will try to extract different entities and attributes and their relationships and we will make a relational schema and use different notions of dependency and how to write queries we will look into those aspects and refine that and finalize a schema for this problem. So, let us work through that. So, the outline the module issue is library information system.

(Refer Slide Time: 01:55)



So, let us start with a basic this is a very small description. So,, but yet it will give us lot of food for thought in this work and while you actually go through the rest of the video. I would suggest that you take a print out of this page or keep this page separately because you will frequently need to refer to it.

So, let me quickly go through this we are talking about an institute library that has over 2 lakh books and over 10,000 members who can use regularly issue the books on loan and return them on the expiry of the period or before that and the library needs a library information system to manage the books the members and as well as the issue return process. So, we are just looking into these aspects not the procurement of books and organization of the books and so on.

Now what is given every book has it is title author publisher etcetera a number of different attributes ISBN number accession number in terms of a book I must explain to you at this stage that any book that you that is published has an ISBN number which is an international number given so, that you can uniquely identify that book.

So, if we are talking about the database management book we are following here then that has a ISBN number, but that does not mean that number actually is unique for the book would not for a specific copy of the book. So, if you buy three copies of the book and put it in the library, then these three copies need to be identify separately by another number which is typically called the accession number.

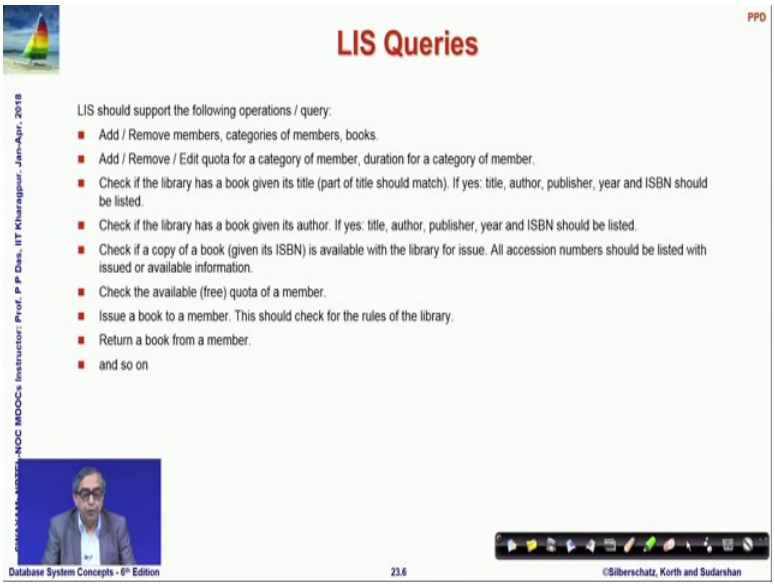
So, naturally the LIS need that every book has ISBN number which says which book it is and the accession number which says which specific copy it is, because they are may be multiple copies of the same book in the library on the member side. There are four categories of members broadly: students and teachers, but amongst students if they could be undergraduate postgraduate or research students and the faculty members. The student are specified by their name, roll number, department, gender, mobile number, date of birth, and the degree that they are doing and every faculty member also has a name, employee id, department, gender, mobile number.

And the date of joining the library to manage these members library also issues a unique membership number to every member and every member has a maximum quota for the number of books that she or he can issue and the maximum duration for which those books can be retain once issued. So, there are different specification for at different categories of member can have different quota and different duration the general rule as specified by this libraries the a book may be issues to a member naturally if it is not issued to someone else the book has to be available also a book may not be issued to a member if another copy of the same book is already issued to the same member.

So, you cannot issue two copies of the same book at the same time no issue will be done to a member if he is kind of a default that is the time of at the time of issue one or more of the other books already issued by the member has already exceeded the duration of the issue.

So, they are overdue for return in that case no issue will be allowed no issue will; obviously, be allowed if the quota exceeds and it is also specified that whenever we talk about name every name will have two parts the first name and the last name. So, this is the this is the given specification based on this we will need to make a good relational schema to represent the tables and manage the queries.

(Refer Slide Time: 05:45)

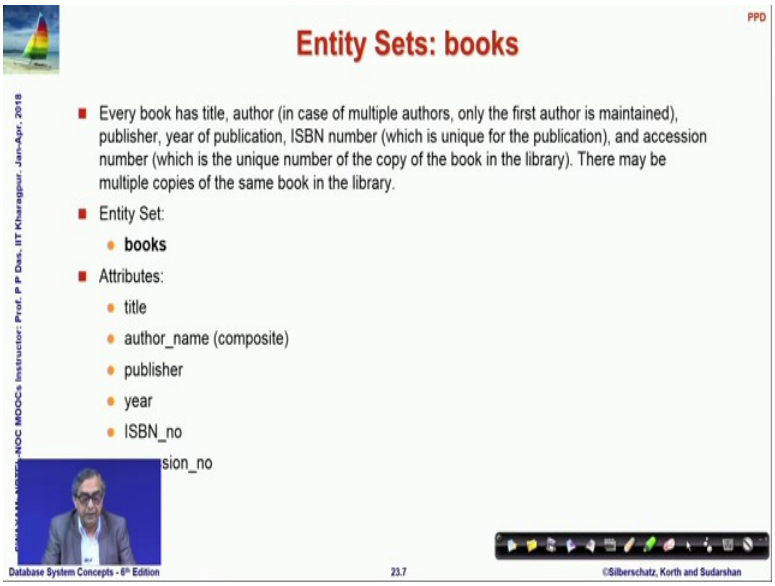


So, let us take a quick look into some of the sample queries this is just a indicative sample. Now, naturally we need a whole lot of you know insert delete update kind of queries for adding or removing members categories of members shapes the books and so, on adding or removing or changing the quota of a category of member the duration for that also we will have queries like to check.

If the library has a given book with a given title and if it is found then the details of those should be listed or we need to check if library has a book given it is author. So, if I say the author it should be possible to locate a book we should able to check, if a copy of a book if I specify the ISBN number, then whether it is available with the library for issue. So, there may be as I said multiple copies of the same book.

So, given the ISBN number it will return all the copies the accession number of all the copies and their issue status; whether they are issued or they are available it should be available it should be possible to check the quota available free quota of a member and certainly it should have features of issuing a book to a member and they should check for the rules of the library as stated it should be possible to return a book and so, on. So, these are the typical queries against which in the backdrop of which we will try to design the database system. So, what we will do?

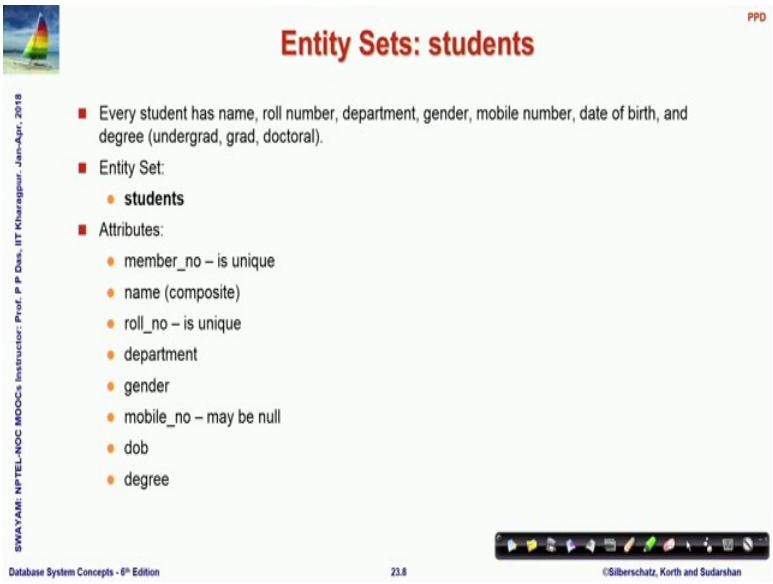
(Refer Slide Time: 07:18)



We will initially start with a specification as given. So, I hope you already have kept a copy to refer to and we will try to extract the different entity sets and the attributes. Naturally, the first entity said that we extract we let us call it books which is about books where in the beginning I have given the basic statement that is given in the so, in the specification.

So, from that we can see that books will be an entity set and it will have a attributes like title author name which is a composite one, because it will have two parts into that the first name and the last name the publisher year ISBN number and accession number. So, these are the attributes available for books. So, this is my first entity set.

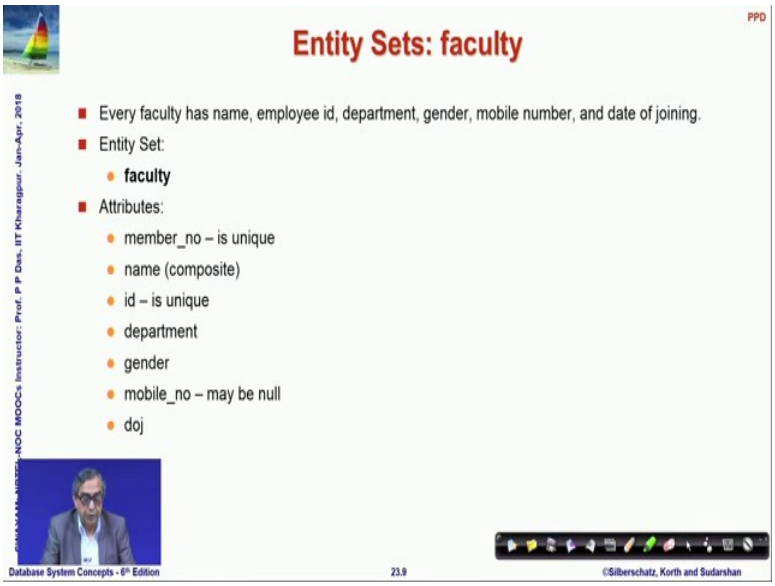
(Refer Slide Time: 08:11)



The second entity set are students. So, this is the statement about the student and from that statement, we can easily extract that entity set here is student and the attributes are member number, because certainly in the context of the library system as we said the; student has to be a member. So, it should be there will be a member number which is has to be unique the composite name the student has a roll number. So, we assuming that the role number is a unique field.

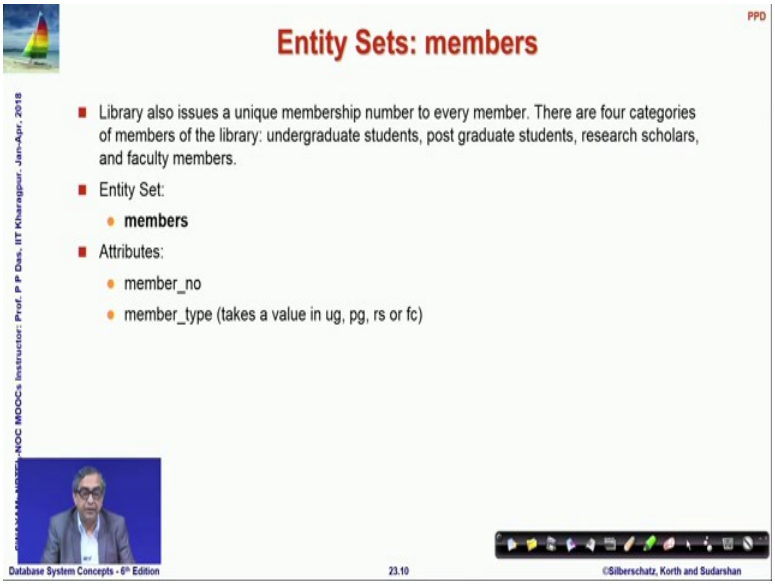
So, knows to students will have the same roll number, then there is department gender mobile number which could be null the; date of birth degree that the student has done is doing. So, those are the different attributes for this entity set.

(Refer Slide Time: 09:03)



So, we have books we have students similarly we will have faculty again the there is this process will continue by extracting different parts from the specification. So, these are statement about the faculty and we know that there will be a faculty entity set with attributes like member number name id and so, on.

(Refer Slide Time: 09:25)

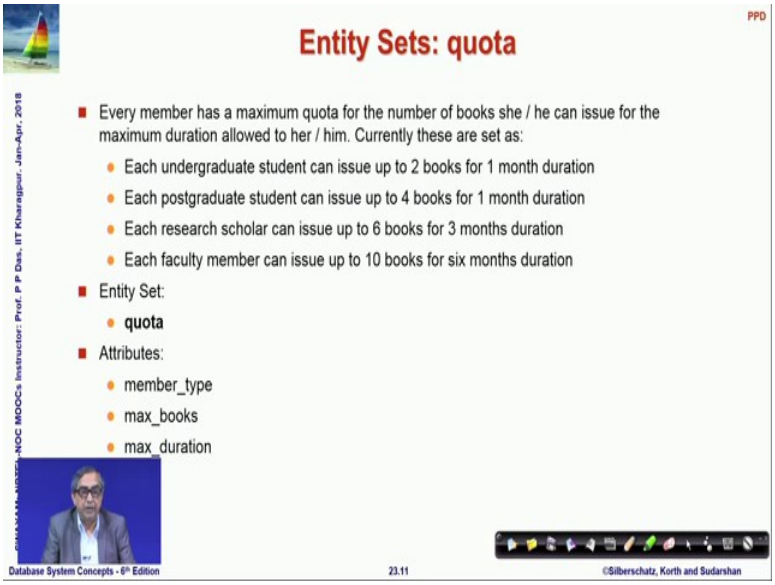


So, it should be quite obvious library has talked of that it can each it will issue unique membership number to every member and there are 4 categories of member. So, let us

we are just making attentive you know suggestion that there could be. So, it looks like members are in entity which the library has to interact with in terms of issue.

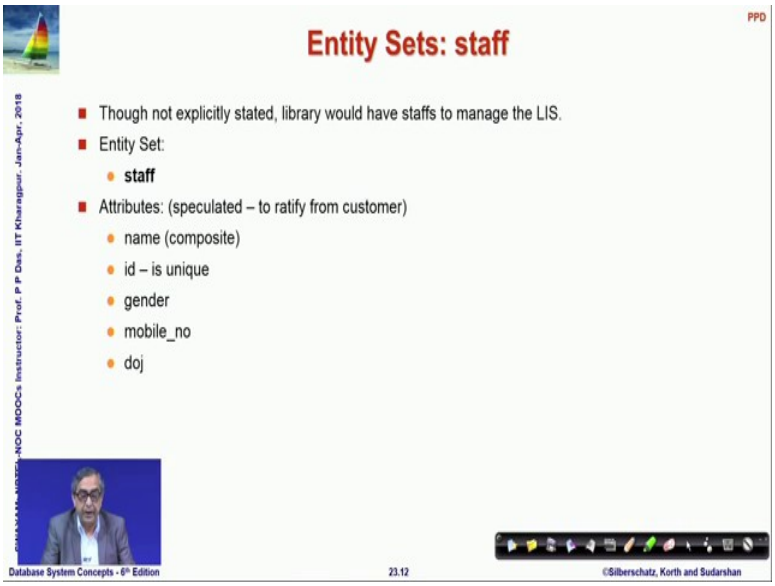
So, let us create an entity set members which has the member number and the member type. So, which can take different types of undergraduate post graduate these different one of these four specific values let us say.

(Refer Slide Time: 10:00)



The rules have talked about having a quota. So, every member will according to it is member category we will have different quota. So, to represent so, quota becomes an entity set. So, we would like to represent that for different member type which is a category; how many number of maximum books can be taken and what could be the maximum duration? So, let us say we will put the maximum duration say in terms of months and with these three attributes we will have an entity set which is quota.

(Refer Slide Time: 10:33)

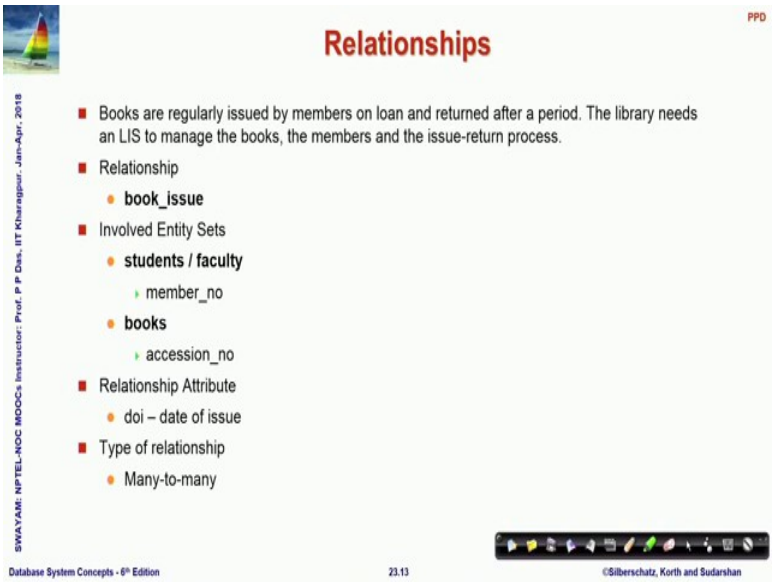


Here, I am talking about another entity sets staff if you again carefully read the specification you will find that there is no mention of this staff in the in the total of the specification, but if we logically think and this is what we often need to do. When we deal with a practical specification like somewhat like, the one that I have given here is that we would need to look little beyond the specification.

So, think about it if I have the library with books students faculty issue process quota and all that then certainly they will have to be some staff of the library; that will actually do all this operation. So, they will be able to log in to the database and actually issue a book return a book check for validity and so, on.

So, let us assume that we have a entity set staff which certainly a staff should have name and id which id should be unique, gender, mobile number and date of joining. So, this is kind of a speculative addition to the design of entity sets and this must be ratified from the customer; when the opportunity arise, but something like that must be there for to make the design complete.

(Refer Slide Time: 11:48)



So, these are ah; obviously, the immediately visible entity sets that we see. So, the other part that we will now have to check on is a relationship. So, again we pick up the relevant statement in this regard books are regularly issued by members on loan and returned after a period the library needs an LIS to manage the books members and the issue return process.

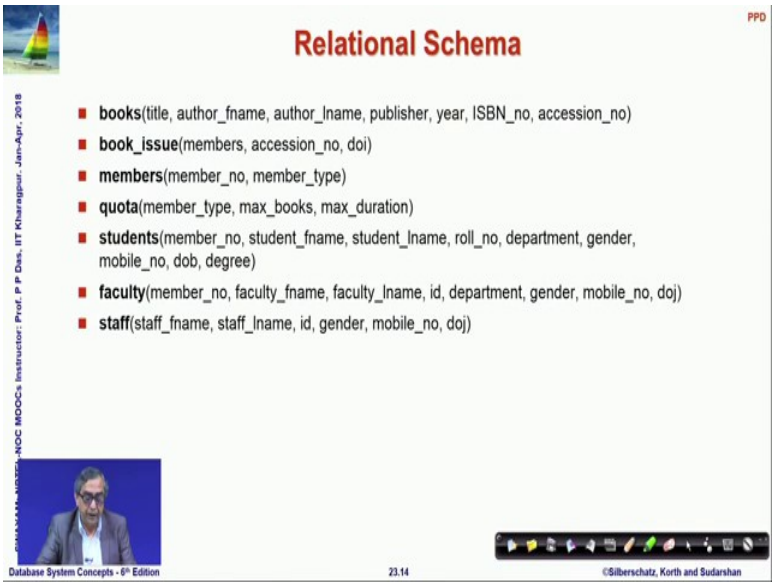
So, certainly there will have to be a relationship of issue between the student or faculty in the books. So, we loosely define this relationship on one side there has to be the books and the other side would be the involved entity set would be either student or faculty so, actually though I am just noting it here as a as a single relationship, but actually there is a relationship of issue between students and books and faculty and books.

So, we will have to see how to handle these kind of situation now certainly the students or faculty are identified by the unique member number of the library that has been given books as we have said are identified by the accession number. Again, please note that we are not talking about the ISBN number, because ISBN number could be same for multiple copies of the book, but when you issue you issue a specific copy. So, that accession number which is a unique for a specific copy needs to be tracked. So, these are the two attributes, which will certainly be involved in the relationship and then you would recall that often relationships of their own attributes.

So, here we have one the date of issue needs to be recorded, because we want to check conditions like; if the borrower is has issued the book and how many for how many months he or she is keeping that book. So, if you have to check that we need to know when the book was issued. So, this is a attribute which does not exist in any of the entity sets that are involved in this relationship neither in students or faculty nor in books, but this is a attribute of the relationship which needs to be specified.

And of course, it is this relationship is a many to many relation; because every student or faculty can issue multiple books and every book may be issued by different, but we can we can in general we can be specific to say that this is many to one also if we want to maintain that at a any given instant a book can be issued only by one person. So, if you look at it from that perspective this will not be many to many this will be treated as many to one.

(Refer Slide Time: 14:24)



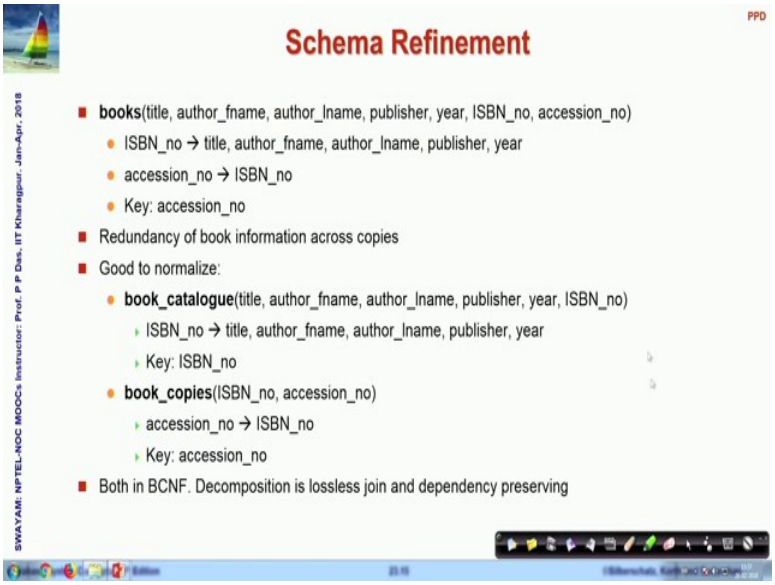
So, having done this finding having found out this entity an attributes and the relationships let us now start with a relational schema.

So, what I have done here; in terms of the relational schema that for each of the entity set. I have created a relational schema by the same name and have put the attributes as identified as attributes of that relational schema. So, this is a very straight forward once we have identification, made this identification process from the original specification. This is a more straight forward process where you just convert every entity set into a

relational schema and also we have converted the relationship there was a there is a relationship called book issue here as you can see the book issue this also we have converted to a relational schema involving the two attributes of member number and the accession number.

So, let us see how this will span out later parts so, having done this.

(Refer Slide Time: 15:38)



The next task would be to work on the refinement of the schema. So, now, it is time that the relational schema is available the first schema is available. So, now, we have to apply all different notions of the relational design to refine this schema and finalize. So, for what will do; we will take every relational schema and we will try to identify the functional dependencies and use that to identify the key.

So, if I look at the books relational schema, then we easily know that **ISBN number →title, author first name; authors last name, publisher, year**. So, here also you may you may just note that since name is stated to be a composite attribute I have designed here to use two different attributes the first name and the last name. So, with that we have this functional dependency which tells me that given ISBN number, I know these details, but of course, that does not tell me what is the accession number because there could be multiple copies, but another functional dependency must hold because, every copy of the same book must have the same ISBN number. So, given the accession number, ISBN number should be determinable.

So, **accession number →ISBN number**. So, if you do the sample computation on this you will easily figure out that a key of this is accession number. So, this is what we currently have this this is what we have done now. So, you can see that if there are say five copies of a book having different accession numbers which are the key they are has been number would may all be same.

So, if I have the same ISBN number I can multiple accession numbers and therefore, there are different records, but if that accession number of two books are I am sorry the ISBN number of two books are same then all of that title, author and first name last name all this attributes will be repeated and if there are multiple copies of the book then each one of them will have a separate entry because they have a separate accession number.

But, their ISBN number and everything else will be redundant. So, we can easily see that the given relational schema actually has redundancy across copies of the book and if we want to look at this from formal theory we can easily check that the keys accession number. So, **ISBN number →title, author, …. etcetera** that functional dependency violates the boyce code normal form actually this is the also not in the third normal form right now.

So, we would do well to reduce this redundancy and it would be good to normalize. So, here I have not gone through the actual steps of normalization, but I am showing you more intuitively as to, how you can normalize; because it is a quite obvious that the accession number is involved only with the ISBN number and which keeps track of the copies.

So, we propose to have under normalization we propose to have one which is let me just highlight and show you. So, book copies where the ISBN number and accession number will be maintained. So, for every accession number we will be able to see the ISBN number which will tell you which book it is and rest of the book details will be kept in this say another new relational schema called book catalogue which will have all of the earlier attributes expect the accession number.

So, now, if you project the; this dependency on book catalogue the dependency will be fully projected. So, the whole dependency is preserved if you project this dependency on the book copies it will also be fully preserved. So, this decomposition is a dependency

preserving and you can easily check that these two can be joint by a natural join using ISBN number as the common attribute.

So, if we take the intersection of the two sets of attributes then ISBN number would be the attribute and **ISBN number →all attributes** in the book catalogues. So, our lossless join condition **r 1 ∩ r 2 → r 1** here. So, this will also give me a lossless join and if you check on each one of these relational schema then this functional dependency the left hand side is a key and therefore, book catalogue is in Boyce Codd normal form and book copies is also in Boyce Codd normal form.

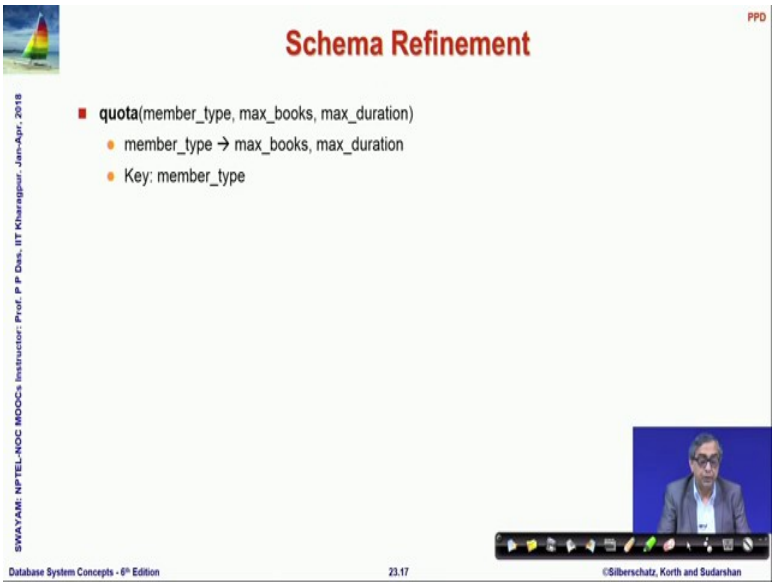
So, you can you could come to the same result by doing the formal process of functional Boyce Codd normal form decomposition, but I have just shown you here as to intuitively how you get this. So, intuitively becomes very clear that you have details of the book that you keep separate in a separate table, because that is not change across the copies and we have a separate table to maintain the specific information about the copies. So, that takes care of the books relationship moving on let us look at the other.

(Refer Slide Time: 21:16)

So, book issue is a relational schema which comes from the relationship between the now between what now book issue is to happen between student faculty and books. So, it is I mean we cannot have two of them of these entity sets occurring in the same. So, initially let us just put that well the library has a concept of a member. So, what if the book issue is a relation simply between the members and the and the books of course, we

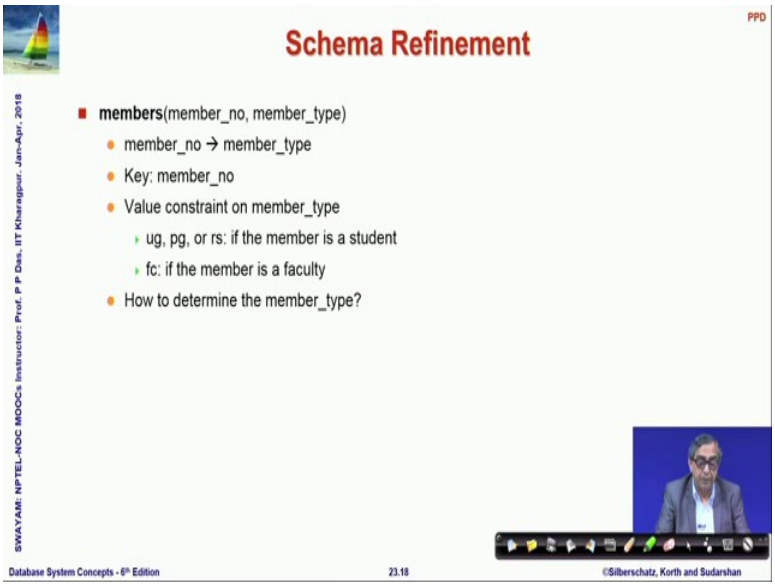
do not know the relationship between members and students or members and faculty, but we can at least refine the book issue to be between member and the book and therefore, member number and accession number together becomes the key which → the date of issue this is already in normal form as you can see.

(Refer Slide Time: 22:11)



Quota is a simple relational schema where member type → the max books and duration and that is the key in normal form

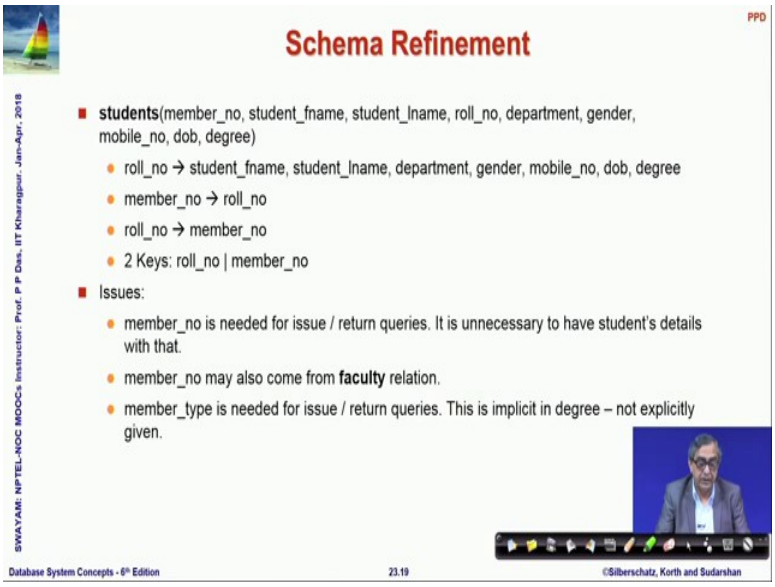
(Refer Slide Time: 22:19)



The members I mean we have identified something like a member which should list all the members of the library.

So, existence of a member number in that list will mean that someone holding that member number is actually a member and it is should also tell me what type of member or what category of member he or she is. So, member number must → the member type and the member type will be constrained in terms of possible 4, 1 of the four possible values are stated here now of course, we will still have to figure out is to where do we get this member type from and so, on and that information is not present here. So, further refinements will be required in this regard.

(Refer Slide Time: 23:02)



Let us go to the students this is the whole schema that we had done. So, **roll number → all the; attributes** specifically **roll number→ member number**, **member number →roll number**, because every student has a unique roll number and; obviously, has a unique member number also **number →all attrib**. So, **member number →roll number roll number→ member number** and so **roll number →rest of attributes**.

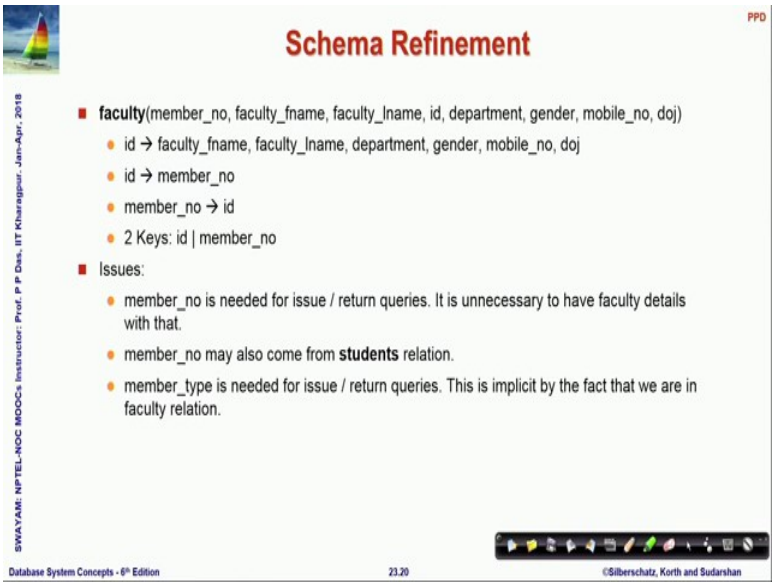
Which mean that this design this relational schema has two keys the roll number and the member number now are we happy with this design that is a question we need to ask. So, we can figure out that member number is needed for issue return or queries book issue has member number, that is; what we were thinking of; now when we want to deal with

this book issue issuing a book to a student and returning a book from the student and soon.

Is it necessary to have all the students details with that one that make every record very heavy and if we want to extra connect with that do some join or if you want to do some query unnecessarily we will have to deal with very large units of data and as we will see in the next couple of modules that if a record becomes larger dealing with it become naturally becomes more cumbersome. So, there is some kind of discomfort at this design similarly a member number is not here it is the student relation. So, it is coming from the student relation, but in general in terms of issue it may also come from faculty relations.

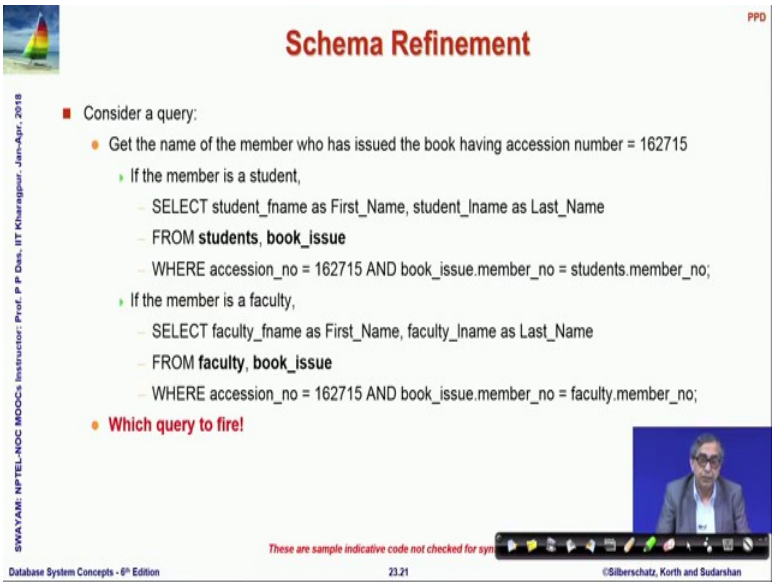
So, how is that handled we do not know then issue return also needs the member type which is implicit here in terms of the field in terms of the attribute degree which tells you that student is a undergraduate postgraduate or research, but it is not explicitly maintained anywhere. So, these are the issues in this form of the design that we that we came.

(Refer Slide Time: 25:06)



Moving on look at the faculty we have a very similar situation as of the **student id → all attributes** of the faculty **, id → member_number**, **member number → id** every faculty has two unique numbers two keys. And we have similar type of issues as we have seen in terms of student. So, we will need to do something in terms of this.

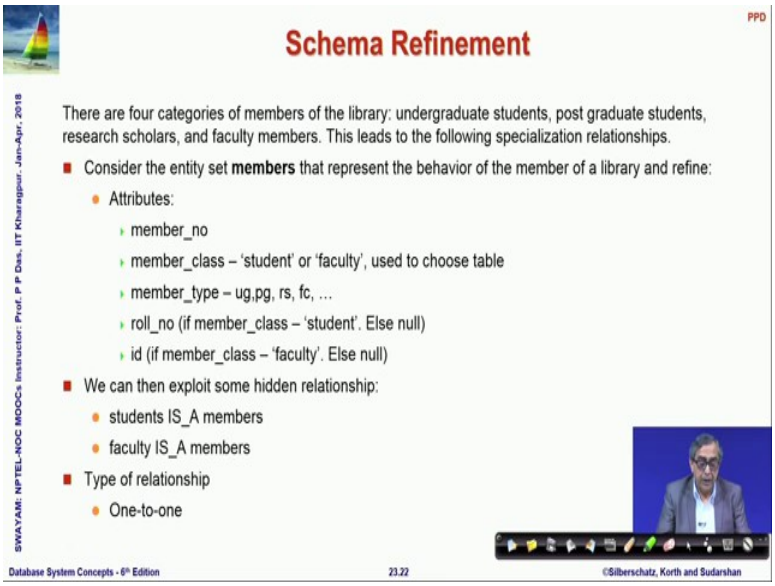
(Refer Slide Time: 25:34)



So, what can us; I mean what kind of issues we will get into. So, let us consider a query that the query is to get the name of a member of the member who has issued a book say having accession number some accession number 162715.

Now, if we have to write a select query for this we will need to know whether the select query should be written on the student or with the faculty. So, if this member is a student then we need the first query where we do a join between student and book issue to find out the student member number who is issued that book where the accession number gives me the particular book issue record from this result will come. Similarly, if the member is a faculty I need a different query. Now, it is a problem because if I have two possible queries and based on the member number I have to decide which query to fire we have not done any mechanism like that. So, this design has problems.

(Refer Slide Time: 26:36)



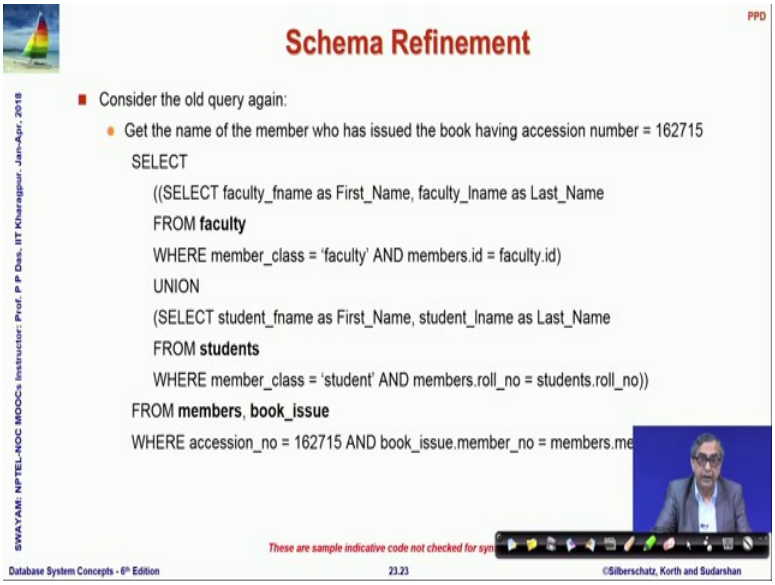
So, let us see what we can d? So, we again go back to the specification and see what the specification says; it said that there are four categories of members undergraduate, postgraduate, research scholar, and faculty members and least to the that least to the specialization of this relationship and we have already done a members concept members entity we did.

Which has a member number and the member type, but now we have just seen that it actually matters as to whether the member is a student or is a faculty is a more unique deciding factor in terms of, how we design our query? So, we introduce a new attribute which was not specified explicitly say; let us call it member class which can take only two values either student or faculty and then retain the member type. So, what it will mean that the; if the student class is student then the member type could be ug, pg or rs and if the member class is faculty.

Then the person is a faculty than the member type should be only fc and then also maintain the roll number and id in this members table. So, the roll number ah; obviously, if it is if the member class is student then the person is a student and the roll number will exist, but the id which is an employee id will be null and at the same time if member class is a faculty for a record then the roll number will be null because, a faculty cannot have a roll number, but the id will be present.

So, we extend the members entity set in relational schema with these attributes and once we do that then we kind of exploit a hidden relationship which was not very explicitly stated anywhere that; now we can say that students is a members faculty is a members, that is; from the perspective of issuing books and using the library both students and faculty can be consider to be members it is kind of a you know virtual entity that we can see here and certainly there will be a one to one relationship between the students and members and faculty and members.

(Refer Slide Time: 28:39)



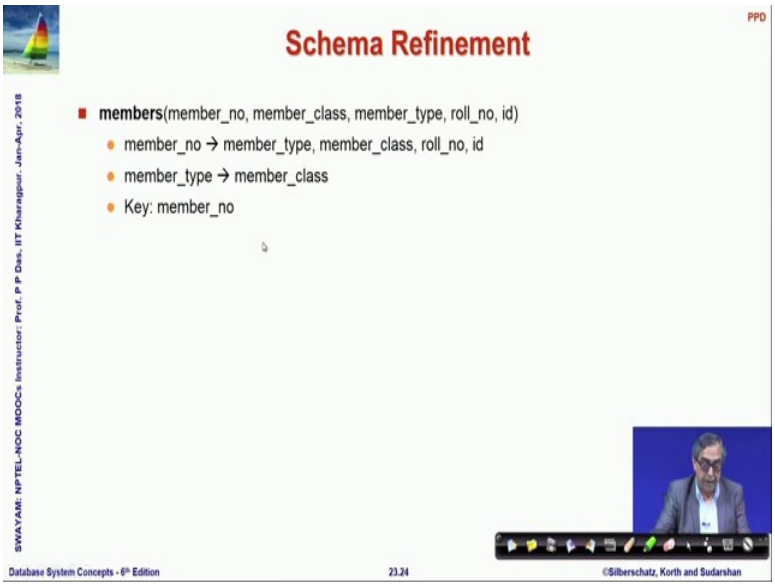
So, let us see what is a consequence of that; in terms of the earlier query. So, we look at go back and look at the query again we will see that problems have got significantly solved, because we now still have to find that member name and this is the basic condition which say.

But, I am sorry this is a basic condition which say the specify the member number who has issued that book, but the actual problem of finding the name can be solved here, because I can I write two queries and take a union of the first query runs on faculty the other query runs on student. Now, here since I have a member class which tell me whether the member is a faculty or the member is a student. So, one of these queries will actually return a null result will not written any record because it will not match this condition, but the other one we will match and will give me the record give me the corresponding names first name and last name of the member and certainly to ensure that

when we are looking at the faculty. We need to check the equality of member id between the members relation and the faculty relation and while dealing with the student we need to check for the equality of the roll number.

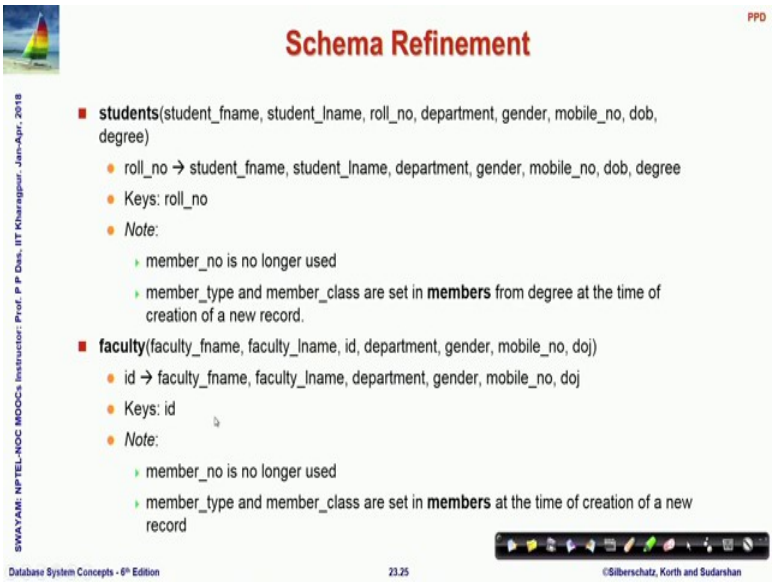
So, in this way by using intelligently using the union feature and the you know nested query feature we can easily write a query and where implicitly. Now, based on the data the switching of which table I am I am actually looking the data from will get solved.

(Refer Slide Time: 30:17)



So, with this refinement my members schema now turns out to be **member number →member class, member type, roll number, and id member, number** and **member type → member class**, because if I know some member type is undergraduate I know member class is student and so, on key; obviously, is one number.

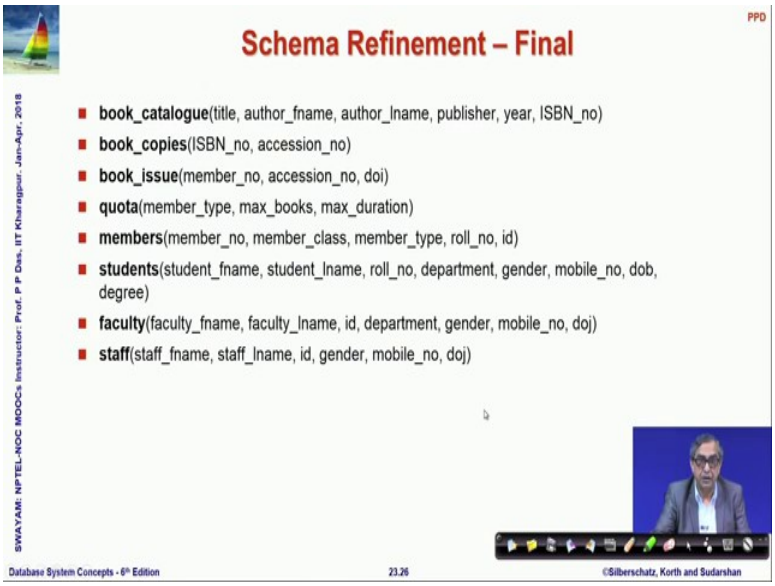
(Refer Slide Time: 30:38)



So having done that, we again go back to the student and faculty, because now we do not need member number in that anymore; because these will not directly be involved in the issue return, because all that you need is just the member number which is already there in the members. So, now, it is a roll number which determines everything member number is no longer used.

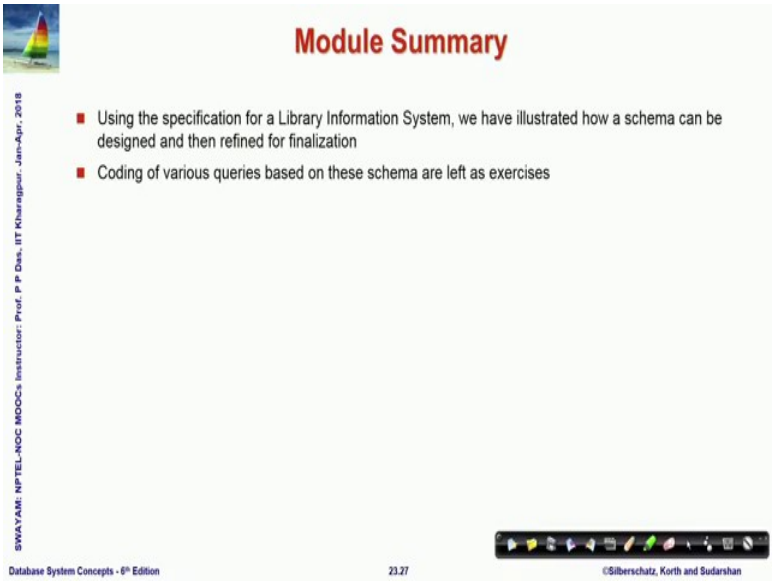
And member type and member class which we have assumed is exist in the in the in the members will have to be derived from the degree value at the time when a new record is created. So, when a new student record is created an entry will happen in this relation as well as a corresponding entry we will need happen in the members relation where the membership number is given and the membership type will be derived from the degree similar change will happen in terms of the faculty as well.

(Refer Slide Time: 31:31)



So, after refinement now we have these eight relational schema from book catalogue to give the details of every book copies which keeps the information about, how many; what are the different copies book issue; is the basic issuing information. We have quota members has the virtual kind of relation that we have created to support the notion of members of the library and students and faculty are related to this members either through roll number or through id in a selective manner and staff we have not touched.

(Refer Slide Time: 32:12)



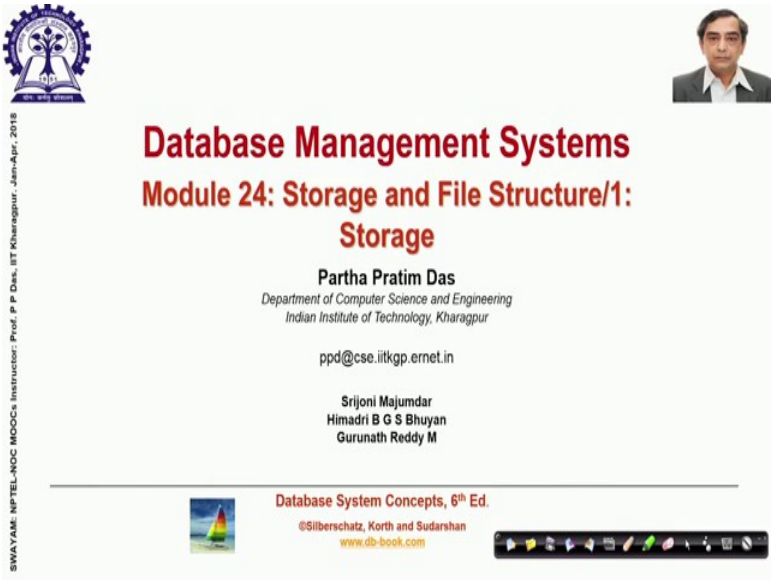
So, this is the final relational schema. In this module I have shown you illustrated you that how starting from a very simple specification you can reason and work through in terms of creating the entity sets attributes and relationships and then get into the relational schema look at the possible functionality dependency and refine that and also look into what will be the requirements of doing your query and come to a final refined schema.

So, various queries that we had talked of and others can be can now be coded on this, but I leave that as an exercise to you please try out coding those queries and you will be able to learn in terms of how to do that well and I will try to also supply some supplementary information on this offline.

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

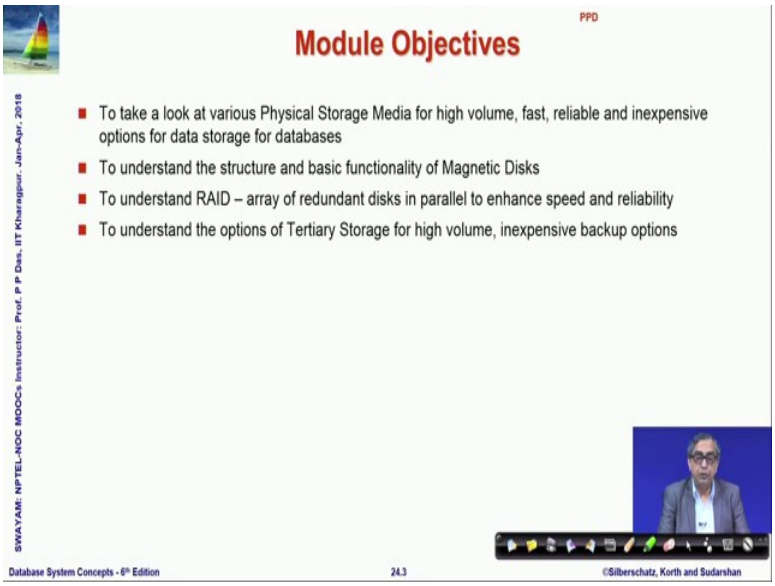
Lecture – 24
Storage and File Structure: Storage

(Refer Slide Time: 00:16)



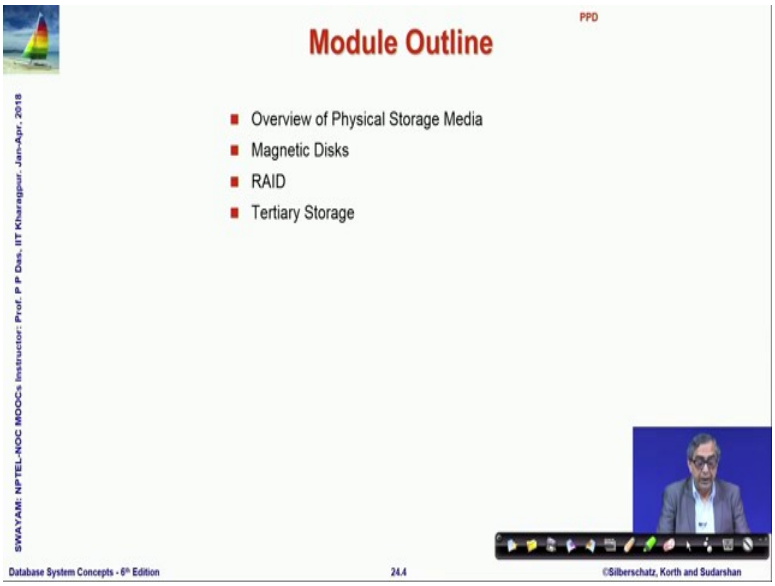
Welcome to module 24 of database management systems in this module and the next we will take a look at the storage and file structure of database systems. So, we will start with the storage.

(Refer Slide Time: 00:30)



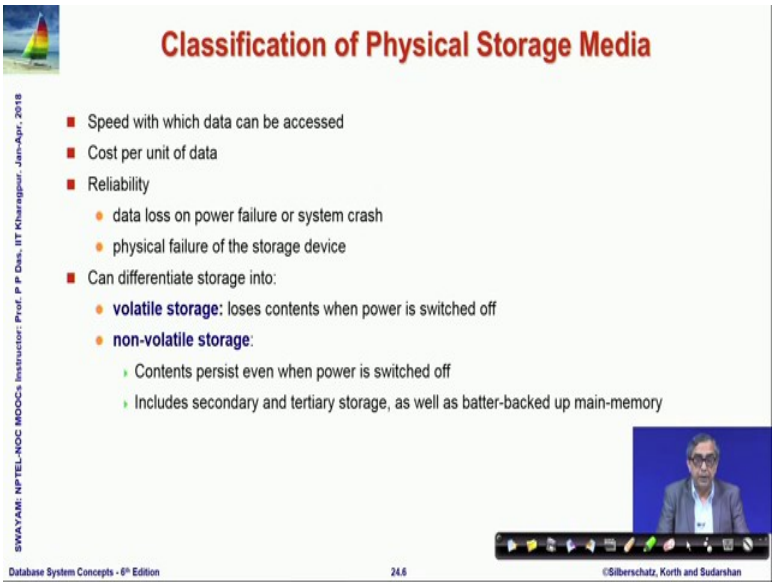
So, specifically we want to look at various physical storage medium because. So, far we have been talking only about the logical layer of the database design and now we want to actually look at the in physical terms how the data will be stored what could be the physical storage medium for high volume fast reliable inexpensive options for databases. We would like to understand the structure and basic functionality of magnetic disks, because they are they are most widely used we will try to take the glimpse about RAID which is a kind of a good option in terms of reliable databases and also look at options for the tertiary storage.

(Refer Slide Time: 01:12)



So, these are the topics that we will quickly cover in this.

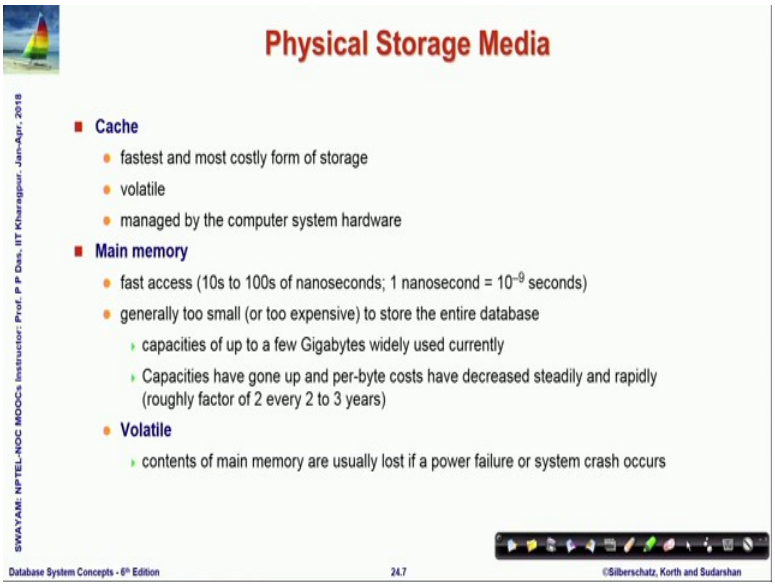
(Refer Slide Time: 01:17)



So, first let us take a overview of the physical storage medium I am sure all of you have known all or parts of this. So, this is, but this is more for completeness to look at from the perspective of a database application. So, some of the the classification of storage media done on different factors the factors includes speed which is the first thing the how fast the data can be accessed the cost per unit of data you can say the rupees per bit or rupees per byte or rupees per kilobyte something like that.

So, which is a cost per unit of data the reliability that is the if we will the data get lost if power fails or if the system crashes and or if this physical you know failure of the storage device and so, on. So, what is the reliability on that; and broadly as you all know we can differentiate storage into volatile storage which loses contents. When the power is switched off and the non volatile storage which are secondary and tertiary storage where the data will continue to stay even when power is off even you have some parts of the memory which may be battery backup which also will be non volatile.

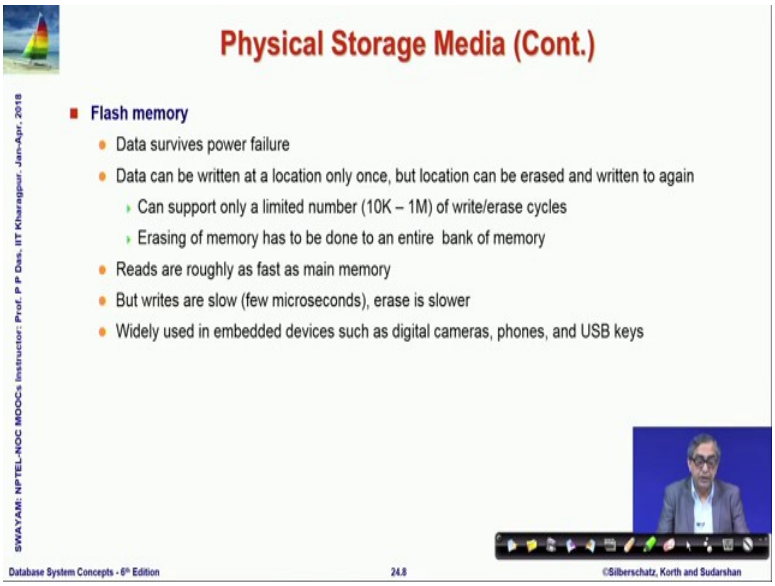
(Refer Slide Time: 02:29)



So, in terms of the physical storage certainly the absolute starting point of the storage is registered in the CPU; we are not talking about that because they are primarily meant for temporary computations. So, in terms of data the first possible level is the cache which is the fastest and most costly form of storage it is volatile in nature and is managed by the computer system hardware.

So, cache typically is a fast semiconductor memory that exist between your main memory and the disk system and it is very fast to work with then comes the main memory which is which has fast access, but compare to cache it may be may be much bigger, but overall it is too small to store an entire database, but I mean every regularly the size of this main memory is increasing. So, the capacity of couple of gigabytes are common these days, but still it is small compare to the requirement of the databases and main memory typically is volatile. So, if the power goes up the system crashes all the data is lost.

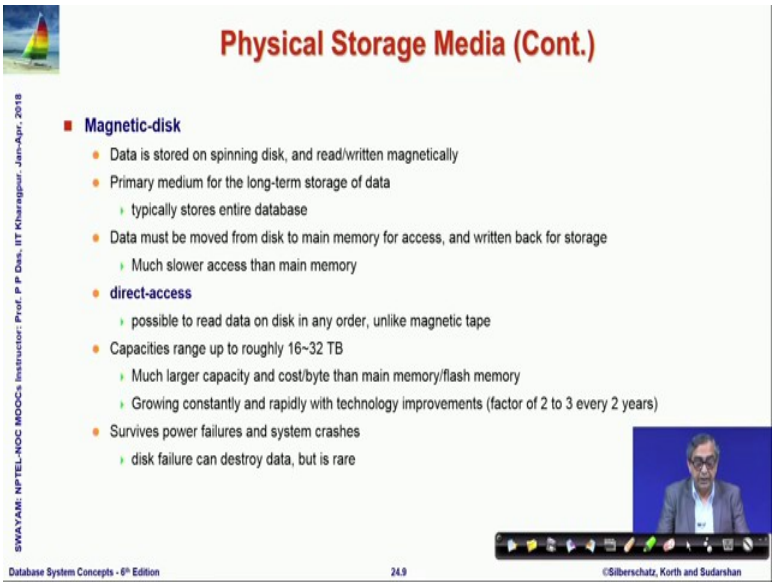
(Refer Slide Time: 03:48)



We have flash memory where the data can survive across power failure; it can be they had data can be written at a location only once, but you can erase and write it again.

So, it is not like in the main memory where you can read write read write like that here you can write and then if you want to write again then you will have to erase and write it. So, the read is very fast in case of flash memory which is almost as fast as a main memory, but writes a slow particularly when you have erase and write it will be a slow process all the kinds of USB keys pen drives digital phone memory that we are often using are actually flash memory.

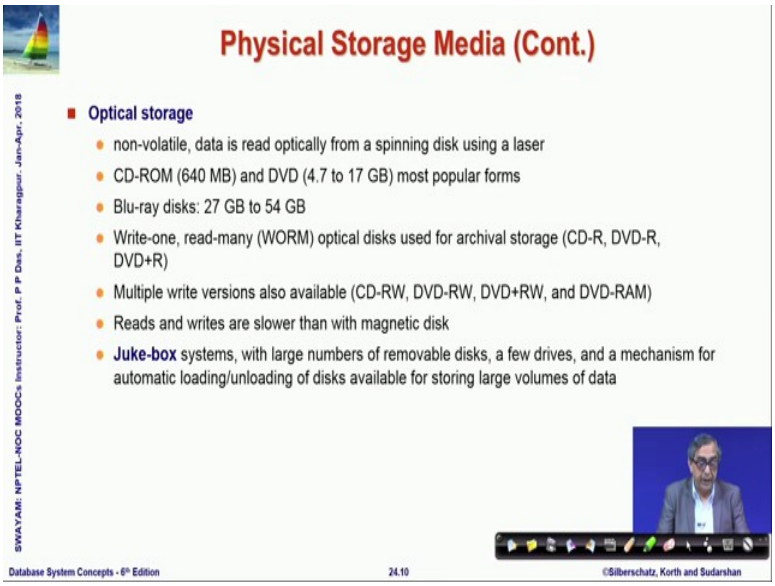
(Refer Slide Time: 04:35)



Then you have the magnetic disk where the data is stored on spinning disk and it is typically written and read magnetically. So, this is the primary medium for long term storage of large volume of data. So, data needs to be moved from disk to the main memory and written back for permanent storage. So, it is ways slower compare to the main memory and, but it is has a kind of direct access which means that it is possible to read data on this disk in any arbitrary order in compare to magnetic disk.

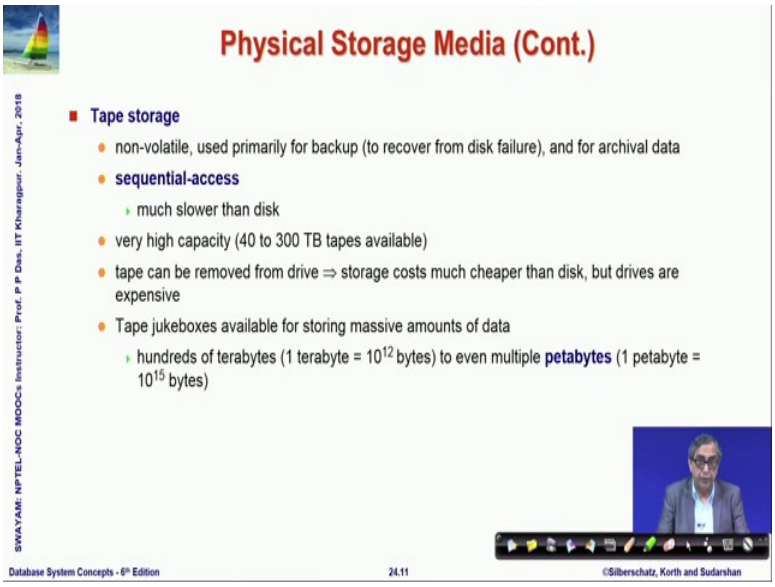
Which is the serial device here it is a, it is kind of a I can do things in parallel at random in any order capacity is go up to tens of terabytes easily and it can survive for failure and system crashes, because it will the magnetic recording will still be there if the disk itself fails then it will the data will get distract, but such a situation is usually rear.

(Refer Slide Time: 05:40)



We have different optical storage devices CD-ROM, DVD and so, on the juke box systems and which are also non volatile and data is written optically here, that is by using a laser light on the spinning disk use a typically optical storages are removable media.

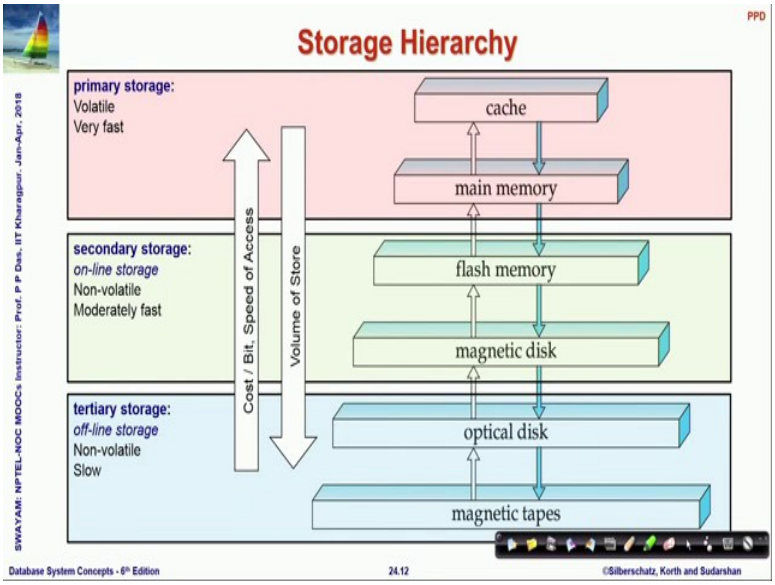
(Refer Slide Time: 06:03)



Then you have the tape storage which usually is a largest volume of storage, but it is as a name suggests it is a tape it is a linear device. So, access can only be sequential. So, if you want to read the 6th record you have to skip of a record 1 to 5, but it can be a very high capacity usually it is slow.

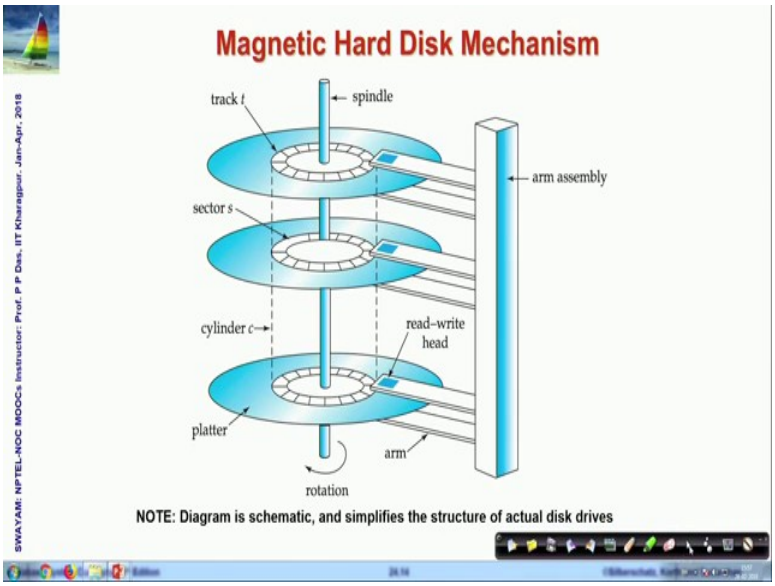
But very large in volume and tape juke boxes can support hundreds of terabytes or even multiple of petabyte. So, that is for offline storage.

(Refer Slide Time: 06:35)



This is a big medium. So, this is the basic storage hierarchy. So, we broadly classify them in to three groups primary storage which is volatile and very fast cache and main memory is within that or secondary storage which is an online store which is non volatile and moderately fast and tertiary storage is called the offline store which is non volatile and slow. So, flash memory and magnetic disk are secondary storage they are non volatile and moderately fast and they are online. So, they exist with the system whereas, optical disk and magnetic disk can be removed and taken elsewhere.

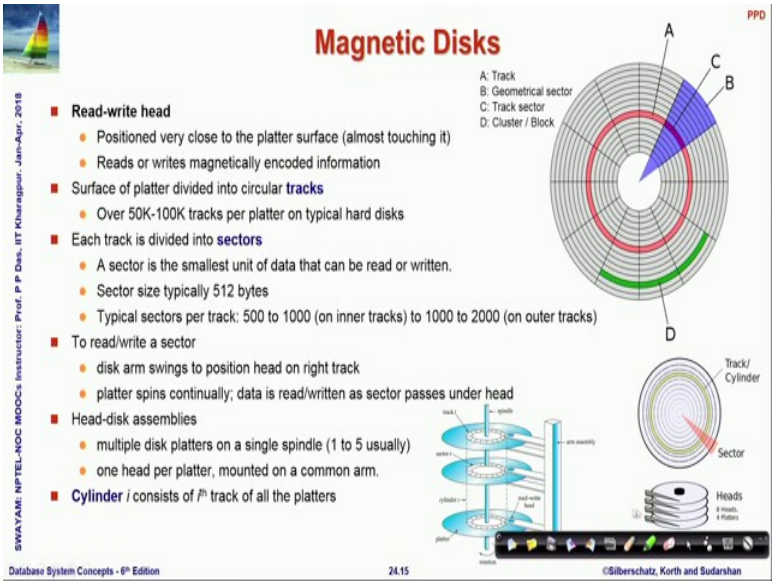
(Refer Slide Time: 07:12)



So, let us quickly take a look into the magnetic disk this is how a typical magnetic disk looks like; these are the different cylinders these are the different disks that we have and these are all different read write head.

So, as you can see all of them can work along a path backward forward like this and they can come and parallelly all of them parallelly can read from the different disks and this disks keep on spinning to help you look at the data anywhere on the disk. So, that is a typical structure of a magnetic.

(Refer Slide Time: 07:50)

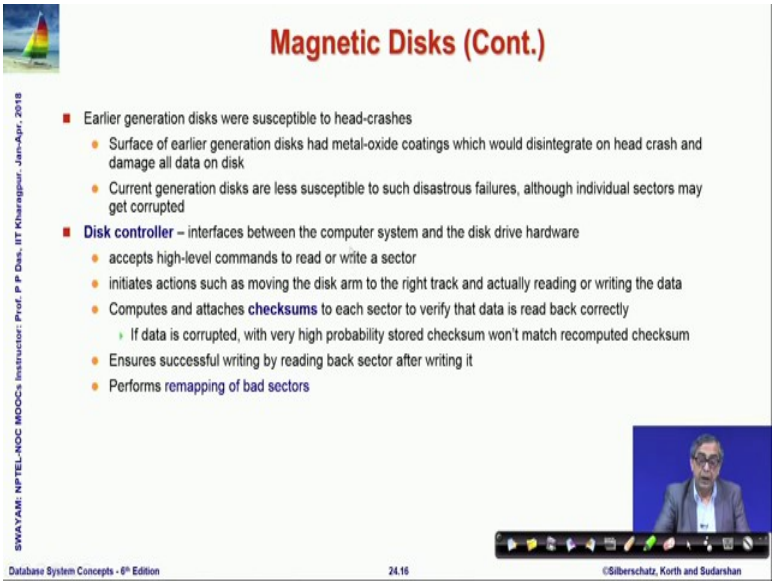


Disk if you look specifically into. So, this is this is the structure we saw and if you specifically look into one particular disk then the disk is radially divided into different sectors portions. So, these are these are all separate portions and you can at a time the head can read one such sector.

So, these are called the geometric sectors and the whole of the ring that you can see the cylindrical ring that you can see is called a track. So, this is a track sector the orange one is a track sector and often we take multiple sectors from a particular track and combine them into one unit this is called the block of data and we will often talk about the block of data.

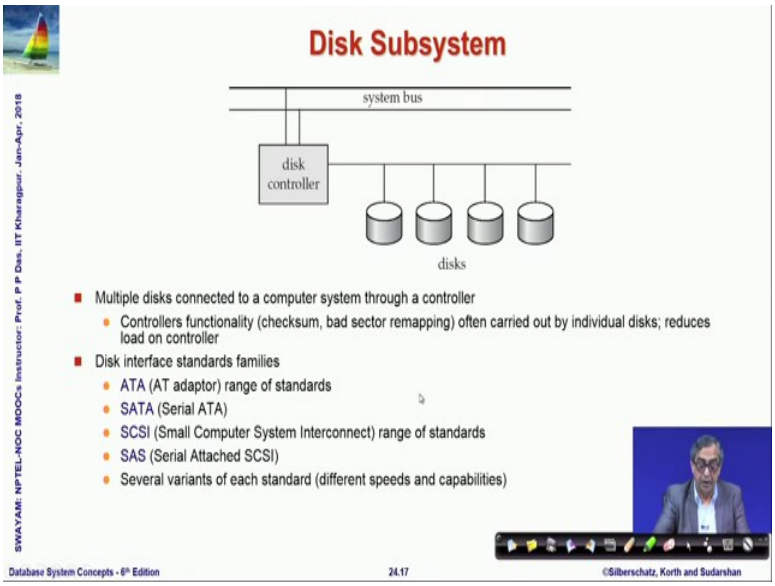
So, here at the typical numbers of how these sizes of this different units turn out to be.

(Refer Slide Time: 08:44)



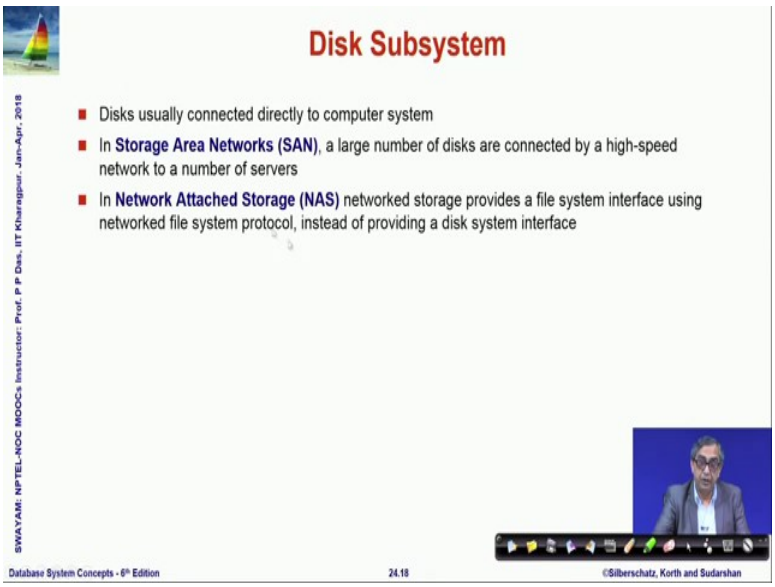
So, the early generation where of disk were susceptible to head crashes, but now a days it is moved it quite stable there are disk controllers which regularly check and manage the; read write into in terms of the sectors naturally the. So, many heads being in parallel the data can be written on to multiple sectors at the same time and so, for doing that.

(Refer Slide Time: 09:12)



We the we will see what is the mechanism for that; and we also have disk subsystems where if you need a large a volume of data to be managed which is larger than a single disk. Then you can connect multiple disk and through a disk controller you can actually read write data on to them and there are different such disk interface standards that you may have heard of.

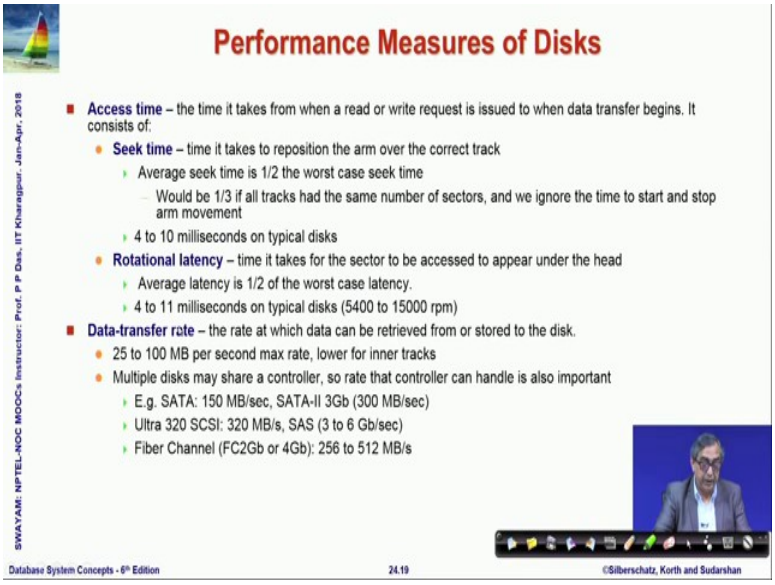
(Refer Slide Time: 09:38)



We will just mentioned that these are the typical storages the SAN and NAN are the typical storages.

So, if you come across these terms we will not go into details of that that takes us into a different course of hardware discussion, but these are the very common storages for database disk systems.

(Refer Slide Time: 09:56)



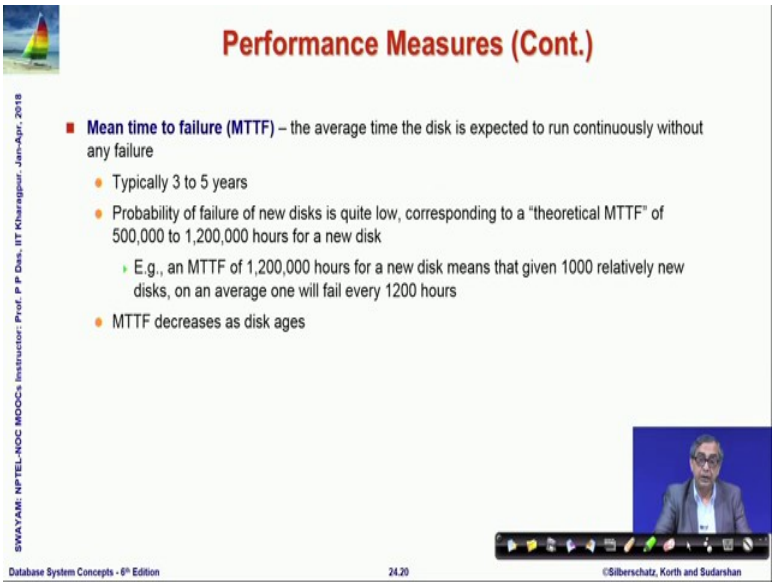
Now basic question is for doing this read write on the disk what kind of performance measures we should look at. So, one main measure is access time if I want to access a record from the disk, then how much time shall I need. So, this is based on two major components one is a seek time now naturally as we have seen that the disk platter as a whole range of read heads which are positioned.

So, to be able to get the data the platter has to the head has to move forward or backward to the right track on which the data is there. So, this time it takes to go from current position to the correct track where, I find the data is called the seek time. Now, even when it is come to the; correct position in terms of the correct track it may not actually be on the correct sector.

Because, it is at the whole track is a is kind of a circle. So, it may be at one part of the circle and the actual data may be in a sector which is in a different part of the circle. So, the disk will have to rotate. So, that the correct sector comes under the head which is already positioned through the seek. So, the time to seek plus the rotational latency the time it takes the for the sake sector to be access is gives the total access time and then comes the other measure which is the data transfer rate as to you using this then what is

the rate how many megabytes and so, on can be transferred at from this. So, it that depends on a besides the access time you will have to see what is the rate at which the disk can be copied from the magnetic medium to the semiconductor medium and transferred.

(Refer Slide Time: 11:52)



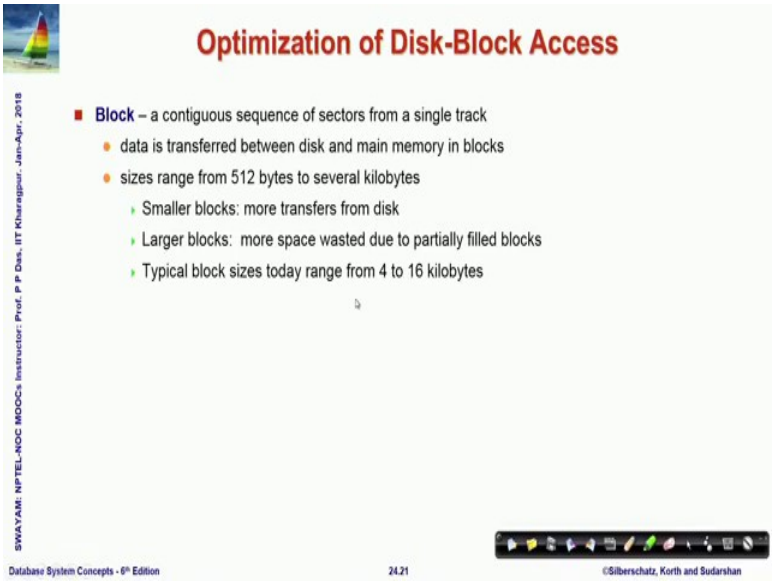
Ah the other performance measure that we are concerned with in the database system is known as MTTF which is mean time to failure.

Because, database systems as you know are dealing with persistent data. So, data has to exist and therefore, the disk on which we keep the data must be very very reliable. So, meantime to failure is conceptually that if you consider two failures of the database of the disk to consecutive failures then what is the time the time elapsed between them the average of the time that has elapse between them now. So, if we say the, it is typically now mean time to failure for this magnetic disk that we use today at typically 3 to 5 years.

So, the probability of the failure of a new disk is very very low. So, it is kind of a theoretical MTTF we which will be said like this which; obviously, in terms of hours it it does not really make a make a physical science. So, what it in technical in in more practical terms, what it means that if you are given thousand relatively new disks then on average one of them will fail every twelve hundred hours.

So, this is what is a. So, MTTF certainly decrease it will it will decrease as a disk becomes old. So, it decreases with age. So, they will start failing sooner than it is to do.

(Refer Slide Time: 13:22)

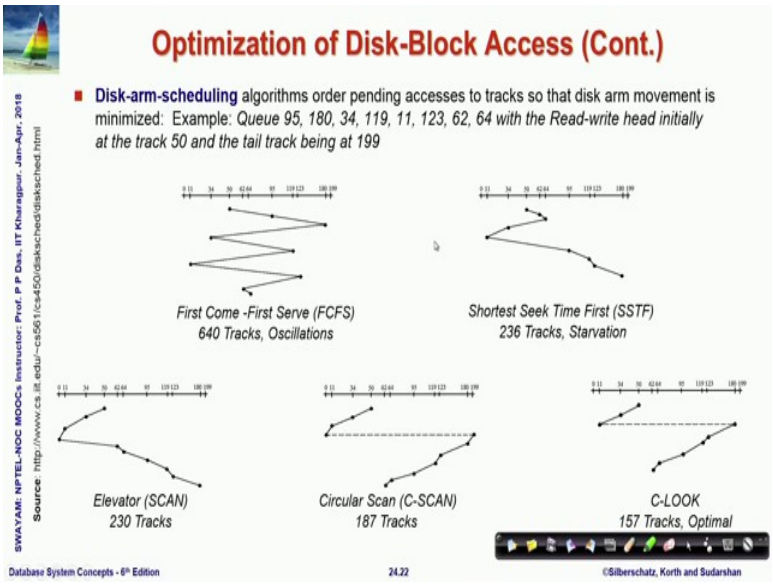


Now we have to basic objective is to transfer the data at a fast rate. So, what we try to optimize is a; what is called a block the contiguous sequence of sectors from a single track which I mentioned earlier. So, this is what we will be read at one go. So, once you access the data one block will be read block size can range from 512 bytes to several kilobytes. If the blocks are smaller than naturally will need more transfers from the disk if the blocks are larger then you might waste lot of space because your part of the block will not can be used with the data.

So, with all that consideration the typical blocks size that use today is 4 to 16 kilobytes. So, you will see that in all our subsequent discussions particularly with the access and the file organization and the indexing we will consider that a block is one unit.

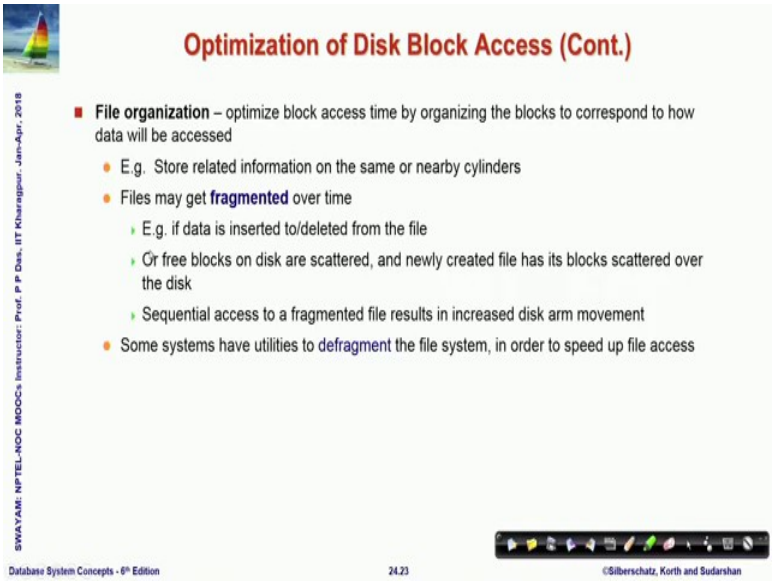
Which can be fetched in one go and that determines the size of the basic information node where the information about the records will be maintained.

(Refer Slide Time: 14:36)



This is the more details in terms of how you move your disk head. So, I think will skip this is more advance material.

(Refer Slide Time: 14:46)

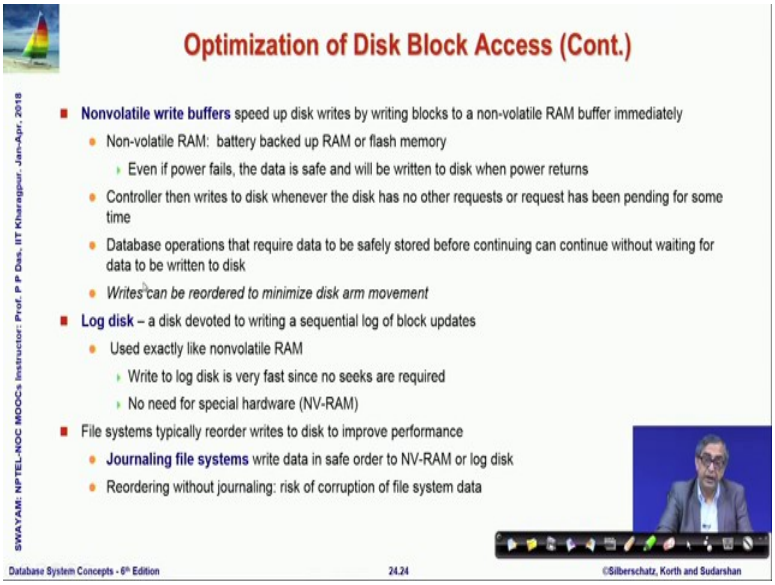


So, to optimize the block access we need to organize the blocks corresponding to how the data will be access. So, certainly if the related data are kept in nearby blocks then naturally you are disk head and the rotation will have to be mean will get minimized. So, the basic idea is store the related information in nearby cylinders in the nearby cylindrical practice, but as you keep on using you may start with that, but as you keep on

using for various types of insert delete and overflows that keeps on happening. So, the data gets very fragmented which means that the data gets spread over a whole lot of widely separated cylinders and so, on.

So, the time to actually seek the data increases. So, we can correct that by doing what is called a defragmentation process. So, by defragmentation what you do is your data which is got distributed all over the disk you try to bring them together again to logically continuous physically contiguous blocks so, that their access time can improve. So, databases often will periodically defragment the file system.

(Refer Slide Time: 16:06)



The other way look at the optimize block access is by using buffers. So, idea of the buffer is suppose you want to write some data to the disk. So, naturally for writing the data also you will need a seek time you will need the rotational latency then we will have to data do the data transfer.

So, if you are trying to do some write you have you can take another option that you actually write that to a buffer a buffer which is in the memory in the it is a in the a semiconductor buffer where you can very quickly write and then when you have enough data in the buffer then you can take them in a single go to the disk.

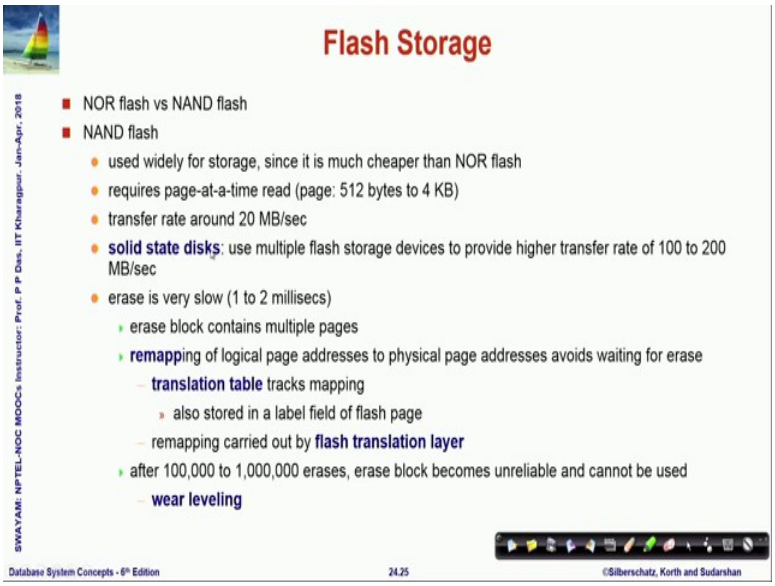
And write that or when the disk is not actually doing something some access you can use those cycles to write that now naturally if you write things onto the buffer then it is

possible that while your buffer has some data which has actually not been written to the disk if the power fails then you will lose that data. So, parts of the data which you think have been written actually have not gone to the disk. So, to take care of that often non volatile memory ram are used which are battery backed up or flash memory.

So, that even if power fails the right buffers will not be lost. So, we say that use non volatile buffers and other option that is used often is you maintain a log disk a log disk is nothing, but when you are writing to the disk you make a sequential log of the updates that you are doing. So, this kind of is a report. So, you are saying that I have written this data to this block written this data to this block.

So, if there is a failure, then if you can go through the log and you can actually retrieve the information of what was lost how far it actually worked correctly and you can used exactly like the non volatile ram and using the log you can find out. So, you are keeping a kind of a general link system you are keeping information of this is what I did this is what I did. So, the all this write information are maintained in terms of the log.

(Refer Slide Time: 18:30)

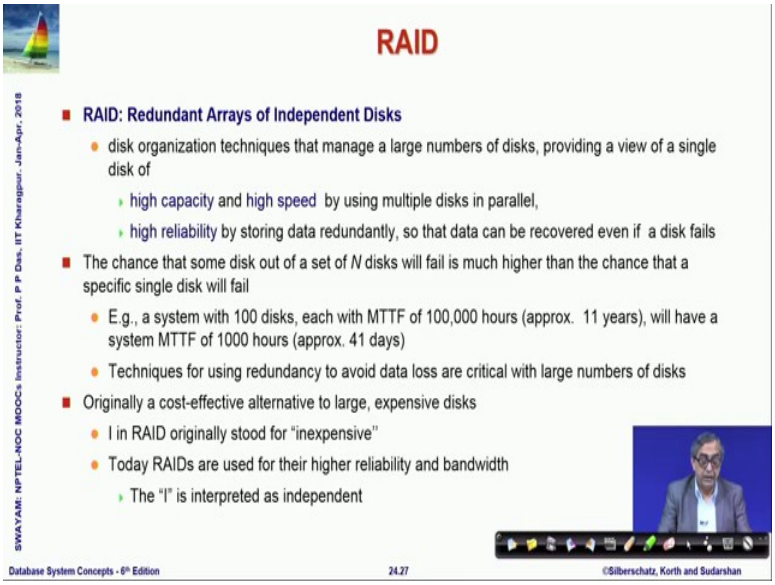


You can use flash storage nowadays the NAND based flash storage is are very common.

(Refer Slide Time: 18:40)

So, here are some details on that let us move on to understanding. So, we just took a look into the basic storage hierarchy and the use of magnetic disks and what are the parameters that control the performance in terms of the read write in terms of the disk RAID is a the full form is redundant array of independent disks.

(Refer Slide Time: 19:05)

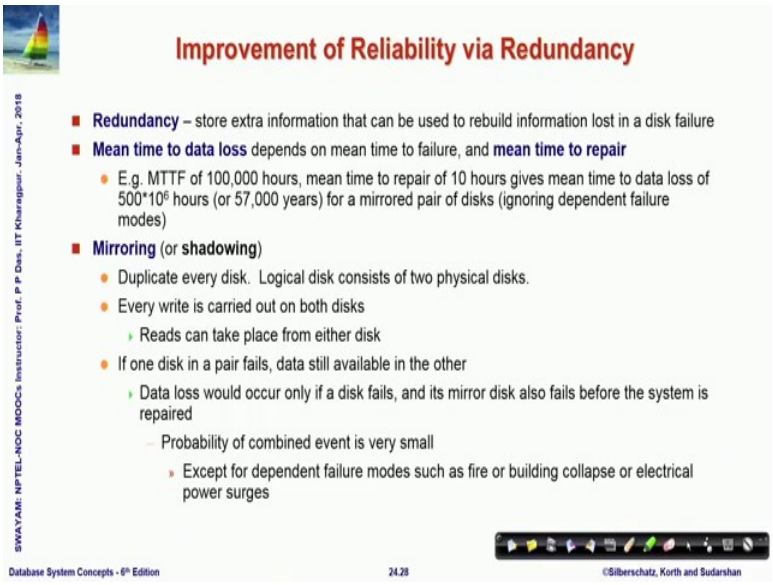


So, the disk organization that manage a large number of disk, but the view that you get you do not get to see the multiple disk you get to see a single disk.

So, by this you can create very high capacity and very high speed and. So, because you have multiple disks so, you organize a data in such a way that when you are writing or you are reading actually you can parallelly read or write from multiple different disks. So, that not only increases capacity, but actually increases a throughput and you can get high reliability also by storing the data redundantly; that is keeping multiple copies and that is what RAID is often quite known for.

So, originally when RAID was originally designed actually the, I in red stood for inexpensive. So, it was kind of a disk array which was inexpensive to afford, but now it is not particularly the expenses is not the primary factor for which we do go for RAID, but we go for RAID for the high capacity high speed and high reliability. So, the I is now interpreted as independent array.

(Refer Slide Time: 20:21)



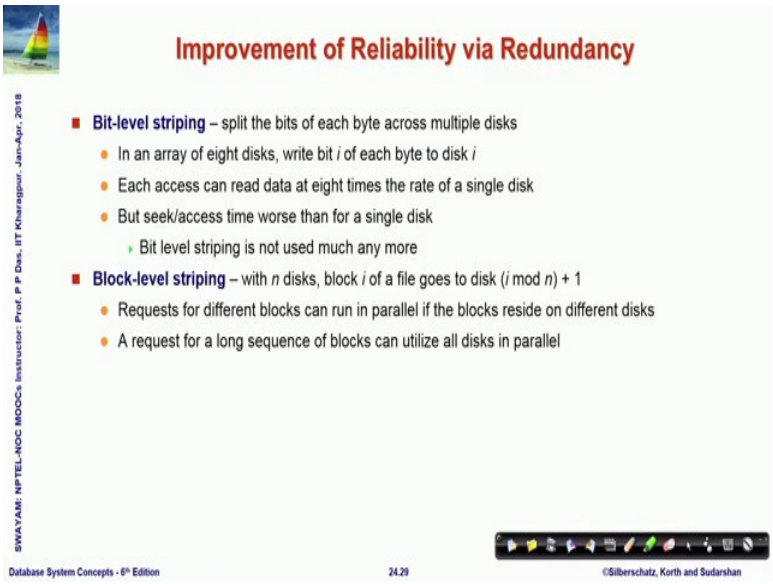
So, we can improve. So, now, the main issue in red is to improve reliability. So, the main the key idea is have redundancy to improve the reliability that is stored the data in multiple copies and can rebuild from the copies once a disk has failed. So, we earlier looked at the MTTF mean time to failure. Now, we are look at another parameter which we say is a mean time to data loss. So, which depends on mean time to failure, because if you are if you are failed then you have lost the data, but when you have failed you have a possibility now, to recover the data to repair it; because you have redundant copies.

So, mean time to data loss it depends on the MTTF plus the mean time to repair how quickly can we use your redundant copies and get back the original data. So, mean time to data loss is the actual key factor which needs to be minimized and for this red does a mirroring or shadowing that is for every disk there is a duplicate there is a clone.

So, it the logically you have one disk, but physically you have actually two disk. So, every write that you do is carried out to both the disks and when you read the read happens on either of the disk usually it it switches between the disks. So, if one of the disk in the pair would fail, then the data can still be recovered from the other and the data loss would occur if a disk fail, but the mirror disk also has to fail before the system has been repair.

So, if one of the disk fail you get the data from the middle disk you continue to do that from the mirror disk you restore the other disk you may be replace and put a new one mirror it again and. So, on, but you can restore that. So, in between this time if the mirror disk also fails; then you have actually lost data, but the probability of that is very very small. So, mirroring gives a at the expense of naturally having lot more of redundant storage the mirroring can actually give you a much higher reliability.

(Refer Slide Time: 22:39)



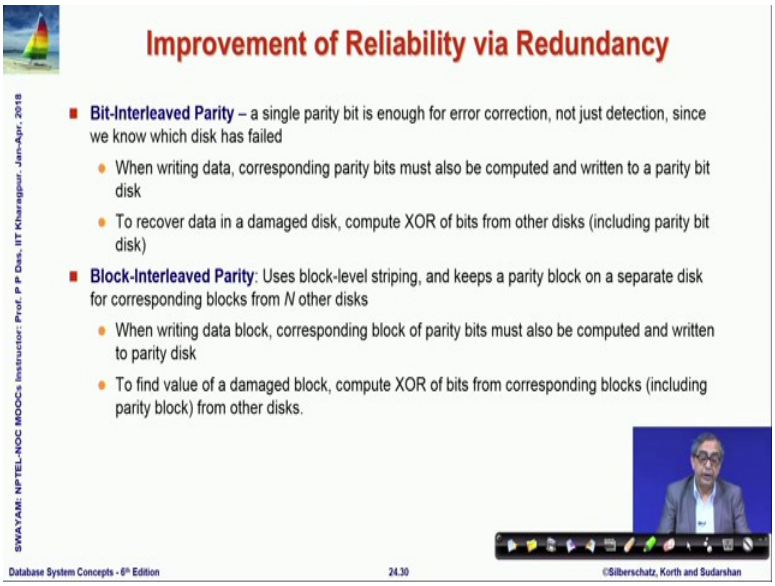
That we expect today another some of the other techniques which improve reliability is bit level and byte level block level striping techniques. So, what you in the basic bit level striping what you do is a say every byte has 8 bits. So, when you are writing a byte you

do not write all the bytes to the same disk you write them to multiple disks. So, you take an array of 8 disks. So, write bit I of each byte to disk I it is. So, did interesting concept you have fragmenting it in a very peculiar way.

So, you have 8 disks and every byte first byte first bit is written to one disk second byte is second bit is written to the second disk, third bit is written to the third disk and so, on. And when you access you can access from all these 8; now naturally which means that this will decrease your throughput to some extent, because you have to collect from all of that reconstruct. So, bits levels striping is not much in use any more instead you have block level striping where with end disk a block I of a file goes to the disk i mod n plus 1.

So, circularly so, the first goes if you have say five disks in the first block goes to disk one second to disk two-fifth to disk 5 and the 6th again back to disk 1. So, the request to different blocks can run in parallel and reside on different disk and if they can easily utilize this parallelism to improve the throughput.

(Refer Slide Time: 24:30)

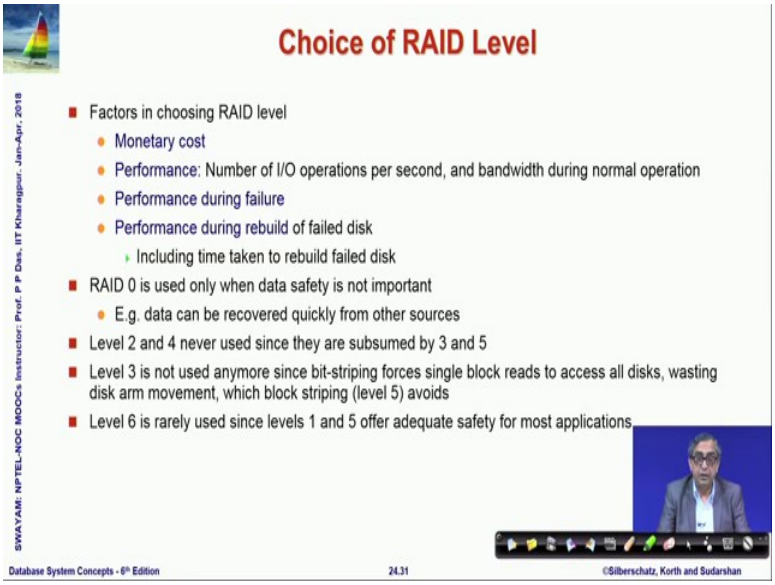


At the same time improve the reliability now how do you improve the reliability. So, for improving the reliability you use the basic you know error correcting coding concept you just use a bit level that again two options one is you can do a bit inter lift parity which means a single parity bit is used which is good for error correction. Usually, we know that if we use a single parity bit we can know a single error we can detect a single error,

but here with a single bit you can correct the single error also because you know in case of a failure you know which particular disk has failed. So, then you can exalt with a data from the other disk and reconstruct the error bit.

So, the other is naturally block inter leaving of the parity which uses block level striping and keeps a parity block on a separate disk for corresponding blocks from n other disks and you can reconstruct in a very similar manner. So, by using block interleaved parity with block striping you can really have a higher throughput with a better reliability.

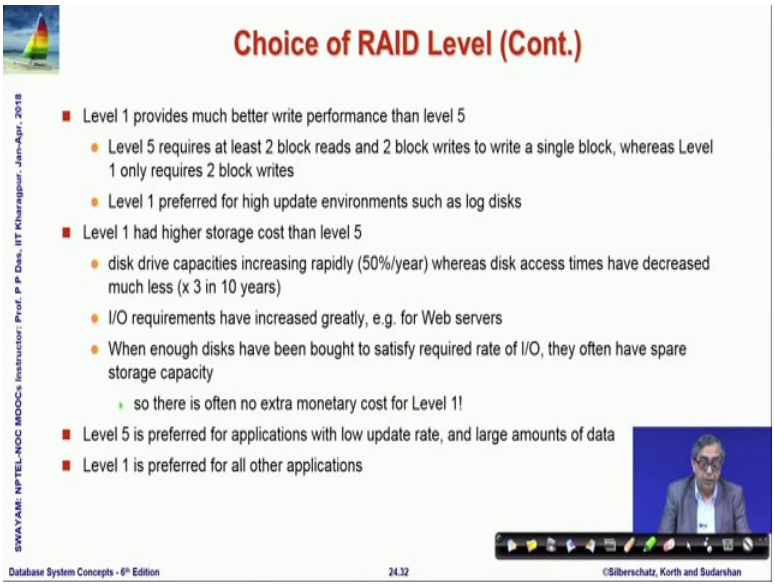
(Refer Slide Time: 25:44)



So, it is a win win situation now naturally when you go for RAID you will find that there are different levels of read that are available today goes up to level 6 right. Now, and the factors that certainly has to be considered is I mean different RAID at different kind of cost the what is the performance what is the performance during failure; that is performance during failure is typically the MTTF and performance during rebuilt is a mean time to data loss so, including the rebuilding and so, on. So, based on these factors different rate levels can be looked at RAID 0 is used only when data safety is not important. So, that is not very common the RAID level 2 and 4 are never used.

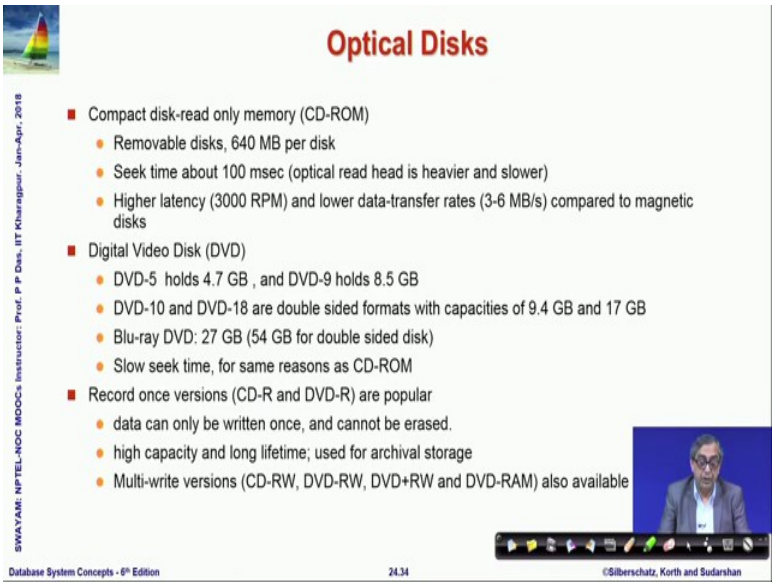
Because, they are they got subsumed in level 3 and 5. So, you can ignore that level three is also not used anymore, because it used bits striping and naturally we talked of that that is not that is worse compare to the block striping which level 5 uses and level 6 is also really used. Since level 1 and 5 adequately support all applications.

(Refer Slide Time: 27:08)



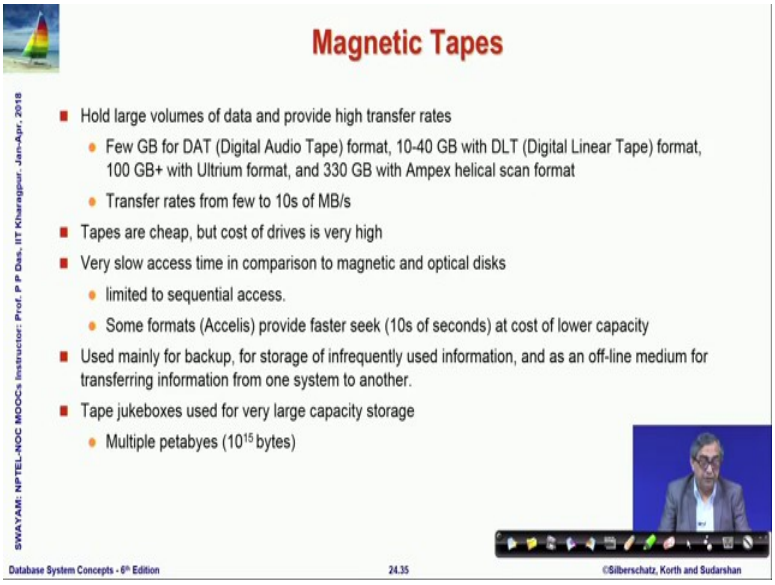
So, the conclusions simply, is that you either use RAID level 1 or you use RAID level 5. So, RAID level 1 gives a better right performance, than RAID 5 and it is certainly level. So, therefore, level 1 is preferred for high update environments such as log disks and so, on whereas, level one has higher storage cost than 5 also and level 5 is preferred for applications that has low update rate and large volume of data. So, if you have very high update you go for level one rate. So, which will give you which will cost you more, but if you have a level 5, then you will be able to get a low update, but have large amount of data stored reliably with less amount of money invested into that.

(Refer Slide Time: 28:15)



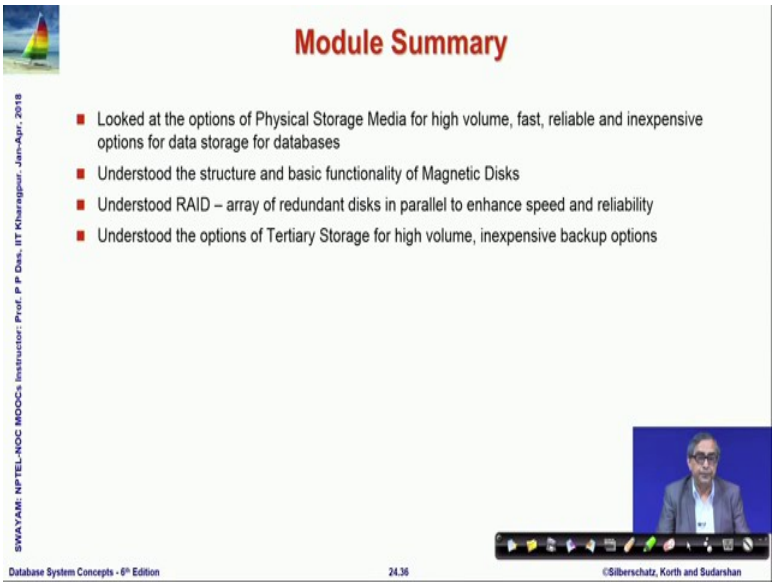
And so, these are the RAID levels then of course, finally there are different tertiary storages compact disk CD-ROM we all are familiar that DVD the record once versions where you just record and use that particularly for different kind of distribution these.

(Refer Slide Time: 28:34)



storages are used there are magnetic tapes which are very large volume and provide a high transfer rate and they are go currently they go into different couple of orders of terabytes even petabytes in size, but the tapes are not really very expensive. So, we can use them to hold really really large databases they are good for Backups and so, on, but the tape drives are quite expensive. So, that will have to keep in mind.

(Refer Slide Time: 29:08)



So, to summarize we have looked at different physical storage media. So, this was I mean besides all the details in the discussion the key take way point for us here is to understand that primarily.

We have a memory which is expensive and which is small in size very high speed. So, all operations that we need that need to be done we will finally, have to happen when the once the data is in memory and on the other side we have all the persistent data in a in some kind of a magnetic disk in certain structure and that is that can support large storage it is persistent it is reliable, but it is relatively slow to access and so, it needs to be used in a intelligent manner and one point that you have specifically noted that there is some unit for every disk system.

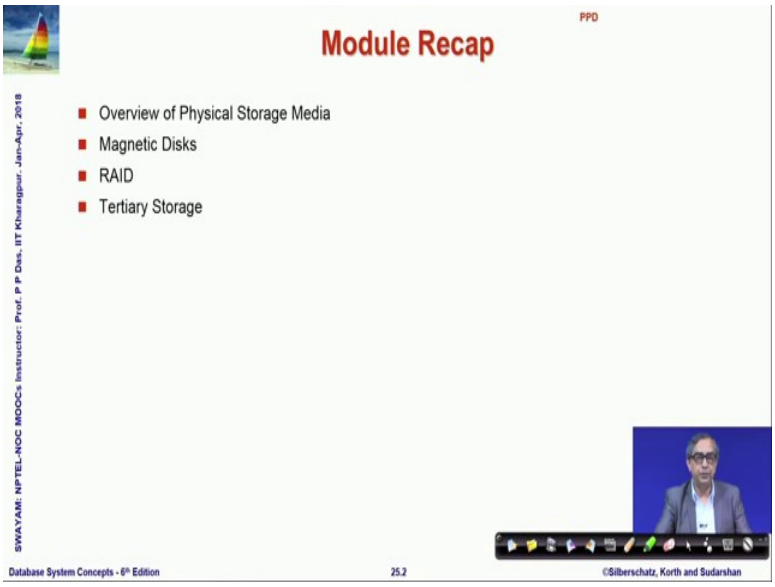
There is some unit called a block or disk block, which is a basic unit of data that will be transferred every time you access the disk. So, you are if your design is aligned with a size of the disk block which is couple of kilobytes then it will be easier to be able to design more optimal physical storage for your files and you can speed up the whole process of search and update that you want to do in the database.

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

Lecture – 25
Storage and File Structure : File Structure

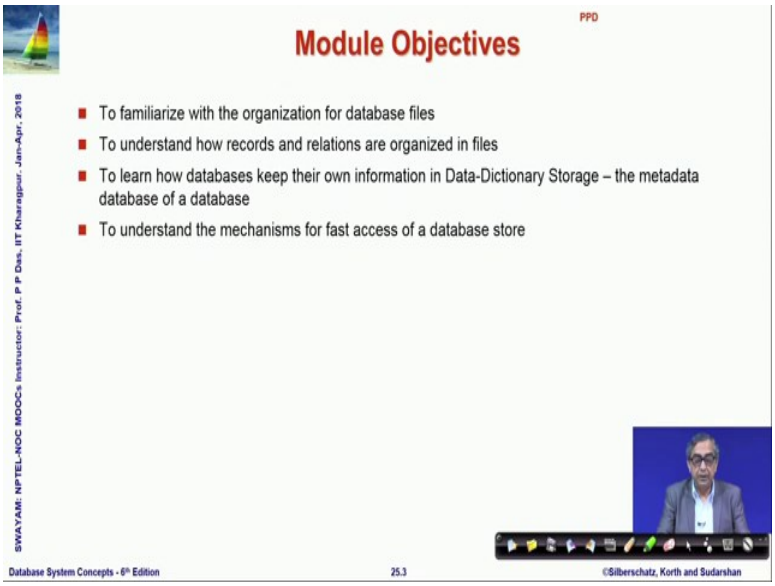
Welcome to module 25 of Database Management Systems. We have been discussing about storage and file structure.

(Refer Slide Time: 00:22)



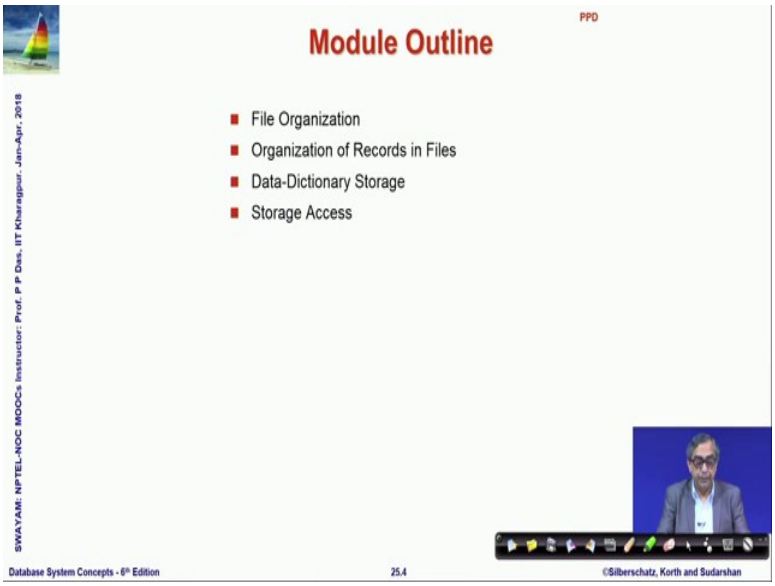
In the last module we have talked about different storage options.

(Refer Slide Time: 00:27)



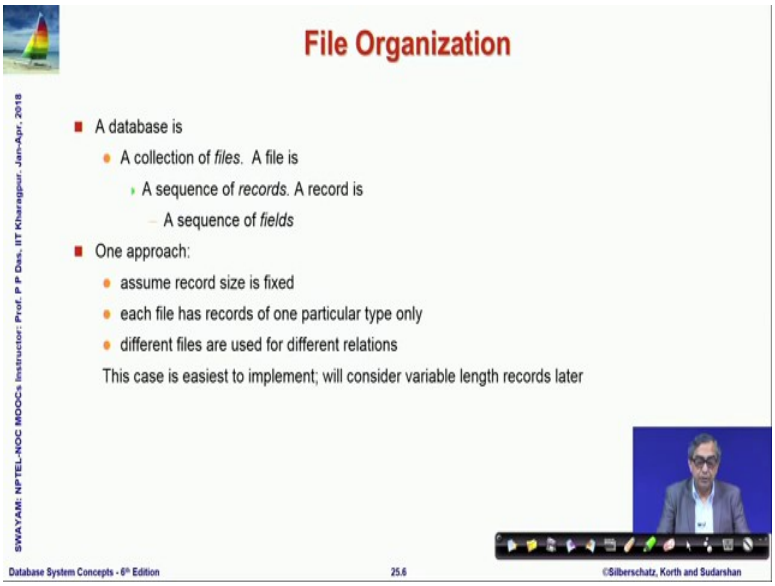
And in this one we will talk about the; organization of database files, what should be the typical structure to store the records in the files. And how the overall database which manage itself we will talk about those issues.

(Refer Slide Time: 00:41)



So, the file organization; so if you look at a database; what is the database? It is a collection of relations.

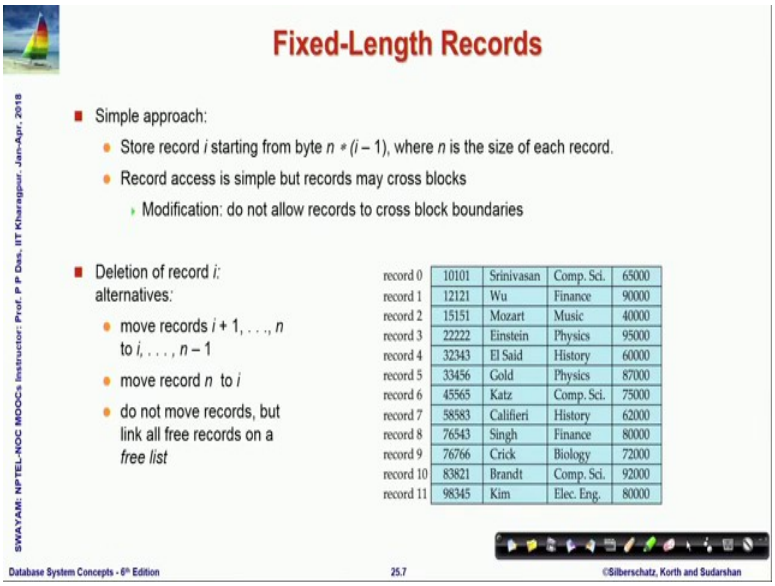
(Refer Slide Time: 00:43)



So, it is a collection of files every relation is a file. Now, what is a file? A file is a sequence of records, and what is a record? It is a sequence of fields. So, this is the hierarchy that exists and this will have to be kept in mind, when we design the organization of how we keep this data.

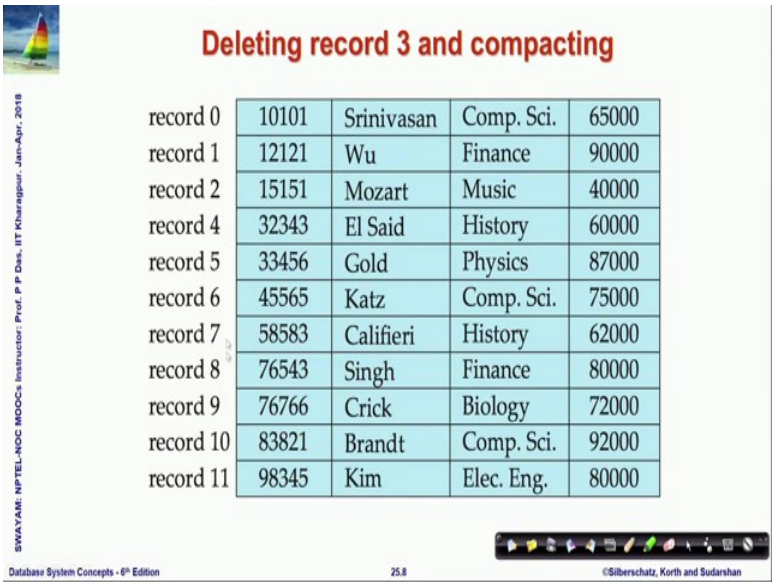
Now, one starting approach could be we can assume that all records are of fixed size which makes a life easier and each file has records of only one type again a simplifying assumption and different files are used for different relations. So, this is a easiest case to implement.

(Refer Slide Time: 01:30)



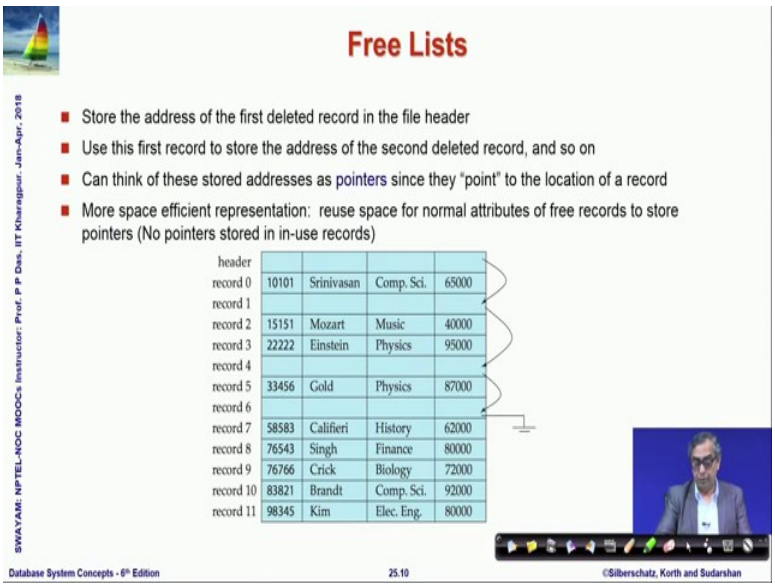
So, we will start with that; so this is what we have we store these are fixed size records. So, we store them one after the other and based on the fixed size, we can easily know what is the starting address of any record and we can access it accordingly. Now, if a if a record is deleted, then there are several things that I can do see that this is a different alternatives, that is if I record delete record I then. So, if we delete any record then we can actually move the records. So, that we consume that space or we can take the last record and move it there or we can simply do not do any move, but use an some additional pointers to denote that these records have become free rather give it to a free list.

(Refer Slide Time: 02:22)



So, these are the three main strategies. So, here we showing the first one the record three has been removed. So, all records have moved up in this it is we have move the last record 11 in the place of record 3. So, record 3 is gone, but still the whole thing remains compact only the point that must be noted that in the earlier one, where well we moved everything then the ordering that existed here of this key of this key field is maintained, but if we move the last record the naturally that ordering has got destroyed. So, it will have implications in terms of indexed organization that will cover in the next modules.

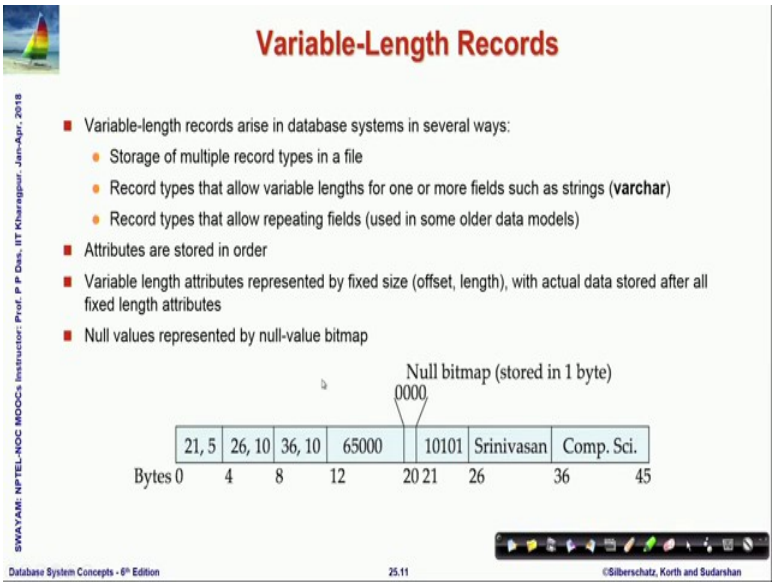
(Refer Slide Time: 03:08)



The third option could be use a free list, which is a nice one because you would you do not neither here neither you destroy the order that existed and no one have to really move records which is expensive, but you just start with a pointer and keep on pointing to the empty records.

And once you delete it you use that space itself to point to the next deleted record. So, whenever you have to you know delete a record all that you need to do is adjust this pointer. So, which is pretty fast and quite efficient way of getting this linked together in terms of; so there is as such no space over it and it is a fastest possible that you can do now in contest to fix length record.

(Refer Slide Time: 03:54)

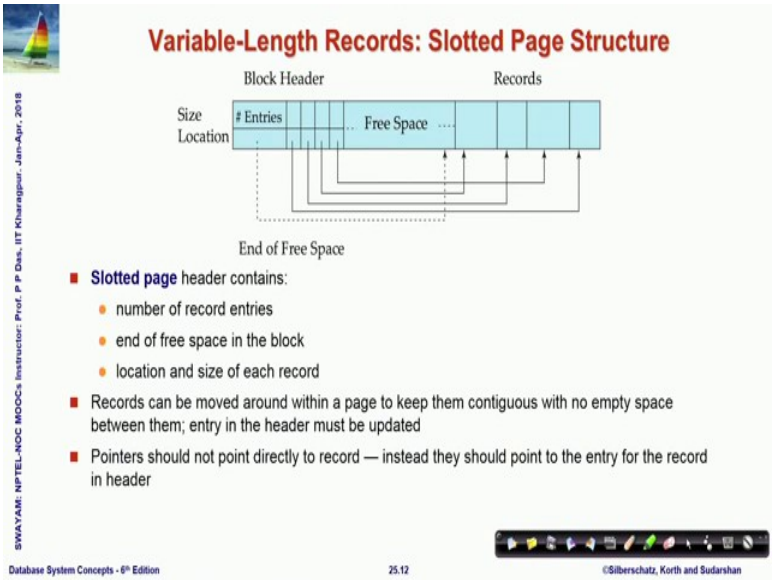


If the record becomes variable length, then certainly every record may of very different size and it is very common for example, we have types like varchar a lot of strings are varchar. So, we just we do not know how much it will take. So, the typical way you represent that is a you represent as to; what is a starting pointer of a particular of the actual value and the size of the value the number of bytes it will take. So, when we say **21,5** which we mean that this field will actually start from location 21 and we will have 5 locations 5 bytes, then the next one is **26, 10**.

So, this will start to 26 and go for 10 such; so what happens is; if you look into this part of the data, then that part is actually for all practical purposes the fix length 1, because here you are just keeping double x for the variable length data or you have some field

which is a fixed length data anyway or you have a null which is stored in one byte, and then you have all the variable stuff at one end. So, you can actually make part of this fixed length by using this kind of encoding. So, this is what is explained here.

(Refer Slide Time: 05:22)

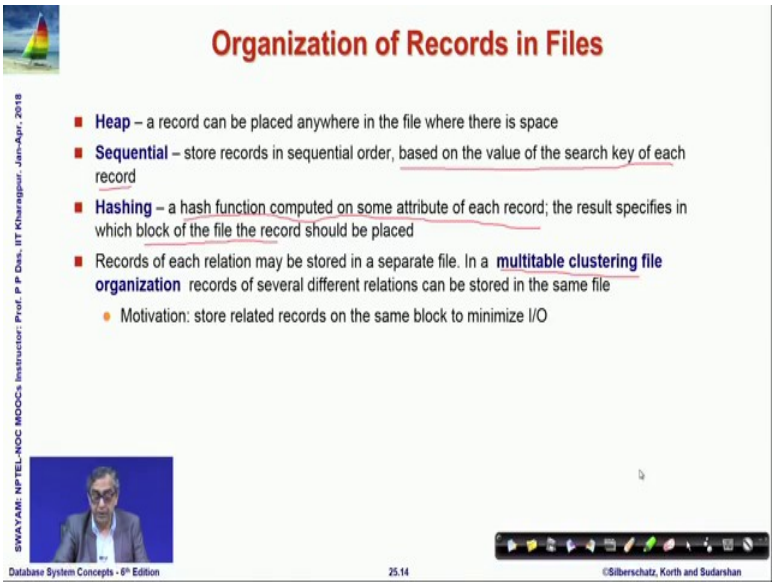


So, for variable length records a one main issue is if you if you keep it like this, then since you are using actually you are using pointers here we saying that this data actually is on 21. So, what will happen is if you change the position of the record if you relocate the record, then all these references will have to be updated. So, that becomes a slotted thing. So, what the slotted page structure does is it does a [li/little] little bit of adjustment it ports a puts a records here at the at the end.

And it has a header it has a. So, it has a block header as in here and the block header has actually pointers to the records and then you have a an entry which points to the end of the free space where more records can still be stored. So, when you refer to a particular record you do not actually refer here. So, you do not refer here, but you refer here. So, what you maintain is the header is actually not changed, but if there are relocations required adjustments required, then that will be done with respect to this. And so, this value will change, but any references made to this location will remain invariant. So, that is the basic idea of the slotted page structure, which can allow you to have the variable length record with easy re locatability in the design.

Now, let us see the given this what is the organization of the records in the file.

(Refer Slide Time: 07:05)

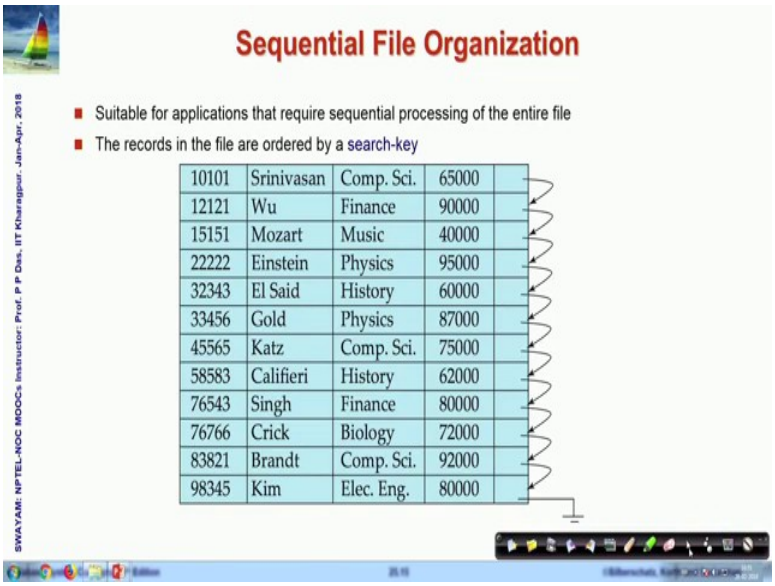


So, there are different organizations that have been tried out the simplest is the heap is a record can be placed anywhere in the file where there is space. And you can link to that that is that is one way certainly there is nothing very smart in terms of doing that, but you can you would possibly like to do better than that. So, one is you can store the; records in a sequential manner let us store records in a sequential order in terms of certain search key.

So, based on the value of the search key you put them in the sequential orders. So, what it will mean that it will become easier to search the records in that way, but it has consequences or you can hash you can use a hash function on a some of the attributes of the record and the results specified on which block which disk block the record will be placed. So, these are the different option and a records of which relation may be stored in a separate file that is a basic convention, but in some cases there could be multi table clustering as well.

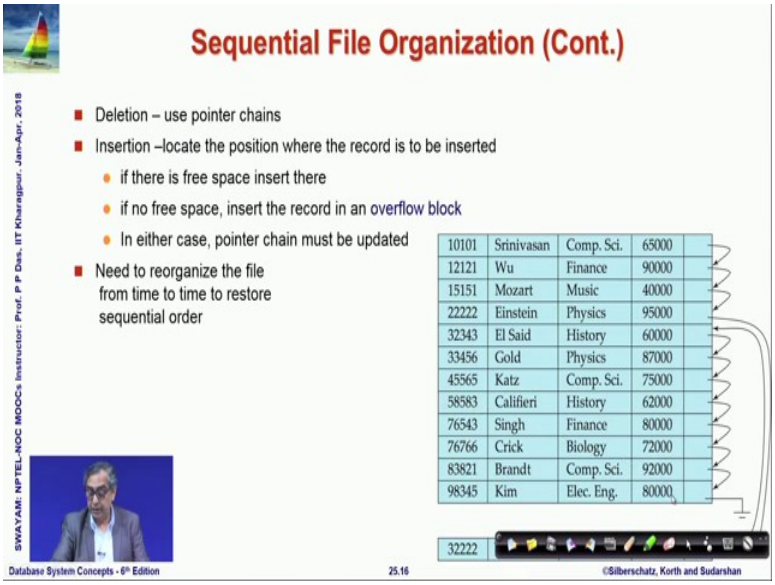
So, let us quickly take a look at these options.

(Refer Slide Time: 08:16)



So, these sequential file organizations. So, these things are kept sequentially here as you can see there all consequentially here and this is the link key of those.

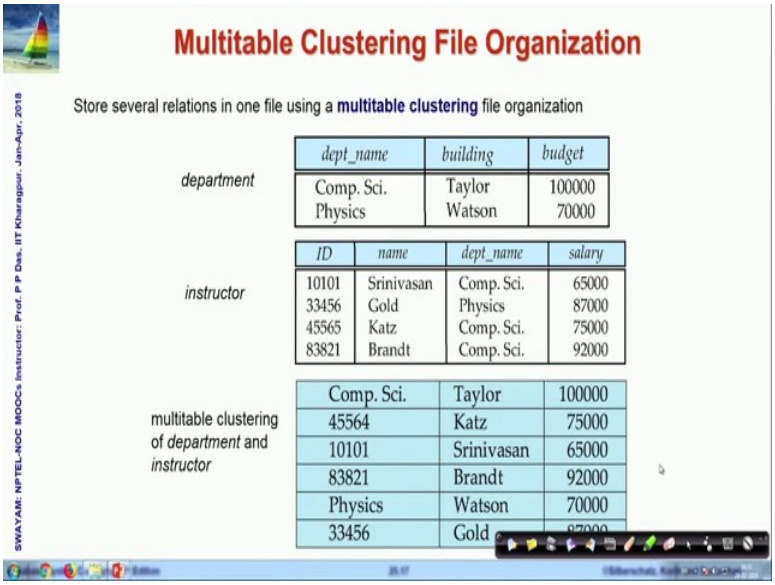
(Refer Slide Time: 08:34)



So, this is the issues of deletion and if you delete you use pointer chains. As you have we have discussed earlier, and if you have to insert then you look for a free space, if you find a free space you can put it there you insert it there if there is no free space then you have to use a overflow block, where you can go and place that separately as the dilemma

shows here; in a multi table clustering what you would do is more than one relation could be kept in the same file.

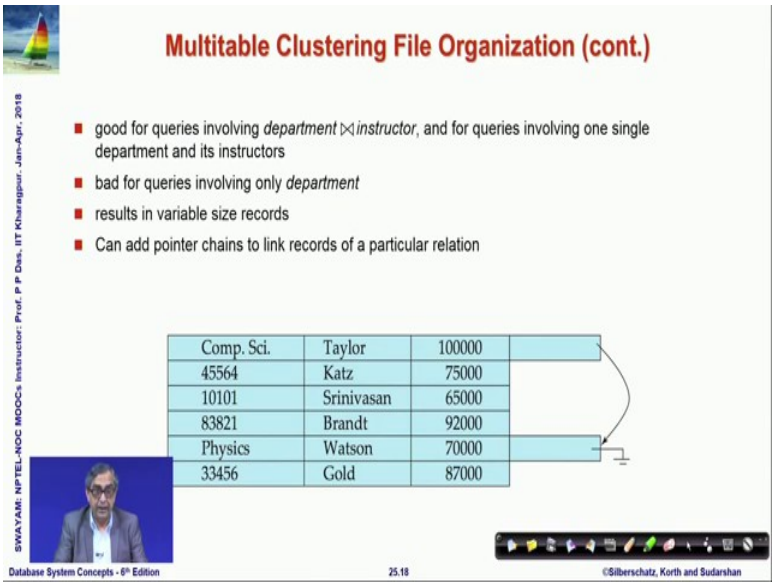
(Refer Slide Time: 09:00)



For example, here we are showing two relations department name building and budget these attributes doing department and instructor, id name, department name, and salary is other instructor in a way keeping them together here naturally, where we keep them together.

For example here in we have one which is here in we have one, which is and entry of record from the department relation. Similarly here is another which is from the department relation whereas, these are entries from the instructor relation; please note that since we are doing it multi table with a department we do not need to keep these information in as a part of the record, but what you mean is if there is a computer science entry here. Then all those records which follow this computer science entry are actually instructors in the computer science till I actually come across another departments entry where which will be followed by instructors for that department. So, that is a basic multi table convention that is to be followed here.

(Refer Slide Time: 10:26)

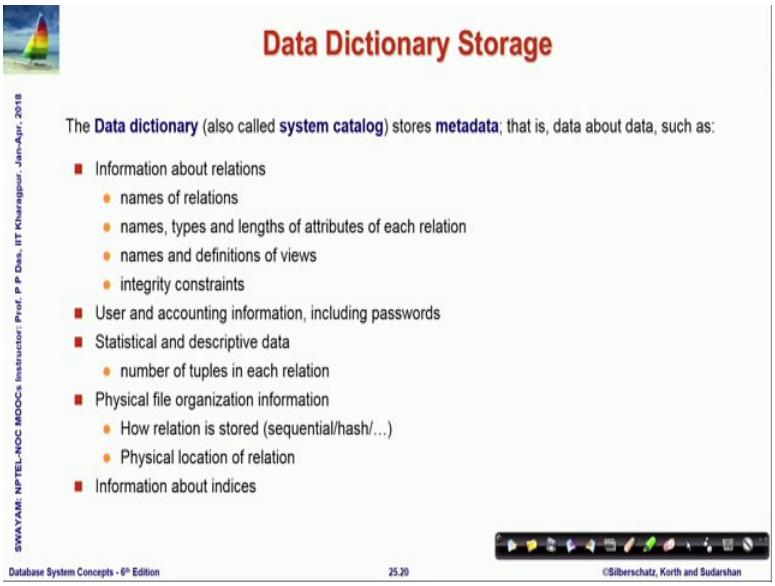


Now, it is actually good for queries that involved joining department with instructor, because based on the value of the department you have the instructors club together and they could be very easily quickly taken together and it is also good for single queries with departments and it is instructors, because as you can see you can if you want to know for example, who are the faculty for at computer science department; then it be very easy to answer that, because you need to search for computer science and then you know all the list of the faculty will be in consecutive block.

So, you can easily lift that, but certainly this is not true, if you want to involve queries which have department only; because that department information are all now sparsely distributed. So, if your query has the department based information to be to be accumulated, and then this may not be a good option. So, that will result in then you can have supporting pointer chains to actually link the department information. So, this is a one kind of a design that you have ok.

Now think about; so the whole so, we have; so, far talked about the relations and relations going to either single files or multi table relations; multi table file where multiple relations are on the same file. Now, if you look at the database as a whole. So, what is a data base?

(Refer Slide Time: 12:08)



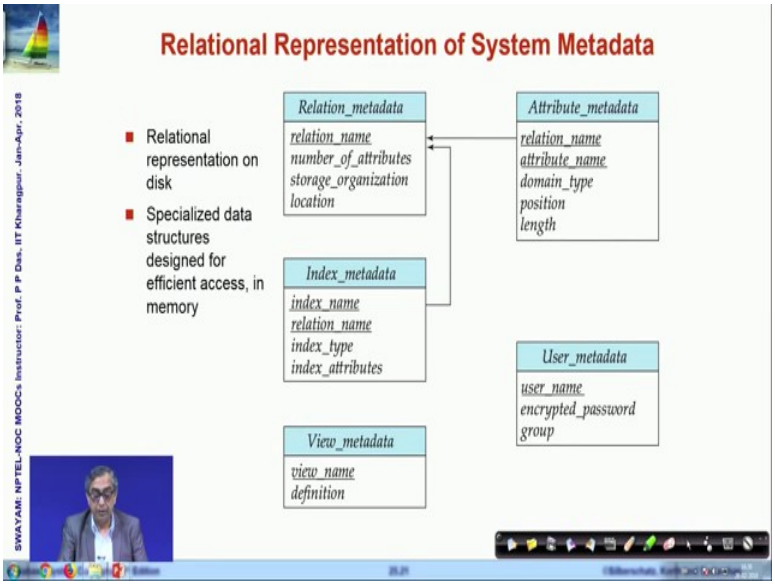
The database as a whole has a whole lot of tables; and so far we have just been focus on the fact that tables we know the tables we know their attributes and the data resides in inside, but if you think in terms of the database, then somebody; somewhere we will need to remember that what are my tables? What are my relations? For a given relation I need to know what are the attributes that the relation has, what is the; you know length type of this attributes I did remember what are the views that I have created on the database the constraints that exist.

So, all of these information which you can say is databases own metadata information needs to be also maintained; and what is done is that; also is maintained as a database within the database system. And such a metadata system is usually known as the name of data dictionary or system catalogues.

So, it has informations like this. So, you put them again, you create the schema design based on the all this metadata information that you need, also you can have you will need to maintain information for users, accounts, passwords and so, on. Then you may have statistical information, where you would like to; we will see the use of statistical information when we talk about indexing in the following week you will see that you need to know, what is the you know how frequently the different queries are fired, what are the number of peoples in each relation and so, on; you may also need to have

information in terms of what has been your choices in terms of the physical locations of file the storage options and so on the index files.

(Refer Slide Time: 14:05)

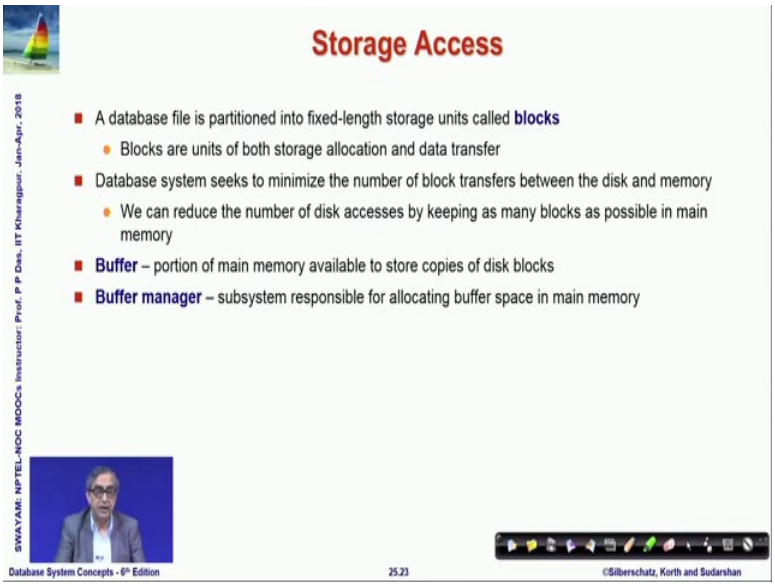


So, here is a sample one. So, what if you look into; so you can again see a number of schema. So, this is saying that the; this is the relational metadata schema which is talking about, what is the different relations? So, every record here is not keeping the data of your application, but its keeping the information that here you have these different relations. For example, couple of modules back we are talking about the library information system.

So, in the library information system we had different we designed different relations the book issue, the book catalogue, the book copies and so, on. So, those relation names and the how many attributes they have? How will you organize the storage, where is that storage will go to this particular table? Then depending on the kind of index that you are creating we have still not discussed about index we will do subsequently; but those index information can be preserved the view information we can have attribute metadata.

So, it is for the relation name what is attribute name and what is the type of that attributes. So, if the relation name is say book catalogue, then the attribute name is title, then what is the domain type. So, we will say this is a varchar; then the position of that attribute, the length if it is given the user metadata all of this are typical things that will go into this system catalogue or the data dictionary that we will require.

(Refer Slide Time: 15:47)



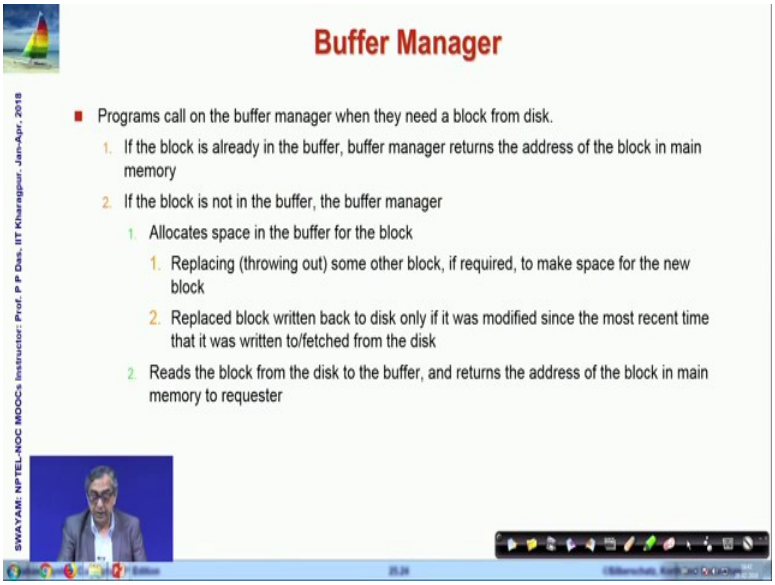
So, finally, the access to the storage the database file as I have been repeatedly saying is partition into fix length units called blocks, because blocks are defined; so that they can be easily allocated and transfer and they are the fastest unit of data that can be transferred between the disk and the memory.

So, unlike many of our typical algorithmic considerations; so when we talk about different algorithms, what we try to minimize? We try to minimize certain expensive operations in the CPU; say the comparison operation or the assignment or may be the memory read operation. But in terms of database systems block is a basic unit of data transfer and the data transfer to and from the disk is the most time taking factor much takes much larger time compare to any in memory operation that we do. So, this kind of becomes the primary unit of cost that we want to minimize.

So, normally we will see that as we talk about index saying and other different kinds of mechanisms, our primary target is to minimize the number of block transfers. So, certainly we can do that by; can reduce the number of disk access by keeping as many blocks as possible in the main memory. So, we can how can you minimize that transfer, if we can keep more of the blocks in our main memory and naturally of course, there is a limitation because main memory is much smaller. So, often we make use of different buffers.

So, a portion of the main memory will be kept to store copies of the disk block. So, every time you need a block you may not want to need to bring it from the; disk storage. So, you keep it in the buffer in main memory, and then you have a management strategy to manage this buffer. So, whenever you want to actually access a record which should be in a particular block; you check whether that block is already available in the buffer; if it is available in the buffer it use that if it is not available in the buffer, then you take it from the disk you will need quite a bit of cycles for that and as you get that from the disk then you keep a copy in the buffer so that it can be used in future.

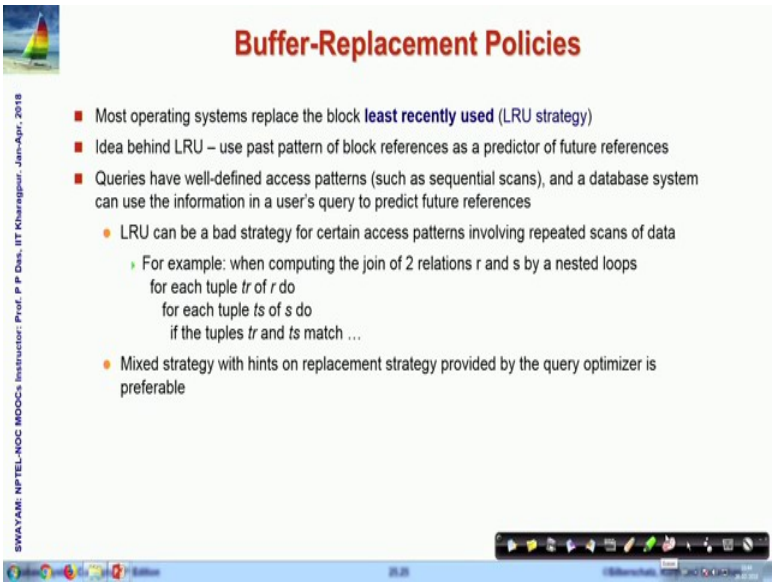
(Refer Slide Time: 18:33)



Now as you keep on doing that, naturally soon you will run out of the buffer memory. So, you will come to a situation, where we need to bring a block from the disk to the memory, but the buffer does not have enough space. So, then we will have to create replace some of the blocks and create space for that. So, here is a basic buffer management strategy.

So, as I said if we if we start if the block is already there in the buffer, then that is given out if the block is not there in the buffer the buffer manager will need to allocate some space how do you allocate space by throwing out some other block which is not required or replace the; then replace the block return back to disk and if it was modified and make space free and then read from the disk and keep a copy in the buffer. So, that is a simple strategy as you can see.

(Refer Slide Time: 19:36)



Now, certainly when you have to replace the block in the buffer, then the question is which block would you replace. Now if you recall from your from similar situations in the in the operating system in terms of memory management, you have read about different strategies for doing replacement and one of the very common strategy more often used is the LRU strategy of the least recently used strategy. So, the idea of behind LRU is use the past pattern of block references as to predict the future. So, least recently used is if this is not been used in the recent past. So, it has less likely hood of being used in the future.

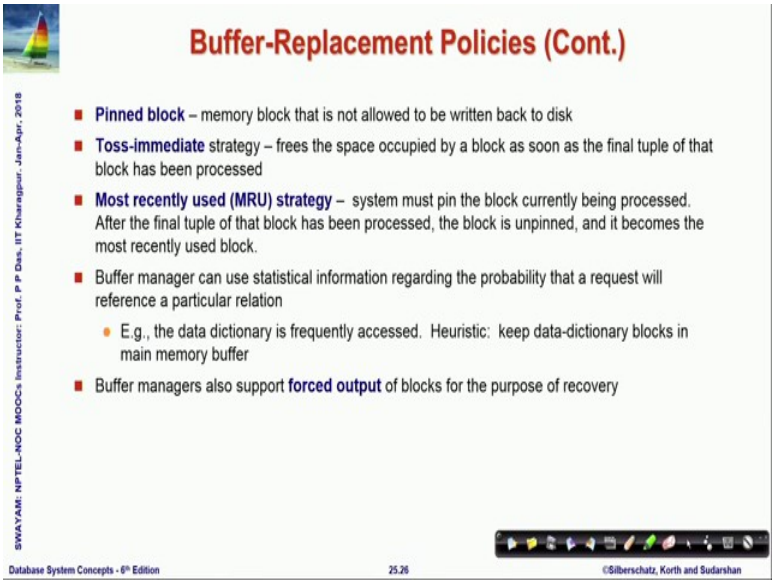
Now, to queries; now here we are trying to do the similar thing in terms of queries. So, they have a well defined access pattern and database system can make use of that and as it turns out LRU can be a bad strategy for example, often you are doing computations in terms of such what you say such nested loops.

So, you have for each tuple you do this; so you have basically trying to do a join. So, you have two relations and you are trying to do a join. So, when you do this when you are going through the inner loop, there will be lot of transfers that will happen; and the original block where you have been holding this is here and you are not accessing that for quite some time.

So, while you are doing this if you if this block, which was which was holding r at that time if that turns out to be a LRU; then you will throw it away, but with that when you

complete this loop and come back here, you will again have to read it from the disk. So, this is not LRU for such nested computations may not be a good strategy, so may be some kind of mixed strategy would work better.

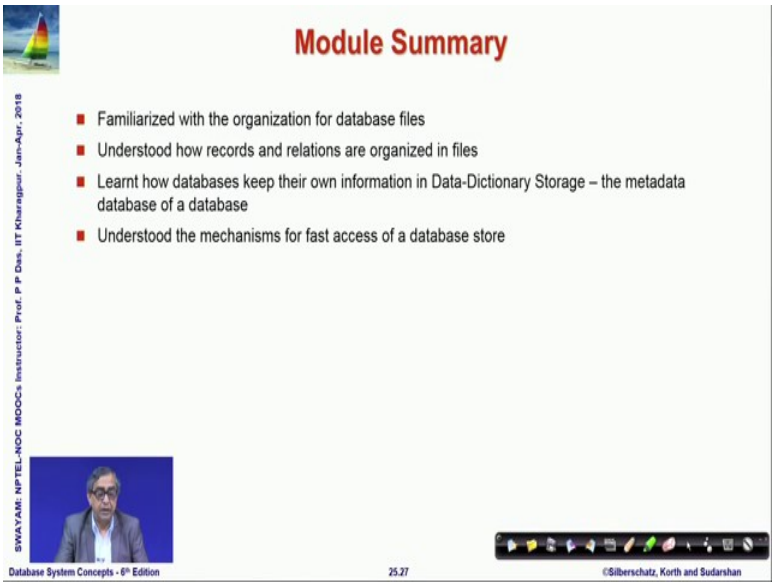
(Refer Slide Time: 21:50)



So, there are several that are used in terms of buffer replacement one is called pin block, where you mark a certain memory block which is not allow to be return back to the disk it has to stay in the buffer or a toss immediate strategy is quite often used. So, it frees the space occupied by a block; as soon as the final tuple has been removed.

So, it is a toss immediate. So, as soon as you are done you just throw it out you are you are done with it so you write it back. Another which is commonly uses most recently used the; if whenever the block is currently being processed, then the system will kind of keep a marker a pin. So, that it is not removed, but after the final tuple has been processed, the block will be unpinned and then it becomes a most recently used block and you can go with defining the most recently used block and having the strategy.

(Refer Slide Time: 23:04).



You can certainly use different kinds of statistical information and in summary; so on this we have talked about the basic organization of database files starting from the fixed record to variable record handling of the different file organization and try to take a look in terms of how records and relations are organized in terms of the in terms of the files and what are the options that we have and we took a look at the data dictionary storage the basic system catalogue, where database keeps its own information.

And then noted that block happens to be the major unit of data transfer between the disk and the main memory and therefore, that is the unit of defining unit of cost that we have to incur, and to minimize that we have a buffering strategy in the main memory, where the disk blocks will be kept a few disk blocks will be kept for quick use whenever required. And there needs to be various different smart replacements strategies for good management of this buffer.

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

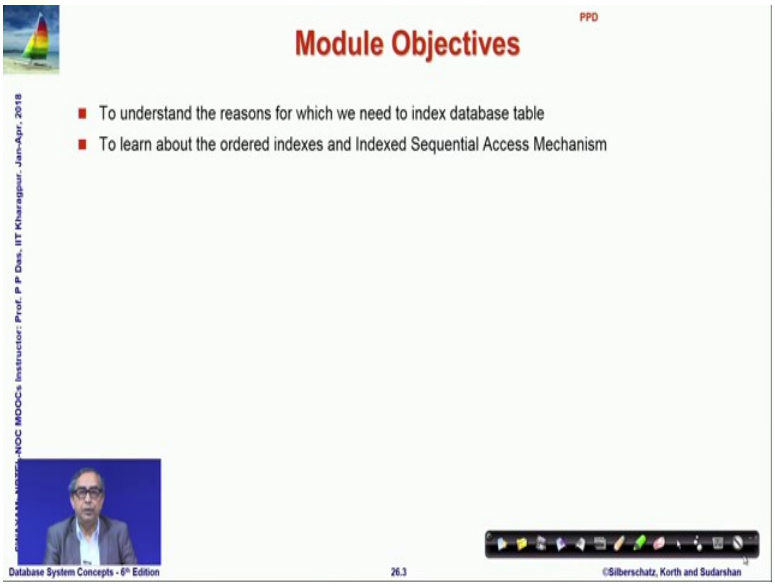
Lecture - 26
Indexing and Hashing/1: Indexing/1

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)

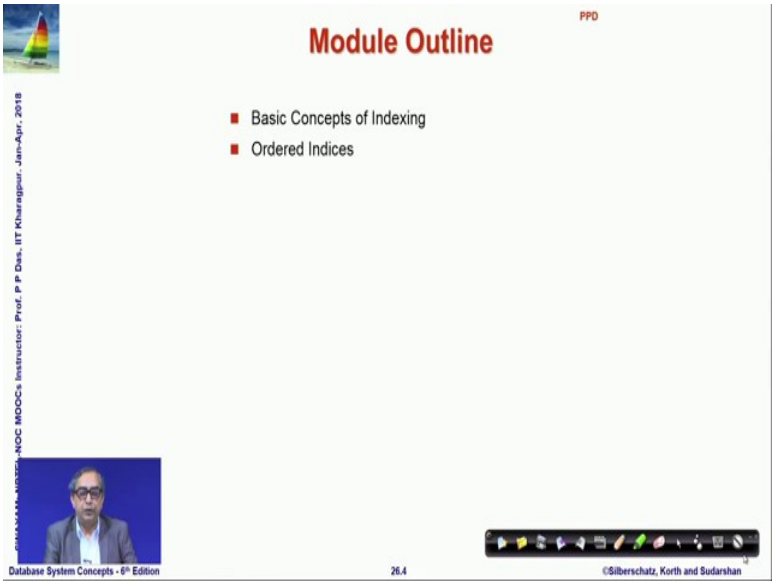
So, this is a quick recap of what we did in the last week primarily talking about application development and then specifically; we focused on storage and file structure that is how a database file can be stored how it can be organized and in a manner so that, we have efficient representation for that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

(Refer Slide Time: 01:01)



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)



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

(Refer Slide Time: 01:40)

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 /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

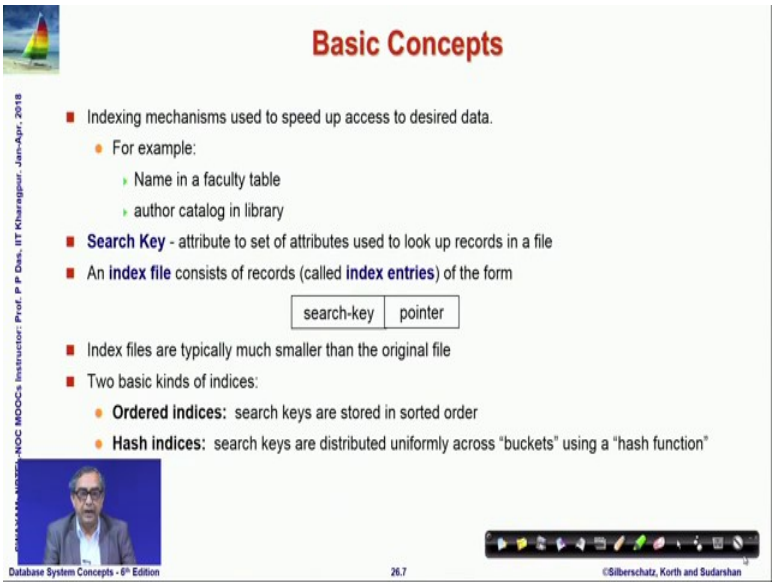
data in a sorted manner and doing some kind of a binary search that is one one possible mechanism through which you can do that.

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.

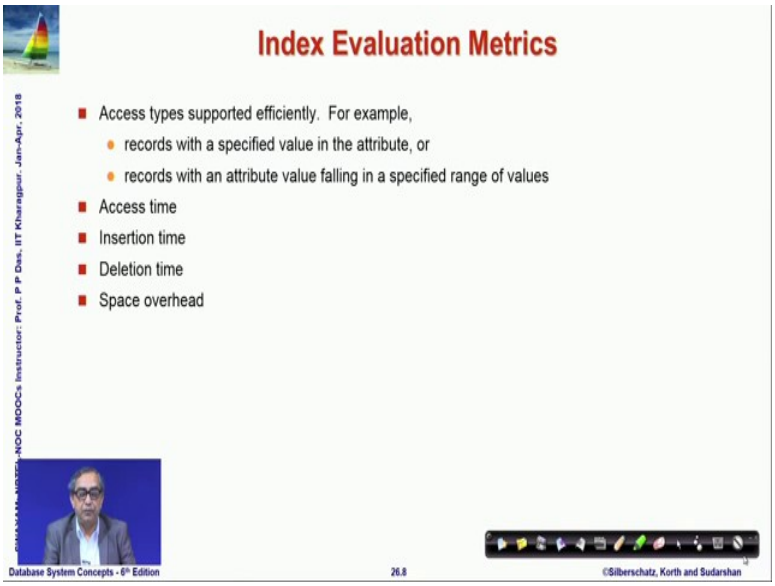
(Refer Slide Time: 05:39)



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.

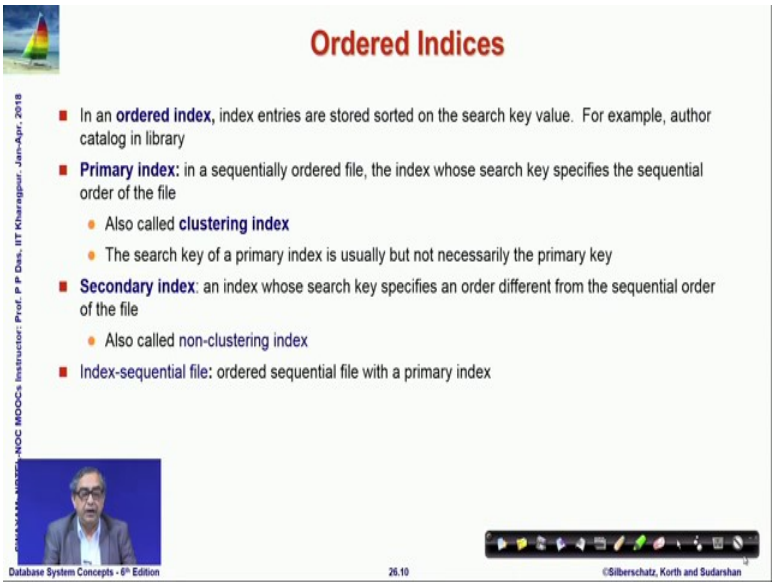
(Refer Slide Time: 06:44)



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.

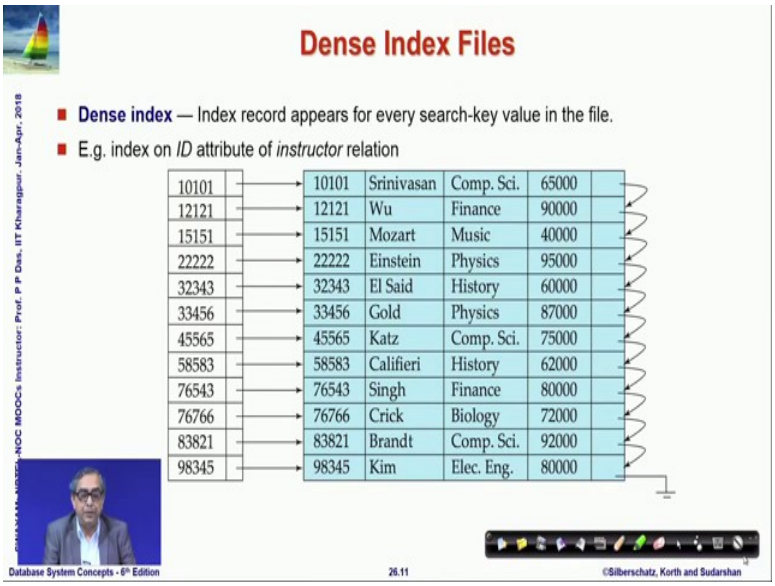
So, that the access time the insertion time the deletion time all these should get as minimized as possible and as you have just seen indexing might mean maintaining additional structures. So, that overhead of space should also be minimal. So, with that with indexing we should be able to do at least 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)



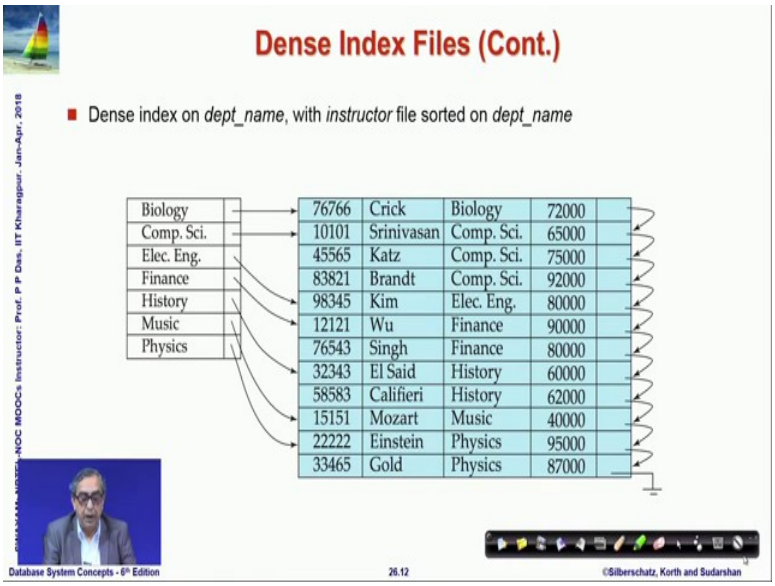
So, let us look at the first concept of ordered index which is pretty much like what we have just sent. So, in a sequentially ordered file the index whose search key specifies the sequential order is known as the primary index. So, the sequential ordering is done based on that primary index it is called 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)



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)

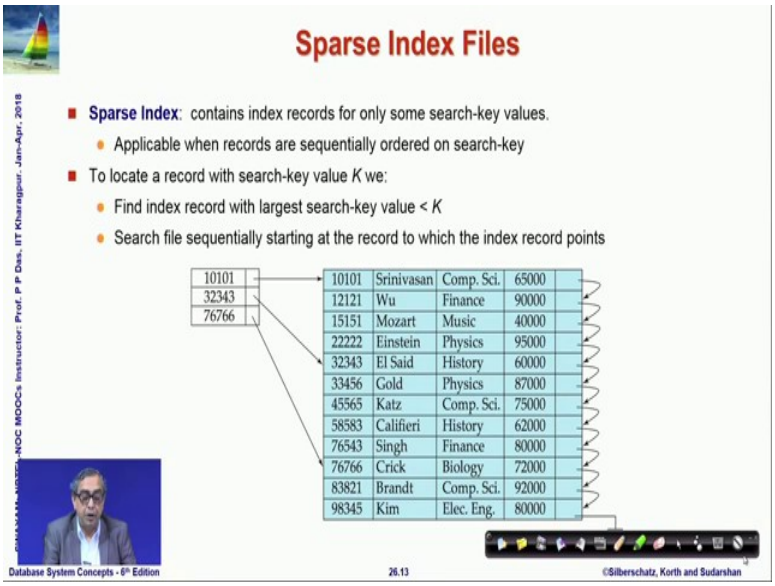


Now, instead of doing id if I want to do a dense index on department name, then naturally the sequential file has to be indexed primarily on the department name and as you can note that unlike the id which is also the primary key and therefore, every id value was unique the department name is not a primary 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.

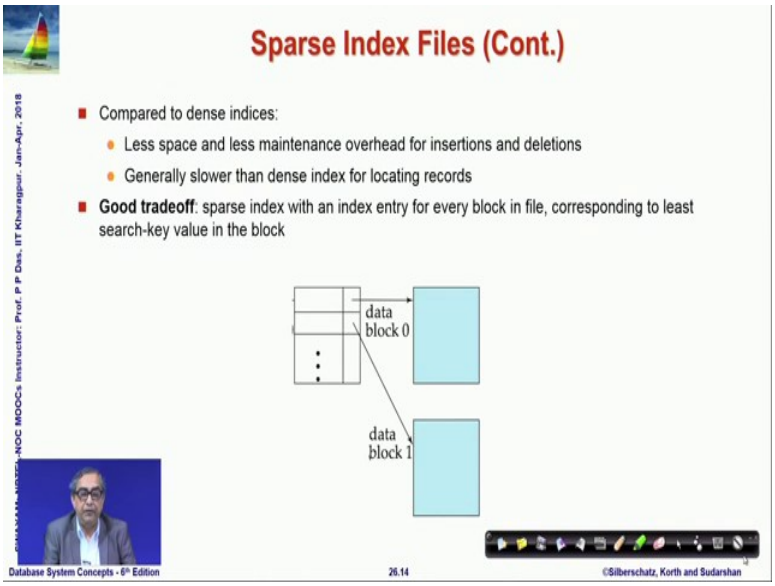
So, now you can understand why we need the link list; because if we are looking for the all those teachers all those faculty who are associated with department computer science, then I can find it on the index file with the department name, computer science traveled to the record first record in that 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)



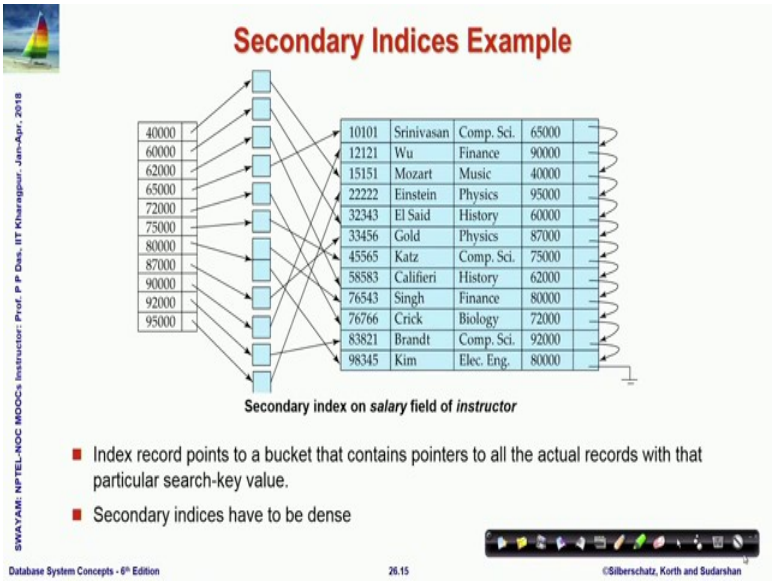
Now instead of dense index we could also do sparse index. With parse index it means that instead of maintaining all the search key values that exist in the file we just take a subset of that and we maintain those in the index file. So, here again we are showing a index sequential structure with id being 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 < K** and then search sequentially or onwards because these are all linked together in the sequential manner.

(Refer Slide Time: 12:18)



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).

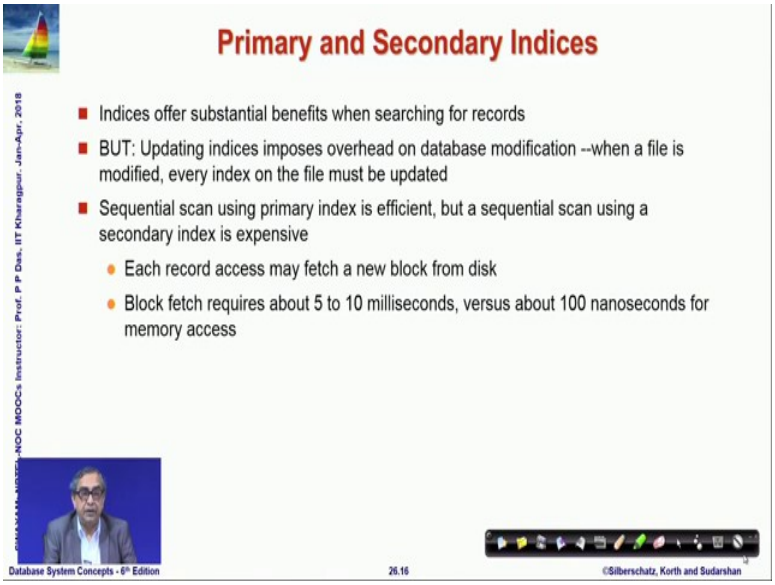


So that way it is it is better than the dense index, but certainly in the dense index I can go to any record directly from indexing in the sparse index I need to first index and then go sequentially. So, it will be in general slower in terms of performance and will need to look at this tradeoff now it 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.

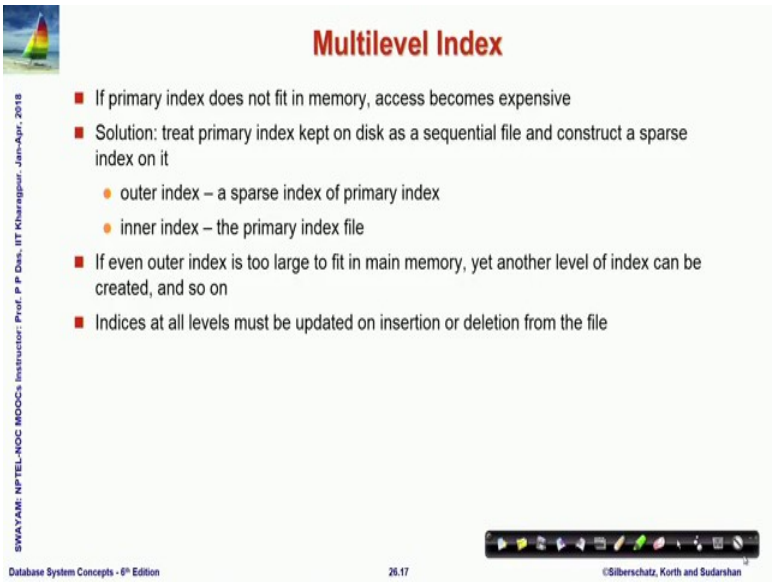
(Refer Slide Time: 14:46)



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)



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 traverse 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.

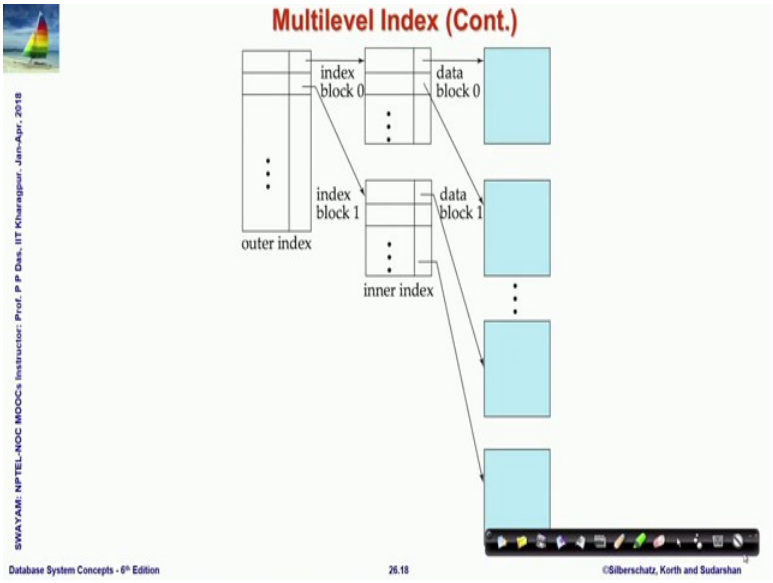
So, because if the primary index is large then that also has to exist in the disk and therefore, bringing that primary index into the memory and then searching will add to additional access cost of the disk and you have already seen in the earlier modules as. So, how these costs are expensive these accesses 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.

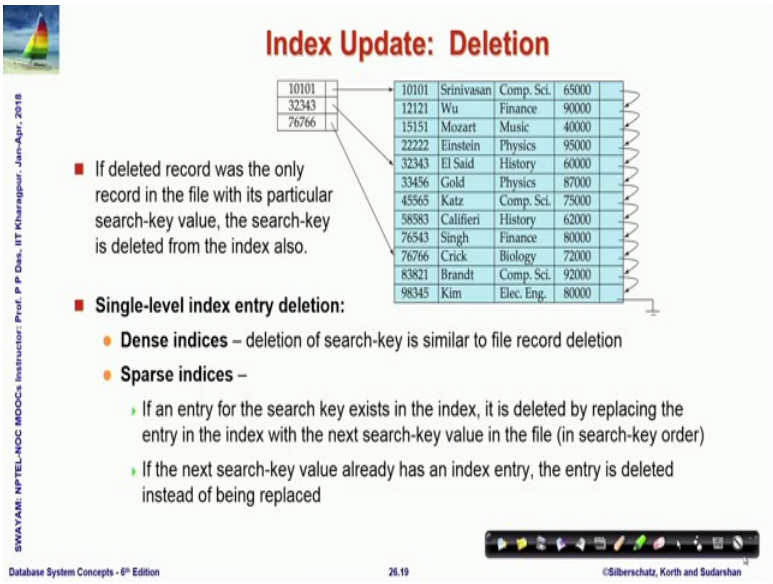
(Refer Slide Time: 17:48)



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 access and then based on that to find another access to for the block in which the record exists.

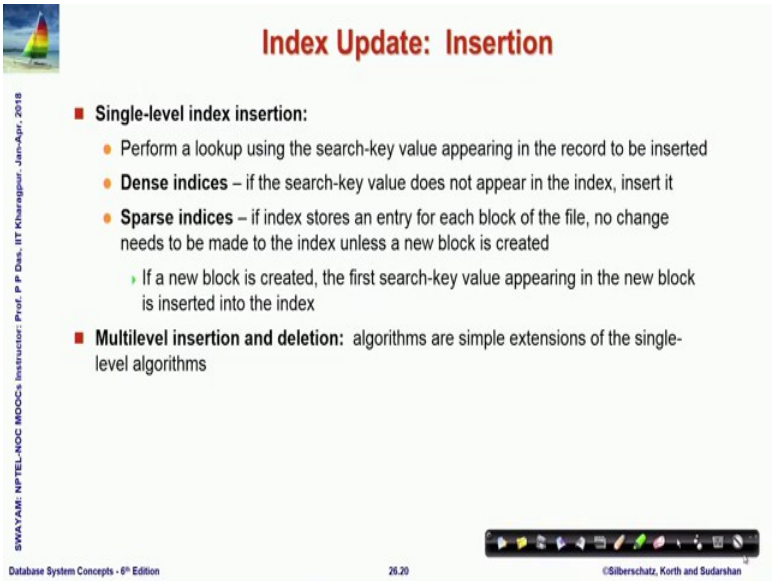
So, with this you would be able to manage with to block accesses in this case and that is how the multiple this would not have been possible if in this case you would not have done the sparse outer index, because you would have required the different parts of the inner index of the primary index to be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

(Refer Slide Time: 18:59)



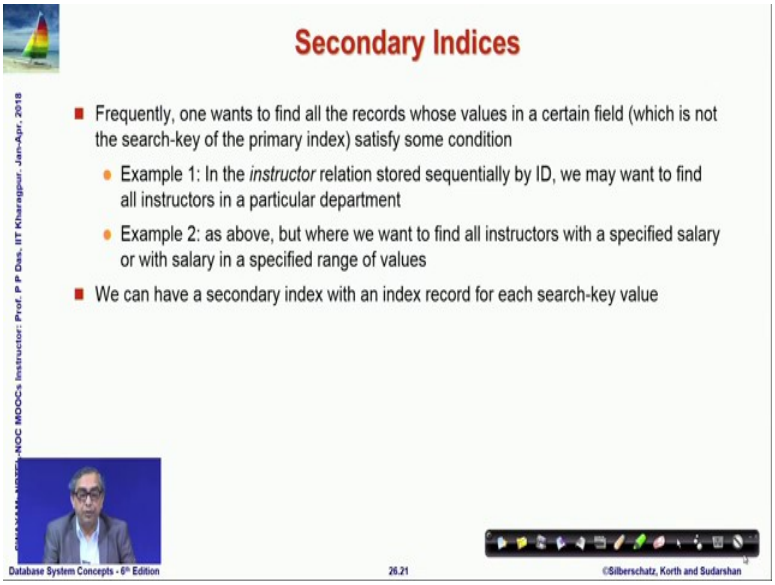
So, updating the index particularly if you do deletion then the if it is a dense index then the deletion of the search key is similar to deletion of the actual record in the file because it is dense if these parts, then naturally you will have to take care of some of the cases, because if it falls within 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)



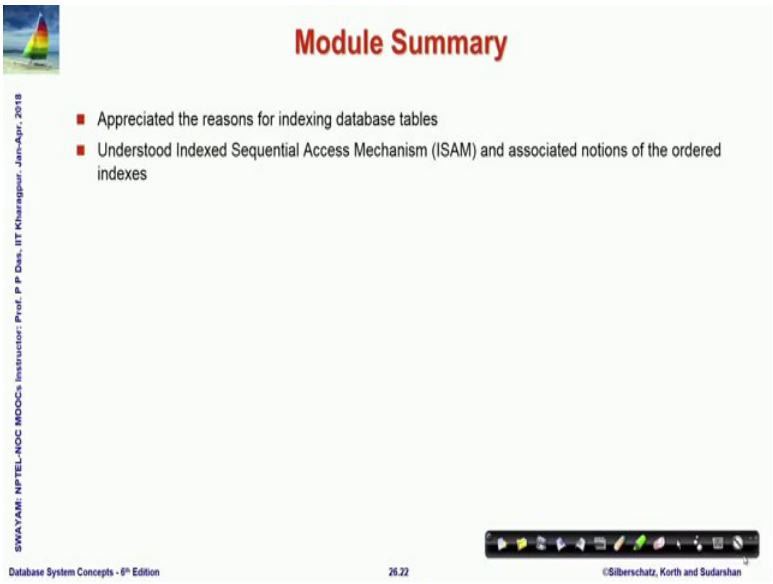
Similar thing will have to be taken care of in terms of insertion. So, first you will have to look up to find out to where the record needs to be inserted and then if it is a dense index if the search key does not appear in the index then you will have to insert it in case of sparse index you will have 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)



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)



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.

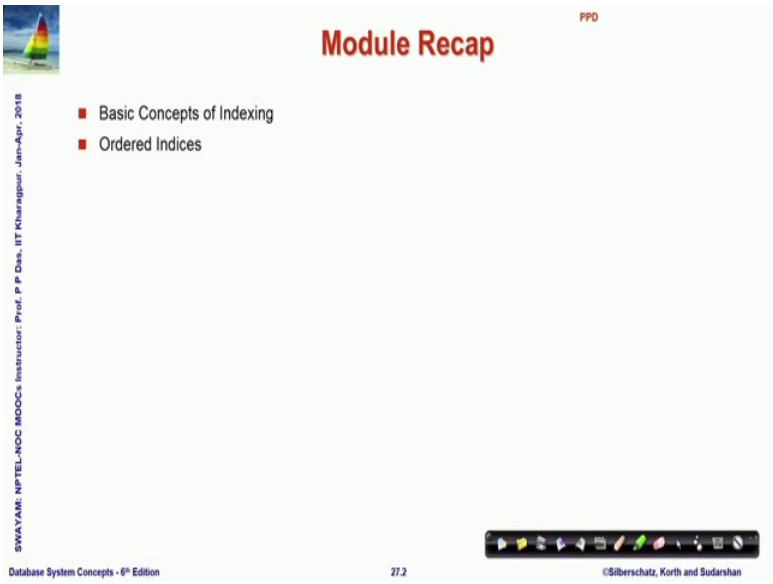
So, even though it is focused indexing is focused on improving the access time it actually improves the time for access insert delete all of that, but we will have to keep in mind that in the process there are certain overheads of index update for insert and delete that will have to be kept as a minimal 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.

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

Lecture - 28
Indexing and Hashing/2: Indexing/2

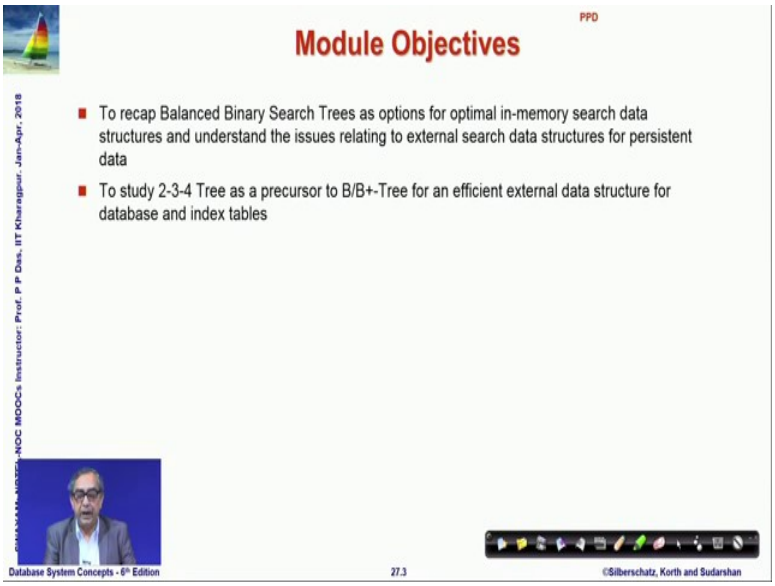
Welcome to module 27 of Database Management Systems. We are discussing indexing and hashing mechanisms in a database and this is the second in that series.

(Refer Slide Time: 00:28)



In the last module, we have discussed the basic requirement of indexing and we have learnt about ordered indexes using primary index.

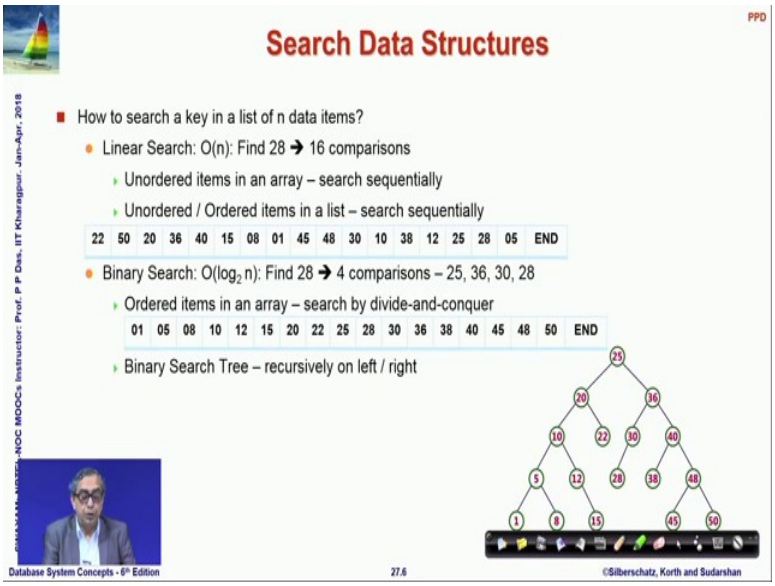
(Refer Slide Time: 00:42)



Which can be dense or parts and the multi level indexes. Now, in this module we would try to look for the basis of how indexing the index file structure can be very efficiently represent it. So, we will do a quick recap of our notions in algorithms goes.

Where we have talked about earlier balanced binary search trees I mean not in this particular course delivery, but I expect that you have gone through algorithms course where you have learnt about balanced binary search trees as options for optimal in memory search data structure and from that will try to understand the issues relating to external search data structures for persistent data and very specifically we will study 2-3-4 tree as a precursor to B/B+ tree which is an very efficient external data structure for databases and index tables. So, these are the two topics to cover.

(Refer Slide Time: 01:47)



So, first let me quickly take you around with search data structure now I should warn you that here I am looking at we are looking at a little different kind of a problem here we are looking at search as it is performed in the algorithms course which mean that when you talk about data structures in algorithms course.

You typically talk of data structures which have two basic properties one they are volatile data structures transient that is they are created when the program starts and you operate on the data structure find queries and then as soon as your program ends they disappeared. So, they are not persistent data in contrast when you are dealing with database we are dealing with persistent data which stays even when no queries being performed.

And the second which is the consequence of the first is the data structure that you are study in algorithms work in memory. So, they work in a small limited space and they could be volatile whereas, the database the data structure required for databases has to reside in the disk. So, we have seen the tradeoff between the; on the storage hierarchy between memory and disk and other layers.

So, they will be brought to memory and then certain operations done and then possibly written back and so, on. So, there is similarity in terms of the strategies, but there is a very significant difference in that that because of the persistency the data structures that are used in the database application in the persistent data application has to work with a

very different kind of cost. So, when we talk about cost of an algorithm say a search algorithm we will say that a search algorithm or a sort algorithm has a certain complexity you saw, you know the merge sort has the complexity order **n logn** and what it actually means.

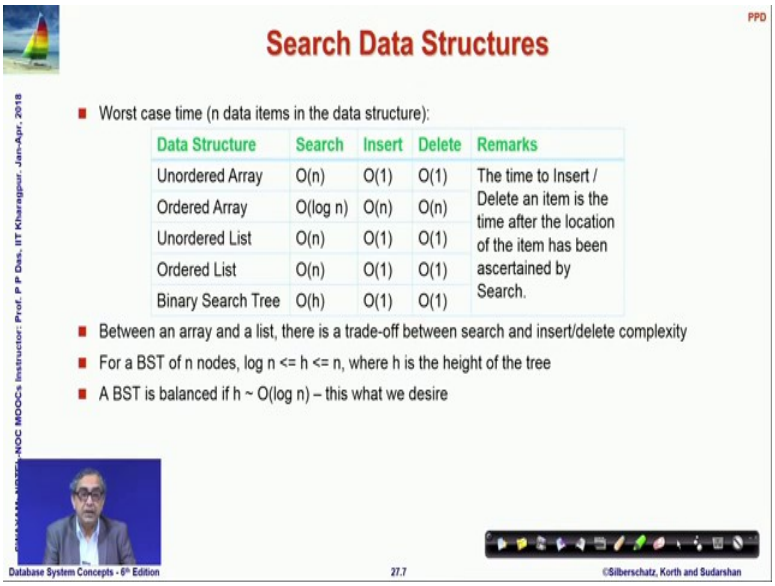
Is a number of comparisons you can estimate to do it in sorting n numbers is approximately n log n, but when we talk about external data structure or disk base data structure then your cost may often not be the operations in the CPU like comparison or addition or assignment your cost will shift to actually the number of disk accesses the page acceses that you have to do, because as you have already noted that the cost of a disk access is much larger couple of orders larger compared to the basic cost of different processor operations.

So, I will start with this in this module I will start talking about data structures which we are found to be efficient in memory and then we will migrate to seeing how they can be used in a with simple extension in the as external data structures as well. So, what we have if you have given a to search a key in a list of n data items what are the different choices I could make use of a linear search either items could be in an array ordered or unordered in an array or they could be on a list either ordered or unordered I can search that list sequentially and find the data item.

So, I have just shown an example here there is a couple of data items given in that array and trying to find 28, there are 16 comparisons that I need to do and we all know that we can do much better if we keep the this item sorted in terms of in an increasing order or say decreasing order here it is increasing order and then we can do a binary search divide and conquer and I can find the same value 28.

Now, in just four comparisons and from that we have come to the also know that this whole binary search order can be easily represented in terms of a binary tree structure called the binary search tree.

(Refer Slide Time: 05:44)



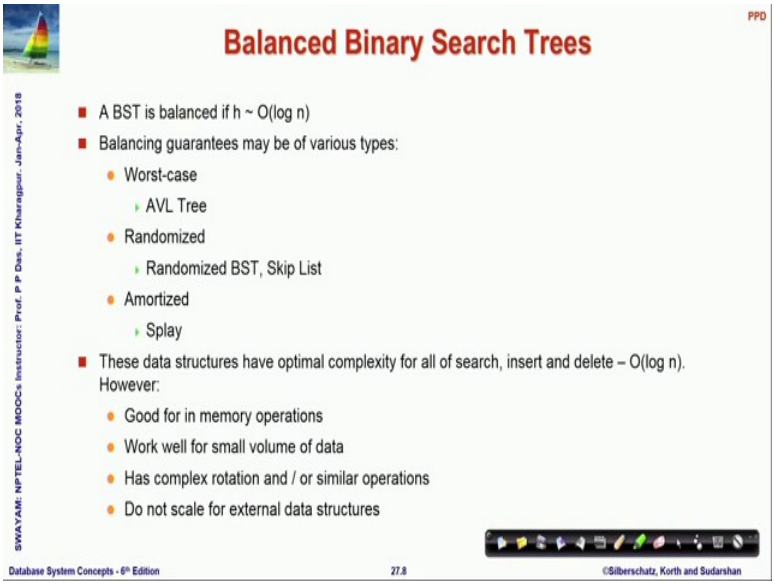
So, if we compare worst case time here again here please keep in mind that we are talking about in memory data structure. So, here the time is primarily that of comparison. So, if you look at the different data structure like unordered array ordered array unordered list and ordered list and binary search tree and if you check the complexity of the three basic operations which we will need in that in a database application the search insert and delete you can find that the search. Usually, is order n unless you have a un ordered array I mean between array and list the search is order n unless you have an ordered array and you can do a binary search.

The insert the time that I show for insert or for delete is usually order one, because when I say insert is order one what it means its that after I have been able to search an item which I need to insert after I have been able to find its position, what is the additional time that you need to actually insert that item? So, that insert cost is often for unorder array and any kind of list is order one because you can just manipulate a couple of pointers and insert that, but if you are using an ordered array to make your search efficient then to insert you need a order n insertion because at the right place you need to move the elements to the right to make the space for the new element, because you need to maintain the ordering that the elements have.

So, kind of we find that between the array and the list there is kind of a trade off in terms of if you want to make search better you have ordered array and that degrades the insert

delete complexity and vice versa. So, to take the benefit of both we the binary search tree is device to a you expect that the search will take the worst case order h cost which h is the height of the tree, because that is the maximum number of comparisons that we will need to reach that leaf node and a BST binary search tree if it is balanced then h would be of the order of log n and this is what we desire.

(Refer Slide Time: 07:58)



So, if you look at BST is balanced h is of the order of log n I am not going into the theory of proving why balancing means h is of the order of log n or how this order of log n comes if you have doubts please refer to your algorithms book you will find plenty of that now that naturally now if a in the data structure if I am regularly inserting and deleting data.

Then it is not guaranteed that it will remain balanced it might for example, if I am inserting the data in a in a say in increasing order in a in a binary search tree then every time the insertion will happen on the rightmost node and if. So, that will mean that I will have along the rightmost node I will have a long chain and therefore, it become like a linear list and therefore, any search in that will not remain optimally it will take order in time.

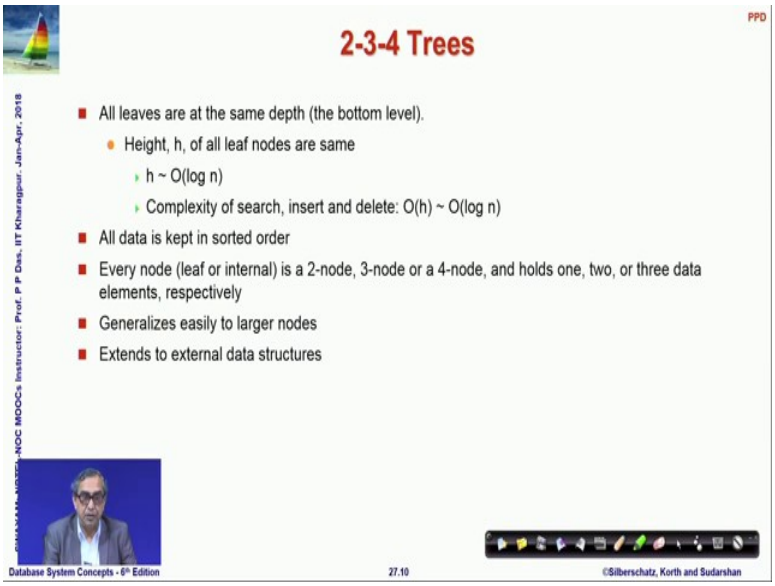
So, there as different strategies that you have studied in terms of balancing just to remind you there are strategies which give you the best possible worst case time of log n which is the famous AVL tree there are randomized strategies in terms of randomized BST skip

list there are amortized strategies which say that I do not really care about what happens with a particular insertion search or deletion, but what I care is if I have done a large number of insert delete search operations on the array then on the average what it should be balance on the average it should be of ordered log n.

So, we have seen all of these different strategies and. So, in an order in an attempt to generalize them for external data structure we note that these are typically good for in memory operations these are good worked well; when you deal with a small volume of data I mean you may be that may still be large, but is small in the sense that the whole data fits into the memory and you can manipulate muscles the whole data in memory and it of course, all these many many of these algorithms.

Particularly, the AVL tree and quite a bit of the randomized BST have complex rotation operations that need to be performed the order different random generators need to be involved in the randomized BST or skip list. So, there are other complexities in this whole factor comparison cost is not the only cost that you have and in the simple thing is they do not scale, what external data structures they are not optimized in terms of minimizing or optimizing the disc accesses or they do not scale to millions and millions of entries and so, on.

(Refer Slide Time: 10:38)



So, you need to look at a different approach and this different approach in the day in terms of database application database structures is called B + tree and what I am going

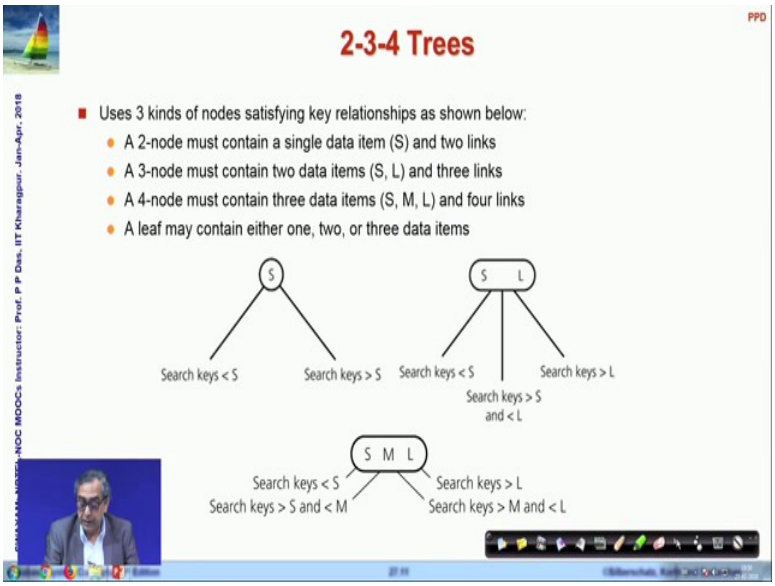
to discuss is an in memory version of that which is a 2-3,-4 trees. So, that we can understand the basic principle of such search data structure which can work with external data with this data, but first understand them in as an in memory version.

So, what is a 2-3-4 tree a 2-3-4 tree in contrast to other BSTs or in contrast to the typical BSTs where you know every operation needs the height of the tree to be balanced because some leaves could happen at a much deeper level some could be at a much shallow level in a 2-3-4 `tree all leaves always are at the same level the same depth the bottom level.

So, height h is the height of all the leaf nodes and that is guaranteed to be of order of log n, if the tree has n number of nodes the complexity of search, delete and insert all are order h that is a consequence of a constant height h or or a fixed height h that given n that is maintained all data are kept in a sorted order. So, if you do a in order traversal of the tree you will get the data in the sorted order, but the (Refer Time: 12:13) difference is in contrast to the BST where every node is a binary node is a is a (Refer Time: 12:19) one key and has two children here every node either a leaf node or an internal node every node is one of the three types it can either be a 2-node or a 3-node or a 4-node.

A 2-node holds one, data 3-node once holds 2 data and 4-node holds 3 data and there they give the name get the name 2-node 3-node and 4-node based on the number of children that they can have the 2-node can have 2 children, 3-node 3 children and 4-node 4 children. They can generalized easily to larger nodes which can have a large number of different types of nodes and it extends very naturally to external data structure. So, that is with the basic about the 2- 3- 4 tree.

(Refer Slide Time: 13:05)

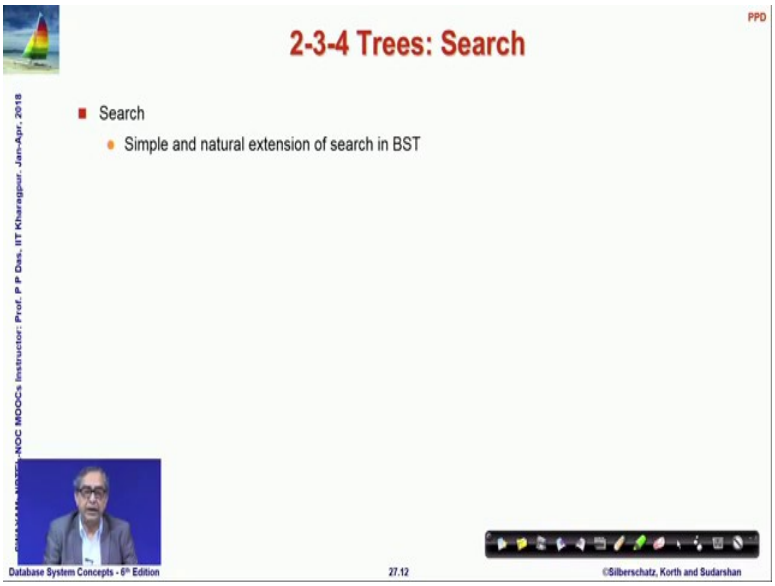


So, let us just go through it in little bit more detail it uses three kinds of node as I have said and now let me just show you. So, if you are talking about a 2-node. So, this is a 2-node. So, there is one data item S and all search keys which are less than S around this side all search keys which are on greater than S around this side.

We are just for the simplicity of discussion where I am assuming that all keys in this data structure are unique. So, there is no repeated keys the repeated keys can be handled in a very easy manner. So, this is one type of node a second type of node is a 3-node which must contain two data items S we are calling it S and L and three links the first is less than S(**Search key<S**) the last is greater than L(**Search key>L**) and the middle is greater than S, but less than L(**S<Search Key<L**) which means actually what we enforce in terms of the two keys that exist at the node S < L.

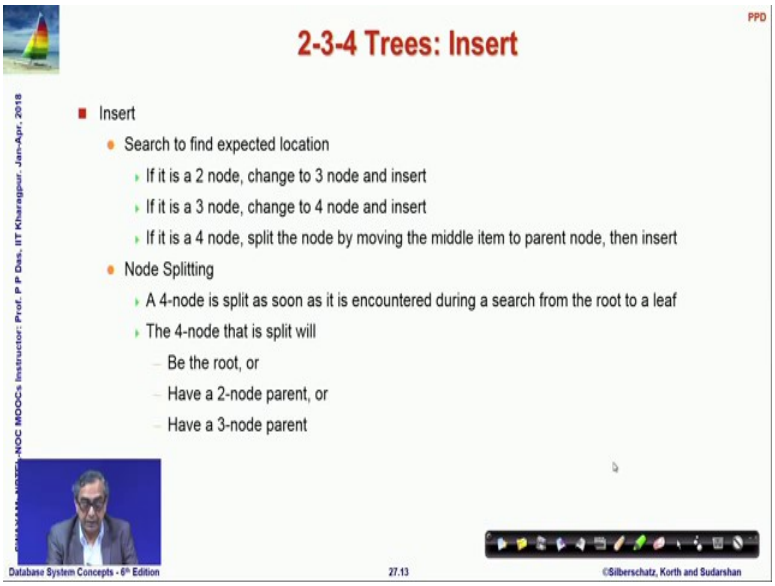
So, values less then L go on one link values between S and L go on the middle link and values > n go on the third link. Similarly, if I have a 4-node here I have three values where S < M < L. So, now, you can understand why the acronyms S, M and L small, medium and large. So, on and we have four links the first link gives you values **Search Key < S** second is **S<Search Key<M** third between **M< Search Key< L** and fourth **Search Key >L**. So, these are the or three different types of nodes that a 2-3-4 tree can support and if it is a leaf node then it can be it can contain either 1, 2 or 3 get items. So, let us go forward.

(Refer Slide Time: 15:01)



Now, to search to search is a simple extension of BST you know how to search in a BST you start with the route see whether the given key to search is greater than the key at the route if it is greater you go to right if it is less you go to left and you do the same thing here for a 2-node for a 3-node all that you will need to find out is between S and L whether it **Search Key** <L then **Search Key** < S then you take the leftmost whether it is **Search Key** >L then you take the rightmost if it is **S<Search Key<L** you take the middle similar strategy you do for 4-node and search is a simple extension of the BST algorithm. So, you can suddenly work it out.

(Refer Slide Time: 15:38)

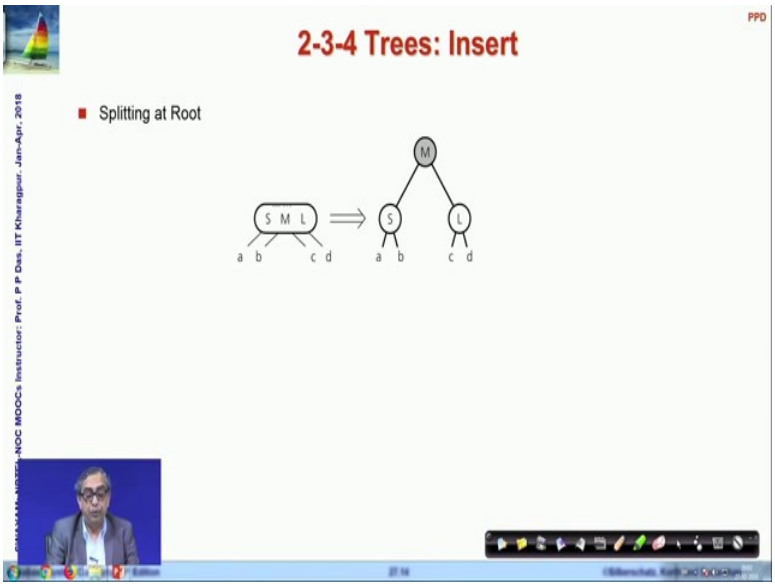


Now, what we will need to do in terms of an insert, insert is very interesting. So, for insert first you search and find the expected location where you are expecting it. So, that is not there. So, you will expect. So, now, what are the possibility? Possibilities where you have found the expected location that node could be a 2-node if it is a 2-node all that you simply need to do is change that where 3-node inserts the second item if it is a 2-node it has only one item. So, just insert this new item there and making it into a 3-node good.

In the second case if you find the location if you find that it is a 3-node. So, it has two items you just change it to a 4-node and insert this is as a third item; obviously, when you insert you will have to decide has to whether where you should insert that depends on whether you are given key is greater than the key that already existed or smaller than that and so, on. Now, the question is what happens; when if you locate the place to be insert itself is already a 4-node. Now, naturally you do not have anything > 4-node. So, if it is a 4-node then all that you need is to split that node you have to split the node. So, you have a 4-node.

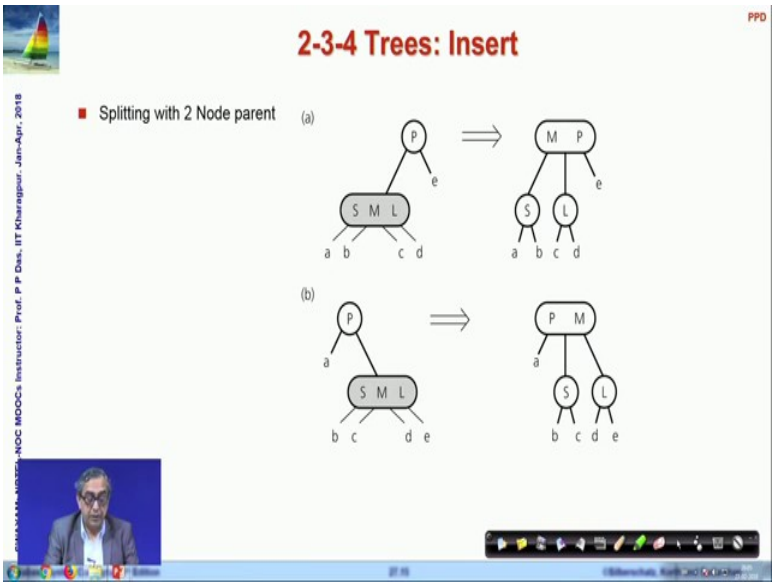
So, you have a 4-node. So, you have you have a data 1 here you have data 2 here, you have data 3 here and you are you know that the your d has to get inserted in this. So, you cannot insert it by changing the node type because this is a maximum allowed. So, all that you want to do is to change that into at different structure and that structure is called a node splitting structure. So, we will we will see in the next slide as to how this is done and we will see how different splitting sequence can happen and what will happen when a 4-node will split when it was a root or the different kinds of parents that it has let us just go there.

(Refer Slide Time: 17:42)



So, what we are saying is the suppose you have a 4-node which is a root now splitting that is actually doing this. So, you have to convince yourself that enough 2-3-4 tree what we have already assumed; whether I represent this or I represent this are algorithmically their equivalent. So, a single root 4-node and a such a structure of your 3- node, two nodes are equivalent why is it? So, for example, if you are looking say if you are looking for a here then it is it say it is less than s. So, you come here now if you are looking for the same a here what happens **a < S**. So, it is it **Search Key < M** remember **S<M<L** So, **a < M**. So, you take this part it is < S to you come here let us take a case of c lets say c. So, if it is c you should come here, now if you check here c if it is falling on this link; that means, it is **c > M** and **c < L**. So, **c >M** you come here **c < L**. So, you come here. So, you reach the same link. So, you can see that actually a 2-3-4 tree is not a unique representation depending on the requirement I can replace 4-nodes in terms of other 2-nodes and actually create a new tree configuration. So, if it is at the root I can get rid of a 4-node and replace it by this equivalent tree.

(Refer Slide Time: 19:27)



Now, suppose there are the 4-node is not at the root it is somewhere else. So, what are the possibilities the; if it is not at the root it must have a parent now the parent could be a 2-node. So, if it is a 2-node then these are the two possibilities if it is a 2-node, then. So, this is a parent node which is a 2-node. So, then the 4-node could be a left child of that or it could be a right child of that if it is a left child, then we use split take the middle item and insert it in the parent and by that process a parent becomes a 3-node from a 2-node and make these become two different 2-nodes.

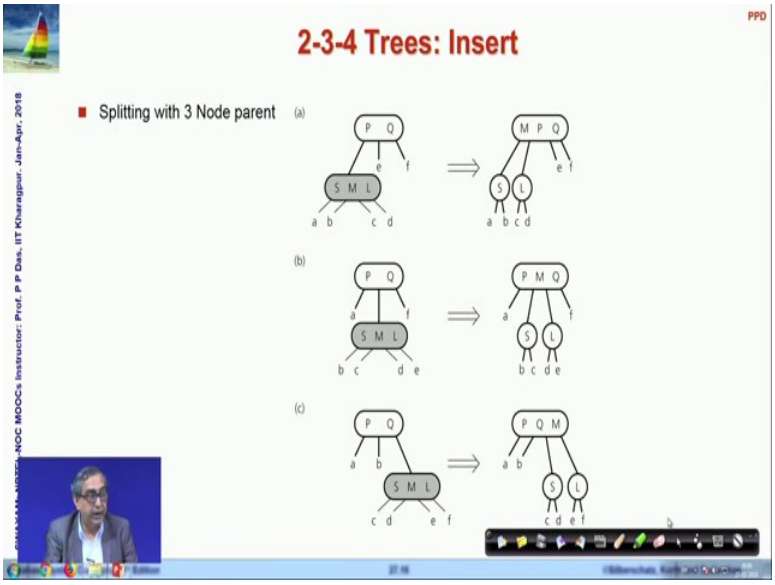
Again the way I was just explaining in the in the previous slide you can convince yourself that this structures are equivalent for example, if I am looking for d let us say if I am looking for d in this tree.

So, if I am looking for **d > L**. So, how do I arrive here now since this is a left child? So, it **d<P**, otherwise it could not have occurred on this side. So, **d<P**. So, **d <P**, **d >L**. So, given that if I search for d here so, **d<P** and since it is we I also have from this the **d >M**, otherwise it would not have come to the; this third link this fourth link it would have been elsewhere. So, I know that **d >M**.

So, and **d<P**. So, if I combined this two, then d has to go on this middle ink, because it is **d>M** and **d<P** and then I have it is **d>L**. So, it has to be on the right. So, it comes to the right position.

So, in this way you can convince yourself in every search case that if the parent is a 2-node, then you can equivalently split the 4-node and make an insertion in the parent 2-node converted into 3-node and get rid of the 4-node altogether that lives us with.

(Refer Slide Time: 21:45)

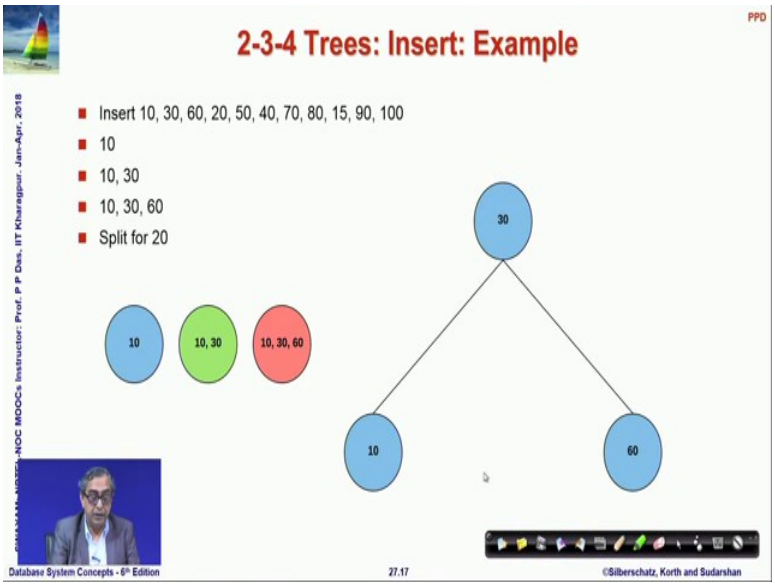


One other case which is if the parent is a 3-node naturally if it is the parent is a 3-node then there are three possibilities. Now, because your 4-node could be a left child a middle child or a right child and in every case you do the same thing you split the 4-node take the middle item put it to the parents. So, parent converts from 3-node to 4-node. Now and your rest of the split and pointed adjustments had done. So, that you get rid of this.

So, what happens in this process in and like the earlier ones here you do not get rid of the 4-nodes altogether, because in replacing one 4-node your creating another 4-node, but in the process what is happening the 4-node is moving up it is it is going one level up.

So, again what you will do is in recursively the new parent 4-node parent will again be split and I just split if it is it is parent that is parent of the parent if that parent is also a 3-node, then that will become a 4-node this will continue till your root becomes a 4-node and we know that when root becomes a 4-node I can always change it to a configuration of 3, 2-nodes. So, in this process as we have shown that we can actually get rid of the 4-nodes in the whole of the 2-3-4 tree when it is required.

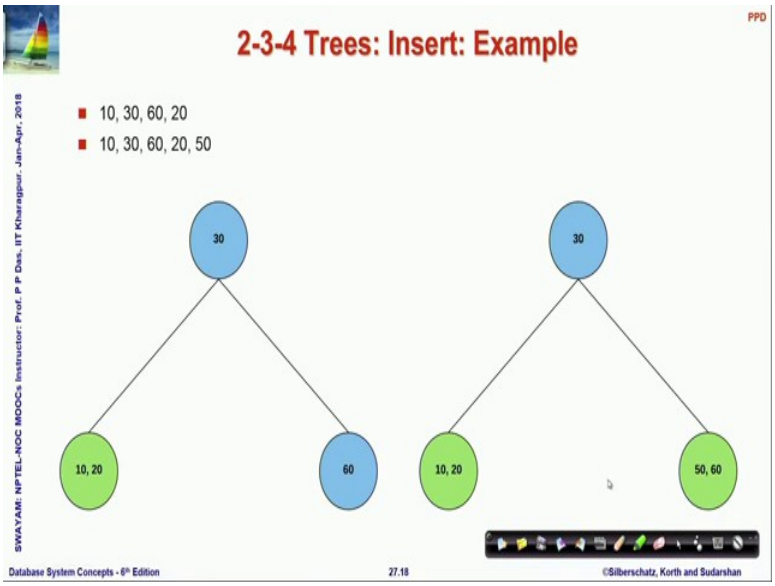
(Refer Slide Time: 23:03)



So, the basic strategy is very simple that will keep on constructing the 2-3-4 tree and whenever we come across a 4-node for the first time we will split that and rearrange. So, that I can get rid of it so, here what I show you is a basically an insert sequence over the next couple of slides where which is trying to insert the data in this following order starting with an empty 2-3-4 tree. So, we first insert 10. So, you get this then we insert 30 that is here. So, 2-node becomes. So, the here the convention that I am I am following is blue is a 2-node green is a 3-node and red is a 4-node. So, then you insert 60 it becomes a 4-node and as soon as it becomes a 4-node the next element to be entered is 20.

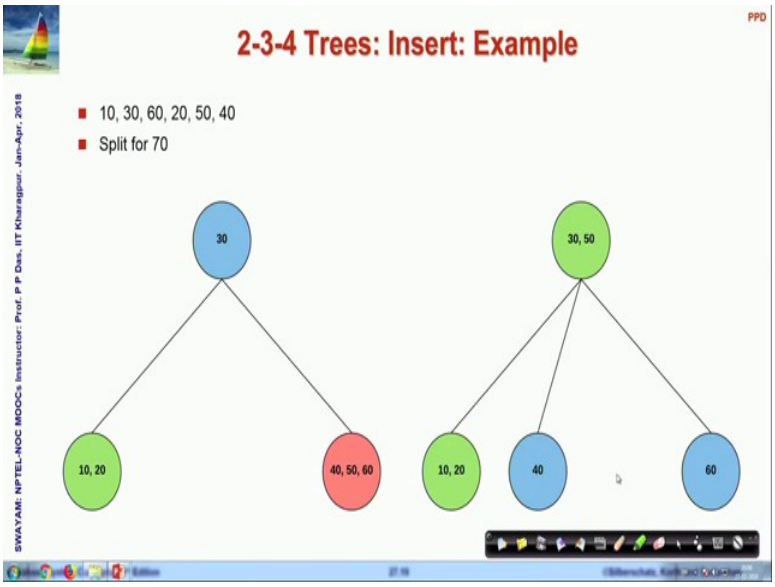
But, before that this 4-node will have to be split and this is the case of splitting at the root, because this is this has no child yet. So, you split and you get the middle moves to the top here as 30, then you have two children 10 and 60 and after the splitting you move on to actually inserting the intended 20 into it.

(Refer Slide Time: 24:19)



So, 20 gets inserted on this side 20 is inserted on this side, because a smaller than this and comes here. So, this 2-node becomes a 3-node then you insert 50 which goes on this side which goes on this side and gets inserted here.

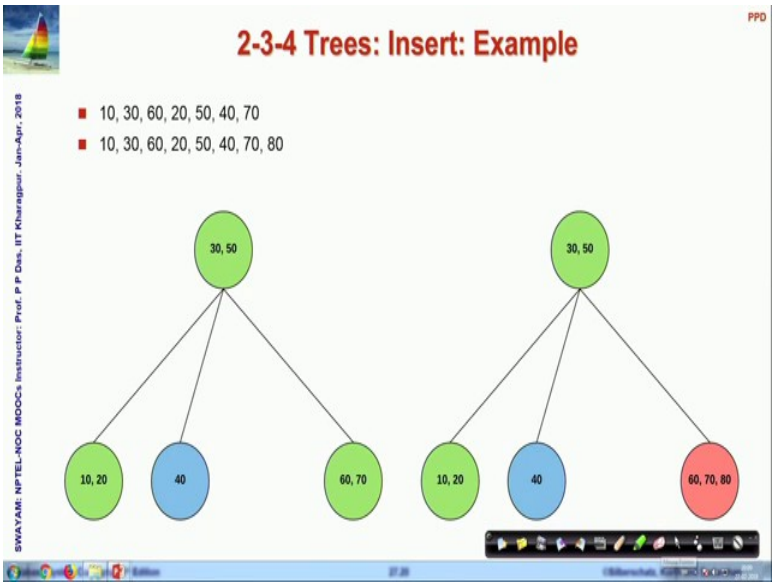
(Refer Slide Time: 24:41)



So, your insertion continues your next to insert is 40 which gets the inserted here it becomes a 4-node and as soon as it becomes a 4-node and before the next insertion of 70 can happen you need to do a split. So, you do a split.

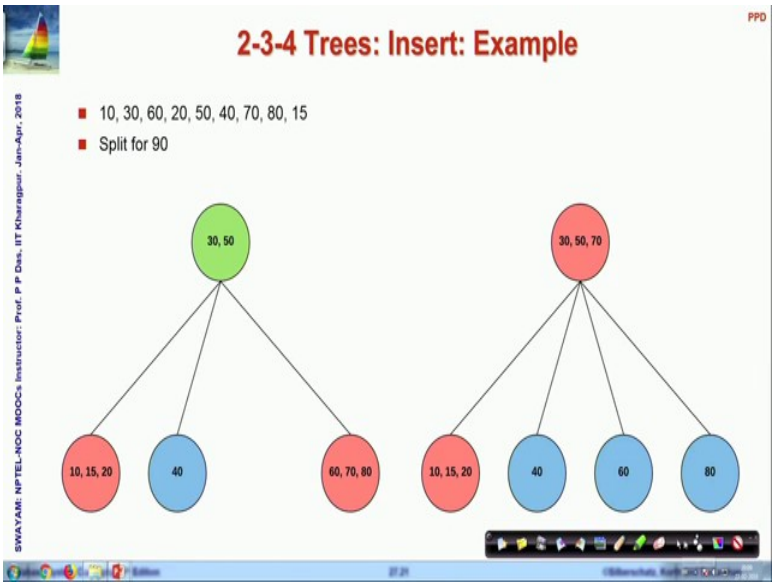
If you do a split then your 50 moves to the parent this becomes a 3-node. Now and your 40 and 60 becomes two 2-node children as this. So, it is a same information is represented, but now in this new 2-3-4 tree you do not have any 4-node. So, in you can go ahead and insert 70.

(Refer Slide Time: 25:19)



So, insert 70, 70 gets inserted here the 2-node becomes 3-node the, this is done. So, then you insert 80, 80 gets inserted here is greater than it comes here and this becomes a 4-node and. So, naturally the next would be two insert 15. So, 15 goes in on to the left. So, 15 has come in here which becomes a 4-node.

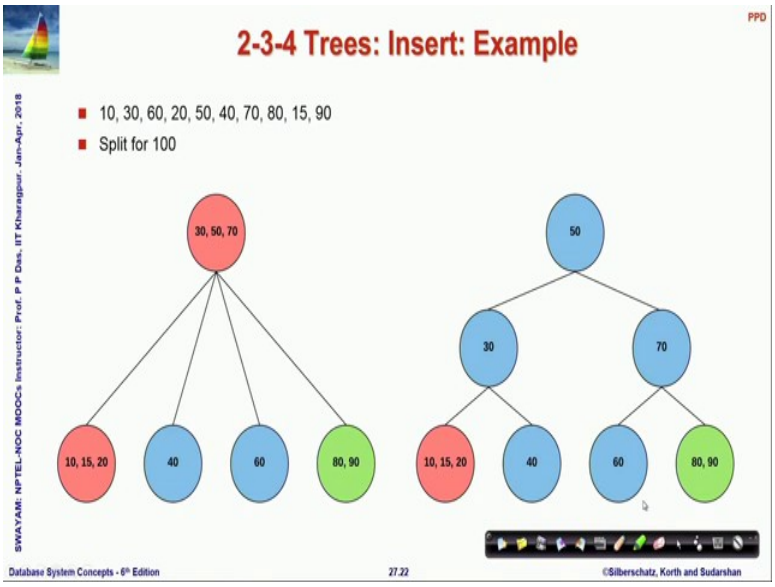
(Refer Slide Time: 25:43)



And the next is to insert 90. So, which has to get inserted here it should have got inserted here. So, this is already a 4-node. So, you need to split.

So, in split and as you split you get 40, 40 was already there you get 60 and 80 and the 70, that existed in the middle goes to the root and your root becomes a 4-node.

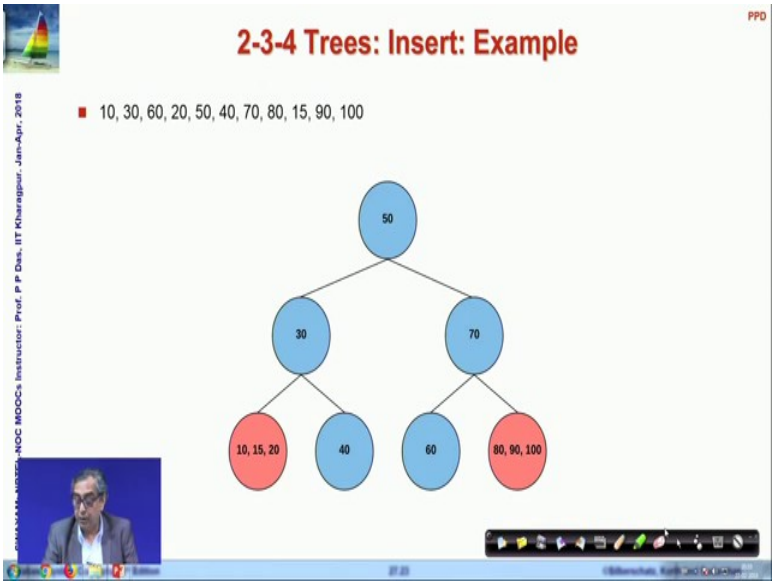
(Refer Slide Time: 26:27)



So, this is the configuration you move on to your 90 gets inserted. Now, you have to insert 100. So, you before that this your root has become a 4-node. So, you do a split. So,

as you do the split this is a new configuration that has happened and this is what has resulted from the split of this 4-node at the root.

(Refer Slide Time: 26:59)



And then naturally you can go ahead and insert 100 and 100 is now inserted. Here, you could do adjustments of changing this this 4-node into by splitting it and moving it onwards I have not shown that, but if you now go back if you just now go back on on these this whole process and you will see that specifically that we claim that all leaf nodes would be in the same level. So, you can see that initial three trees all are at the same level, then leaf and their level 0 and then when you have had this split then the leaf nodes 10 and 60 are both at level once.

So, we can see that when actually you split at the root you add one level to all the leaf nodes and that is the only time your height changes. So, beyond that you look at this the level has not changed I am going to the next the level has not changed instruments to be height 1 level does not change height 1 does not change does not change till the left here and then we have a case again of splitting a 4-node at the root.

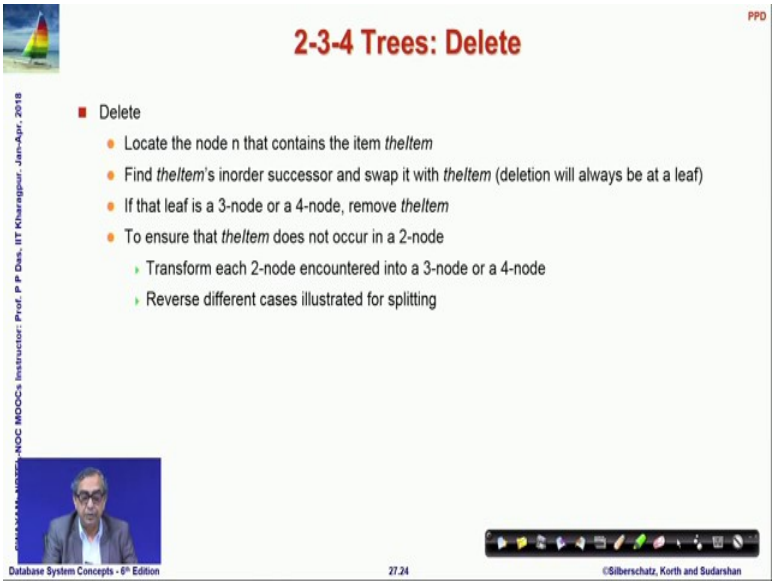
When the one level has been added to all the; so, all the leaf nodes where at level 1 now all of them are at level 2. So, in a 2-3-4 tree you achieve this invariance of all leaf nodes being at the same level by maintaining that the only time the height changes is when you split a 4-node at the root and add one level uniformly to all of them.

And then rest of the logic is quite simple that these are here you can do actually follow the same logic as of the binary search tree analysis that if there are n n items here; then the maximum height could actually be order log n; because we though all nodes are not binary, but the nodes that are not binary actually have more data.

So, they will be they will the height will always be log n or less than that of course, there is an issue of concluding the complexity, because now you will argue that there are 3-nodes or 4-nodes where more than one comparison is required to decide about the node, but the counter argument to that is even if that be the case even if you have along the path of the tree from the root to any leaf node.

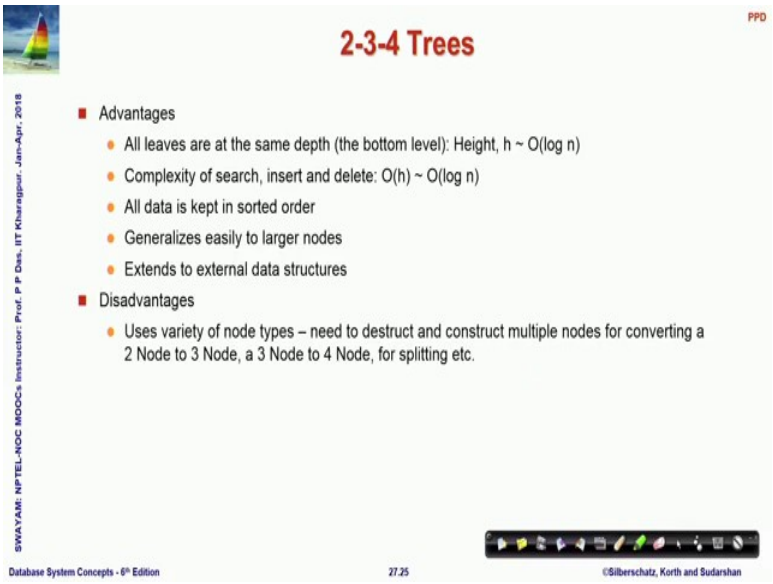
Even if all of the nodes are of are 4-node that cannot happen as you have seen because you will keep on splitting and distributing that, but if all of them are or 4-node also then what will add is simply a factor of three two rather additional with the log n and the overall complexity remains to be log n will go.

(Refer Slide Time: 29:59)



Now, if you have to delete you have to do very similar operations I have not shown the details, but you look at the node and then actually find the in order successor for that and swap it with the item and then you can do the other arrangements of collapsing the nodes as we have done the splitting of nodes. Now I have to do the collapsing of nodes following the same river structure and leave that as an exercise to you just work it out at home.

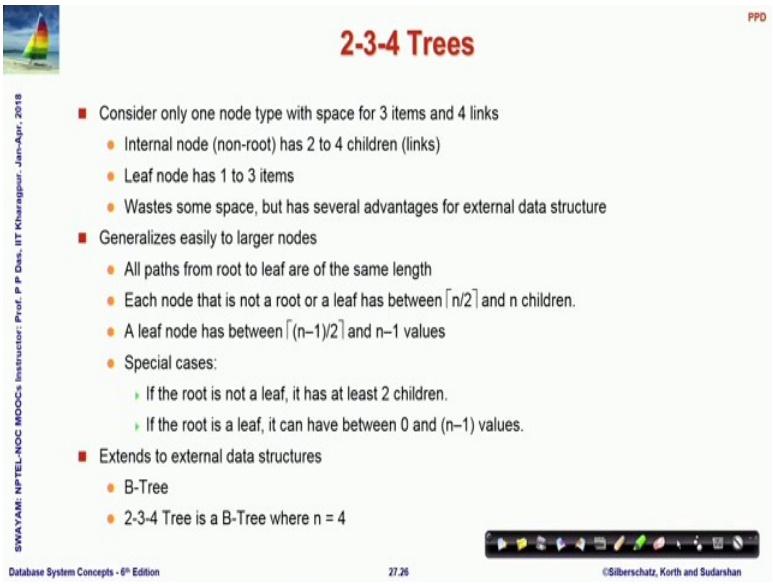
(Refer Slide Time: 30:28)



So, if you look at if the 2-3-4 tree and there is a time to justify why we are doing this all leaves are the same depth which is a great advantage the height is order log n always complexity of search insert delete all are order log n all data kept in sorted order it generalizes easily to larger nodes are and extends to external data structure. So, what you mean by larger nodes, let us let us look at that a little bit before that of course, 2-3-4 tree has a major disadvantage compared to binary research trees of the other kinds because it uses a variety of node types. So, you I mean when you change from a say 2-node to a 3-node you actually if you think in programming terms you have lot of additional cost, because you need to distract the 2-node and create a 3-node.

If you split a 4-node and create 3, 2-nodes as required, then you have to distract the 4-node and create 3, 2-nodes. So, there are lot of over rights in terms of that.

(Refer Slide Time: 31:32)



So, what leaf if we if we just simplify that process and if you assume that, there is only one node type which has enough space for three items and four links we do not have two I mean we functionally there will be 2-node 3-node 4-node, but physically let them with a same type of node. So, any internal node can have what is specification? So, there the same type any internal node has 2 to 4 children and a leaf node has 1 to 3 items that is what all that we are saying.

So, by doing this we will waste some space, but we have several advantages particularly when you look at this for external data structure. So, what will happen is if I can think about 2-3-4 tree in that manner then I can; obviously, generalize that it is not necessary that I will have to restrict myself at 4 links, 4 children, I can do more than that.

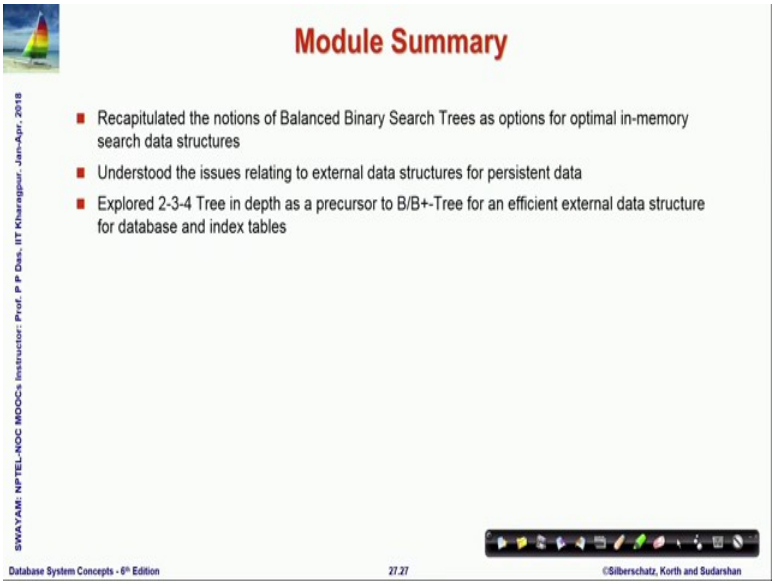
So, in general I can say that there are a node has n children and it is each node is not a leaf not a root or a leaf will have between **n / 2** and **n** children. So, put n as 4 you will find that it becomes 2-3-4 tree and a leaf node will have **(n-1) / 2** which is 1 and for enough n being 4 and n-1 or 3 values which is what did you have.

So, if you just generalize from 2-3-4 to n/2 to n and have a a container have a node container which can have either n/ 2 or (n / 2) + 1 or (n / 2) + 2 or maximum up to n - 1 data items and corresponding number of children then we will be very easily be able to arrange for a similar data structure which will have all the nodes all the leaf nodes at the same level and. So, this is the structure which is which extends very easily and this is a

fundamental structure of what he says a B-tree or a B + tree will see the differences shortly.

But this is a basic notion and the strategies of node splitting and node merging in case of deletion and the algorithm of insertion deletion that we have discussed here will simply get generalize in case of B-tree.

(Refer Slide Time: 33:57)



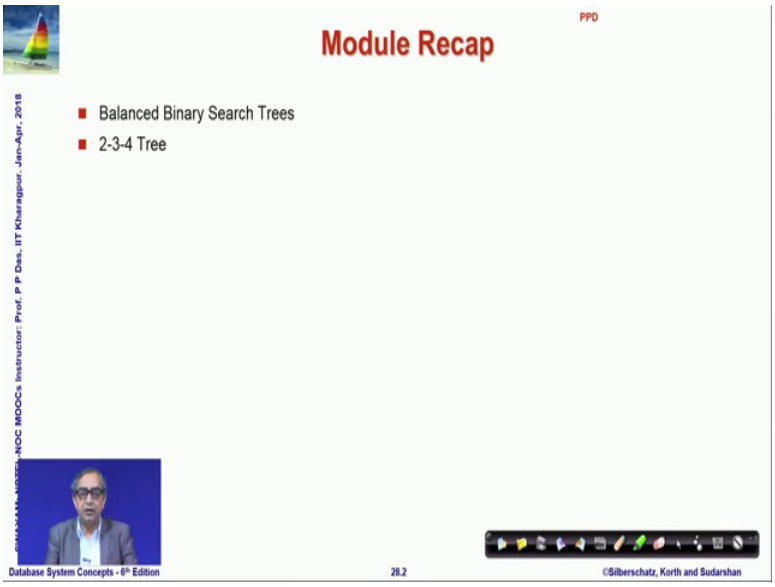
When we go to the next module so, we have recapitulate in on the balanced binary search tree and we introduced a the notion of a 2- 3- 4 tree which is a precursor to B + tree B- tree which is which are efficient external data structures and will be covered in the next module.

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

Lecture - 28
Indexing and Hashing/3 : Indexing/3

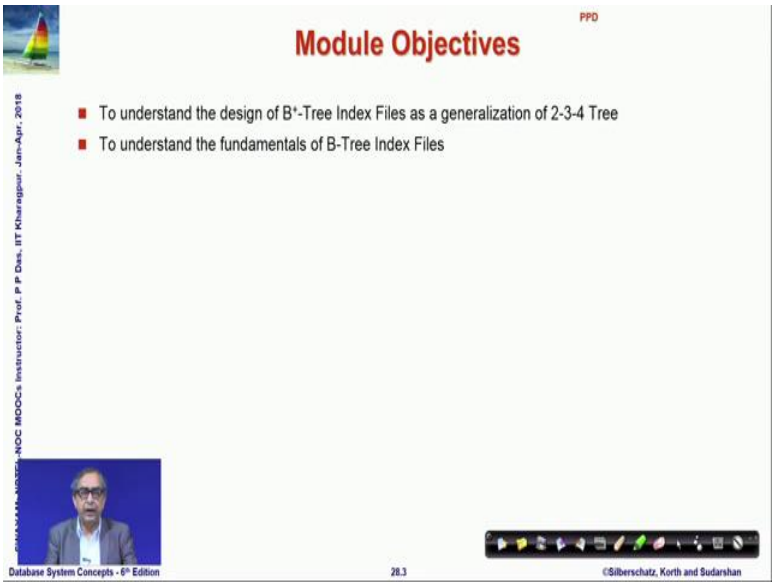
Welcome to module 28 of Database Management Systems. We have been discussing about indexing and hashing.

(Refer Slide Time: 00:28)



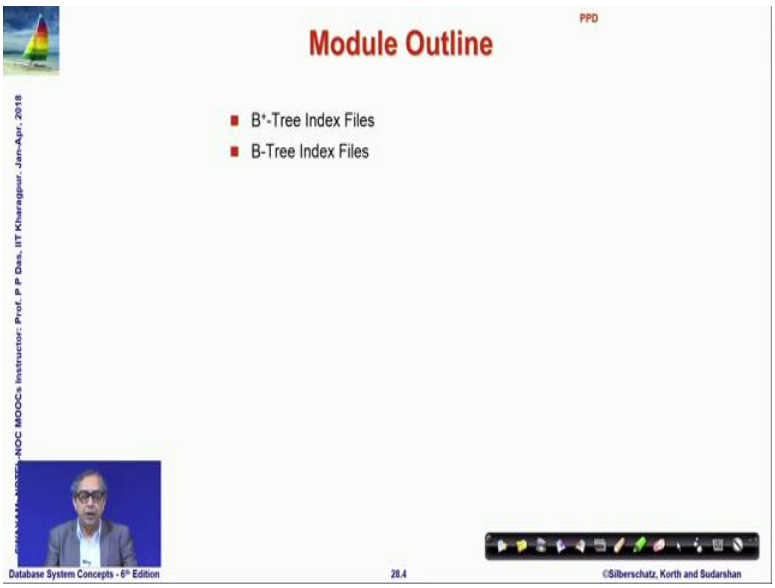
This is the third module; in that continuation.

(Refer Slide Time: 00:38)



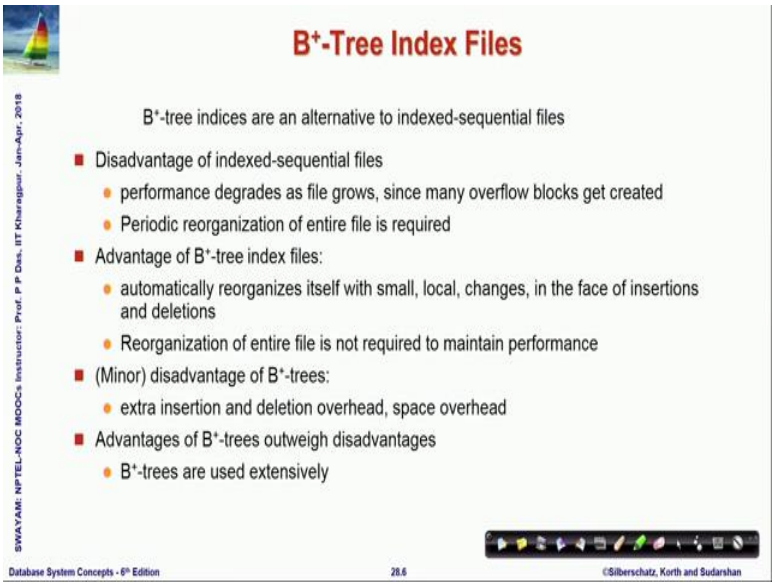
In the last module we have taken a quick look at the balanced BST and specifically a and different kind of inline data structure called 2-3-4 tree, which can be of very good use in terms of understanding B+ tree, which we want to study in this module and we will also take a quick look at the B tree.

(Refer Slide Time: 00:50)



So, now, B + tree is the main data structure is or one of the main data structures to be used for index files.

(Refer Slide Time: 01:00)

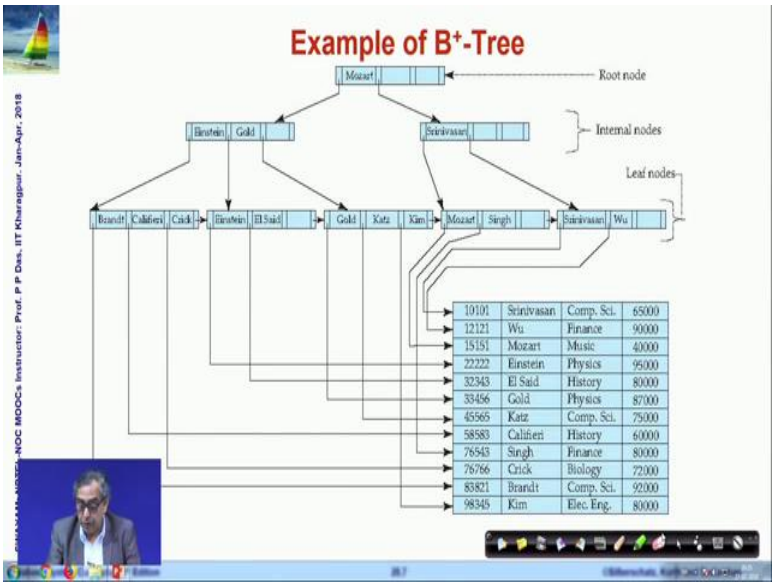


So, B + tree has now; what we have seen we have seen the ordered indexes. We have seen the index sequential files, where you could keep the index file in a sorted manner in the primary index you could build secondary index on that and so, on, but that is not an efficient way of doing things, because the performance keeps on degrading as the file grows.

Since many overflow blocks will get created, because certainly if you if you are growing, then naturally you have created say sparse index on uncertain values and if there are more records in that bucket. Then naturally you need to have linked buckets. So, periodic reorganization of the entire file becomes required which is a very costly affair.

In contrast advantage of B + tree is it automatically reorganizes itself in small bits and pieces with local changes and so, on; whenever insertions and deletions happen and the reorganization of the entire file is not required for the purpose of maintenance. Of course, there are a little bit of disadvantage the extra insertion and deletion overhead exist for the small you know micro reorganization there is little bit of space over it, but in the face of the advantage that we get it outweighs the advantage is outweighs the disadvantages and B + trees are used quite extensively.

(Refer Slide Time: 02:28)



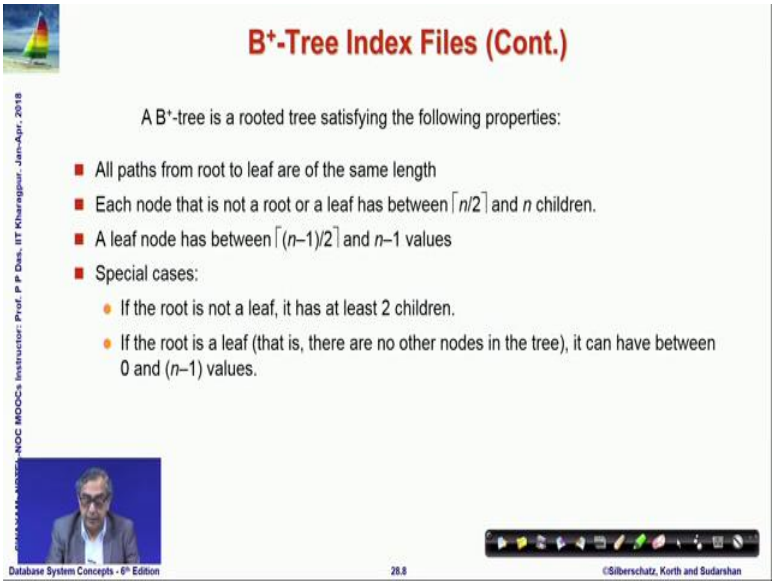
So, just recall the notion of 2-3-4 tree that we had discussed and look at this diagram. So, 2-3-4 tree have different types of node : 2 node 3 node and 4 node. So, we said that there could be a node which can be only partially filled and it has a different number of children pointing to the; conditions of how different keys are ordered in that particular node.

So, here we I show an instance of a B + tree, which is basically trying to represent this file in terms of the creating indexes. So, if the index is actually based on the name. So, this is the root node that you have and for an instance; we are taking a structure where every node can have 3 data items and 4 links and it could be it could be more it could be less, but this is just for an example. So, as you can see; so if we have this link, then on the left of Mozart, then it means all keys which are less than Mozart will be available on this link below; the link that exists here is for all keys which are greater than Mozart and less than right. Now there is nothing.

So, those will occur here. So, as you can see that Einstein, Gold, Brandt all these will come on this length Srinivasan, Singh, Wu all this come on this side the Mozart itself comes on this side. Now, if I look at this node the next level loads. Now this link has values which are less than Einstein as you can see this has values which are between Einstein and Gold. So, Einstein and l set these are values which are more than gold.

So, this is this is a and as you can as you can see that though all nodes are shown to be of the same type as we had mentioned at the end of the 2-3-4 tree discussion, but it has variable number of entries. So, the number of links are between n/2 and n. So, n here is 4. So, you have at least either at least two entries or maximum up to 4 entries that can go on here.

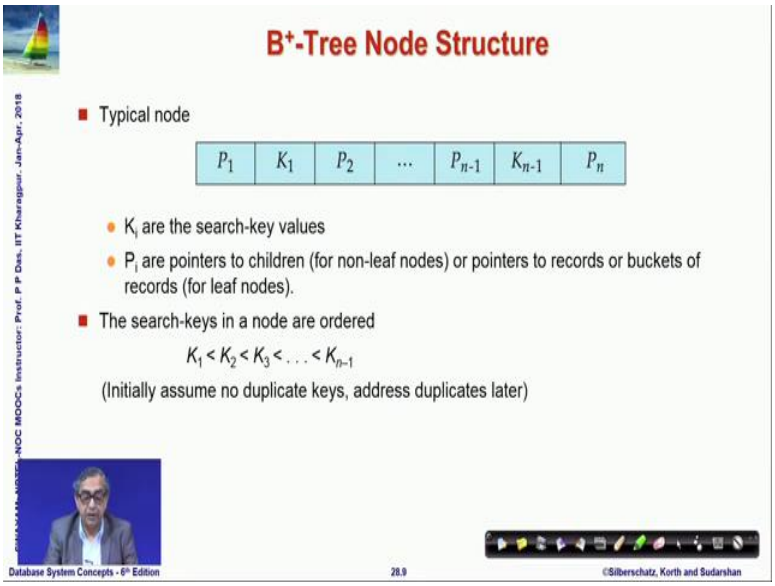
(Refer Slide Time: 05:08)



So, this is the basic observe definition of a B + tree. All paths from root two leaf are of the same length. This is again something you should observe here, because if you if you see all of these paths all of them have the same length here then the length is 2. So, that is a basic property of 2-3-4 tree generalized into B + tree.

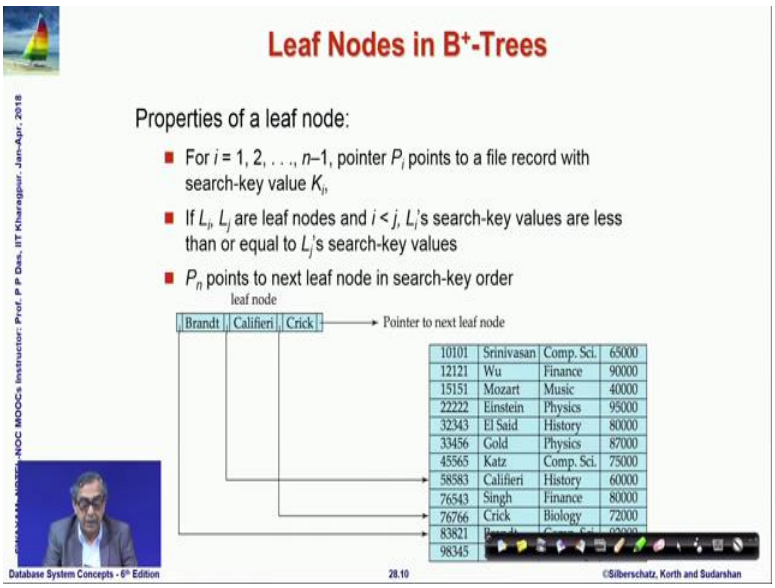
So, each node that is not a root is a leaf level has between n / 2 to n children. Leaf node has (n -1) / 2 to n -1 value. And the if the root is not a leaf, then it has at least 2 children and if the root is a leaf there is no other nodes in the tree then it can have between 0 to n-1 values which are quite obvious.

(Refer Slide Time: 05:54)



So, naturally a typical node will look like this, where the pointers and key values alternate starting with a pointer P 1, then key K 1 and so, on and ending with a point at P n. And the search keys are strictly ordered $K_1 < K_2 < K_{n-1}$ these are facts that we have seen for 2-3-4 tree.

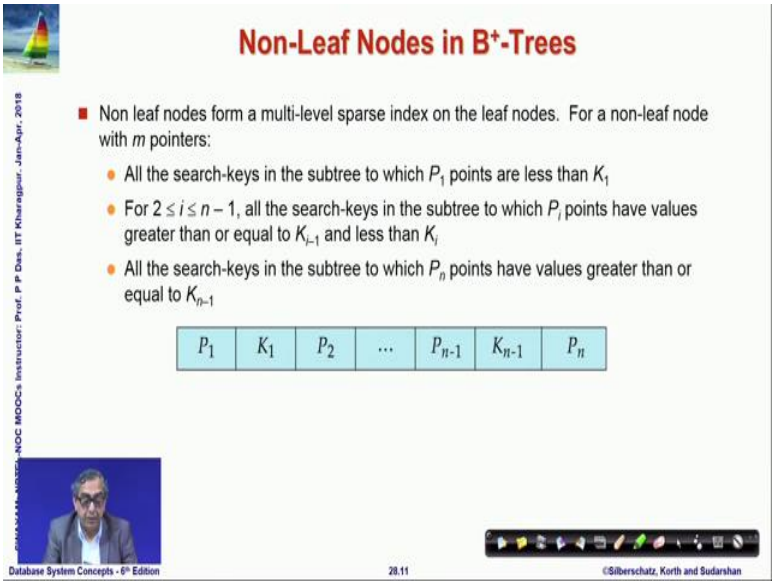
(Refer Slide Time: 06:14)



So, for a leaf node the pointer P_i points to the file record with the search key K_i and if there are two leaf nodes L_i and L_j and **i < j**, then $\mathbf{L_i <= L_j}$ search key values. So, this is

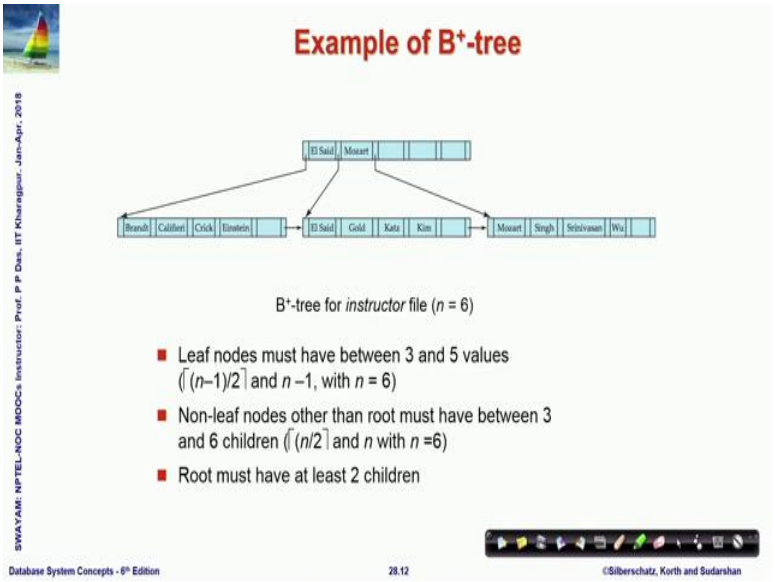
the basic ordering that we had seen in 2-3-4 tree, that is what is getting generalized for a non leaf node.

(Refer Slide Time: 06:40)



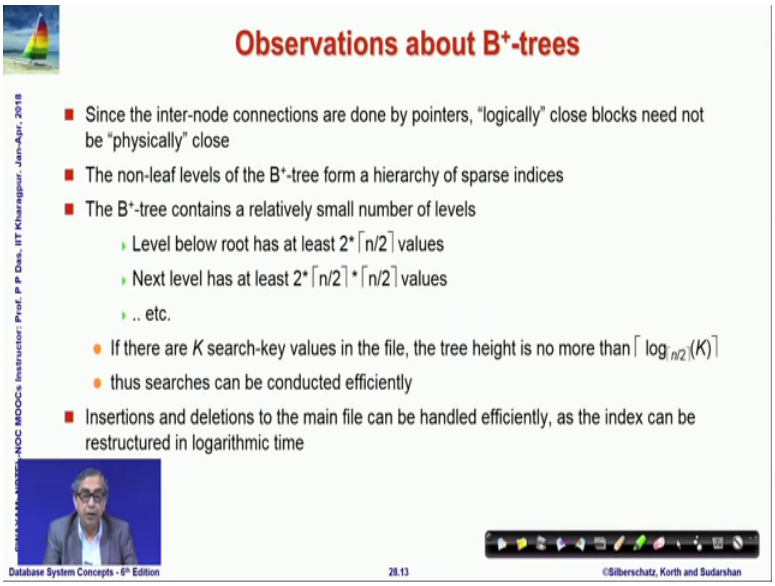
Similarly all search-keys in the subtree which P_1 points to a $<K_1$, then for all that P_n points to are $> K_{n-1}$. And in the other cases they are between the two consecutive key values that exist between the pointers.

(Refer Slide Time: 07:01)



So, this is an example of a simple case which is n where n = 6.

(Refer Slide Time: 07:10)



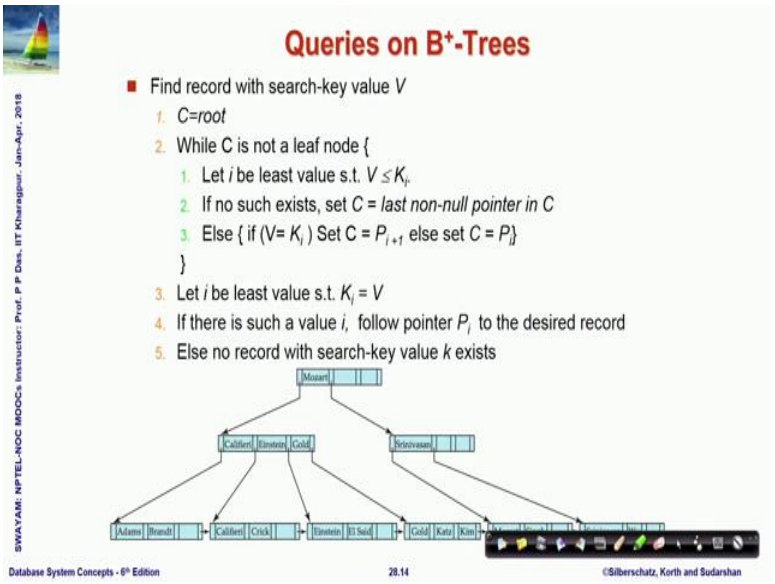
So, since the inter-node connections are done by pointers, "logically" closed blocks are not "physically" close. So, that is a key idea there is a key observation about the B + tree. So, 2 nodes the records which are logically closed are may not actually be physically close, because the pointers actually define the closeness in terms of the ordering of the values.

So, B + tree contains relatively small number of levels, we will see what that level would be? So, what will happen; if the level below root has two values at the most at least and the below that will have n/2 values, because every node has to be at least half field. We have said every node we will have to have n/2 lengths to n links it cannot be less than that less than n / 2 link.

So, the next level as n /2, then the next level has 2 * n/2 * n/2 and so, on. So, every time you every level you go down you can basically increasing by a factor of n/2, which as you all know simply means that the number of levels or the height is log K to the base n/2, where K is a number of search key values that exist on the tree. So, larger the end smaller is this value. So, larger the node size is smaller is a is a height and therefore, the number of insertion number of you know access operations that need to be performed.

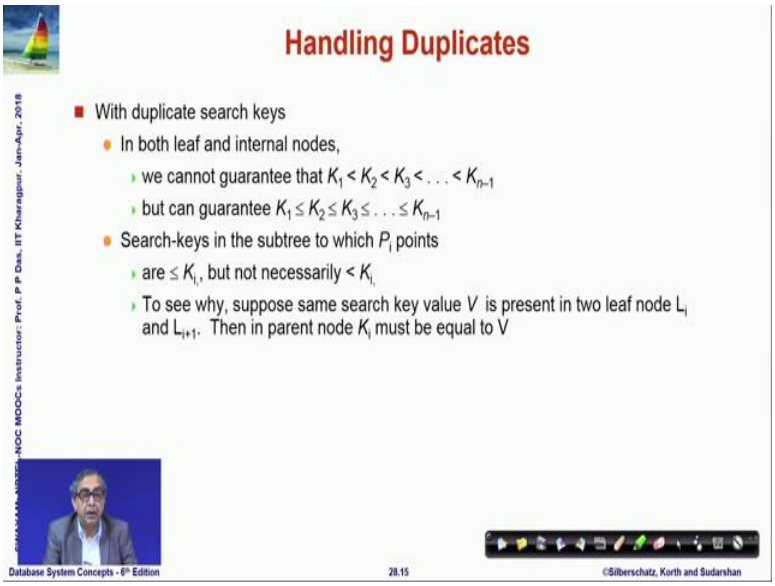
So, insertion, deletions to the main file can be handled efficiently as the index can be restructured in logarithmic time as you have just seen.

(Refer Slide Time: 09:01)



So, search should be very simple, because its just an extension of what you did in 2-3-4 trees. So, algorithm is given here I will skip it, because we have already done this in detail.

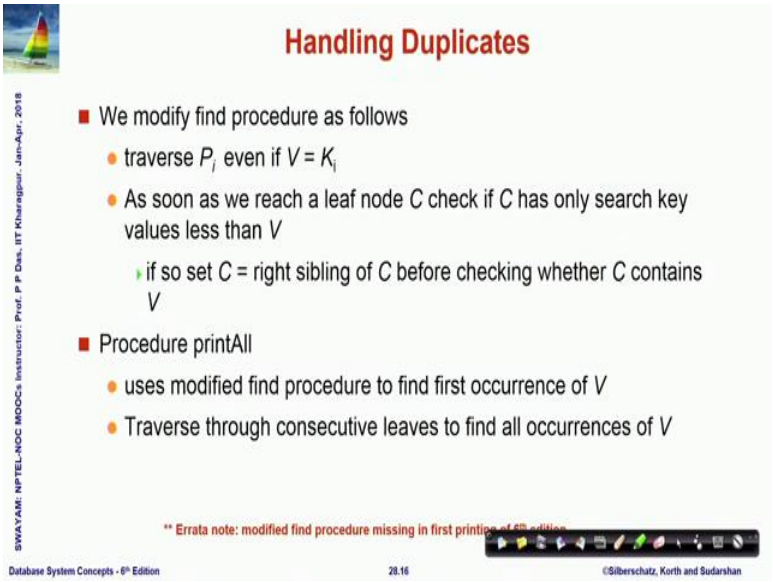
(Refer Slide Time: 09:13)



Now, what we introduced I started saying that there are no duplicates. So, the keys follow strict ordering, but the whole assumption will also hold good, if you allow the equality between the consecutive keys, but only difference is there could be multiple keys which are all equal; and if that happens then you have to use the same key value

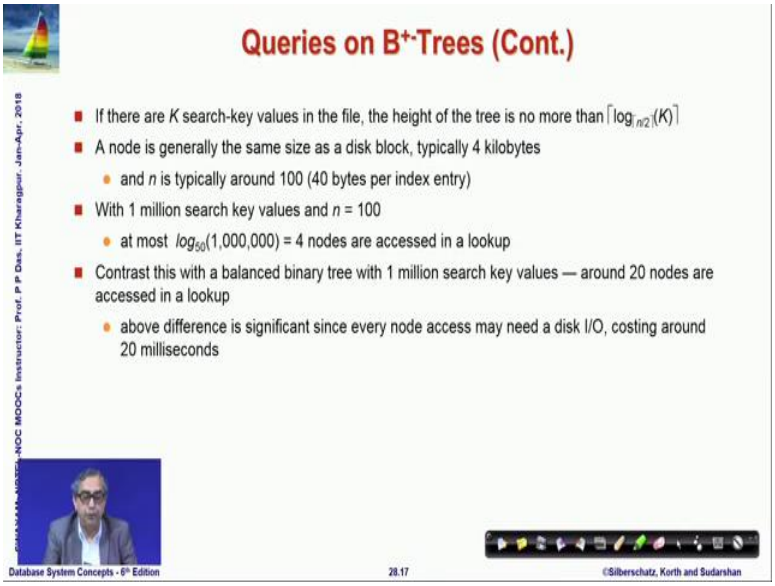
present at the two leaf nodes and the parent will also have the same leaf node same value.

(Refer Slide Time: 09:43)



So, for doing in the case of such duplicates will have to a little bit modify the procedure for doing the search and say printing all values and so, on. So, you could go through that.

(Refer Slide Time: 09:58)



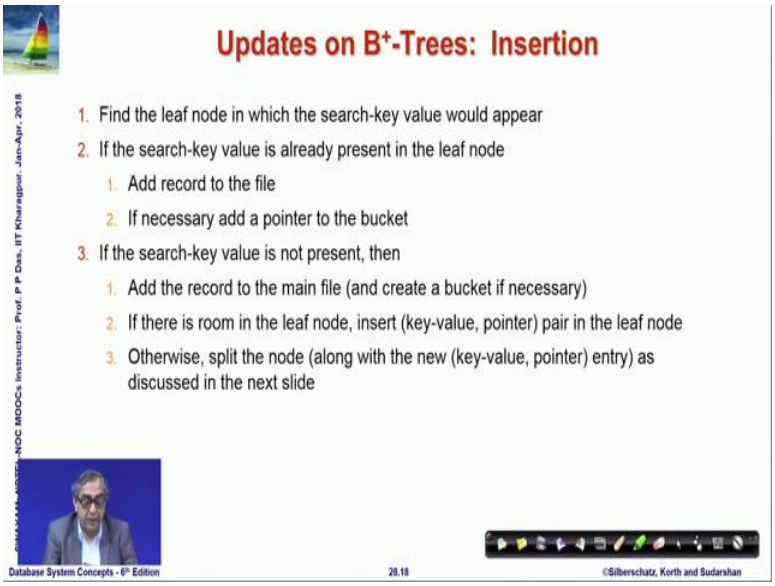
So, if there is a key search-key values in the file, then let us see what the cost is coming to actually, then the height of the tree is not more than $\log K_{n/2}$. So, if we say that the every node. So, how large would be the node. Now again I would remind you that we are

moving from 2-3-4 tree, which was a in memory data structure to a external data structure. So, our main cost is a disk access. So, what would you like to make this node size, if we make the node size too small, then there will be too many nodes and every node will have to be accessed? So, as you can see this is $\log_{n/2}$.

So, we benefit by making n larger, larger the n this log value or the height will be less, but can I make n arbitrary large then n will not fit into one disk block. So, it would it cannot be accessed in one fetch from the disk to the memory. So, we would typically like to make it is customary to make the node as the same size as the disk block, which is typically say 4 kilobyte or 8 kilobyte like that and therefore, the if that is a size then it the n will be typically around 100, because if 4 kilobytes is a is a total space and if I assume that 40 bytes per index entry, which is very typical, then n would be about 100.

So, if I assume that my index file has actually 1 million search key values to look for, then I will need 1 million to the base 100 by 250. So, 1 million log 1 million to the base 50 which is approximately 4 node accesses in a lookup table. So, that is amazingly fast if you contrast this with binary balanced binary tree which will be log 1 million to the base 2; which would be about 20 nodes accesses 20 disk accesses for this lookup. So, this is the core reason that B + trees are preferred and with this if even, when you have couple of million records in a in a table you can actually manage with a very small number of node accesses for the lookup, which makes the realization of algorithms possible in the next couple of slides.

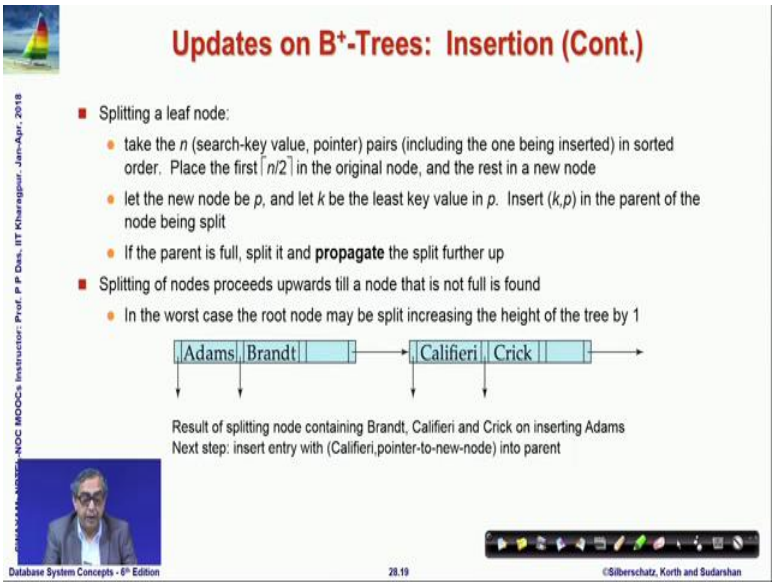
(Refer Slide Time: 12:23)



I have discussed about how to update B + trees talked about the insertion and the deletion process. I will skip them in the in the presentation, now because as we have discussed the process of insertion in depth in terms of the 2-3-4 tree the only difference here is that this is in a generalized framework, but follows exactly the same idea of node splitting and keeping in mind that in case of 2-3-4 tree you move from 2 to 3 and 3 to 4 node here. All that you will have to remember is you always make sure that you have every node half filled, because n / 2 is a minimum requirement.

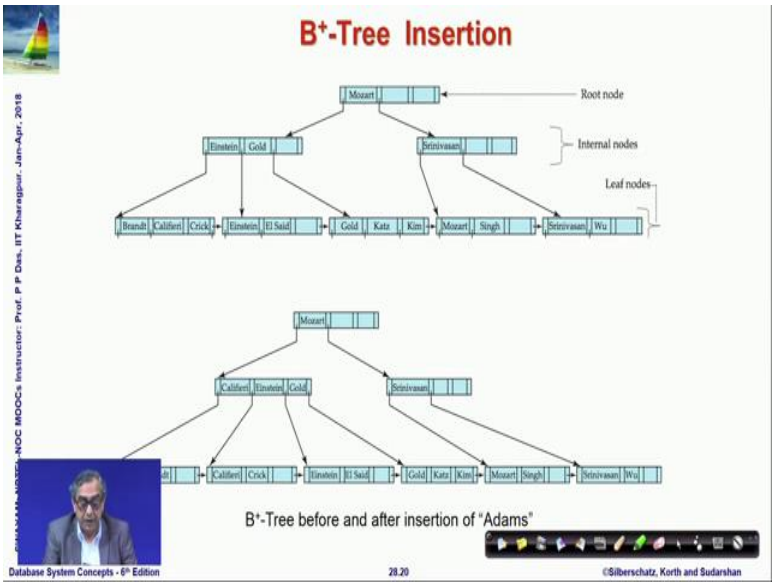
So, you keep on inserting in a node till it becomes full, when it becomes full you cannot insert any more you divide it and split it into two nodes. So, that each one of the them become at least half filled and that is the simple logic and rest of it you can figured out by following on the 2-3-4 tree insertion. So, this is the first algorithm.

(Refer Slide Time: 13:33)



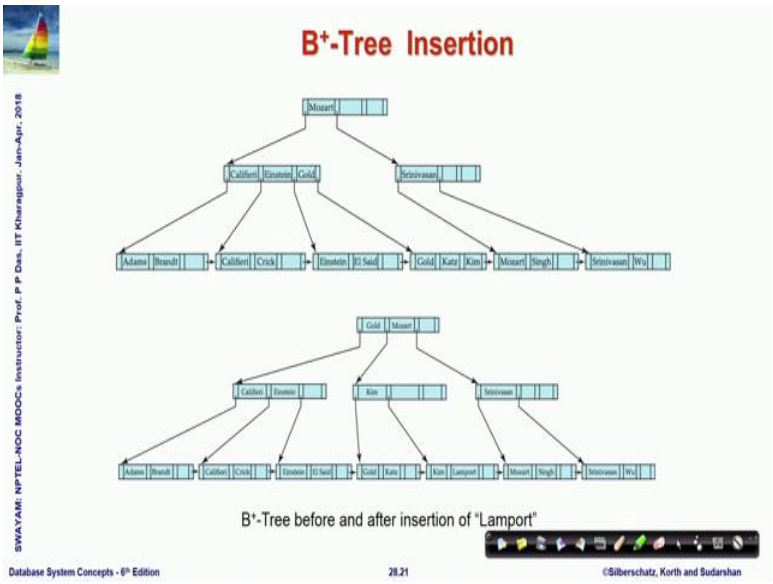
Then we have shown here the strategy to splitting the node, which I have just you know discussed and the same notion of propagating the middle element of the split continues here go to next and here the examples shown in terms of the B + tree.

(Refer Slide Time: 13:47)



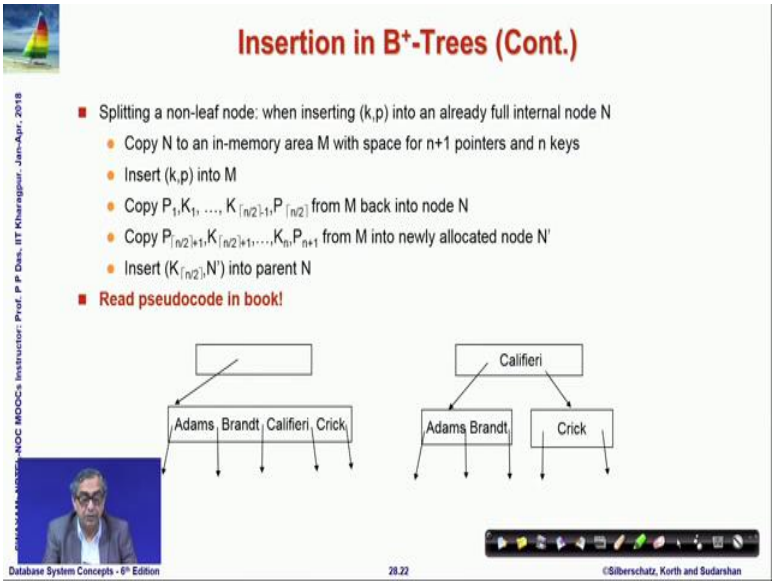
Before and after insertion of a certain key you can go through that and convince yourself.

(Refer Slide Time: 13:57)



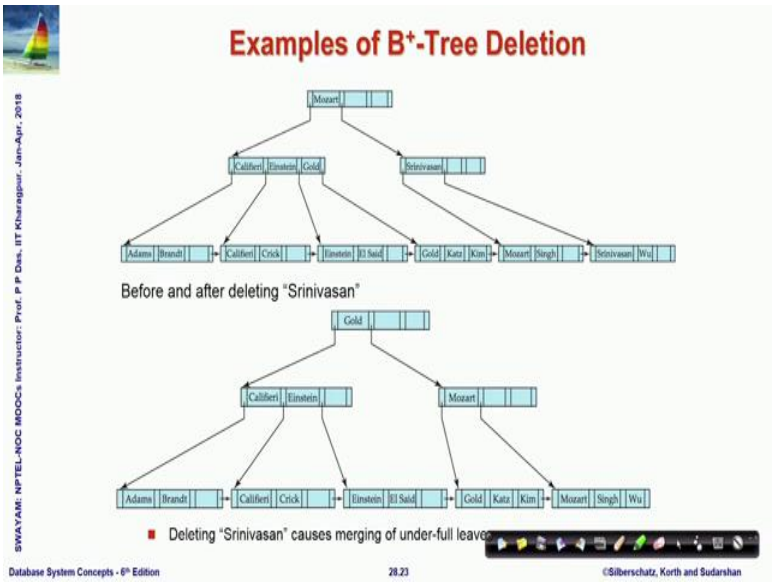
There are some more steps in the algorithm please go through them carefully and try to understand.

(Refer Slide Time: 14:04)



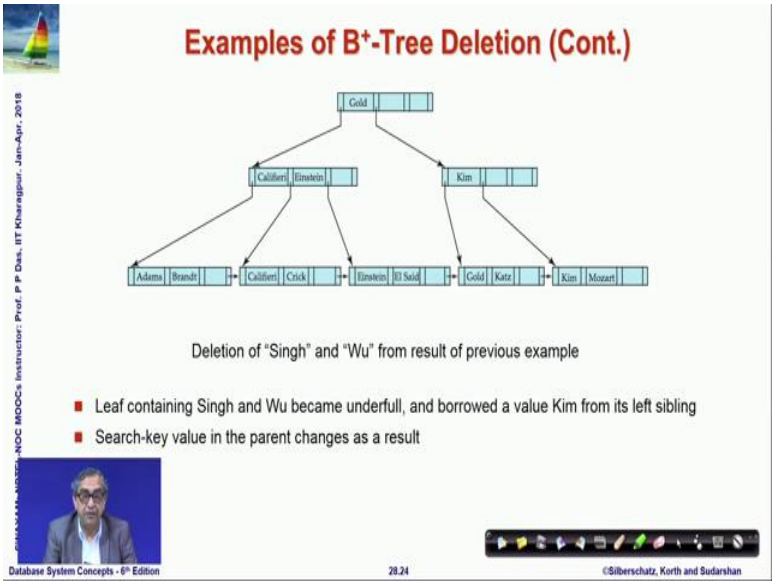
The whole process and then this is the basic algorithm written in a very cryptic pseudocode, I should say you should refer to the book actually to, for and study the whole pseudocode to understand the algorithm better and work through examples as well.

(Refer Slide Time: 14:19)



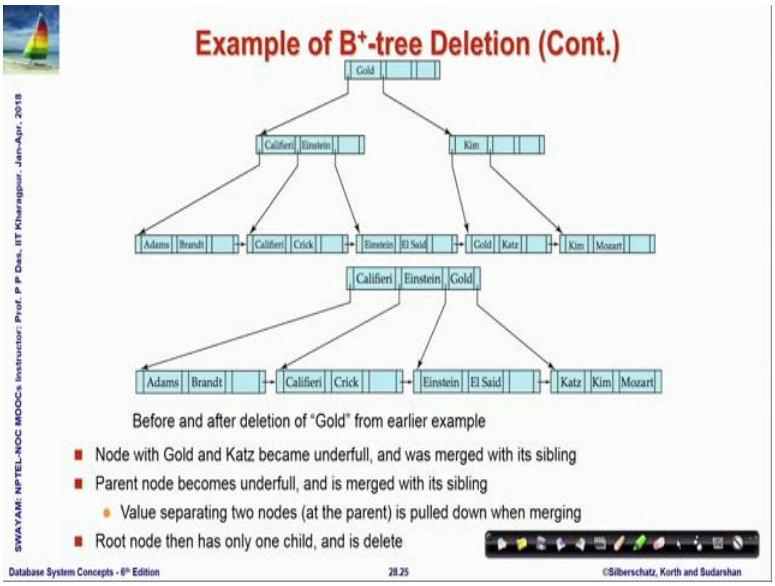
Similarly, examples of deletion in B + tree; so the trees are shown before and after deletion of Srinivasan, then if we delete like that; now in case of in contrast to splitting.

(Refer Slide Time: 14:43)



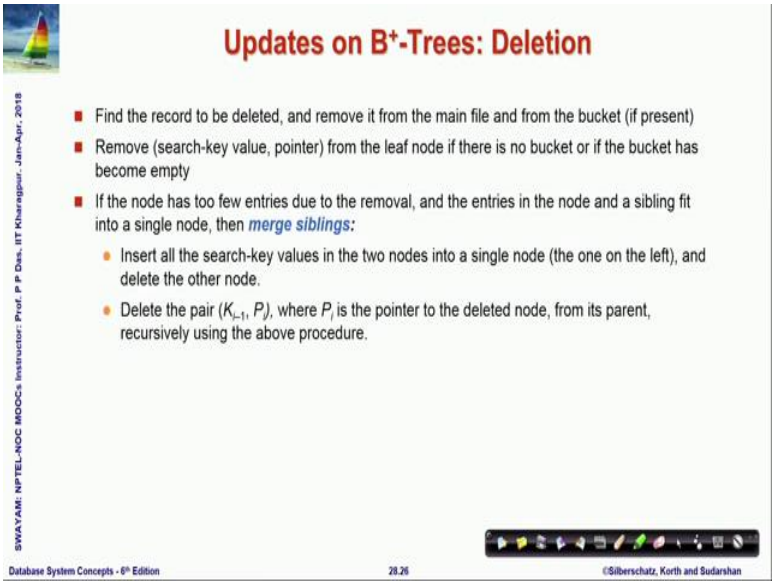
Now I will have merging of nodes which will start happening there are some more steps in the deletion shown here, please go through them and work this out they should not be you should not have any difficulty in understanding them given your background in the 2-3-4 tree.

(Refer Slide Time: 14:56)



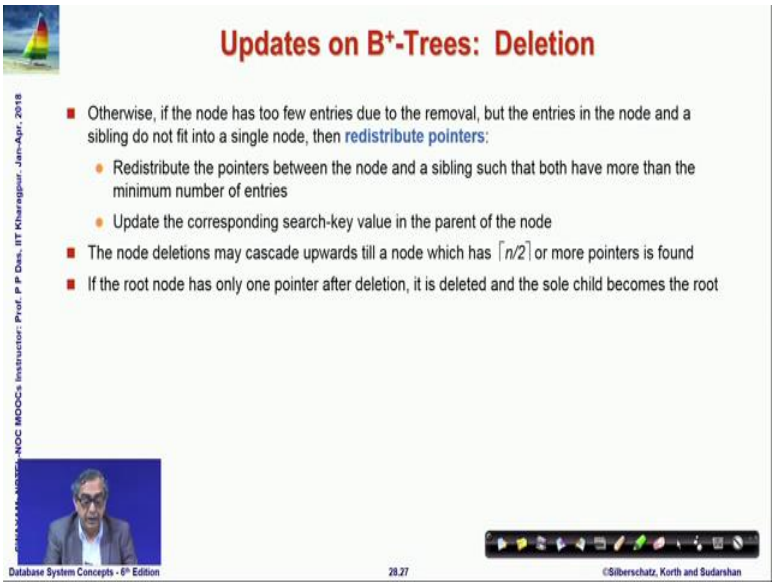
So, more steps in the deletion. So, this is the deletion process in terms of algorithmic steps and what you need to do for deletion.

(Refer Slide Time: 15:05)



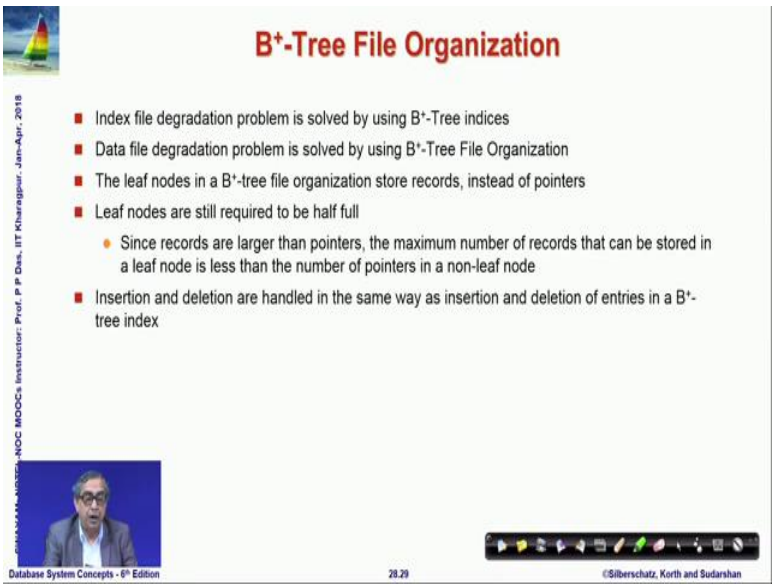
So, this is all detailed here just. So, B + tree file organization is takes care of the degradation problem.

(Refer Slide Time: 15:12)



In terms of the index files which would have happened, if we were used pure ordered indices like, the index sequential access method for storing the index files. So, that is now taken care of and even the data File degradation problem can also be solved by using B + Tree organization.

(Refer Slide Time: 15:20)

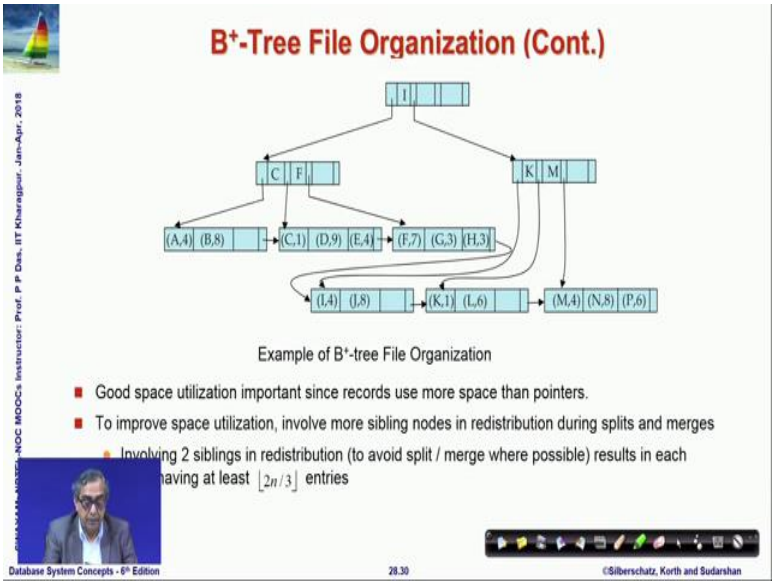


So, it can be used for both maintaining the the index as well as the actual data file and the leaf nodes in the B + tree file organization stored the records instead of pointers. So, you finally, have the records there and the leaf nodes are still required to be half full

since they are records, but since records are larger than the maximum number of records that can be stored would be less than the number of pointers in a non leaf node insertion and deletions are handled in the same way as in the B + tree index file.

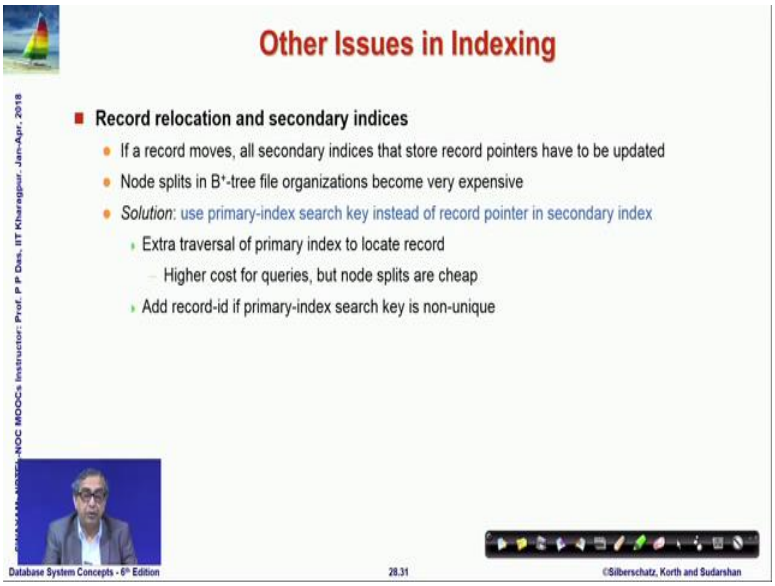
So, here all that we are explaining that. So, far we have not explained the whole B + tree in terms of index file organization and we are saying that you can do the same thing with the data file and only at the leaf level you will have to actually keep the data records for maintenance.

(Refer Slide Time: 16:47)



So, this is showing some instances of the B + tree organization.

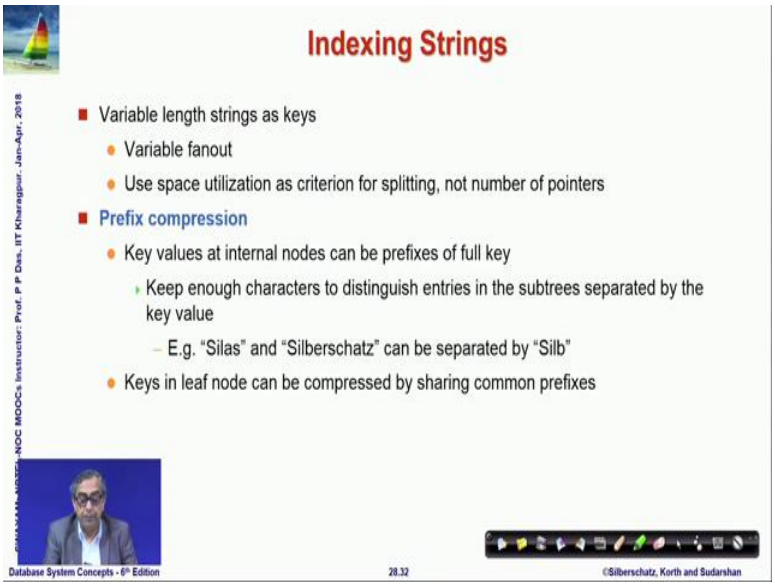
(Refer Slide Time: 16:54)



So, there is a couple of other issues the record relocation and secondary index, if a record moves all secondary indices that store record pointers will also have to be updated node splits in B + tree file organization is very expensive. So, what we do is? We use primary index search key instead of record pointer in the secondary index. So, in the secondary index we do not actually keep the direct record pointer instead, we keep the search-key of the primary index and we know that the primary index can be very efficiently searched. So, what happens is when in the secondary index when you have been able to actually find that you do not get a pointer directly to the record, but you get the search key through which you can use the primary index and actually go to that.

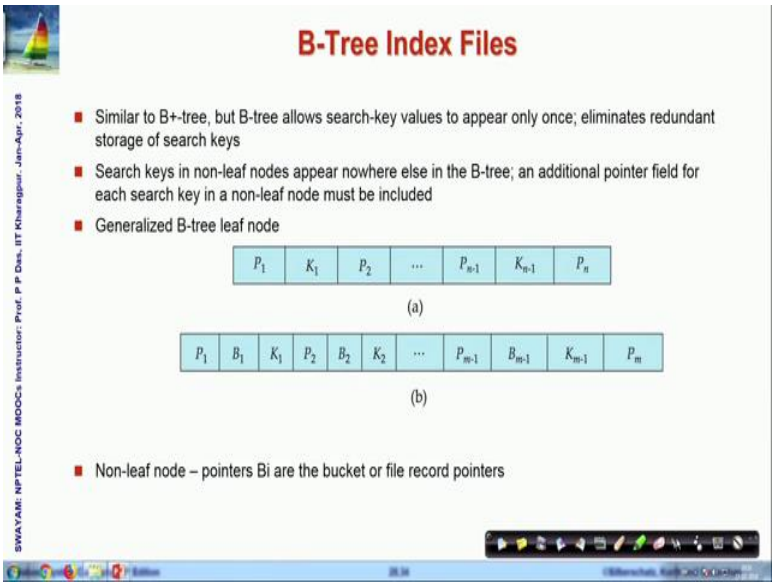
But with that you get yourself get rid of the requirement of maintaining different secondary index structures and getting into several record relocationess problems.

(Refer Slide Time: 18:05)



There are your indexing also may need to take care of other issues of string your variable length string could be keys which are variable fan out and so, the general strategy in handling indexing with string is to do a kind of what is known as prefix compression. So, you kind of find out what is the shortest prefix which can distinguish between the strings. So, if you have Silas and Silberschatz then you can easily make out that Silb would be a separating string between these two. So, Silb will match with Silberschatz, but or not will match with the first one. So, you do not need to look beyond that so, we can just keep enough characters to distinguish entries in the subtree separated I by the key values and keys in the leaf node can be compressed by sharing common prefixes.

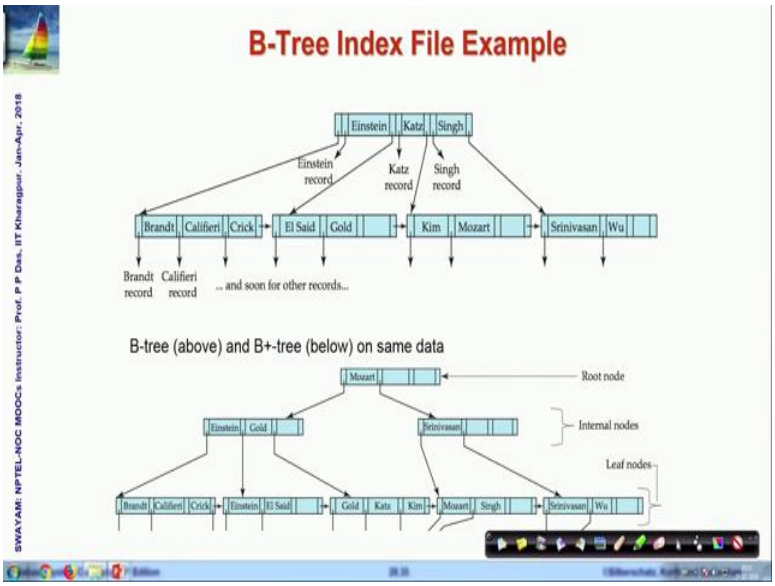
(Refer Slide Time: 19:12)



So, that is a very common strategy next let us just take a quick look into the B tree index file which is another alternate possibility the basic difference between a B + tree. And B tree allows search key values to appear only once, if you if you remember in the B + tree your search key values where which occurs in an internal load keeps on occurring at multiple node levels also B+, B tree does not allow that the search key non leaf nodes appear nowhere else in the B tree.

So, if it does not then naturally the question is when where will the actual record value we found out for this key. So, what you do is in the node itself you introduce another field after along with the key which is the; pointer to the actual record. So, as you can as you can see here let us get back. So, as you can see this is this was a general structure of the B + tree node. And, now what we are doing is we are putting in separate pointers along with the key which will actually maintain the data for that key which will be pointers to the actual record, because this earlier in B + tree all records. Finally, appear in terms of the leaf level nodes only they are their pointers come in the leaf level whereas, here the there is no repetition of the search key along the structure. So, they come wherever there.

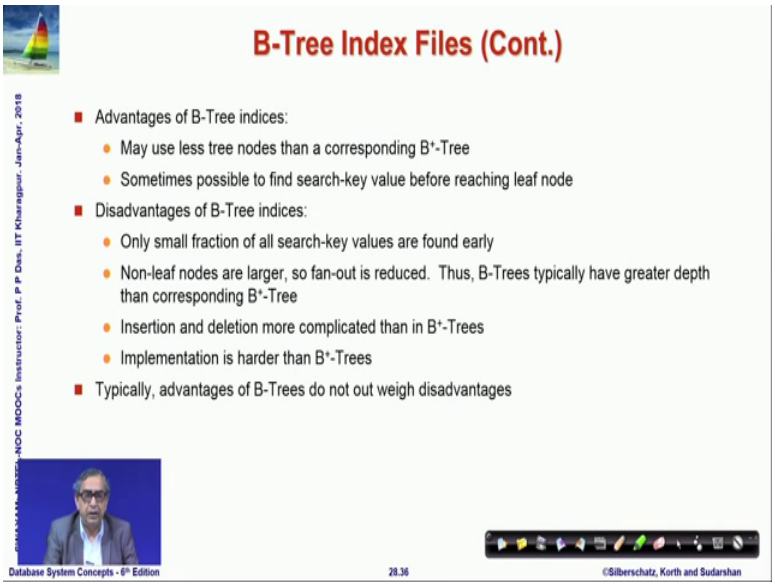
(Refer Slide Time: 20:43)



So, let me just show you an example. So, if you look into this carefully. So, this is what you have seen is a B + tree. So, you can see that Mozart happened here, it also happened here and this is the leaf level. So, from here actually you get pointers to the; to the record for Mozart.

Similarly, Einstein happens here and it happens here Srinivasan happens here in. So, there are multiple times there happening this in contrast is a B tree representation where Einstein, if it happens then alongside with it the pointer to the Einstein's record exists, if brands happen here along with it the brands record exists and Einstein would not happen anywhere else in the tree. So, you do not have the second instance of the Einstein or this instance of the Mozart in the B tree. So, naturally that is the basic optimization that B tree does?

(Refer Slide Time: 21:48)

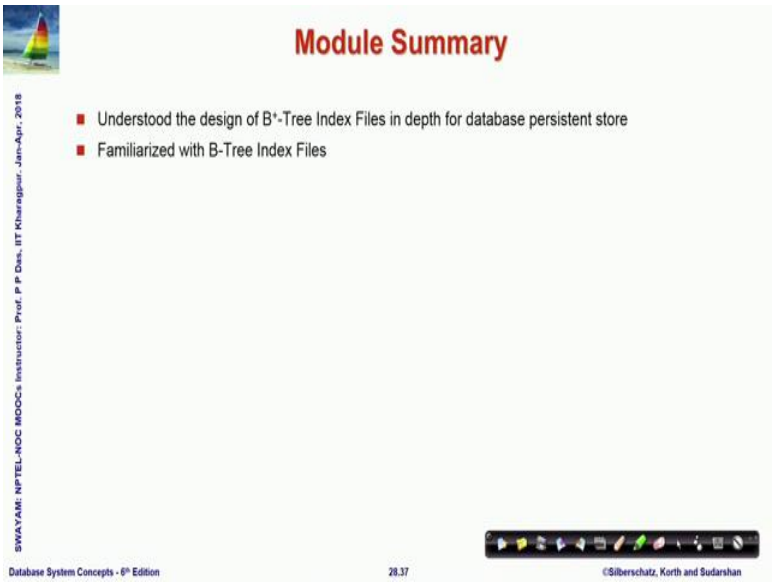


. So, it is advantages it may use less notes than the corresponding B + tree sometimes it is possible to find the search key value even before reaching the leaf node. So, search could be efficient, but it does have a lot of disadvantages, because what happens is only small fraction of all search key values are actually found early non leaf nodes are larger.

Now, because you have pointers to the data as well, so the fan out gets reduced which means that the number of children you can have is gets reduced. So, though you are expecting to get a benefit, because you are not having to go to the leaf every time, but you pay off because your fan out gets less. So, if your fan out get less naturally the tree has a greater depth. Now, because you can you are fanning out less number of children at every node. So, it has a greater depth. So, eventually your cost increases the naturally the deletions insertions are more complicated than in B + tree and implementation is more difficult.

So, typically the advantages of B tree do not outweigh the disadvantages.

(Refer Slide Time: 23:01)



So, it is not very frequent that you will use B trees, but they are use at times, but that is not a very common thing and we stick to B + tree for both of the data file as well as index file storage. So, in this module you have understood the design of B + index B + tree index files in depth for the purpose of data base persistent store and I would again remind you that whole discussion of how B + tree is organized and how operations of access insert delete are done in B + tree. I have introduced them in keeping in parallel with the simpler in memory data structure for this which is a 2-3-4 tree discussed in the last module.

So, while going through the insertion deletion processes of B+ tree, if you have difficulty following I would request that you go back to the 2-3-4 tree that is kind the simplest situation that can have that can occur and understand that and then you come back to the specific points in the B + tree algorithm and also always keep in mind. When you refer to 2-3-4 tree for understanding also always keep in mind that in case of B + tree all node types are same and the basic requirement is every node must be at least half full all the time except of course, for the root and in addition we have also familiarized with B tree and reason that B tree possibly is not a very powerful is not powerful enough it does not give enough advantages so, that to we would like to use it in place of B + tree.

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

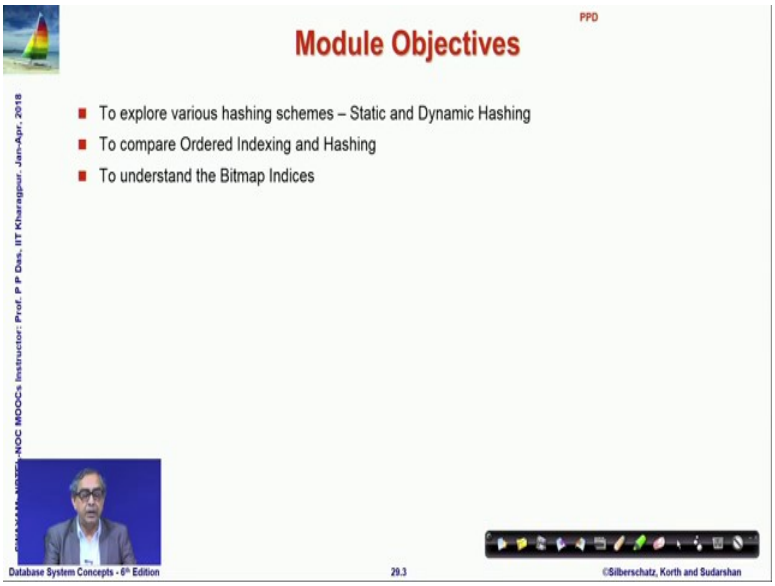
Lecture - 29
Indexing and Hashing/4 : Hashing

Welcome to module 29 of Database Management Systems; we have been talking about indexing and hashing and this is a fourth in the series.

(Refer Slide Time: 00:26)

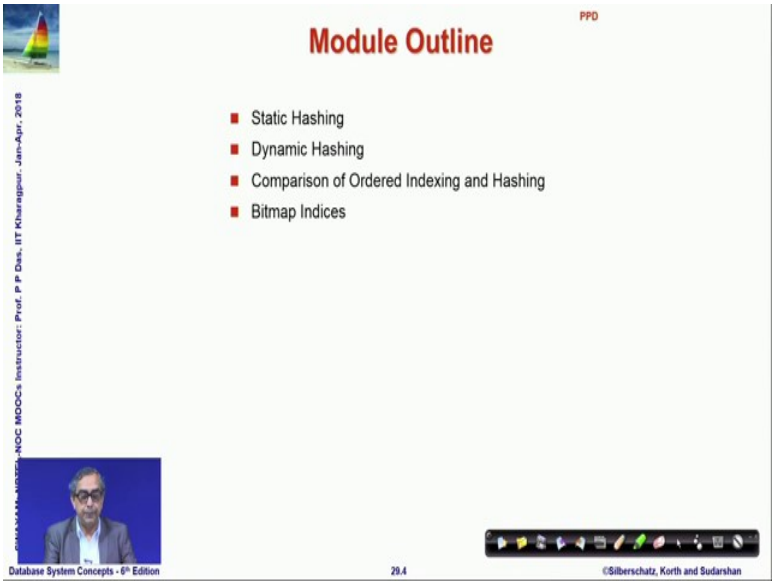
In the previous 3 we have talked about different aspects of indexing and specifically in the last module, we have introduced the most powerful data structure B+ tree for index files.

(Refer Slide Time: 00:39)



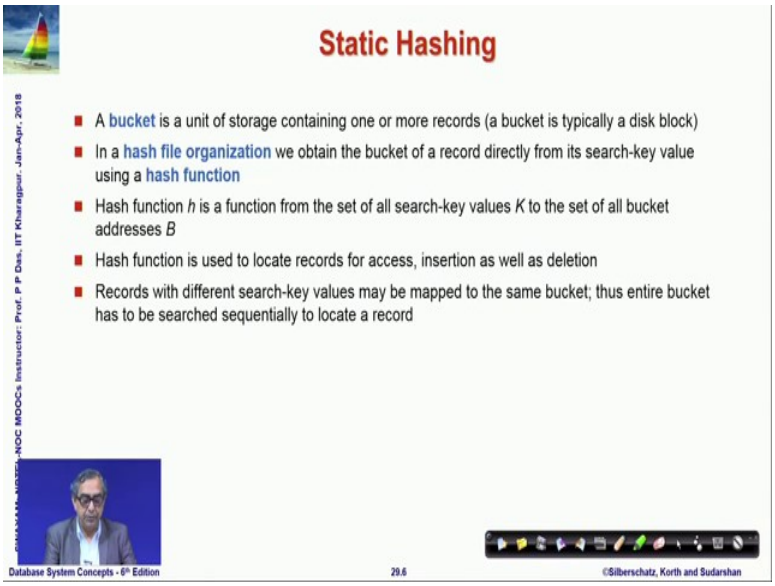
In this module, we will take a look into a explore into various hashing schemes for achieving the similar targets we will look at static and dynamic hashing. And we will then compare it between the ordered indexing that we have discussed already and hashing and we will also understand about what are called bitmap indices.

(Refer Slide Time: 01:03)



So, these are the module outline.

(Refer Slide Time: 01:07)

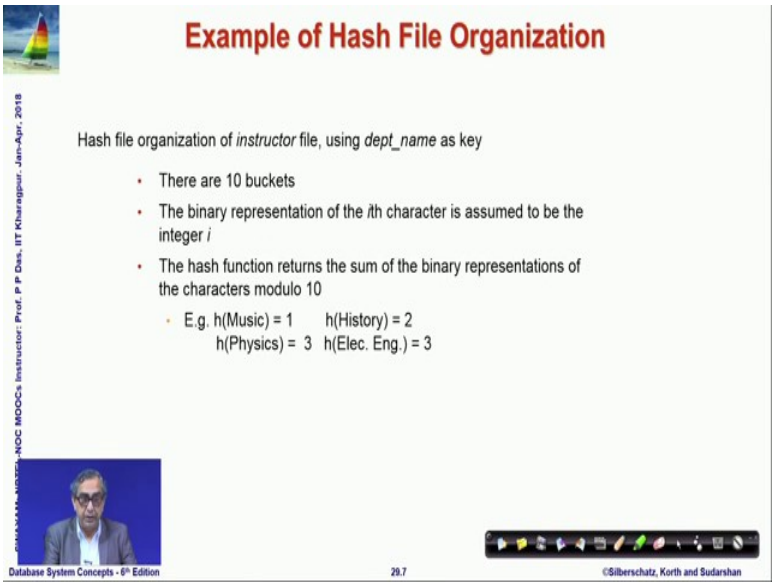


So, static hashing I am assuming that all of you know about basic concept of hashing. So, it what it does a there is a bucket is a unit of storage containing one or more records. So, that is the basic logical concept typically in physical terms a bucket can be a disk block. So, a hash file organization obtains we in a hash file organization; we attempt to obtain a bucket for a record directly from its search key value using a hash function.

So, this is where it becomes very different compared to the ordered indexing for which we saw all this LSM method and the B+ tree where we went through different index structure, but here we want to make use of a mathematical hash function. So, that given the key ideally I should be able to get the bucket in which that particular record containing the key exists that is the requirement.

So, hash function h is a function from the set of all search key values K to the set of all bucket addresses B. So, it is a mathematical function and it is used to locate the records for access insert as well as delete records with different search key values may be mapped to the same bucket right this is not what ideally we wanted, but it is possible there is a entire bucket has to be searched sequentially once you reach there to look at a record, we can make use of other techniques there we will come to that.

(Refer Slide Time: 02:33)

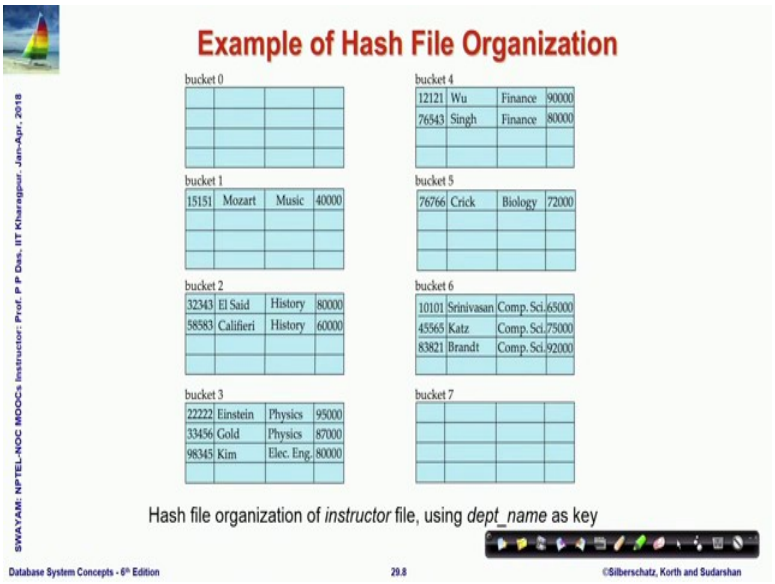


So, let us take a quick example hash file organization of an instructor file using say department named as key. So, we need to design a hash function. So, let us assume that on the address space B we have 10 buckets. So, every bucket is designated by a by a serial number bucket 0 to bucket 9 and we take department name is a key. So, it is a is a character string.

So, we take the binary representation of the i^{th} character and assume it to be the integer I simply every character you take its binary representation and think as if it is an integer. And then as a hash function we add these integer values of binary representations modulo 10. So, M hash value of music we take the binary representation of m which is the ascii code of M capital M; then add the ascii code of u the lower case u and so, on and do that modulo 10 and we get a value which is 1.

So, naturally since we are doing modulo 10 which is the number of buckets here. So, we will get a result for the hash function which is between 0 to 9 which, is a bucket address where it is expected.

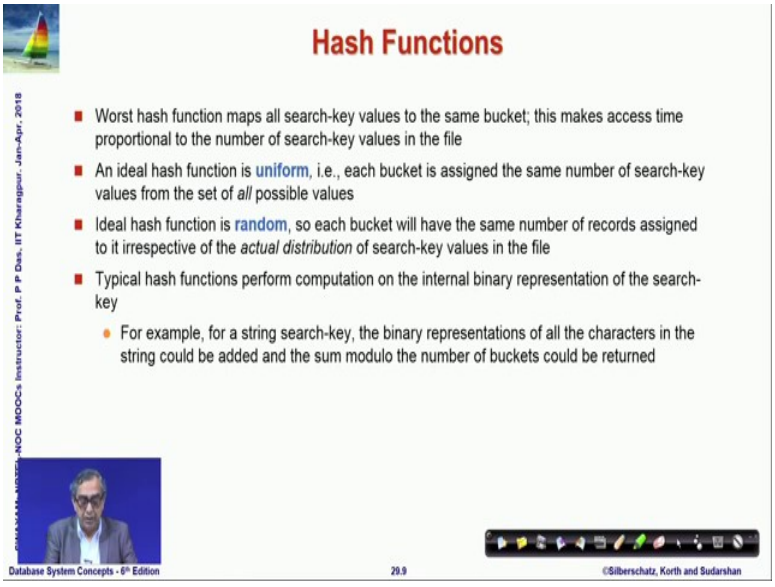
(Refer Slide Time: 03:51)



So, here we are showing an example. So, you can see in the earlier slide we are showing music is 1 history is 2 physics and electrical engineering both are hash value 3.

So, you can see here in bucket 2 since history has value 2. So, those records El Said and Califfieri records go to bucket 2 whereas, physics and electrical engineering both have hash value 3. So, that Einstein gold and Kim came all go to the bucket 3 similarly it happens with the other buckets as well not all buckets are shown here shown only 8 buckets are shown, but in this way we can actually directly map them to the buckets.

(Refer Slide Time: 04:33)



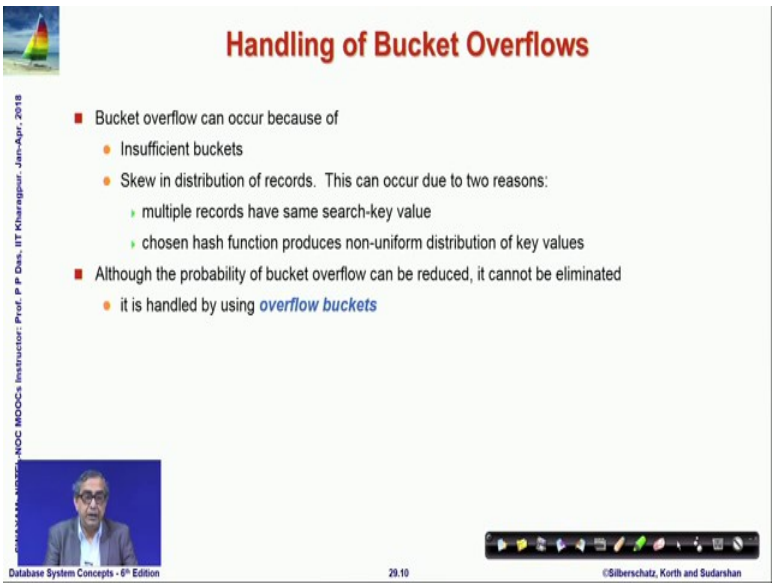
And; so, such a hash function would be really useful now the question is a it is a mathematical function; so, how good or how bad it is. So, we will say that the worst possible hash function is one which maps all key values to the same bucket. So, that everything will have to within them serially; so, that is of no use.

So, the ideal one would be which will distribute the different search keys values in different buckets in an uniform manner to the from the set of all possible values. So, that would be that will be nice to have and ideal would be that if the hash function is random which means that. So, that each bucket will I mean it will generate from the key value it will generate the bucket number, it will generate the bucket address in kind of a random manner.

So, that in a random phenomena; so, that irrespective of what kind of actual distribution the search keys may have the buckets over which the distribute will be more or less the same. So, every bucket will have same number of records things will be balanced.

A typical hash function performs computation on the internal binary representation of the search key that is the basic that that is the one that you have just seen. So, if it is a string then you treat the characters as they are binary representations as integer do some modulo a number exactly what we did in the last case.

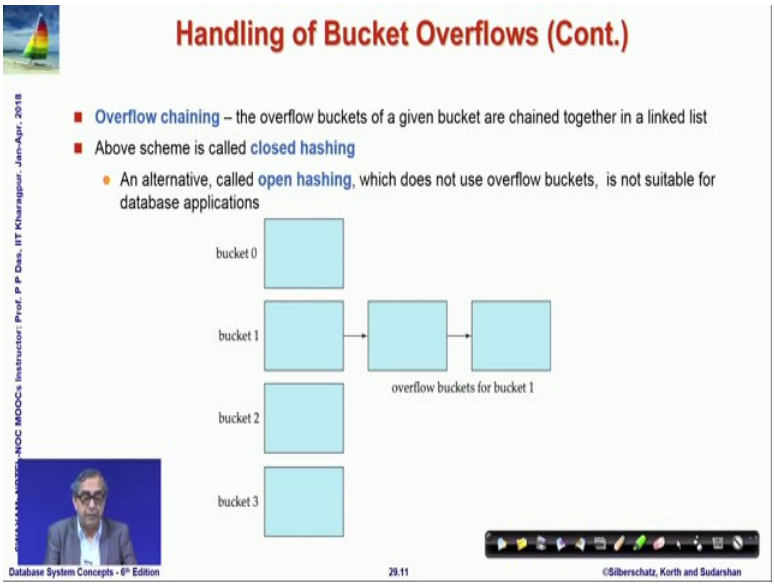
(Refer Slide Time: 05:11)



Now the question is the buckets have a certain size we said that a bucket is a is a disk block. So, a bucket can overflow because there may not be enough sufficient buckets to keep all the records. So, it will not fit in or your distribution could be skewed. So, there may be many buckets where there are lot of space left, but some buckets may have a too many records coming on to it because of the behavior of the hash function. So, that multiple records have the same key value or chosen hash function produces non uniform distribution and so, on.

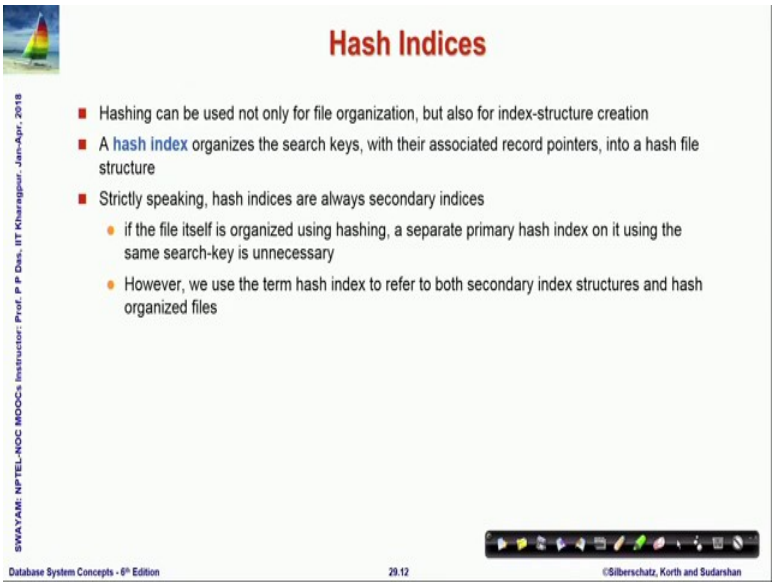
So, if that happens then the probability of bucket flow; bucket overflow will happen and we can try to reduce that, but it cannot be eliminated. So, all that you do is to have overflow bucket which is nothing, but having other buckets connected to this target bucket in a chain.

(Refer Slide Time: 07:04)



So, this is called a overflow chaining as you can see there are 4 buckets shown here and bucket 1 we are saying showing are connected with other two buckets which are the overflow buckets for bucket 1. So, that this kind of a scheme is called closed hashing there is an alternate scheme called open hashing, which does not use a bucket overflow and, but it is not therefore, suitable for database applications and we will not discuss it here.

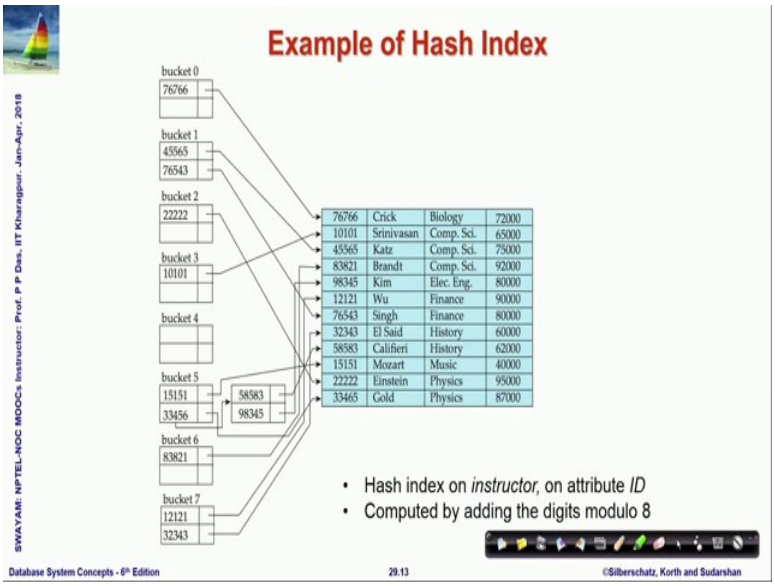
(Refer Slide Time: 07:31)



So, hash indices can be used only for file organization I mean not only for file organization, but they can also be used for indexed structure creation like we did for B plus tree we can use the hash indices for index structure also. So, hash index organizes the search keys with their associated record pointers into a hash file structure exactly in the same way its hashing otherwise.

So, but the you can note that the hash indices are always kind of secondary indices because if a file itself is organized using hashing; then a separate primary hash index on it using the same search keys are necessary. Because if if you are talking about primary hash indexing then it will mean that you are taking the primary search key and creating a hash index on that, but if the file is hash created by hash indexing then that already exists. So, anything that you create in terms of indexing is basically a secondary indexing structure in a hash organized file.

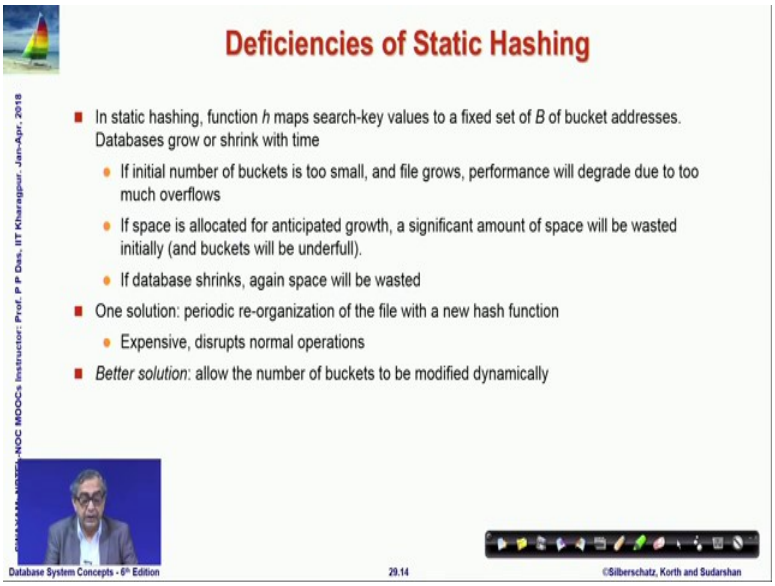
(Refer Slide Time: 08:38)



So, this is kind of hash indexing example. So, here I am I am showing the hash indexing with the ID of this table and the index is computed by adding the digits modulo 8 assuming that there are 8 buckets. So, if you take; so, if you look at bucket 0 then the key that has gone there is 76766 which is 7 + 6; 13, 20 + 6; 26 +6 32 modulo 8 is 0.

So, it goes into bucket 0 it happens that way if, but if we look into bucket 4 you will find that the 4 IDs actually all have this value 5 under the hash function. So, they all need to go to this bucket and therefore, but the bucket size assumed here is just 2. So, after the 2 indices have gone in there a overflow chain is created and another overflow bucket is used to keep the next two IDs there. So, this is how a hash index can be created.

(Refer Slide Time: 09:45)



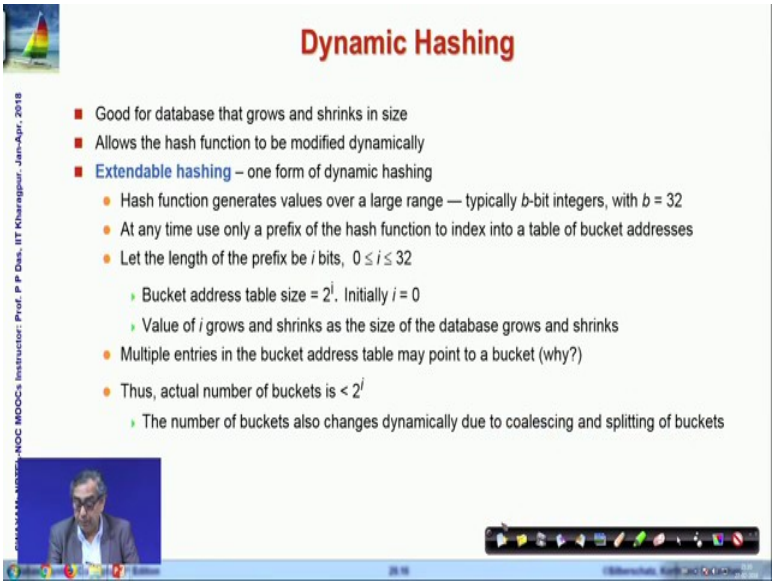
Now, this is this kind of a scheme where you start with a fixed number of buckets and then you design a hashing function which maps the search key values to this fixed set of buckets is known as a static hashing it is static because you start with a fixed number of buckets.

So, yeah naturally the question is what should be this value of B the number of buckets. Now if it is initially too small then the file keeps on growing the performance will degrade because you will have too many overflow chains and if the all advantages of having done the hashing will get lost. On the other hand if you take a too large a B then you will unnecessarily allocate a lot of space anticipating growth, but it may take a very significant amount of time to utilize that that space or also it is possible that it the database at certain point of time grew to a large size and then it started shrinking and then again space will get wasted.

So, static hashing has these limitations. So, naturally what you will have to do is to periodically reorganize the file with a new hash function which is certainly very expensive because it changes the positions of all records. So, it disrupts the normal operation; so, it would be better if we could allow to change the number of buckets to be changed dynamically at the as the database grows. So, if the database grows it can use more and more buckets and if we could adjust this in the hashing scheme inherently; then

it will certainly be better as a solution. So, that gives rise to what is known as dynamic hashing.

(Refer Slide Time: 11:30)



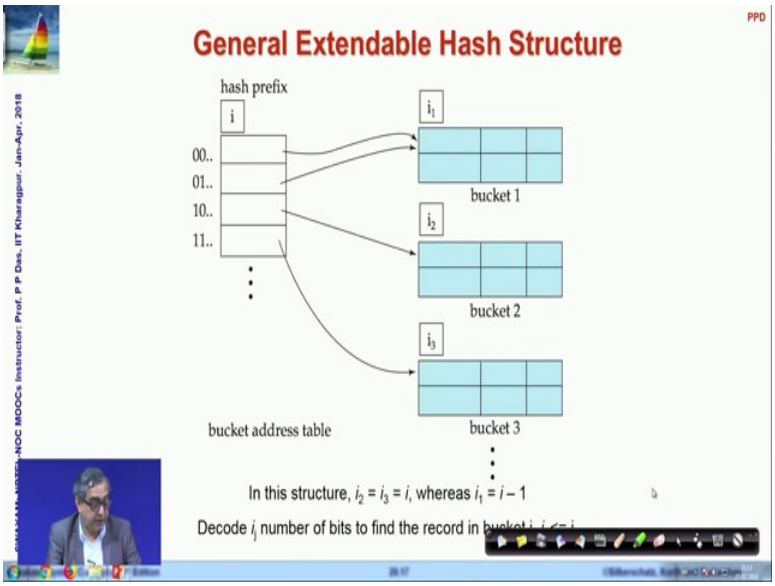
So, it is certainly good for databases that regularly grows and shrinks in size, allows the hash function to be modified dynamically. Of the different dynamic hashing schemes I will discuss the extendable hashing which is a very popular scheme. So, let us see what how it works. So, it at the hash function will generate the value over a large range say typically a B bit integer say 32 bit integer now.

So, what you have is you have generated a value which is hash value which is say over 32 bits, but what you do at any time you use only a prefix of that; you only use a part frontal part of that to index the table to the bucket address and the length of that prefix is i bits; then naturally it could be at least theoretically it could be 0 that is you do not use any prefix and it could be up to that you use all the prefixes.

And so, therefore, if you are using i bits then the bucket address table the possible you know bucket addresses that you could have is 2^i initially you keep that as 0.

So, then the address table will actually point to different buckets let us start moving to an example and see what is happening.

(Refer Slide Time: 12:57)

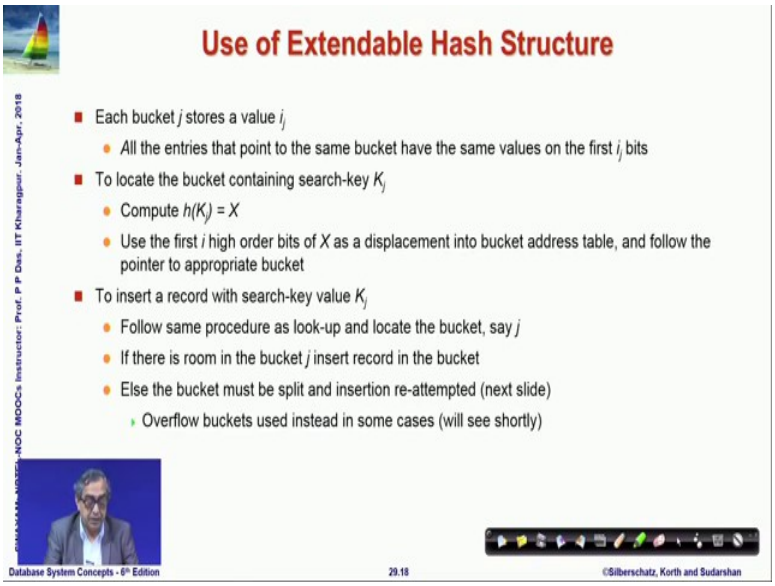


So, this is the general scheme. So, you have a hash prefix which is using i number of bits and therefore, different values of i number of bits. So, there will be 2^i entries naturally you have different buckets here, but you may not actually have all 2^i buckets you may have less than that as it is shown here that bucket 2 and bucket 3 exist, but bucket 1 is a holder for both this prefix 0 0 as well as prefix 0 1.

So, and on on top of every bucket you have a kind of bucket depth given. So, it is a number of bits that you need to explore in the in the representation in the; so, that you can distinguish the different records of that bucket.

Naturally the maximum value of any of these i is the i here, but it could be less than that. So, I am I am sure this is this probably is not making much sense immediately. So, let me move to my detailed discussion.

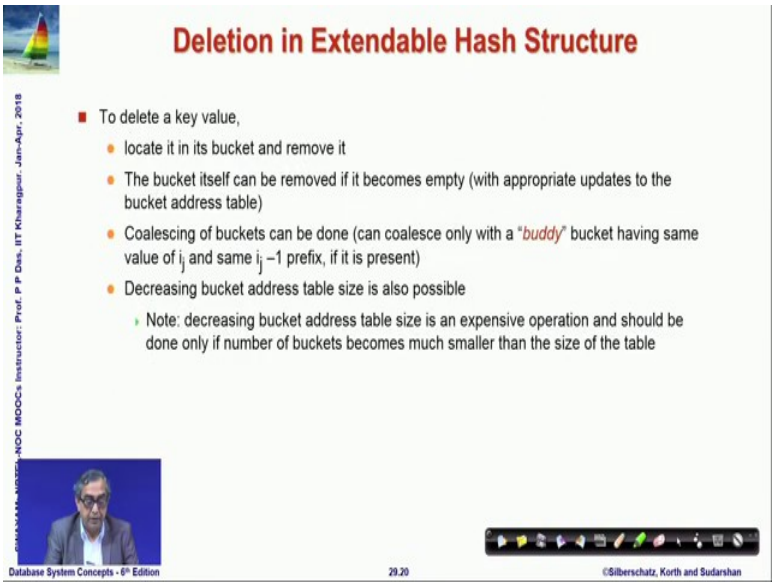
(Refer Slide Time: 14:19)



So, what I was saying that each bucket j stores a value i_j. So, this is the all the entries that point to this bucket has the same value on the first i_j bits. So, this i_j bits are are identical. So, all of them have come to this bucket. So, how do you look at the bucket that contains the search key K_j(subscript)? So, it compute the hash function which is X user prefix i bits of X as a displacement into the buckets address table and follow the pointer to the appropriate bucket. $h_i(K_j)=X$

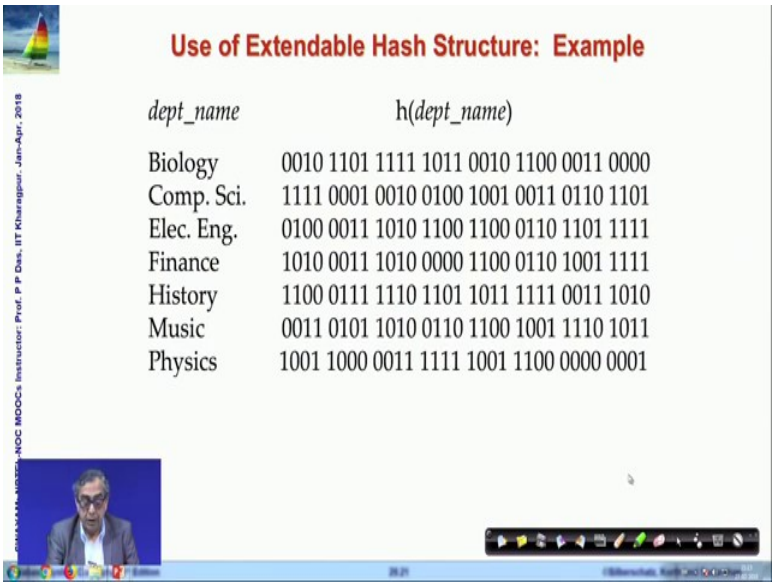
Now if I have to insert a record with a search key K_j(subscript); you will follow that same procedure as a lookup and look at the bucket j and then you will have to look for making some space. So, let me do something.

(Refer Slide Time: 15:15)



Let me before going through this statement of the algorithm.

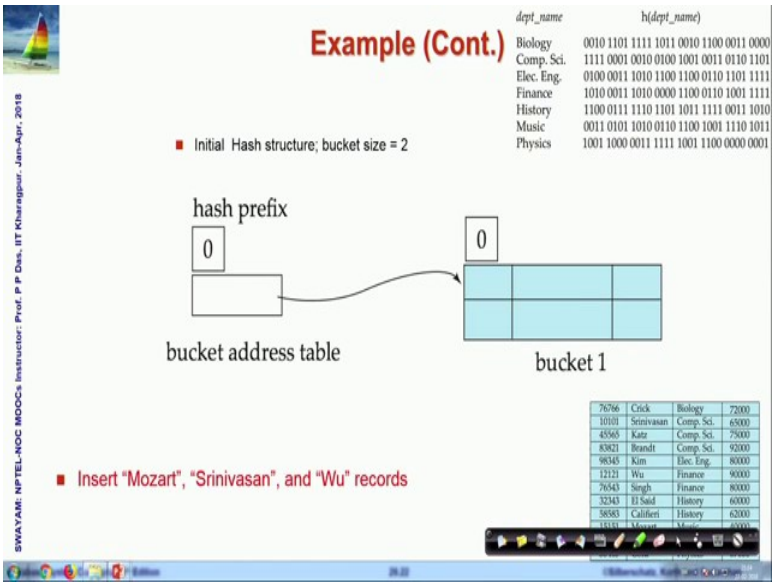
(Refer Slide Time: 15:20)



Let me just go through an example first and we can come back to this formal statement.

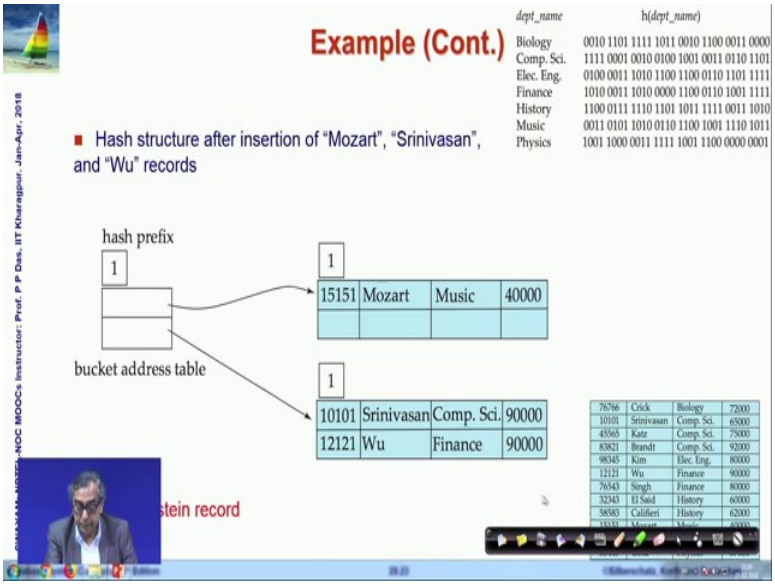
So, what we are trying to do is there is the department names which we are using as a key to do this hashing index and they are represented in terms of the binary representation. So, this is this is the hash of that department name and hashed into you can you can easily see this is 1, 2, 3, 4, 5, 6, 7, 8. So, this is hashed into 32 bit number.

(Refer Slide Time: 16:01)



Now, what do we do? So, initially we start with. So, this is this is all the different hash values that you can see I am sorry this is all the different hash values and this is the table that I need to actually represent. So, initially there is nothing. So, I try to I will try to insert Mozart Srinivasan and these 3 records here. So, let me try that.

(Refer Slide Time: 16:33)



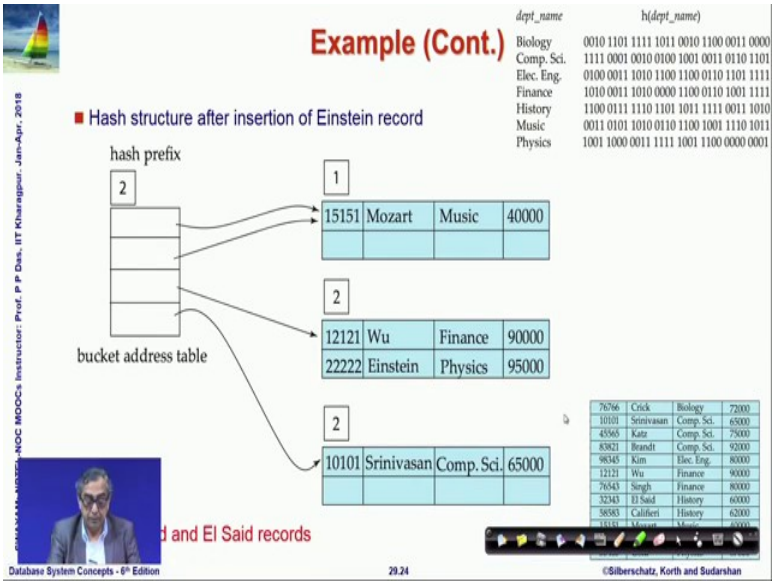
So, if I look at Mozart then Mozart is from the department of music. So, and Srinivasan is from computer science Wu is from finance. So, let us look at this. So, Mozart is from music Srinivasan is from computer science and Wu is from finance. So, these are the 3

now if we look into the prefixes of their hash values; you can see that their hash values are 1 1 and for music it is 0 right.

So, if I use a hash prefix which has just one bit and naturally therefore, I have two entries 0 and 1 then music with the value 0 maps to this bucket where I entered the record for music. And computer science and finance the records corresponding to them has a hash value prefix 1. So, they both map to discipline this is how it can get started. So, you you find out while inserting you find out where is Mozart and based on that you create this.

Now, let us try to insert Einstein.

(Refer Slide Time: 18:09)



So, to insert Einstein what do we find? So, what all we already have? We have music, we have computer science, we have finance and now Einstein comes in Einstein is from physics. So, what would happen when you try to insert Einstein? You already had computer science with 1 as 1 prefix and finance with 1 prefix.

So, you had in bucket two corresponding to 1 you already have 2 records that bucket is full assuming that it can take only 2 records. So, now, you get another which is value 1; so, its value is 1. So, what I need to do? I need to actually expand this now how do I do that? I cannot expand this because there is no more space left. So, all that I need to do is to actually expand the bucket address table.

So, earlier if I if I just if I just go back. So, if I just go back earlier we had only two entries because we are using only 1 bit in the prefix and with that I could not have inserted Einstein oh is from department of physics which also has a 1 bit prefix which is 1 it was. So, I need more space; so, I have increased the prefix level to 2 going here and now naturally I have if I have increases to 2; now I have let me erase this these entries and now I look at for music I look at 2 for physics 1 0, for finance 1 0, for computer science 1 1.

So, now, I find that after I have moved from looking at 1 bit of prefix to 2 bits of prefix now finance and computer science which was earlier together because I was looking at 1 bit now becomes different, but finance and physics both come to the same 1 0.

So, in the hash bucket address table 1 0 you have finance and physics coming in with Wu and Einstein records and computer science which has got 1 1 the Srinivasan record goes to a new bucket which comes from 1 1 here. Now the interesting fact is what happens to Mozart who which was there if you if you remember the earlier structure this is we had only 1 here. So, this was going to Mozart this was 1 and Mozart was here because music had a prefix 0; now music has a prefix 0 0.

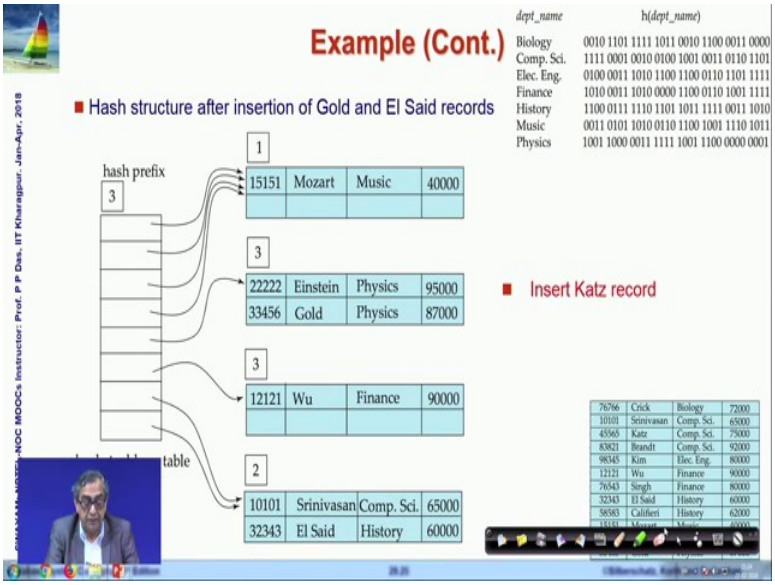
So, what he would have expected? You would have expected that therefore, since 0 0 has come and 0 1 has also come in. So, you would have expected that to have another bucket here which 0 1 is, but then you observe that actually that would be a quite a wasteful to do because you do not actually have a record which has a prefix 0 1.

Out of these two which are we are looking at two prefixes, but you do not really right now need to look at both the prefixes you can still resolve just by based on the first prefix 1. So, you do a simple trick you do not change the prefix level of the particular bucket you say it is 1. Because it is you just need to look at one bit to be able to come to the records in this bucket and the globally it has changed to 2 bits prefix, but locally you keep it as 1.

And with that what you have? You have 0 1 which has a bucket address table entry actually does not have a bucket because there is no records for that. So, you let that point to the same bucket. So, this is a very critical observation that these numbers are basically the local depth; the local information of how many bits in the prefix you need to look at to be able to resolve for coming to this bucket for the records that you currently have.

Whereas this is the global one this is a global maximum that you have. So, naturally local depth cannot exceed the global depth, but if it is equal then you have a unique mapping from the bucket address table entry to the bucket, but if the local depth as in here is less than the global depth; in terms of the number of prefix bits you are looking at then multiple bucket address table pointers actually end up in the same bucket and that is the main principle of this algorithm we can just continue further inserting gold and said into this.

(Refer Slide Time: 23:47)



So, as you try to insert gold and said gold is also from physics which we already had. So, physics and said is from history which we did not have. So, history finance computers let me let me just mark them by the side. So, you have now computer science, finance, history, music and physics.

Now, you will find that you need to you now have physics is 1 0 and you have two records for that and music is continues to be 0 0 ah; obviously, history is 1 1 same as computer science. So, that has to go on this and finances on 1 0 now, but what happens is when you try to do this; you could not have inserted more records because you have run out of space in the buckets. So, again you have run out of that; so, you need to expand in terms of the number of bits that you look at.

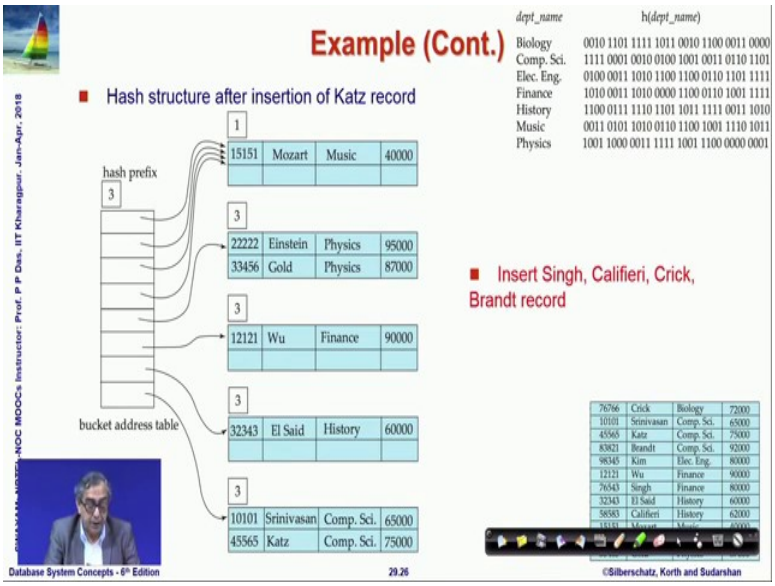
So, you increase that to 3 and now you have 0 0 0 to 1 1 1, but as I have explained not all buckets really need to look into. So, many bits Mozart this bucket continues to B with a

local depth of 1 because if you look into all these 4 different cases; then music is the only one which has a prefix 0, everyone else has a prefix 1. So, if I know that it is 0 then it comes to only this bucket and nowhere else consequently all these 4 bucket address pointers actually go to this bucket table.

Whereas these two for physics I have 1 0 and for finance we have 1 0 here and these come to. So, physics now is looking into 3; so, it is 1 0 0 finance is into 3 it is 1 0 1. So, both physics and finance go to different buckets; now coming to computer science it is 1 1 1 and there is no. So, computer science is 1 1 1 and there is no 1 1 0. So, the 1 1 0 bucket address table pointer continues to point to the same bucket and the local depth value is just 2 <3 in the global table.

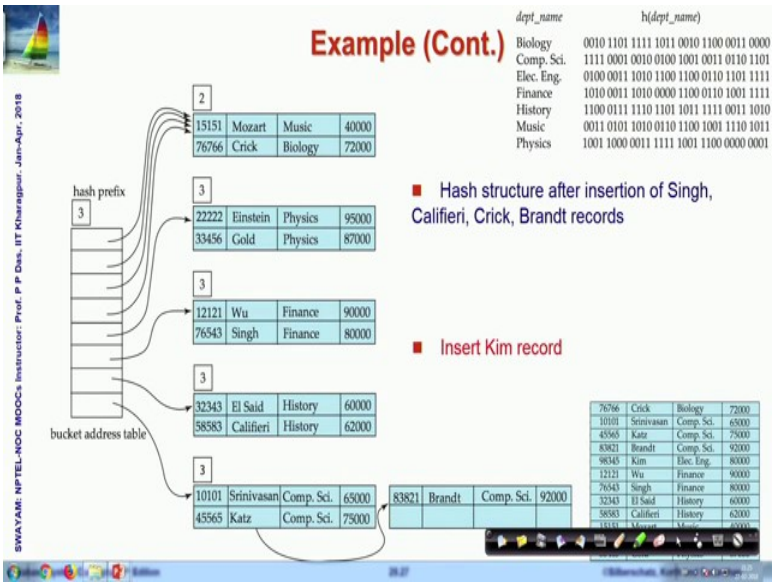
So, this is the basic process of doing dynamic hashing. So, I will not ah; so, the whole example in terms of this table I have given here worked out.

(Refer Slide Time: 26:49)



So, you can just go through every step and try to convince yourself.

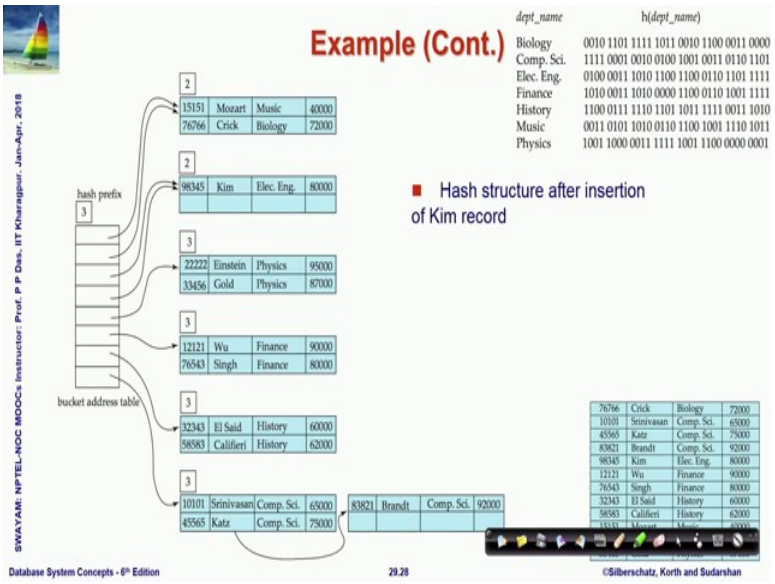
(Refer Slide Time: 26:54)



This is an interesting case that happens here where again you come to computer science professors to be entered. So, at level 3 of prefix you have all of them have prefix 1 1 1. So, you would have required to split or increase the prefix level globally the prefix level to 4, but assuming that there is an upper bound on the number of prefix levels; you can do which decides the size of the bucket address table. If that is given to be 3 you certainly cannot increase it further; so, all that you will have to do is actually do a kind of an overflow chain here as well.

So, all of them are 1 1 1 here which brings you to this you cannot find it you go to this and all 1 1 ones in future will have to be. So, it is a its kind of a tradeoff between what is the size of the global depth, how many prefixes globally you would like to look at what is the size of every bucket that you will have to maintain and what is the kind of chaining that you will have to accept because of that. So, this is what happens particularly.

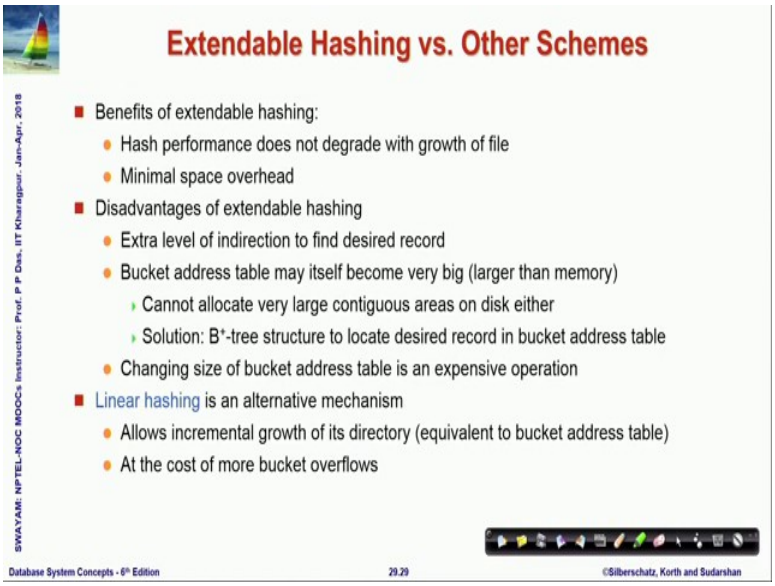
(Refer Slide Time: 28:05)



So, you can continue in this way and this is a final table where all things have been hashed well. So, this is the basic extendable hashing scheme it has in this the performance does not degrade with the growth of the file and there is very minimal overhead of the space. But it does have disadvantages for example, there is a extra level of indirection to find the desired record because it the hash then come to the hash bucket address table and then go to the bucket address table itself may be very big because it is exponential in the size of the number of beds.

So, it could be larger than memory if that. So, much of you know a contiguous allocation may not be possible. So, you will need to have another possibly a B + tree structure to locate the desired record in the bucket address table first. And then changing the bucket address table will become a quite an expensive operation. So, the growth will become.

(Refer Slide Time: 28:13)

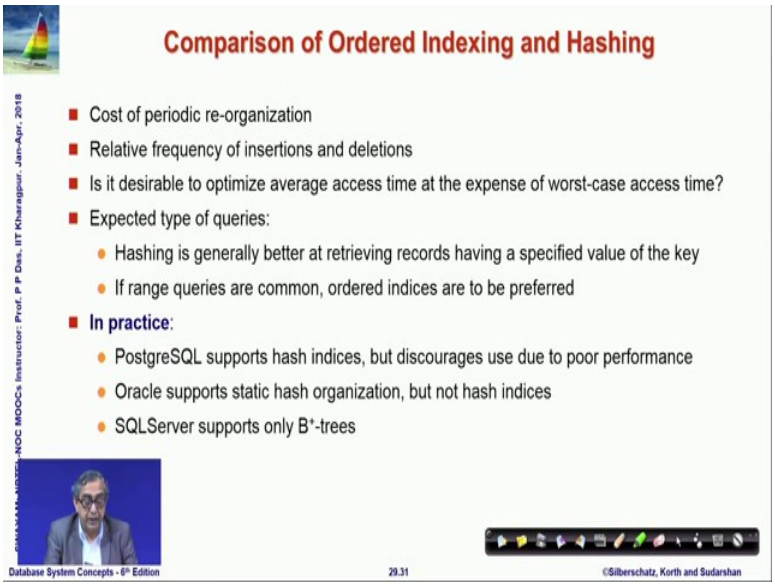


So, there are several disadvantages that also this scheme has. So, another alternate is to use a linear hashing allows incremental growth of his directory at the cost of more bucket overflows of course,.

(Refer Slide Time: 29:25)

I would quickly try to compare the two major schemes that we have discussed.

(Refer Slide Time: 29:30)



The ordered indexing and the hashing now naturally ordered indexing has suffers from the cost of periodic reorganization. And because the indexing will have to be maintained the hashing is better in terms of that relative you will have to look at the relative frequency of insertion deletion that decides much of the cost between going between these two schemes.

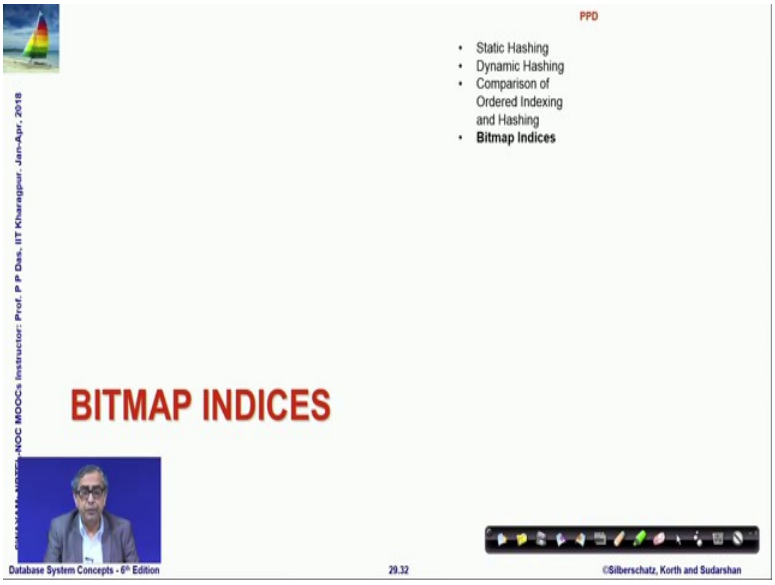
You will have to see is it desirable to optimize average access time at the expense of worst case access time. For example there could be several ways to organize; so, that your average become your worst case may be really really bad, but as long as your averages is very good you should be happy about it. So, those kind of hashing schemes should be more preferred. So, you also depends on the kind of expected type of query.

So, for example, hashing is better in terms of retrieving records which have a specific value of the key because you can directly map from that key to the bucket. And if range queries are common then as we have seen ordered indices would make it make much better sense because in terms of the ordering you can quickly get all the required records at the same physical location nearby physical location.

If you would like to understand as to what the industry practices are it is very mixed. And if you just look into 3 of the very common database systems PostgreSQL does support hash index, but recommends does not recommend it because of the poor performance oracle supports static hash organization, but not hash indices SQL server

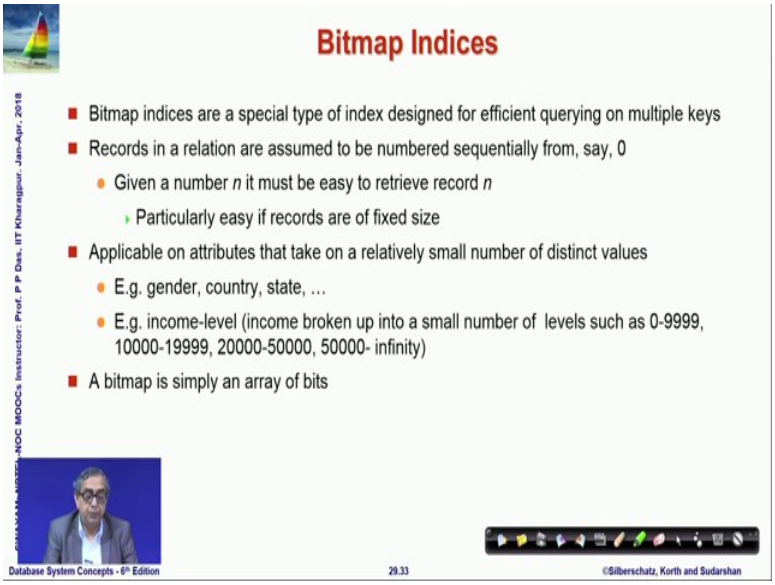
supports only B + trees no hash index space scheme. So, of course, you can see that there as the community is mixed in terms of it is a reaction to whether its indexing or hashing, but hashing powerful at least in limited ways is a powerful technique to go with.

(Refer Slide Time: 31:35)



The last two that I would like to just quickly remind I mean take you through is what is known as bitmap indexes.

(Refer Slide Time: 31:43)

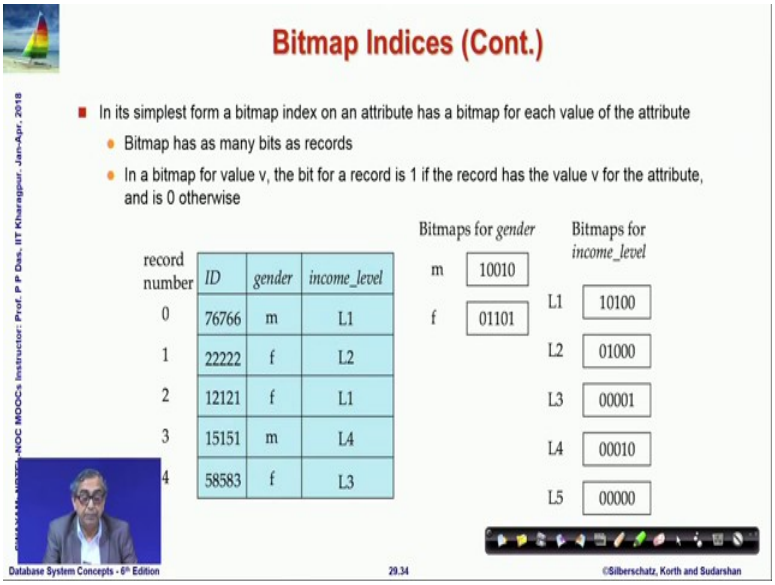


Bitmap indexing is a very simple idea. So, what you it is a special type of indexing which is designed for querying on multiple keys; the basic idea is that if let us assume

that all records in a relation are numbered from 0 to n and let us say that you are talking about attributes which can take very small number of distinct values.

So, bitmap indexing is not for any attribute. So, consider attributes such a very small number of distinct value say gender which has two possible values or few possible values the country state. So, take those or maybe you can you can nominally bucket a range of numbers source income level 5, 6, 10 income levels. So, small range of possibilities and bitmap is simply array of bits. So, take an array of possible array for the records and for the possible values you mark 1 or 2 0.

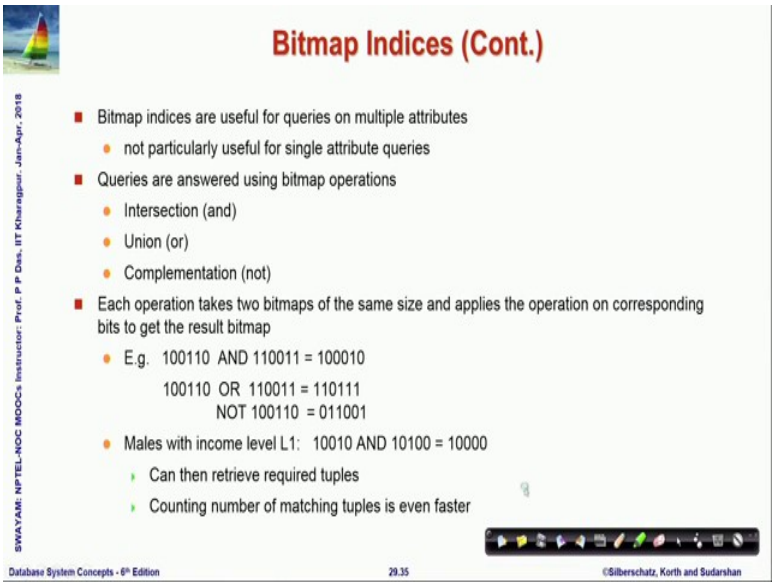
(Refer Slide Time: 32:39)



So, this here is an example showing it. So, we are showing bitmap index for gender. So, you have a array for m the male gender and f female gender and if you look into the record 076766 has male under m gender m. And therefore, in the male gender bitmap index the first bit is 1 in f it is 0; so, actually m and f are complimentary.

Similarly, for the income levels you have 5 different bitmaps encoding; the 5 different possible levels in the income that you can have.

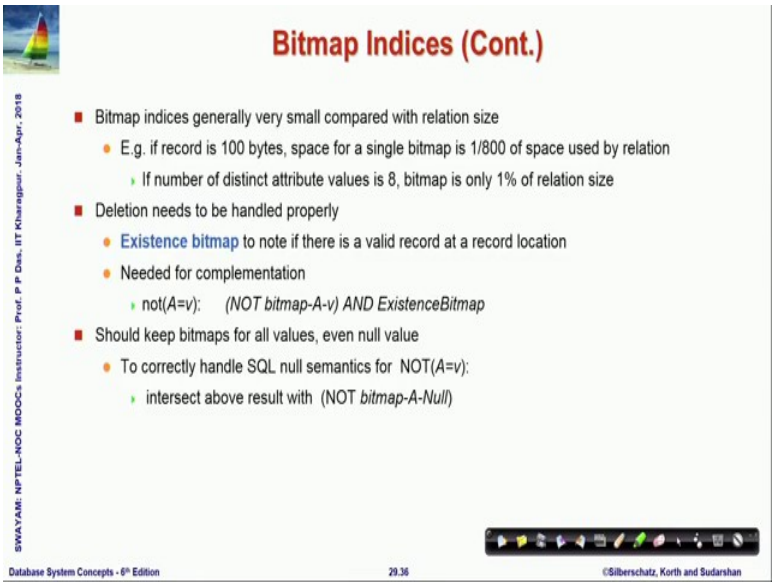
(Refer Slide Time: 33:21)



Now the big advantage of bitmap indices are doing different queries on multiple attributes. And for example, the often queries have intersection union and they can be simply computed in terms of bitmapped operations. So, if you have two different values to be two conditions to check in terms of bitmap indices; then you can just make there and whatever satisfy.

So, say if you are looking at males at for example, here males at income level L 1, then you can you can just take the bitmap for gender and bitmap for income level and do the ending and you get that the first record has value 1.

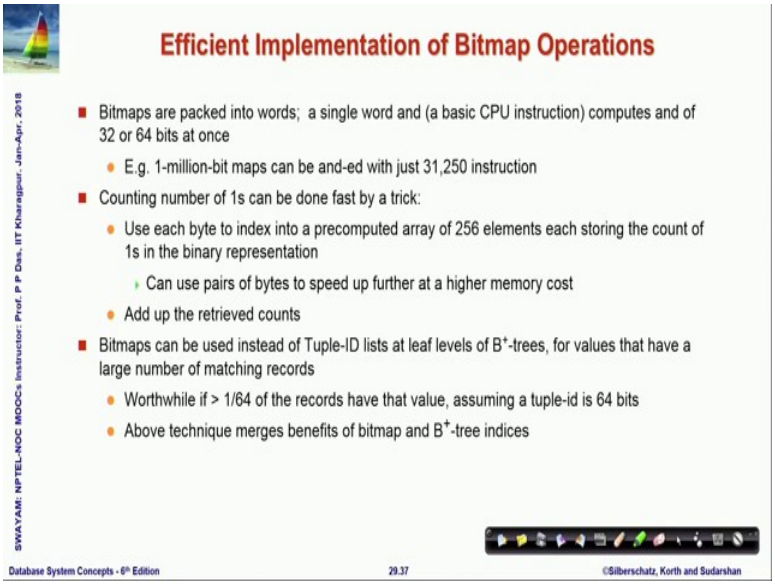
(Refer Slide Time: 34:08)



So, that is answer and you can quickly come to that. So, bitmap indices generally very I mean naturally they are they are they are small in compared to the relation size because you are doing bitmap indexing only if the attribute can take small number of distinct values.

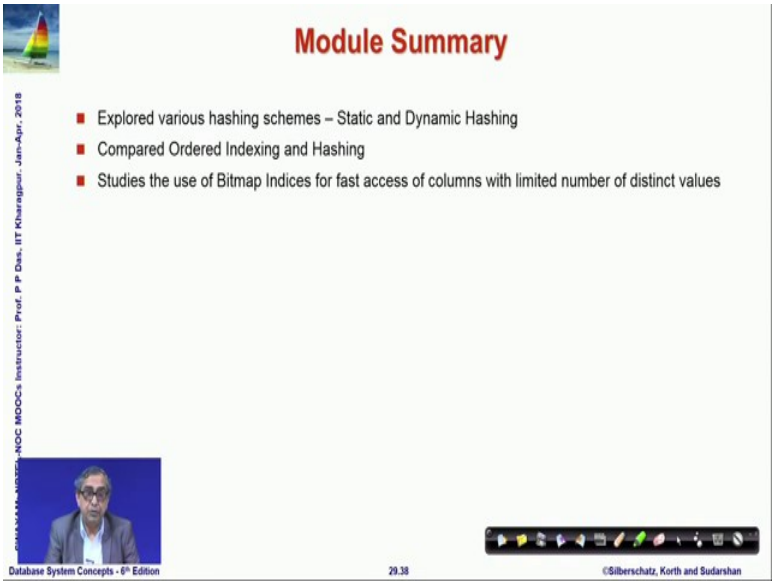
Of course, the deletion has to be handled properly look at this and should keep bitmap for all values even if there are null values you must keep that otherwise you will lose track of that.

(Refer Slide Time: 34:33)



And there are several efficient implementations some information I have given, but is we do not want to go in much depth here.

(Refer Slide Time: 34:45)



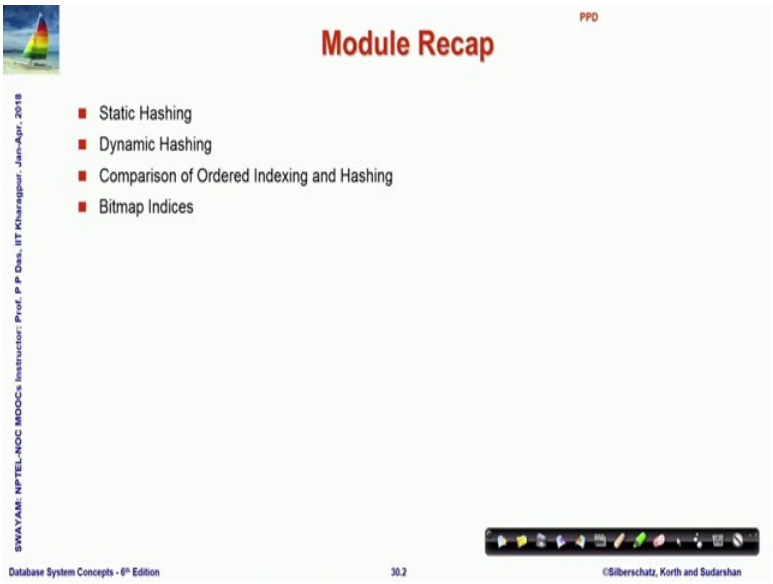
But several compression techniques are possible in terms of bitmaps; in the next module I will talk little bit more about how to use that. In this module to summarize we have talked about various hashing schemes static and dynamic hashing, compared the order indexing with hashing and introduced the notion of bitmap indices.

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

Lecture - 30
Indexing and Hashing/5 : Index Design

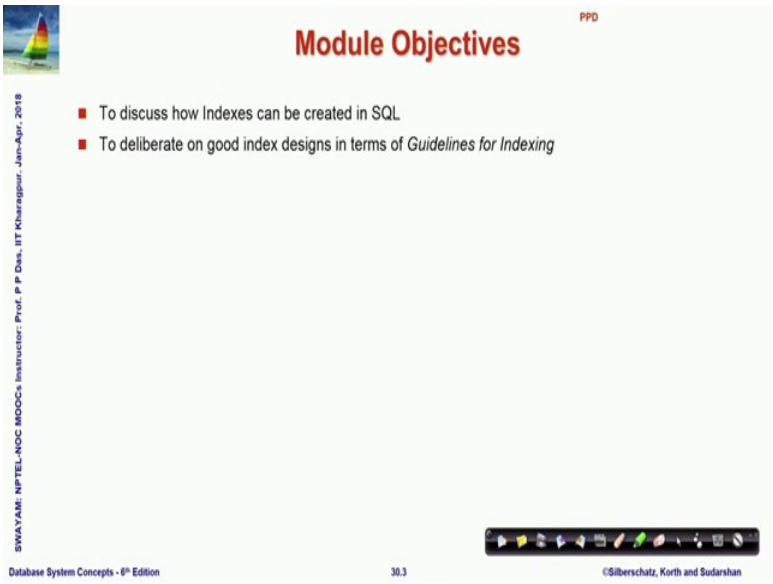
Welcome to module 30 of Database Management Systems. We have been discussing about indexing and hashing and this is a concluding module on that.

(Refer Slide Time: 00:27)



We have in the last module discussed about different hashing techniques static and dynamic and compare that in introduce bitmap indices.

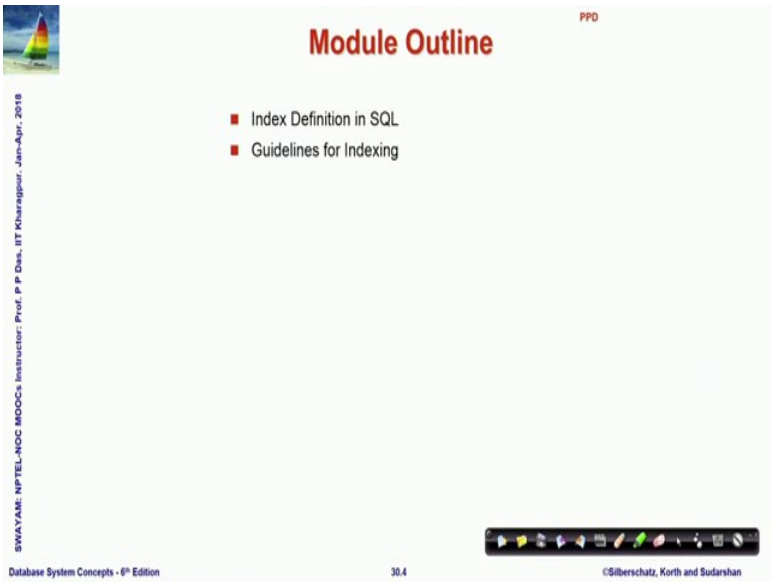
(Refer Slide Time: 00:35)



Now, in this module we would specifically look into the use cases, we will check as to how indexes can be created in SQL first. Because we will have to use it you have already known the theory of various different indices and so, on so, how do you actually tell the system to index.

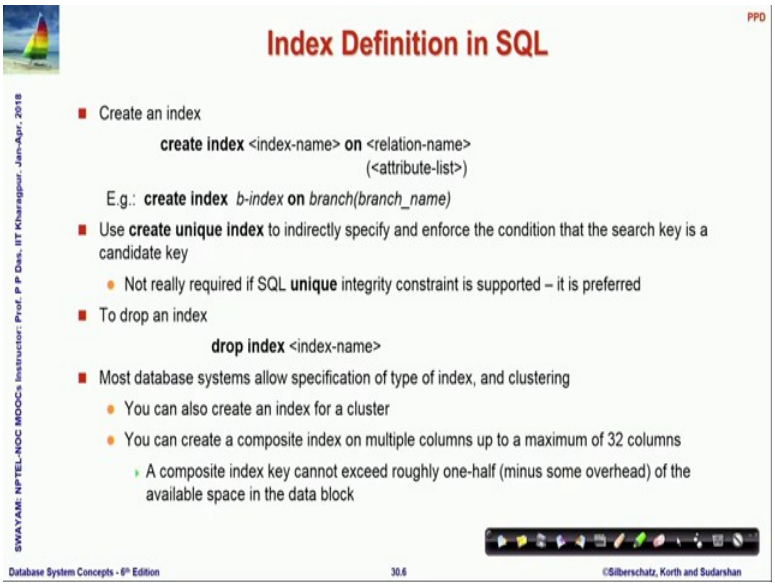
And the second is the important thing as to when should you index and on what. So, we talked about a few guidelines for indexing.

(Refer Slide Time: 01:05)



So, that is that is what we want to learn.

(Refer Slide Time: 01:09)

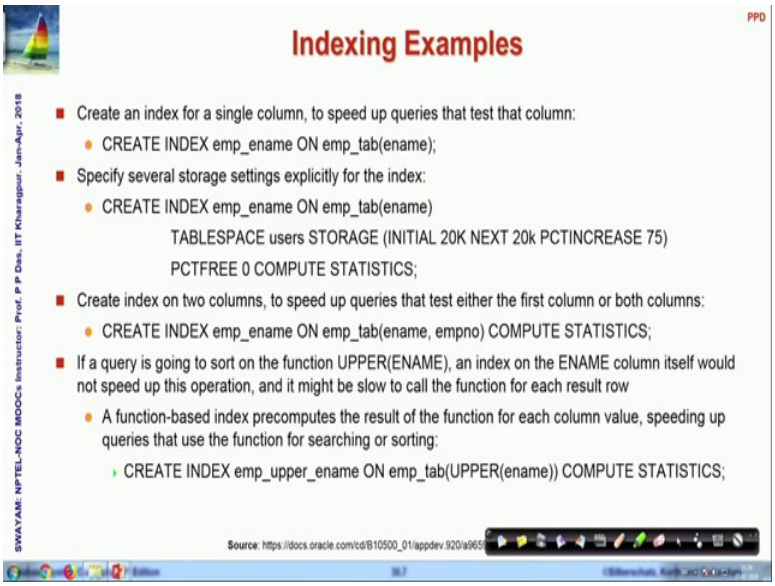


So, index can be defined in SQL in very similar syntax as you create a table. So, you say create index put a name for that index and then you say on which relation are you indexing and put the list of attributes on which you are indexing. So, if branch can relation can be indexed on branch name and I may call that b- index.

Now, there is a way to also express create unique index if you say create unique index and it will expect that the search key is a candidate key because I mean in the sense it all values of that will have to be distinct unique. Now, this used to be very common to do this kind of indexing earlier, but now it is more preferred that you can use unique integrity constraint in terms of the create table which we have already discussed. And that will ensure that you have that kind of a condition satisfied and you may not create unique index for that.

If you do not want an indexes actually do drop index and put the index name. So, most database system allow specification of the type of indexing and clustering that you want to do. So, you can create an index for a cluster also and you can create index for composite index for multiple columns.

(Refer Slide Time: 02:32)



So, let us run through quickly couple of examples create an index for a single column to speed up queries the test that column. So, we are saying employee emp_tab is a is a relation which has an attribute ename the employee name and we want to create an index on that if we do that then any search that is based on the employee name will become really fast.

Now, while you create the index you could also use optionally various other factors which relate to particularly the storage setting you can set what is you know what is the storage you want to keep for the index how you would like to increase increment that and so, on what is the table space. And very most interestingly you could say that compute statistics now this is something which is optional, but is very useful. For example, if you are not sure as to how your data is getting distributed in different relations and how really they are queried you would not know whether an index is good or it is inappropriate.

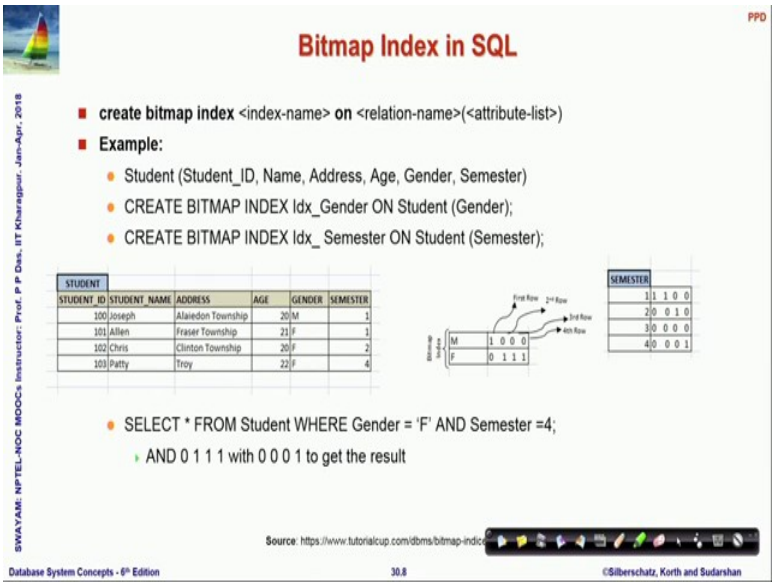
So, it is good to actually compute that statistics in terms of the index that you want to know that by doing this index what kind of accesses have happened? So, compute statistics will tell the database system to keep on computing this which you can subsequently refer to. You can create index on two columns also; so, here we are showing one where emp_tab is indexed on employee name emp_ename and employee

number together. So, you saying that you create an index on both of these and compute the statistics at the same time.

Now, other ways there are index that can be created on functions. So, suppose if there is a query which going to sort based on the uppercase writing of the e_name. So, if I just index the e_name then that itself would not speed up the operation because while you want to sort then the e_name will have to be changed into the upper case by upper and that is every time will have to do that for every record and then actually apply the sorting comparisons.

So, that will become a slow process, but you can do a function based indexing where you can specify as you can as you can see here the function based indexing where you say that you index based on upper e_name. So, what will happen your actual values are in impossibly lower case or mixed case, but your index emp upper e_name will get created on the in the order of the upper case of e_name and will be very useful in terms of the sorting later on.

(Refer Slide Time: 05:36)



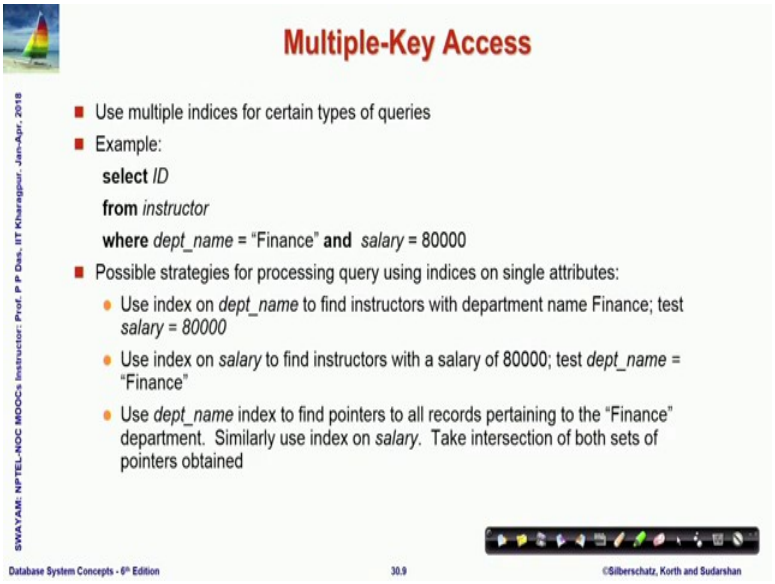
Now, you can like the normal index you can also create the bitmap index. So, you just say create bitmap index on the name and rest of the structure is similar. So, if there is a student relation which has these fields I can we can create an index on gender; we can create another index on semester these are very typical candidate for bitmap index

because gender can take 2 values here male and female semester can take 4 values 1, 2, 3, 4.

So, the bitmap are shown here and then if I want to do a select where the gender is F and semester is 4; then it is basically anding the bitmap of F which is 0 1 1 and the bitmap of semester 4 which is 0 0 1. So, if we if we and these two we will find that we have the result which is 0 0 0 (011 AND 001). So, which tells me that student id the fourth record of the student ID 103 is a result.

So, this is how bitmap indexing can be used in SQL.

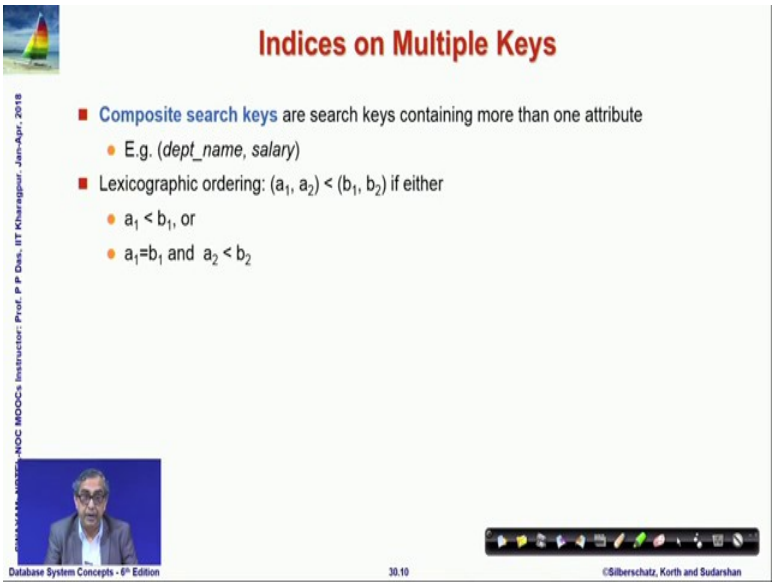
(Refer Slide Time: 06:50)



And actually this the whole thing can be used subsequently in multiple key access for example, if you are doing a query where it is you have department name is finance and salary is 8000, then there could be several strategies for processing this query using the index values for example, if you have single index on single attributes. So, use you can use the index on department name to find instructors which have department and finance. And then test if the salaries 80000 or you can use index on salary to find instructors with salary 80000 and then test if department name is finance.

Or you can use department name index to find pointers to all records that part in to finance department. Index on salary to find all records that part in to 80000 salary and then take intersection of the both sets to get the final result.

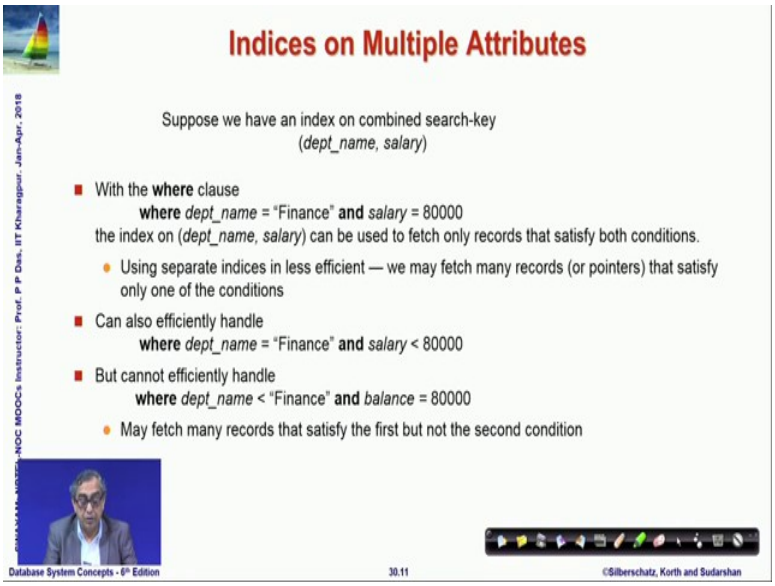
(Refer Slide Time: 08:04)



So, multiple key access could be achieved in terms of various single indexing single attribute indexing also; When we are doing composite search keys then naturally if there are 2 then the indexing means that you will have to define a combined lexical order. So, department name salary means that either department it is it is ordered to indexes are ordered in terms of just the department name.

Or if the department name is same then they are ordered in terms of salary this ordering in which you write the attributes in the multi composite search key is very important because you can see that for the two if the department name is same then the salary will be compared, but not the other way around.

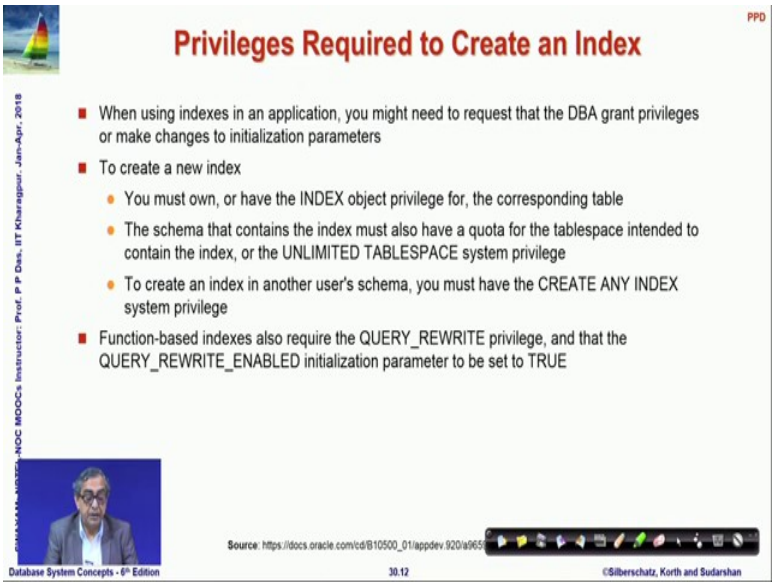
(Refer Slide Time: 08:50)



So, when you have index on multiple attributes say again going back to that same example of department naming finance and salary being 80000. So, using separate index is less efficient though we saw how that can be done, but we can also efficiently handle if we have this indexing on department name and salary. Then we can also easily handle queries like department name is finance and salary < 100; it is not because as you can easily figure out because if I can find the equality then I also know what is less.

But note that we cannot efficiently handle if I say that where department name is less than finance and balance I am sorry this should be salary is 80000. The reason is that the ordering of the attributes in this composite key is department name salary. So, if there is if department name is less than is there is no way to check for the equality of salary, but if the department name equals then there is a possibility of checking on the salary. So, because of this ordering this will may fetch many records that satisfy the first one, but not the second condition.

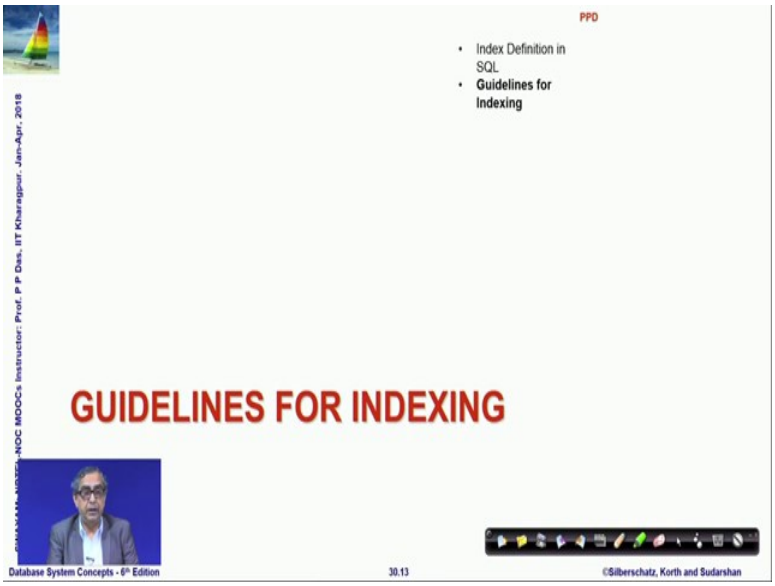
(Refer Slide Time: 10:16)



Now, you should also remember that you need a special privilege to create an index because this is partly in the domain of the administrator's job. So, you need the specific privileges access rights to be able to do that. So, to create a new index either you have to own or own that that those set of tables on which you are creating the index and or have the index object privilege for those tables or the schema that contains the index might also have a quota.

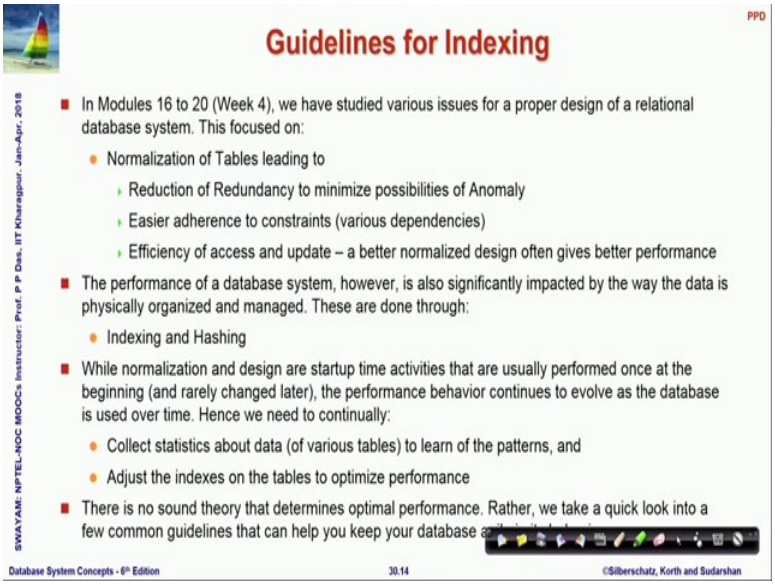
So, that you can because creating the index means you are will be using the temporary tablespace on a on a regular basis. And, but with this you will not be able to create index in some other user schema for that you need a global right which is the create any index kind of system privilege. So, also check if you are not being able to create an index check what is your privilege that exists function based indexes require other privileges; please check on that.

(Refer Slide Time: 11:28)



Now, let us. So, we have seen how to create index how to use that in terms of the SQL application SQL query system.

(Refer Slide Time: 11:43)



Now, we will quickly take a look into why how should we index and where. So, if you recall in the modules 16 to 20 in the week four we have studied various issues of a proper design of relational database system, we focused on normalization of tables that can reduce redundancy and minimize anomaly how can we easily adhere to various

constraints how to improve the efficiency of access and update a better normalized design often gives better performance.

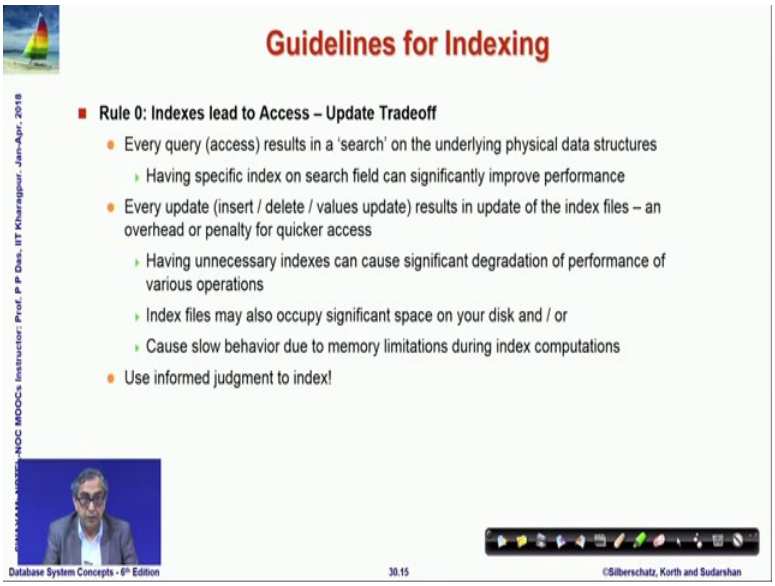
For example, we optimizing the minimizing the requirement for computing join and all those. So, those advantages we have saw, but the actual performance of a database system is significantly impacted by the way the physical data is organized and managed which does not come across in terms of the logical design that we have seen. So, these are what are being achieved in terms of indexing and hashing. So, this is where we we need to understand the actual boundary to the physical organization and that is what we have been trying to do.

So, if you think back while you are normalizing at the design level. So, those are the startup time activities; so, usually we will design and normalize and you know make the create table and do all that at the beginning of a database system. And it is really it will really be changed later because it will have severe implications, but the performance behavior will continue to evolve will continue to change because the design does not tell you exactly what the statistics of that data would be what the behavior of the data would be. So, it will evolve as data base is used over time.

Hence you will need to continuously collect statistics about the data of various tables to learn of the patterns as to which table is getting heavier which where what are the attributes on which more accesses are happening, where what kind of queries you are getting and you have to adjust the indexes on the tables to optimize the performance.

So, that is the whole requirement all about unfortunately unlike the functional dependency or multivalued dependency theories that we studied in the design space; there is no sound theory that determines optimal performance. So, all that have is more and expertise that you develop through experience. So, what I will take you through are a set of few common guidelines about how to keep your database agile while you are you go through the life cycle of different data coming in and going out.

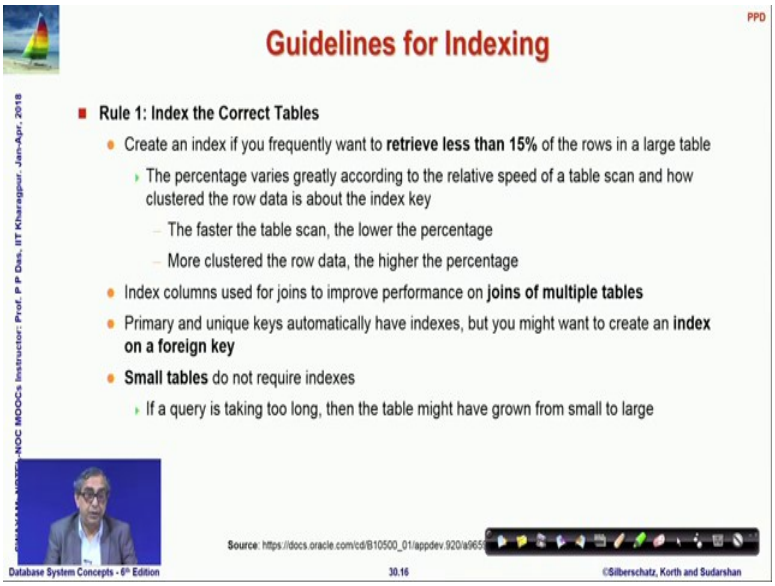
(Refer Slide Time: 14:23)



So, the first rule I say rule 0 is the indexes lead to access update tradeoff we have already seen this at every query results in a search in the underlying physical data structure as we have understood. Having specific index on search certainly can improve performance, but as we have already noted every update with the be it insert, delete or update of values will result in update of the index files.

So, it is an overhead or penalty for quicker access that we are paying. So, having unnecessary indexes can cause significant degradation of performance. Index files will also occupy significant space on your disk and it may actually cause to slow down your behavior due to memory limitation during index computation. So, rule 0 indexes lead to this trade off always be watchful about that use judgment to index.

(Refer Slide Time: 15:26)

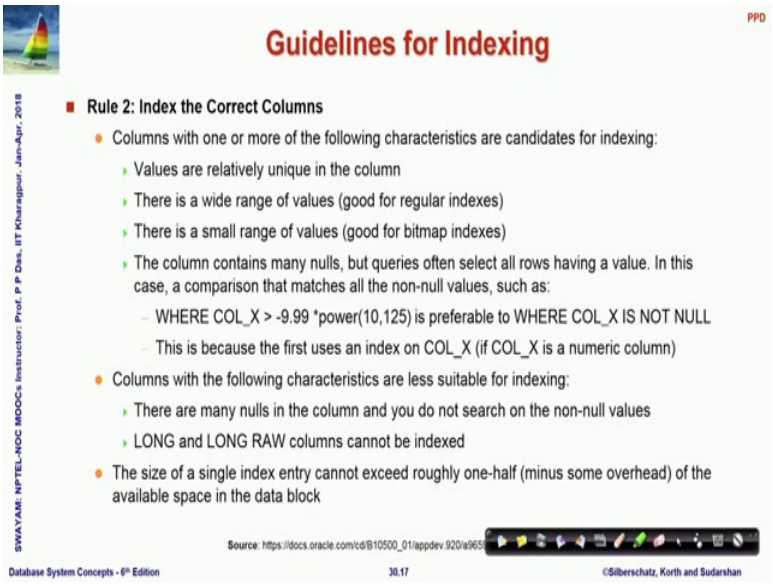


Rule 1 index the correct tables decide which tables are best candidates for index to creating an index if you frequently want to retrieve say less than 15 percent of the rows in a large table. Now first 15 percent is a ballpark number this can vary greatly according to the relative speed of the table scan ah.

Fast of the table scan you can use a lower percentage of cut off more cluster the row data you can use a higher percentage for cut up. Index tables index columns used for joining multiple tables if you have situations or multiple tables are used in joins on a on a moderately regular basis then the columns are used in the join in these tables; these tables should be indexed based on those.

The primary and unique keys automatically have indexes, but you might want to you have an index on the foreign key. So, consider that small tables do not require index if a query is requiring unnecessarily long time or unexpectedly long time; it is time to check if the table has become really big compared to small and it might be term to index that.

(Refer Slide Time: 16:45)

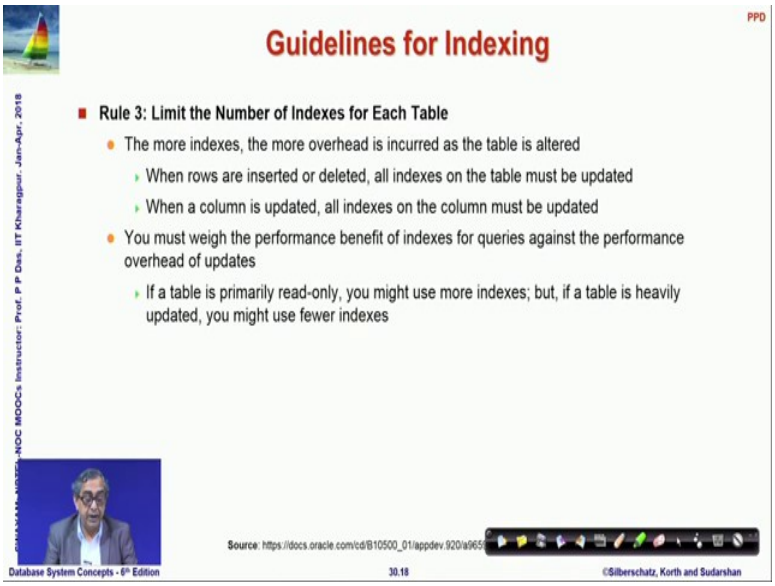


So, rule 1 index the correct tables and certainly related to that is index the correct columns. The columns with some of the characteristics I have just noted down are good candidates for indexing values are relatively unique in the column, then indexing will give you a good benefit. There is a wide range of values where you can your regular indexes will work well.

There is a small range of values where bitmap indexes will give you good results. So, use those in column contains many nulls, but queries often select all rows having a value. So, there are column have lot of null values, but whenever you do a query then you actually take out rows which have values. So, in those cases you can you can actually if you involve certain I mean if you write the SQL query by keeping the condition in a way. So, that the index can be used that is for example, you could put a condition such that only non null values will be matched.

Compared to that if you have just taken (Refer Time: 17:54) at non null check your first query would run faster; if you have an index on the COL_X because the query would be able to work on that index. So, these are things that you should do in terms of the column. And if a column has the kind of characteristic that there are many nulls in the column and you typically do not search non null values; then it is good it it is better not to index those columns long and long row columns cannot be indexed anyway. So, this remember the rule 2 index the correct columns.

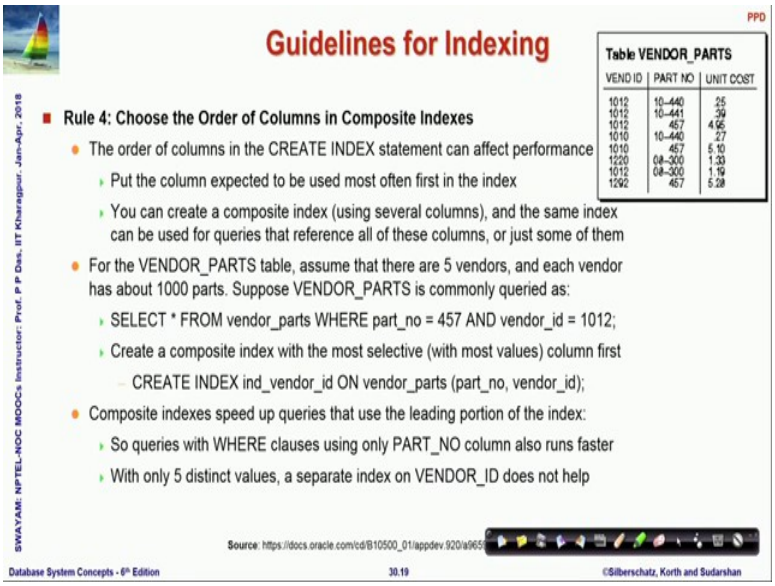
(Refer Slide Time: 18:29)



Then the rule 3 limit the number of indexes for each table the more the index more overhead we have already seen this as a part of rule 2 rule 0 as well. When rows are inserted or deleted indexes of a table must be updated when columns are updated all the indexes on the column must be updated. So, there is a lot of cost; so, half as limited number of indices as will start with purposed.

So, you must regularly weigh the benefit of having the indexed for queries against the performance overhead of the updates. For example, if a table is primarily read only you might use more indexes because the overhead will be less, but if a table is heavily updated you might use fewer number of indices.

(Refer Slide Time: 19:14)



Rule 4 choose the order of columns in the composite index. So, you have already seen couple of slides back I talk to you to the impact of what is the impact of ordering in other columns in terms of composite index. So, the order of columns in the create index statement can affect performance. So, the column that you expected to be used most often put that as the first index.

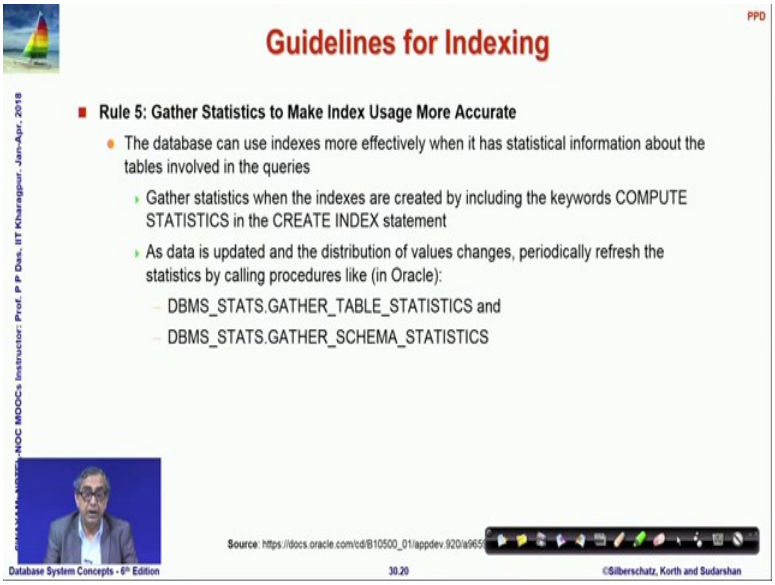
Because it is also possible that you are actually not doing a query which takes the whole of the composite search key, but a part of it, but if you have a composite search key index you will still benefit if the query is using the attributes from the first part of the index. So, here I am showing some example say there is a vendors part table and let us say there are 5 vendors and let us say. So, there is vendor id part number and unit cost forget about unit cost for this consideration.

So, it is primarily part number and vendor id. So, let us say there are 5 vendors and each vendor has about 1000 parts. So, and let us say that it is queried like this that the part number is such and such and vendor id is such and such you get you select all all that matches.

Now, if you create a composite index then it should be on part number and vendor id not the other way around. Because if you if you do that then queries where only part number is used will also run faster, but the vendor id here is not a very good candidate for indexing as a as a first attribute because it has only five possible values very small

number of value. So, indexing really does not help here it cannot discriminate after indexing there will be lot of clusters still found.

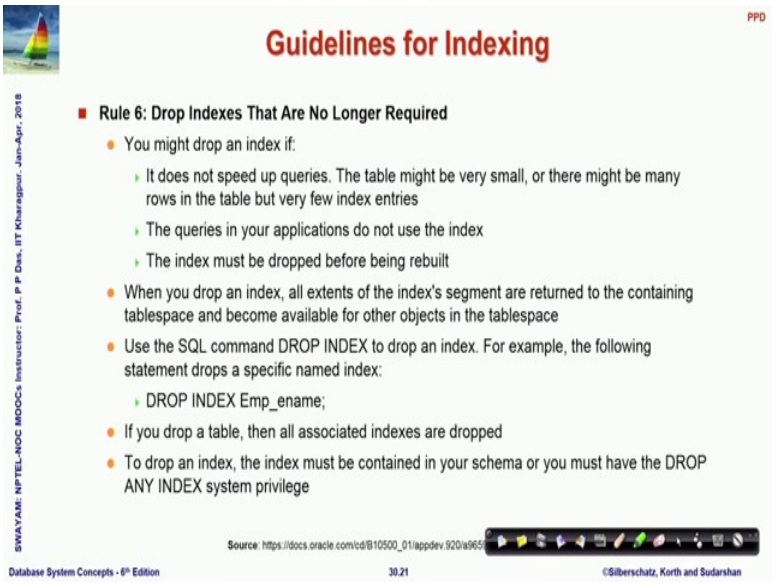
(Refer Slide Time: 21:17)



Rule number 5 gather statistics to make index usage more accurate that is a that is a very very important factor the database can use indexes more effectively if the statistical information is available. So, gather statistics learn from that we have already discussed how to gather statistics from the create index statement.

And then there are functions these are function names in oracle in your system you might want to check up what these functions are called. So, by that you can find out statistics about the tables and the schema that you have and use that information to subsequently optimize the index.

(Refer Slide Time: 21:58)

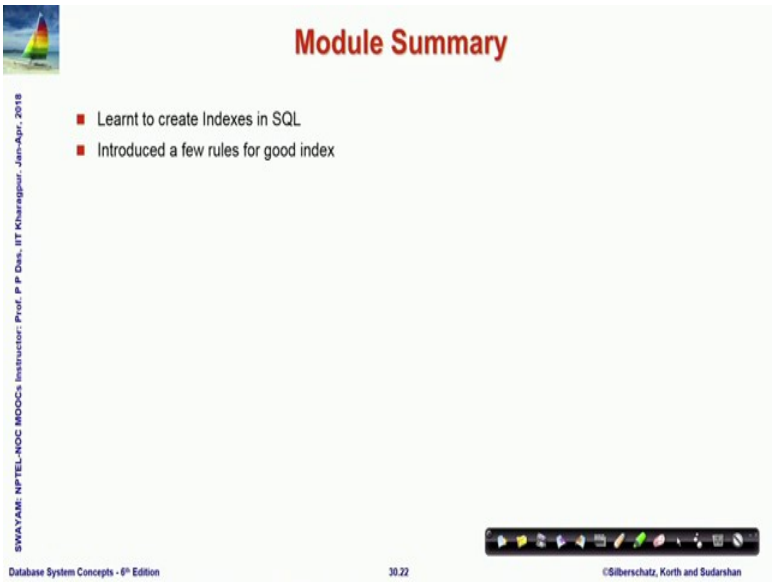


The last rule 6 is drop index that are no longer required. So, if an index might be dropped because for several reasons for example, it is it simply does not speed up the queries. So, table might have become too small there will be many rows in the table, but very few index increase right we have seen these are not the ideal.

So, it may not have been the case earlier and now it may be the case. So, in that case that index should be drop because it is not helping you the queries in your application do not use the index the query you have certain indexes and the queries are done on other attributes or other composite attributes. So, it is not the indexes of no value and; obviously, index must be dropped before being rebuilt if you are rebuilding if you are creating new index in a new way then. So, make a judgment and drop indexes which are no longer required as an when you observe that in drop indexes and improve the performance.

When you drop an index all extents of the indexes segment are return to the tablespace. So, this is basically the space management and SQL command for this you already know now please keep in mind that if you drop a table then all associated indexes are automatically dropped because; obviously, if the data is not there then how about their index. So, to drop an index you need the drop any index system privilege we talked about privileges earlier too.

(Refer Slide Time: 23:26)



So, this summarizes our discussions on indexing and hashing. So, here in this particular module you have learned to create index in SQL and introduce few rules for good index. Overall in this week in all the 5 modules we have learnt about how to speed up query processing, how to speed up the execution of access insert delete queries in your database through the lifetime. And we have looked at various different indexing schemes, we have looked at hashing and made comparisons.

So, one take back that you can certainly have is the most important indexing scheme or indexing structure is the B + tree which can be used for data files as well as for index files and several like SQL server uses the B+ tree only. Now, hashing options we have looked at and we have seen that it has a varied acceptability it is a powerful technique, but not all systems use that equally strongly.

And we have then made understood that indexing a database or tables on different attributes is a very delicate responsibility which has to be done with a lot of judgment. And for that statistics must be rightly collected and good judgment in terms of the distribution of the data, access of the data nature of queries all these need to be considered carefully so, that you can really get good performance from the design that you have.

So, on top of the knowledge of good design that you acquired through the all the theory of normalization and you know redundancy removal; your good judgment in terms of