DATA BASE MANAGEMENT SYSTEM

Prof. Samiran Chattopadhyay

Prof. Partha Pratim Das

Computer Science & Engineering

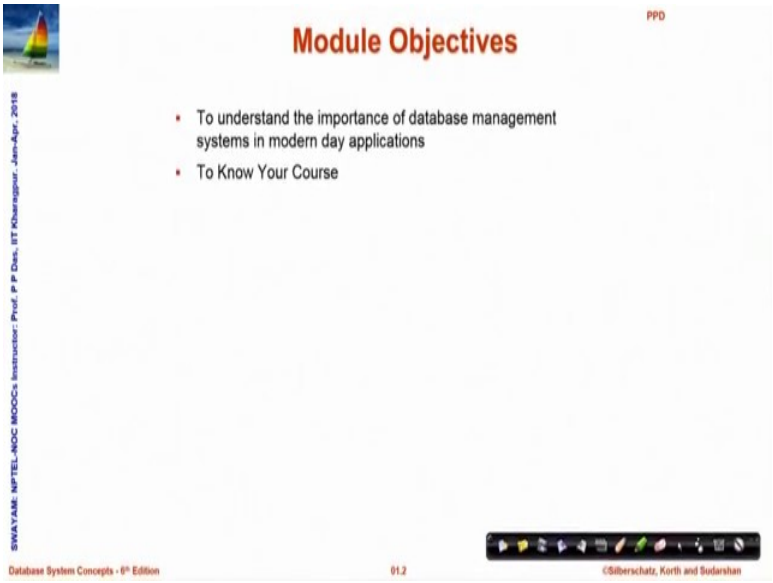
Indian Institute of Technology, Kharagpur

INDEX

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

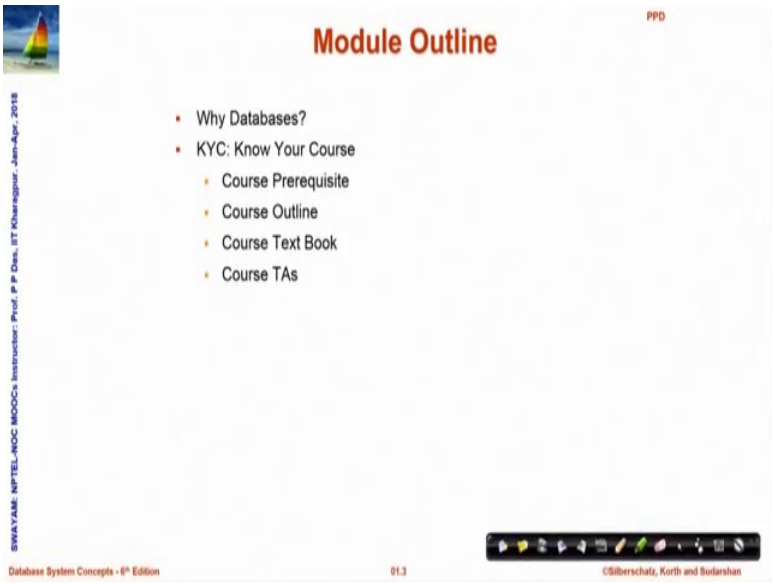
Lecture - 01
Course Overview

(Refer Slide Time: 00:45)



Welcome to database management systems. In this course, we will have 40 modules; each module would be of about half an hour. So, this is the first module, where we would talk about the overview of the course. So, we will discuss the importance of database management systems in modern day applications, and we will familiarize you with different aspects of the course.

(Refer Slide Time: 01:06)

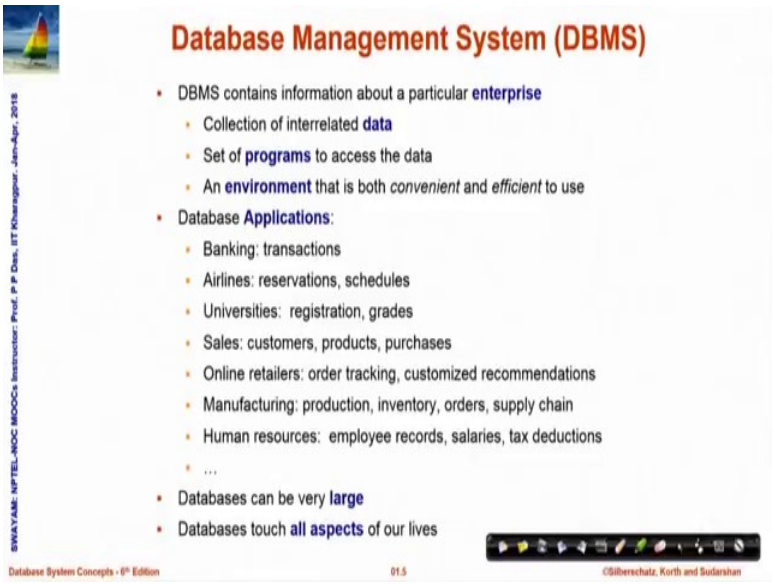


So, this will be the outline. First, we will try to explain why we need databases, and then we will run through a KYC on the course prerequisites, course outline, the textbook and the TAs who will help us in this course.

(Refer Slide Time: 01:28)

So, first why do we need databases?

(Refer Slide Time: 01:32)



A database management system contains information about a particular enterprise. So, it deals with collection of interrelated data. We have a set of programs which access and manipulate that data; and together a DBMS presents an environment for convenient as well as efficient use of data of the enterprise under consideration. So, there could be different database application.

Actually, if we look around in the world that we are living today, every aspect of our life has been touched by certain database application. Some of the very common and wide applications will include banking; we are doing net banking very regularly. Net banking is highly distributed database application that allow us to do different kinds of retail as well as corporate banking.

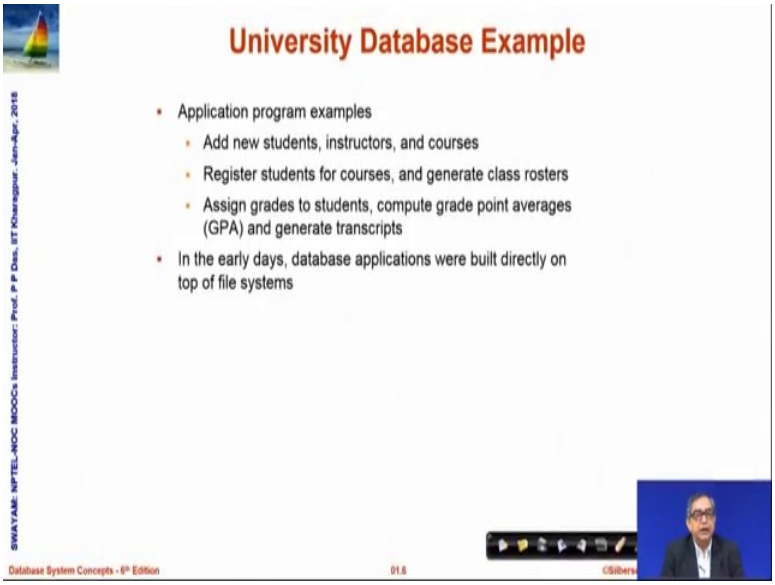
We perform different kind of booking reservation, railway reservation, airline reservation, hotel reservations all of these are now possible over internet as well these are all big database applications. When we are attending colleges, universities, the students, courses, teachers, course enrolments, performance of the students, and different courses, examination all are parts of huge university database applications.

There are different sales applications, online retailing applications like I am sure all of you have been using some of the Amazon, Flipkart, E-bay, Snapdeal, all these are online retail database applications which allow us to order to select items to make payments and to track the deliveries of different items that we have ordered. There are database

applications in terms of manufacturing, production systems, inventory management, any big factory of manufacturing need to use a huge database application to manage the supply chain, the inventory, the orders everything. There are database applications in human resources applications like Linkedin are huge human resource application of database, the social media applications all involve different kinds of database applications.

So, in this database management system course, we are going to understand how such applications can be designed developed and managed over a period of time. Database applications are typically characterized by the fact that they are very large. If we have a relatively small set of data then possibly we will be able to manage it in terms of using an excel sheet or a couple of excel sheets, but soon when it goes beyond a certain size we need a database application. So, the fact of being large is a critical factor of any DBMS. So, with all this we as we observe the database says cover all aspects of our life and hence understanding DBMSs is a critical requirement for any computer science information technology student.

(Refer Slide Time: 05:54)

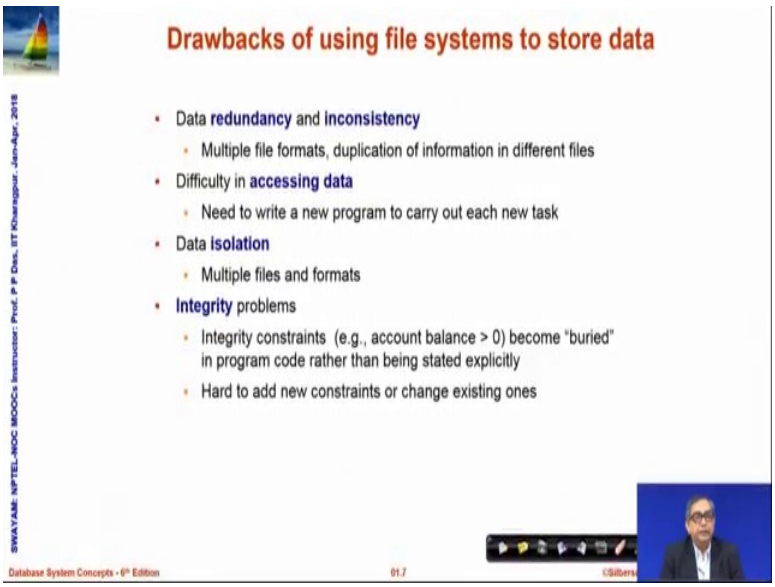


So, if we talk about a specific database example say university database, then we will have several application programs which will allow us to do several required applications like when new students join, we would need to add the students. We will need to add new courses when courses are floated; we will need to add new instructors

when faculty join; we will need to do allotments; we will need to do registration of students for courses. We will need to conduct examinations, we will need to assign grades to students when they are graded for different courses and compute their GPA and so on. So, all the activities that a university has to deal with are application programs of varied kinds that the university database need to work with.

In the earlier days, before the days of databases, typically such applications used to be managed through file systems. We all know that all our system has a file system, where different files text as well as data files can be stored, and information can be written in these files in a certain order. And just to quickly recall file systems typically have a large number of sequential files which can be written and read in a certain order and some random access files where you can reach a particular point in the file to do certain access operation certain manipulation operations. So, in the earlier days, it was a collection of file systems which managed large enterprise data as is required.

(Refer Slide Time: 07:53)



But over time it was observed that the file systems to store data to manage data has a lot of drawbacks. For example, if you look at in a file system, there is often a lot of data redundancy and inconsistency. These are terms which we will loosely define here and as we go across along the course, we will understand these terms better. But redundancy just to explain redundancy is a concept where the same data is written at multiple places in different forms and that may give rise to several forms of inconsistencies because if

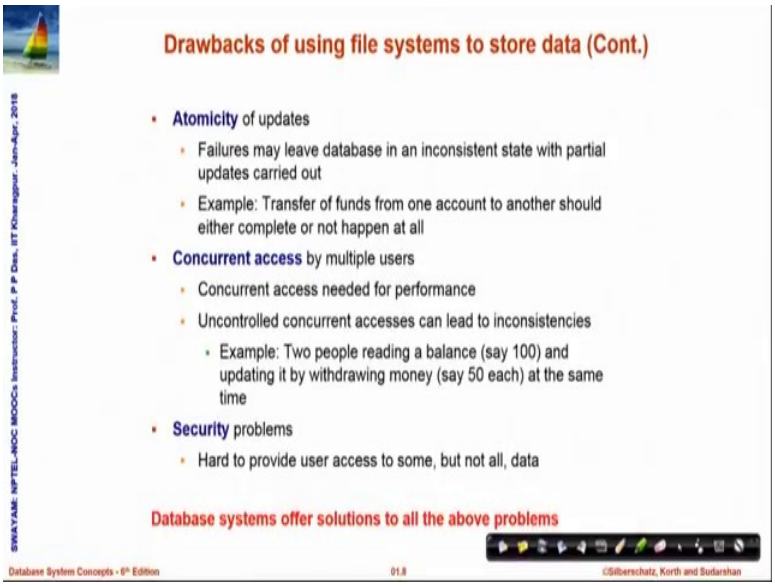
you write the same data in multiple files, because you need to deal with many of these aspects.

So, there is a file for students, there is a file for teachers, there is a file for particular courses, there is a file for enrolment and so on several data items may be redundantly copied in multiple files. And once that is done then we can it is very easy to get inconsistent in terms of the data because you may update the data in one file, and may forget to update the data in another file, so that is one of the first problems with using the file systems.

Then this difficulty in accessing the data because as I said the data is the files often are sequential in nature, even if they are random access, then every task might need to use data from multiple files. And opening those files and reaching to the appropriate point of access is a non-trivial task. Then there is issues of data isolation, because there are multiple types of files, there are multiple formats used therein. Very importantly there are a lot of integrity problems, the database any database application need to have a lot of integrity.

For example, if you want to withdraw money from an account - bank account, then certainly the balance needs to be positive, you can withdraw only that much amount up to that much amount, which exists in the account. So, any application will need to check for this. So, if you use a file system based application to store the data then at every point wherever you are updating the balance, you will need to make such checks which make the application quite complicated, and often creates the possibility that certain integrity checks may be missed out. So, it is hard to code these new constraints over a period of time.

(Refer Slide Time: 10:40)



Then there are issues of atomicity of update. What atomicity means is the ability to do certain operations in a as a single unit. So, what you want is either that operation happens or the and if it happens then it happens in full in totality, whereas otherwise it may not happen at all. For example, consider that there is a funds being transformed from one account to another, so this means the account from which the funds are being transferred needs to be debited certain amount, and that same amount has to get credited to the accounts to which it is being paid.

Now, if for some reason of failure or because of the fact that there was link issues or something, if you are not able to make this whole transfer, then it is possible that you have already debited the account, but you have not been able to credit that account. Now, this will be a major cause of inconsistency in the database. So, what you want is if the transfer can happen then it must happen in totality that is what that debit and the credit must happen together or nothing should happen at all. So, there are several examples of requirement of automaticity for a update which is critical for maintaining consistency of the data.

The other aspect which has become very, very deeply required in every aspect of database is concurrency of access. If there is a database then certainly there is not a single user, there are multiple hundreds of users you think about net banking, you think about railway reservation, multiple users are trying to make bookings in multiple trains

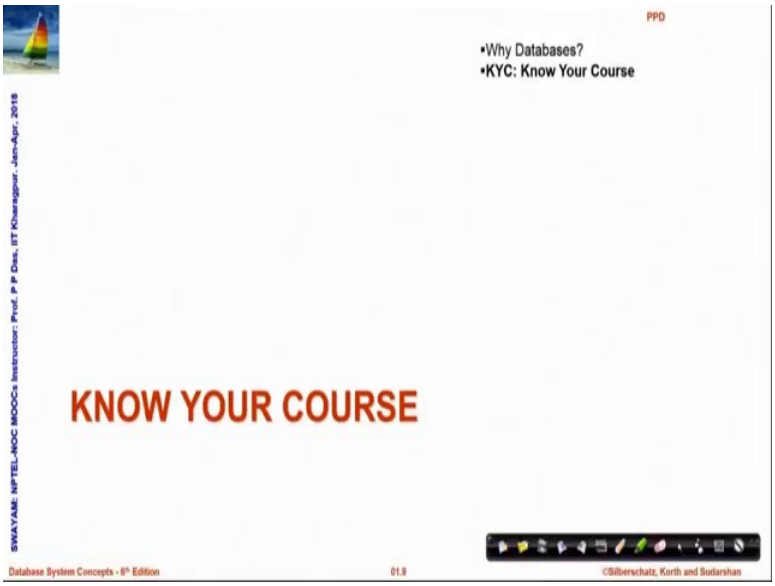
from varied stations to vary stations in different classes and so on. So, all of this must go on at the same time that is what is called the concurrency of update.

So, it is quite possible that while you are trying to update, you check berth availability on a certain train on a certain date that you intend to travel. At the same time, someone else may be checking for the berth availability on the same train on the same date, and there could be conflict of concurrency because there may be one berth available and you are trying to book that you have seen that one berth is available.

So, you go ahead and book it try to book it, and there is another user who also saw that one berth is available and that user makes payments and tries to book that. So, concurrency needs to make sure that both of these users should not be allowed to make the booking to the same berth because then that will be a disaster. So, uncontrolled concurrency can add to several inconsistencies in the application.

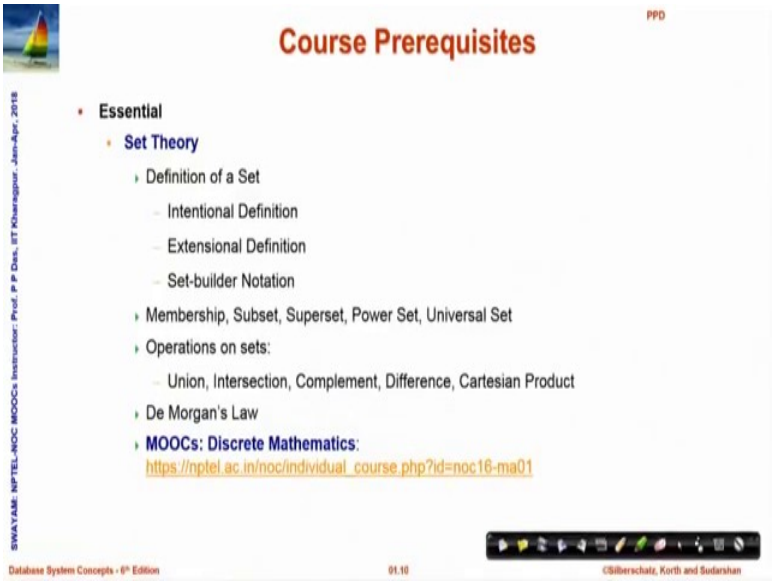
Then certainly there are you all would be very familiar that today we are living under a whole lot of security threats. So, there has to be proper security that it should be possible to access the data by a user to the extent the user is allowed to do that. So, as a user you should be able to access certain parts of the data, the manager of the system, the administrator should be able to access a bigger part of the data possibly. So, security is hard to provide in terms of a file system based applications to store data. So, all these with we conclude the data based systems had those which provide solution to take care against all those above problems, and we are trying to learn how to do such things.

(Refer Slide Time: 14:37)



So, moving on I would quickly take you through familiarizing with you with the overall plan of the course.

(Refer Slide Time: 14:47)

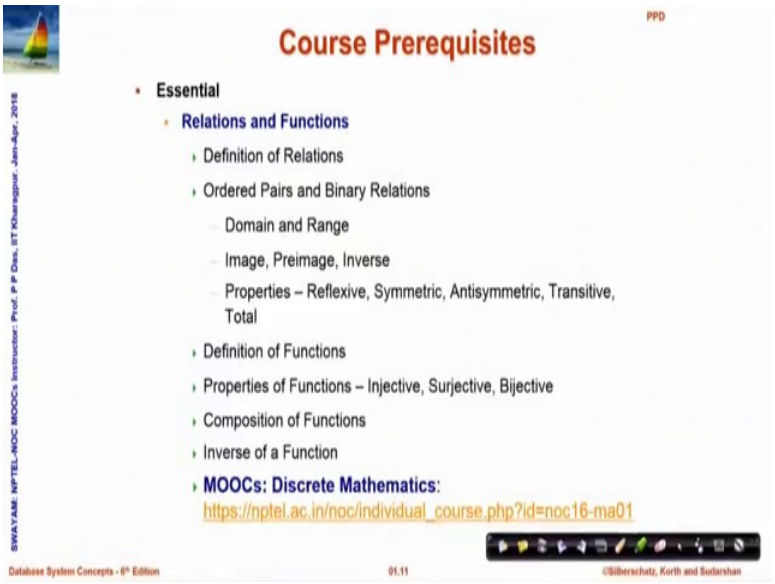


So, what we first I talk about are the course prerequisites. These prerequisites are kind of certain elementary level knowledge in computer science and related discrete mathematics that you should have. You should be that would make it easier for you to understand and follow the course; otherwise if you find at any stage that you are finding any of this aspect difficult to follow in the course then I would advise that you go back to

some of the background material and try to study them. So, I have tried to list down these prerequisite topics, one certainly is the set theory because that is the basic premise on which databases are designed on.

So, starting from definition of the set membership, concepts of subset, superset, power set, different operations of union, intersection, complementation, difference Cartesian product De Morgans law, all these basic set theory you should be very familiar and conversant with. If you are not I have mentioned one MOOC's course this is a past course, but you can access the videos and the contents which has a very nice discussion on these aspects of set theory which you may refer to.

(Refer Slide Time: 16:13)

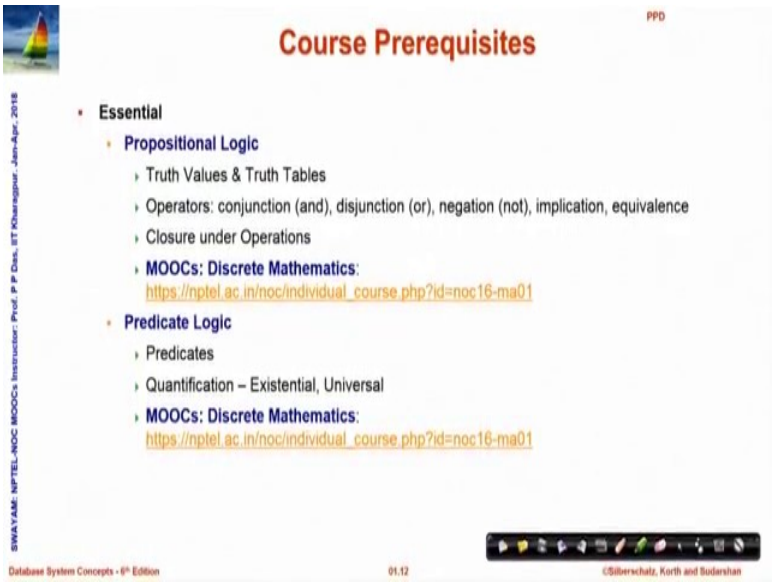


Moving on the next, which goes on top of the sets are the concept of relations and functions. We all know that a relation is a subset of a set. So, if I have a set A, then I can define a binary relation over that set A, which is basically the pair of elements from set A or in other words the relation binary relation is a subset of the cross product of A with itself where the domain and the range are related together. So, there are concepts of the image of a domain the pre-image of the range the inverse relation, and several basic properties of relations like the relation being reflexive, symmetric, anti-symmetric, transitive, total relations and so on.

You should be familiar with these; otherwise, you will find difficult to follow major concepts in the database systems because this tribal system primarily the one that we are

going to take you through in this course is relational in model. So, it is heavily based on relations and functions. And you should understand one specifically a relation becomes a function, and when what is meant by functions being injective, subjective, bijective, what is meant by composition, and inverse of functions and so on. Again if you need, you can refer to this MOOC's course on discrete mathematics to brush up your knowledge about relations and functions.

(Refer Slide Time: 17:52)

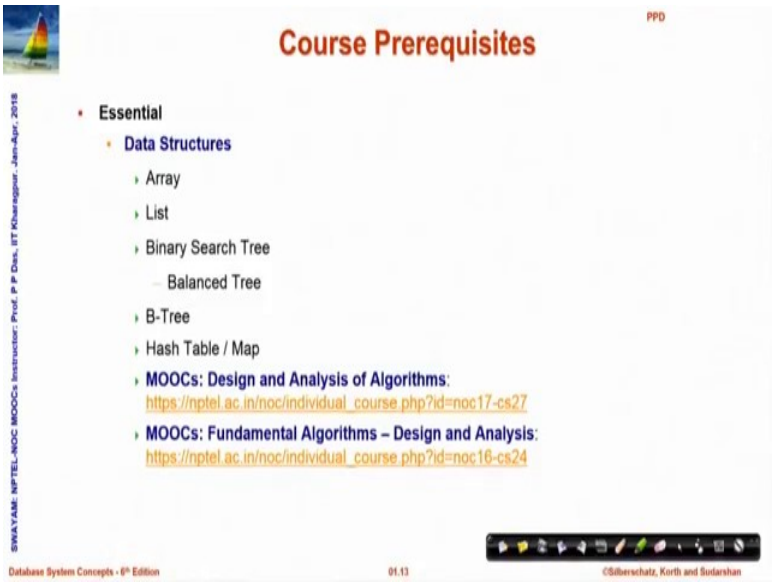


We also need you to have a basic understanding of the propositional logic which is truth values true and false; and the different operations of conjunction, disjunction, negation that is and or not, what is meant by implication, what is meant by equivalence. You know that given two variables, which have or two propositions which can take a value true or false the conjunction of them can be represented in terms of a truth table where we say that only when both these propositions are true, then the resultant conjunctive proposition becomes true; otherwise, a conjunctive proposition is false. So, you should be familiar with these concepts; if you are not, please brush up your ideas about propositional logic.

We need a little bit of predicate logic as well a predicate logic in contrast to propositional logic deals with quantification that the knowledge of existential and universal quantifier. When you say that whether certain proposition predicates hold for all values in the domain or for some value in the domain whether there exists some value for which it

holds or whether for all values it holds. And based on that predicate logic is build up we do not need very advanced concept here just basic level familiarization will help and the same MOOC's course on discrete mathematics would be of your help in case you need to brush it up further.

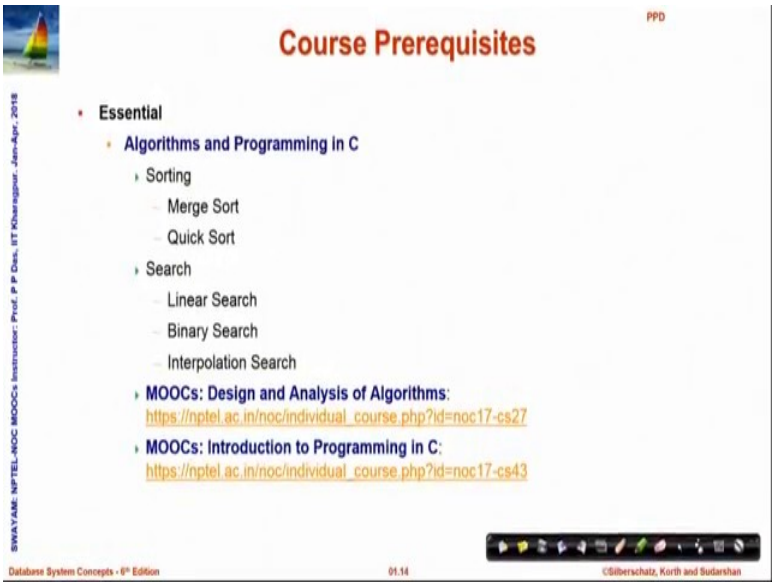
(Refer Slide Time: 19:33)



On aspects of computer science, certainly you need a good familiarity with the data structures array list particularly binary search tree, what is called a binary search tree, what is meant by a height of a binary search tree, when we say that a binary search tree is balanced. What are the ways and conditions of balancing is particularly the B-trees for organizing good search trees hash tables, what is hashing we need you to be familiar with this concept, because the databases will be heavily designed based on the concepts of B trees and hash tables and so on.

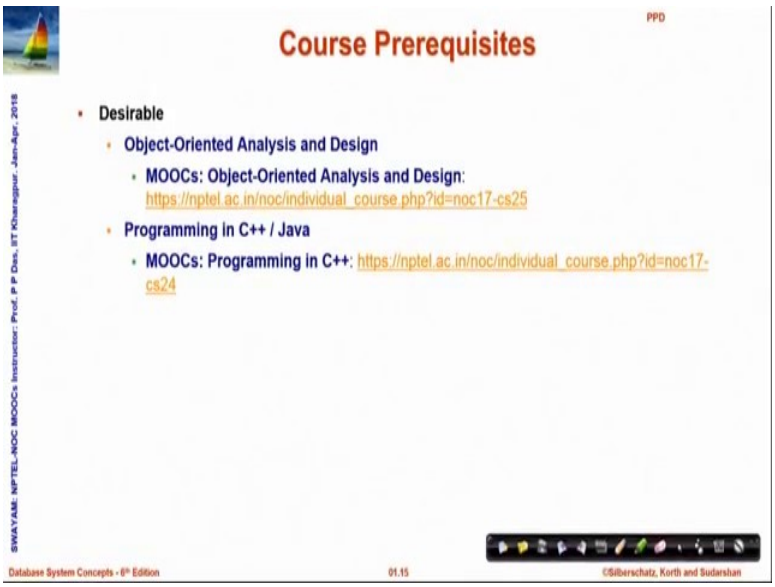
There are courses on design and analysis of algorithms, fundamental of algorithms have mentioned two excellent courses in MOOC's from which you can brush up if you need to.

(Refer Slide Time: 20:31)



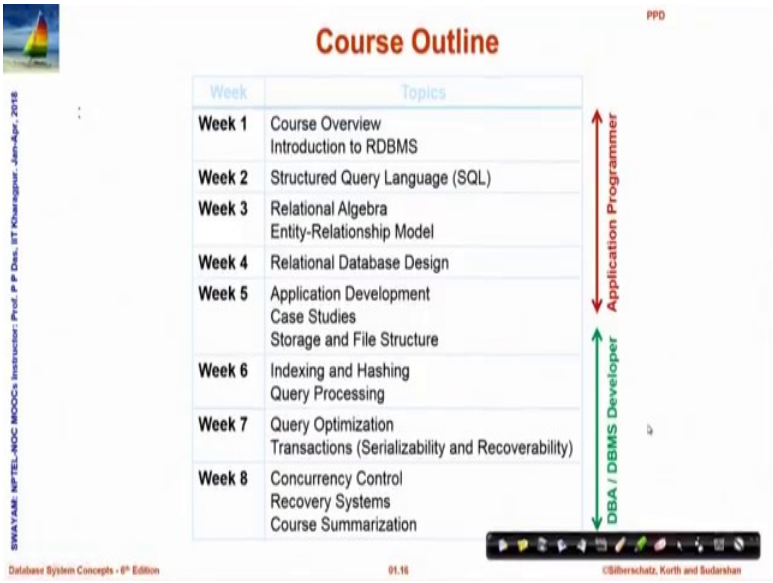
Certainly you need certain familiarity with common algorithms particularly I had mentioned sorting and searching algorithms, because these are critical for database applications. And again the same MOOC's courses would be of your help. And it will be good to have familiarity with programming in C, because several of the applications need some application high level application programming to be performed. And we would assume that those aspects we described in C because that is a fundamental and most commonly known language.

(Refer Slide Time: 21:11)



Besides these prerequisites which I have marked as essential, because they will certainly be required for the major parts of the course, it will be good if you have some familiarity with object oriented analysis; and design and some language which is more heavily object oriented object aligned in nature like C++ or java. So, again some MOOCs courses the related MOOCs courses are mentioned here in case you need to brush up.

(Refer Slide Time: 21:46)



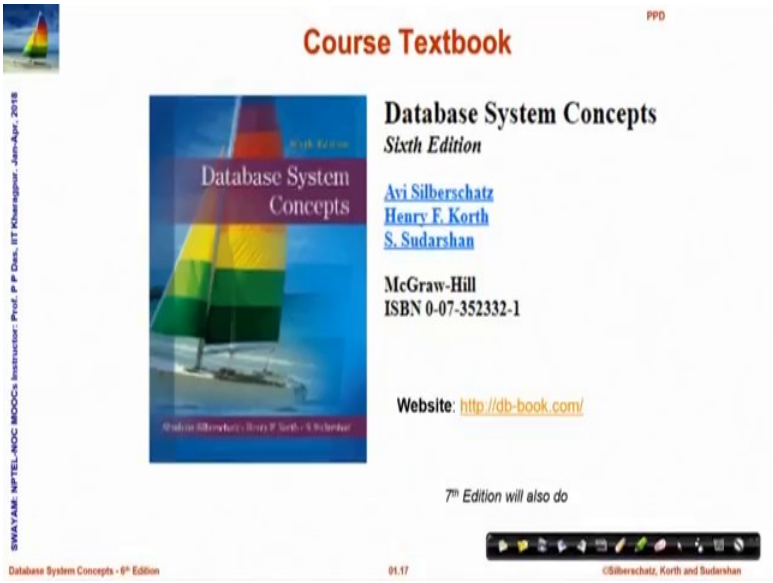
Moving on this is your course outline. So, as I said the course comprise 40 modules. So, it is divided into 8 weeks. So, this plan is given based on what we do in different weeks from week 1 to week 8. So, as the course unfolds, you will be able to we will take you through these modules on these topics. And on the right, you can find that I have marked that the initial part of the course which we cover from week one to first half of week 5 is primarily meant for application programming which it means that the database system has already been designed and basic premises the schemas and constraints have been set up.

But now you want to write different data query different data manipulation applications and that is where the large volume of database engineers work. So, they are called application programmers, so that is I should say the first level in terms of a database understanding. And you must target to become a master of application programming to get started with.

The other half of the course which start from the middle of week five with the storage and file structure and goes on till the next is meant for the analysts who are responsible either for designing a particular database which the application programmer can use tune that for performance index it properly design queries to be efficient. So, these kind of analysts will be involved the more with the understanding of the second part of the course.

And the second part of the course would also be useful for the database a DBMS designers. If you want to really become an advanced programmer we want to work on database engineering in terms of creating database management systems not merely creating databases or database applications, then you need to have a good initial command over the second half of the course. So, while you prepare for the course where you go through the modules please keep this in mind that your familiarity with the application programming must be at the highest level. And in the later parts will be relatively little advanced, but they are required for good design and good development of consistent efficient system in future.

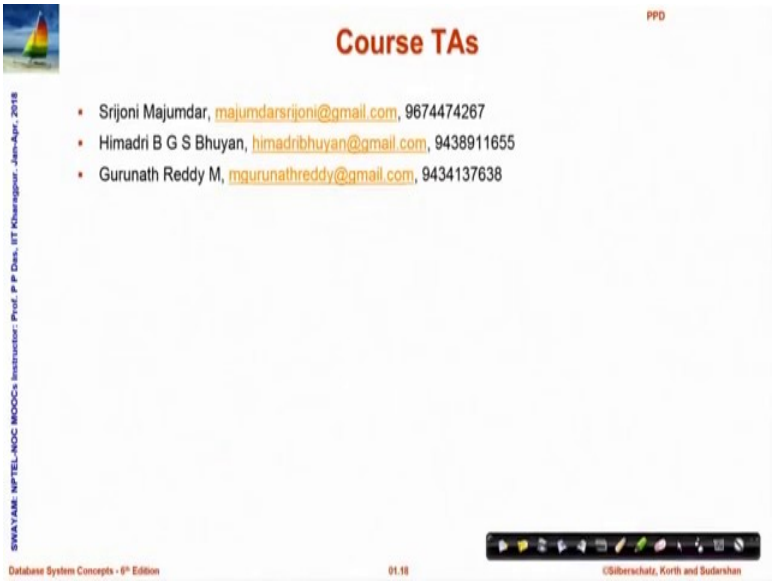
(Refer Slide Time: 24:39)



We will follow a textbook. This is as you can see this is a sixth edition that I am following in this course. It is called Database System Concept by Silberschatz, Korth and Sudarshan. This is kind of a classical book in database systems; current version actually is the seventh edition. So, if you get access to the seventh edition, you can use that as

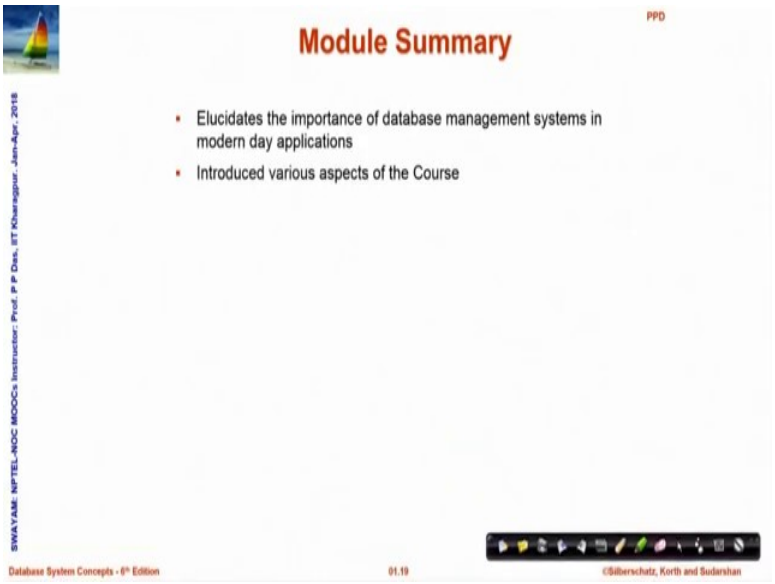
well, but whatever we are following in this course sixth edition is good enough. So, I advise the, that you try to get a copy of this book to yourself.

(Refer Slide Time: 25:24)



So, moving on we have different three TAs we will help you in this course Srijoni Majumdar, Himadri Bhushan Bhuyan and Gurunath Reddy. So, these are their emails; I have also put their mobile numbers. However, I would advise that unless you are really stuck avoid calling them on the mobile, because they are research students as well, busy with their own work as well, but they can certainly put all your questions on the forum, which will be promptly responded by some of these TAs or by myself. And in case you would have very specific follow ups to do, you can write email to one or all of these TAs. So, this is about the course overview.

(Refer Slide Time: 26:17)



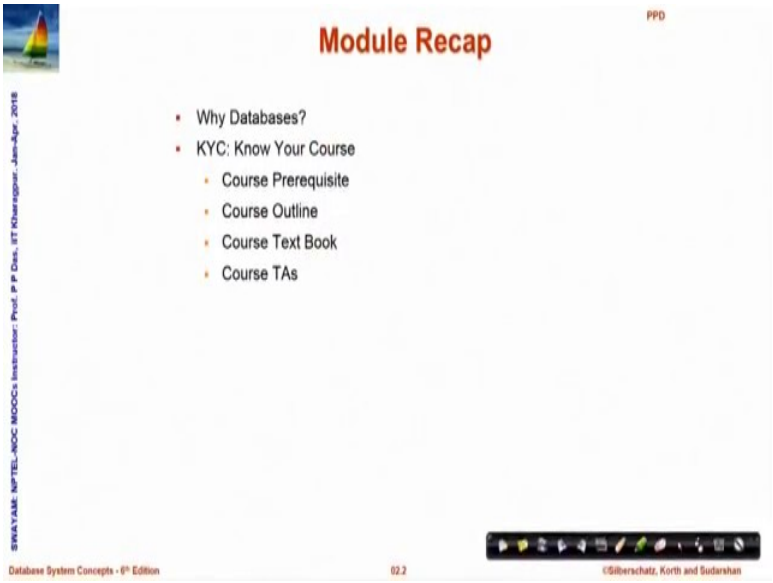
So, in this module we have discussed about the importance about database management systems in the modern day applications. And we have tried to familiarize you with different aspects of the course. And I reiterate that please give due consideration to the prerequisites as mentioned, we have floated in an assignment called assignment zero where there are questions on different aspects of these prerequisites, please try to solve that assignment and see your performance. This assignment I would like to mention that this assignment will not go in the final evaluation of the performance of the course; this is just to give you an idea for self assessment of your preparedness for this course.

So, if you find that on questions on certain topics say on relation function or on data structure, if you have not been able to answer the questions well then it will be good to check back and go through the prerequisites once more. But please keep in mind that database management system is a course which depends on these background knowledge quite heavily. So, if you have gaps in understanding those topics, then all through the course you will get into several difficulties in understanding and problem solving. So, that is about our first module. So, from the next module, we will start introducing the database management system. Enjoy the course, and try to learn, try to become a good database engineer.

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

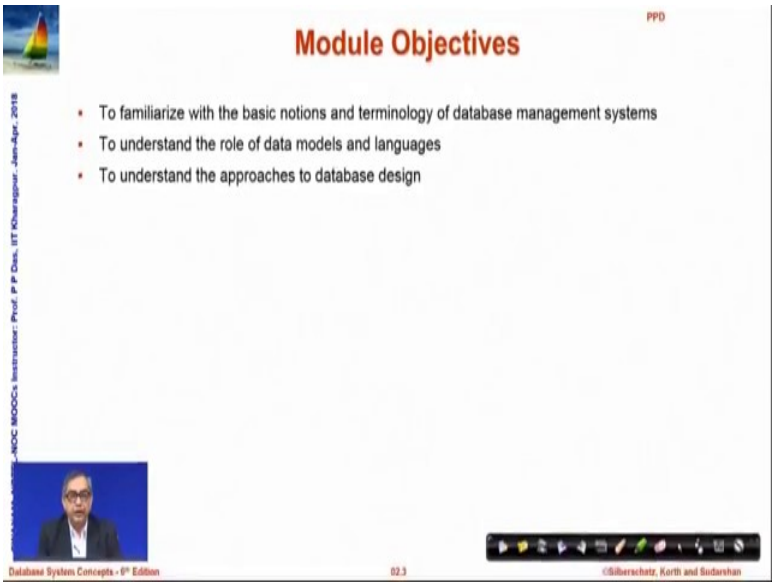
Lecture - 02
Introduction to DBMS/1

(Refer Slide Time: 00:42)



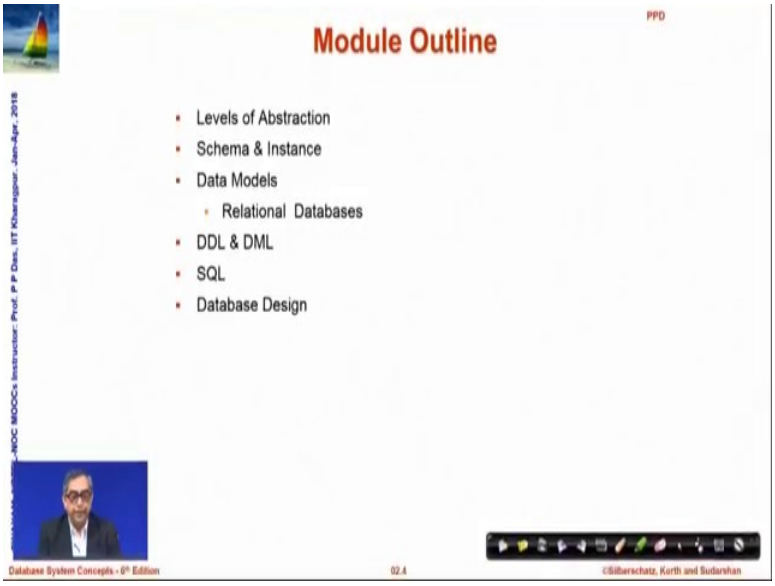
Welcome to module two of database management systems. In this module, we will talk primarily about the introduction to the DBMS. This discussion will span two modules that is the current one module 2 and the next one that is module 3. Just to quickly recap in the last module we have discussed about why we need databases, and we have introduced you to different aspects of the course.

(Refer Slide Time: 00:54)



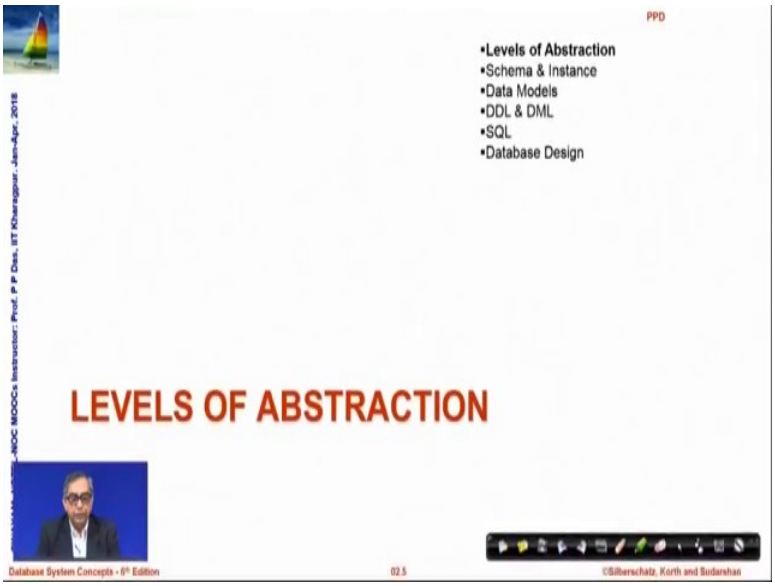
In view of this, in this module, we would familiarize you with basic notions and terminology of a database management system just at an introductory level. We will try to understand the role of data models and specific languages for database systems. And we would also outline the approach to database design.

(Refer Slide Time: 01:21)



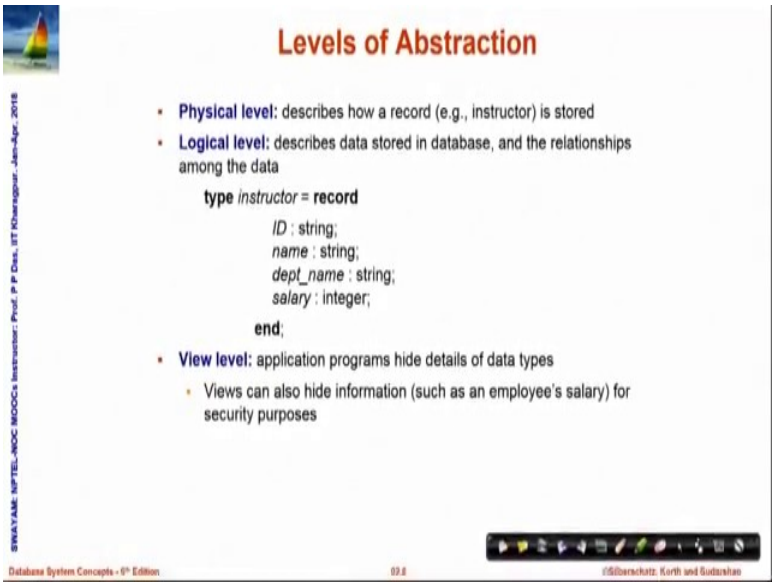
So, the module outline would be like this. And as we go along you will be able to follow which particular aspect of this outline we are discussing about.

(Refer Slide Time: 01:35)



So, to start with we talk about levels of abstraction.

(Refer Slide Time: 01:40)



A database system like any other system is conceptualized in terms of three levels of abstraction. At the lower most level is the, what we say is a physical data level or the physical level which describes how a record is actually stored, so that is about the physical storage in the system. At the next level, we talk about it we say it is a logical level which describes the data stored in databases and its relationship amongst the data. So, you can any data that is stored you can think about it as a record. So, if we here we

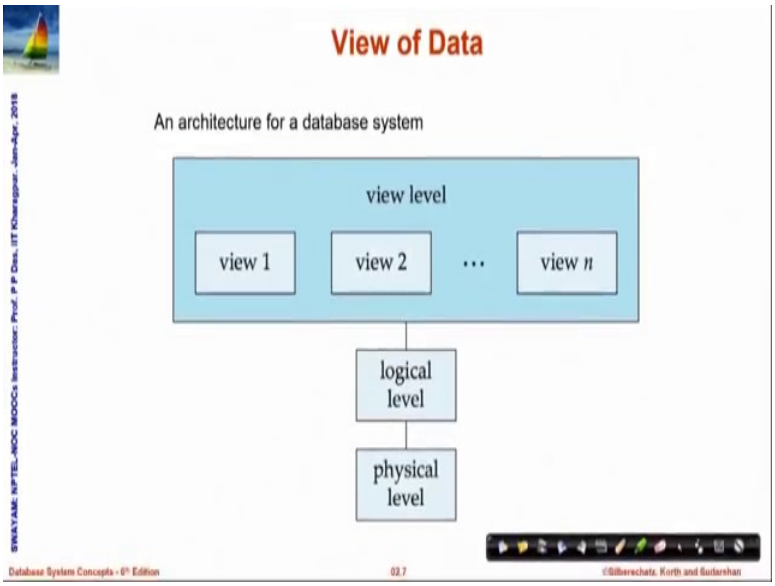
are illustrating the record of an instructor who teaches a course. So, as you know the record is a collection of multiple fields of different types. So, here we have field to describe the ID identifying number or shrink of an instructor, we have the name of the instructor, the name of the department, the salary and so on.

So, this logically says this is the entity this is a record or this is the structure of a record that I want at a logical way. So, this in contrast to the physical level, logical level is not concerned particularly with how that these data, how this string the number and all that will be actually stored, and how these multiples of hundreds and thousands of records will actually be stored so that they can be efficiently used. But we are just concerned with the logical view that I should be able to deal with records as they are.

At the third level which is it can say the topmost level is called the view level where the application program tries to view the data. And when the application program tries to view the data, it deals with the details that it needs to; and rest of the details are usually hidden from this view. For example, if here we talked about the university database in the last module, so if you are talking about the university database, and then you are a student. And you are when you access the database you should be able to see what all courses you are enrolled in or where is that course being held who is the instructor of that course and so on. But you should not be able to access or see the view of what is the instructors salary or for that matter, what are the grades that are obtained to by different students in different courses and so on.

Whereas, an instructor may be able to view the performance of the students in multiple different courses particularly the ones that he or she is involved in evaluation, but she again may not be allowed to check the salary of other instructors. So, view level is a high level where of abstraction where you try to provide the information about the data in terms of what the application needs, what the users of that application need, but you do not actually deal with the details of all the records that the logical level has.

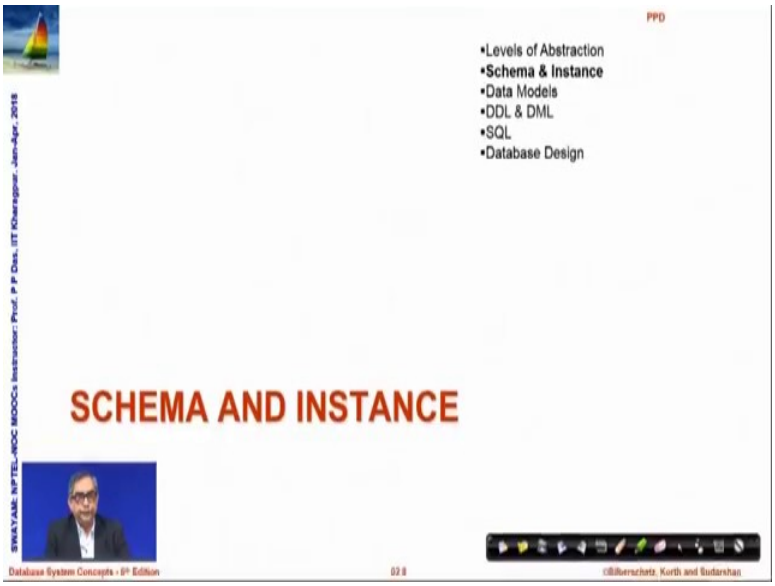
(Refer Slide Time: 05:29)



So, these are three levels form the basic structure of a database system architecture of a database system. As you can see the physical level using that a logical level of records are formed. Physical level typically is in terms of database files is binary in nature, the organization of those files. The logical level deals with the records and the different fields of the records the schema of the database and so on.

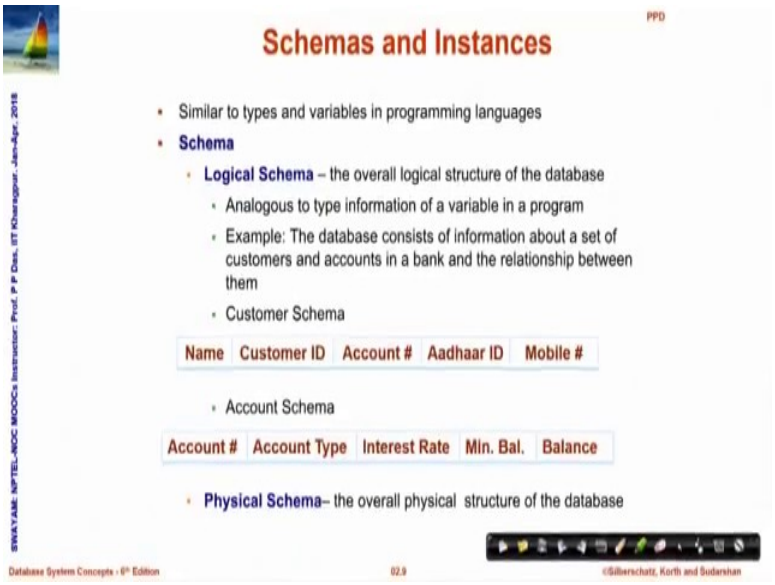
And the view level is something which is constructed from the logical level in terms of different views that the different applications need. I am sure at this stage you may not understand the whole of these levels and their implications, but this is just to give you an idea of the existence of three levels, and the need to deal with the three levels. And as we go along, we will see that we will refer to these levels more and more and discuss about the specific aspects of those.

(Refer Slide Time: 06:27)



Next, let us talk about schema and instance.

(Refer Slide Time: 06:34)



We will very regularly keep on referring about schemas and instance. The schema is in a very simple terms say if we talk about first a logical schema, it is a way a certain data needs to be organized, it is a plan for organizing data. So, if you can draw a parallel then say when a building is constructed, a plan is prepared. And according to that plan several buildings a in a say residential complex may be constructed. So, there is a difference between the plan which gives you the layout of where different rooms should be where

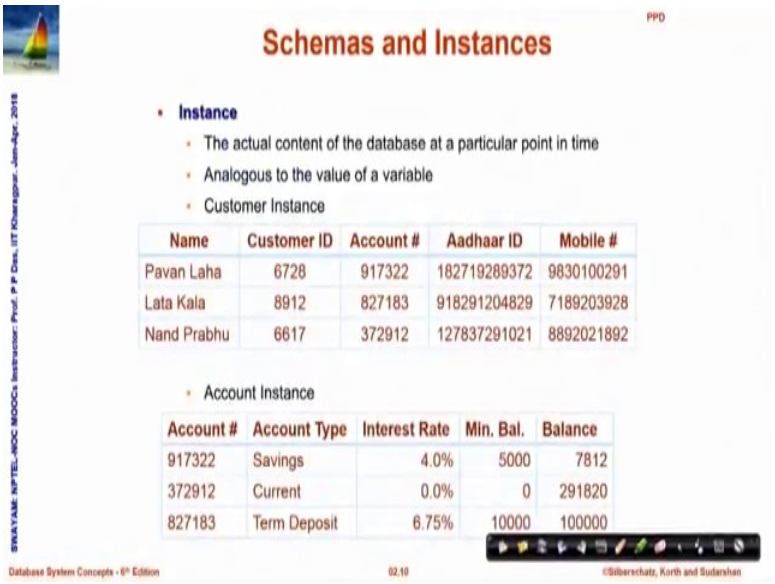
there is a staircase where is the courtier and so on and the actual building when or the group of buildings which are constructed. So, the schema primarily talks about what is the plan to organize the data.

So, if we talk about a customer schema, it has multiple different fields, it should talk about the name of the customer, ID of the customer, it is account possibly the other ID the mobile number and so on. So, the fact that these the fields need to be present for describing a customer, forms a customer schema. Whereas, when we talk about a specific schema of account that the customer holds with a bank, then we need the account number, account type, interest rate, minimum balance, the current balance and so on. These are the fields of information that we need; and we need to know what is the type of every such field, and all those and those kind of information from the schema information.

And again in line with the abstractions of physical logical and view as we did, schema also can be at a logical schema which is corresponding to the logical level of abstraction. And we may also have a physical schema which tells actually how the data is physically organized in the database, what are the different database files, how they are indexed and so on.

So, all these information which we can say is kind of a metadata information. This is not actually that it is not the customer schema is not saying who the customers are, the account schema is not saying, what are the accounts, what is their balance. But it is saying that if a customer needs to be defined; then what is the information that you need to know, what is the information that you need to manage. If an account need to be described then what is the different fields that are important. So, this schematic or this metadata is called the schema of a database.

(Refer Slide Time: 09:25)



Now, based on this schema specific instances of databases happen, instances when you actually have populated different records according to the schema. Now, naturally once the schema is fixed, your records will need to have values in all of those fields that the schema has; and every value must be of the type that the particular field is specified with. So, I have just shown here certain sample instance of customer schema, where you can see three customers with their name, customer ID, account number, other ID, and mobile number, these are all fictitious data of course, but this is just to give you an idea of how this customer instance would look like.

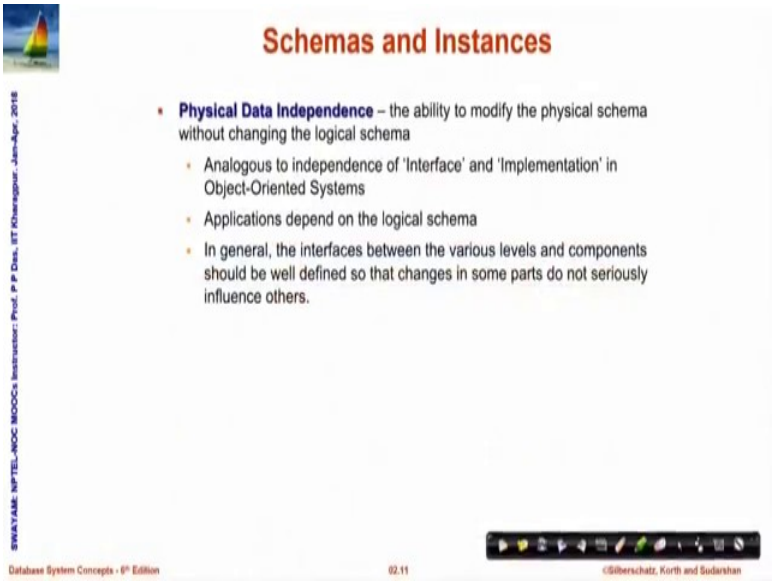
Similarly, we have shown what is a accounts instance, so you can see that there is a some kind of a relationship that you can see between these instances. For example, the first customer ID on the customer instance can be seen as a first I am sorry the first account ID in the customer instance can be seen as a first entry first record in the account instance and so on. So, when we actually populate the schema with different records and this is what keeps on changing.

So, certainly when we do operations on the database, then certainly very regularly new records will get added, some records might get deleted from this instance, and fields of certain records may keep on changing. For example, in an accounts instance very regularly whenever a transaction is done, the account balance will get changed; maybe at a certain time the bank might decide to change the interest rate for a certain type of

account then the interest rate field will get changed, new customers may come into the customer instance and so on. So, instances keep on regularly being updated manipulated added deleted updated, but the schema remains unchanged.

So, change of the schema is very rare in a database and needs to be done only when the database is designed or when it is being upgraded. Because once you change the schema, it changes the way you look at the whole world, you look at the whole database scenario. So, if you are changing the schema at a logical level, then naturally the your view will also get affected, because you are using these schemas to decide how you would like to present a transaction application to the user or for a balance check application to the user and so on.

(Refer Slide Time: 12:22)



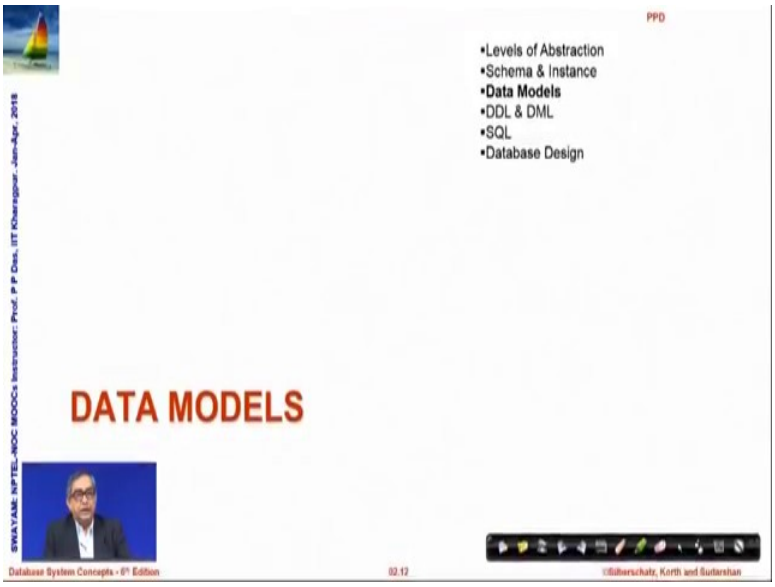
But, of course, what do we would want is between the physical schema and the logical schema we normally would want certain independence. What it means is the logical schema is what you need to deal with, because it is linked with the view that you have at a higher level of abstraction. On the other end, the logical schema is based on the physical schema; physical schema is how you are actually organizing the information in terms of the binary files the database files.

Now, certainly you want that logical schema not to change because if it changes then at a view level all your applications will have to change. But it is quite possible that your physical schema the way you have organizing files and so on might need a change,

because maybe it is just that you had designed the database for 10,000 records and you already have 9000 records and you would like to expand it to maybe 1,00,000 records.

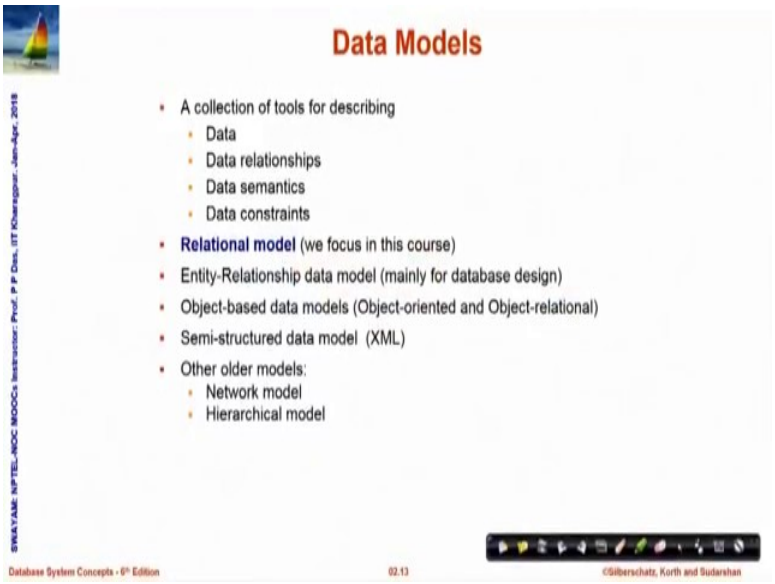
And the physical this system needs to be different you may need to reorganize the files and so on, you may need to index it in a different way and all this, but you would like to do that change in the physical level without requiring any change at a logical schema. So, this property of a database schema is very required which we say is a physical data independence or the ability to change the physical schema without actually affecting the logical schema or the view level. So, that will be a critical factor that will have to keep in mind.

(Refer Slide Time: 14:16)



So, next is data models.

(Refer Slide Time: 14:20)



Data models are a collection of tools that describe the data because we are talking about a database system. So, certainly the main focus here is to be able to model that it to be able to represent the data, so that talks about relationships between data, it talks about the meaning of the data the data semantics, it talks about data constraints. For example, it is an account balance, we just refer to account balance in the account schema in the instance, now, the question is will can the account balance be negative, the answer is yes or no.

If the bank mandates that I can only withdraw as much money up to which I have deposited, then account balance cannot be negative, but if the bank is giving me the facility to overdraft then I may be able to draw more money than I have actually deposited. So, my account balance might note negative. In some banks it could be that the bank says that well there is a minimum balance. So, minimum balance is 5,000. So, which means that my account balance cannot go below five thousand rupees the bank will not allow those transactions. So, these are examples of different constraints that might exist in the real world, and therefore, will be required in terms of the data model representation.

So, there are several data models that exist today. The most widely used the most popular and most powerful in terms of a certain section of database applications which we are commonly interested in is a relational model and that is what we focus in this course. We

will have a major discourse in terms of the relational model and lot of things will be developed based on that. But it is not easy to directly design a database in terms of the relational model, because you first need to understand the real world in which the design is happening.

So, we normally start with a less formal model known as the entity-relationship data model or an ER model, ER diagram, these are commonly called. So, if you recall your knowledge about object oriented systems, and if you happen to know UML, you already know about ER models and corresponding class diagrams that that result. So, we will talk briefly about your model and show you how to do modelling on the real world in terms of the ER diagrams. But, then they are not actually models which the database systems directly used they are subsequently converted to some relational or other model and which the database systems will use.

Next is a object-based data models. You all would be knowing familiar with the fact that objects give a better power to represent the system which objects are not like simple strings or numbers or characters like that, they are encapsulated concept of an entity which can be manipulated in a certain way. So, like in the real world, you have several objects, it would have been nice to have similar objects in the databases. So, quite a few database system have been designed used which are object-based database systems. So, there are models for those. However, we will do little of that in this course, because it is little bit advanced in notion.

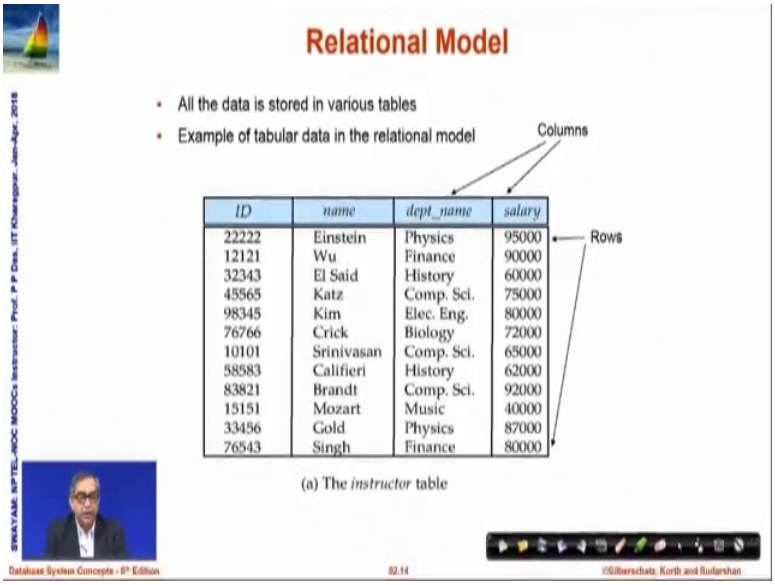
The other model, which has become very popular is called the semi-structured data model. It is primarily in terms of XML. I am sure all of you would be familiar with the basics of XML, which is extensible mark up language in which you can create use tags and use different mark ups to describe the data. You can say this is the field and this is the value kind of. And this is become a very nice way to represent the data. And XML or the semi-structured models are particularly useful today to exchange data between different systems.

I may be using a my SQL kind of database system, my friend may be using an oracle system, and we need to exchange data tables between these two, these two systems will represent the data in the in physical schema which are not may not be compatible. So, we can represent both of them in terms of XML models convert the data. So, I convert the

data into XML, give it to my friend and my friend can import from that XML into the database. So, it is a XML is a data model which is frequently used in terms of data exchanges.

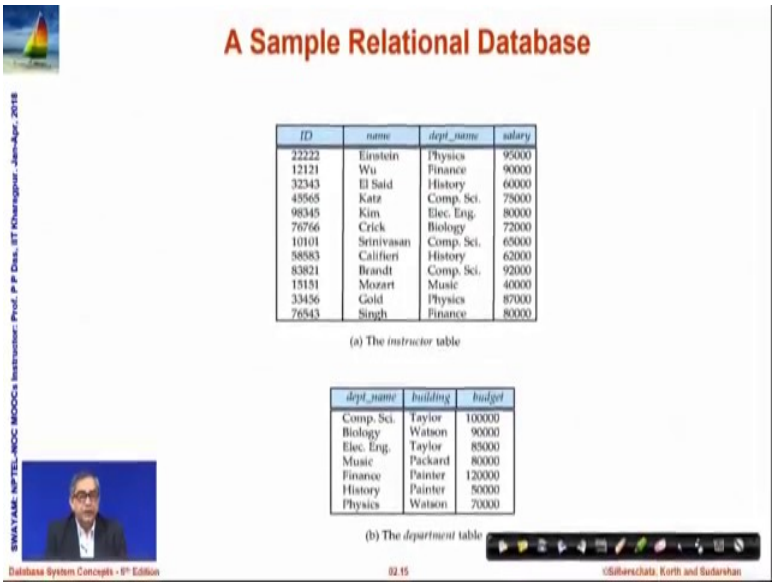
Then there are several other models like the network model, hierarchical model which used to be very popular in the early days of database systems before relational model came into force. They still exist in terms of the some of the databases. And some of the newer emerging big data databases actually we have started using this old concept in a new way again. So, this is a overall set of data model.

(Refer Slide Time: 19:58)



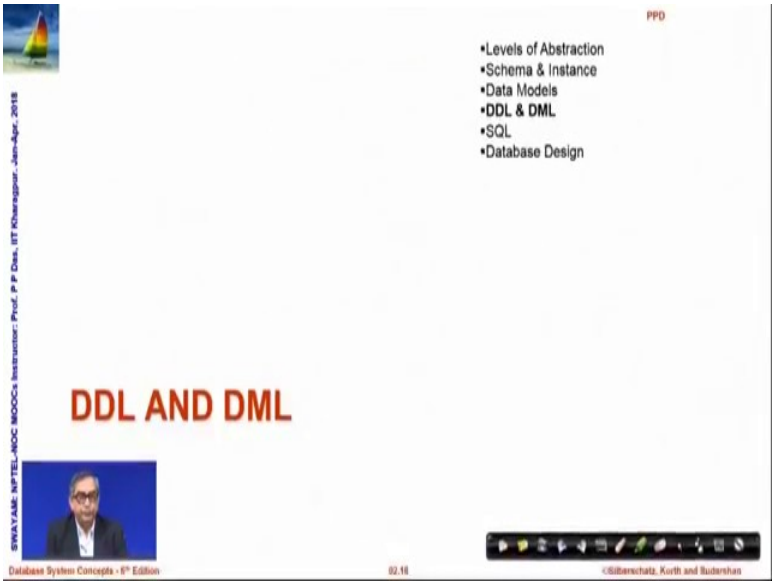
So, here I am just showing an example of a relational model data which is simply looks like a table. So, you can see that on top row in the blue are the names of the different fields which describe the schema. It says that it has an ID, it has a name, it has a departments name, it has salary. They are trying to describe a particular instructor, and then a whole lot of records rows in that table, which are every row individually is a particular data entry or a record. So, columns are attributes, and rows are records that the relational model described.

(Refer Slide Time: 20:40)



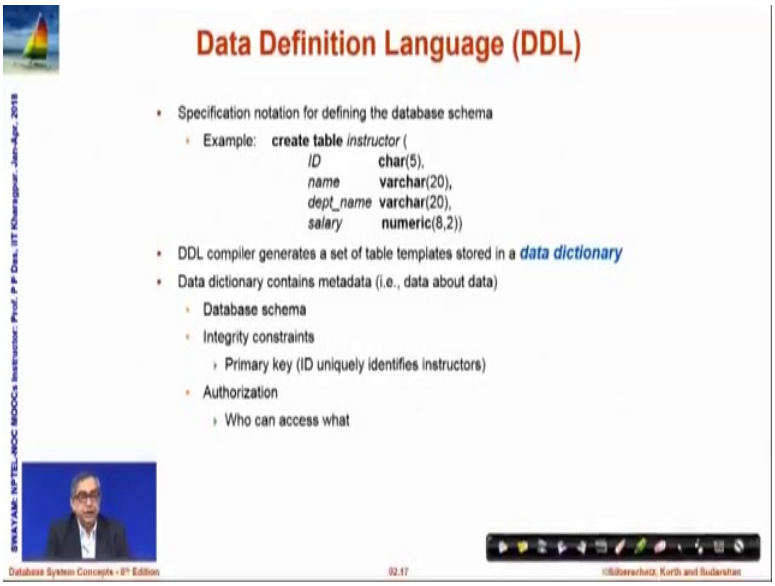
Some more of that the instructor table along with the department table. So, the table below describes details of a department, so that has its own schema and the individual records. We have of course seen similar instances already in terms of the customer and accounts instance that we have just discussed a couple of while ago.

(Refer Slide Time: 21:03)



Let me introduce these two terms DDL and DML.

(Refer Slide Time: 21:10)



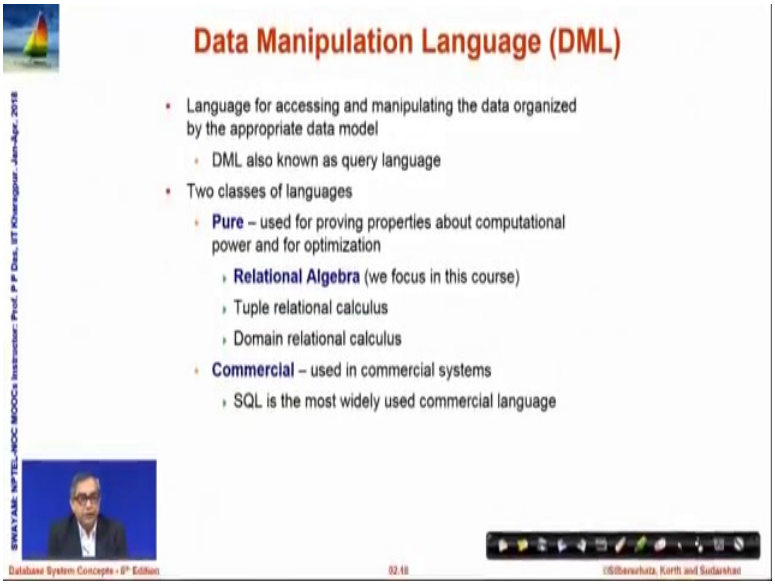
DDL talks about data definition language. So, what the concept wise what we are trying to say is certainly we have a schema and we have instance. So, we need certain language constructs to be able to define a schema and certain other language constructs to be able to manipulate the data in that schema or they are basically manipulate the instances. So, DDL is the language or part of the language which is used to define and manipulate the schema of a database that is why schema is a way to define a database. So, it is called a data definition language.

So, you can define that I will am going to have a table called instructor which will have four different fields, each having certain types of data. So, it says that the ID will be a five character data; the name would we would have a variable length, because you cannot say that the name will be of a fixed length, but it will be a variable length that is what varchar is, but the length will not exceed 20. And similarly, salary will be a numeric data with up going up to 8 figures, and having a decimal part having two parts.

So, this way of defining the schema in terms of the different attributes and their types or columns in the table or trying to define the structure of that table is a main issue of the data definition language. So, the data definition language compiler who generates a set of tables in the data dictionary, where the data dictionary basically contains metadata as I said the schema is nothing but a metadata about the database tables. So, which will have the database schema, it will have different integrity constraints, it could say that well the

account balance cannot be negative or account balance cannot be less than the minimum balance. So, these are different integrity constraints. It could say that this is the primary key, we will talk more in more depth in terms of what is key. And it could also specify the authorization as to who is allowed to access which part of the data and so on, so that is these all are part of the schema definition and forms the DDL of the language.

(Refer Slide Time: 23:36)

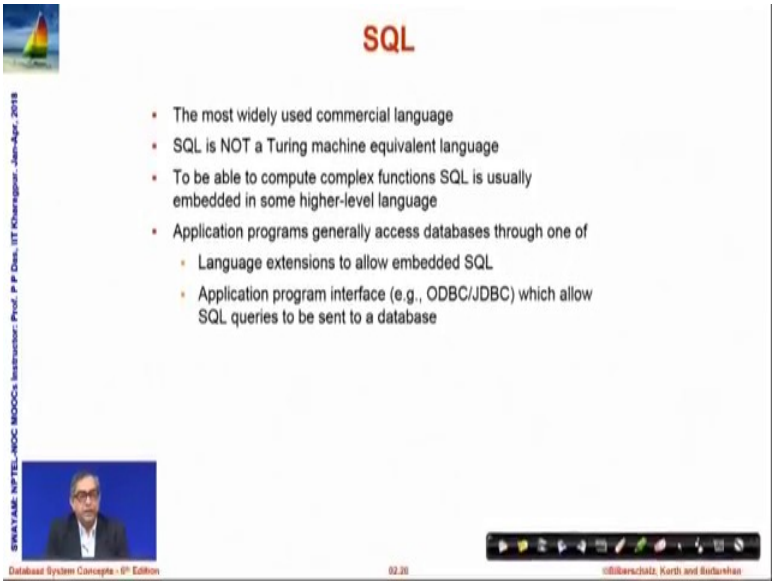


In contrast, the data manipulation language is a language for accessing and manipulating the data organized. So, it is for access, update, addition of new records, deletion of existing records and so on. And very commonly we will refer to the data manipulation language as a query language, because this is what you want to know what exists in the database. So, the query language will be designed, they are designed primarily in one of the two ways. One group of languages is known as a pure language, they are more mathematical in nature. They have a formal basis that can you can prove that whatever do you do in these languages are correct, and will give you the correct result.

So, they are different languages based on the relational model, they are called relational algebra, tuple relational calculus, domain relational calculus and so on. Of these three, we will in this course deal only with relational algebra. There are mathematical proof which show that whatever you can do in relational algebra you can do it in tuple relational calculus and vice versa. Similarly, whatever you can do in relational algebra, you can do in domain relational calculations and so on. In one sense that these languages

are equally powerful; the same thing can be done in any one of them, but we will just take the simplest of them and study here in terms of the relational algebra, but these are more mathematical representations are not easy to write as a program. So, normally we will use certain commercial query language which is called SQL for most of our applications and we will do the coding in that.

(Refer Slide Time: 25:19)

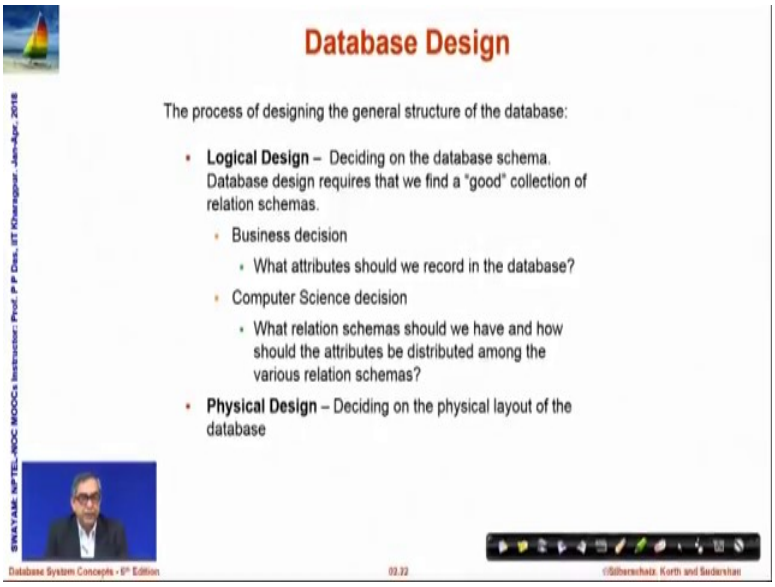


So, SQL which is a most widely used commercial language and mind you this is not a Turing equivalent language which means that everything that can be that need to be computed cannot be computed in SQL, there are certain computations which SQL cannot do. It is a limitation; it is a restricted language. So, often SQL is used in conjunction with some common high-level programming language like C or C++ and so on. So, whatever is there can be done in SQL in terms of data manipulation will be done in terms of the relational model, but there could be additional logic that needs to be built in, in terms of the high level language. So, application programs are typically written through them. So, we can do this through a process of embedding that is put in SQL as a part of a C program or use certain libraries which can actually take a query from C, and fire it on the SQL database.

(Refer Slide Time: 26:26)

So, we will see how to do this in the course of time.aspect.

(Refer Slide Time: 26:32)

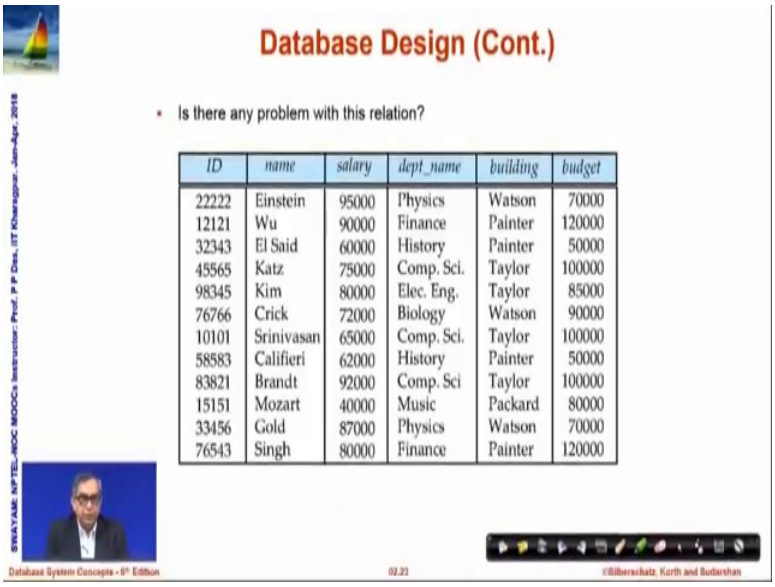


Coming to the database design this is a process through which the databases need to be designed. And certainly the first part of the design is the logical design where you want you need to identify what are the schemas and you know what are the constraints that apply, what is authorization required. And first set of decisions those are related to the business as we say. Business means it is basically comes from the domain. So, it is if I am doing a university database, the business decisions will come in terms of you know I

have courses, students, instructors, and the instructor teach courses, can an instructor teach multiple courses, can a course be taught by multiple instructors these kind of business decisions are critical for the database design.

And then there is a whole set of computer science decision or the data based decisions to decide as to if this is the kind of business information that you want to keep in the database, then what is the kind of relation, what is the kind of schemas that we should use what are should be the attributes, which attribute should be of what type what should be strain, what should be numbers and so on. So, these are formed the basis of the physical logical design. And of course, we then need a physical design which decides on the physical layout of the data, what are the different database files, how they should be indexed and so on.

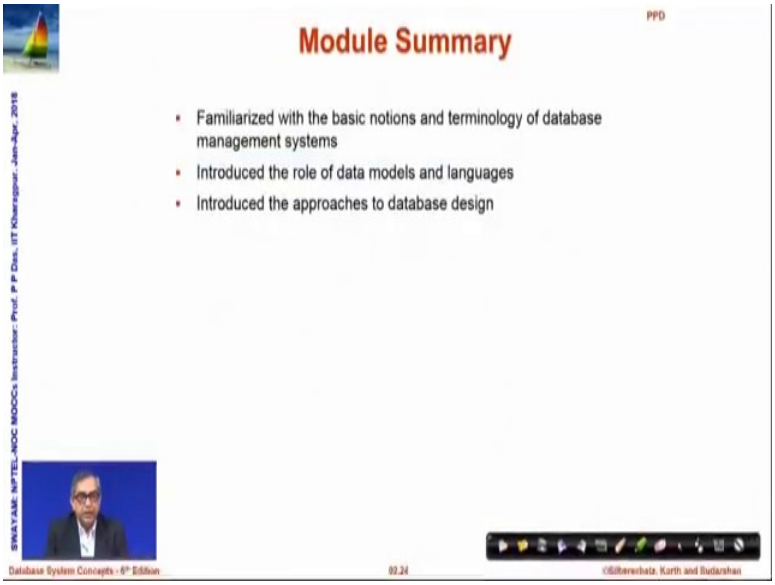
(Refer Slide Time: 27:53)



So, here we are showing an example table. So, it has multiple fields. It shows the instructors expanded form of the instructor table you saw earlier. It is expanded with the departments name and the building in which it is housed. So, if you look carefully that this certainly comes from the business decision that you need to know the department to which an instructor belongs and certainly you need to know the building in which that that department exists. So, knowing the department of the instructor and the building of where that department is are critical, but the question is this a good design, is this so we

will discuss as to when why this is a good, this may not be a good design to represent the data.

(Refer Slide Time: 28:41)



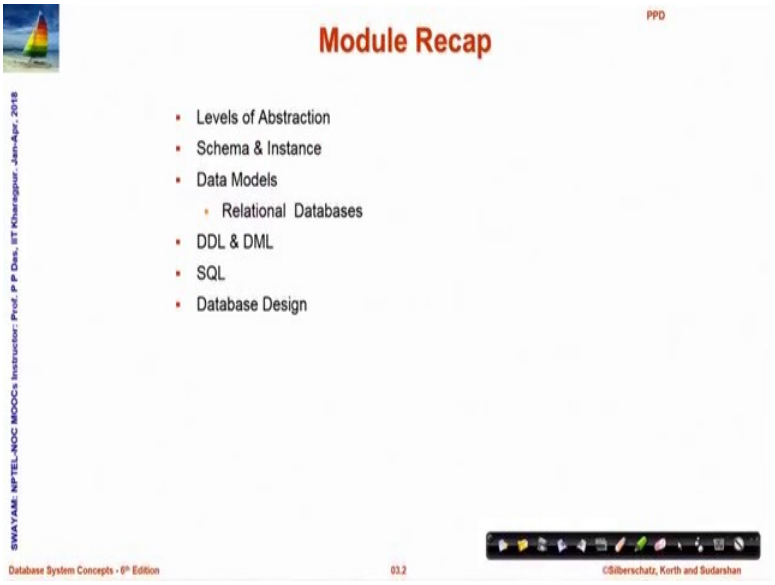
So, in this module, we have taken you through the basic notions and terminology of database management systems, highlighting primarily the levels of abstraction, the schema an instance, the basic data models the languages that you need DDL, DML and the commercial SQL language. And we have also tried to give you a glimpse of the approach that is required in terms of the database design. We will elaborate on this more in the second part of our introduction to DBMS which will be taken up in module 3.

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

Lecture - 03
Introduction to DBMS/2

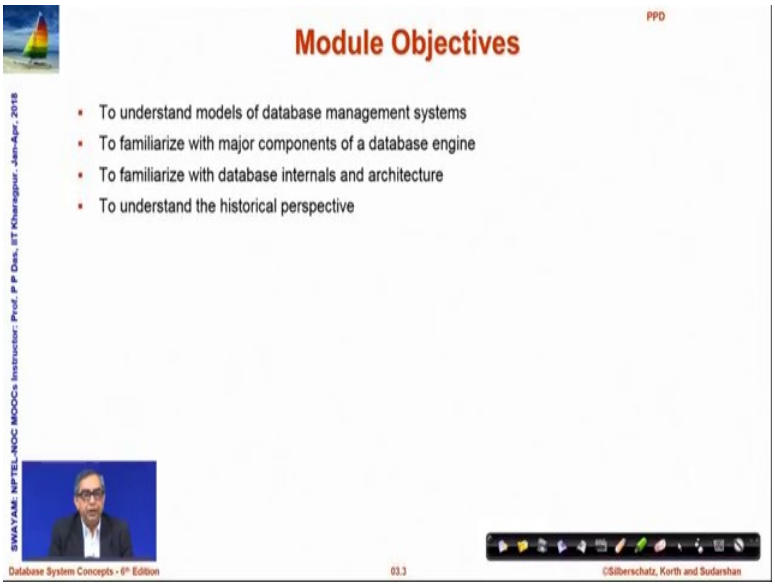
Welcome to module 3 of Database Management Systems course. We started discussions introducing the database management systems in module 2. This is the second and concluding part of that discussion.

(Refer Slide Time: 00:37)



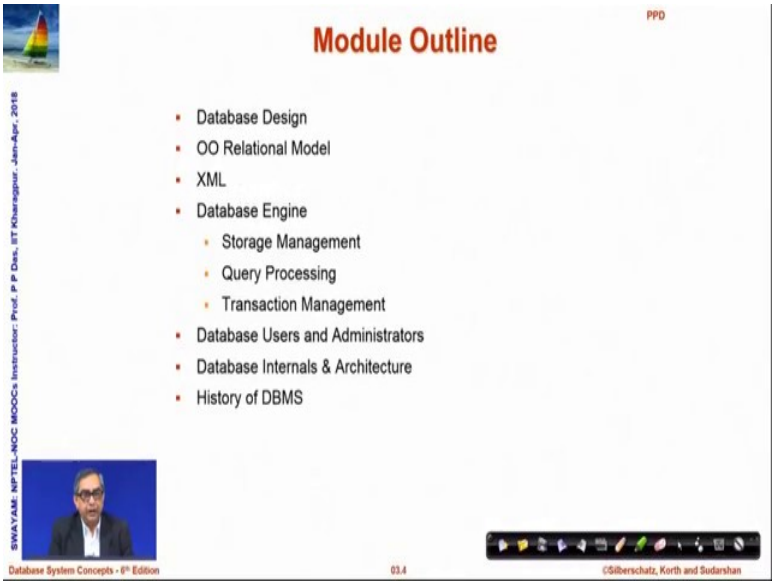
So, this is what, these are the aspects that we are discussed earlier starting from level of abstraction to the outline of database design.

(Refer Slide Time: 00:47)



In the current module, we would like to understand the modules of database management systems little bit more and we will try to familiarize with the concept of major components of database engine, will elaborate on those and will familiarize. So, with the basic architecture of a database management system, some of the internal components and we will present a brief historical perspective of the DBMS.

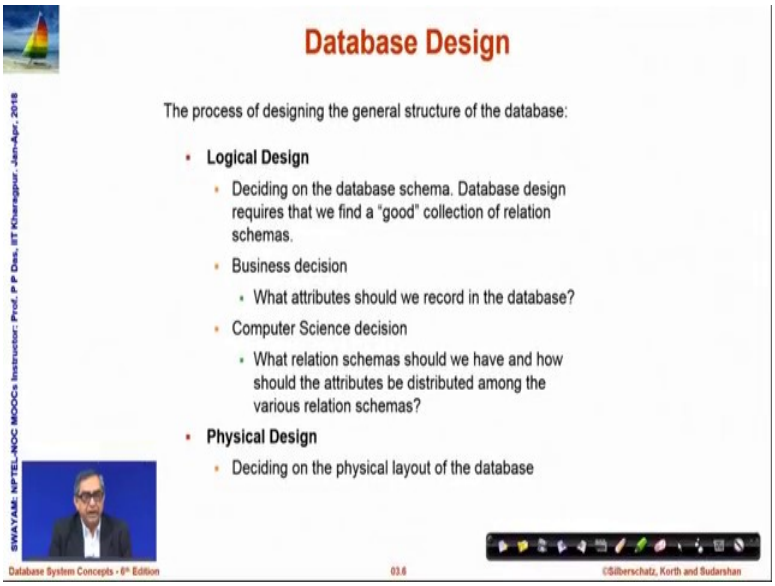
(Refer Slide Time: 01:21)



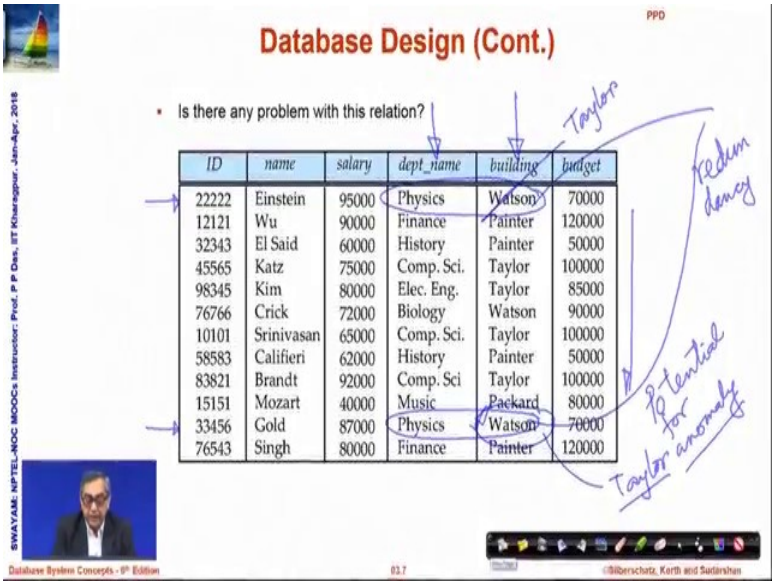
So, this is the outline that we will follow.

(Refer Slide Time: 01:25)

(Refer Slide Time: 01:34)



(Refer Slide Time: 01:46)



ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

So, we have already discussed about the database design, I would like to raise a few further issues about that. So, we have seen that, there is a logical design which is driven by the business decisions and refined by the computer science decisions, there is a physical design as well; and based on that we had at presented this particular table asking, whether, this database is a good design or not.

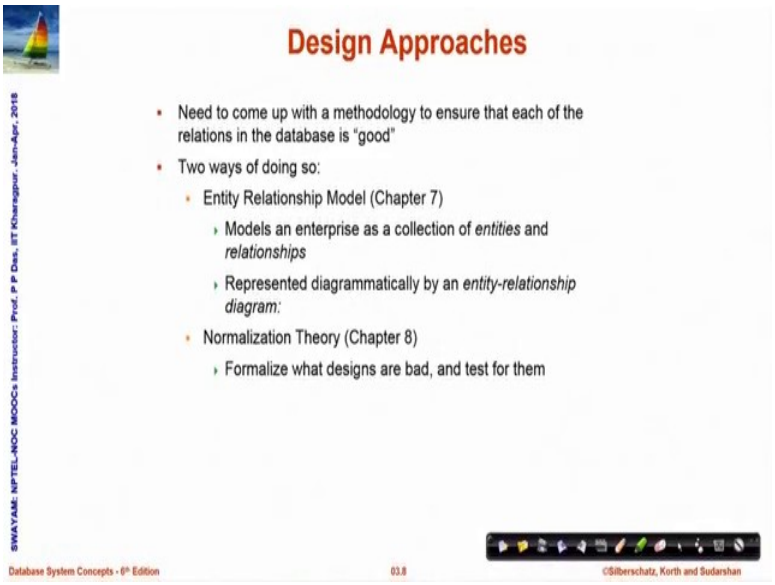
So, let us have a little look into this for example, we have introduced the department name and the building in which the department is housed. So, if we look at there are multiple instructors say, let us say Professor Einstein, who teaches in the Physics department, that is housed in the Watson building and if we look through there is a Professor Gold, who also teaches in the Physics department and naturally that is housed in the Watson building.

Now, the question is; so, Physics department, if it is housed in the Watson building then, all the instructors in this table, all the instructors who are part of the Physics department, would have their department housed in the Watson building. So, there is a certain issue of redundancy in these two, that is, the same information is given more than once, which is not a very desirable thing.

The consequence of this could be suppose, tomorrow the university decides to move this Physics department from Watson to the Taylor building. This will mean that once this is moved, then this Watson will have to be changed to Taylor, also this Watson will also have to be changed to Taylor. All instances of Watson that corresponded to the Physics

department in this table, will have to be changed to Taylor, and that is not a good scenario. So, it is not only that we have redundancy, we have potential for anomaly; that is program the application programmer might forget to update the building say, for this entry then, we will be in an inconsistent database. So, to put it in simple terms that this is not a good design and there are several issues to consider, in terms of whether some design is good or some design needs refinement.

(Refer Slide Time: 04:22)



So, we need to come up with a methodology to ensure that, each of the relations in the database is good. So, we primarily follow two approaches in doing this. One is, using the entity relationship model, which models the enterprise as a collection of entities or concepts or if you are familiar with the object orientation classes and the relationships that hold between these entities.

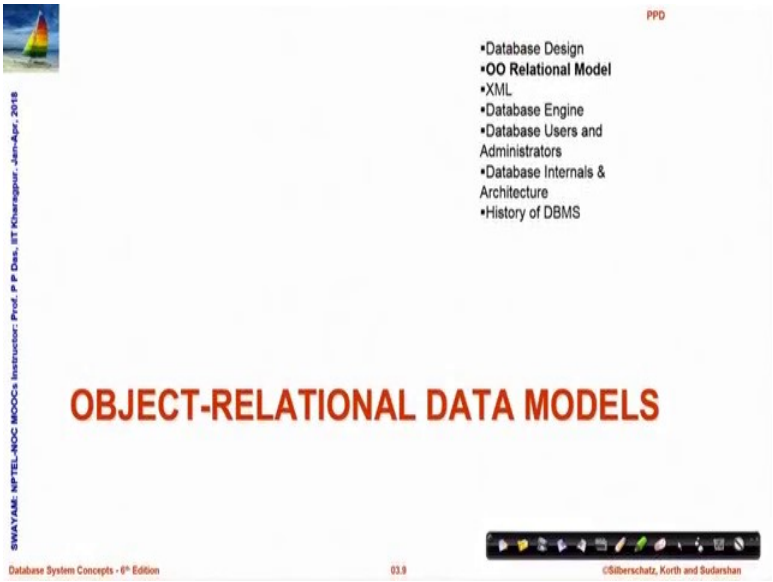
So, in a university database the entities are students, courses, teachers and the relationships are a teacher teaches a set of courses, a student attends a set of courses and so on, the teachers supervise a set of students for projects and so on and then represent them diagrammatically in terms of a ER diagram entity relationship diagram and once that has been done then, we try to follow a certain normalization theory.

This normalization theory tries to capture that what are the properties that must hold in this database design, that must be satisfied on this database design, in terms of what is known as database dependencies, there are, varied forms of dependencies functional

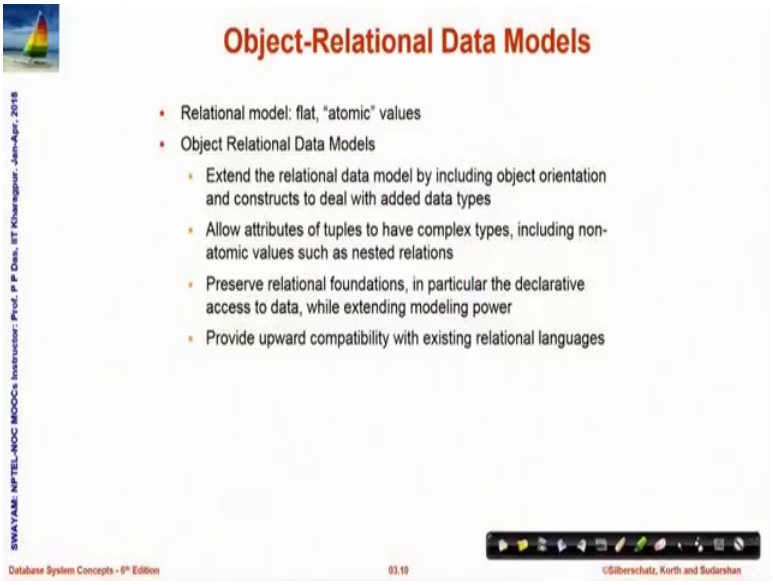
dependencies, multi value dependencies, joint dependencies and so on and try to formalize and evaluate whether a design is good or it is bad, test them for quality and then normalize to make them better; make them the best possible that can happen.

So, this is something that is starting from the entity relationship model, which captures the real world to the actual database schema, there is a process of representation and then capturing of ground truths, that should hold in the database system and then normalize the database is a basic requirement of the design approach.

(Refer Slide Time: 06:34)



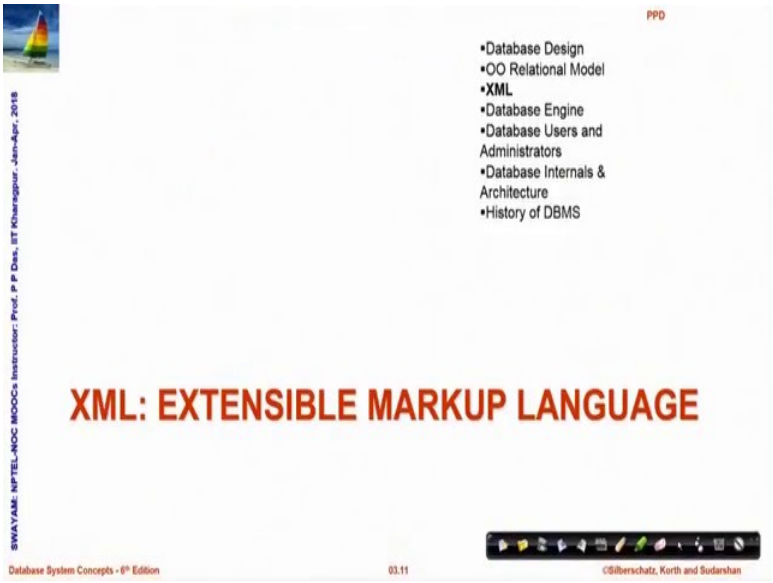
(Refer Slide Time: 06:39)



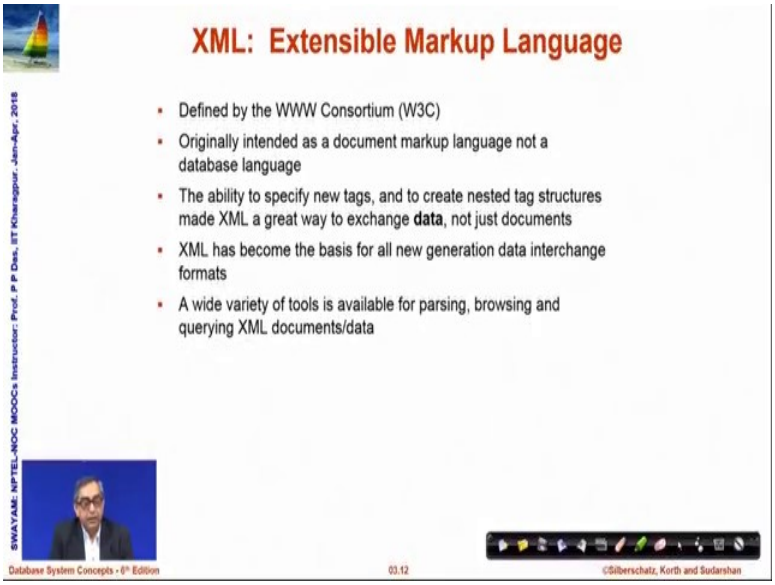
We have talked about object relational data models for a few more points about that, that in a relational model everything is flat, every value is atomic, in the sense that everything if you, look back and think in terms of C then, every field is a value which can be a simple, you know, built in type like integer, like fixed length string, variable length string, a floating point number like that, but I cannot have a composite you know, object kind of fields.

But in a relational data model we extend in the object relational data model, we extend the relational model by including the object orientation and the constituent constructs to deal with added data types, higher data types where attributes are allowed to have complex types, non atomic values that may allow things like nested relation; that is a value could itself be a relation, could itself be a table but, we try to preserve the relational foundation and we will see what those foundations mean and provide upward compatibility to existing relational databases. So, this is what the basic concept of object relational data models are and as I said that, we will just glimpse through it, but this is not the primary objective that we will try to cover.

(Refer Slide Time: 08:04)



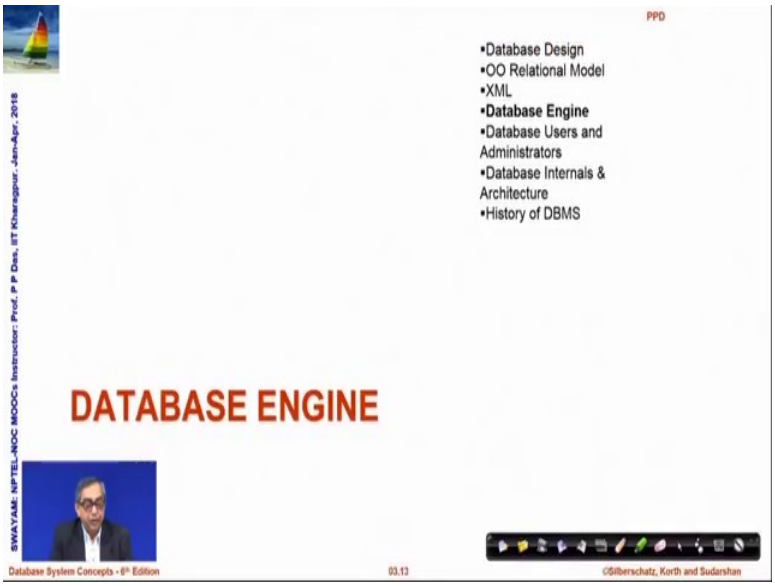
(Refer Slide Time: 08:07)



In contrast, the XML extensible markup language was defined by W3C, and it was originally intended for marking up document languages. It was not designed as a database language, it was designed for marking up. So, it is kind of saying that this particular element should be put in capital, this should be in blue colour, this means a verb, this means a paragraph, there should be a page break here, those kind of markups, But subsequently, it turned out that the way XML deals with different components in terms of tags, and the ability to create nested tags, makes a great language for exchange of data, As I explained in the last module also.

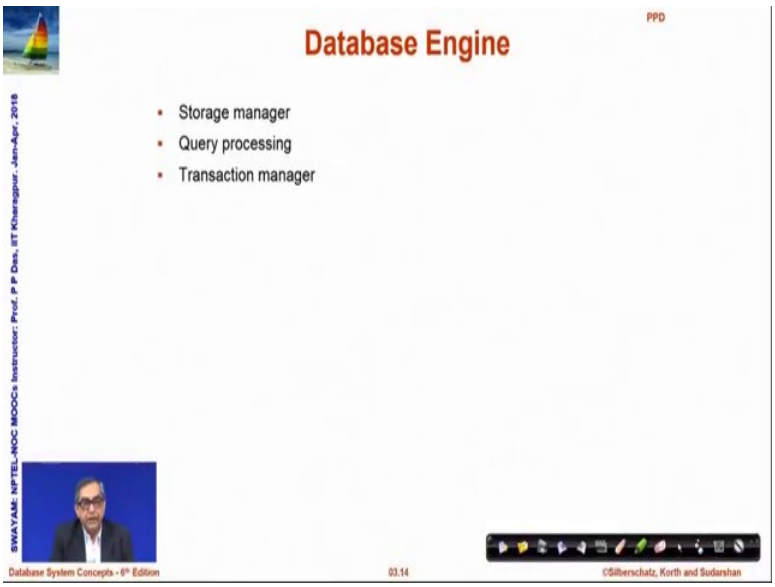
So, has become the basis for all kinds of new generation data interchange format. So, as I explained, that any database should be able to convert the data instances of the tables in terms of corresponding XML format and then you take it to some other database, in which you are intending to interchange the data and that target database should be able to import from that XML structure, and it has become widely available that you have different tools for parsing, browsing and querying XML content document data and so on. So, if you are familiar with C programming, I hope so you are! You can look up certain XML parsing and try out, there are great tools to learn.

(Refer Slide Time: 09:53)



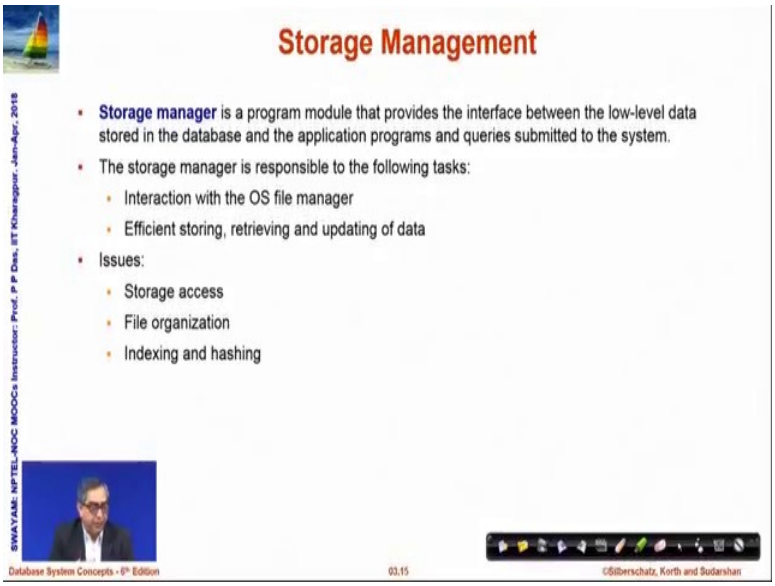
Moving on, let us briefly look at what is the core of a database management system, the database engine.

(Refer Slide Time: 10:03)



The database engine, primarily contains 3 major components; the storage manager, the query processing engine, sub engine and the transaction manager.

(Refer Slide Time: 10:14)



The storage manager, is a module or a collection of modules in a database management system, that provide the interface between the low level data and the application program. So, we have looked at the storage manager is the one, which is a bridge between the physical level of abstraction and the logical level of abstraction, then finally, to the view level of abstraction. So, the storage manager has to deal with interactions with the operating system on which the DBMS is kept, the file manager of the operating system, it is responsible for efficient storage, retrieval update of the data, it is responsible to make sure that if there are certain problems in the file system, then the data is not corrupted and so on.

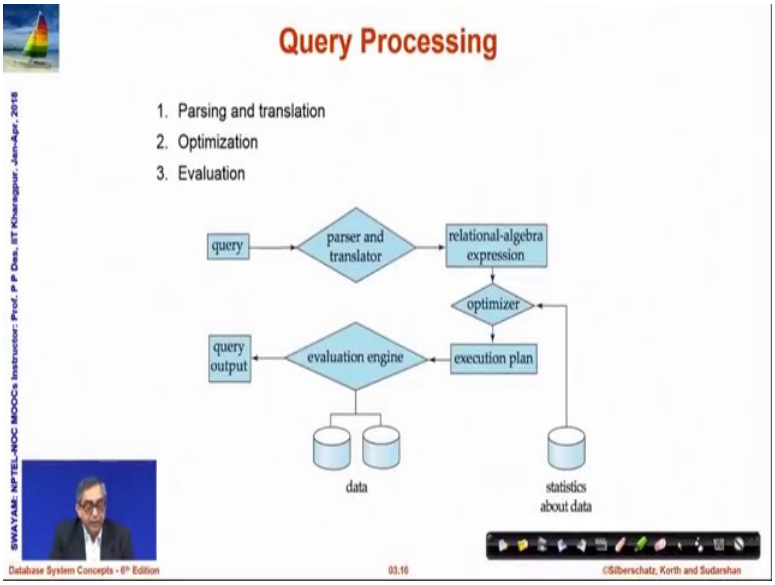
So, the issues certainly that involve are :- the access to the storage, the organization of the files and very importantly indexing and hashing and we will talk about the concept of indexing later in the course. It primarily says that, if I want to, for example, you can simply understand that if you have a large chunk of data that you want to organize for efficient search then, you can use the binary search tree in simple algorithm terms. The binary search tree needs to be organized in terms of one data component.

We say that, well there is one value based on which you can say that, comparison is done. So, that at every node if that value is smaller, you go to the left sub tree, if that value is larger, you go to the right sub tree and so on. So, if we want to organize the

records of a database system in terms of such a binary search tree, then the question certainly is, which field do I use for the search tree comparison.

Now, whatever field I used for search tree comparison, on that field the searching would be very efficient, but if I want to search on a value for a different field, the searching would not remain that efficient. So, indexing is a mechanism by which, you can actually create auxiliary search trees on multiple fields. So that, the search on multiple fields can be made efficient and we will talk about this later when, that particular module comes up, but the storage manage has to deal with such issues.

(Refer Slide Time: 12:43)



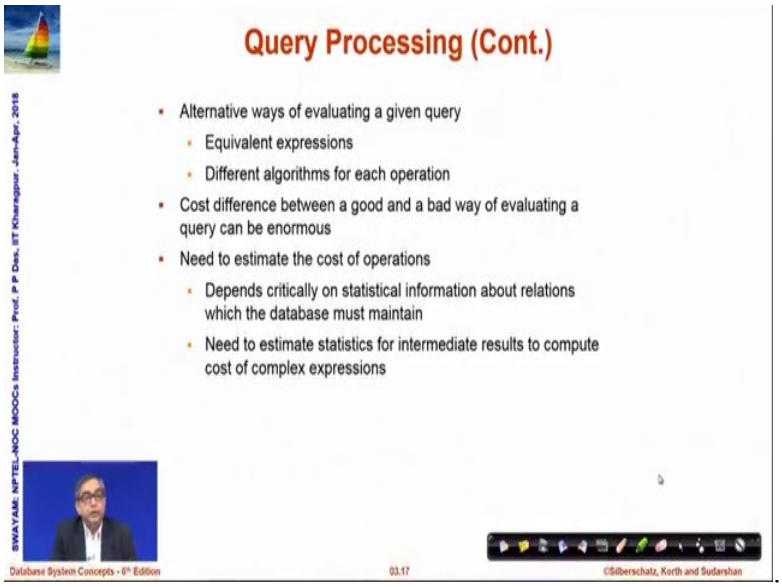
Moving on, the query processing is, if we have already talked about the language; the DDL, the DML, the query language. So, it is some kind of, like the C program, it is some kind of a text based programming code. So, naturally that code needs to be parsed and translated, as we typically do in a C compiler. So, there needs to be a query compiler. So, it parses and analyses the code, but translated unlike the C program, which translates the C program into an intermediate code and then into the binary instructions of the machine, the assembly binary instructions of the machine.

The query translator, translates the query into relational algebra expressions. I said that, there are two kinds of languages :- the commercial query language and the pure language. So, it translated in terms of a program in the pure language, which could be a

relational algebra language and then it tries to optimize. So, that is a critical term to be noted that there is an optimizer.

So, this optimizer is a critical component, which tries to make sure that the query, when it is run on your data will run with the most you know, least amount of time in an effective manner. So, then an execution plan needs to be decided, we will be able to understand this when we go to the actual relational algebra execution plan, basically says that if, there are multiple operations in that query to be performed, then how those operations, in which order they should be performed and where should temporary tables be used, where they should be skipped and so on and then, once that has been done then it passes on to an evaluation engine which actually runs that query on the data that you have, the instances of the data that you have, and that brings out the resultant query output which is another table of results that we get. So, this query processing is a core part of a database engine, which actually allows us to write text based queries and relative data efficiently, change-update data efficiently, insert data efficiently, and so on.

(Refer Slide Time: 15:12)

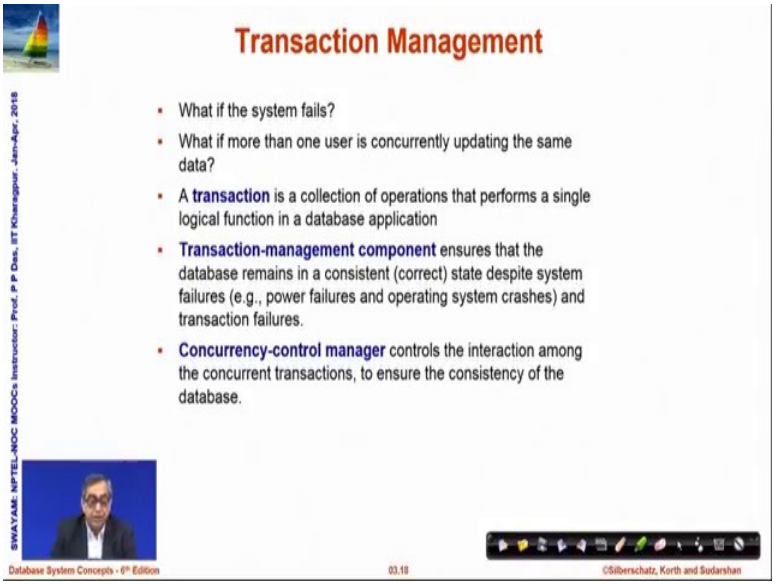


So, when we do this, we need to look at alternative ways of evaluating a query. There could be different ways to write the same thing, these are called equivalence expression, equivalent expressions and what are the good algorithms for doing each and every operation, there is a cost between good and bad way of evaluating. So, this has to be understood that, the same thing you can compute in a, you have seen this similar

concepts in normal programming language also, I mean we have seen for example, for sorting the several ways to sort and some are better some are not as efficient.

So, if the similar things in terms of a query need to be evaluated and the cost between good and bad ways need to be figured out. So, then we need to estimate the cost of every operation. It depends on the information of what has happened in the past, the statistical information and need to estimate those statistics for intermediate results. These are a couple of things that the query processing sub engine in a database will do.

(Refer Slide Time: 16:36)



So, beyond the storage manager and the query processor we have a transaction management system, which is very critical and core of the database system. It primarily has to deal with two fundamental issues of a database. One, what if a system fails; see, database systems are unlike the programs that you have written so far. A program starts execution ends, the program always deals with transient data, the data did not exist before your program started, it ceases to exist after your program ends. So, a program; however, complicated; however, important has a limited lifetime.

A database in contrast, has a much longer lifetime which deals with persistent data, that is very important to understand ,that is the each application whether I am doing a bank fund transfer, whether I am making a credit card payment, to whether I am checking the balance or I am booking a railway ticket, whether, I am purchasing a book from Amazon, each one of the applications are like the normal program, it has a fixed lifetime.

If I start it, I do certain operations, I am done with it, but the data that is behind it the data of my accounts, my account balance, my transactions, my different bank charges, all that need to stay on and on and on and beyond every particular operation that I have done on the database. So, which means that if this database system fails, at some stage for some reason, then we have an enormous impact of that and that is not something that we can absorb that something that we can accept.

So, a database system has to come with the concept of recovery. It must be possible, if the system fails, it must be possible to recover it, to a certain earlier point where it is consistent. So, transaction management system is responsible to guarantee this kind of recover ability of databases. Then, the other question that we have discussed about earlier also, is a multiple users are you accessing the same database, the same set of data, at the same time. So, what how to make sure that more than one user can concurrently use an update without the data getting inconsistent, that is as I had mentioned, there is only one seat available, one berth available on a particular train, on a particular date and two users at the same time has initiated a booking.

It should not happen, that both of them get the booking. So, one should get, one should not get and that needs to be the complexity is high, for this kind of you know, decisions because in the databases, as applications are significantly distributed, Indian railways have no idea of who is going to do what booking, of which berth, from where at, which point of time. So, transaction management system is, as the name suggests defines something called, a transaction which always keeps the database consistent and operable. So, a transaction is a collection of operation, that performs a single logical function in a database application. This is very critical. It is a collection of operations and performs a single logical function.

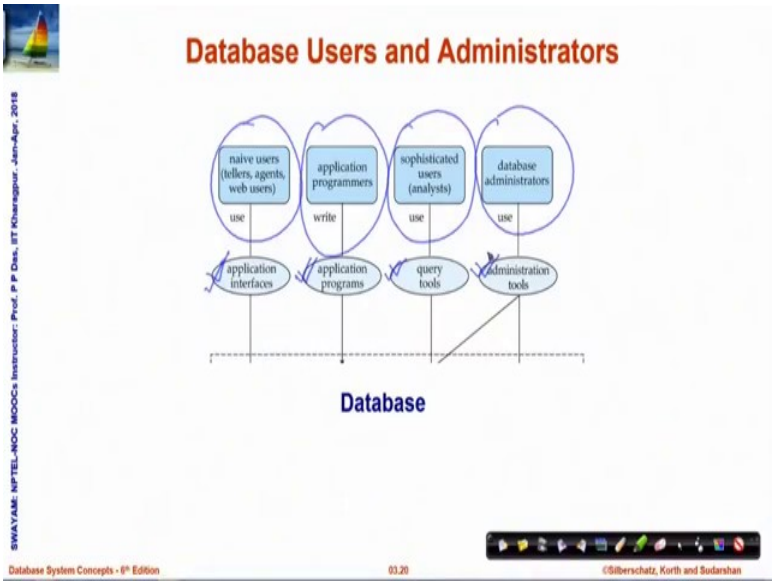
So, it does not do anything and everything, it just does a single particular logical operation and that is what forms the transaction. So, a transaction management system is a component, that ensures that with all these transactions happening, hundreds and thousands of transactions happening every second in a database system, in a typical database system; the data should still remain consistent, in a consistent state, in spite of failures, in spite of concurrences, it must make sure that at no point of time it should happen that, an amount has been debited from an account and has not been credited to account or an amount has been credited to an account and has not been debited to the

corresponding account or the same seat same berth is booked by two persons at the same time and so on, so forth. So, this also includes concurrency control manager, which controls the interaction among different concurrent transactions, which ensures the consistency in the database and provides the total safety.

(Refer Slide Time: 21:32)

So, in total, we have seen the different components of the database engine comprising the storage manager, comprising the query processor, and the transaction manager. Now, we will just have a quick look in terms of who are the typical users of a database system.

(Refer Slide Time: 21:57)



So, if we see grossly, the users of a database system can be grouped into, I mean you can group it in multiple different ways, but this is a typical way to group that, you have the naive users, those like the secretarial staff, who sits at the teller of the bank. Now, that person does not know database management system, but that person just needs to know the particular application. He knows a few set of screens; graphical screens, what needs to be filled up, where which button needs to be clicked and so on and can use this database through an application interface.

So, this is a lowest level of user. Then, you have the set of application programmers, about whom I talked about in my course overview presentation, that application programming is a big chunk of you know, IT services that databases need, who actually write the application programs, while the naive user is similarly using it application programmers are responsible for writing coding this application program. So, they need to understand the database designs they need to understand how to write the query language, how to fit with the application data, input, output all the systems.

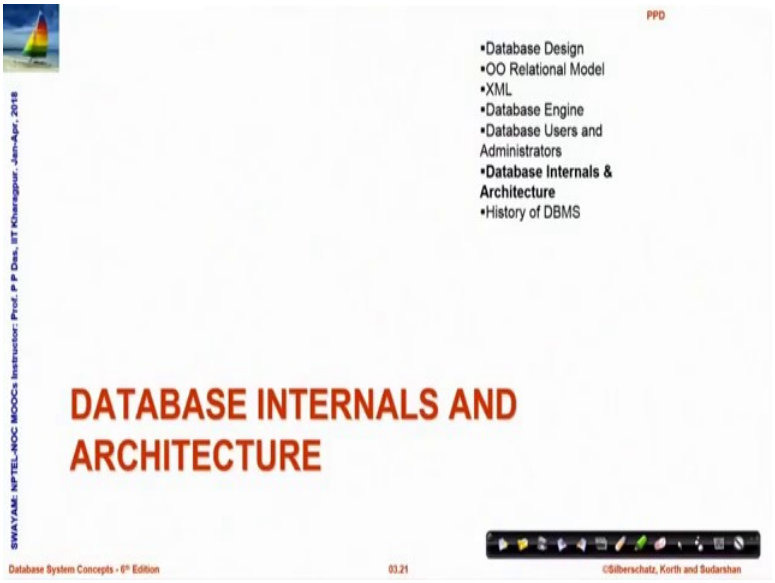
The next levels are the analysts, who are called the sophisticated users. So, they design different kinds of query tools, they are responsible for the design of the database, that is a schema the different constraints, the authorizations and so on and manages that over a period of time, when the application requirements change they might need to redesign the schema, migrate the data from an old schema to a new schema. So, analysts are higher level of programmers, they have far more solid understanding of the database management system to be able to design different kind of query tools that, the application programmer will use and at the end there are database administrators.

So, database administrators are people with specialized rights, who can do a lot of privileged operation on the database. For example, taking backups of databases, for example, creating different users. For example, if there has been a failure then how to do the failure recovery scripts, recovering the database and so on. So, they do all kinds of administration tasks, but not the usual day to day data maintenance and you know, query processing and so on.

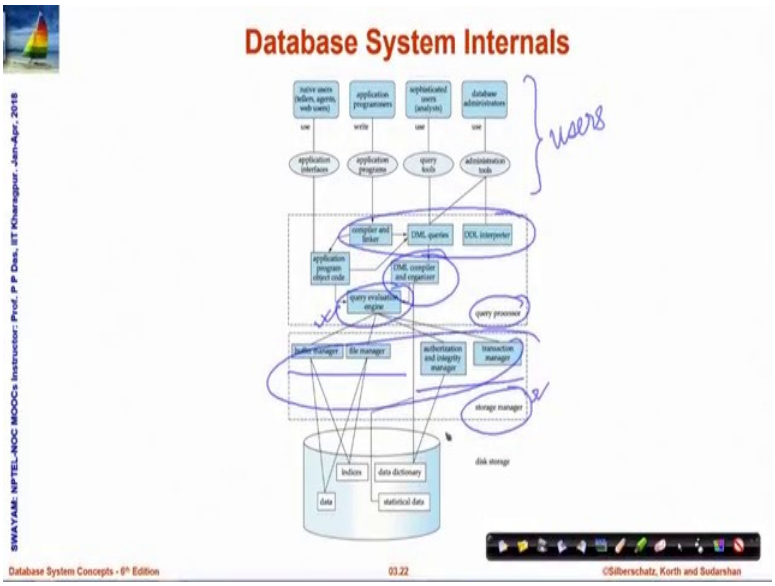
So, if you would like to know your positions then I would say that by this course, you are going to position yourself amongst the application programmers and the analysts and as I mentioned that the first half of the course is focused on application programming aspect

and the second half would be more focused on the analysis and some of the administrative will do little bit of administration, but not nearly serious administration tasks.

(Refer Slide Time: 25:33)



(Refer Slide Time: 25:35)

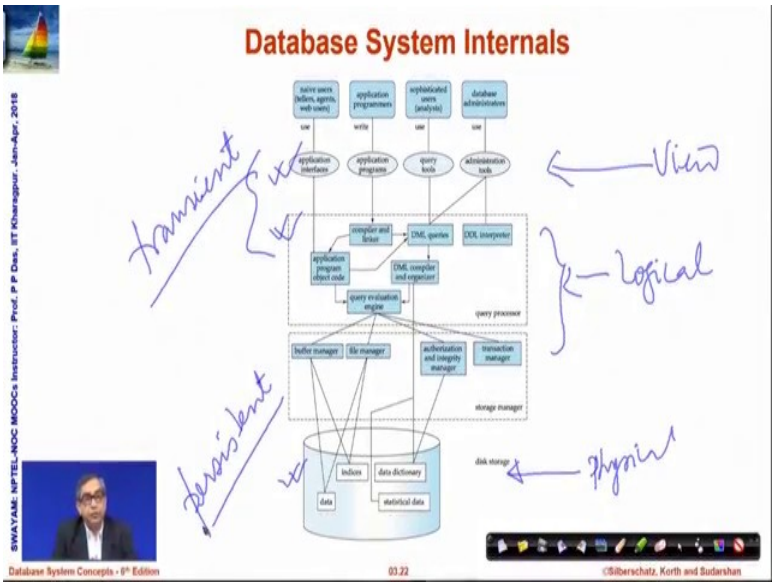


Now, we will take a quick look into the database internals and architecture. So, I will take you to this diagram. I am sorry this diagram is little small, in terms of the script size. So, please refer to the actual presentation. So, if you look at the top here is the users. So, it is trying to show what different users use. This is a query processor, that we have

known. So, the query processor gets a query and so that naturally, this query comes from the application program. So, the compiler link these are the basic processing, then the compiler organization, the evaluation engine which actually takes care of the processing of the whole query and then it goes to the storage manager which is now taking this whatever the evaluation engine needs to do, has to go through different modules in the storage manager, which we will talk about these modules, when we do have a discussion module on the storage management.

Later in this course, and we will then talk about what is a file manager and what is authorization and so on, but these are the sub components of the storage management needs to do and then the storage manager is the only one who deals with the actual data, the actual disk storage the different files and so on.

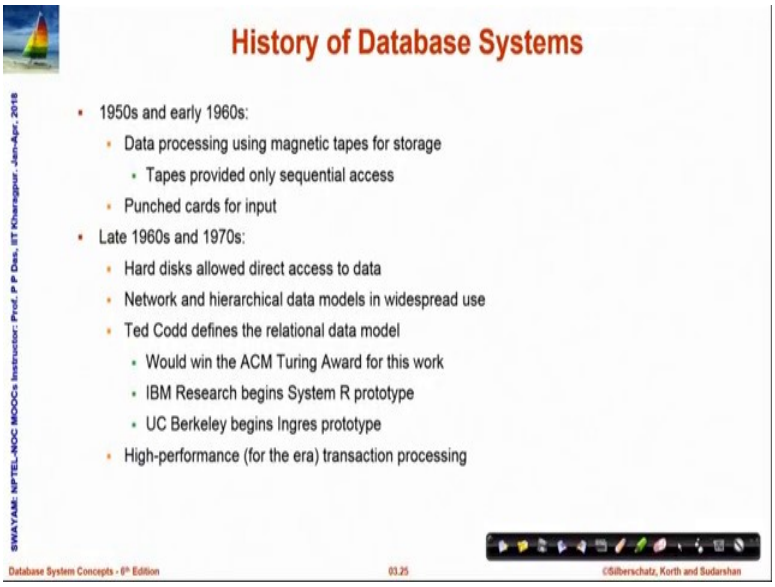
(Refer Slide Time: 27:05)



So, as you can see that, the whole system is kind of layered in terms of so, this is your basic physical layer that you have, and this is your final view layer that you have and in between this is a logical layer, that you deal with. So, you can see that the abstractions, as we had talked about are also mapped in terms of them, way the actual database system architecture is managed and finally, the data stays in the disk storage which ensures that whatever data I have is actually persistent in nature. It does not go away. Any data that stays within here or within the application interfaces transient.

So, these data are transient, they will disappear as soon as the application is over, but these data are persistent, they exist beyond this and architectures supports that whole gamut from of all applications or transiency of the applications based on the persistence of the data at the storage. So, that is a basic architectural view of a typical database systems. The actual architecture we form more complex, but we just want to take a schematic view. So, that we can understand it better.

(Refer Slide Time: 28:24)



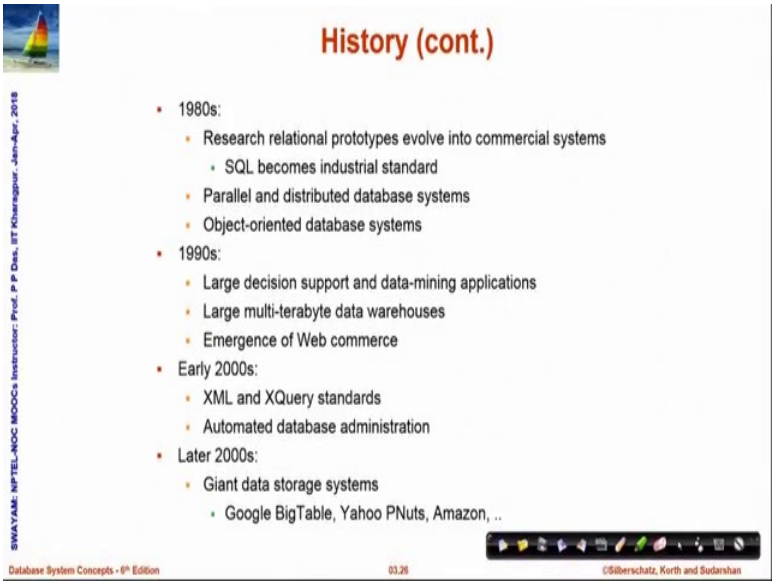
So, the actual architecture may again vary, based on the computer system that you are using. It could be centralized, we will talk about some of these at later modules. Centralized in the sense that there could be one database server or you know, a group of database servers at the same physical location connected together to which, all applications, all users will connect to, it could be in terms of a client server model which is a very typical client server model, that the programming systems are. So, that typically for example, any of the net based internet based database applications we are looking at are necessarily client server in nature.

What you are doing in the browser is a client and there is a server back far back there the multiple tiers in between them. Databases could be single processes or for performance they could be in terms of multiprocessor parallel databases, the data sets themselves could be so large that, it may not be possible to search on them through a single

processor, in a reasonable time, they could be distributed the data itself could be distributed, tables could get so large that I may not be able to keep them at a single point.

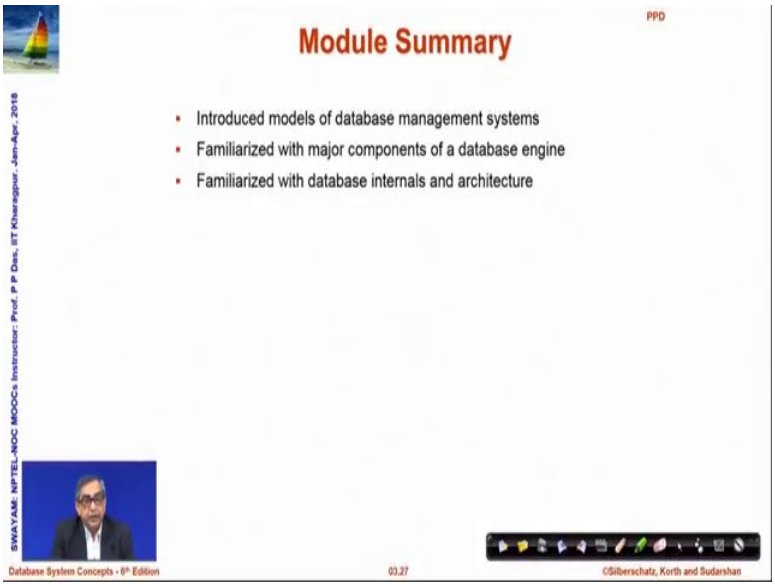
So, these are different aspects of you know, complex real life database system that exist and we will deal with some of those aspects over the course of time, but you have to keep in mind that a final architecture of a database and its associated application will have some of these factors that you will need to know and maybe decide on in terms of history well and I will not go through each and every point here. This is more for completeness and to give you an essence of how things have been going. So, database system started in the 50's and early 60's, then major developments of these relational models and all that started happening in the 70's.

(Refer Slide Time: 30:36)

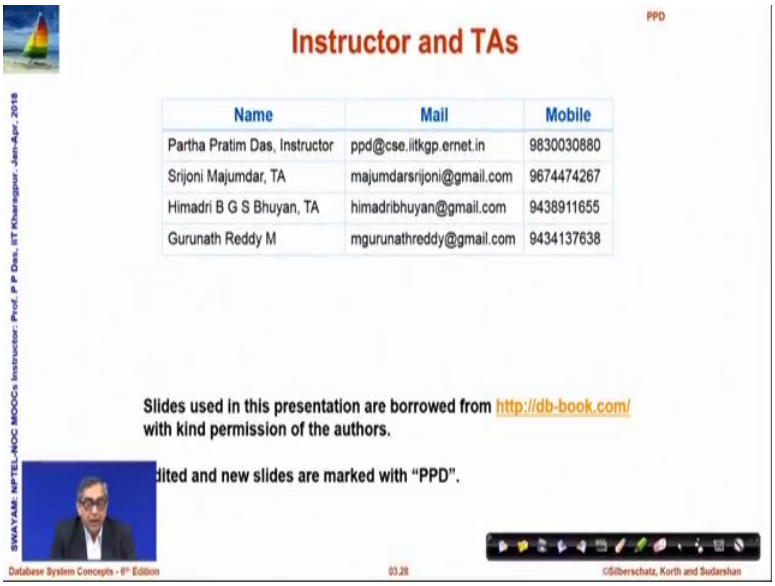


And in 80's, really it proliferated large in terms of prototypes and commercial systems, parallel distributed systems, object based systems started happening in 80's 90's. It really exploded in terms of large decision support, data mining applications, you know applications widespread use of internet based data applications and systems emergence of Google and all that. From 20's, early in early 2,000 it has been XML and automated database administration and now what we are facing in at the big data, which are certainly aspects of other courses, but at the back of back of back at the last layer then often is a strong relational system that exists.

(Refer Slide Time: 31:28)



(Refer Slide Time: 31:41)



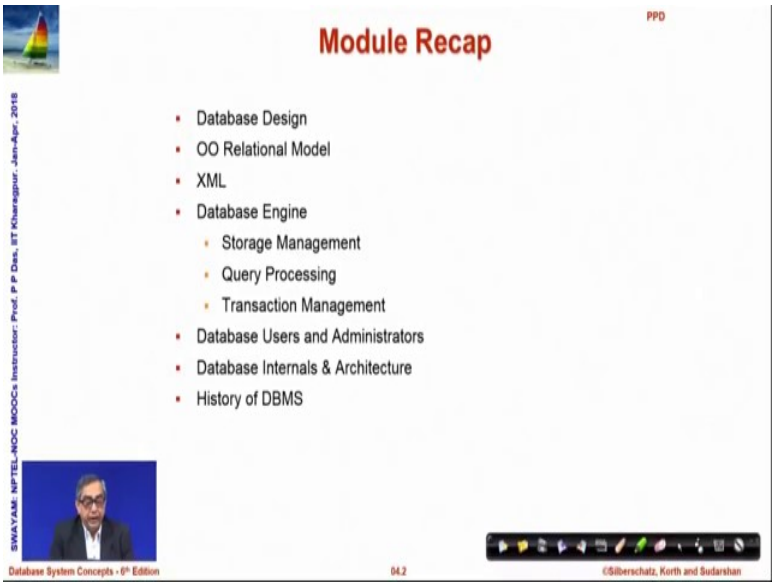
So, to summarize, in this module, we have introduced the models of database management system, the major components of a database engine and discussed about the internals and architecture, and this will conclude our discussion on the internals of database management systems, and next we will move on to exposing more on the relational model.

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

Lecture - 04
Introduction to Relational Model/1

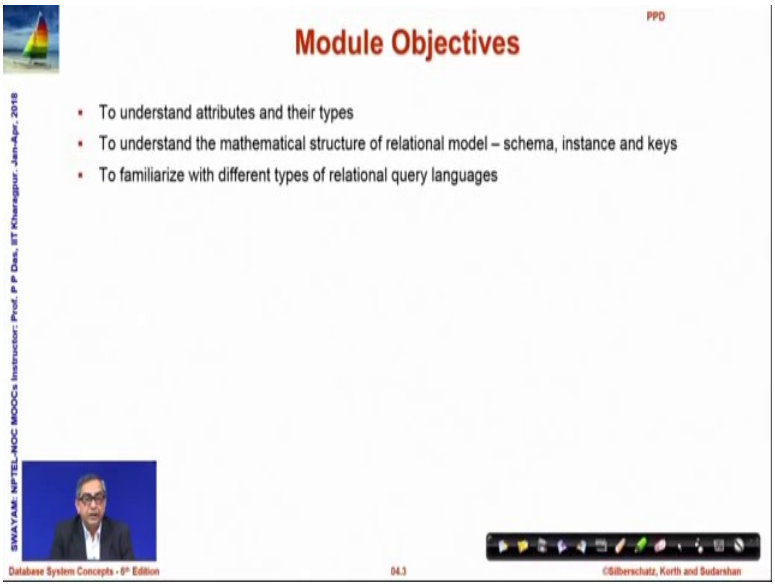
Welcome to module 4 of database management systems, in the last two modules, we have introduced the basic notions of DBMS in this module and the next; we make an introduction to the relational model, which we said is a major data model that we are going to use.

(Refer Slide Time: 00:46)



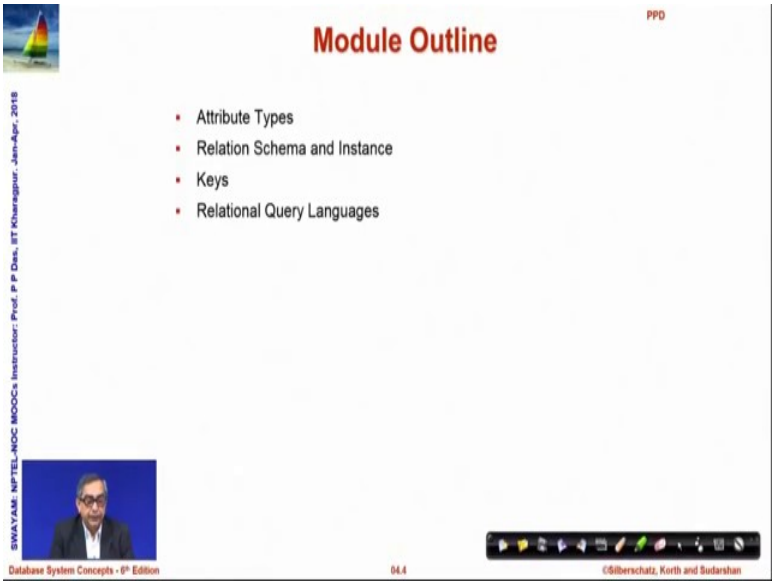
So, this is what we did in the last module.

(Refer Slide Time: 00:52)

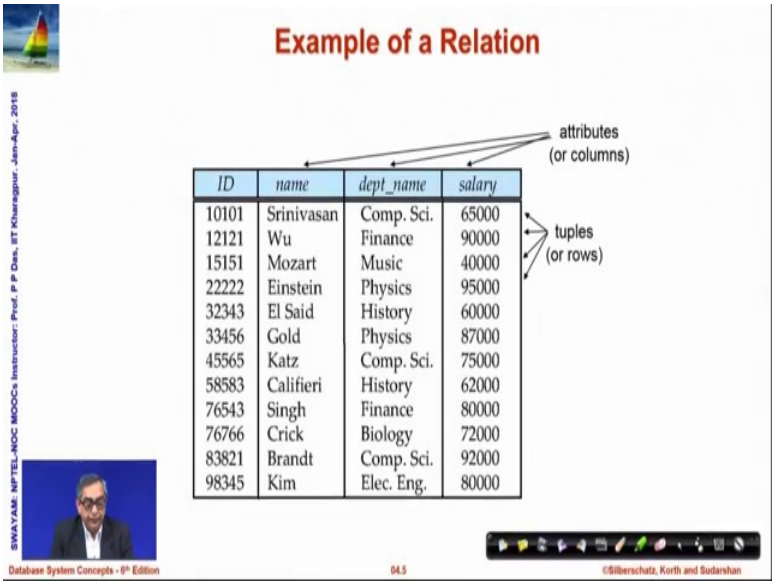


And in the current one, our objective will be to understand key concepts of relational model that is attributes and their types, the basic mathematical structure of instance schema and what is known as keys and to familiarize with different types of relational query languages. This is a module outline that we will follow.

(Refer Slide Time: 01:17)



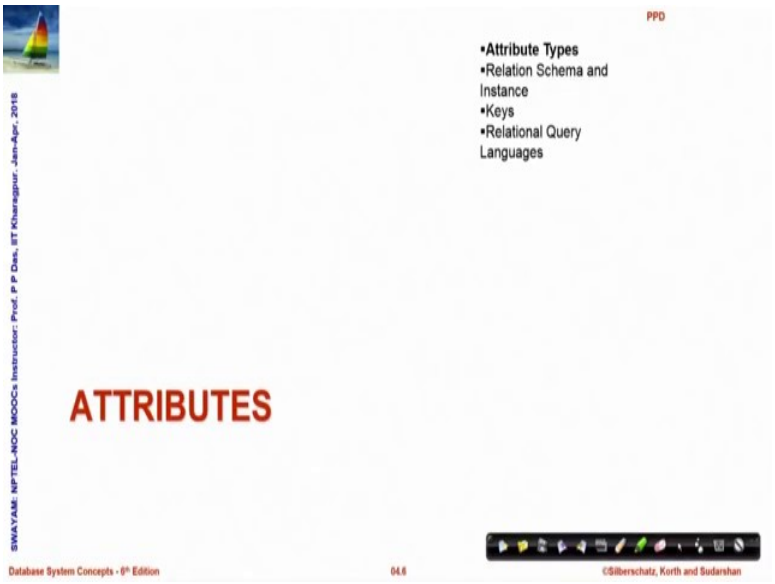
(Refer Slide Time: 01:22)



So, we again repeat this example from past, this is an example of instructors, in a university a table of instructors given by attributes or columns I D name, department name and salary. These are the four columns, the four I Ds and multiple rows, which are specific rows or we often refer to them as tuples. So, you can say that since there are 4 attributes, that is every row has 4 columns.

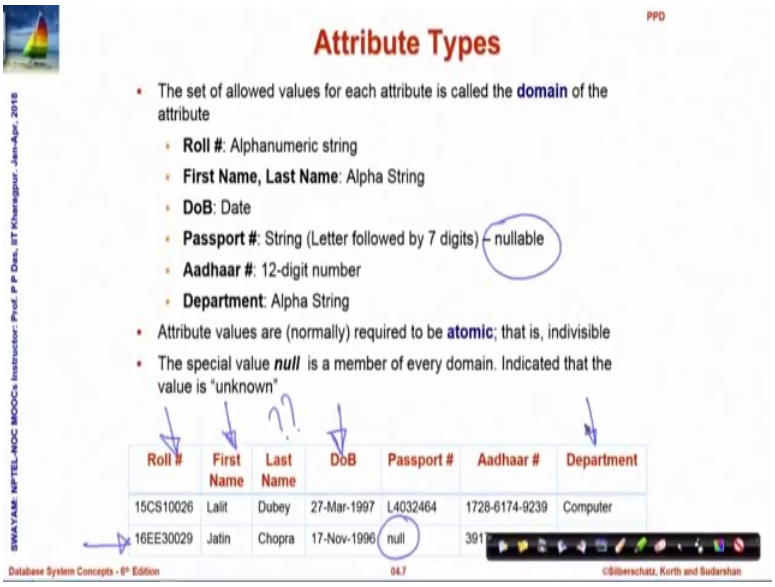
So, this is a 4 tuple that we have and such a table is called a relation is as simple as that. So, this is it, whenever we talk about a relation, we have a number of fields number of attributes number of columns, whatever way, we said of a table and that table according to those columns, it has multiple 0 1 or any number of rows of values, filled in and that is what is a relation.

(Refer Slide Time: 02:34)



Now; so, let us look at attributes more specifically.

(Refer Slide Time: 02:37)



So, attributes each column is an attribute as you said, this every attribute has a domain. The domain is a set of possible values that attribute can take. So, if you just look into the example here, so, I am trying to define a table having different students. So, there is a roll number for a student, there is a first name last name, the date of birth; DOB, the passport number, the Aadhaar card number, the department to which the student belongs and so on. So, let us say these 1 2 3 4 5 6 7 are the different attributes.

Now, if we look into every attribute, then every attribute has a set of possible values of which some value is entered in a particular row. For example, the roll number is an alphanumeric string, as you can see, it has numeric as well as it has letters whereas, the first name or the last name are simple alpha strings. In fact, we can also say that the roll number actually is not only alphanumeric, it has a fixed length, here is it has length of 9. So, you can say alphanumeric strings of length 9 are eligible for being values of this domain.

There could be more restrictions, but that the domain will be certain collection of values which are possible as values of that attribute, when you talk about D o B that certainly has to be a date. So, it is written in the form of d d m m m y y y y that is two digit date, three letter month codes and four digit year, the passport number is a string, a letter followed by seven digits. The other number is a twelve digit number, the department is alpha string and so on. So, the domain is a set corresponding to an attribute, which define that all possible values that attribute can take ok.

Now, these attribute values if you look at they are atomic in nature that is you cannot divide them into smaller parts. So, what I mean is say when we are talking about date of birth the whole date of birth type the date type is one atomic value. For example, if you were to code this in C what you could do you could possibly create a structure with three fields; one is date, one is a month, one is a year and we will say that this composite record composite structure is actually my date.

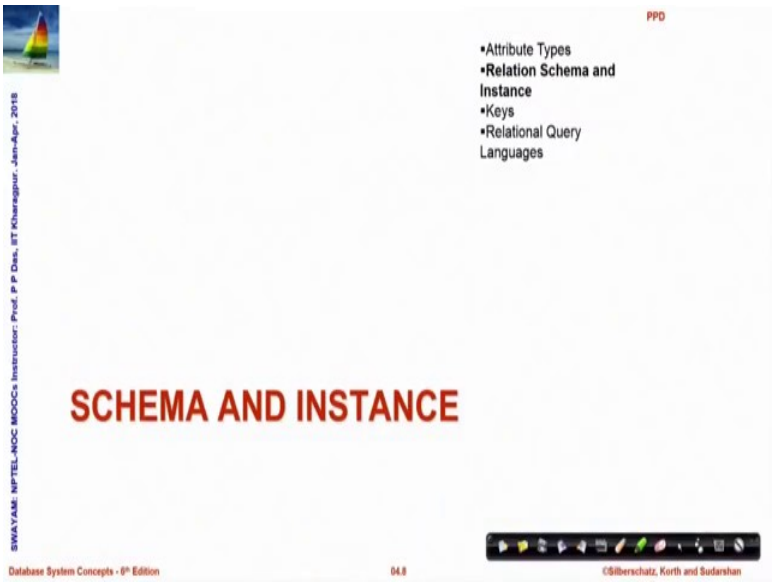
They can do a ls type def, you could do if you are working in C++, you will define a class called date, which has these components and as well as operations with them, but that kind of types are not allowed in a relational database, it has to be an atomic type. So, in a relational database will give you atomic type called date, but all of these are pre specified and has to be taken as one unit, other atomic types are integer like we do not have an integer field here. There are strings; there are numerical values, which are kind of floating point values and so on.

Now, some attribute may have a special value called the null value, which is the member. It is domain; actually every attribute of any domain can have this special value. The null value is not actually a value; it is actually an absence of a value. So, it says that this value is not known. So, if you look into the example above, then you will see that for

passport we have said that the passport is a string letter followed by seven digits and it is null able, which means that in the passport field, I may have a value, may have this null value which means that it is not that, the passport is null, what it is saying is this passport number for this particular student, the row number 2 is not known, is unknown.

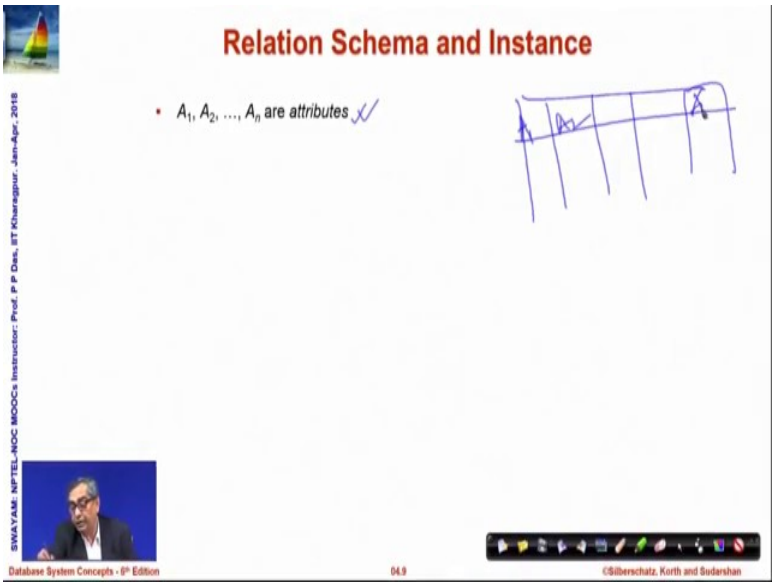
Now, all fields may or may not be null able. For example, will not allow D O B to be null able, date of birth has to be there, will not allow roll number to be null able, will not allow first name to be null able, but we may allow last name to be null able. It is been a style, let not to use your last name, many people just use one name. So, you could allow that, it is not known, it is not there whereas, department may not be null able, it must be there. So, null is a very critical concept and what it actually does? It actually creates a lot of issues and complications in terms of defining many operations. So, understanding null as a value in terms of an attribute is a critical requirement for the design.

(Refer Slide Time: 08:03)



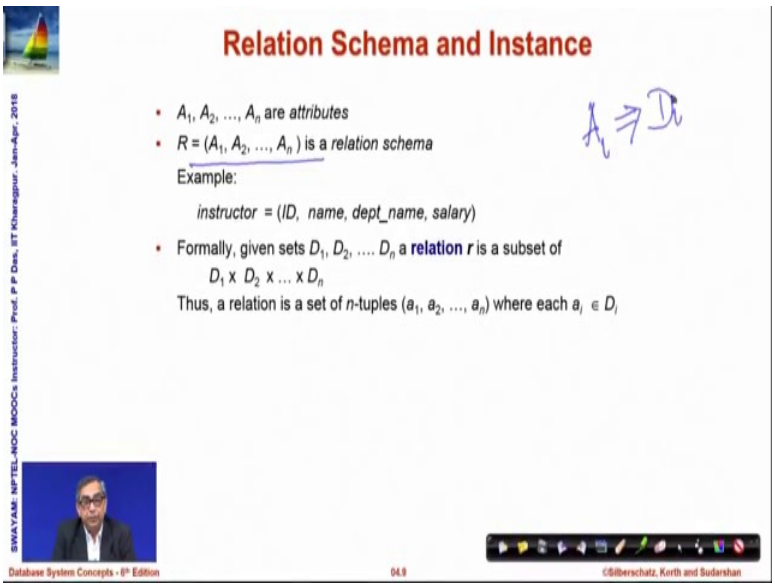
Now, coming to the schema and instance, we have discussed about the basic understanding of schema and instance. So, understanding them formally now, we say that if we have a schema. So, it is like a table having multiple columns say, there are n columns, having names A 1 to A n, then this A 1 to A n are the attributes.

(Refer Slide Time: 08:26)



So, these are the different attributes. So, if I have this, then it basically means that I have a table, where these are the columns A 1 A 2 A n like this.

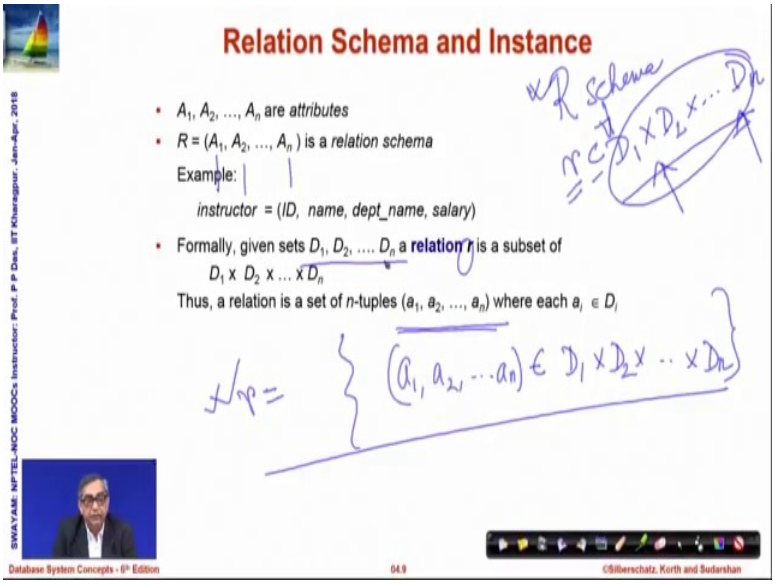
(Refer Slide Time: 08:50)



So, then a relational schema is a collection of these attributes. So, it is a collection of all these attributes. So, we said R is a relational schema, which has attributes A 1 to A n. Now, every attribute A i has a domain D i. So, for every attribute, I have a set of values that are possible. So, if you, if you recall then here we had different, these are the different attributes and these are their different domain. So, D o B is an attribute and the

domain is date. So, any possible date, other is an attribute and this is the domain, which is A. So, all attributes, each attribute will need to have certain domain and those are marked by the D Sets. So, we will say that a particular relation a particular relation R.

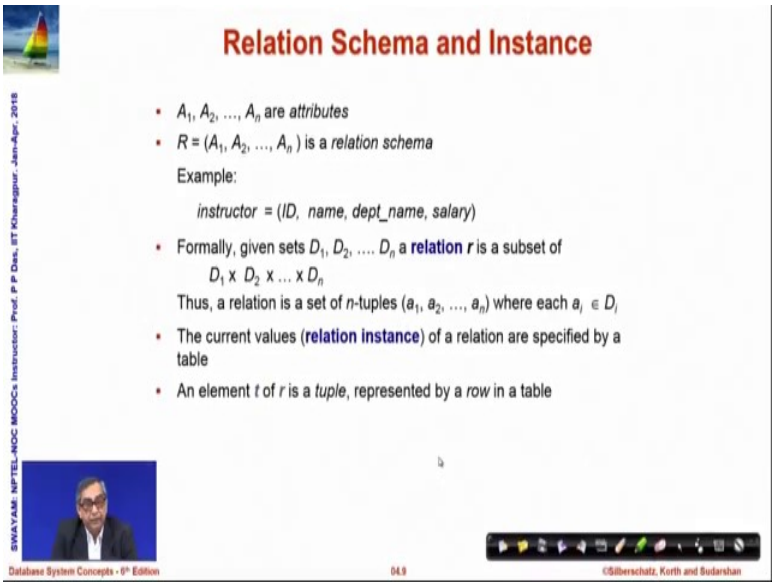
(Refer Slide Time: 10:01)



So, R is a schema. So, a particular relation R is a subset of **D 1 + D 2 × ….. D l**. So, recall the mathematical notion of relation which says that a relation is basically a subset of a set. So, these are the possible values. So, the first attribute can take values from D 1, second attribute can take values from D 2 and so on and the nth attribute can take values from D n. So, any specific row, any specific record is a set of values for A 1 A 2 A n and therefore, is a member of this Cartesian product and the relation is a subset of that. So, this is a D value is an n tuple, which is a subset of this (a1, a2…. an).

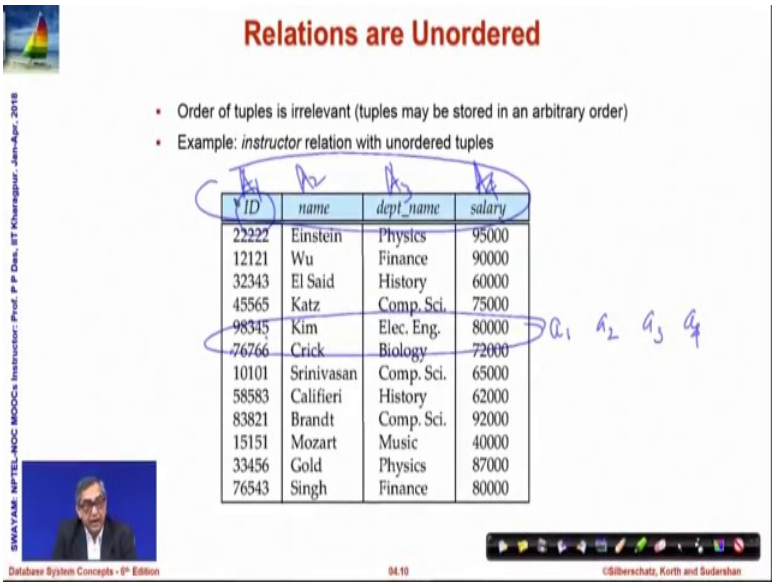
This particular record is an element of this Cartesian product set and R necessarily is a set of such tuples that is a mathematical view of the schema and the instance. So, this is the schema and this is the instance corresponding to that schema based on the different domains of the different attributes and this is the notion that we will continue using. So, please try to follow this carefully.

(Refer Slide Time: 11:46)



Now, whenever we have an instance, we mark that as a table and every such table.

(Refer Slide Time: 11:56)



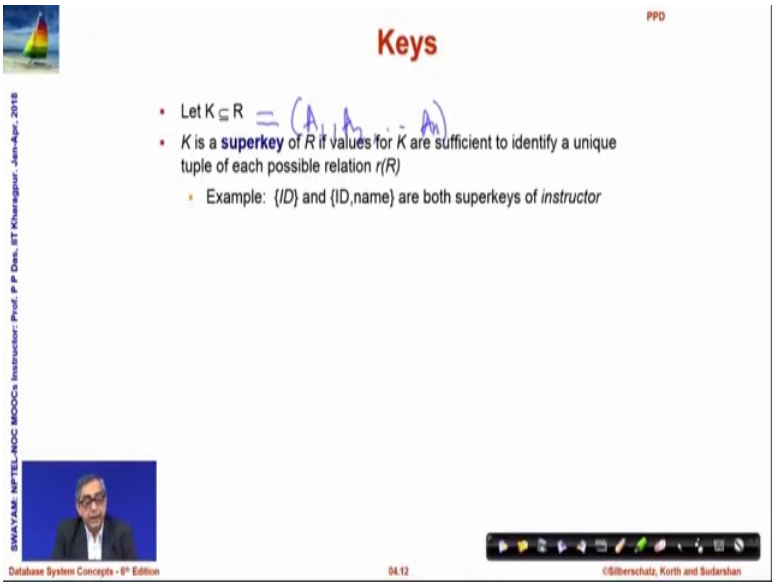
So, here you have now understood it very well. So, these are my attributes. So, this is A 1, this is A 2, this is A 3, this is A 4 and any one i name is at the different values a 2 a 3 a 1 a 2 a 3 a 4 98345 is a 1 Kim is a 2 and so on. Now, naturally this, it is not visible from the instance, because we are taking an instance view, we are not being able to see, what that domain is that will be visible? If we look at the corresponding D D l, the definition language description of the schema, which must have specified I D as a numeric value,

the name as a string value the department name as another string value whereas, salary as a numeric value and so on. Now, what is important to note here is a relation necessarily is a set as we said is a set, which is the as the relation R is a set, this is a set, which is a subset of this set.

(Refer Slide Time: 13:04)

So, we know the elements in a set are do not have any ordering, they are unordered. So, a relation is necessarily unordered. So, it does not really matter that in terms of this collection of rows, which row is at what position, if I reorder them, the relation does not change it is just that they are a collection of this set of rows. So, that lack of ordering is critical information that we will have to remember in mind next concept is key.

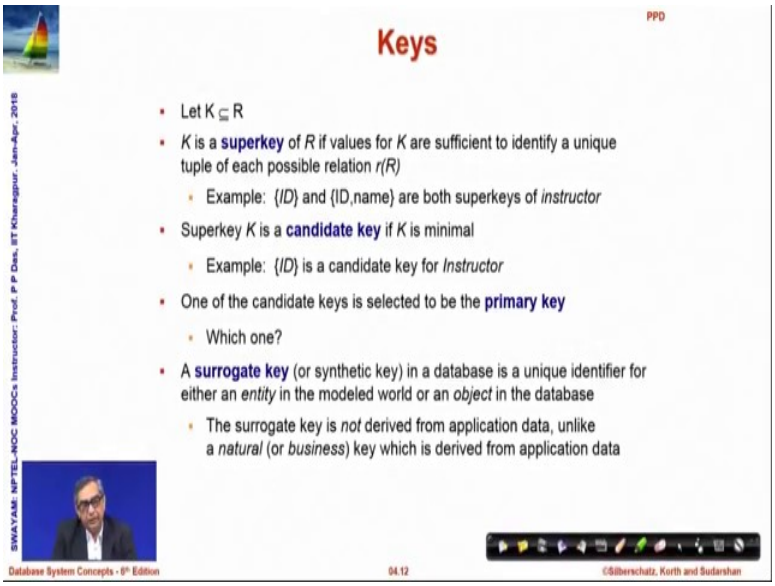
(Refer Slide Time: 13:49)



So, R as we have seen is a relational schema, which is a collection of attributes **A 1, A 2 …..A n**. Now, K let K is a subset of R. So, it is one or more attributes, it has to be a non-empty subset. Now, we will say that K is a super key of R, if we consider the values of different tuples in the attributes of K and we find that there cannot be two tuples, which are different, but match on these attributes, which mean that the values of the attributes of K, uniquely identify each row of the relation, then we will say that K is a super key of R.

So, the instructor table that we have seen, I D is a super key similarly. So, K can be taken as a singleton set of attribute I D or K can be thought of as the set comprising I D and name both of them are super keys of instructor.

(Refer Slide Time: 15:32)

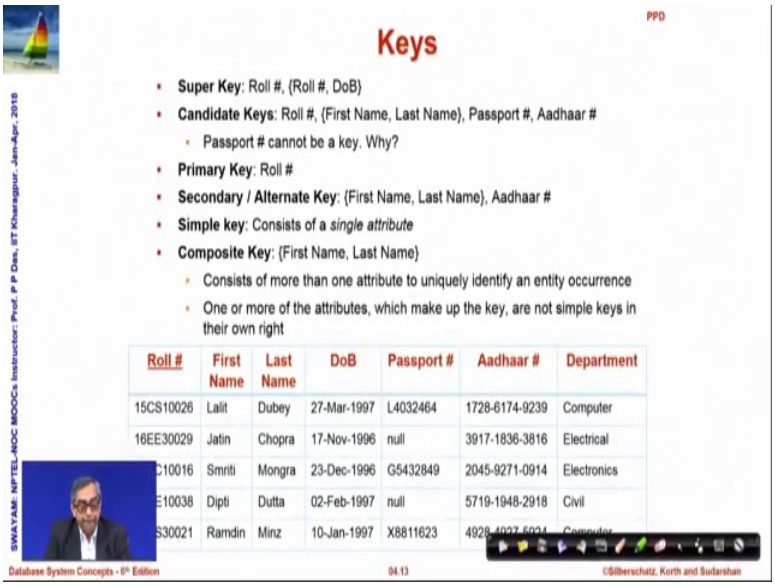


Now, we say a super key K is a candidate key, if K is minimal. So, the idea is like this that this is a key, super key, this is also a super key, but certainly this is a subset of this. This is smaller than this. So, we will say this is a candidate key, but this is not a candidate key, because it does not satisfy the minimality condition.

There could be multiple candidate key in a relation, if there are multiple candidates key then we select one to be the primary key. Now; obviously, there is a question of which one we select, but anyone can be selected as a primary key, which is the key of the relation and we will see that in some cases, there is concept of surrogate keys.

So, if I have a relation where there is no attribute, whose value can uniquely identify each and every row of the table then I might synthetically generate a value for example, like a serial number, I can generate a serial number and say that this is my value. So, that serial number or that computer generated field value has no business implication, the real world did not have this value, it is not like a Aadhaar card number or like a passport number, but it is a value which is purely generated to identify every row uniquely. So, such keys are known as surrogate keys or synthetic keys.

(Refer Slide Time: 17:40)



Now, let us look at some examples, this is again the same student database, I just shown a while ago the same set of columns, but I have added few more rows. Now, if we look at what could be a super key there are several candidates, but I have just written a few roll number is certainly a key, because I am assuming that the university assigns roll numbers to uniquely identify every student.

So, there cannot be two rows in this table, which match in the value of the roll number and does not match in the values of the other fields. So, roll number can uniquely identify, if it can then any super set of attributes, which continual number will also be a super key. So, roll number and date of birth together is a super key that can also unique to identify every row trivial. What are the candidate keys? Now, there are of course, that could be several other super keys that has to be kept in mind, the candidate keys are roll number is a candidate key, the first name last name together, we can say is a candidate key.

So, we are saying that not only the first name, but if we take this pair, you remember the key, the set of attributes forming a super key is a set. It is not an individual field. So, I say the first name last name together, from say, key well. This does make some assumption, because if I say the first name last name together from say key; that means, that there cannot be two records in this student table, where the first name and last name match, but the records are different. So, which mean that no, two students having the

same first name and last name can be enrolled in the university. This is a restrictive assumption right, but I am just making that assumption to illustrate the different possibilities; then what is the other possibility passport number? Everybody has a unique passport number. So, passport number could also be a key, could be a candidate key. Aadhaar number; everybody has a unique Aaadhaar number. So, that can be a key and so on.

So, these are called the candidate keys. Now of course, we can observe that given the data it is clear and it was also mentioned when the schema was designed this passport number cannot be a key. Why can it not be a key? Can two students have same passport number? Of course, not every student has a unique passport number, but it is possible that some student does not have a passport. So, if some student does not have a passport then the passport number field of that student is a null, the passport number is a null able field, if the passport number is null then it is possible that multiple students may not have passports.

So, as we can see here that this student Jatin Chopra does not have a passport. So, similarly, Dipti Dutta does not have a passport either. So, certainly if this were to be the key then for all records, for which passport number is nil, this value would not be able to distinguish them in terms of the rows of the table. So, we have to say that passport number cannot be a key or in other words, we can say that no key can be a null able field.

No key attribute or a participant to a key attribute could be a null able field right. So, this is one observation here. And, so that clearly also implies that, if we say that Adhaar number is a valid candidate key that will mean that for admission to that university having Adhaar number, would be mandatory, if somebody does not have a Adhaar number that will have to be null, which is not allowed.

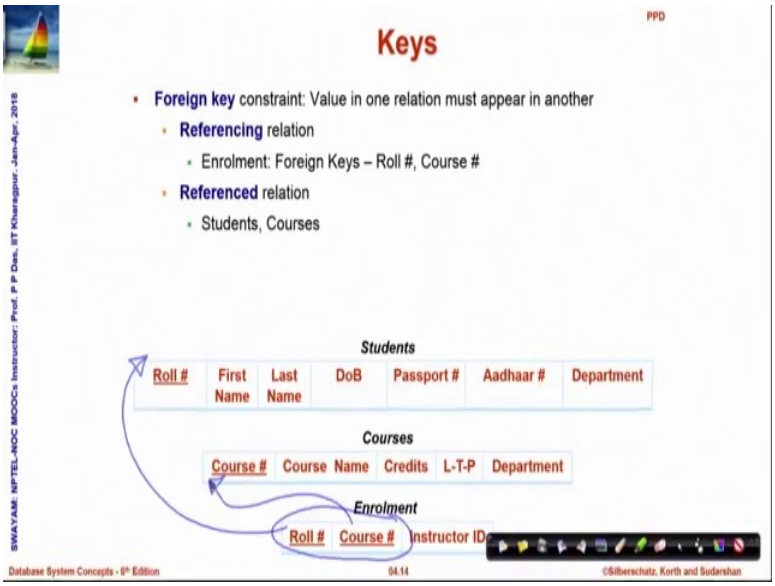
So, let us move on. So, one of these candidate keys have to be made the primary keys. Let us say; we make roll number, the primary key and since, we make roll number the primary key in the schema. We underlined the roll number attribute; this would be a common way to show that roll number is a primary key. So, the others that are not taken as a primary key are called the secondary or alternate key. So, first name last name pair

could be an alternate key Aaadhaar number could be an alternate key and so on. A key is said to be simple, if it consists of a single attribute.

So, roll number is a simple key, Aadhaar numbered is a simple key, if it were taken to be primary, but first name last name pair, if we take that to be a primary that will not be considered assemble simple key, because it has more than one attribute naturally the other, if you have a simple key.

They have other side is a composite key is one, which has more than one field such that none of those fields individually can act as a key, but together they can act as a key. So, first name itself cannot be a key last name itself cannot be a key, but together. They can be a key of course, under the assumption that no two students with the same first name last name are given admission. So, these are the different types of keys that can happen.

(Refer Slide Time: 23:32)



Let us have some more views with the keys, we extend the schema and besides a student I introduce two more schema; one is called the courses, which is given by course number, course name credits L T P. L T P is number of hours of lectures tutorials and practical's and the department. So, these are the different fields and from the convention already stated you can figure out that course number is the key primary key of this relation. I use another schema, which is enrolment, which describes which student is attending which course. So, it has a roll number and the course number.

So, roll number of the student attending the particular course number and it also has an instructor I D as to who is teaching that course given this. You can see that in the enrolment relationship, I have this pair roll number and course number, which will certainly be the key for enrolment, because if I have two rows in enrolment. How they will be distinguished, they cannot be distinguished by roll number, because a particular student may take multiple courses.

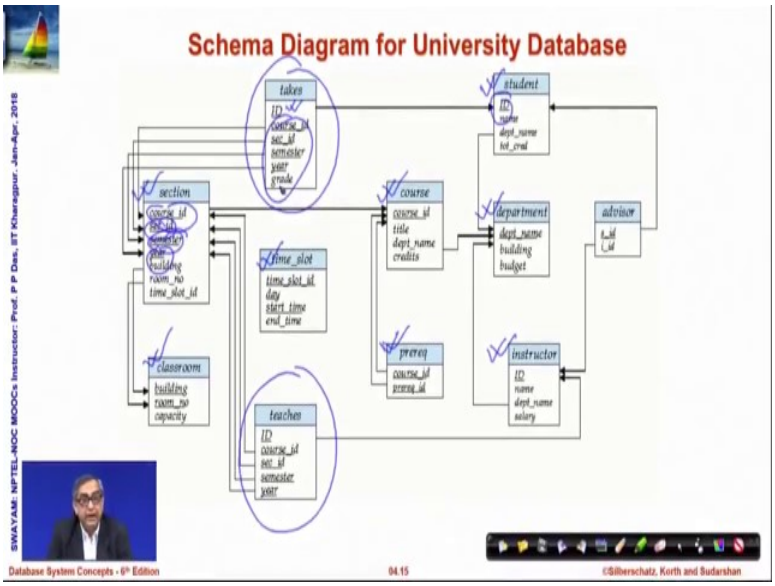
So, there will be multiple records having the same roll number, but different course number, the course number by itself cannot be the key, because every course will have multiple students. So, there will be multiple rows having the same course number, but all different roll numbers, but if we take this together, roll number and course number together then that forms a key.

Now, such a key such a key having roll number the roll number itself is a key of another relation the course number itself is a key of another relation. So, when we take the keys of other relations to form the key of a relation then we say that these are foreign keys. So, roll number and course number are foreign keys in student and course and since from enrolment the student and courses are being referenced are being referred.

So, we say enrolment is a referencing relation and students and courses other reference relation and we will often like to also mention as to what is a foreign key of a relational schema, because that will help us understand how the different schemas are interrelated and we will see that, this will come out directly from the notion of entities and relationships of a year model of a year diagram, a key is called, to be said to be compound, if it consists of more than one attribute to uniquely identify an entity occurrence.

So, each attribute which makes up the key is a simple key, in it's own right, mind you there is a subtle, it sounds very similar. We talked about composite key; earlier, we talked up; we are talking about compound key here, the subtlety of the difference is in a composite key, every component attribute is not a simple key by itself, but and the components come from the same table in a compound key. The components are simple key in their own right, in some other table and are put together as a compound key in the given table. So, the roll number, course number in the enrolment table is a compound key.

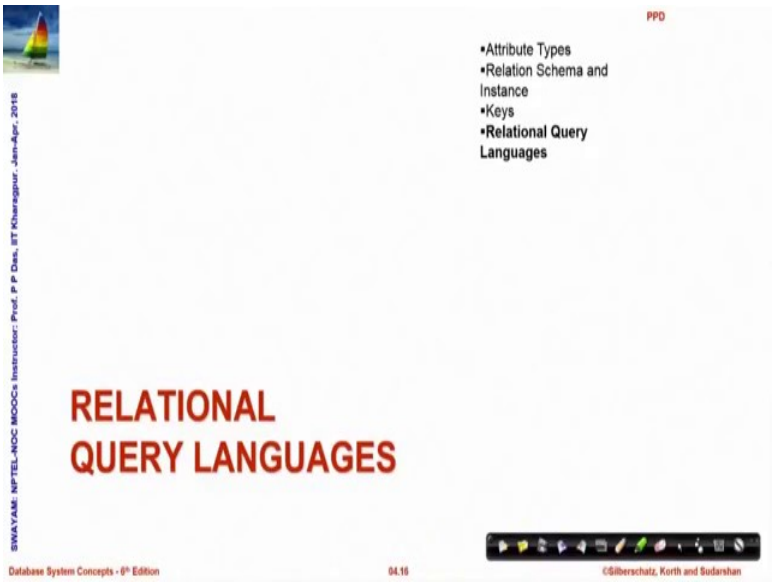
(Refer Slide Time: 27:31)



So, with this I would request you to spend some time with this relatively elaborated schema compared to what we have done already of the university database. So, every rectangular box shows a relational schema on top of each in blue, is written the name of that relation relational schema. So, it has a relational schema like courses, the students the instructors, the departments, the prerequisites, the time slots, the classrooms and so on.

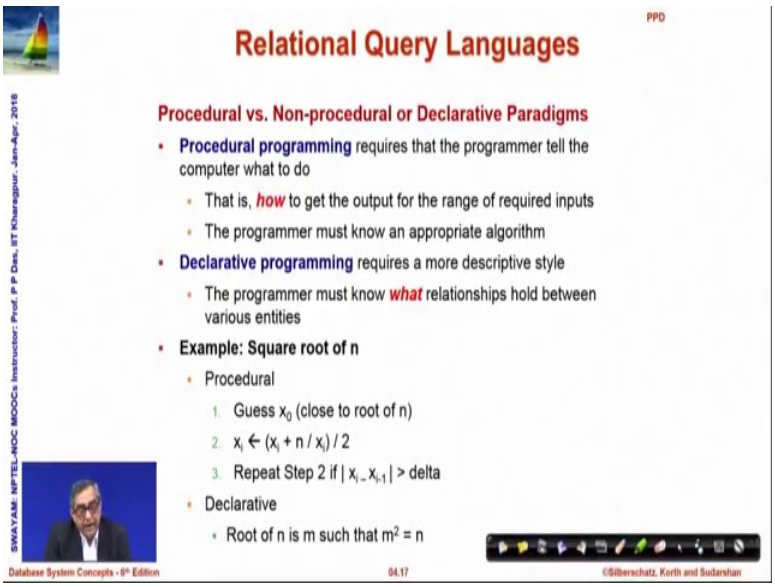
The sections and the relationships between them for example, the relationship is takes is a relationship, which relates students with different sections, with courses, teachers is another relationship, which relates to instructors with sections. So, it is showing you directly as to how the keys of this, what are the attributes? What are the key attributes primary key attributes and also what are the foreign keys that we have in this for example, intakes this is a foreign key, which is featured here, course I D section, I D semester here are the foreign key part of the takes that exists here. So, please study this schema. We will keep on regularly referring to this schema in future as well. So, this is what we have here.

(Refer Slide Time: 29:29)



Now, we move on to the relational query language, we briefly talk about the relational query language.

(Refer Slide Time: 29:33)



Now, we will have to in this the key thing that we need to understand is the relational query language is somewhat, different from the programming languages that you have studied so far which are procedural in nature, in contrast the relational query language is non-procedural or declarative in nature, a procedural programming language requires that the programmer tell the computer how to get the output given, the input a pro

program is about finding output, for a given input and you write a procedure, the sequence of steps that need to be done. So, that given the input, you can compute the output.

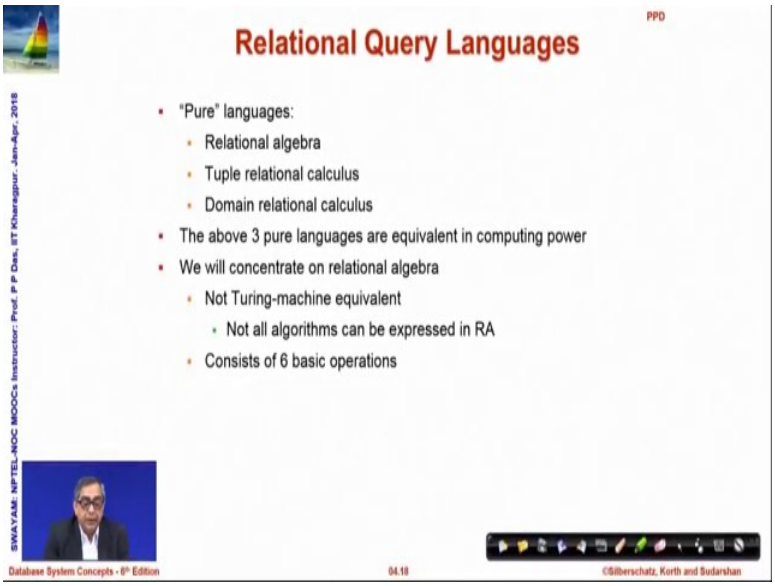
So, you say how that computation has to happen and the programmer must know that algorithm in contrast, in declarative programming, you say what you want? You will do not say how that needs to be computed? How that will be computed? You may not even know that, you may not even know a single algorithm to compute the output, but you specify what output you need. So, this distinction between how and what of programming differentiates procedural and declarative programming.

So, all that you have studied so far in terms of C C++, java python and all that are procedural programming, where you necessarily have to specify, how you will have necessarily; have to specify what the algorithm is, but in declarative you just say what you need. So, just a simple you know pathological example to understand this difference. Suppose, we were interested in computing the square root of a number n assuming n is a positive number, the procedural step would be something like; this is an algorithm that you guess a X o, which is a square root, which is close to the root of n.

I mean some guess you make and then you repeatedly refine this estimate by taking the arithmetic mean of the estimate and the quotient of the division of n by this estimate. So, you take an arithmetic mean and find the new estimate and repeat the steps, I mean as long as the difference between the two conservative estimates is more than a certain value delta, this is a procedural algorithm, you are giving an algorithm. So, given and following this algorithm, we will find the square root declaratively, you can just say that what is the result? I want to result m such that $\mathbf{m^2 = n}$.

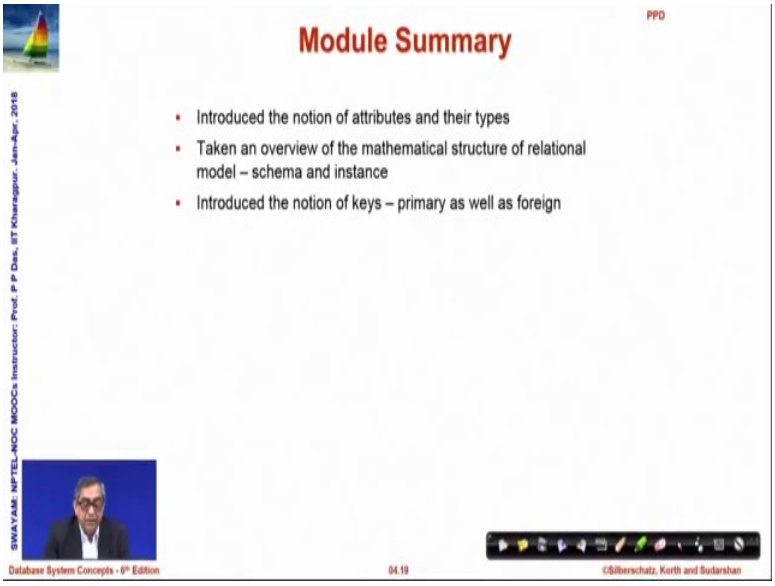
So, you are again asking for the same feel, you are expecting the same output, but the way you are saying is not an algorithm, you are rather specifying a predicate, which must be true in your output. You are saying that the predicate is m square must be n. So, whatever m is that square of it must equal n. So, this style is known as declarative whereas, the earlier style is known as procedure.

(Refer Slide Time: 32:38)



All query languages, relational query languages are declarative in nature. We have talked about the pure languages, are they are all equivalent? We mentioned that earlier also and also again to remember that none of them are actually Turing equivalent; that means, that not all algorithms can be expressed in them or specifically relational algebra, which we will look at in more depth and the relational algebra will consist of six basic operations, which we will discuss in the next module.

(Refer Slide Time: 33:11)



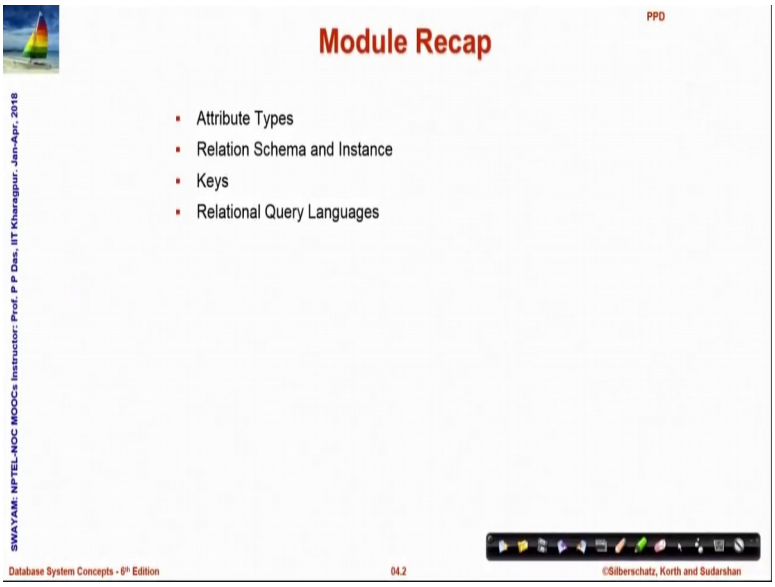
So, to sum up we have introduced the notion of attributes and their types, we have taken an overview of the mathematical structure of the relational model schema and instance, we would say mathematically they are relations, mathematically they made a mapping and we have introduced the very important concept of keys and in that very specifically, what is a primary key as well as what is a foreign key? In the next module, we will discuss about the different operations of relational model relational logic.

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

Lecture -04
Introduction to Relational Model/2

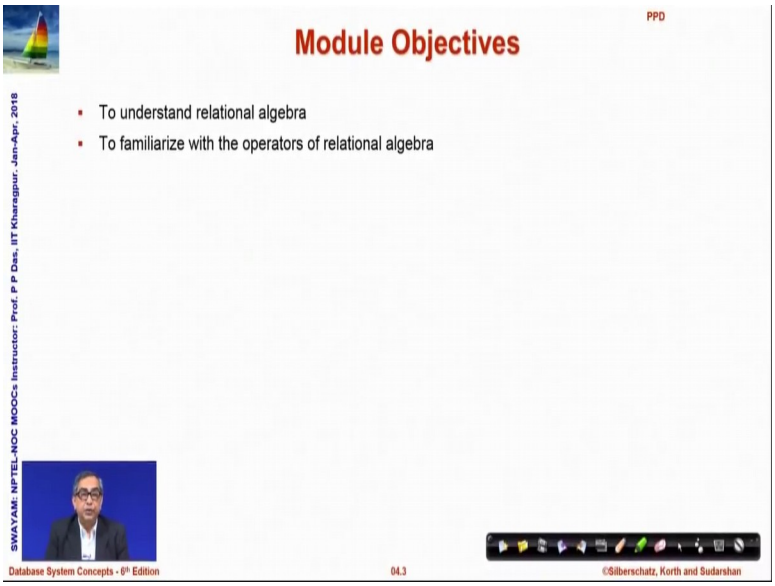
Welcome to module 5 of database management systems. In the previous module, we started discussions on introducing relational model we will conclude that in this module.

(Refer Slide Time: 00:36)



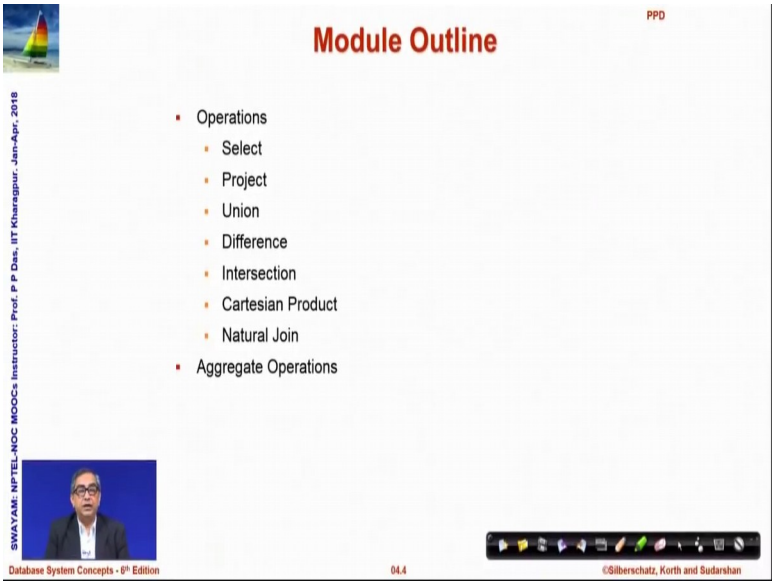
So, in the last module we have talked about attributes relational schemas and instances in mathematical form and very importantly we have tried to introduce discuss about the concept of keys.

(Refer Slide Time: 00:53)



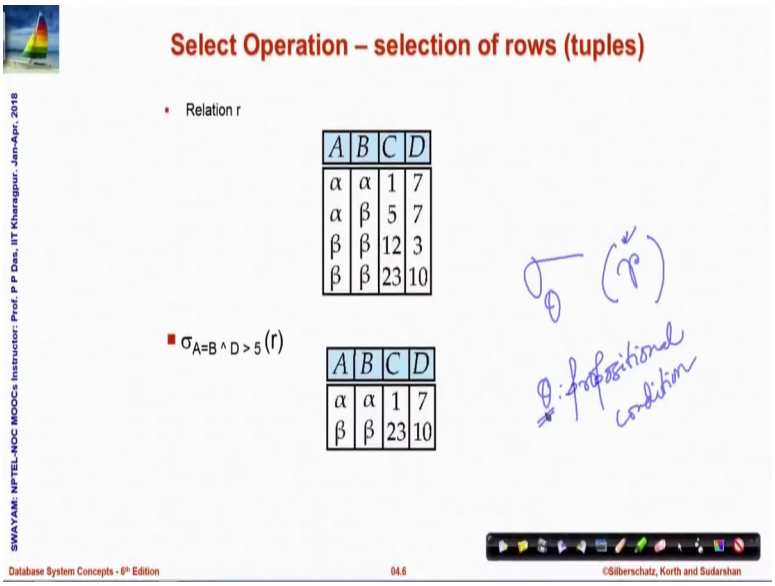
In this module we, will try to understand more on the relational algebra and familiarize with the operations of relational algebra.

(Refer Slide Time: 01:00)



So, these are the different operations that we will look at select project union and so on. Some of them are simple set theoretic operations some are newly defined operations and we look at the aggregators.

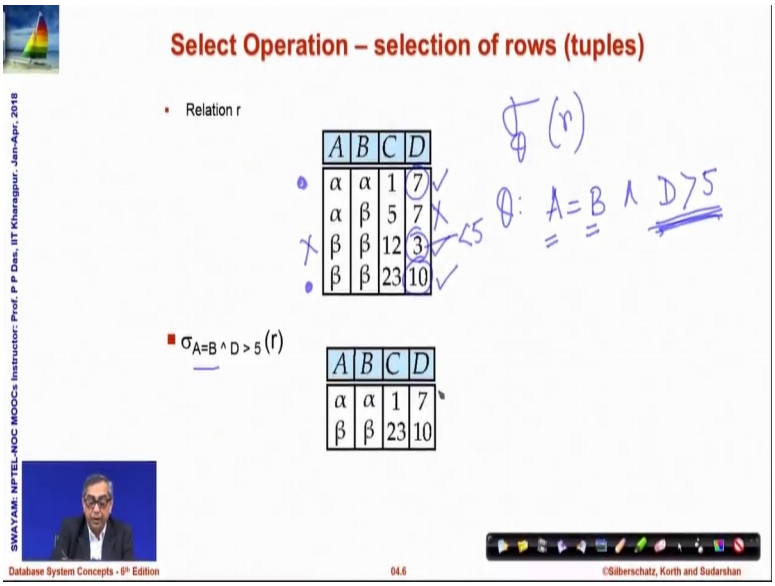
(Refer Slide Time: 01:18)



So, relational operators, so what is a relation as we have seen already? A relation is nothing but a table. It has a set of columns and it has a set of rows or records that fill up data according to those columns. Select is an operation, which chooses a subset of rows from a relation based on a certain condition. So, it is written in terms of in relational algebra we write it with a notion of notation of **σ** and following a parenthesis we put the name of the relation.

So, we say we are selecting from the relation r and then we put a condition here, which is a propositional condition. So, if for this all rows of r will be checked if a row will satisfy this condition. Then it will be included in the result, if it does not satisfy the condition, then it will not be included in the result.

(Refer Slide Time: 02:53).

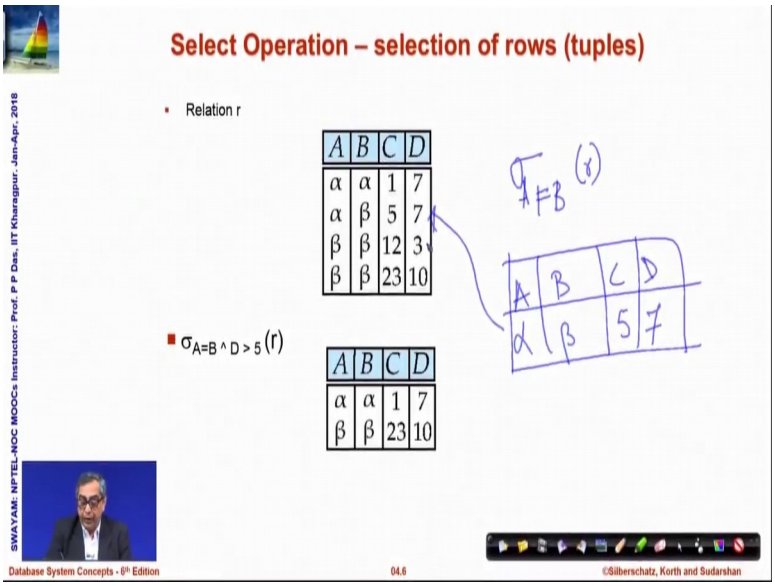


So, let us look at this example. So, our condition is sorry let this put this back. So, our condition **Θ** is **A=B ^ D>5**. So, you are saying that that any row to be selected, the value of it is A attribute should equal the value of it is B attribute and when that happens the value of it is **D >5**. So, you can easily if we look through this we can easily say by the first condition **A=B** you can say that this row does not satisfy this condition.

Because, A is **α** and B is **β** whereas, these 3 rows satisfy because **A= β**. Then you again look at D we find that this **D<5**. We say this also does not satisfy, because it fails the second condition both have to hold. So, we finally, come to that this record and this record r the selected record in the result **α α** 1 7, **β β** 23 10. In both of these records **A=B** in both of these records **D>5**.

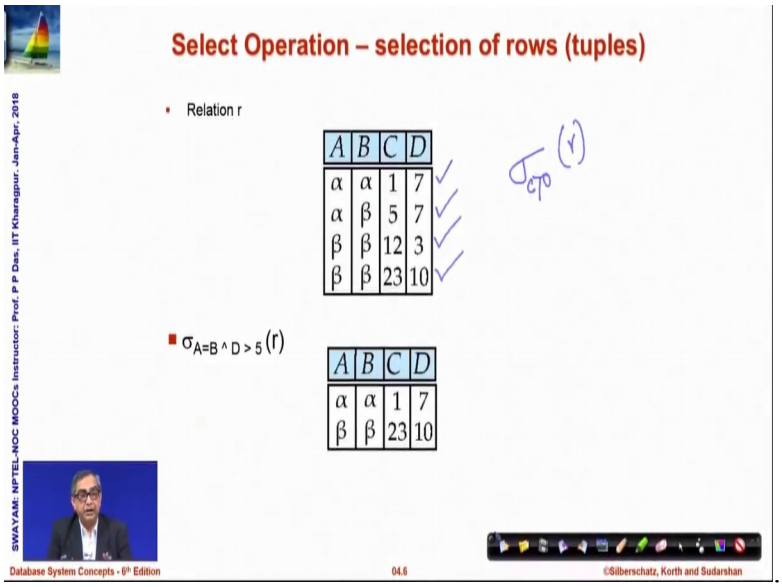
So, selection is a process is the operation which selects a subset of rows from a table, from a relation and creates a new relation. Based on a selection condition that must be true for all the rows for all the records that have been selected, but the set of columns do not change they remain the same.

(Refer Slide Time: 04:57)



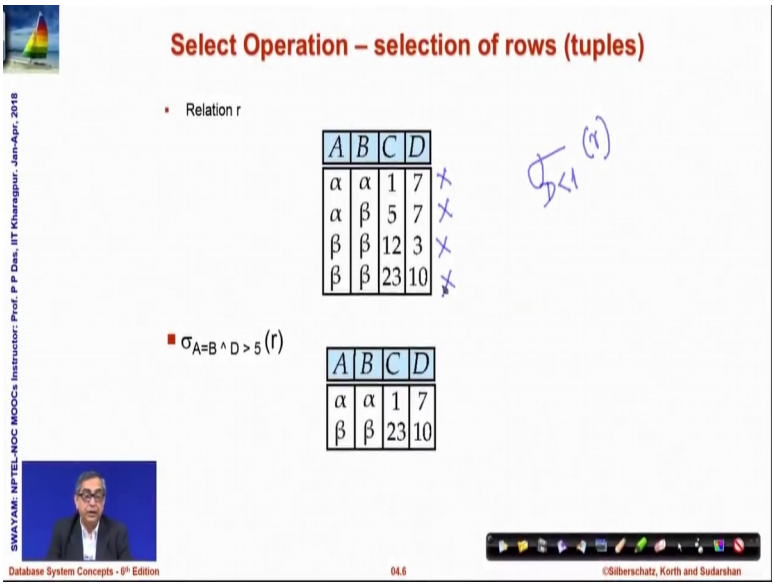
So, for example, if I if in the contrary if we say, that $\sigma_{A \neq B}$ **(r)** then naturally, I will have a relation with fields A B C D and that relation will be only this row. Where that is the only row where **A≠B** we can.

(Refer Slide Time: 05:26)



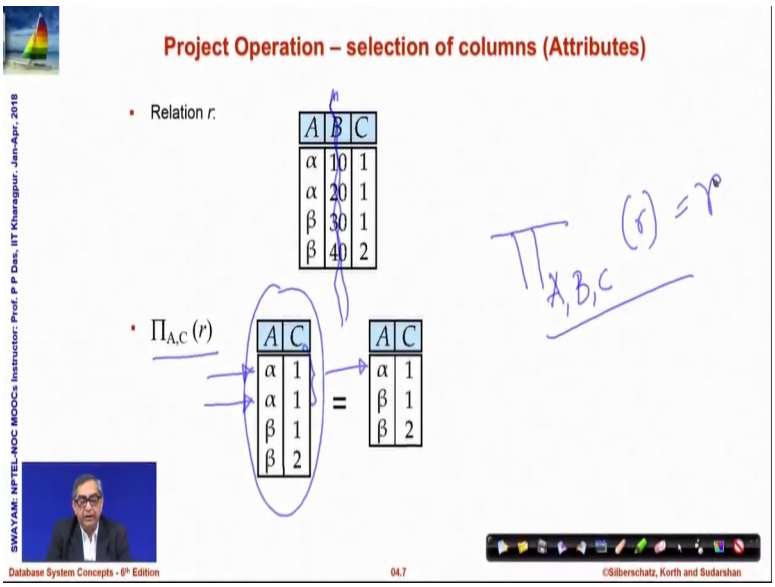
I can have a selection saying r where **C > 0**. Naturally, this will satisfy, this will satisfy, this will satisfy, this will satisfy. So, this whole relation would be the result of the selection. So, it is possible that it is not necessary that some rows will have to get eliminated in the result.

(Refer Slide Time: 05:54)



I say that this is the selection is **D <1** this will fail, this will fail, this will fail this will fail. So, all of them will fail so, it is possible that the result of an operation could be either the whole relation as we saw last time or a null relation which has where none of the record with feature because, none of the record satisfy the condition. So, that is a basic select operation.

(Refer Slide Time: 06:38)



Let us move on look at the next one is called a projection operation. So, select chooses a subset of the rows. Projection necessarily chooses projects a set of columns from the

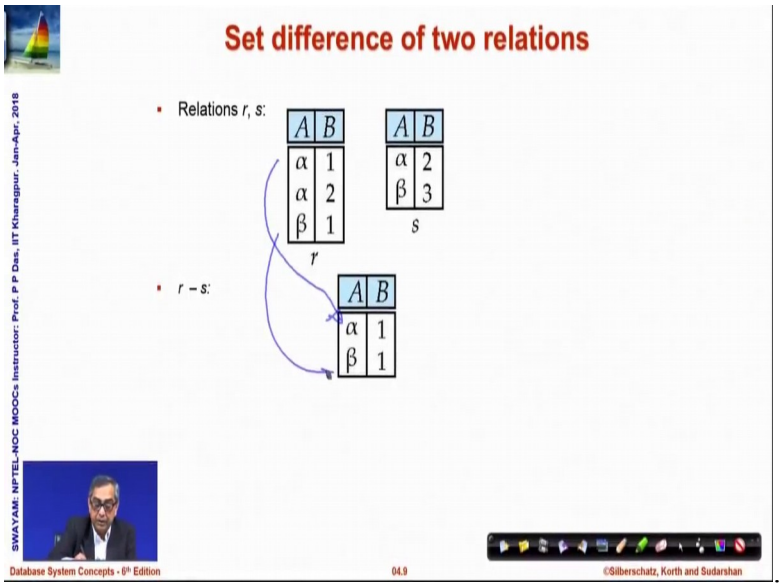
original relation. So, this is a quite straightforward to see, it is written in terms of this notation **Π** and then you write the columns that you want in the result of projection. So, we say this is $\Pi_{A,C}$. So, which means basically the column, which is not selected in the projection, you can simply forget about that simply erase it. If you erase that you get this relation and once you get that, please recall that a relation is a set and in a set every element has to be distinct. So, after erasing B this first and the second row have become identical.

So, naturally both of them cannot be there, it will have to be made distinct by erasing any one of them. And hence, they become one row in the result; obviously, I can project on any of the infield single or all the fields also I can do a $\Pi_{A, B, C}(r)$of course, that means, in this case that will mean that it is a set which is equal to r anyway, but; obviously, I must have at least 1 column at least one attribute to project on. I cannot project on a null set of attributes because that does not give me a schema.

So, there will have to be some attribute one or more attribute on which I project. So, selection and projection, selection has given me the set of rows to retain and projection has given me what are the columns to retain in the result. And combining them I can do several different operations in a database table which can give me several interesting results. Before proceeding further,

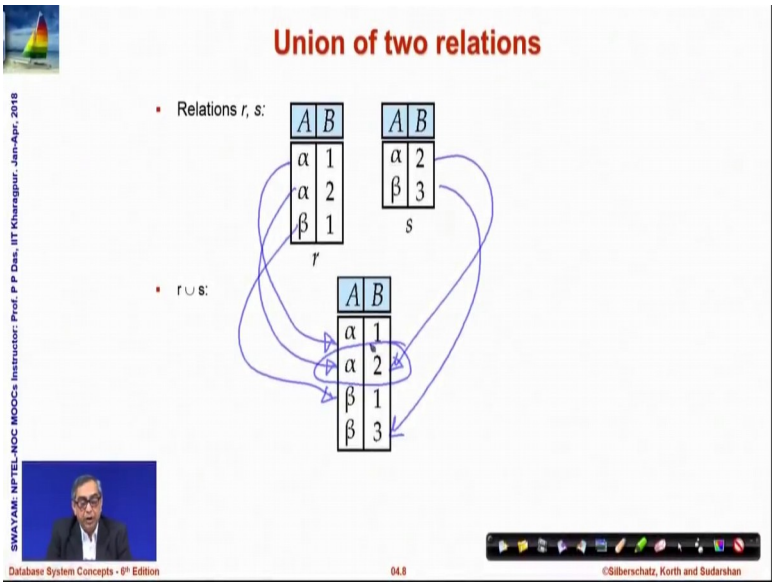
Let us look into some of the typical other operations that relational algebra allows. The next one is $\cup$ given two relations I can take a $\cup$ this is nothing but a set theoretic union. The two relations r and s must have the same set of columns A and B because, if the columns are not same, then the $\cup$ does not make sense. They because certainly if the columns are different attributes are different their types and type of data values would be different. So, they cannot be put to a same table so when two relations have the same set of attributes then their instances can be taken a union of.

(Refer Slide Time: 09:49)



So, all records that exist in both these relations will be put together into a single table. So, here α1 is coming here β 1 is coming here.

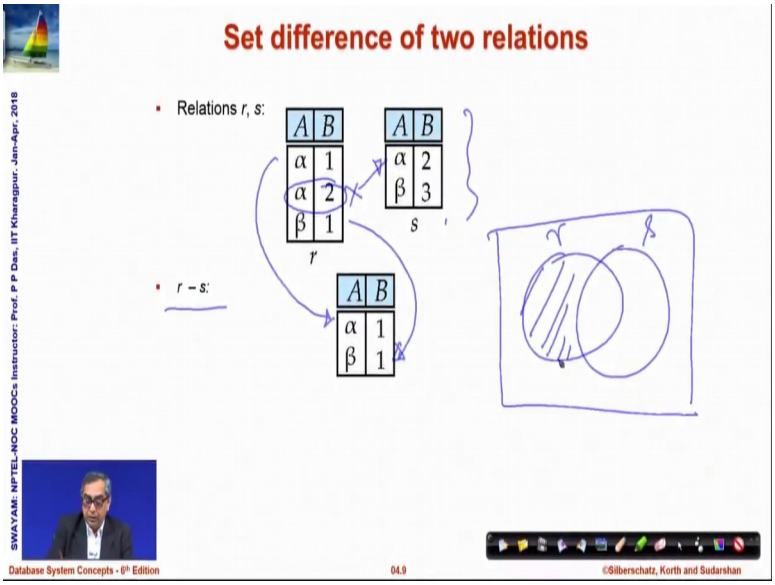
(Refer Slide Time: 10:09)



In terms of relation, α 1 is coming here, α 2 is coming here, β 1 is coming here, β 3 is coming here and α 2 is coming here. You can see that this record α 2 exists in both the relations. And by the set theoretic notion of uniqueness, in the $\mathbf{U}$ they have to be unitified. So, one of them will be removed, it does not matter because they are identical

anyway. So, for relations having the same set of attributes we can simply make a **U** of all it is records.

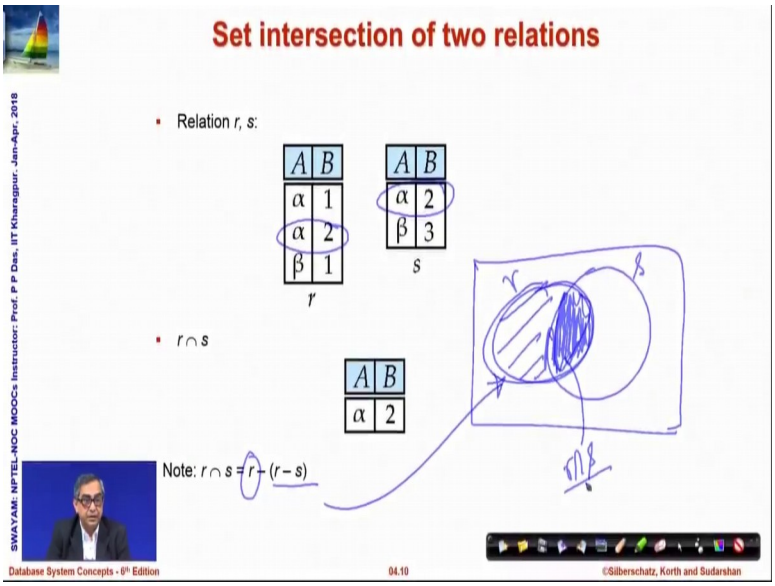
(Refer Slide Time: 10:57)



So, other the 3rd operation is the a 4th operation is doing a set difference it is works simply as set theoretic difference. Again, the two relations must have the same set of attributes and I can do a difference of **r-s** which mean that all tuples which exist in r, but do not exist in s will be included. So, this is included because, this is not here, but this is not included because it is in s this is included.

Because, this is this does not exist in the set s so it is the set, so you take the set r and then erase all the records like this which exist in s and you get **r-s**. So, if I look into just to recap if I look into the Venn diagram, then this is these are set **r-s** ∈ **r**, but ∉ **s**. So, there is a 4th operation that one can do with the in relational algebra.

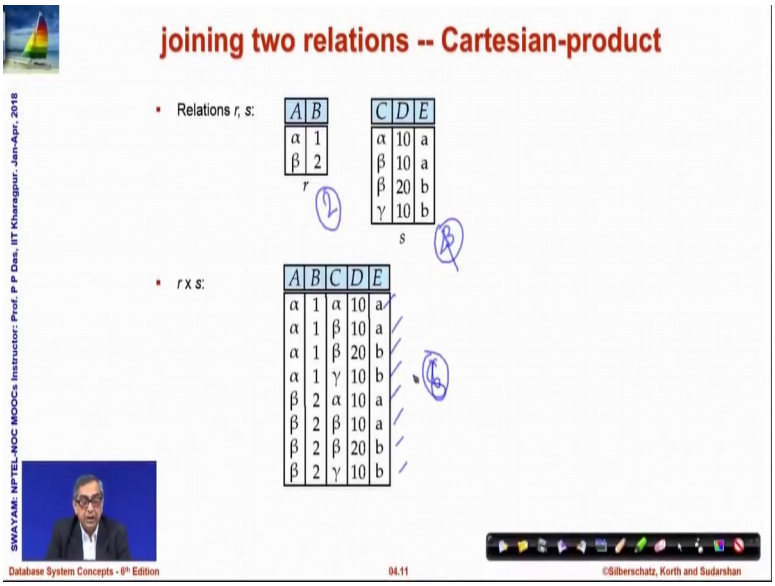
(Refer Slide Time: 12:28)



Fifth is a set intersection(∩) of 2 relations. So, again the two relations need to have the same set of attributes. You can take their intersection, which is the record which belongs to both. It is the record that belongs to both of them and as you are aware now, set intersection actually is not a new operation. It is not a fundamental operation because, if I have r, if I have s, then this is **r -s**.

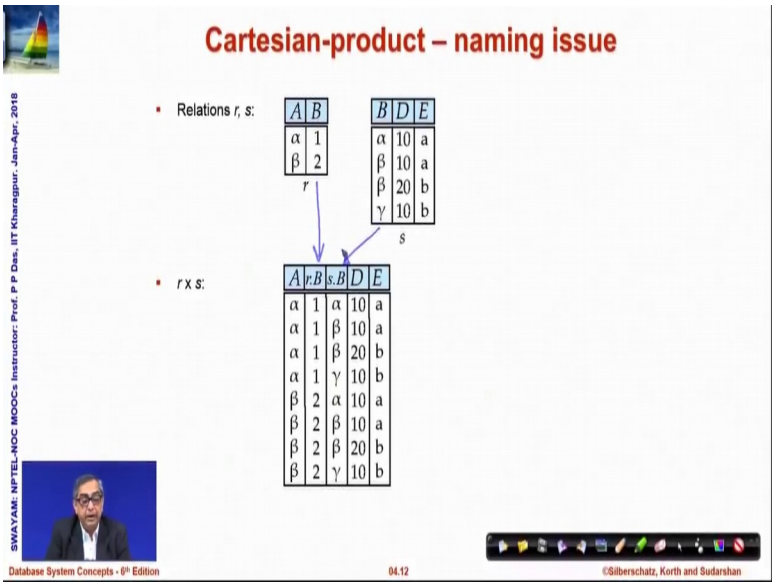
Now, if I subtract this **r- s** which is this set from r which is this bigger set, then what will be remaining? This is what will be remaining? So, if I subtract **r- s** from r then what will remain? Is necessarily the intersection of this is **r ∩ s**. So, set intersection is not a fundamental operation of relational algebra. But, can be used because it can be expressed in terms of set difference.

(Refer Slide Time: 14:03)



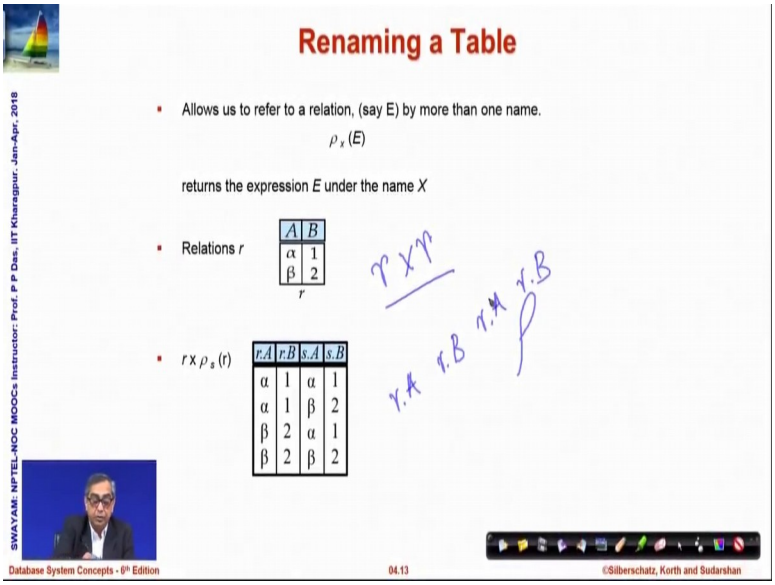
Next comes how can we join two different relations which have different set of attributes? So, relation r has A B and relation s has C D E. So, we can take a Cartesian product(×). So, taking Cartesian product is making all possible combinations. So, necessarily if since this has two relations and this has 3 relations. So, this will have 1 2 3 4 5 6 7 8 this has 4 relations. So, these are 8, 8 total all possible pairing of relations of r or records of r and records of s are included. So, that is the Cartesian product all possible combinations. This is this is how we can join two relations, but certainly what is a important is something which we will discuss shortly.

(Refer Slide Time: 15:00)



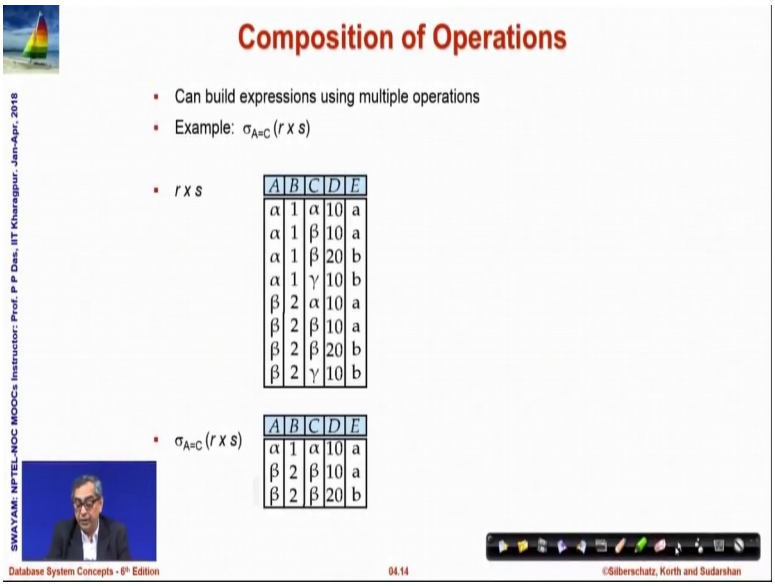
Now, in the Cartesian product then issue may happen because there could be attributes which are common between two relations. So, if you if two attributes are common when you take Cartesian product how do you put their name? Because as with the example show here between r and s the attribute b is common so, how do you take care of that? So, when the, such common names happen then we actually change the name of the attribute with the name of the relation. So, b coming from r will be called **r.b** and b coming from s will be called **s.b**.

(Refer Slide Time: 15:46)



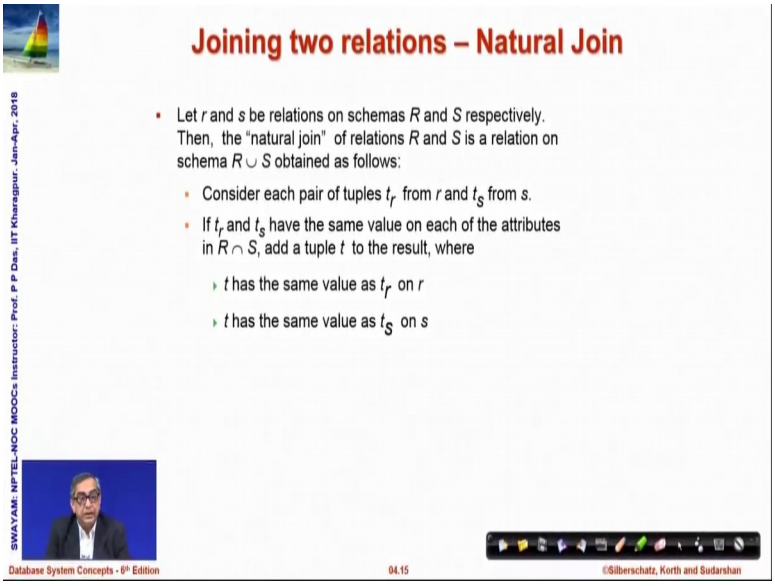
And accordingly, the relational algebra, gives you a way to rename a table and put it is name differently. So, this is given a this is given by this relation is given by this symbol **P**(rho). So, you can using a relation r you can actually give it a different name s and do that in terms of. So, with that you can actually because otherwise you cannot compute **r × r**. Because, if you try to do compute **r × r** then you will have **r. a r .b** and you will again have **r. a r .b**. So, you are using this to rename r to s and then compute this. So, renaming a table is another feature which is provided of course, it is not a fundamental operation of the algebra, but this is what makes the any kind of Cartesian product possible.

(Refer Slide Time: 16:55)



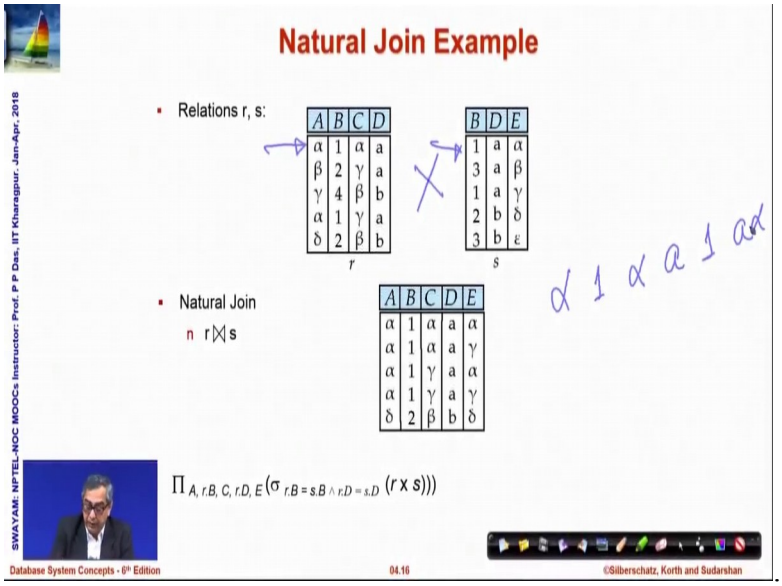
Finally, we can make composition of operations that if for example, what we show here is we have two relations r and s you have taken a Cartesian product and then we have taken a selection. So, taken a Cartesian product of r and s to produce the table as you can see the process($\mathbf{r} \times \mathbf{s}$) and then $\boldsymbol{\sigma_{A=C}}$ $\mathbf{(r \times s)}$ based on that condition. So, all these operations can be combined in multiple different ways to give you really complex relational algebra operations.

(Refer Slide Time: 17:33)



There is a nice operation which is a derived one which can be written in terms of other operations which is called a natural join(⋈) which we will use very heavily.

(Refer Slide Time: 17:48)

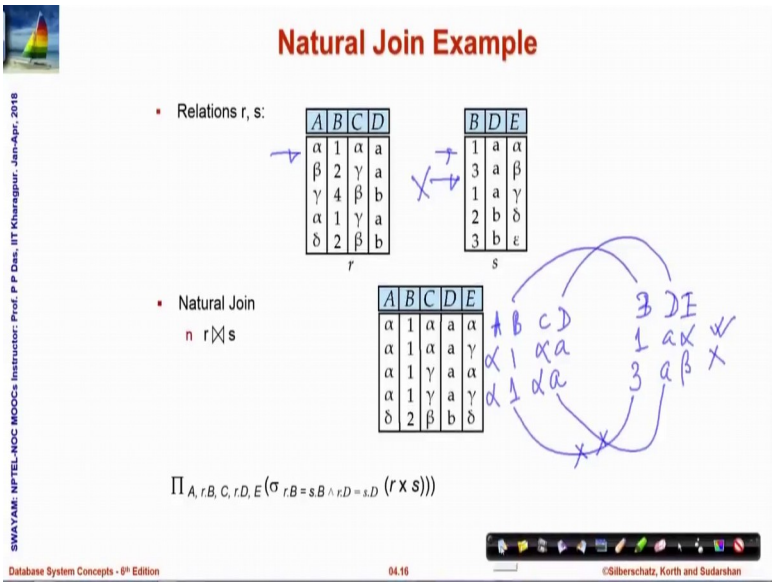


Let me first show you an example of that, suppose I have two relations r and s and what is important is there are some attributes which are common between them. Now, we saw earlier that in Cartesian product, in terms of common attributes. We basically the attributes got renamed in terms of the table name, but this is not what we are looking at in relation natural joint.

What we want to say is, if an attribute is common between two tables then, while you join them, the records from two fields can be joined if their value on that common attribute is same. So, it is what it tries to do is it tries to make a Cartesian product of this two tables (**r** × **s**)first take all possible combinations, but then you select only those rows where the values are identical between columns having the same name. So, for example, if you if you if you look into this row and this row. So, what will happen in the Cartesian product I will have this **α 1 α a 1 a alpha**.

Let me write it in smaller. So, I am doing a Cartesian product I am looking at this row I am looking at this row.

(Refer Slide Time: 19:39)

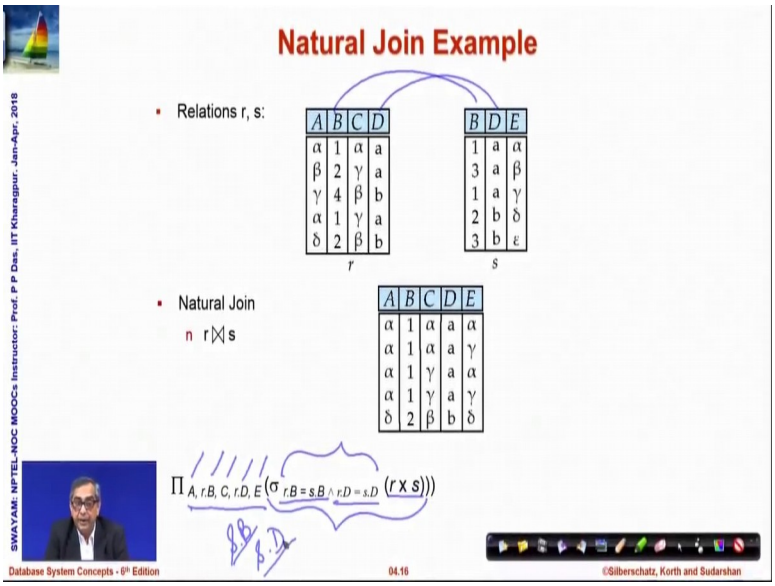


So, I have A B C D I have B D E and **α α** a 1 a **α**. Now, they match on B they match on D. So, I will say this is this will get retained, but in the Cartesian product I will also have the first row of r going with the second row of s **α** 1 **α** a 3 a β. Here, the B does not match D does not does match, but the B does not match. So, this particular entry will not go in the final result. To take the Cartesian product and you only retain those rows where the values match for the identically named attribute. That is why, so you take the Cartesian product now look at the expression the attributes common attributes are B and C, B and D.

So, the B attribute is **r.B** and **s.B**. So, we say that in the Cartesian product **r.B** = **s.B**. The name is common; the value will have to be same similarly D is a common attribute. So, r.D value in the **r.D** and the value in the **s.D** has to be same. So, based on the Cartesian product you do a selection for equality of values on attributes which are identical between the two column between the two relations between the two tables.

This is a final selection, as you do that you get a table where there are two Bs r.B s.B there are two Ds r.D s.D, but according to this selection for all at all records for all rows the value on r.B and value on s.B are same value on r.D and value on r s.B are same. Because that is how we have done the selection. So, there is no reason to keep two B columns or two D columns.

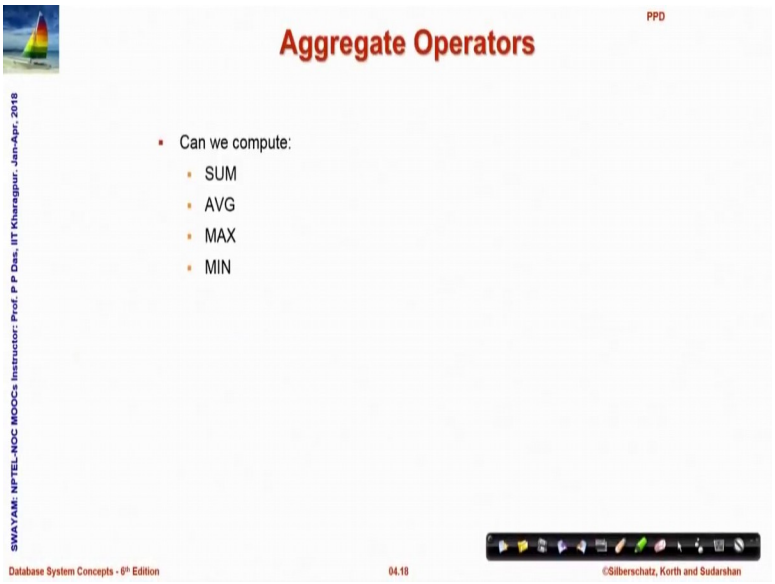
(Refer Slide Time: 22:33)



So now, you project based on **A r .B C r.D and E** which means that s.B and s.D are left out you do not project them. So, after you have done this projection, you get the final result of the natural join, which has a union of all the attributes that the two relations at A B C D and B D E union is A B C D E and you have all those records whose values matched on the common attributes between relation r and relation D. So, you can say that if I now do a selection, if I now do a projection on A B C or rather A B C D.

I will get a subset of r if I do a projection on B D E I will get a subset of s. So, this is the natural join operation in relational algebra as you can see this is a derived operation because we could use the Cartesian product selection and projection to get this, but as I tell you we will see more of this when we look at the look at all these different aj query coding, but natural join is one of the most widely used most fundamental relational algebra operation that you will often need beyond selection and projection. So, these were the 6 operations and the important derived operations of relational algebra. Besides that relational algebra has some aggregation operators.

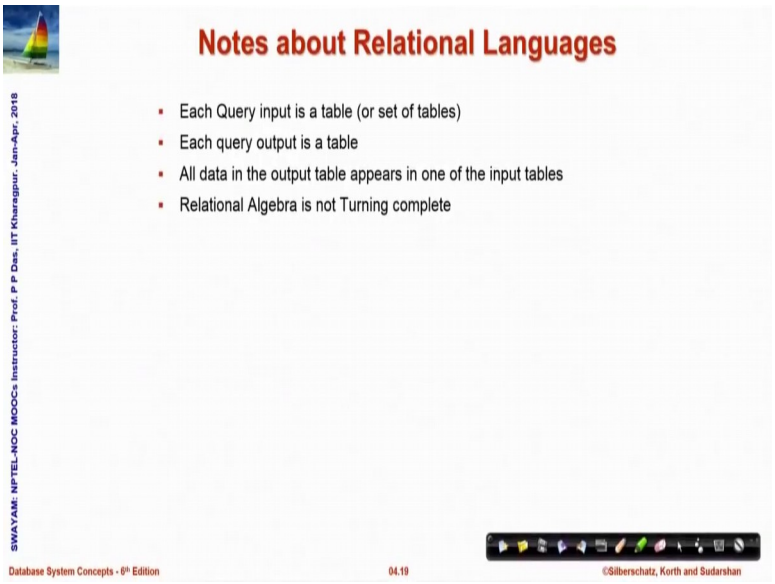
(Refer Slide Time: 24:13)



For example, given a table we could compute the sum of values on a column we could compute average of values, max of values, mean of values. So, these somehow aggregate values of multiple rows on a particular column. And therefore, these are called aggregate operators we will see when we talk about SQL. We will see how this really can be coded in SQL and used, but these are this become very convenient to use because often we will need to know.

If this is I mean these are the instructors. And so, let us see which instructor has a maximum load of courses? How many based on hours or which instructor has? What is the average load on the different instructors and so on? So, in every possible context different aggregate operators are frequently required and they are also available as operators in most of the pure as well as commercial query languages.

(Refer Slide Time: 25:18)

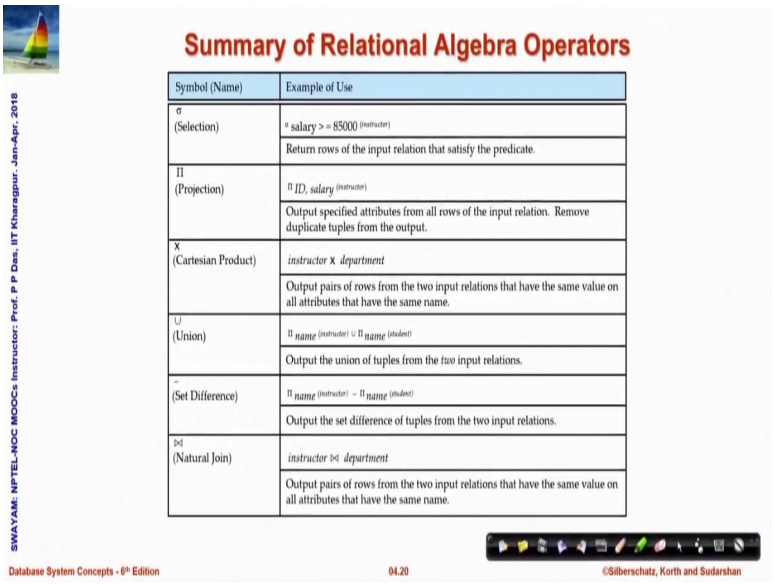


Finally, to note that relational algebra in relational average every query input is a table. And the output is also a table, it is always manipulating one or more tables into a single table that is what we are, I mean in very simple terms that is the way you can look at it. And all data in the output table appears in one of the input tables that is no new data gets generated. It is basically taking, selecting, combining data from different input tables. It does not generate a new data that is that has to be that is if I see that a, we an attribute for a particular row has a value 15.

Then there must be some input table where there is a record where in that field as an attribute value 50. Otherwise, this cannot happen again relational algebra is not turing complete in the sense that, there are algorithms which cannot be coded in relational algebra. We mentioned this earlier too and that is the reason that is a foundational reason of why the SQL language commercial SQL language is based on relational algebra is not turing complete there.

So, we might need to use other programming languages along with the relational algebra coding for solving some of the application problems.

(Refer Slide Time: 26:50)

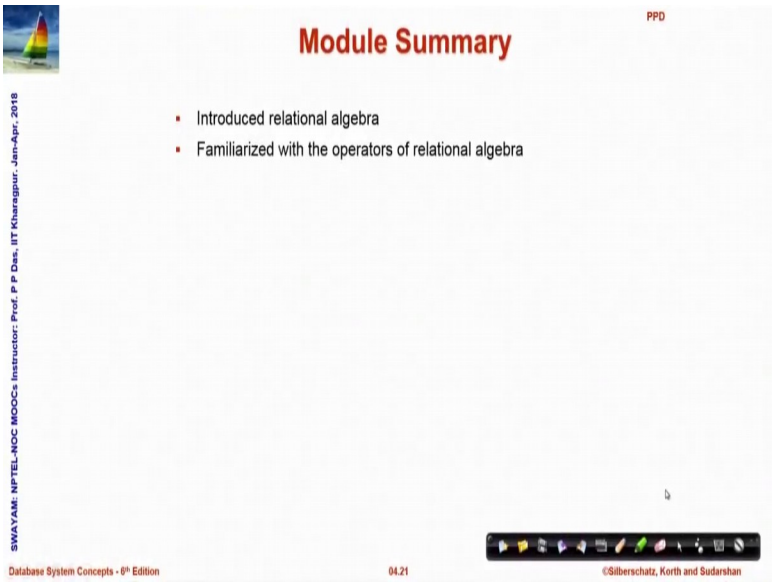


To summarize this is the, so this table is what you not only should remember, but you should become a expert of in terms of the operators of relational algebra which we will start using very heavily as we start doing the query coding and processing. So, we talked about selection, which takes rows selectively we talked about projection which takes out certain columns of a table. We talked about Cartesian product of two relations which make all possible combined relations.

We talked about union of records from two tables having identical set of attributes. We talked about set difference which again is the difference of records of one relation from another, given that they have identical set of attributes. We have shown that set difference can be used to also compute set intersection, it is not in the fundamental operation, but is the derived 1 and we have shown a very interesting operation based on Cartesian product selection and projection called natural join where two tables can be joined based on 1 or more common attributes they have.

Now, I am sure you have already noted that if I am doing a natural join between two tables. Which do not have any common attribute then the result is merely the Cartesian product because the selection around the Cartesian product has no condition to select any of the you know any of the fields any of the rows separately. So, it merely turns out to be a Cartesian product.

(Refer Slide Time: 28:38)



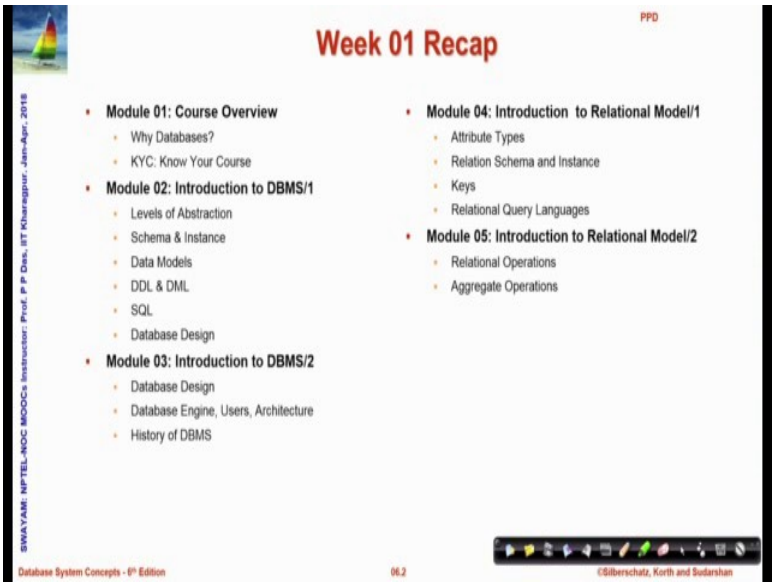
So, in this module, we have introduced the relational algebra and we have familiarized ourselves with the fundamental and derived operators of relational algebra. Going forward in the next module, will take a deeper look into the relational model and start progressing towards the query design and database design.

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

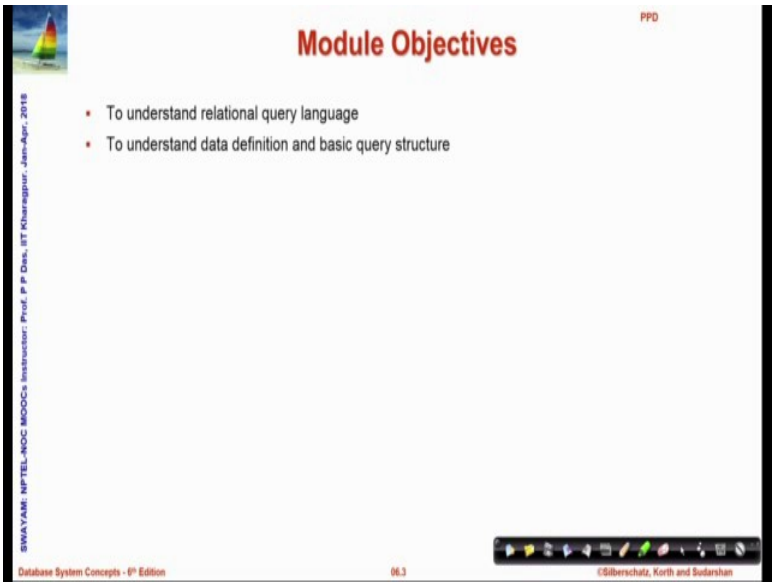
Lecture-06
Introduction to SQL/1

Welcome to module 6 of database management systems, this is the starting of week 2. In week 1 we have done 5 modules, after the overview of the course, we have primarily introduced the basic notions of DBMS and we have discussed about the relational module, the fundamentals of it.

(Refer Slide Time: 00:28)



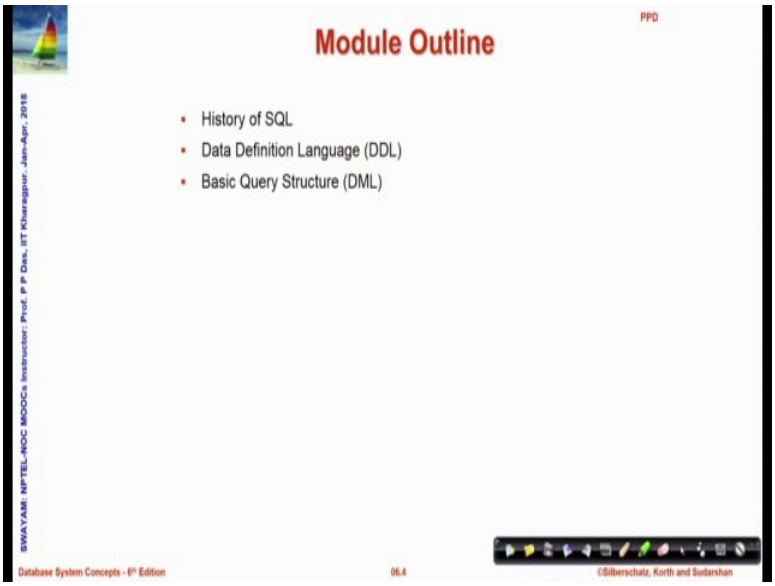
(Refer Slide Time: 00:54)



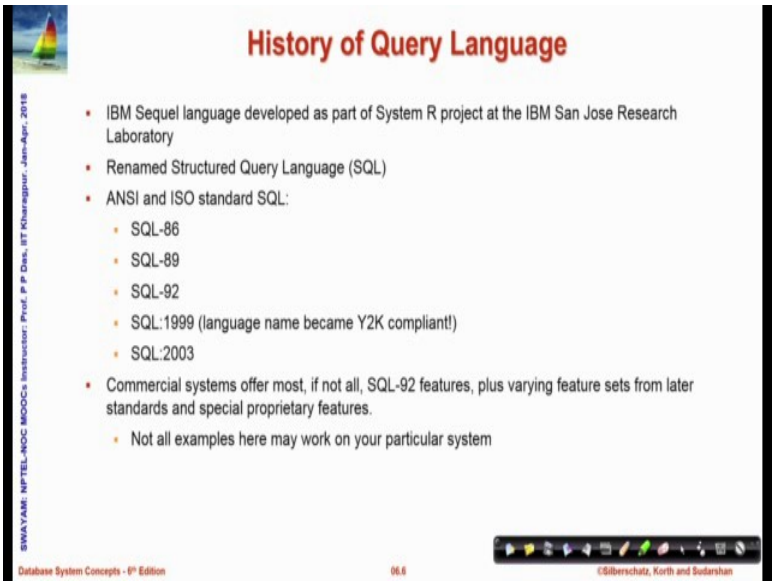
In this background, in this week we are primarily focusing on the query language the structured query language SQL. So, all the 5 modules will relate to discussions on query language and this module 6 7 and 8, the 3 modules will introduce SQL at a first level and the last 2 modules 8 and 9, I am sorry 9 and 10 will discuss about intermediate, that is somewhat advanced level of features in the SQL.

The objective of the current module is to, understand the relational query language and particularly the data definition and the basic query structure, that will hold for all SQL queries, particularly this the modules this week would be important for writing any kind of database applications, squaring the database to find information from the existing data and to manipulate it. So, please put a lot of focus in the whole material of this week and practice them well, to understand the basic issues of database systems, in depth in a well oriented manner. In this module we will first talk about the history and then we will see how to define data and start manipulating them.

(Refer Slide Time: 02:25)



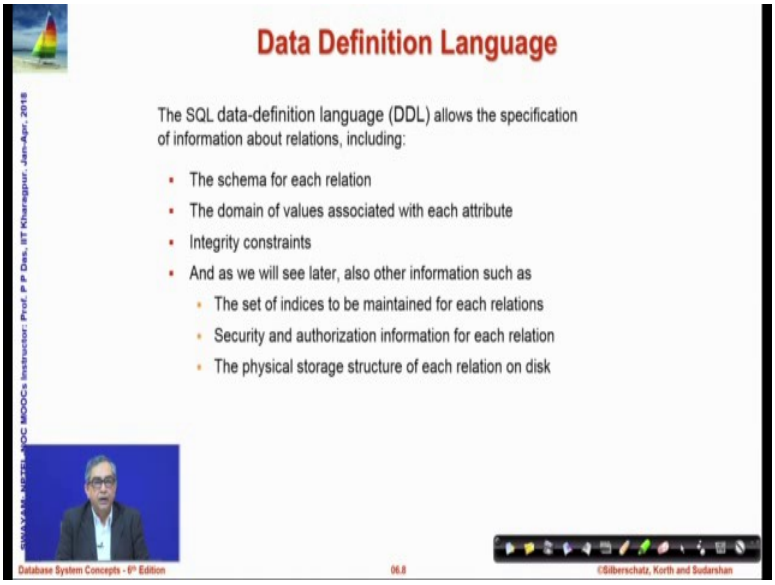
(Refer Slide Time: 02:37)



SQL was originally called IBM sequel language and was a part of system R, it was subsequently renamed as structured query language and like any other good programming language that we have, SQL also gets standardized by ANSI and ISO and there have been several standards of SQL, that has come up with SQL 92 being the most popular one, and the commercial system, most of them try to provide support for SQL 92 features, but they do vary between themselves. So, it is possible that the examples that we show here, may or may not all of them execute in the system that you are using. So, you will have to look at what standard your system is actually following.

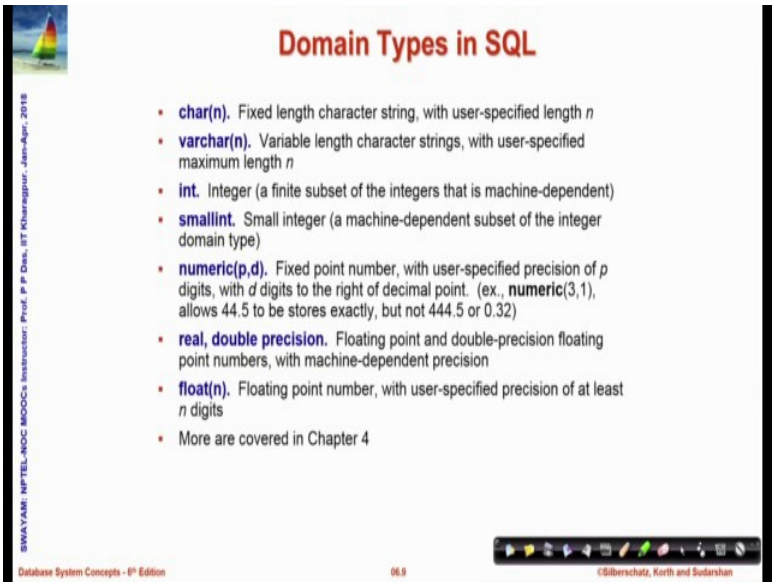
So, the first what we will talk about is a DDL data definition language, as we had discussed earlier this is the features to create that schema, the tables in a database management system.

(Refer Slide Time: 03:59)



So, it allows for the creation or definition of the schema, for each relation that we have in the database it specifies the domain of values associated with each attribute of the schema and it also defines a variety of integrity constraints, later in the course we will see that, it also has to specify other related information like indexing, security authorization, physical storage and so on.

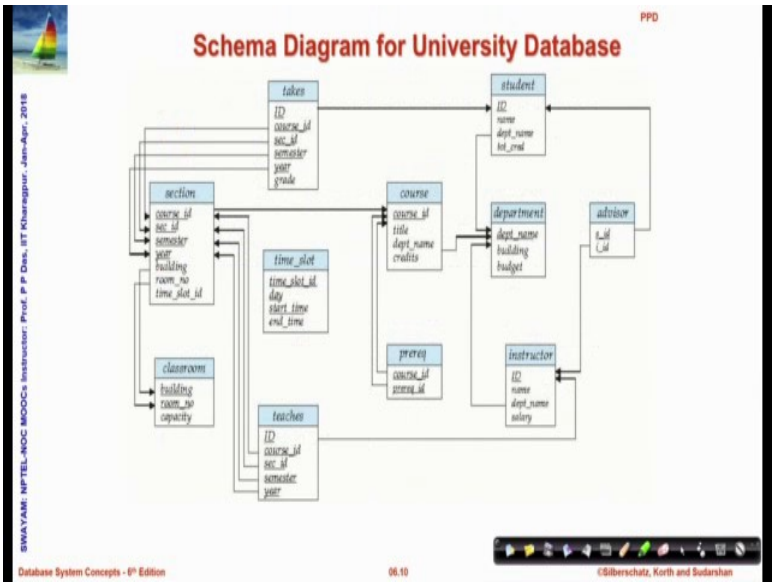
(Refer Slide Time: 04:30)



So, first the domain of possible values. So, we have already specified that, every domain in SQL is more of an atomic nature. So, they are more like the primitive or built in data types of languages like c, c++, java. So, the common domain types are character which are basically strings of character, having a certain length then you can have variable character string which means that the length here specifies that, the maximum length that the string can take, but a string could be shorter than that integer; obviously, then small integer, which in a system may give a smaller range of integer values.

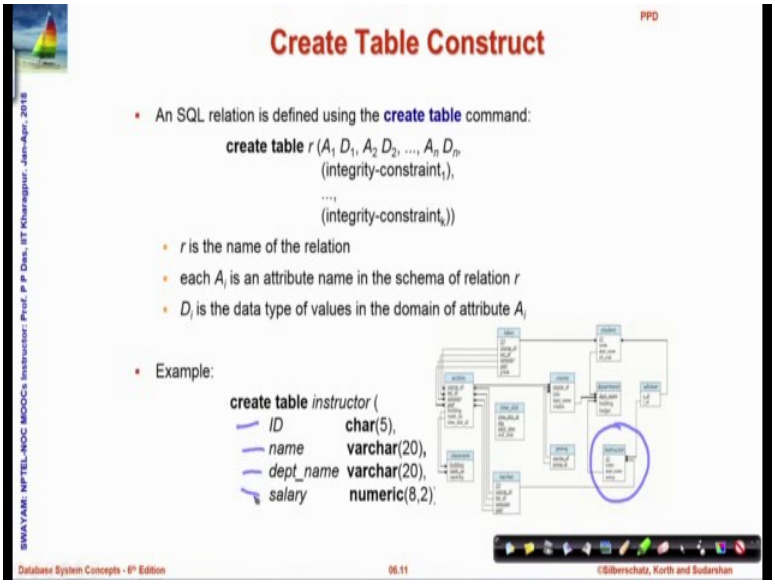
Then there is a numeric type which is often very important, which says what is the precision of the numbers that are to be written in this format stored in this format. So, d basically gives that precision value and p gives the size, then you can have real and double precision numbers you can have floating point numbers and so on, and there are some more data types, which we will discuss later in the course.

(Refer Slide Time: 05:52)



Given all these domain types; so, what we will try to do is, here is a schema for the university database, which has multiple different relations designed in that showing the attributes and marking out, what are the keys? And what are the foreign keys? So, we would take examples of some of these and try to code them in the SQL.

(Refer Slide Time: 06:19)

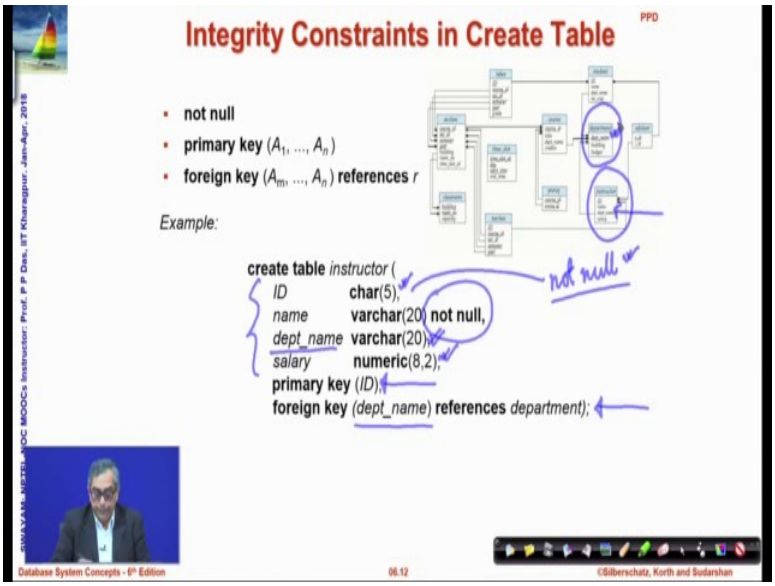


Now, to create a table this is how you go about, the SQL keywords CREATE TABLE is the basic command. So, with the CREATE TABLE you have to specify a name, here the name that is given is in terms of this name r, which is the name of the relation and then

you provide a series of attributes, separating by comma A_i are different attributes and for every attribute, there is a corresponding type domain type specified. So, it says that A_1 is of domain type D_1, A_2 is of domain type D_2 and so on. And all of these attribute descriptions, are then followed by a series of integrity constraints. It is possible that a CREATE TABLE may not provide any constraint, but often you will have a number of constraints to work with. So, here is one example.

So, in this we are trying to code the creation of this instructor table, as you can see it has 4 different fields ID, name, department name and salary and for each one we have specified the domain type. So, ID is char 5, this means that the identity of this table instructor will be strings of length 5 whereas, the name or the department name are strings, but they have a maximum length 20, but they could have be of variable length where a salary is of numeric type having specification (8, 2). So, it can have 2 decimal places and be of size 8 maximum. So, this is the basic form of definition that we have for creating a table, defining a table or defining a schema.

(Refer Slide Time: 08:28)



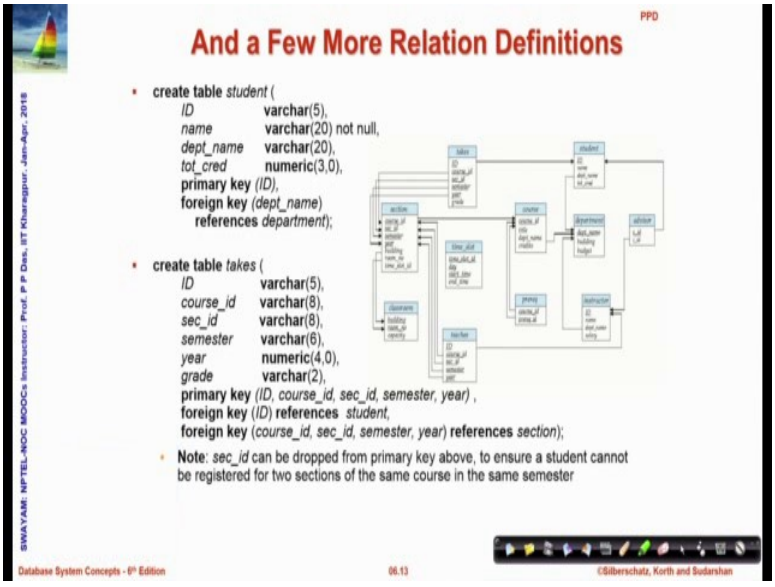
Now, we can add a number of integrity constraints to the CREATE TABLE, 3 integrity constraints we will discuss here, one is not null, one is primary key and other third is foreign key. So, not null we specify whether a field can be null or not, primary key as we have seen will specify the attributes which form the primary key, and the foreign key will specify the attributes which reference some other table and our key in that table. So,

here is an example, in the instructor relation here, we had seen this part, the attribute what we have added here is, this not null. So, we say the name is not null which means that, in the instructor table it is not possible to insert a record, where the name of the instructor is null that is unknown, but it is possible. if the same thing is not said about department name same thing is not said about salary. So, it is possible that these could be null.

Now, we additionally say that primary key is ID. So, this field ID is a primary key and it is a property of SQL CREATE TABLE command that, if an attribute is referred as a primary key, then it cannot be not null. So, you do not need to specify that is here you do not need to write not null, because it is a primary key it will be known to be not null, because certainly we have discussed that key is the distinguishing attribute in a database table. So, it cannot be null.

So, it will not be able to distinguish similarly, we have finally we have the third integrity constant which is foreign key which says that, it is referencing this table department and the foreign key of this is here, the dep_name this particular field is a foreign key, which is a key of the department table. And so, we will be able to refer this, from this table as a foreign key and we know that it will be key in the department table. So, these are the ways to specify the integrity constraint.

(Refer Slide Time: 11:24)



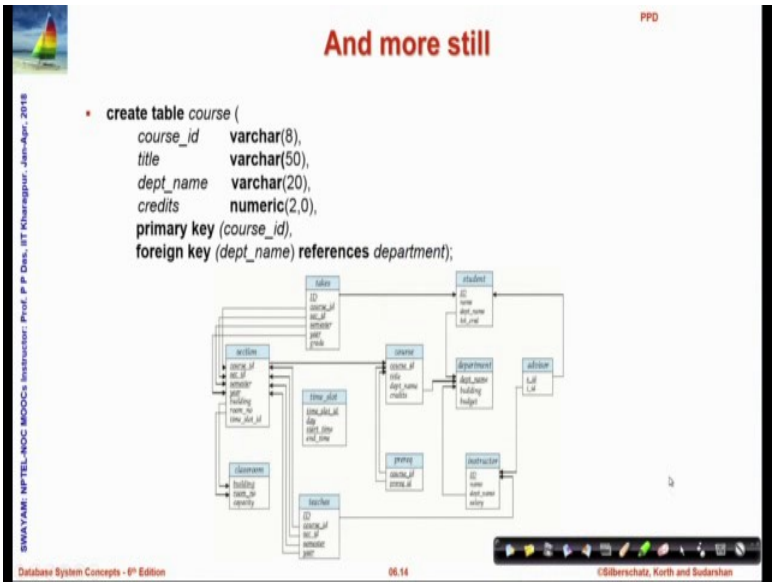
So, here are couple of more examples. So, I will not go through them in detail, I will request you to take time and carefully understand them, again these are about different relations that exist here, about the student and about the courses that the students take, and in every case we have specified the set of fields, that you have in the table in the design of the schema are listed, in the CREATE TABLE the ID information about the primary key is provided and also the information about the foreign key.

Here department name is the foreign key which is mapping to this point, similar things can be observed about the takes relationship which show how students are actually taking courses. So, it relates different it has a set of fields but it has 2 kinds of foreign keys, one that relate to the student through the ID and this combination of attributes which refer to the section.

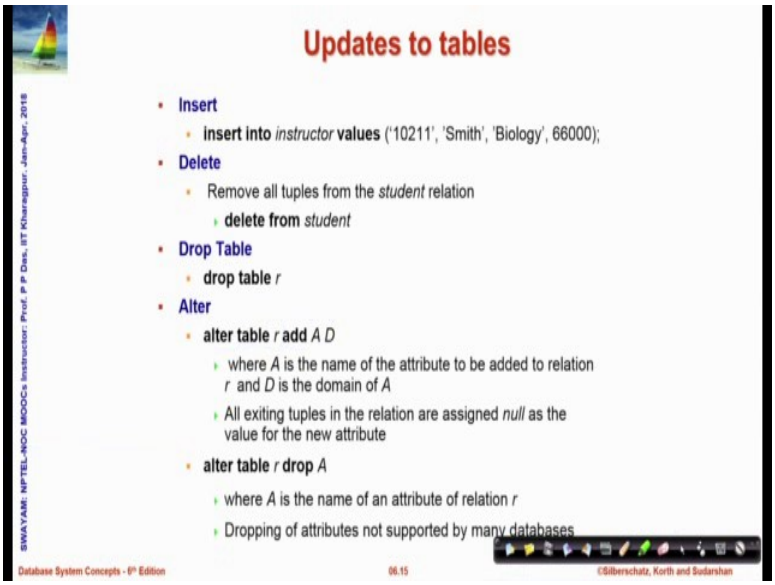
So, this is how different tables can be created using the data definition language, here is a note that you should observe that if you consider this section ID, the section ID is a part of the primary key which means that 2 records cannot be same, if they are different in the section ID, then such records are allowed. So which means that it is possible that, a student can attend or take a course in the same semester, in the same year with 2 different section IDs because they are primary keys.

So, they can be different. So, if we drop this from the primary key then we will enforce the condition that, no student will be able to take a course, in 2 sections in the same semester and the same year. So, these are the different design choices that we have and we will move on, here is one more example trying to show you the CREATE TABLE command for the course relation, that we have in the university database.

(Refer Slide Time: 13:59)



(Refer Slide Time: 14:11)

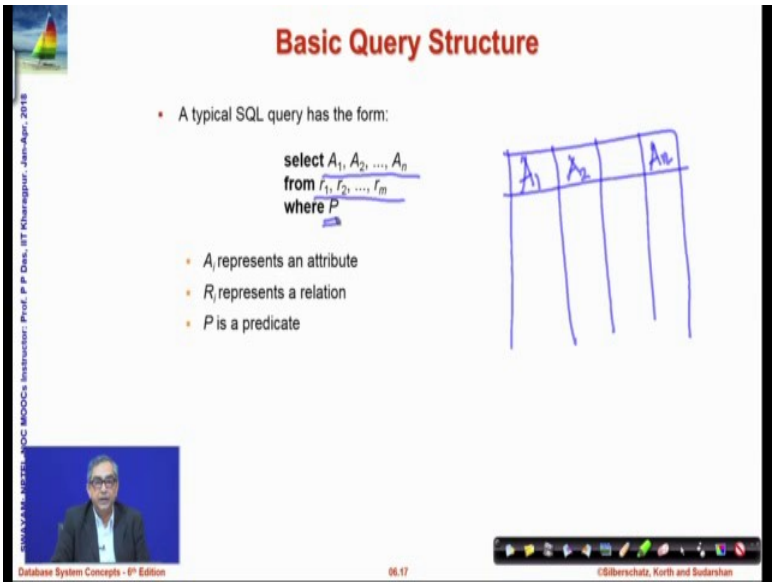


Moving on, let us look at how to update or actually put in different records, in a table which has already been created, the basic command is insert, and the keywords for that is INSERT INTO and values, in between you write the name of the relation, where the record will have to be inserted and then the values, will have to be put as a tuple in the same order in which you have defined the attributes of that relation, and certainly each of the values like this is ID value, next is the name value, the department, the salary, each one of them should be from the same domain type, as has been specified during the CREATE TABLE come on.

So, these things will have to remember. And so, every record will get inserted through one insert command, similarly a deletion can be done by DELETE FROM students, if you do DELETE FROM students without specifying which record you want to delete, basically all records will get deleted. We will see how selective deletion will happen, that will come on later. DROP TABLE is a command to remove a table, a table that has been created can be removed from the database altogether, by doing DROP TABLE and the relation name you can also change the schema of a table by using ALTER TABLE.

So, the form is altar table is a key words, you can add a new attribute to relation are by writing the name of the attribute and the domain of the attribute one after the other. Similarly, it is possible also to drop an attribute that already exists and the syntax for that will be, ALTER TABLE the relation name drop is a keyword and the name of the attribute, mind you all database systems may not allow you to drop an attribute to ALTER TABLE to remove attributes. And so, it works in some and it does not work in the rest.

(Refer Slide Time: 16:54)



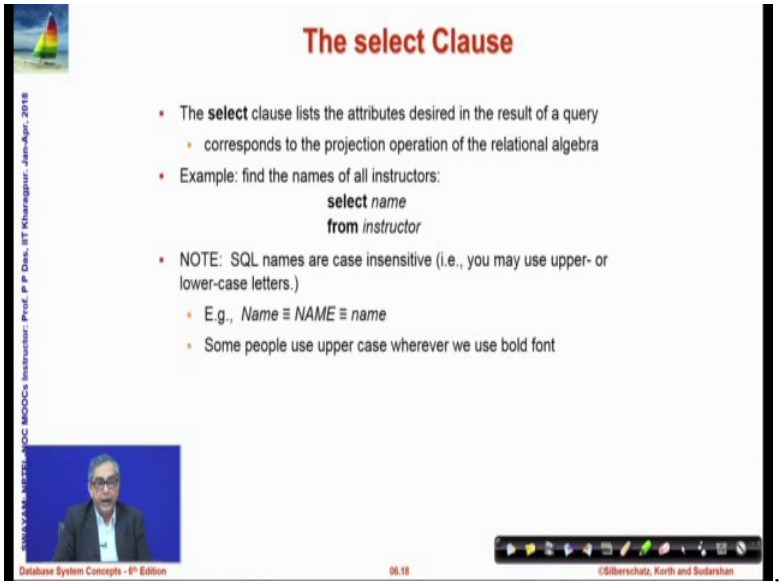
Now, that was about the definition of the table and the basic definition of the data. So, now we will get into the basics query structure which is with tables with existing data, how do I query and find out different information.

So, the structure of an SQL query and this you should observe very carefully is normally said to be, SELECT FROM WHERE colloquially will often say, let us have a SELECT

FROM WHERE. So, it has 3 keywords select which is followed by a set of; this is a set of attributes. So, this specifies that, when a select query runs it will finally give us a new relation and in that relation, the attributes that will be available are the attributes that feature in the select list. The next clause or the next keyword in this is from, which specifies a set of existing relations.

So, r_1, r_2, r_m represent different relations and these are the relations, which will be used to actually find the information extract the information. Finally, the where clause has a predicate as a condition, which specify that what condition has to be satisfied. So, that certain peoples from the relations r_1 to r_m, will be chosen and put in this new selected result, table in terms of the attributes A_1 to A_n.

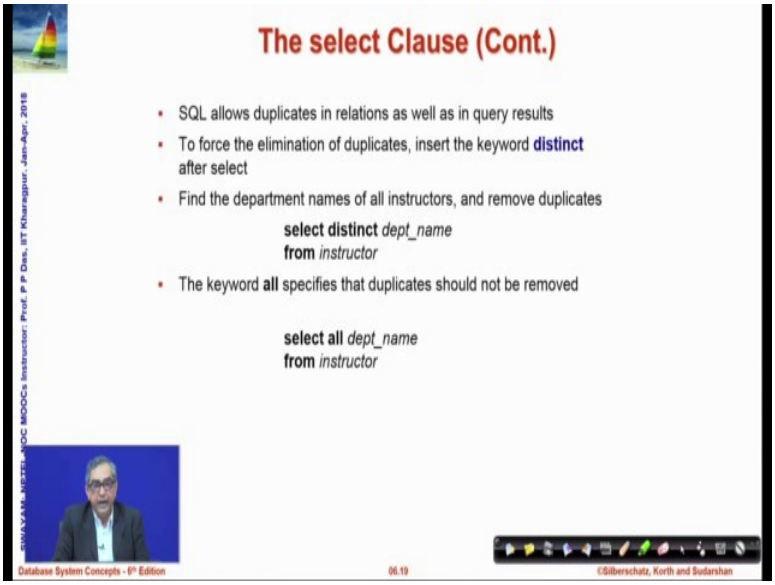
(Refer Slide Time: 18:37)



So, this is the basic understanding of the structure of the SQL query and naturally as I mentioned that will result in a relation. Now, we will go over each and every clause carefully, the select clause as I said will list all the attributes. So, it is like a projection (**Π**) in terms of the relational algebra that we have done. So, if we write select name from instructor then this will result in finding the names of all instructors from the instructor table, because this is, this you know is a relation, because it is happening it is a featuring in the from clause and in select, we are saying that the attribute that we want to select is the attribute name.

So, it will the instructor table has 4 attributes ID, name, dept. name and salary from that it will simply take the name of the instructor and list that in the output table. So, the basic form of selection that happens and this point you may also note, that in SQL everything is case insensitive, it does not matter whether you write in upper case or lower case. So, you can choose the style that you prefer to use.

(Refer Slide Time: 19:51)



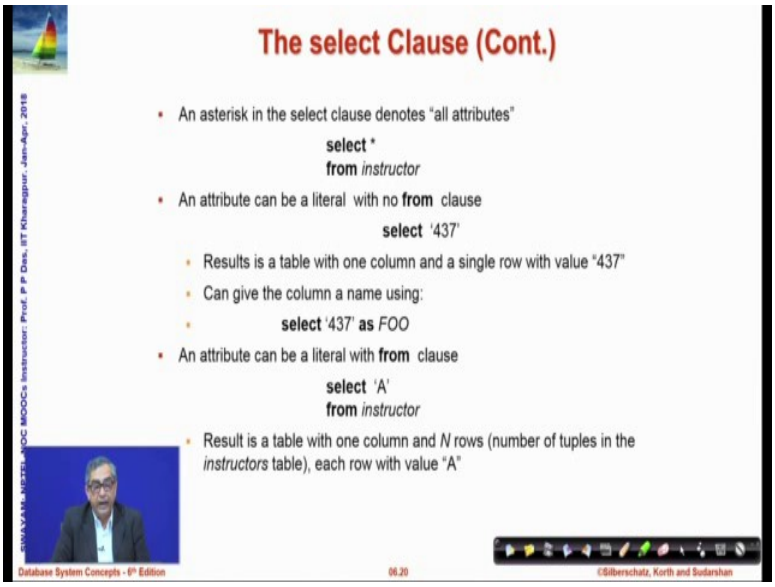
This is a very important factor, that you should keep in mind that we said, while introducing relational algebra, that in the relational algebra every relation is a set and which means that according to set theory. We cannot have 2 tuples in the same relation, which are identical in all it is values, because set theory does not allow that, but please keep in mind that SQL actually allows duplicates in relations. So, it is possible that in the same relation in the same table, I may have more than one record which are identical in all the fields, in all the attributes of that table and this will have lot of consequences and we will see how often, this property will have to be used.

So, if you want a typical set theoretic kind of output, that is if you want the result to be distinct, all records to be distinct, then you have to explicitly say that you want distinct values to be selected. So, all that you are doing is, you are after select and before the attribute name, you introduce another keyword distinct.

So, select distinct depth name from instructor will actually, select the departments of all instructors and quite why if it just selects department name of all instructor, then it is

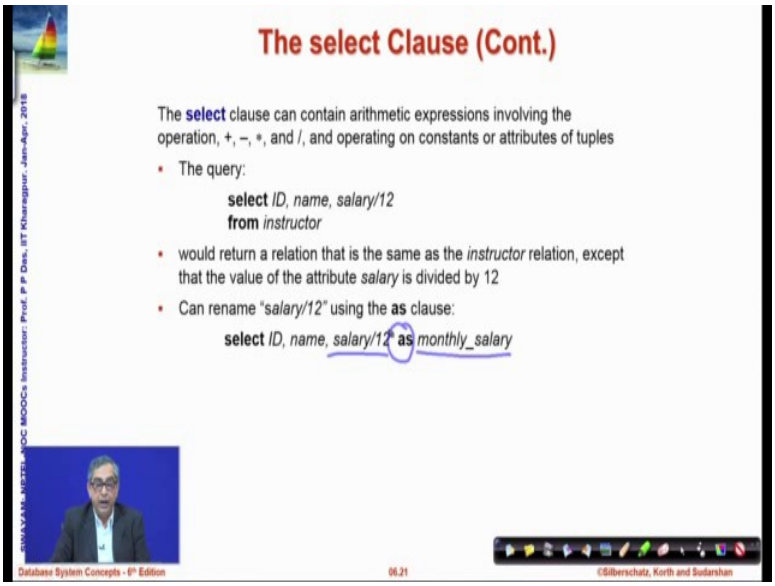
quite possible that the same department name will appear number of times, because every department has multiple instructors. But when we use distinct, then every name will feature only once in that selection, then you can also specify another keyword all, which ensures that the duplicates are not removed. So, if you do select all depth name, then all the names will feature with duplicates, if some department had 3 instructors the name of that department will feature thrice.

(Refer Slide Time: 22:05)



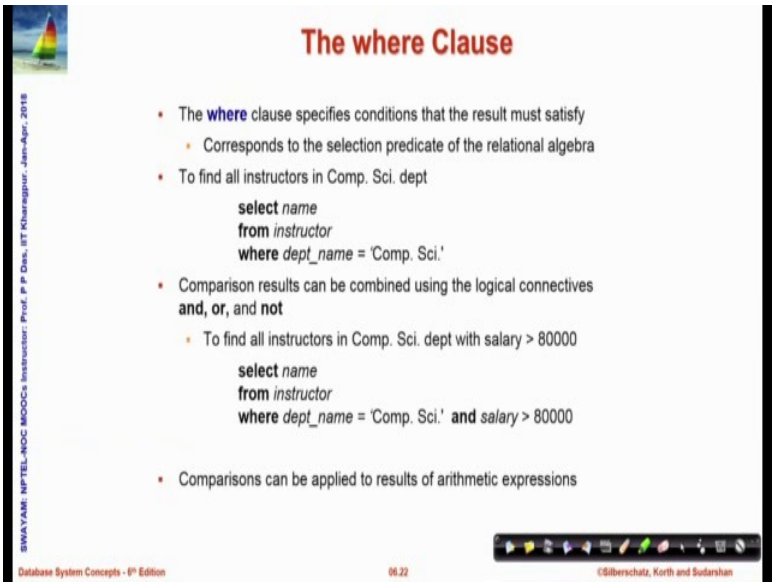
You can use an asterisk after select, to specify that you are interested in all the attributes that the relation or the collection of relations in the from clause has. You can also specify a select with a literal and without a from clause, if you do that then it will simply return you a table with a single row having that literal value and you can also rename that table, using what is known as clause as a command. So, this will give you a table Foo, where there is only one row and that row has an entry 437. You can use that, for other purposes also you can do a select of a literal, from a table with using a from clause where in, you will get a single column table, where as many is as there are records in the instructor will be produced.

(Refer Slide Time: 23:16)



Select clause can also use basic arithmetic operations for example, here we are showing a select where the third attribute as you can see, the third attribute is salary by 12, assuming that the instructor table has a salary number which is annual salary by 12 naturally you give you the monthly salary. So, those such arithmetic choices can also be made. you can also rename that field that particular salary by 12 field by a new name, as I said as can be used to rename. So, if we if you use that then, when you get the output you will get the column names are ID, name and monthly salary and in monthly salary you will actually have a computation, which is salary by 12 and in the same way, you can use multiple different kinds of arithmetic operators.

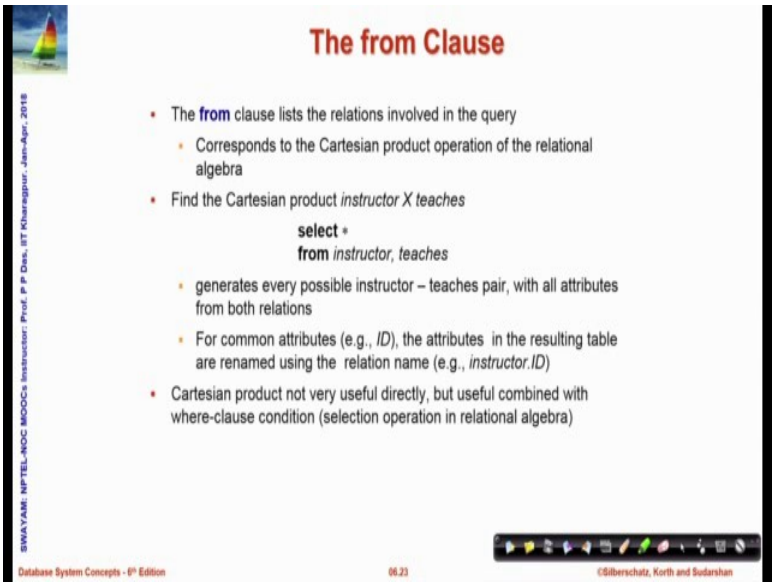
(Refer Slide Time: 24:16)



Now, we come to the where clause, where clause specifies the condition is a predicate which corresponds to the selection predicate of relational algebra. So, it will specify some condition, here is an example if we want to find all instructors from the instructor table, who are associated with computer science department, then you can say select name from instructor and to specify that they are from the computer science department, you will specify department name is equal to computer science.

So, this will ensure that you select the records only when this condition is satisfied. So, all records for which department name is different from computer science will not be included here. You can also write predicates using the different logical connectives and or not and so on. So, here is an example, where you are finding all instructors in computer science with salary greater than 80000. So, here we have used and clause. So, only records where the department name is computer science and salary is greater than 8000 will be chosen in the result. So, and then the projection (**Π**) will be done on the name of those instructors, you can apply comparisons of arithmetic expression. So, where clause can really write different kind of things.

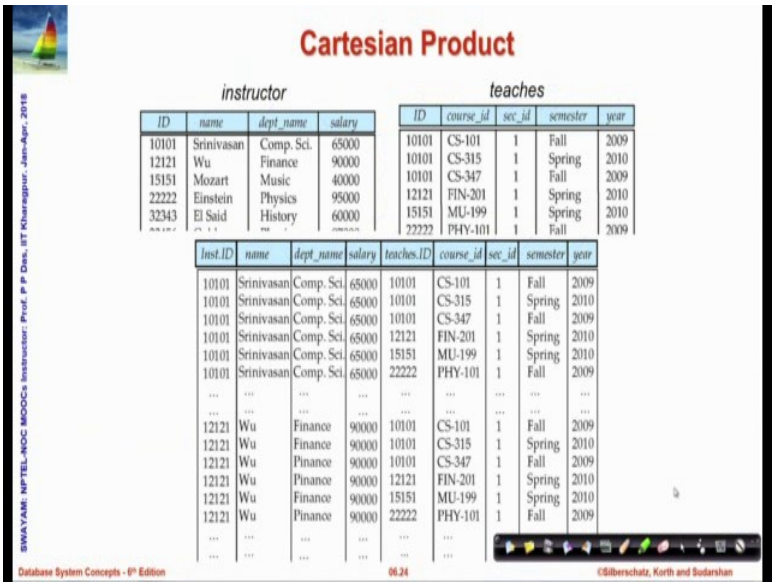
(Refer Slide Time: 26:04)



Finally, the from clause is sets all the different relations from where you are actually looking for the records. So, it kind of corresponds to the Cartesian product of the relational algebra. So, if we want to say compute instructor Cartesian product teachers, then you can say SELECT * instructor one table comma teachers. So, this will choose records from instructed relation, as well as from teacher's relation and in all possible combined way, it will put them in the output we have used a star.

So, all fields of in instructor and all fields of teaches will be there in the output, and since some fields may have identical name like ID in instructor and there is ID in teachers, they will be qualified by the name of the relation, from can have 1 relation, 2 relation any number of relations as you require. So, this will cause the Cartesian product to be computed which may not be very useful as an independent feature, but we will see in the next module how it can give very important computations, in terms of computing joints and so on.

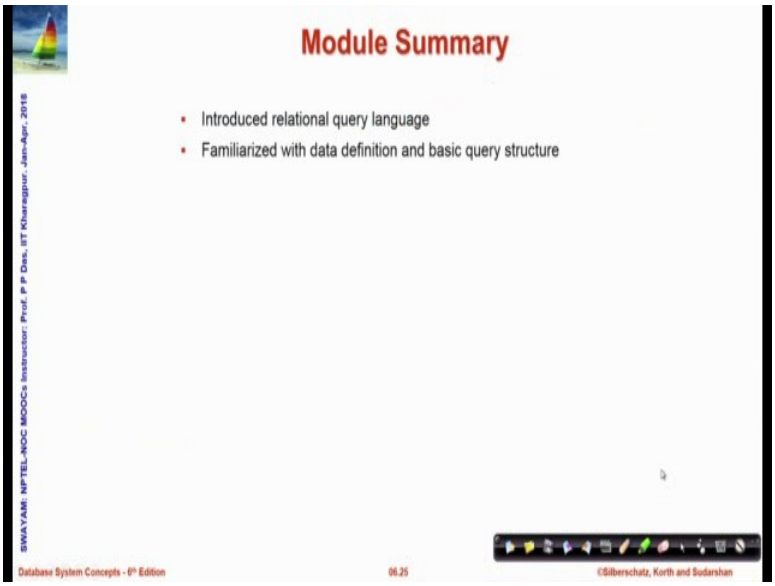
(Refer Slide Time: 27:37)



So, here is an example of the Cartesian product that we talked of. So, here is a instructor relation, the teacher's relation and as you can see when we have done this cartesian product, that is SELECT * from instructor comma teachers, then all fields this is an ID of the instructor, there is an ID in teachers.

So, that is also specified here qualified by the name of the relation whereas, name comes in directly because there is no attribute called name in teachers, the department name comes in directly, salary comes in, directly course ID comes in, section ID comes in, semester comes in, year comes in and so on. And the combination of all tuples in their instructor relation, against all tuples of the teacher's relation all possible combinations have come in this result, which eventually is a cartesian product of the relational algebra.

(Refer Slide Time: 28:50)



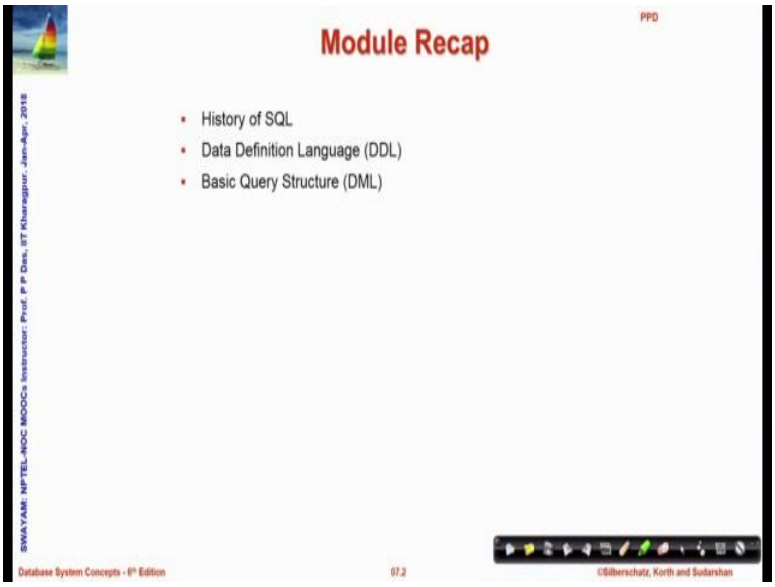
So, this is what we have to summarize, we have introduced the relational query language and particularly familiarize ourselves with the data definition that is creation of the table creation, of the schema with the attribute names, domain types and constraints. And the updates to the table in terms of insertion and deletion of values or addition or deletion of attributes or removing a table altogether and then, we have given the basic structure of the SELECT FROM WHERE query of SQL, which will be the key language feature of a query language, that we will continue to discuss all through this course.

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

Lecture - 07
Introduction to SQL/2

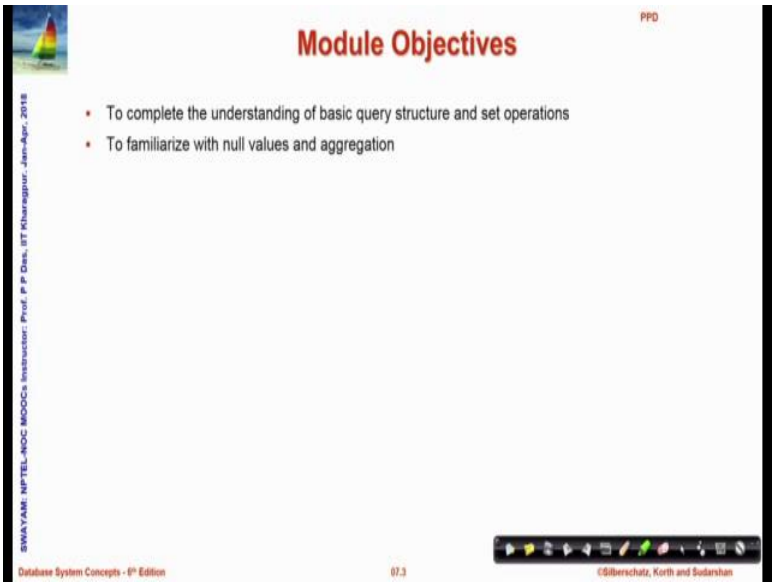
Welcome to module 7 of database management systems, this is a second part out of total 3 of introduction to SQL.

(Refer Slide Time: 00:31)



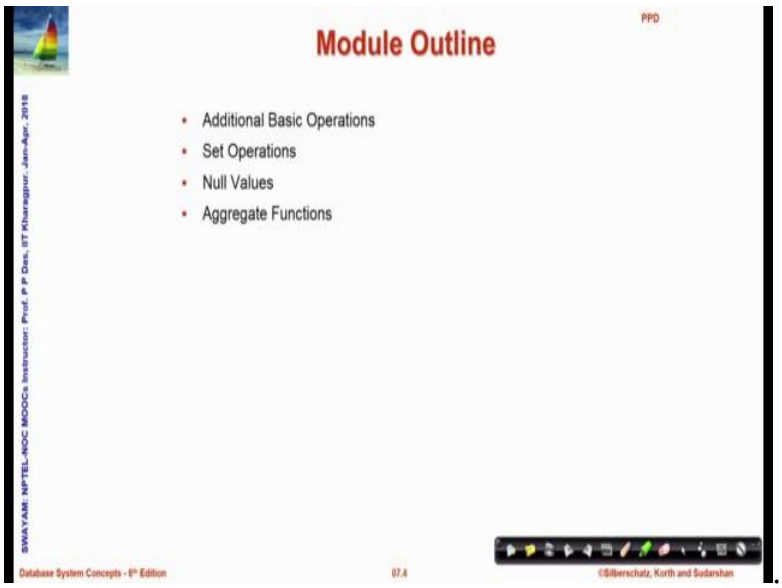
In the last module, we have discussed about the evolution of SQL the data definition language part of it and the basic structure of queries.

(Refer Slide Time: 00:42)



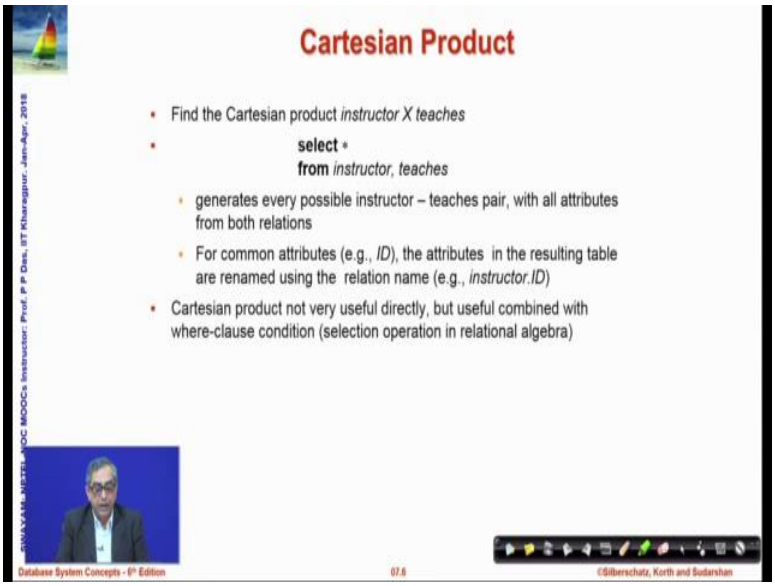
In the current module we will complete the understanding of the basics queries structure and will see how, common set theoretic operations can be performed in terms of queries. We will familiarize ourselves with the handling of null values and aggregation operation that will be frequently required for forming queries.

(Refer Slide Time: 01:05)



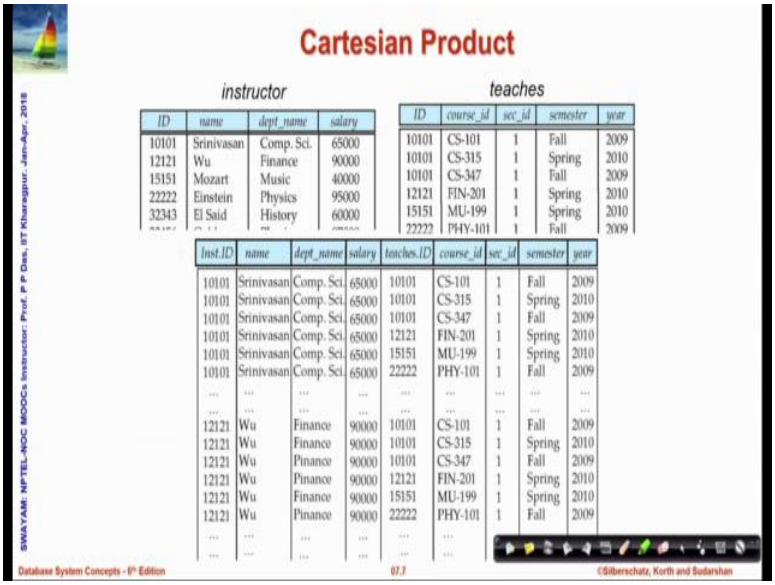
So, this is a module outline, these are the topics, that we will discuss. So, we started the discussion of some more basic operations in the query.

(Refer Slide Time: 01:15)

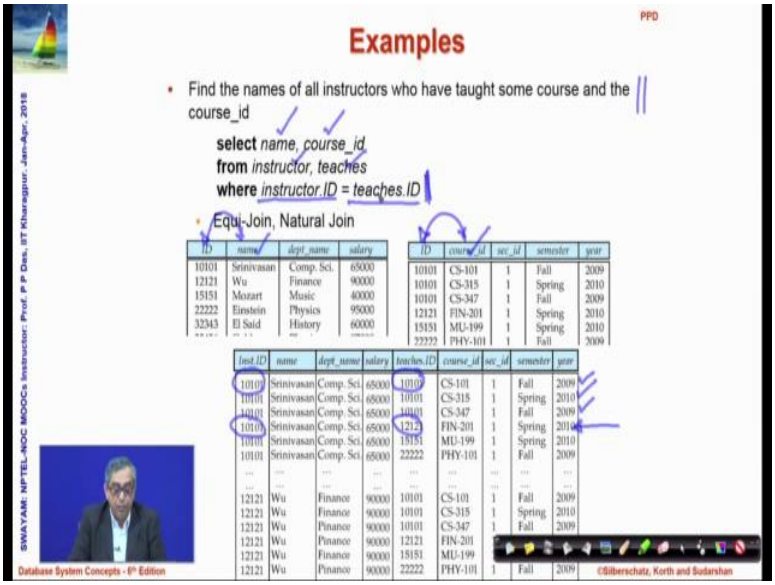


We have already discussed this that, if we do SELECT * FROM 2 tables then it results in a Cartesian product we have seen this result earlier..

(Refer Slide Time: 01:26)



(Refer Slide Time: 01:32)

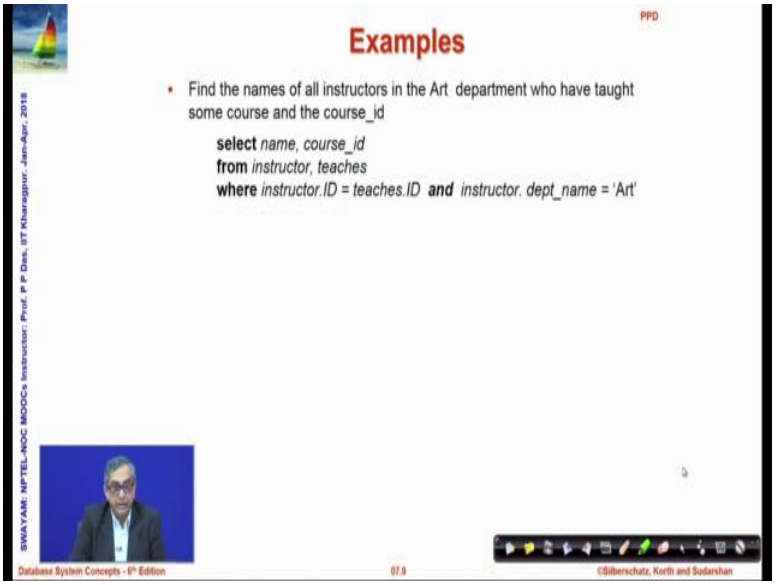


Know by itself as we had said that, by itself the Cartesian product may not be very useful, but suppose we want to answer this kind of a query, that find all names of all instructors who have taught some course and for those you also write the course_id. So, what we are interested in is, the name of the instructor and course_id.

So, we put that on the select course these are the 2 tables that are required because, the name of the instructor is there in the instructor table relation and then the relationship between which instructors teach which course is in the teachers table. So, in the instructor table we have the name in the teachers table, we have the course_id and also this relationship as to which course is taught by which teacher and here, we have the relationship between which id of the instructor has which name. So, what we do is, when we do this Cartesian product will get something like this, as we have already seen, but we want to qualify it by with a predicate which say that, we will out of all these combinations will choose only those where, the instructor.ID equals the teaches.ID. So, if we look at say in the first row the instructor.ID is this, teachers.ID is this, which is id of the teachers table they are same. So, which says the this instructors Srinivasan actually teaches the course CS-101 whereas, if you look into this row, it says that instructor.ID is this and teaches.ID is this and we are really not interested in this combination, because this combination does not convey anything meaningful. So, by the use of this where clause, we will try to choose only those records where, these 2 ids are same which will tell us that, this particular instructor actually teaches that particular course.

So, if we do that, then you will find that majority of the records that, actually came about in the Cartesian product are eliminated from this result. So, I have struck them out. So, you can currently see in this part you can see only 4 records the 3 courses taught by Srinivasan. So, and the one course taught by you. So, in the output you will have this name, this name part and this course_id part because, you are projecting on these 2. So, this will be the output table which will get generated and just to remind you, this is the notion of natural join that, we had discussed in relational algebra in this case we will actually call it equi join because, we are using a equality condition after the Cartesian product to join these 2 relationship. So, this is a very critical operation in many of cases in our database query system.

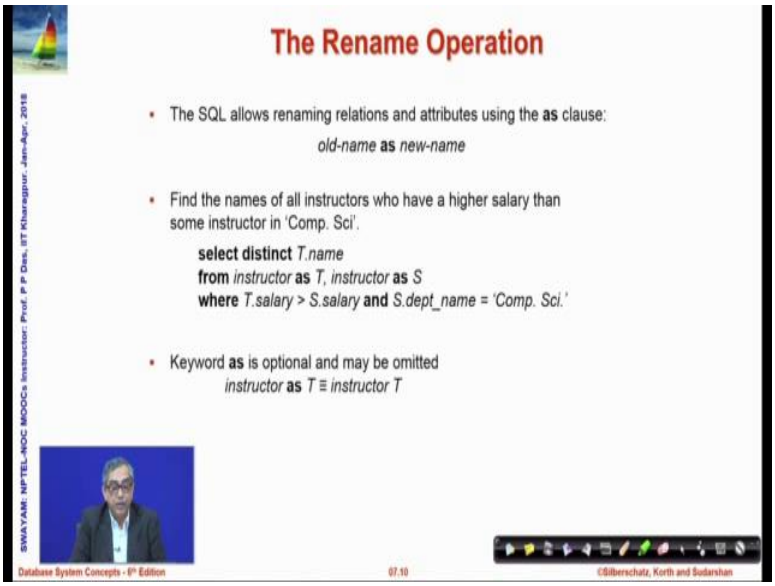
(Refer Slide Time: 05:23)



This is another extension of this similar example. So, here we have added another predicate in the where clause, specifying that instructed or department name is Art.

So, which means this will now, give the names of all instructors in the art department only who have taught some course and specify their course_id. So, in different such ways, you can manipulate and create queries.

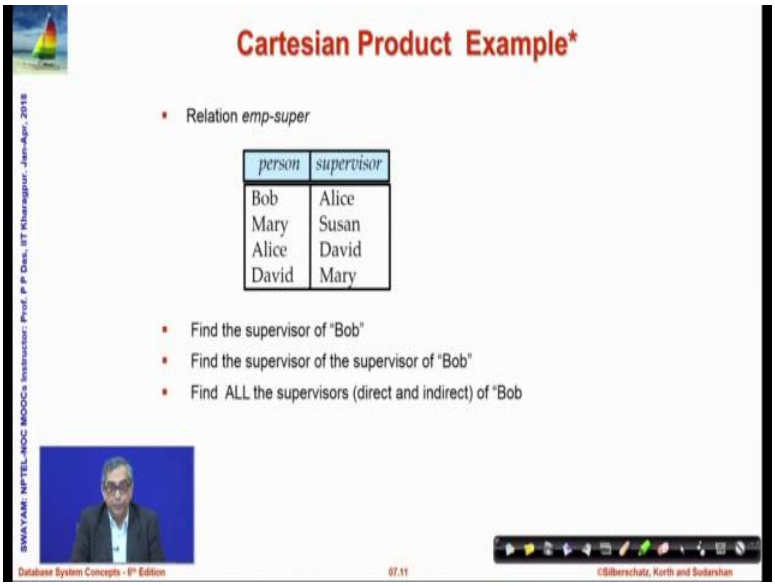
(Refer Slide Time: 05:57)



It is possible to rename, we have already seen examples you can rename a relation you can rename an attribute and the style is to use AS. So, here you can see that, in the SELECT query we have said from instructor as T. So, the name of this relation can be treated as T and we again see that as instructor as S. So, actually what we are doing, we are doing a join between the same relation; instructor and instructor.

And we are trying to find out all instructor who have higher salary than some instructor in computer science. So, the some instructor in computer science is specified by this condition, because department has to be computer science and the fact that salary is higher. So, as if you treat that, though it is actually a join between a product between instructor and instructor the same relation, but by renaming you treat them as if they are 2 different tables having name T and S, and then it becomes easier to write this kind of query. So, otherwise it is it is quite difficult to write this query to find out, because you need to actually create a product of one relation with itself.

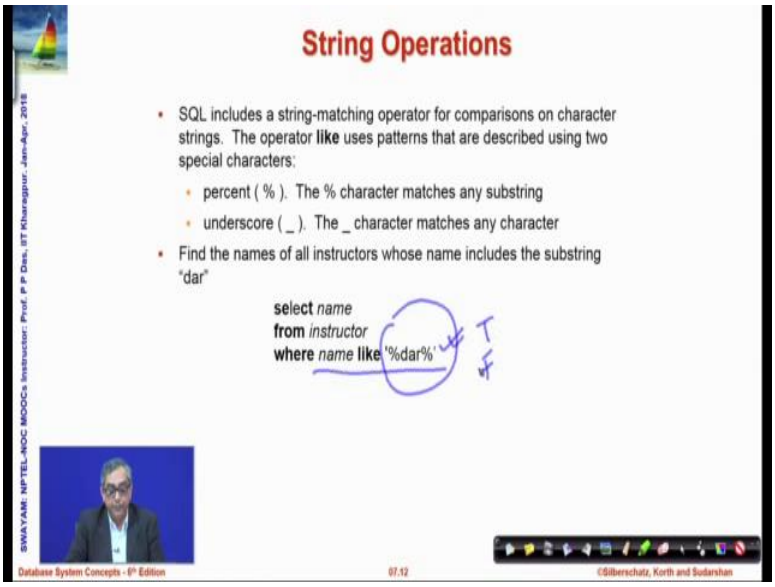
(Refer Slide Time: 07:30)



Keyword AS is optional you can just write instructor, and then the name the new name that you want to give and that itself will work here. Is another Cartesian product example, here given a relation which lists a person and his or her supervisor, we want to find out all supervisors direct or indirect of that person.

So, I leave this as an exercise to you, to think over as to how we can actually compute this query?

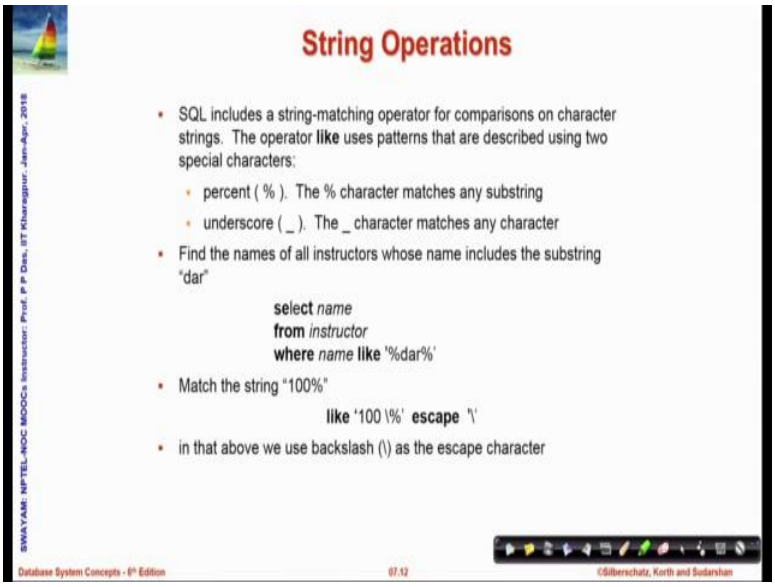
(Refer Slide Time: 07:56)



SQL supports several string operations and of particular interest at 2 specific symbols, characters which allow us doing certain match, percentage is used to match any substring and underscore is used to match any particular character and we use a keyword LIKE, to find out different string patterns that can be matched. So, here we want to find the names of all instructors, whose name includes the substring "dar" and by writing this so saying the predicate is formed is named like this.

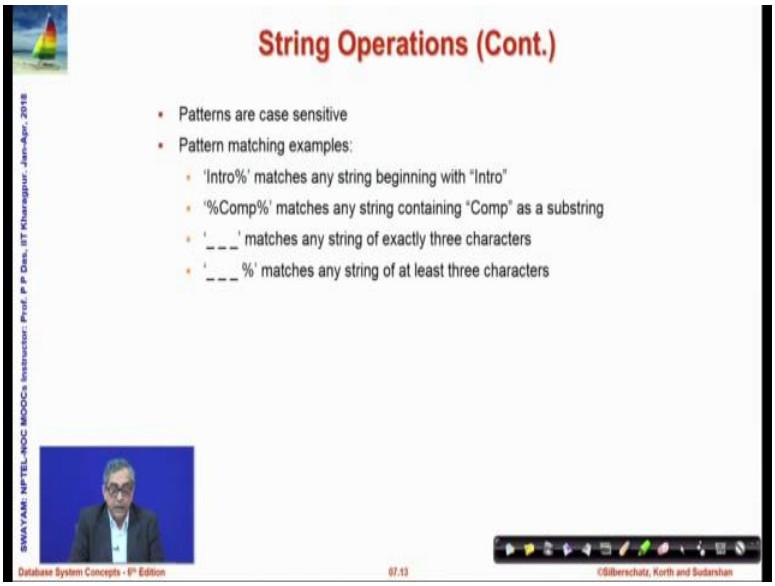
So, what is there is a percentage before there is a percentage after. So, anywhere "dar" will feature in the name, this predicate will turn out to be true otherwise if there is no "dar" in the name the predicate will turn out to be false and that particular record will not get selected. So, in this way we can using LIKE, we can actually do different kinds string operations as conditions in the weight loss.

(Refer Slide Time: 09:07)



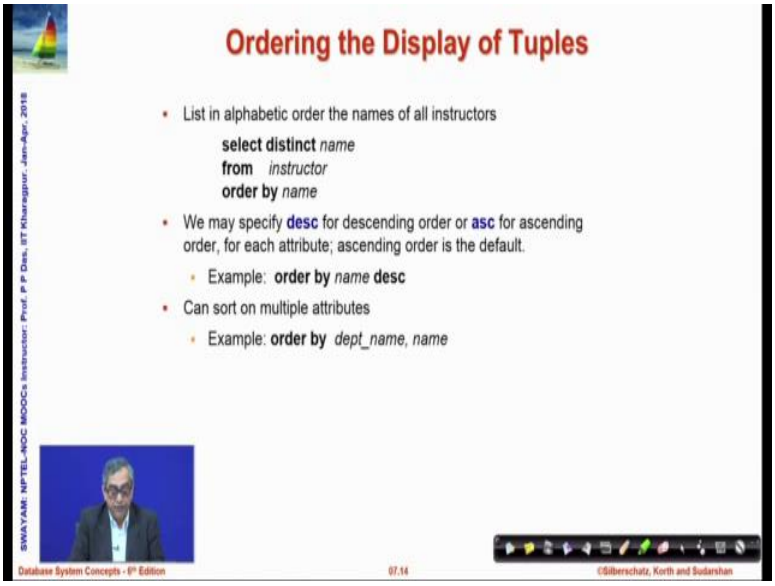
So now, naturally this brings in an issue of for what if my string itself has a percentage or an underscore character. So, the rule followed is you will need to escape that, with the escape character that you define. This is a style which you have seen in C programming as well.

(Refer Slide Time: 09:29)



Patterns are certainly case sensitive. So, it depends it will distinguish between upper case as well as lower case, and these are different examples of string matching that you can do where, you can match at this beginning of a string, end of a string, anywhere in the string, specific number of characters in a string and so on. SQL supports concatenation, conversion of lower to upper case and vice versa and different other common string operations, those are available as functions in SQL and can be used for convenience.

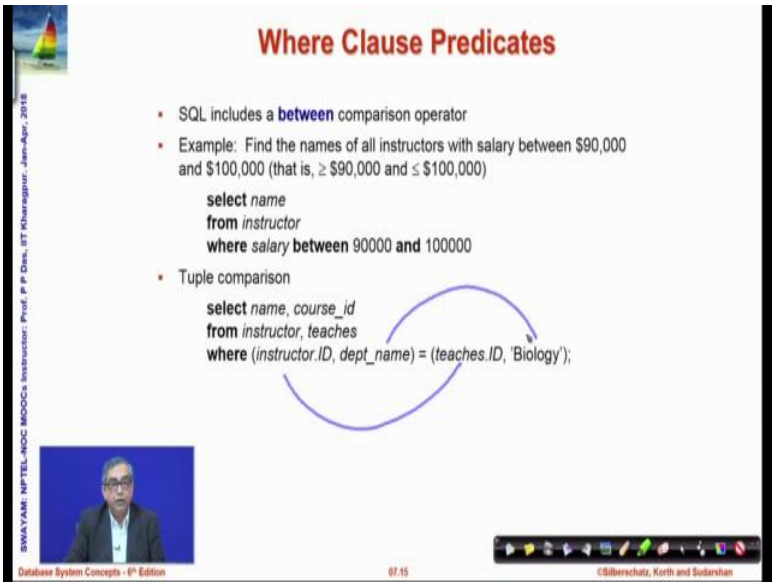
(Refer Slide Time: 10:09)



Now, let us address a different question. let us say, we have computed a query and then often we would want, that the result be ordered in according to certain order particularly the value of certain field if we want the result to be ordered, then SQL allows you to do that by another clause that you add to the query, which is called order by. So, what this will do, we have already seen this query this will find out the names of all the instructors and the names will occur in a distinct manner because, distinct is specified, but then the output will be in terms ordered by the name.

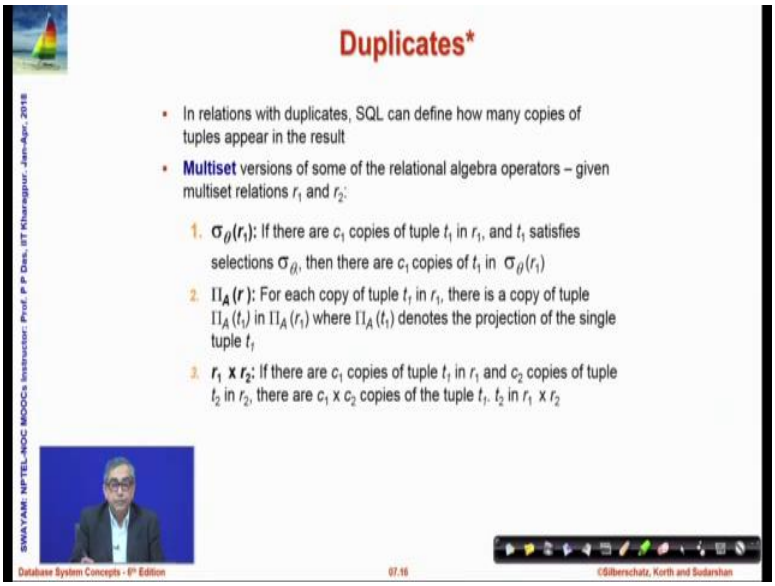
And the ordering can be descending or ascending, you can control that, by specifying whether you want descending or ascending by default the ordering is ascending. So, that makes the presentation of the result of and very easy and you can certainly sort on multiple fields as well, so it can be ordered based on combination of fields.

(Refer Slide Time: 11:15)



SQL where clause also allows between as a comparison parameter. So, between can specify 2 values. So that, whenever the field value will be between these 2 given values the condition will be predicated will be taken to be true otherwise is taken to be false. You can compare based on people as well. So, in this case you could have written, you could have checked for equality of instructor.ID with teachers.ID and department name with the literal biology, but you can compact it by writing a tuple notation as is shown here. So, these are common convenient ways of writing different where clause.

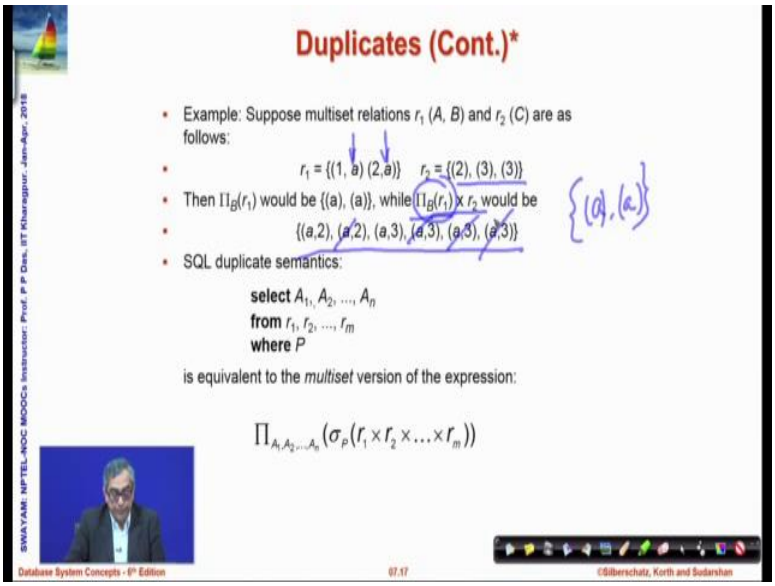
(Refer Slide Time: 12:02)



Now, we have specified that SQL does carry duplicates. So, unlike relational algebra which set theoretically specify that, their duplicates should not be there if in SQL there could be duplicate entries in the same relation. So, there is a; this is called when duplicates are allowed in set theory, then such sets where duplicates are allowed and known as multi sets. So, there are multi set versions of the SQL queries or so to say the relational algebra operations.

So, you have a selection, which can be multi set selection, which means that, if there are certain c1 number of copies of a tuple in the relation, which satisfy the condition theta then all of them will feature in the result. And all those copies can be seen simultaneously, because it is a multi-set condition, similar definitions are hold for projection, as well as for Cartesian product. So, I will leave it to you to go through the details and convince yourself that, these multi set relations really extend the traditional single set distinct definition of the relational algebra.

(Refer Slide Time: 13:27)



So, here is an example where, there are 2 multi set relations as you can see particularly this one, which has identical duplicate entries. So, using that you can define a projection of r1 on B, which will certainly give you it is you are doing projection on B. So, it will give you A only. So, you will have this result itself will be a multi set because you will get 2 as. So, this result will be like a a, and then you have r 2 with which you are doing the Cartesian product. So, you will have all possible combinations all these 6 are the result in the SQL whereas, in set theory typically the result should have been only these 2 tuples.

(Refer Slide Time: 14:38)

(Refer Slide Time: 14:43)

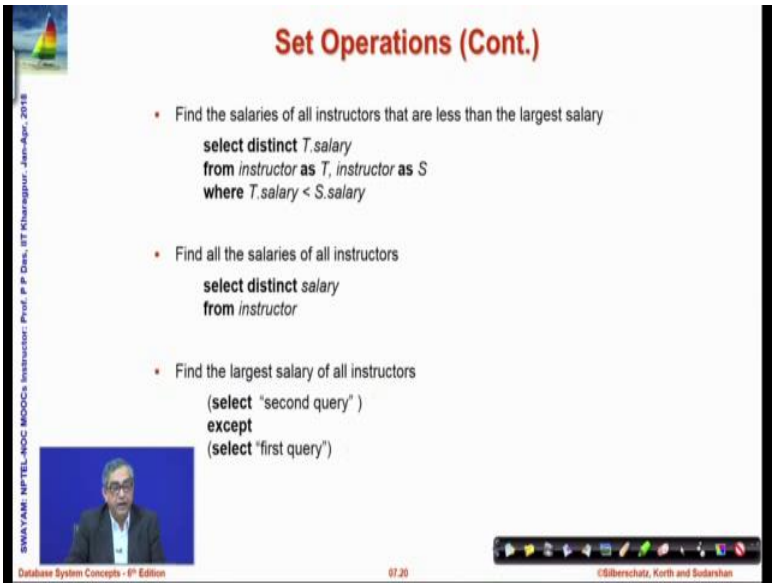
Now, we take a quick look into the common set operations. So, it is possible to do union, intersection, difference kind of operations very easily with SQL queries. So, suppose we want to find all courses, that ran in fall 2009 or in spring 2010. So, certainly the first part of the query is simple. This will give you all courses that run. So, you are taking out the course_id from section is where, the course running information is provided and you part putting 2 conditions, which say that they actually this courses ran in fall 2009.

So, this is the first query the second query says the courses, that run in spring 2010 and you are you have an or condition in the statement of what you are looking for. So, you do a union, union is another keyword. So, this will simply give you a relation of the course_id attribute as the only attribute, which has records from the first as well as the second query. Similarly, you can find out the courses that run, both in fall 2009 and spring 2010 by using intersect, which basically give you the intersection of the result of the first and the second query.

You could also do difference set difference by doing find courses, that ran in fall 2009, but not in 2010. So, what will that mean, that will mean that the result of the first query from the results of the second query be subtracted be done a difference from. So, those that, had run in the fall 2009 and then was again run in spring 2010 will get removed. So, that is done through the accept keyword.

So, in this way you can very easily do set operations, whenever that is easy to conceive; obviously, you can write these queries in several other different forms, but this is just to show you how set theoretic operations can be easily written.

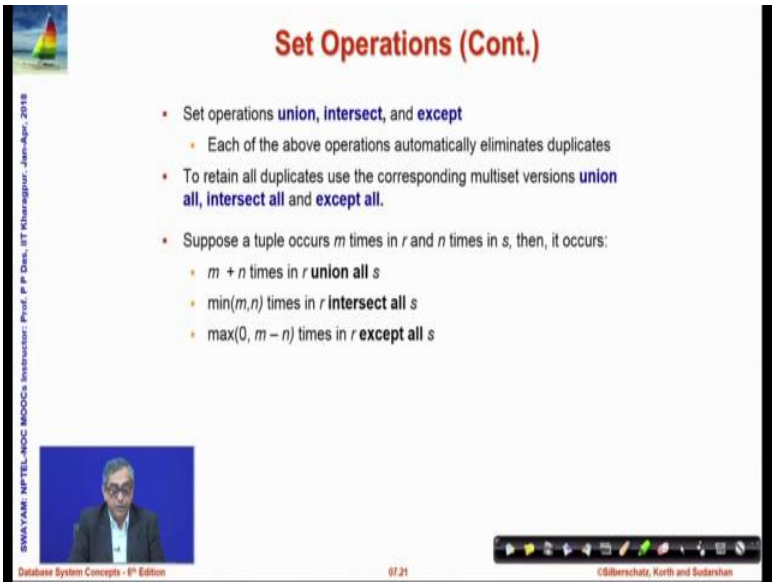
(Refer Slide Time: 17:00)



You can do set operations like this. in terms of fine salaries of all instructions that are less than a largest salary. So, again we are using renaming to think of the same relation as 2, and then as if from the relation T you are trying to look at relation S and finding out what are the salaries, which are smaller than that and certainly whatever comes out is the one which is not the largest because, certainly the largest will not satisfy this particular condition, because it will get compared with itself. You can find salaries of all instructors, and then you can find the largest salary.

So, this is all salaries which are less than largest, this is all salaries including the largest. So, what happens, if you subtract that is from this if you subtract this from all salaries, if you remove the salaries, that are not largest naturally what you get is a largest salary. So, this is a interesting way to find the largest salary we will see later on that there could be several other ways particularly the use of aggregate function, which make these computations easier to perform, but these are the typical ways to use set theoretic operations.

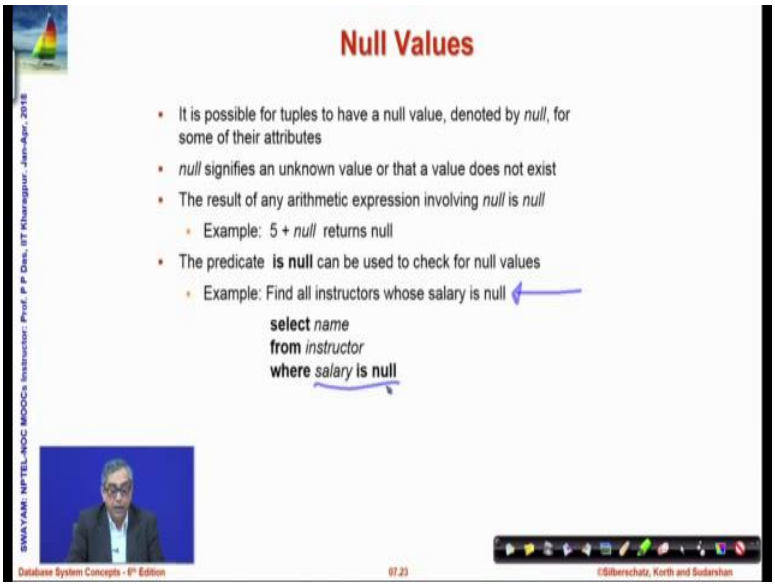
(Refer Slide Time: 18:29)



The set operations, so we have seen 3 of them union, intersect and except they automatically these operations are set theoretic. So, each of them automatically, eliminate the duplicate, unlike what SQL by default, SQL by default does what, allows duplicates, but set operations will eliminate duplicates, because they are set operations. So, if you want the SQL type of behaviour, if you want the duplicates to be preserved, then you can have a multi set version of these set operations, which are known as union all, intersect all, except all like that.

And naturally if you do these operations then, here is the simple formula of the number of tuples, that will get computed in different cases. you can study and convince yourself that these are the correct numbers. Let us, go to the treatment of we talked about null values, that we said that it is possible that certain records in a relation may have one or more attributes where, the value is not known and to represent, that the value is not known we are putting a placeholder called null.

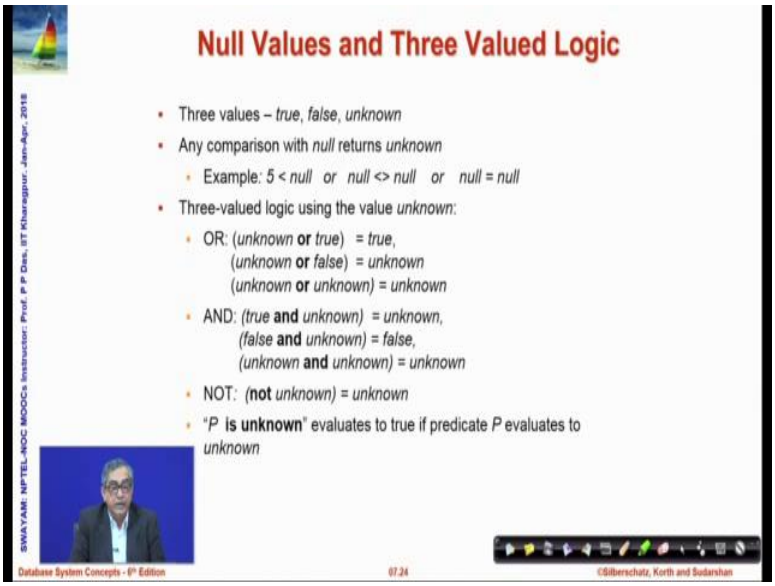
(Refer Slide Time: 19:52)



So, let us see what is the consequence of that null value in terms of doing these query operations. So, the null signifies an unknown value. So, if I do 5 plus null then naturally the result is null. So, what you are saying that I am adding an unknown quantity to 5. So, then what would you say is the result is unknown. So, that is the basic semantics of adding null to a number. So, it is possible to check, if particularly a field is a null for a record, and that is done by a predicate “is null”.

So, in this particular query we are trying to find all instructors, whose salary is null that is not known. So, this is a predicate. So, for a particular record for which salaries null, this will become true and that will get included in the result, but for all records for which, there is some value for the salary. So, salary is known it is not null those will not get included in the result.

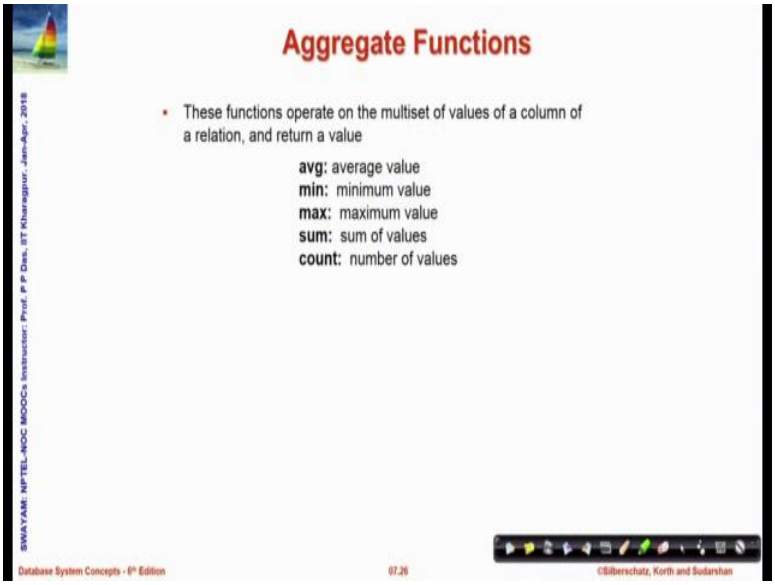
(Refer Slide Time: 20:58)



So, the basic semantics of null is then, combined with the truth values because, we know our basic predicate logic is 2 values true and false. But now, you have a third value unknown that is, you may not know the value of a predicate. So, how does it play around with the true and false values, you can reason through that quite easily if you are comparing with a null in whatever way, then naturally the result is unknown. So, it returns a null, if you are doing any connectives for example, if you are doing or of null or true, then the result should be true because, in or we say that if any of the components is true then the result is true.

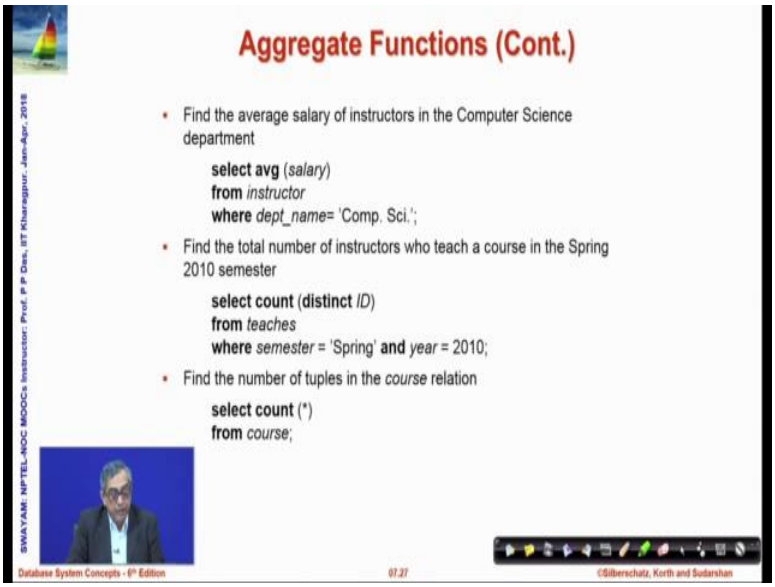
So, here you do not need to know what is that unknown well you can say it is true, but if you do or with false or of unknown with false the second row or of unknown with false if you do this, then naturally this is unknown because, since this is false the result would be true only if unknown value is true and the result would be false if the unknown value is false, you do not know what that unknown value is. So, you have to say that you have result is unknown. So, using that same logic you could see verify, I would ask you to verify offline at home you please verify, that all these combinations of true false with unknown are valid. So, if P is unknown as a predicate will evaluate to true if P is not known..

(Refer Slide Time: 23:05)



Now, we come to the aggregate functions, there are several aggregate functions they can be used for convenience and these are the common months, that operate on the multi set values naturally aggregate functions operate on a particular column they try to aggregate in a particular column and return a single value for example, average would be meaning, that you are trying to find average of the values of a particular column.

(Refer Slide Time: 23:28)



So, here is an example. So, we are trying to find the average salary of instructors in computer science department. So, naturally what you output is average salary. So, mind you this will output relation will have one attribute, which is average salary and since, average salary is a single quantity it will only have one, and here I have made use of this

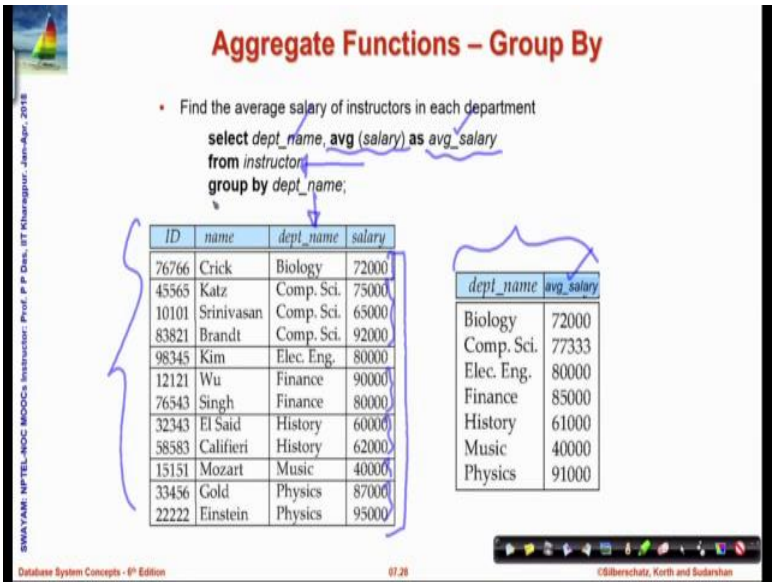
aggregate function average. So, it says you do average on the attribute salary and where do you get that attribute, from you get it from the table instructor and then we are saying that we are not interested to find average of salary of all instructors. We are interested to find the average salary of those instructors who work for computer science. So, you put this back clause. So, this will ensure, that you find the average salary of instructors in computer science department.

So, in similar way you can use other aggregate functions LIKE, if you want to know the total number of instructors who teach a course in this semester. So, you first put the where clause, naturally you where will you find this information you will find this information in teachers, teachers is the relation which tells you which instructor is teaching what course. So, that comes in from, then you have to specify that teaching a course in spring 2010 semester. So, the where clause specifies, that the semester is spring and the year is 2010. So, this will give you all records, which show that some instructor is teaching the course in spring 2010 semester.

Now, naturally there could be multiple the same instructor could happen multiple times, because an instructor may be teaching more than one course. So, you make the instructor.ID, instructor.ID that you have here, you make that distinct. So, that you get only those instructors, every instructor who is teaching one course or more than one course will feature only once in this total list, and then you simply count it. use aggregate function count on that.

So, that will tell you how many instructors are teaching some course in spring 2010 mind you, this is critical to use this key word distinct, because unless you use that, then all that you will eventually find out is not the number of instructors who are teaching the course you will find out the number of courses, that are being offered in spring 2010 because, there could be the same instructor teaching more than one course. If you just want to count the number of tuples you can do count on star. because, what is star is all the attributes. So, from if we want to find out the number of course, you suggest to count * on course.

(Refer Slide Time: 27:03)



So, this is showing you the computation of average salary of instructors in each department.

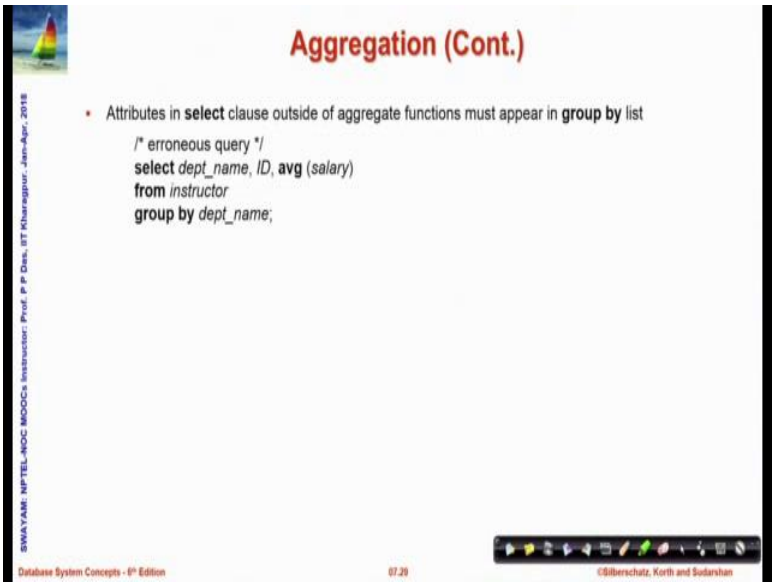
So now, what do you want to do is earlier you try to find out the average salary in one department. Now, you want that for all the departments for each department I want. So, my result now, is not a single row, it is not a single value it is a pair where, I show the department and the average salary in that department. So, this is what I want visible and this is what I have.

So, naturally the information comes from instructor, that is from what I want is a department name and the average salary and I want to give it a nice name avg_salary. So, I have done a rename. So, I get avg_salary here, but then what I want is I do not want an average d1 over this whole set of fields. I want separate average to be done here, to be done here, to be done on this to be on this. So, these are these are basically groupings by the department as you can see, that this particular relation has been sorted according to the department name.

So, when I want to do apply an aggregate function on certain subgroups of records, I use this particular clause GROUP BY, and use a name of a field. So, what it does is if the values in the GROUP BY field in this case department name are identical those records are put together and over those records and average is computed. So, the average, that is computed over these records are put in here, average that is computed in terms of these

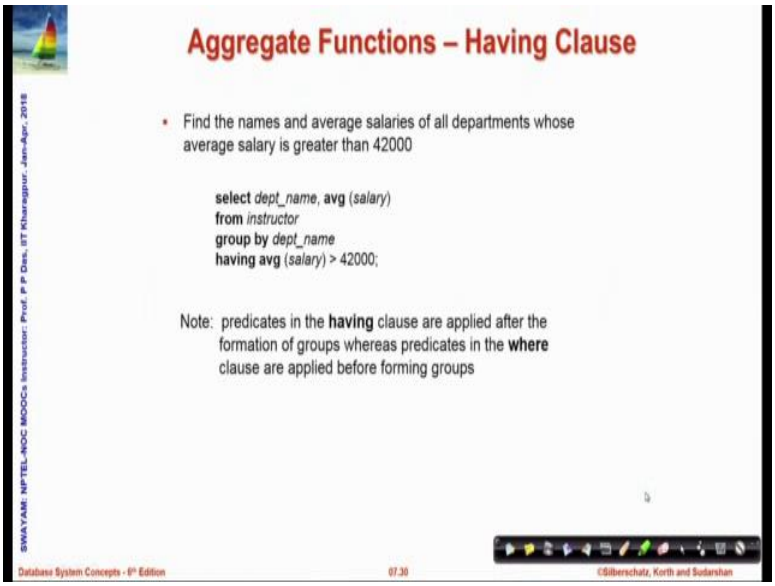
records are put in here, there is only one record, so average that is computed in terms of that is put in here. So, GROUP BY is a very useful feature along with the aggregation functions and it allows you to club information based on certain attribute and then compute the aggregation on some other field.

(Refer Slide Time: 29:27)



Mind you, you will have to do GROUP BY and create the result id result table you will have to make sure that, all your result table attribute are used in the GROUP BY, which is not an aggregate function. So, here ID is not used. So, this is not a query, that SQL would support.

(Refer Slide Time: 29:53)

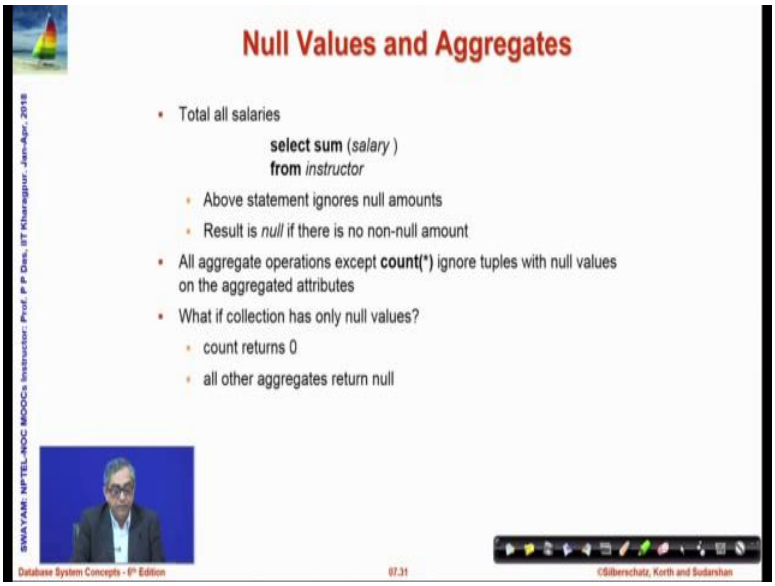


You can further refine your result is saying that, fine names and average salary of all departments; this much you have already done.

Now, you are qualifying that, whose average salary is greater than 42000. So, of all that we have for example, if we look in here for example, in this music department average salary is less than 42000. So, you do not want that in the result you want only those where, the average salary is greater than 42000 and the way to do that is to add another clause called HAVING they say that, the average salary is greater than 42000. So, you are adding another predicate for actually qualifying the aggregated value. Now, the HAVING clause actually applies after along with the GROUP BY because, naturally the HAVING relates to the grouping.

So, once the grouping has happened groups have been formed then the HAVING clause will be evaluated on that, in contrast where clause also has a predicate, but the where clause is applied before forming the groups. So, this point this note has to be understood carefully because, if you have a where clause to choose the records they it will first apply, then out of those records chosen the grouping will happen and once the grouping has happened, then the aggregate function will evaluate and the HAVING clause will get evaluated, the predicate of HAVING clause will get evaluated.

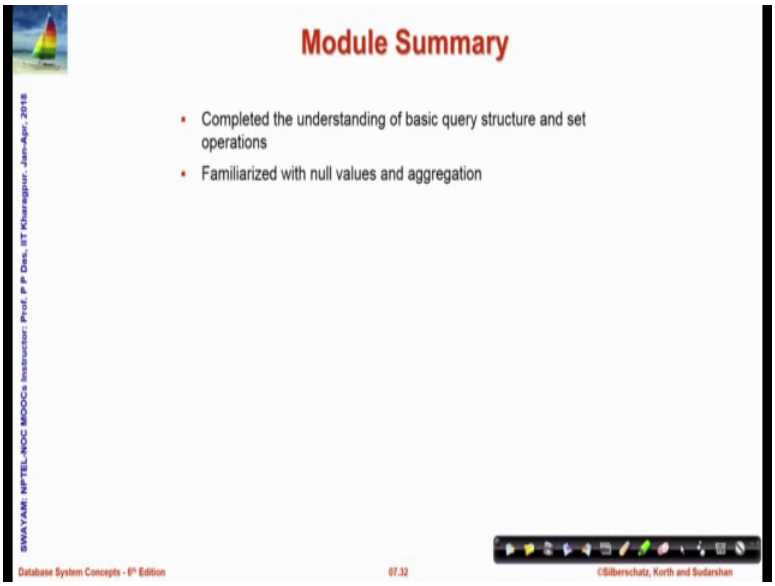
(Refer Slide Time: 31:45)



Certainly, if there are null values in terms of aggregates, then there is a question of what will happen. So, the general strategy is that, whenever you perform aggregation then the null values are all ignored. So, if on that field there is no value which is not null, that is if all values are null then the result is null otherwise the result is computed by ignoring the null values.

So, these are what do you have of course, if you count then, if the collection has only null values the count will return you 0, but all other aggregates will return you simply null.

(Refer Slide Time: 32:29)



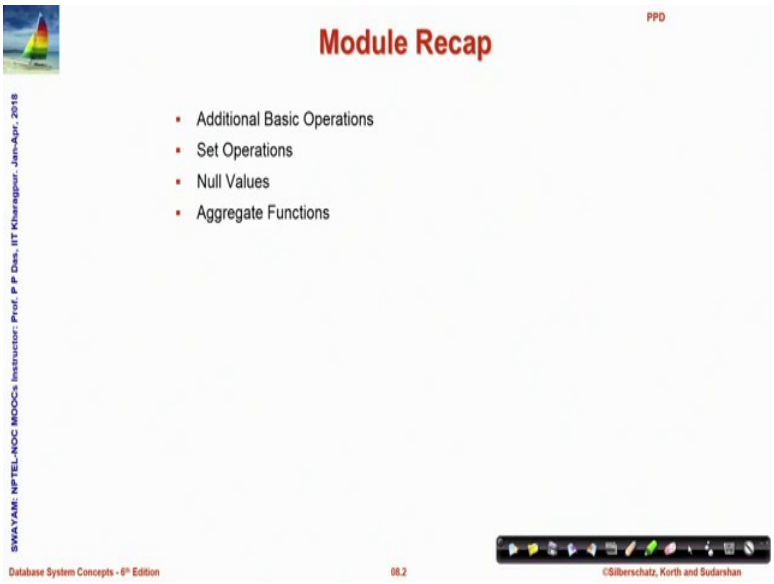
So, to summarize we have started the basic understanding of the basics query structure in the last module. Now, we have completed that with some more additional operations, we have understood the set theoretic operations and very importantly we have familiarized with the treatment of null values and aggregation functions particularly the GROUP BY and HAVING clauses and how do null values and aggregation interact in terms of an SQL query.

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

Lecture - 08
Introduction to SQL/3

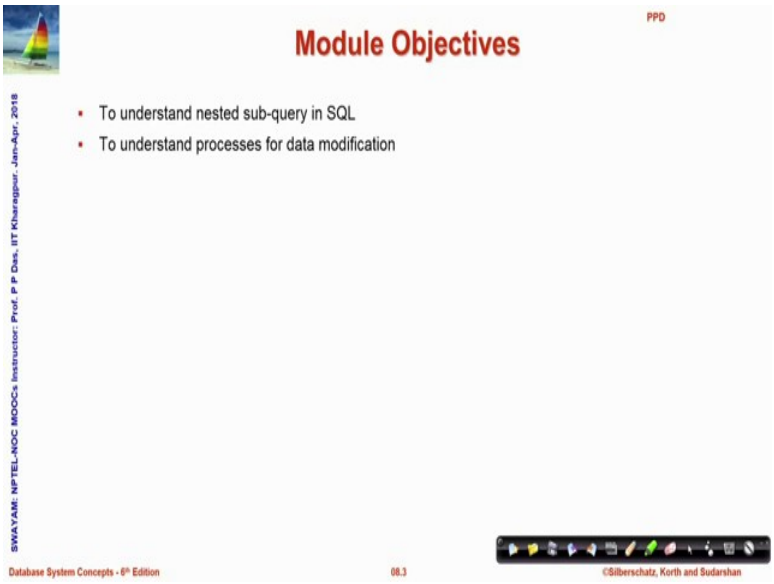
Welcome to module 8 of Database Management Systems. We have been discussing basic SQL queries and this is the third and closing part of that introductory discussion that we started in the 6th module.

(Refer Slide Time: 00:34)



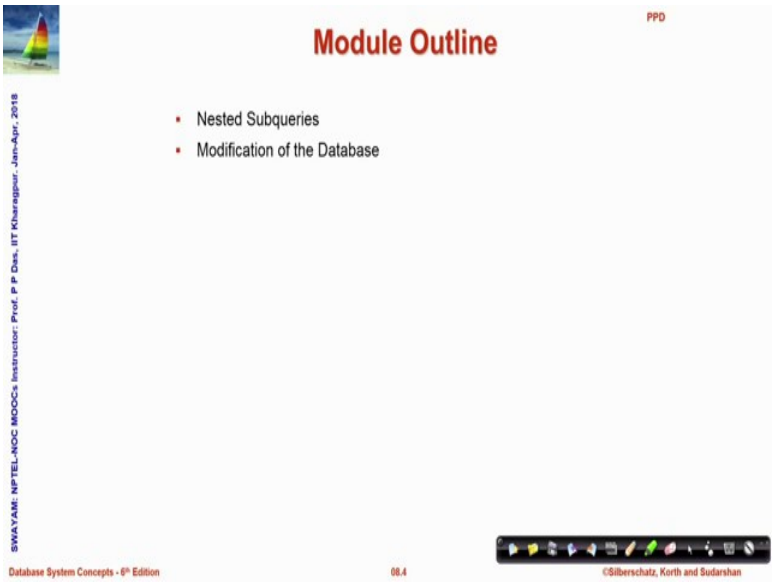
So, just to quickly recap, this is these are the things that, we did in the last module completing the understanding of basic operations the null values and aggregate functions.

(Refer Slide Time: 00:44)



In the current module, we want to understand a feature which is very very important in SQL query forming it is called the nested query or more formally nested sub query. In SQL and we would like to understand the process of data modification.

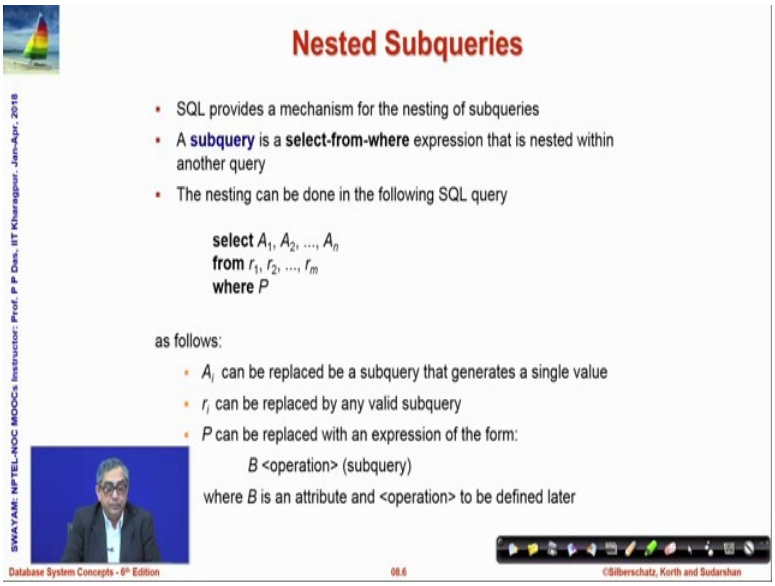
(Refer Slide Time: 01:05)



And those are the two things that are outlined here.

So, let us start with nested sub queries.

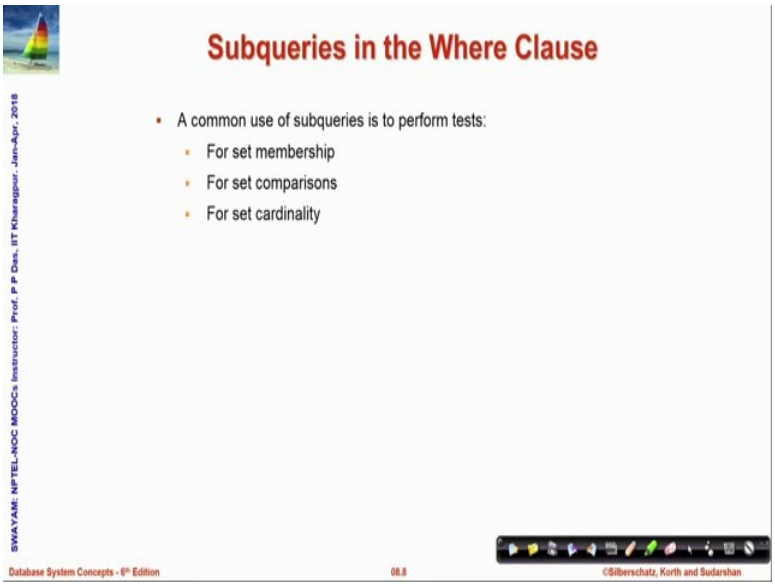
(Refer Slide Time: 01:12)



A sub query is necessarily a SELECT FROM WHERE expression that is nested within another query, this is. So, these are the key things where expression. So, it is nothing new over; what we have already learned? But it is a part of another query it is nested within another query and that is the reason it is called a sub query. So, it by itself is not the result. this will be used, this is a SELECT FROM WHERE expression that will be used in a nested form in another some other queries to actually generate the result.

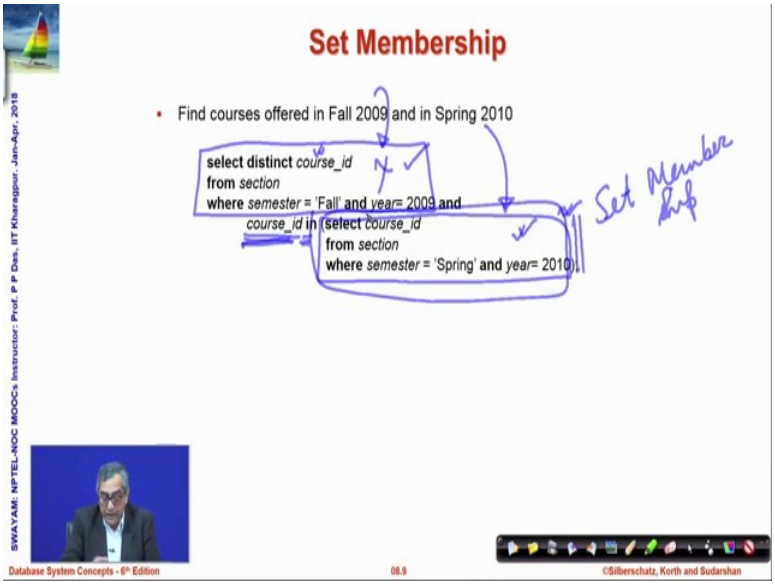
So, that is a nested sub query. So, the nesting can be done in one or more of the three clauses that a SELECT FROM WHERE SQL query has. An attribute can be replaced a relation can be any valid sub query or a sub query could form the part of a predicate in the where clause all of these are possible. So, we will discuss them one by one. So, first we will start by discussing how sub queries work in the where clause.

(Refer Slide Time: 02:44)



The common use for you having sub queries in the where clause is to perform different kind of tests for membership comparison cardinality and so, on.

(Refer Slide Time: 02:58)



So, let us look at this; you have already seen this query before find courses offered in fall 2009, and in fall in spring 2010. Earlier we have shown different ways of coding this, now we are showing yet another way to be able to code this in SQL.

So, first from the beginning certainly we need the courses. So, the SELECT clause is trivial it has to be the distinct course id is certainly the information will come from

section which has information about offering of courses as we have also seen already. So, those two are no brainer. Now let us see what how do I find courses that are offered in fall 2009 and in spring 2010. So, again the first part, the courses offered in fall 2009 is coded in this part; in part of that where clause predicate when he says semester has to be fall and here is 2009. So, this part is also done.

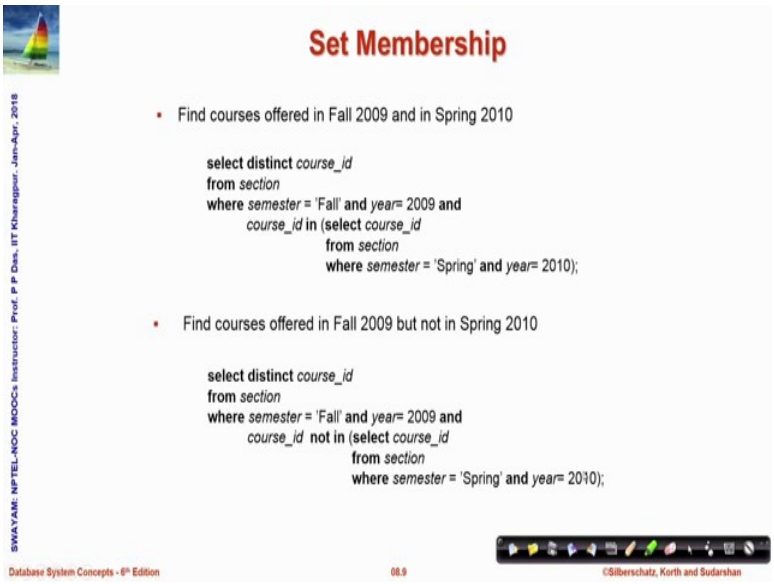
Now, we need to ensure that whatever I mean, if I assume that after this this part were not there, then this will only give me courses which are offered in fall 2009, but we want the courses that are in fall 2009 and in spring 2010. So, we do something interesting what I do is, we write a separate query here which is courses that are offered in spring 2010. SELECT course_Id just in the same way, SELECT course_Id, section, semester year. So, this particular query will give me the courses offered in spring 2009. So, what do we have in one part? I have so, if you look at this part this courses that happened in fall 2009; if we look at this part courses that happened in spring 2010 good.

Now, what I want, I need that the; it I am interested in the courses that happened in both. So, for a course that exists here I want to specify that that course Id must be present here. So, what I am checking for? I am checking for a set membership; this is a set right. So, I am trying to check, whether that course Id which is being selected in the first part exists in, in is another keyword. In this particular, this particular relation that is specified by the second part of this query which is courses offered in spring 2010.

If it is, if the course Id is present, then that course must have been offered in both the semesters if it is not present, then it is offered only in fall 2009, and not in spring 2010 and certainly the courses that were not offered in fall 2009, and only offered in spring 2010 will feature here, but they do not exist here. So, they will never come up for tests. So, as a result what I get finally, is the effect of computing courses that are offered in fall 2009 and in spring 2010. this part of the query which I used as a part of the where clause is my nested sub query.

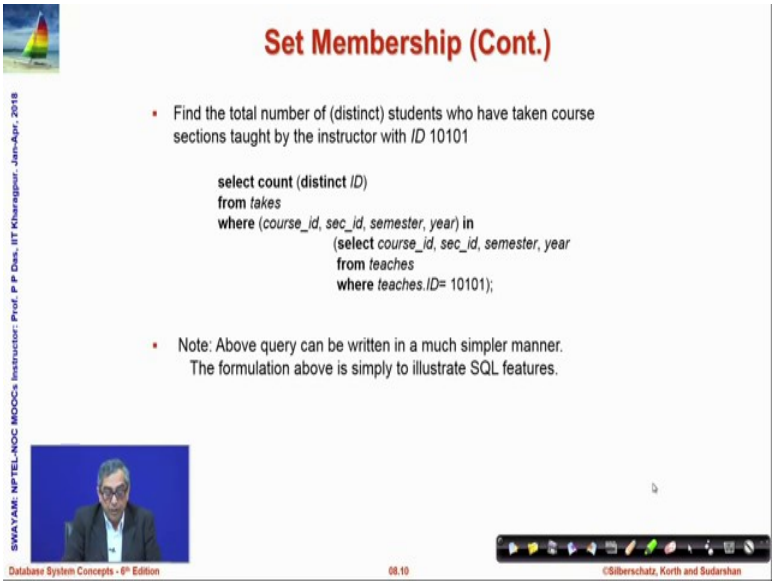
And in this case as we have seen it is used for set membership and this is a basic idea of using nested sub queries; that is a nested sub query will always give you a relation. So, you try to put that relation in the right context of the where clause from clause or SELECT clause, and then make use of it in building up your logical.

(Refer Slide Time: 07:14)



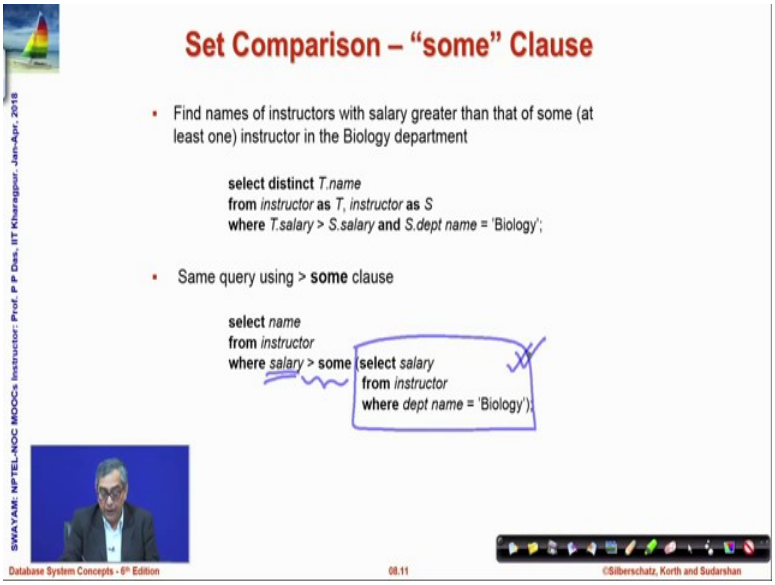
So, let us run through some examples this is, what you are saying is, earlier one was the courses offered in both here we are trying to do kind of the difference saying the courses offered in fall 2009, but not in spring 2010 certainly we easily get that by changing the membership to negation of the membership earlier it was in now you do not in you will simply get that it is up to you to take some examples and convince yourself that this kind of a nesting will work.

(Refer Slide Time: 07:43)



We find the set of the find the total number of distinct students, we have taken the course section taught by the instructor Id, some Id is given. So, again we form a nested query here is a nested query, which tells me the courses taught by this particular teacher 10101, and then we check set membership in terms of this course Id, section Id that is fields of the takes relation to see that, whether that particular tuple can exist in the course offered by the specific teacher; if it does then take out that I d which is which will turn out to be the student Id. In this case because that is the text relation has the student I d take out the student I d and count it as distinct. So, this can simply give you the answer to this squared; obviously, we agree that this can be written in a simpler form also, but we are including it here just for the sake of illustrating the feature.

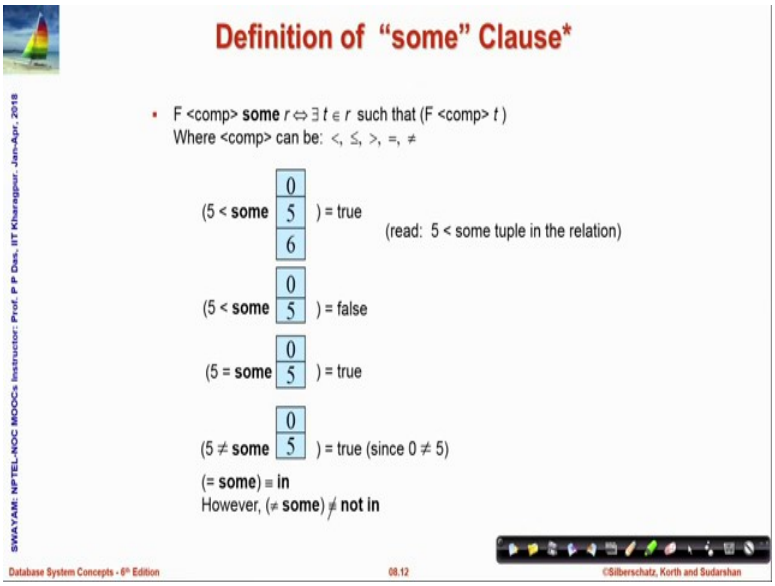
(Refer Slide Time: 09:03)



There is another clause called the some clause look at this fine names of instructor; salary is greater than that of some which means at least one instructor in biology department and we have already seen this coding before.

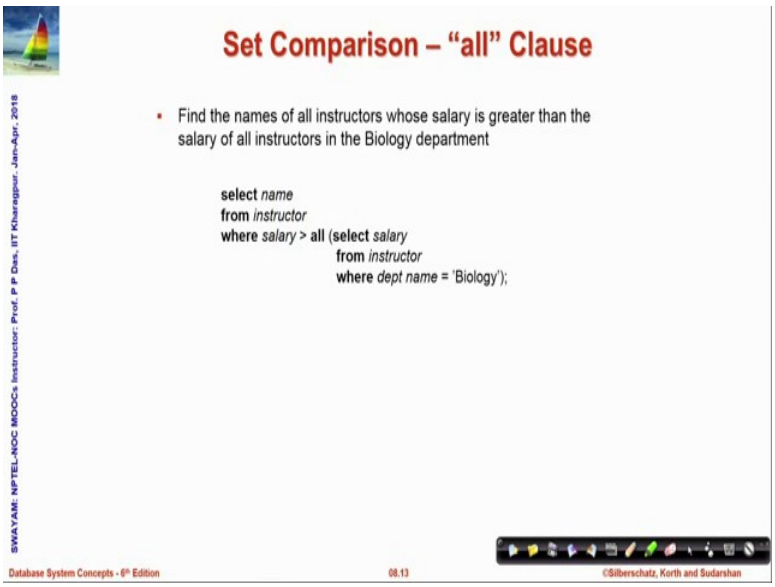
Now, we can do this in terms of the nested query by using, again this is certainly the salary of instructors in biology department and we are doing greater than sum; that means, that the salary here being checked must find at least one record here. So that it is greater than that salary value. So, it is greater than some is a nice way to find existential records using the nested sub query.

(Refer Slide Time: 10:02)



The logic of some clause, I have detailed out here. So, we will not go through each one of them in this discussion.

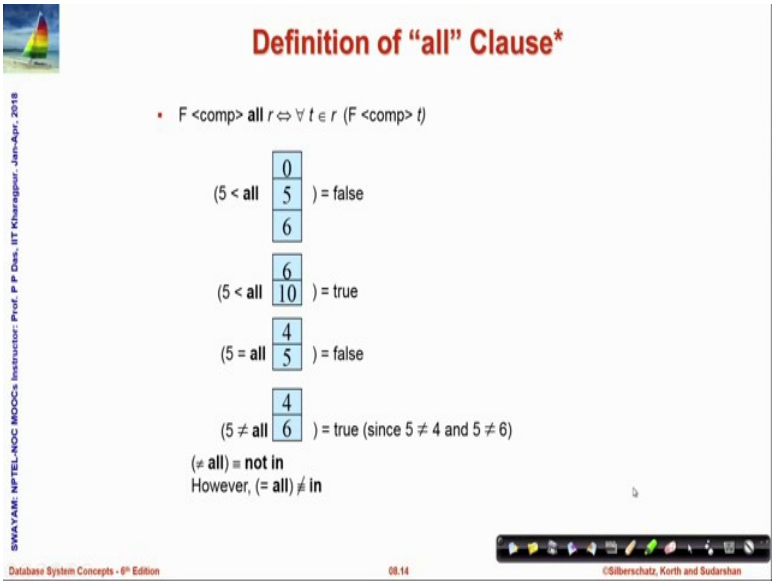
(Refer Slide Time: 10:09)



I leave it unto you to study and convince yourself, that you understand the semantics of some. So, similarly we have an all clause which say that if we want to say, they find the names of all instructors; whose salary is greater than the salary of all instructors in the biology department in case of sum we can write or we will write all and it will check every salary will check with the whole set of salaries in this sub query. And only if that is

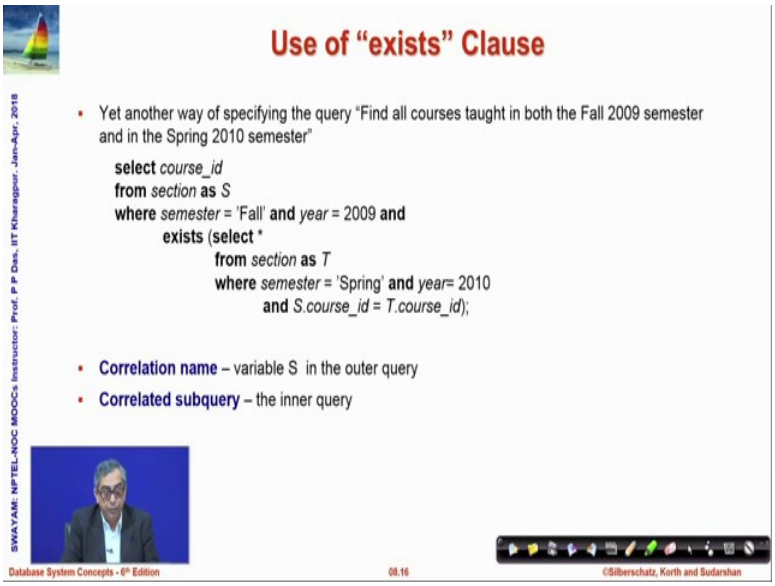
greater than that particular record, that particular name will be included in the result. Otherwise it will be excluded from the result.

(Refer Slide Time: 10:52)



Similar to some there is a basic semantics of all which is also worked out here and I leave that to your study at home.

(Refer Slide Time: 11:03)



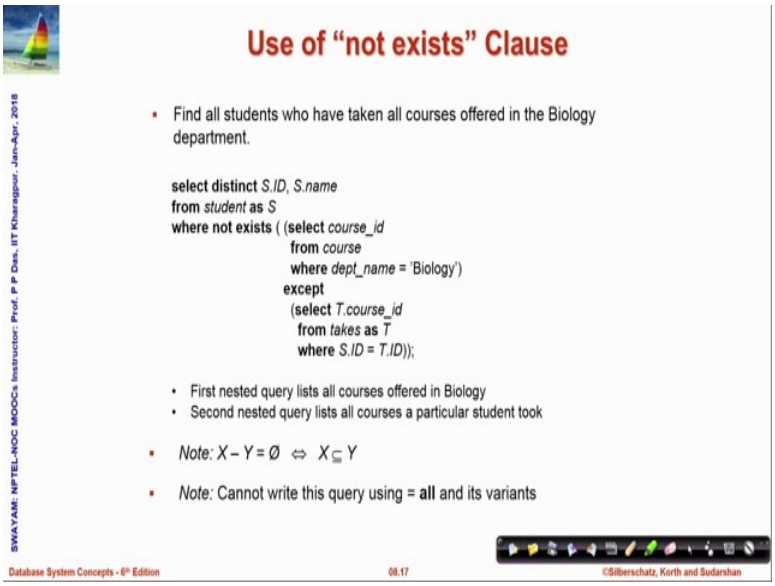
You can test for empty relations by using the exists construct. So, if you say exists r, then that is a predicate which mean that r is not empty; if r is empty then exist is false and not exist is the negation of exist. So, it can be used to specify the query like find all courses

taught both in fall 2009 and spring 2010. So, all that you have to do earlier you did it by set membership.

So, now you are trying to do this by this exists. So, you are saying that this is again the same query which gives you the courses that are in spring 2010 and also in this fall 2009 and you check whether this relation, whether this particular nested query is an empty one or not. if it is an empty one, then exist will fill and the whole where clause will fail, if it was not an empty one then you have found such an entry it was offered and therefore, it will get included.

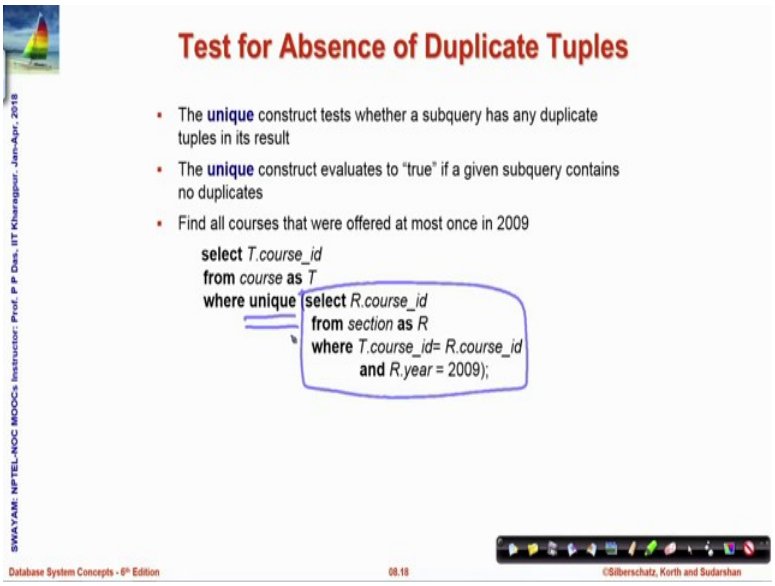
So, these are just different ways of expressing similar things, but what you should note is the nested sub query is a very convenient way to frame up the logic in multiple different ways that you would like to do. So, these are the different names given the correlation name and the correlated sub query. incidentally; the nested query is often referred to as the inner query and the query in which the nesting has happened, is known as the outer query.

(Refer Slide Time: 13:04)



Here is another example which illustrate the use of not exist; so which I leave it for your own study.

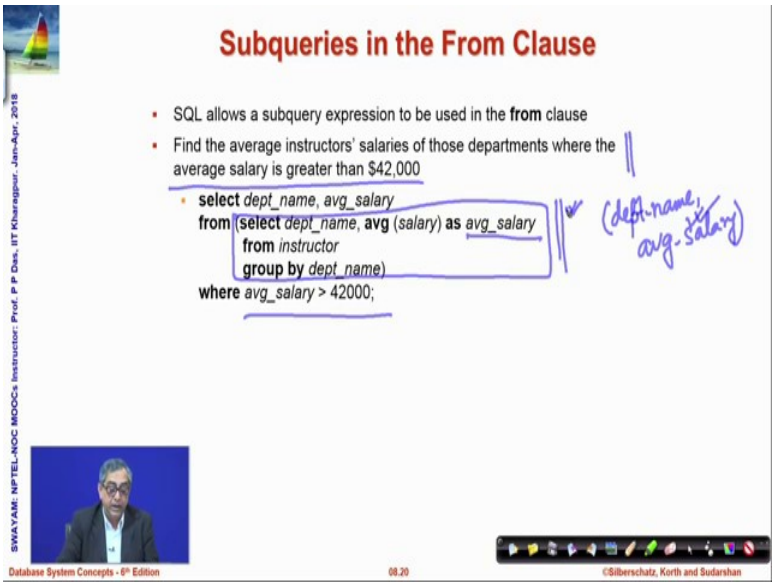
(Refer Slide Time: 13:15)



We can check for uniqueness that is; test for absence of duplicate tuples by using the unique keyword. So, we can you can see here that here is a nested query and using unique to find out all courses that were offered at most once in 2009. So, if it a course was offered more than once, then naturally multiple records will feature and the result the unique will fail. unique will be true only if; there is only one entry which shows that it is offered at most once in that semester.

Now, I move on. So, we have been discussing about sub queries in the where clause; now we move on to sub queries in the from clause.

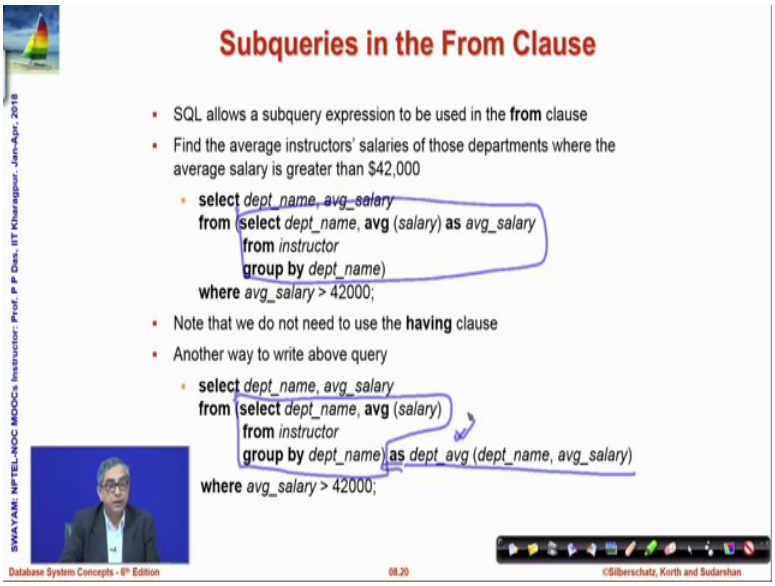
(Refer Slide Time: 14:09)



So, as we have already seen a nested sub query is a relation. So, it can naturally be used in the place of any relation that we have in the from clause. So, we are trying to find out average instructor salaries of those departments where the average salary is greater than 42,000. So, look at. this is a nested sub query. So, where what is been found here this will compute the average salary department wise average salary which we have already seen group by department name and then you do average on the salary and you give it that field a new name.

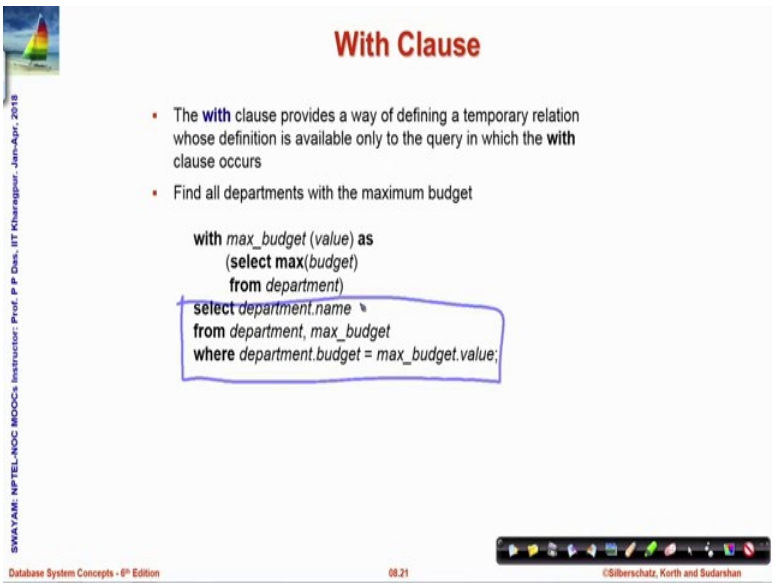
So which means that; this is equivalent to having a relation which has two attributes depth name and avg_salary. So, from that you are now trying to do the selection and what is the condition that the average salary has to be written. So, you already have that as a part of the field the average salary. So, you just need to put that in the where clause and you have only those coming out of this particular relation where average salary is greater than 42,000 to be selected in this SELECT query. So, these will feature in the output of this selection.

(Refer Slide Time: 15:56)



So, that is how you can very easily use a nested sub query in the; from clause and for this we did earlier. We solve this problem using the having clause, but they here we will not need we did not need the having clause to do this there is this is another way. So, here what we have done is the same; if we if we look into the nested sub query. This is actually the same. all that we have done we have given it a new name by the renaming feature, and then this as if becomes a relation and on that the computation is done rest of it is similar.

(Refer Slide Time: 16:50)



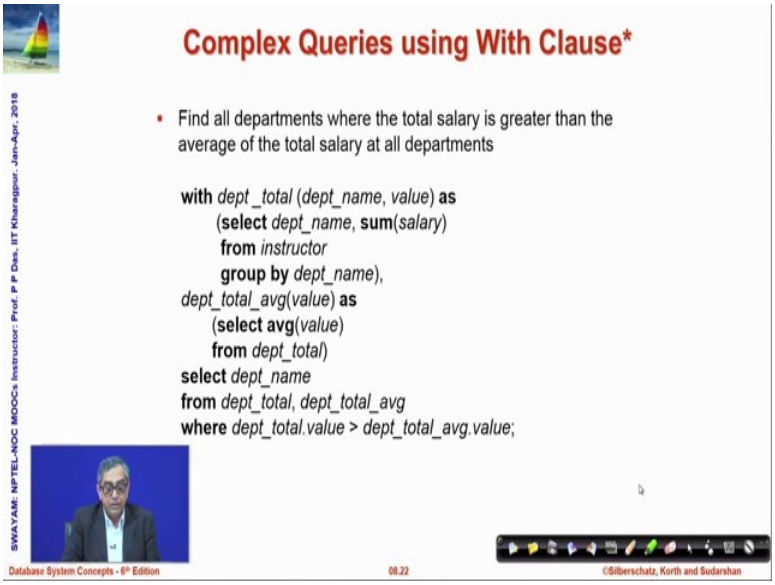
There is a with clause that provides a way of computing a temporary relation and that can be subsequently used.

So, let us look at example. we are trying to find all departments with maximum budget. So, this is my basic query, we want to find department name, department dot name, that is; a departments name from the department table and the budget must be same as maximum budget. So, for that I need to know the maximum budget that exists across the department. So, look at what has been done, here we have a nested query which aggregates the maximum budget from the departments.

So, this gives you the value of the maximum budget. We make that into a temporary table max budget with an attribute value. So, this is renaming. So, you cannot see the renaming is being used in very interesting ways. So, this is my nested query that gives me a relation and this is my definition of the relation. So, max budget now is a temporary relation a relation that I used subsequently in my from clause and with allows me to do that this relation will not be available.

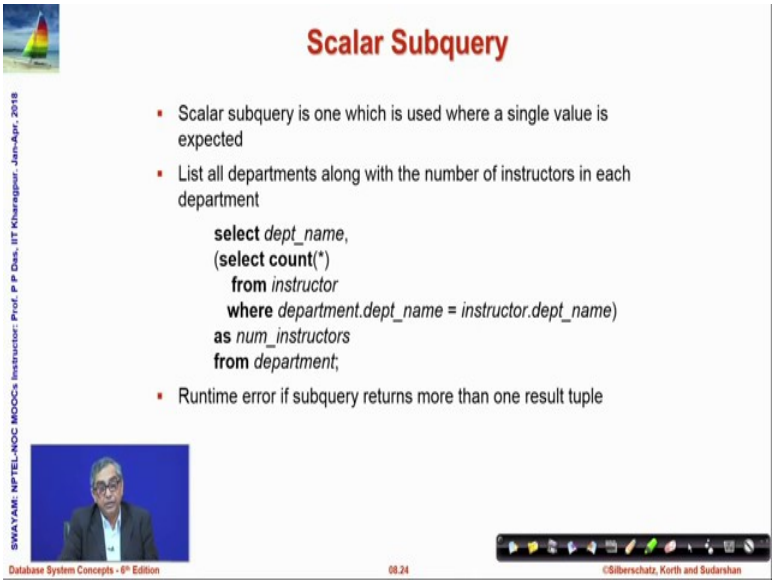
Otherwise after this query this relation will not exist this is just a temporary one computed for this query. So, this gives me the budget value, this gives me the department and department specific budget, and this condition tells me that I can choose all the departments which has the maximum budget very nice way of using this.

(Refer Slide Time: 19:06)



So, with clause can be used in even more involved way again this is an example which is more complex use and I leave it to you to practice study and understand. we move on to sub queries in the SELECT clause finally.

(Refer Slide Time: 19:26)



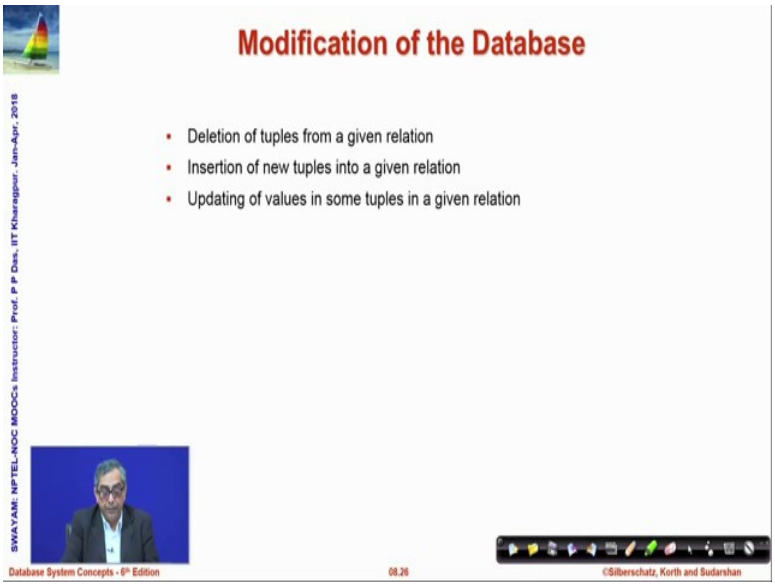
A scalar sub query is one; where there is a single value is expected. So, we can very easily use that in the SELECT. So, what if you look at this part which is the sub query? So, you are saying list all departments along with number of instructors each department has. So, this condition tells that, the from the instructor; we are taking out those that department name where the instructor works to count them and then you form that as a new attribute mind you in while we were using this in the from clause.

We were treating that as a relation because nested query will give a relation, but here in the SELECT clause the entities are attributes. So, this as is renaming of attribute which means, but this is a relation that is why this notion of scalar sub query is required, that is though this is a relation what does the relation compute it computes a single value. So, that value we are putting as an attribute named num instructors.

So, we have department name and the number of instructor, then for each and every department; that we actually have from the department list. So, it is a very interesting way of using this nested sub query in terms of the SELECT clause. naturally; since in the SELECT clause, I cannot have, I mean every entry in the SELECT clause has to be an attribute pure relations are not possible.

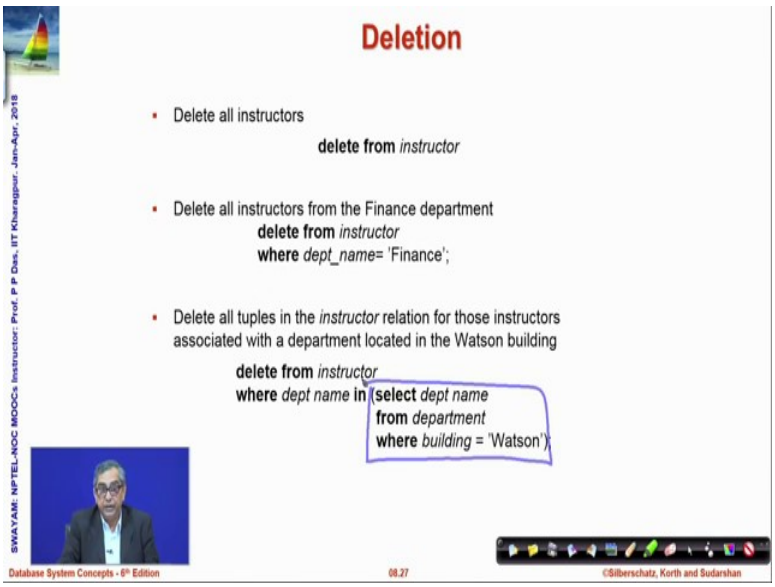
So, if the sub query returns more than one table which cannot be conceived as one or more attributes, then it will be runtime error that will not be allowed; because we do not know how to handle multiple tuples in terms of a select clause. ok. Next we move on to discussing the modifications to the database, how do we modify the database?

(Refer Slide Time: 22:12)



So, we will look into some of the ways of changing the records or removing records from that earlier. We saw a delete of record which removed all records from a relation, but now we will see selective deletion insertion and update of values.

(Refer Slide Time: 22:33)

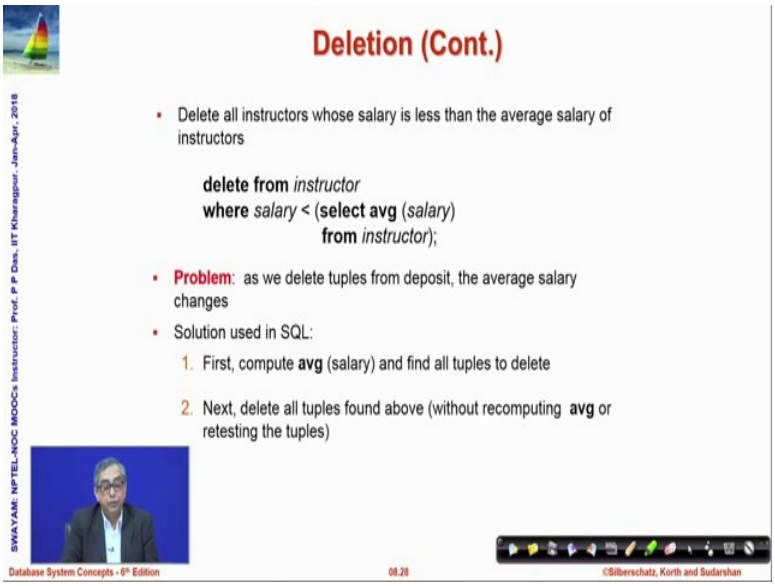


Now, deleting all instructors are easy DELETE FROM instructor all records sorry this and this becomes an empty table, but suppose we want to delete all instructors from the finance department, then like we do in the SELECT FROM WHERE we again use the where clause as a predicate and say that DELETE FROM instructor, but you do the deletion provided this condition is satisfied that is; department name is same as finance. So, it is very similar to the SELECT FROM WHERE, but the effect is unlike SELECT FROM WHERE, where no tables change in the database here.

The table is actually changing; because these instructor records are deleted whose department name was finance. The third example shows delete all tuples in the instructor relation for those instructors, associated with the department located in the Watson building. So, Watson building may have multiple departments. So, all instructors who worked on those departments which are located in the Watson building that will go. So, you do, this is again you are using nested query.

Now, you know how to use a nested query. So, you use nested query which will give you the names of departments, which gives you a relation with a single attribute with names of departments housed in the Watson building, then you use the set membership to check whether a particular department belongs to that set, if it does then it is in Watson building; otherwise, it is not in Watson building if it does belong to the Watson building then this where clause becomes true and the corresponding instructor record is deleted and that is of this whole different kinds of selective deletion can happen.

(Refer Slide Time: 24:34)

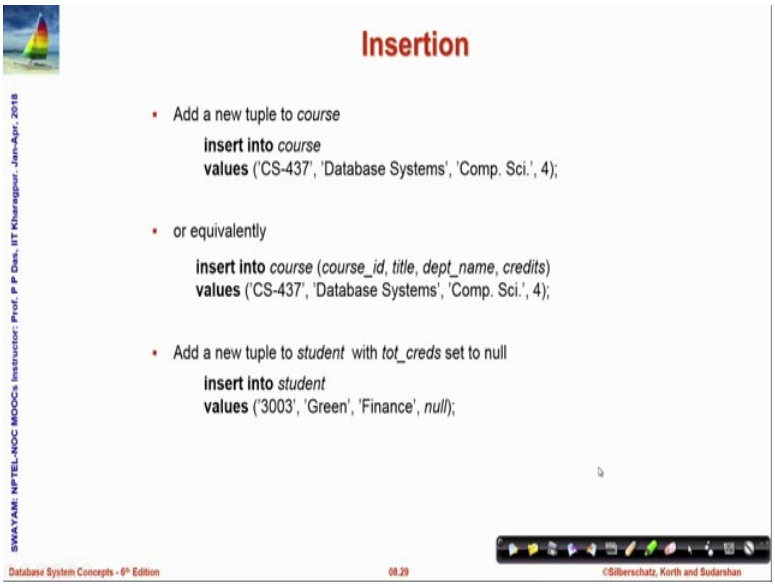


Delete all instructors whose salary is less than the average salary of instructor. Again this is; so, you compute the selection sub query which computes the average salary and check if the salary is less than the average salary and delete that. sounds straightforward, but just wait, just wait, I mean did we do it do a right thing an average salary is computed by taking the sum of all salaries in the relation. And then dividing it by the number of relations this has to be the average salary certainly if I remove a record then the average itself will change.

So, if I write the query in this manner then what I am saying on the face of it looks correct, but then actually can it be correct because the moment a condition is satisfied and that record is deleted this average value itself has changed. So, that is not. So, that will depend then the result will depend on the order in which the deletion is happening, but that is not what was meant what was meant is take all the records for the present find out the average find out all records which have a salary less than that average and remove them.

So, this initially you know easy trivial looking solution is not actually correct. So, you will have to do the solution in two stages: first compute the average salary, find all tuples to delete? Next delete all tuples found above without re-computing. The average in the present solution the average is recomputed, which is the wrong thing.

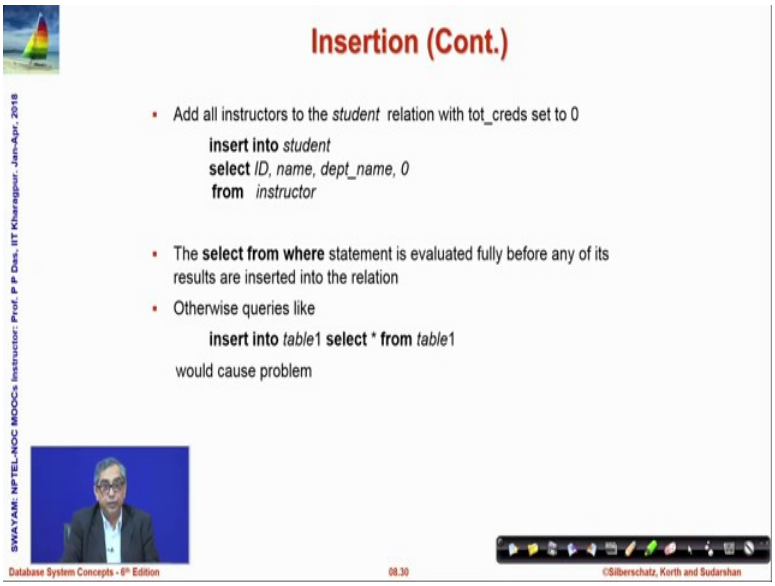
(Refer Slide Time: 26:32)



So, again I will leave that for you to solve. we move on to looking at modifications in terms of insertion. So, we had seen this earlier we can add a new tuple by inserting to then the relation names, and then you save values and the tuple of values. We can specify the; if we if we do not remember the order of attributes in the relation then we can also specify the order in which we are spaced actually giving the information.

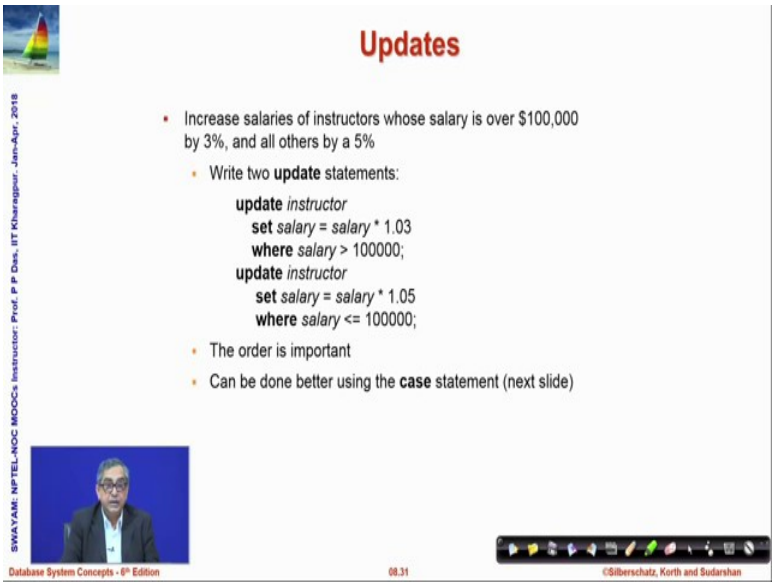
So, you are saying INSERT INTO course and what we have done here is we have actually specified the order in which that tributes occurred and that order and the order of values must be the same. In the first case, this order of values is decided by the order of the attributes that exist in terms of the create table. Add a new tuple to student with total credit set to null that is I do not know; if we were adding a student initially does not have a credit right. The credit is a nullable field the credit will be earned after the student has gone through the courses and all that. So, if I do not know the value of a field then I can write null which is a special value designating unknown for at the time of insertion.

(Refer Slide Time: 28:00)



And all instructors to the student relation with total credit set to 0. So, I can also combine insert with select. So, we are taking the first part this part is selection which generates a whole lot of records having ID, name, department name and the total credit set to 0. From the instructor and insert them into the students. So, these will get inserted into the students SELECT FROM WHERE statement is evaluated fully. So, this first select from will be done before any of its results are inserted in the relation. So, that is the basic condition that SQL guarantees; because, if that were not the case then such situations will become circular and will cause problem.

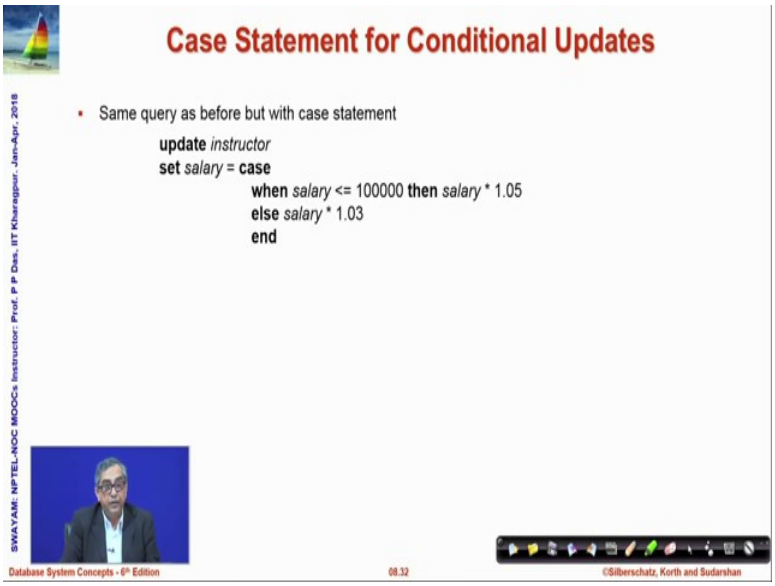
(Refer Slide Time: 29:03)



Updates can be done based on particular values. So, you can update a table and what it means that? It you could update the values of specific fields.

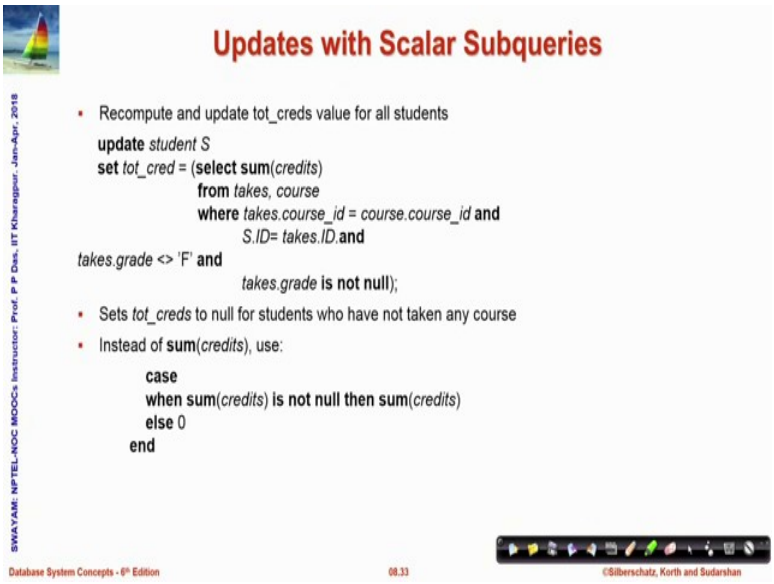
So, here in the in the first case; we are giving trying to give a 3 percent salary raise for salaries which are more than 100,000 and some 5 percent raise for salaries which are less than equal to 100,000 and mind you this order in which you do the update is important, because if you do the later update first then someone who was what qualified in the later part was less than 100,000 with the increase will become more than 100,000 and will also qualify for the second one. So, that will become wrong. So, update often is dependent on the order.

(Refer Slide Time: 29:58)



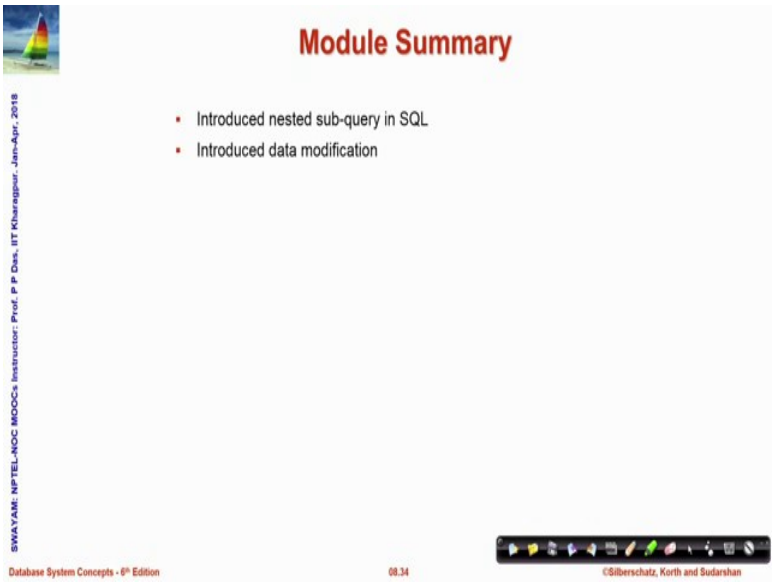
And therefore, you have yet another ah feature to take care of this when you have a specific order to do things it is called the case. So, you say when salary case is a new keyword, when is a key word when salary is less than equal to 100,000, then this is how you hike otherwise this is how you hike. So

(Refer Slide Time: 30:28)



It can it looks more like the; if statement of C, C++ you can do updates with scalar sub queries. We have seen scalar sub queries already. So, you can use a scalar sub query, again I would not go through that details please study and you will be able to understand.

(Refer Slide Time: 30:50)



So, these are different examples. So to summarize, we have introduced a very powerful feature in SQL query known as nested sub query, where we can write a SELECT FROM WHERE expression as a part of the where predicate or as a relation in the from clause or as one or more collection of attributes in the select clause and it can be used in several

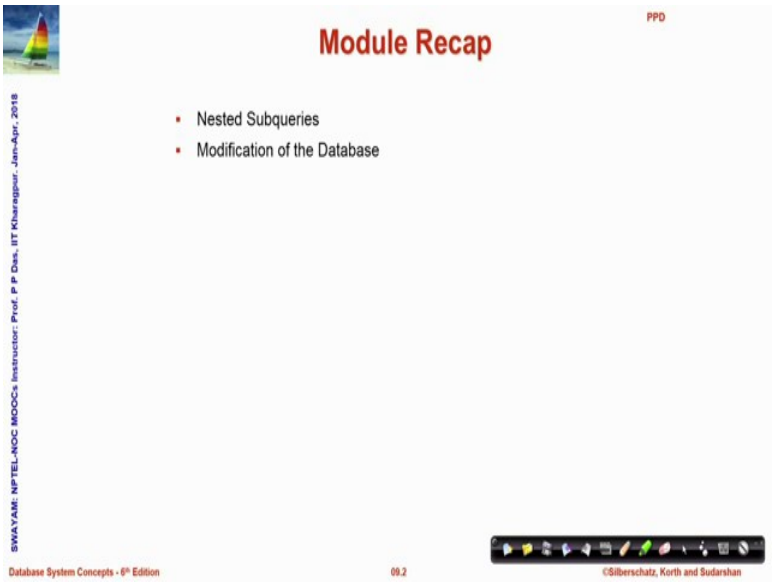
other places also. We have seen the ways to perform data modification in terms of deleting inserting and updating records. And we have also seen how nested sub query often may be very useful not only in terms of performing a query, but also in terms of performing certain data modifications.

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

Lecture - 09
Intermediate SQL/1

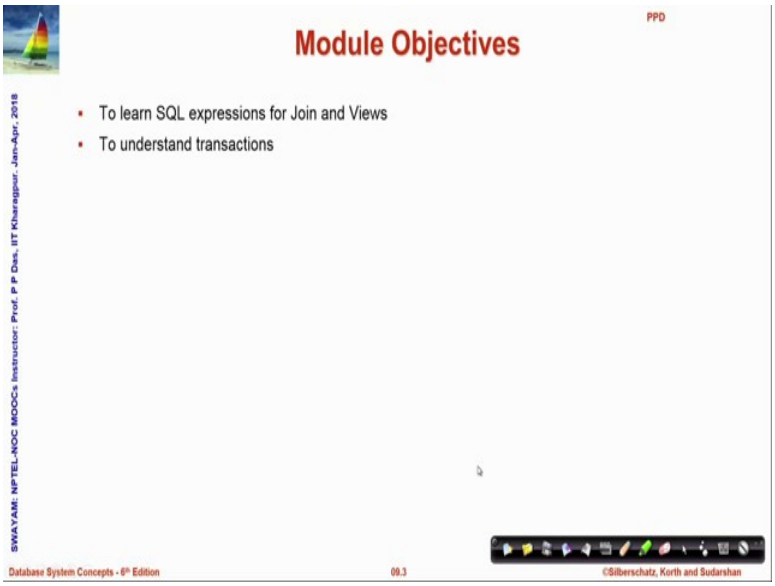
Welcome to module 9 of database management systems. We have discussed about the introductory level of SQL the structured query language. in this module and the next we will take up some more intermediate level features of SQL. So, these modules are called intermediate SQL.

(Refer Slide Time: 00:42)



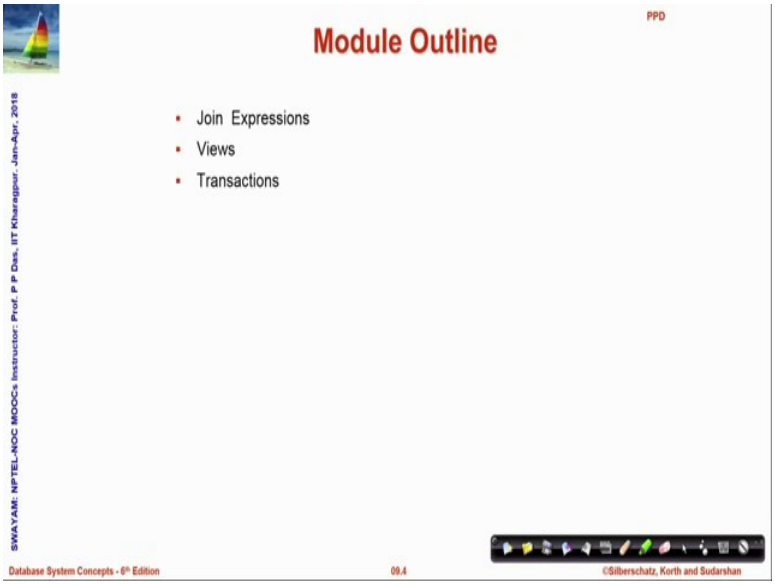
So, in the last module this is what we have done which was part of the closing part of the introductory SQL. The nested sub queries and modifications to the database.

(Refer Slide Time: 00:55)



Today, we will, in this module learn about SQL expressions for join and views and we will take a quick look into understanding the transaction.

(Refer Slide Time: 01:11)



This is the module outline as it will span.

(Refer Slide Time: 01:15)

So, we start with the join expressions in SQL join as we have already introduced.

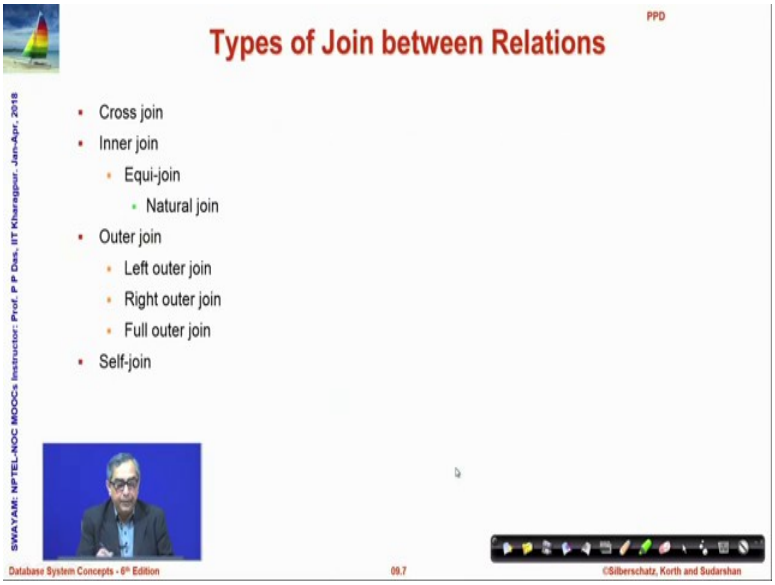
(Refer Slide Time: 01:21)

Takes two relations and returns a result as another relation. So, it is two different instances of two schemas and we try to connect them according to certain properties.

So, a join operation, is primarily a Cartesian product, which relates tuples in two relations under certain conditions of match. It also specifies that after the joining has been done, what are the tuples, which will be present in the output join. So, join

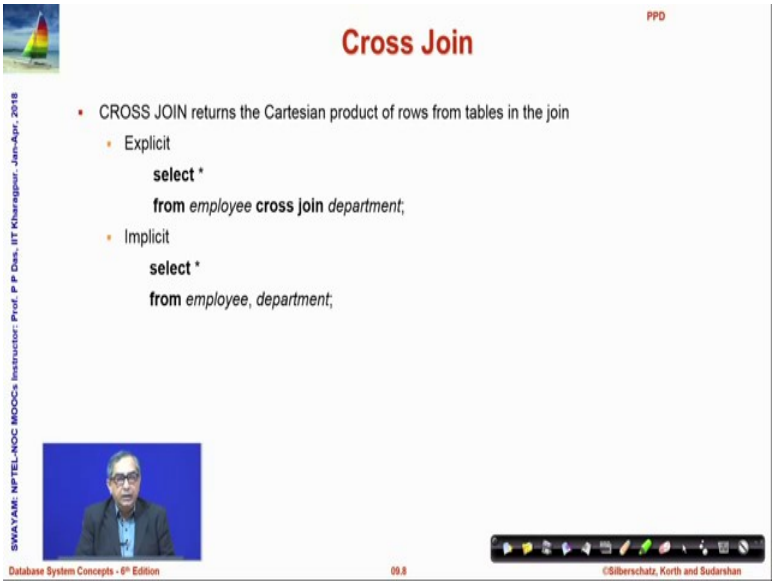
operation typically uses sub query, is used in a sub query, in the FROM clause we will see those usages later.

(Refer Slide Time: 02:19)



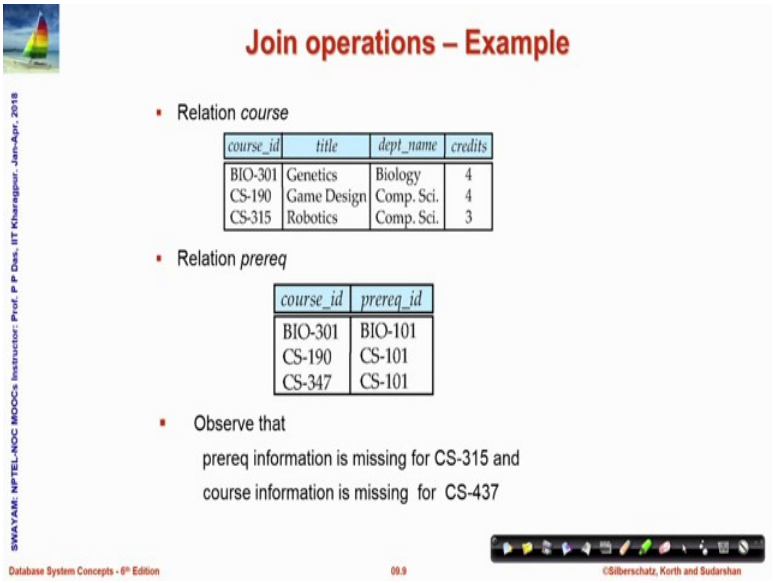
So, if we look into the different types of join that SQL support, these are the different classifications. So, we have CROSS JOIN, we have INNER JOIN, which specifically could be EQUI JOIN and even more specifically NATURAL JOIN and we will see there are variety of OUTER JOIN, that are possible. There could be a SELF JOIN also, where one relation is joined with itself.

(Refer Slide Time: 02:51)



CROSS JOIN is just a formal join based name for Cartesian product of two rows. So, you could explicitly do a CROSS JOIN, which you can see here or you can implicitly also do a CROSS JOIN, but by specifying two or more relations in from clause, and taking all the attributes from there, we have seen these kind of Cartesian products earlier. So, CROSS JOIN here is more as placeholder. in the context of the join semantics, the pure Cartesian product is a CROSS JOIN, but what would be more interesting is, when we take different kinds of inner and OUTER JOINs.

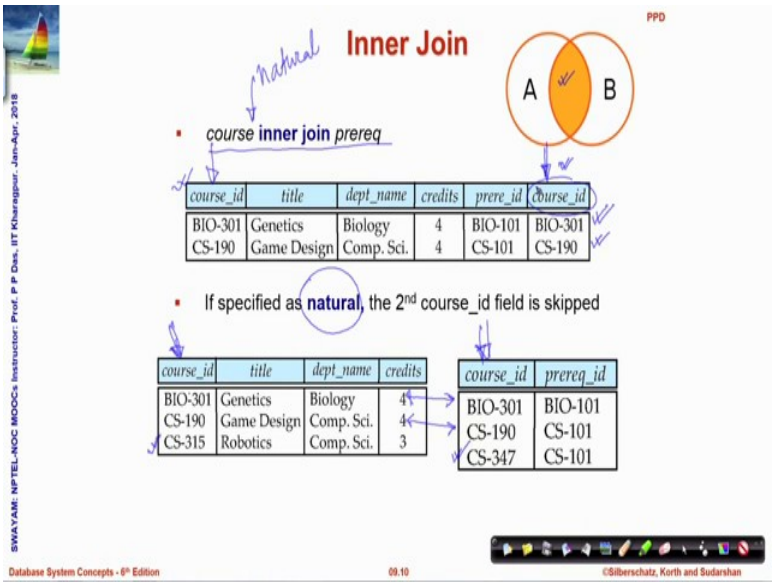
(Refer Slide Time: 03:30)



So, let us start with a simple example to understand the issues. So, there is a relation course, which has four attributes and in this particular instance, it has three tuples; three rows and there is another relation prereq, which specifies the prerequisite for every course.

So, it has two attributes; the course_id and the corresponding prerequisite_id. It also has three rows; three tuples here, and if you look at the instances carefully, you will find that the three courses that are specified in the course relation all are not specified in the prereq relation. Bio-301 and CS-190 is present in prereq, but CS-315 is not present in at the same time, the prereq has one particular tuple, specifying the prerequisite of CS-347, which in turn is not present in the course relation. So, with this observation, let us start trying to see what different joint mean.

(Refer Slide Time: 04:54)



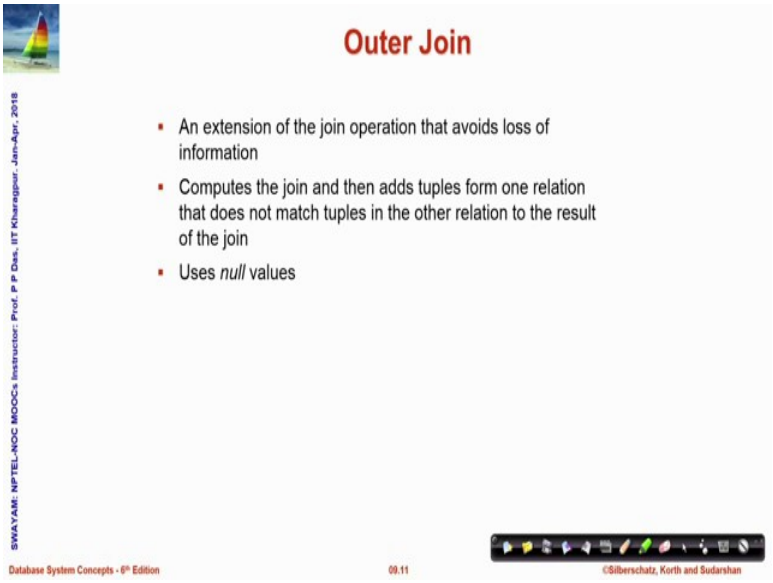
So, in a join is computed, then in terms of the two relations that we have, there is an attribute course_id, which is common. So, once we have taken the cross product, we will from the cross product, only retain those rows, where the course_id in relation course and the course_id in relation prereq are same. So, when we do this particular record, when it gets mapped with this corresponding record, it will generate the corresponding output record. Similarly, CS 190, when it is mapped to the CS 190, in the prereq, it will generate the second record, we have already understood this, the third record in the courses CS 315 has no match here in prereq.

So, that will not feature in the output. Similarly, in the prereqs C S 347 that exist has no match in courses. So, that also will not appear in the output and also in the output you find that the course_id has actually featured twice. This is the first column course_id comes from course. So, it should more formally be called **course.course_id** whereas, the second one comes from prereq. So, it should that should formally be called **prereq.course_id.**

Now, if in addition to saying that, this is an INNER JOIN, if I also specify the word natural, I can say natural here, if say natural, then this second duplicate attribute course_id will be dropped from the output that becomes a NATURAL JOIN, INNER JOIN as the name suggests, finds out the inner part of the two relations.

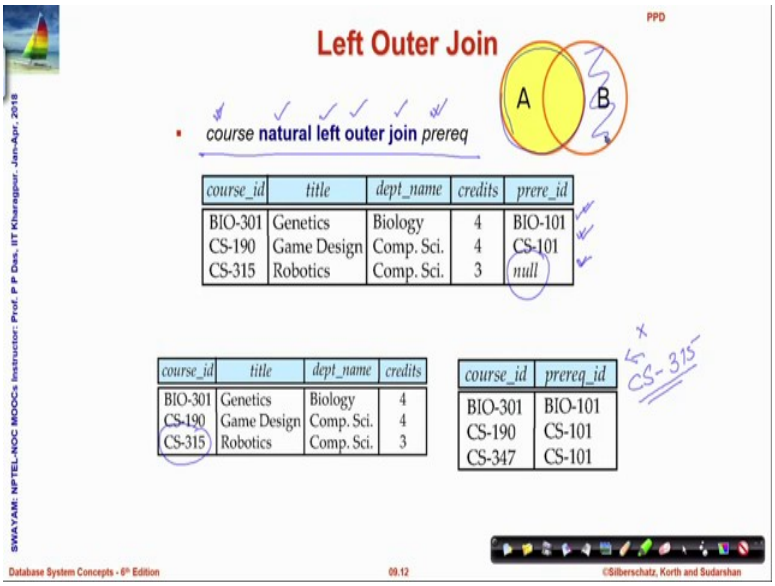
So, if we look at the two relations as A and B only those records, which are both have instance in A as well as B, in terms of equality of this course_id attribute will come in the output. So, this is the basic type of join, INNER JOIN which is most commonly used.

(Refer Slide Time: 07:34)



Now, we can extend this into a different kind of join, known as OUTER JOIN in the INNER JOIN. As you have seen that courses that exist in the course relation, but are not there in the prereq or the ones that exist in the prereq and is not there in the course are not featuring in the final INNER JOIN output. So, there is some loss of information in terms of this. So, why we are doing this we can compute and add tuples from one relation that may not match with any tuple in the other relation and if we want to do that then naturally for the other attributes of that tuple in the target relation we will not know the values. So, we will use null values this is the basic idea of OUTER JOIN.

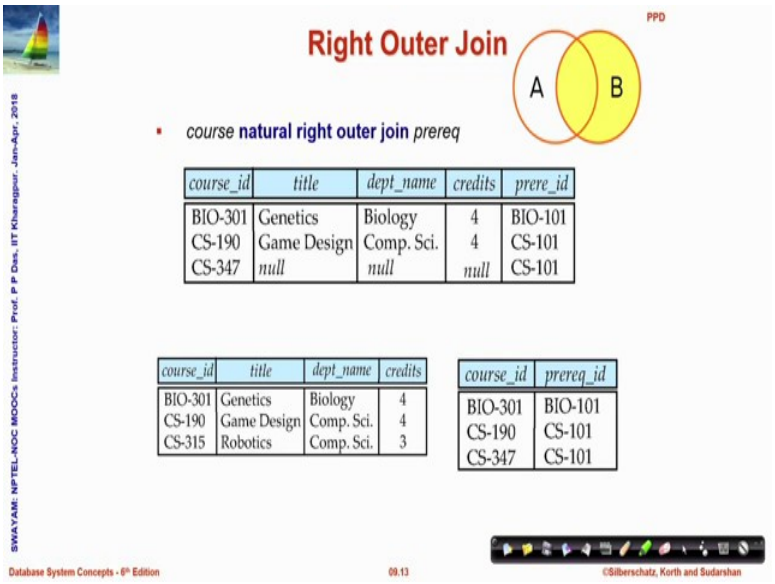
(Refer Slide Time: 08:34)



So, let us see what it specifically means? We first talked about left OUTER JOIN, left in the sense that we have, this is how it is written. Left OUTER JOIN is a sequence of commands that you give, you are also saying it is natural, which means that the common attribute will not feature twice in the output and this is the left relation and this is the right relation. So, left OUTER JOIN specifies that in the output all records of the left relation, in this case the course relation must feature.

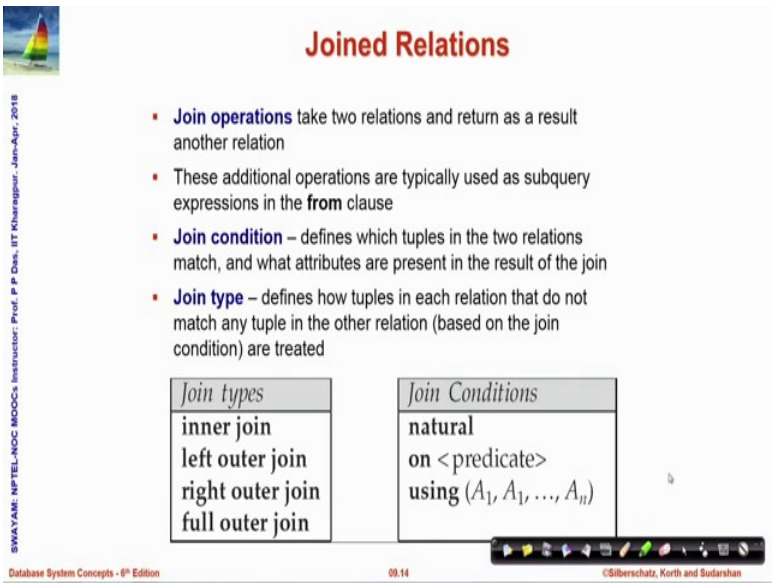
So, naturally when we do the join, we will get these two records as we have got in terms of the INNER JOIN, in terms of course 315 the CS 315, the third course there is no instance in the prereq, we will still have that in the output, but since the prereq value for that, the prerequisite value is not known, the prerequisite id will be set to null here. So, left OUTER JOIN ensure that, all relations of the left relation, all tuples of the left relation will necessarily feature in the output and that is a reason. If you see in the Venn diagram, the whole of this set, A is shown whereas, this part certainly will not feature.

(Refer Slide Time: 10:05)



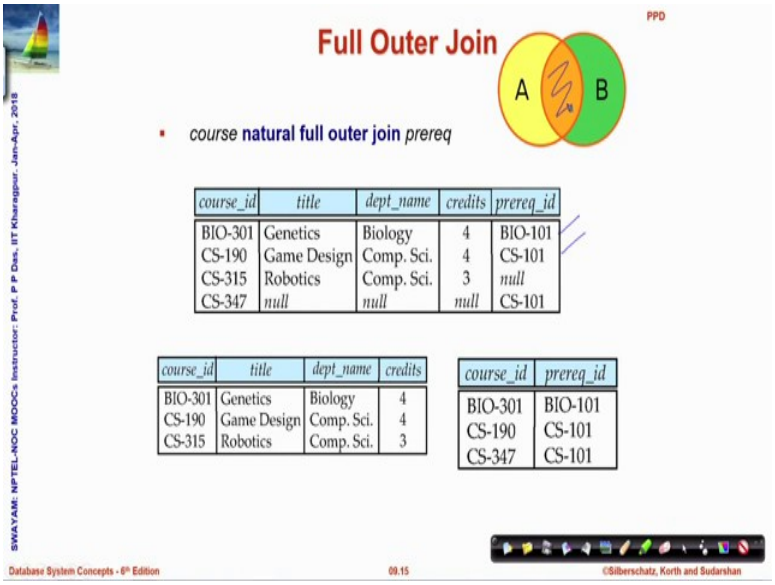
Now, similarly we can have a right OUTER JOIN, where the concept is the same except that. Now, we ensure that all records of the right relation, in this case the prereq relation will feature and therefore, C S 2347 for which there is no entry in the course relation will also come as a record and since we do not know the title department name and credits for these fields, we will put them as null and this again is a natural one. So, course_id is featuring only once. So, you will understand that since, we have a left version and we have a right version.

(Refer Slide Time: 10:47)



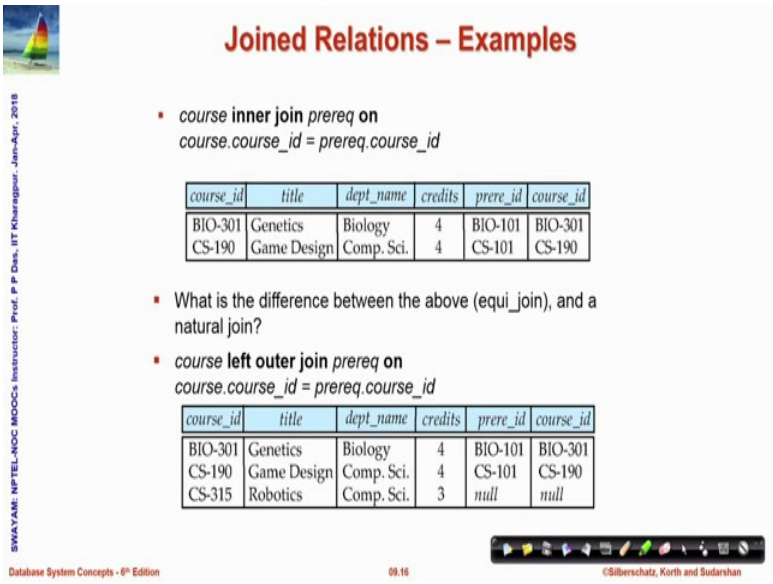
We can actually have a full version as well. So, if we look into the join relations in general, it takes two relations and returns a result and those additional operations are used in the sub query in from and there is a set of join conditions. So, these are the join conditions that we are specifying, whether it is natural and we will soon see that we can actually not depend only on the attributes that are common, we can actually specify that which attributes should be used in computing the joint. So, those are on condition and the using clause, we will just illustrate them soon and finally, there are four types of join that can happen, that is the INNER JOIN. We have seen the left OUTER JOIN, right OUTER JOIN and we will soon see the full OUTER JOIN.

(Refer Slide Time: 11:52)



So, full OUTER JOIN as you must have guessed will ensure that you get certainly the tuples from the INNER JOIN, which is here, you will get the tuple from the left OUTER JOIN that is here, that is a tuple which exists in course and there is no corresponding matching tuple in the prereq and you will also get the tuple from the right OUTER JOIN, that is for tuple, which exists in the prereq relation, but there is no corresponding tuple in the course relation and corresponding missing values are all set to null. So, these three kinds of OUTER JOIN are possible.

(Refer Slide Time: 12:38)

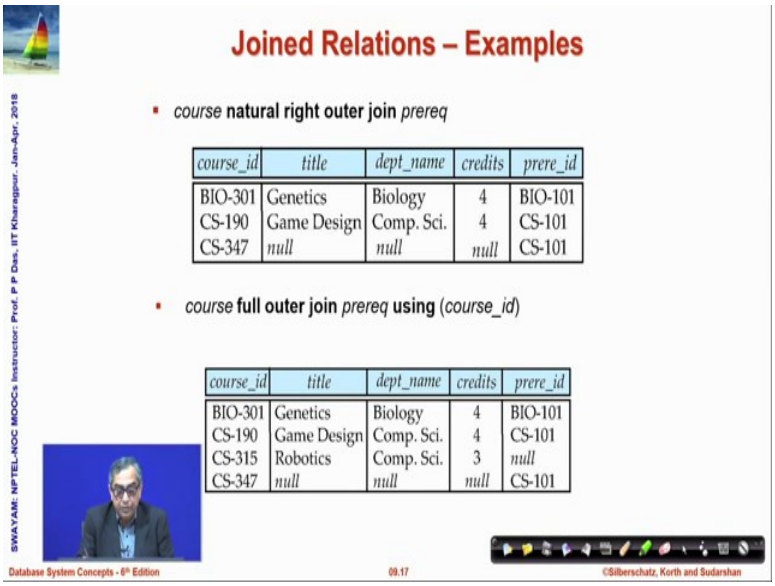


So, you can also specify join by saying that explicitly saying what attribute we want to join on and if you specify that, then you are saying it is a course INNER JOIN prereq. This part was same then you are putting an on clause, saying in the on clause, you will have to provide a predicate that is, which field should equate or match with what field. So, you are saying course.course_id = prereq.course_id.

So, this result incidentally happens to be same as just doing the INNER JOIN, but we are illustrating that, on clause can explicitly use. For example, between the two relations, we have more than one common attribute, but we may want to actually do the INNER JOIN based on only one of them or equality on two of them and so on..

So, this kind of a join, where INNER JOIN, where you set two fields to be equal or two or more fields to be equal is also known as EQUI JOIN and since we have not specified natural, you can again observe, then the course_id attribute has occurred twice. If it was said natural then the second course_id attribute would not have come in the result, this is a showing. The left OUTER JOIN in terms of on clause and we have seen similar results. And now, this can be seen in terms of the on clause as well, and you can see in that second course_id field. This entry is null, because actually you do not have that in the prerequisite set and; obviously, this set will be null, this field will be null..

(Refer Slide Time: 14:35)

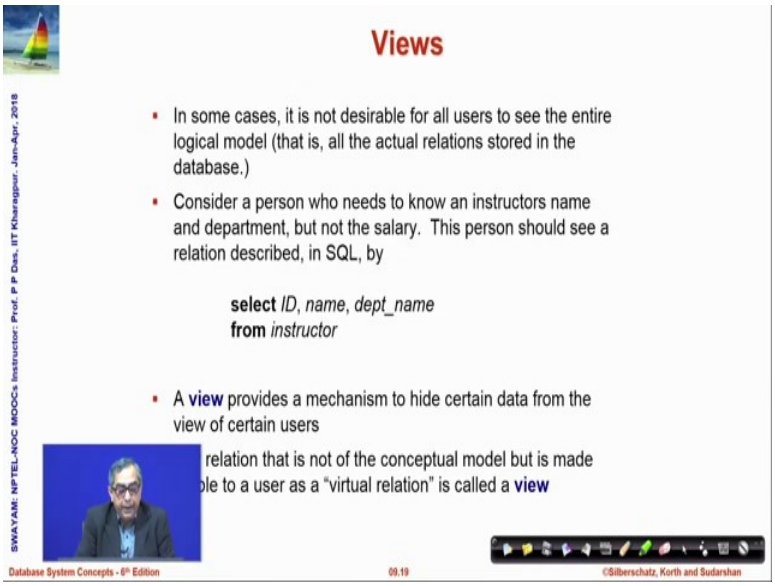


So, this is another example, showing you the natural right OUTER JOIN, this is you, showing you full OUTER JOIN and we are showing the use of the using clause. We can say using and put a set of attributes and the meaning is the join will be performed, based on those attributes. So, here in this case again the join will be based on course_id.

(Refer Slide Time: 15:01)

So, that was about different kinds of join that we can do, which we going forward. We will see that form, a very critical has a very critical place, in terms of query formulation. Now, we take you to a different concept known as views now.

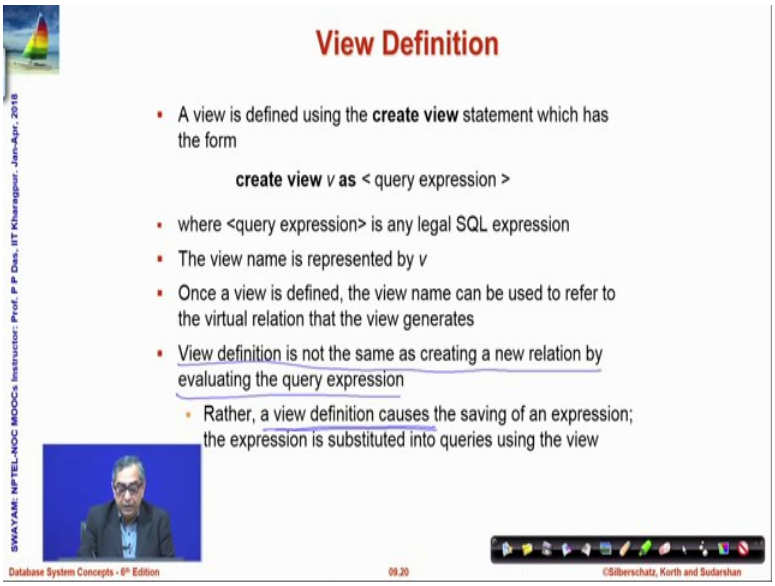
(Refer Slide Time: 15:20)



We have seen that. So, far we have been computing certain query results, based on one or more relations one or more instances. Now, in some cases, we may want the results to be restrictive in terms of based on the user or based on the context, in which the result should be used. So, we may not want all fields of a result to be visible to all the users or to the application. So, we may not expose the whole logical model and in those cases, we introduce a view. So, here we are showing one, where, from the instructor relation, we are only picking up three fields and we are not picking up the salary field.

Now, you would think that, well this is what we can do in terms of the normal query and certainly then, what is the point of using this? Now, what we can do is, we can create this not adjust as a query, but as a view one, once we create this as a view, it actually this query expression is treated as what is known as a view expression and every time you want to use that view. The actual tuples in that view are computed, but this is not actually a relation that exists in the database. So, it is a kind of, can be thought of as a kind of virtual relation, which exists, which can be seen only when you use that.

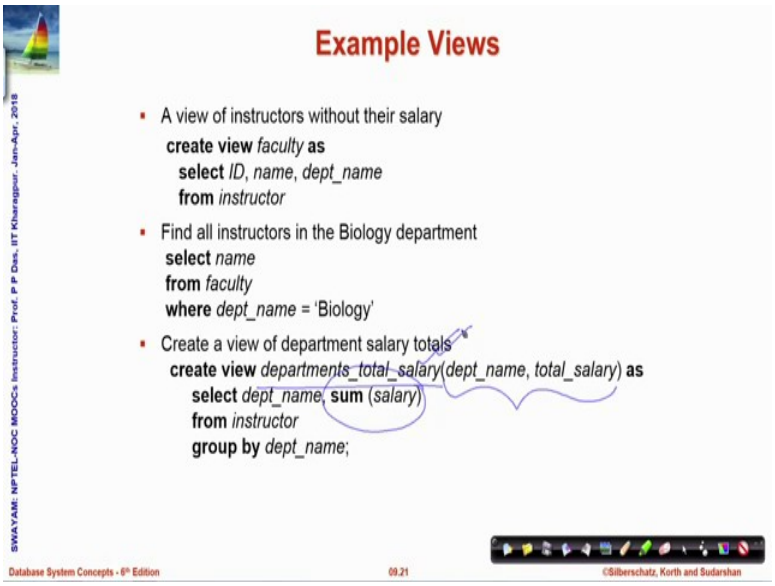
(Refer Slide Time: 17:22)



So, there is a subtle but very strong difference between, actually computing a result, through a select query and defining a view, based on a select query and then making use of the view, as if it were actually a relation that existed. So, to do this, this is how we go about. It is the send text, is very similar to the create table. So, you do a create view give a name and then you specify as is the connective and specify the query expression, which is an SQL query, which will let you compute the view, every time you actually need it..

So, this is a view name, once a view is defined, the view name can be used as a virtual relation, it can be used exactly as we use any of the really existing relation, the conceptual relations that we have created, through create table. So, it is the difference, this is what needs to be understood very well, the view definition is not the same as creating a new relation, once you create the new relation, the time you have created it, you get the result and that result is explicitly available as a set of tuples as a table rather a view is a definition, which you stored in the database as an expression. So, every time you make use of that view, at that time the set of tuples are computed. It is not existing in the database as stored like the real relations and based on that computation, all the rest of the query will actually be executed.

(Refer Slide Time: 18:58)



So, let us take a quick look, this is a create view, we have created the view of a, of view called faculty, from instructor relation. Instructor relation is the real one, I existing one and faculty is a view expression being created and in that, what we have done? Simply, we have taken a done, a projection, we have lived left out the salary field. Now, we can make use of that view, you can see that we are doing from faculty.

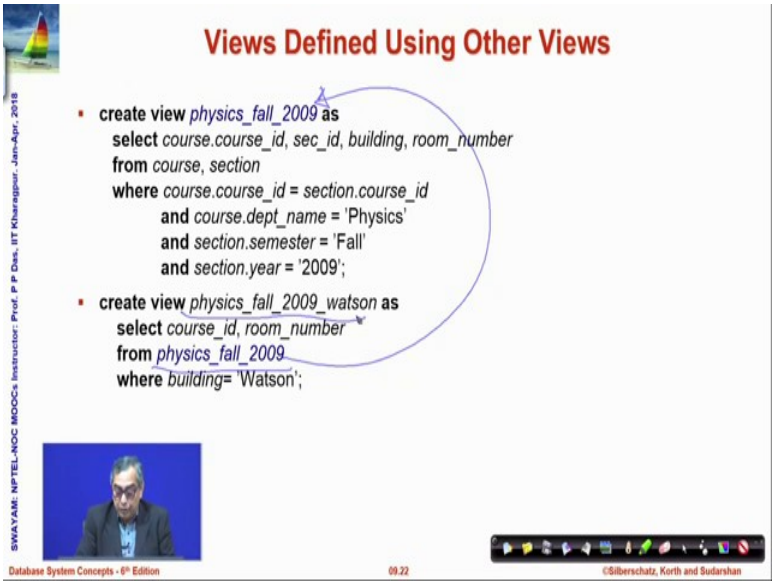
So, this actually is a view, but this behaves as if this is varying relation. So, from faculty we are trying to find out the name of all those faculties; who belong to the biology department. So, what will happen, when they want to execute this query? This will refer to this view. So, to execute this query, it will have to first execute this query, get the temporary virtual instance of the virtual relation, created and based on that, this query will be computed and the results will be given accordingly.

So, that is the basic purpose of the view that the whole thing, the whole view expression remains as an abstraction in the database and computed whenever it is used. So, this is showing you another view, which shows certain computed information. For example, we are creating a view for departmental total salary, which will show as two fields department name and total salary, which has been created by aggregation.

So, anytime we make use of this view in a from clause, we will get, we will feel as if such a relation really exists where the department name and the total salary of the instructors in that department are stored, but it really does not exist. It is computed

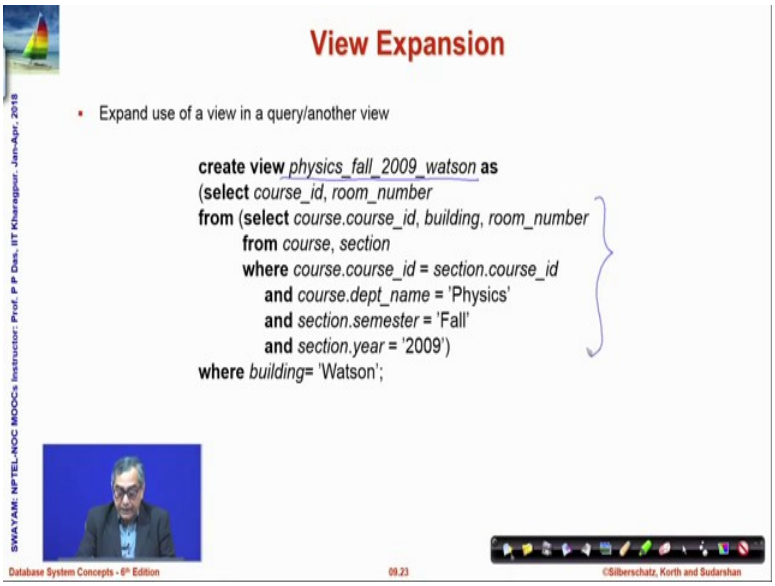
whenever it is needed, whenever it is used, you can actually use views to create other views.

(Refer Slide Time: 20:52)



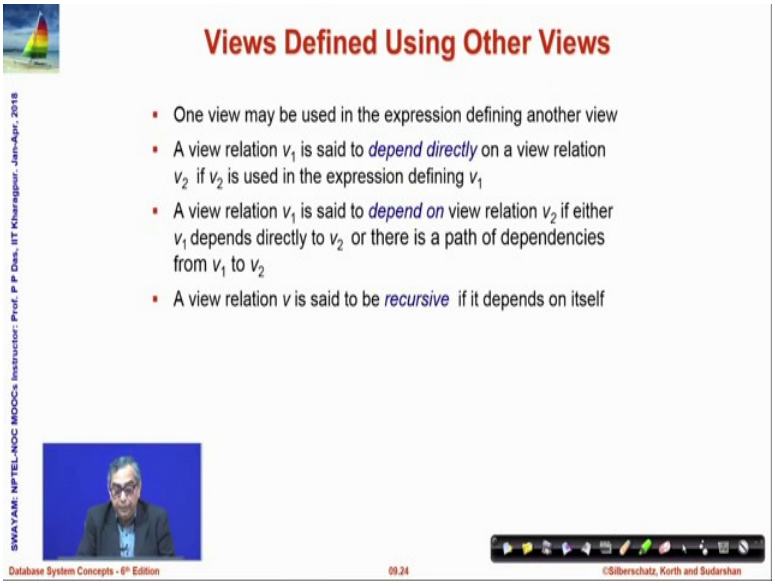
For example, this is one view which is the view of physics_fall_2009, which are all courses that are offered in physics, from the physics department in the semester fall of year 2009 and using that we can create another view. See here again, we are in the, from clause we are using this view. So, creating this using this view, we are creating yet another view, which shows the courses that ran in the Watson building. So, views can be used as I have already said as any other.

(Refer Slide Time: 21:35)



Actual relation, but they do not really exist. So, if you expand out, if you just put the physics fall 2009 expression, within the view definition of physics for 2009 Watson, this is your earlier view relation. So, this is known as view expansion. So, this is actually the query that, you are executing and that has a lot of value.

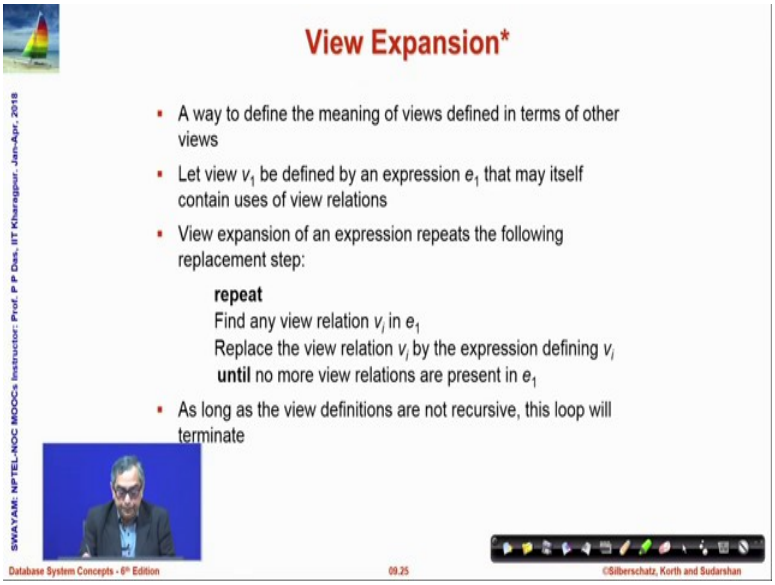
(Refer Slide Time: 22:04)



So, as we have said views can be defined indirectly from one relation. So, these are called direct dependence or they could be defined in terms of a chain of relations v1 in

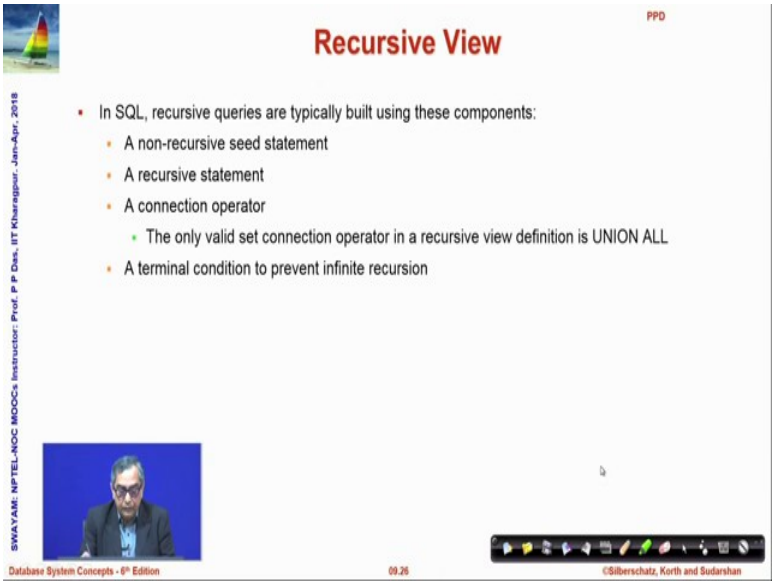
terms of v2, v2 in terms of v3 and so on. And a view relation can be recursive also, that a view could be in terms of itself.

(Refer Slide Time: 22:29)



A lot of power view expansion is the process that SQL uses to evaluate a view. So, I would request you to study this and understand that this process works. This is pretty much like pseudo code C program.

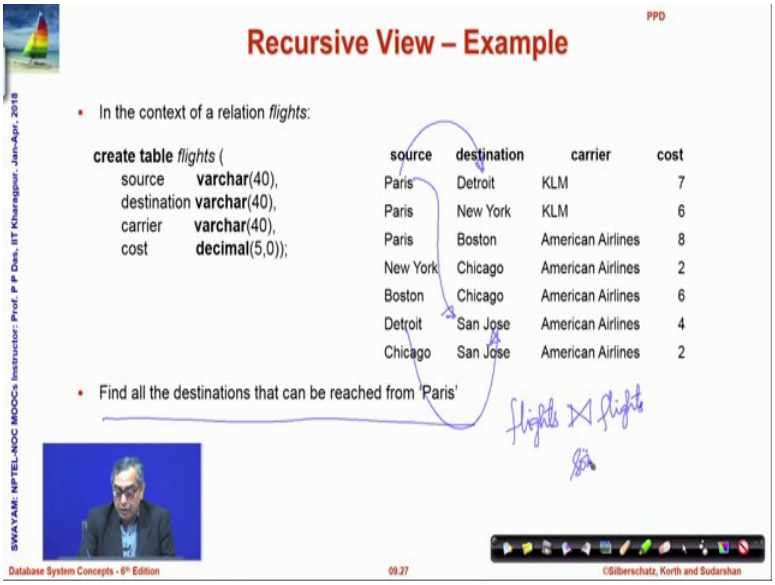
(Refer Slide Time: 22:47)



Now, moving to recursive views, the views where the same relation can be used in the view to define another view, we need like every other recursive structure. We need first a

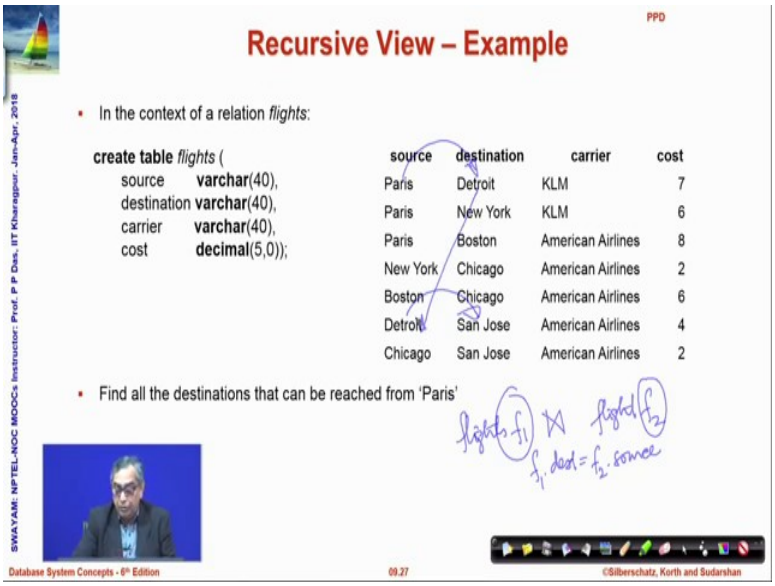
non-recursive statement, which is called the seed statement. we need a recursive statement, which can recur, we need a connection operator, which can connect the non-recursive and the recursive results together put them together, the only connective that is valid is union, all that is multi set union and we also need some kind of a terminal condition to guarantee that the recursion really a terminus, it does not go on forever.

(Refer Slide Time: 23:36)



So, let us take an example. So, this is in context of a relation flights, where the four fields are as specified and there is an instance shown, which show that different source destination of different carriers, carrying people from one source to the other destination and what do you want to find is all destinations that can be reached from Paris. Now, you can see that from Paris, if I can reach Detroit and from Detroit I can reach San Jose, then I can actually reach San Jose from Paris. So, that is the basic reach ability. So, that will necessarily, if I want to compute that, then I will be able to compute this very easily by doing NATURAL JOIN of flights with flights provided, I take say source.

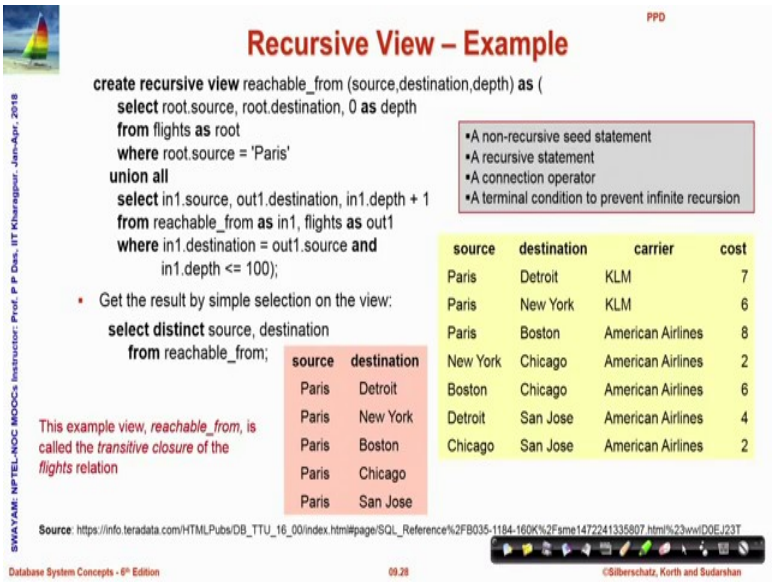
(Refer Slide Time: 24:38)



Let us compute it like this, flights f1 join, flights f2 and I will have **f1.destination = f2.sources**. So, the idea is, if something goes from Paris to Detroit that is in f 1 and if some flight goes from Detroit to San Jose that is in f 2, then the destination in f1 and the source in f2 have to be equated. So, if we do this kind of a self EQUI JOIN, then we will be able to find out all flights that go from Paris to San Jose or all places that you can reach from Paris in one hop.

Naturally, once you reach, once you do that then you may be able to go to another destination in two hops and once you do that then, you may be able to reach another, yet another destination in three hops and so on. So, we do not really know how many hops, maximum would be required to compute this reachability information. So, that is the reason we need to make use of the recursion and so, this is how we express it.

(Refer Slide Time: 25:54)



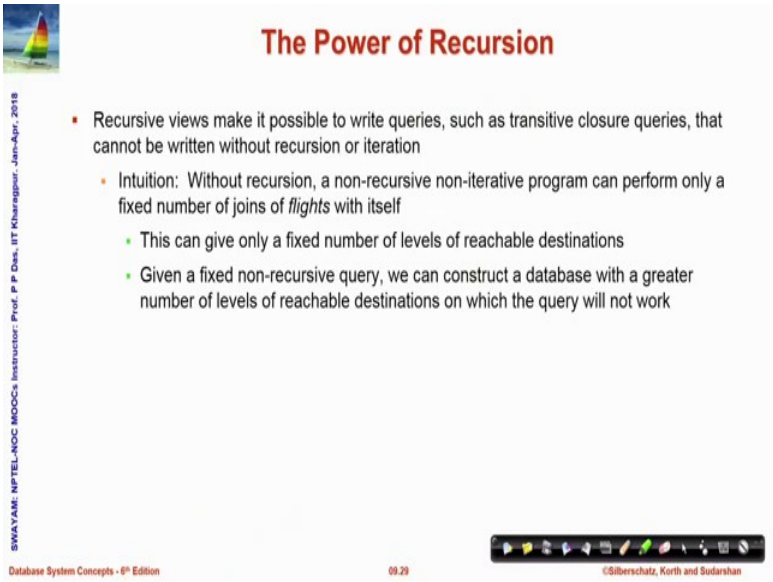
So, if we look into this, we are specifying that is a recursive view. It will happen with itself, this is the name and this is what we want to compute source destination and we take another dummy attribute kind of which specify the depth of recursion. So, the present instance of the relation is at depth 0.

So, which defines your non recursive seat part. so, it says select. So, you have renamed it as flights as route, you are specified that it has to start from Paris and you can find out the source destination pair at depth 0, then you specify the recursive part that is the second hop has to be defined. So, we are saying that, if you had the reachability then call, let us call it in one. This reachability maybe in one hop that is a depth 0 maybe in 2 hops that is a depth 1 maybe at 3 hops; that is a depth 2 and you take another instance of flight as out 1 and what you need is the destination in the first in one has to be same as the source in the other so that they get connected and then you output the source from the first one destination from the second one and naturally the depth has got incremented, because you have done added one more hop and so, this is the and you add another condition saying that **in1.depth** should be <= to 100. This is as I mentioned is a terminal condition which makes sure that, you do not get into infinite recursion. So, this view, recursive view cannot be used to compute any reachability, which has more than 101 hops.

So, that is to be noted and finally, we need to connect these two results, which is the initial start seed and the recursive one. So, this is the connection operator.

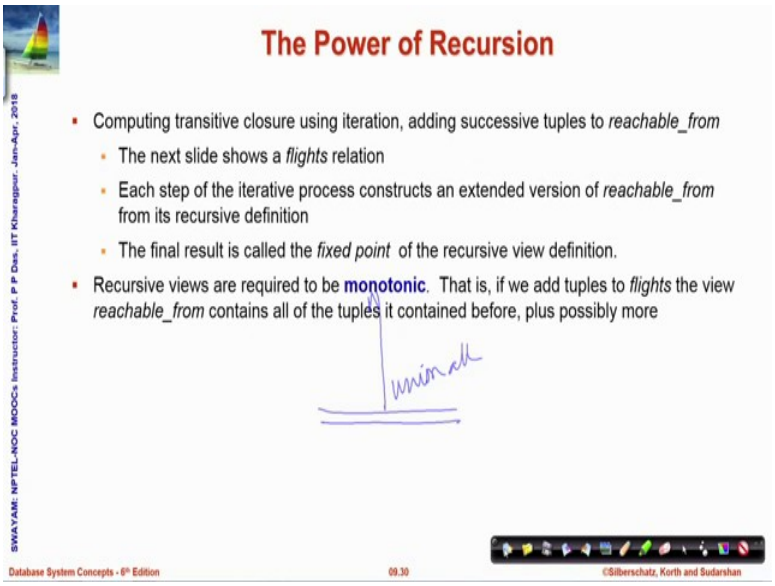
So, this is basically the idea of the recursive view, those of you who are more familiar with discrete structure, would have known among relations in some more depth, you would know that, we can define a transitive closure of a binary relation. So, this recursive view is necessarily computing the transitive closure from the flights relation. So, this is the instance of the flights and on the final computation, this is what you get. This gives you all the destinations that can eventually be reached from the source Paris.

(Refer Slide Time: 28:41)



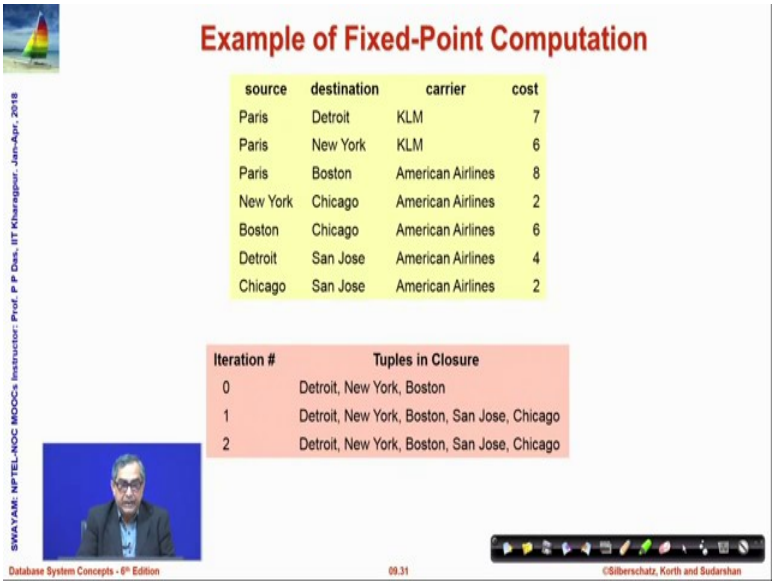
So, the recursive is very powerful in the sense that without recursion a non-recursive version can only find flights up to a certain number of hops and whatever ma query you write it is always possible to write out a database instance which will have more hops and your query will necessarily fail.

(Refer Slide Time: 29:05)



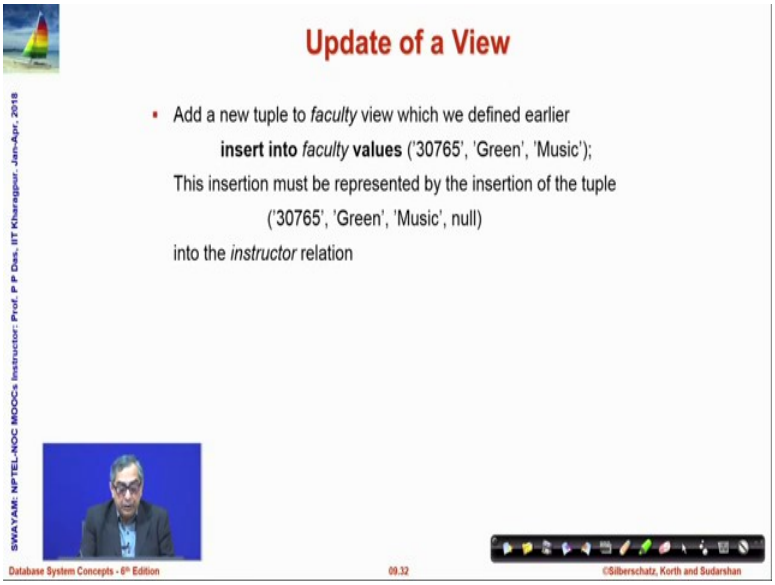
So, we make use of the recursion here to make sure that you can actually extend this to whatever depth you want and to compute this we keep on computing till no changes are possible and in that sense this recursive views are said to be monotonic in that every time you compute your result necessarily becomes larger and that is the reason you for the purpose of being monotonic you are actually making use of the union all. So, that makes it all inclusive

(Refer Slide Time: 29:49)



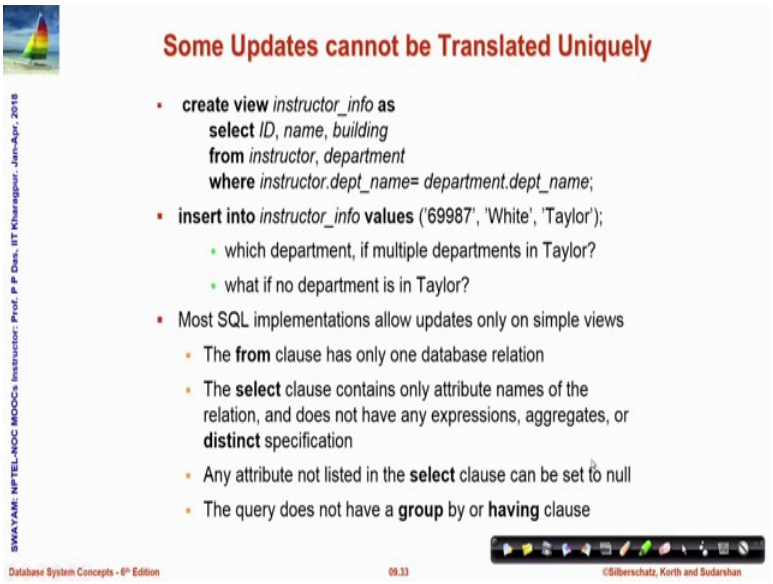
So, now, if we go and this is the instance and this you can. Here, I have shown that how the iteration actually happens in the iteration 0, in the flights itself, you had three destinations, then you add two more in iteration 1, in iteration 2, you do not add anything else. So, your result henceforth will not change. So, you have reached a fixed point and computations, are over.

(Refer Slide Time: 30:10)



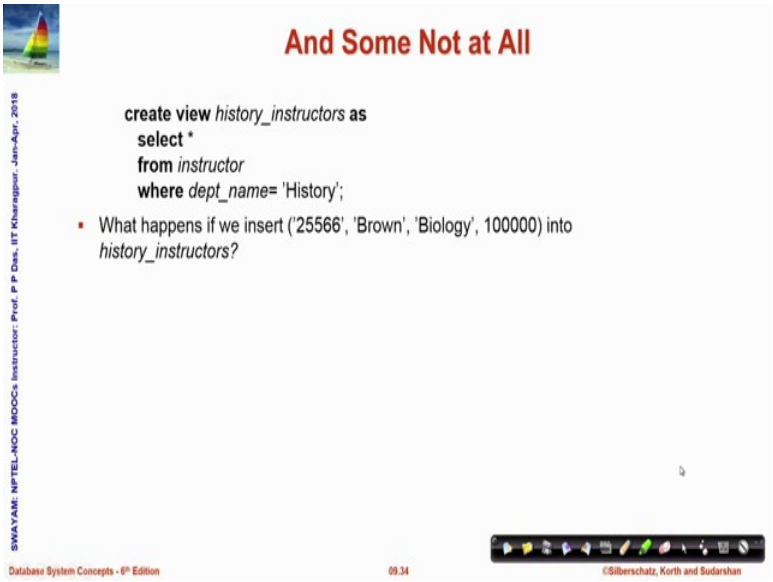
You can also update a view you can insert a record directly into a view, but since view only is partial information on the relation, when you INSERT INTO a view since, view is virtual. There will have to be an insertion, in the real relation and in the real relation you may not know certain fields. So, if you are doing this insertion into faculty, which is a view of instructor, then the salary field is not known. So, in the actual instructor a null will have to get inserted in the salary field. So, the salary field needs to be null able kind of field so updates on views have certain restrictions.

(Refer Slide Time: 30:47)

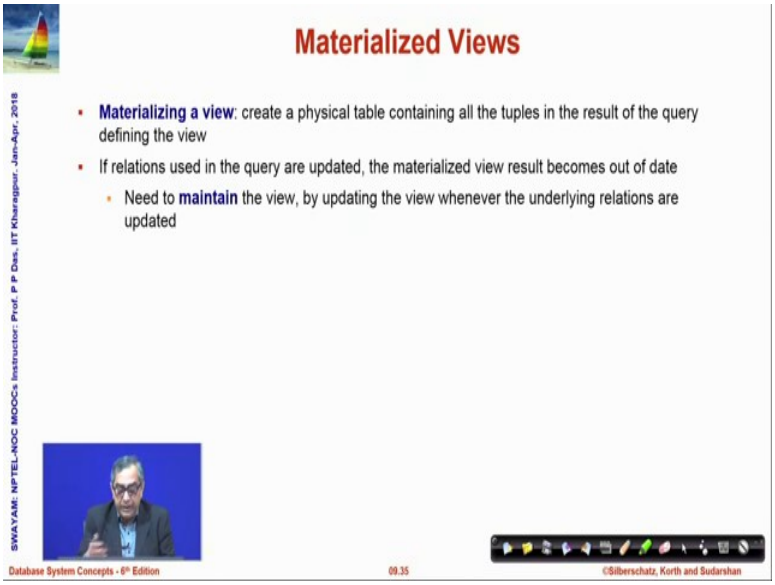


So, there are some more instances that I have given, which you can study and try to understand that what are the difficulties of updating on the view. So, it can be done, but it has to be done in a restrictive sense. So, these are the different conditions that has to happen for views to be updated.

(Refer Slide Time: 31:08)

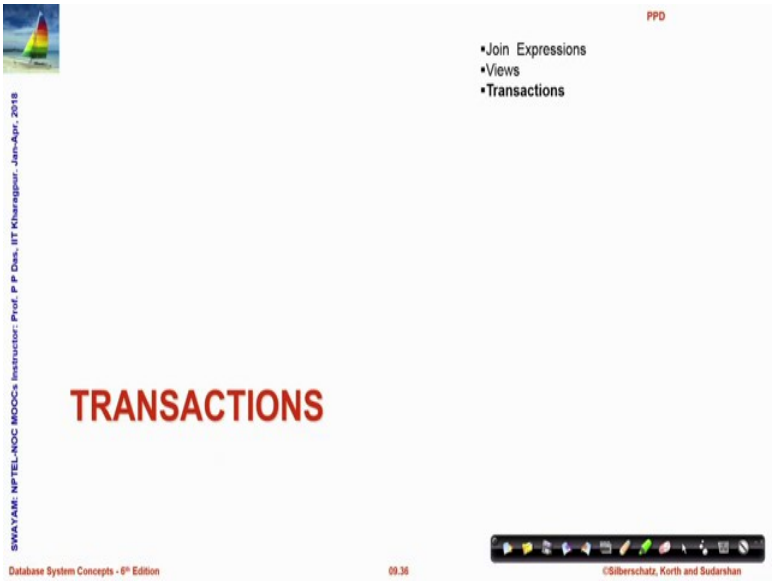


(Refer Slide Time: 31:11)

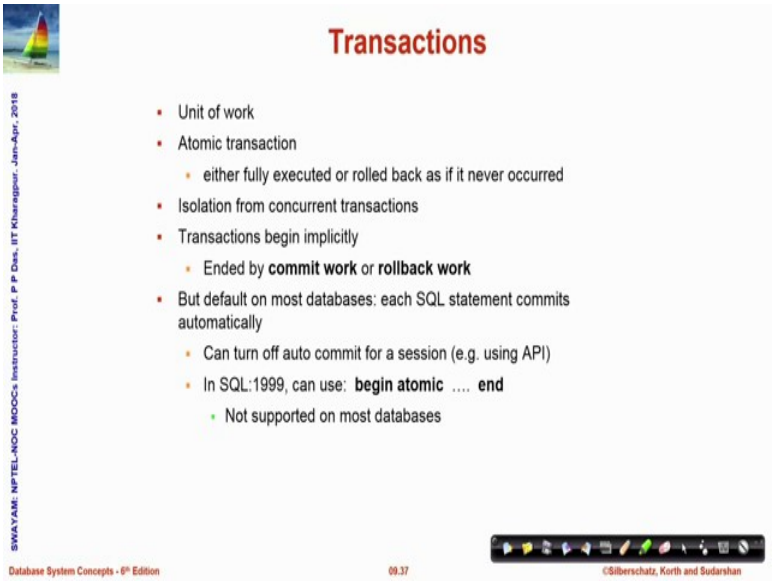


So, please go through these slides to understand what are there in terms of the views? Finally, view is a virtual relation, but it can be materialized also, that is materializing is basically computing a physical relation at the at the instance of the view, but naturally if you materialized then there is a certain point of time, where you have materialized where you have made it into a physical relation and hence, if your original source data in the view changes in future the materialized view also need to be updated otherwise, your data will get bad.

(Refer Slide Time: 31:43)



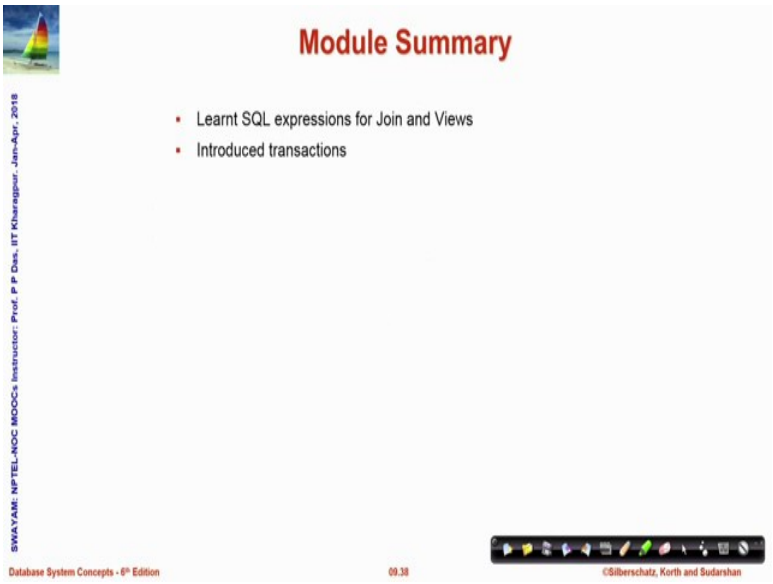
(Refer Slide Time: 31:49)



Finally, in this module, we mentioned that, there is something called transactions, which we will take up at a later stage, in much depth. This is just to get you familiar with atom a transaction is a unit of work, which is usually atomic, which is either fully executed or if it fails, it will be rolled back as if it never occurred and this is required for isolation in concurrent transactions. So, we will talk about this lot more when we take up concurrency and related issues.

So, come transactions implicitly begin and they end by either committing the work that they have successfully finished or rolling back that, this cannot be done. So, there are some features in the SQL for doing transactions and, but usually you can transactions, commit by default and the only it is exceptions, when the rollback is happening and we will see more of that later.

(Refer Slide Time: 32:49)

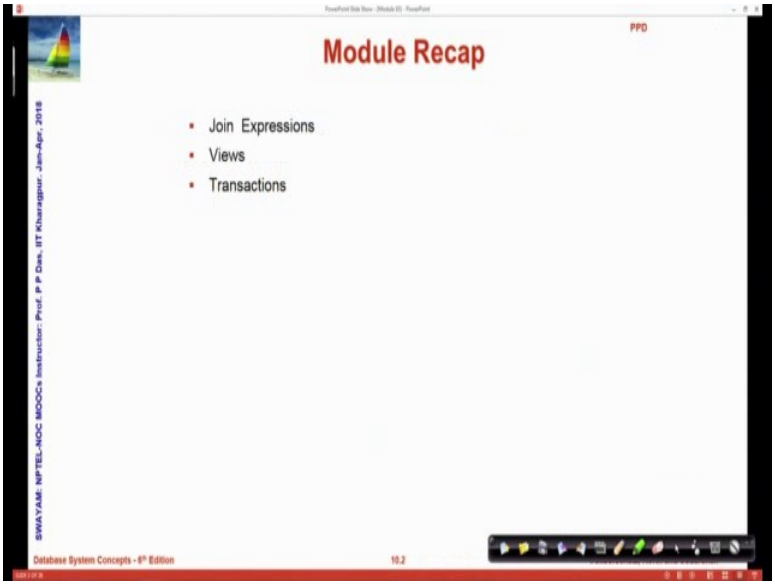


So, to summarize in this module, we have learnt about two important SQL features in terms of join and views and we just introduced the basic notion of committing transactions.

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

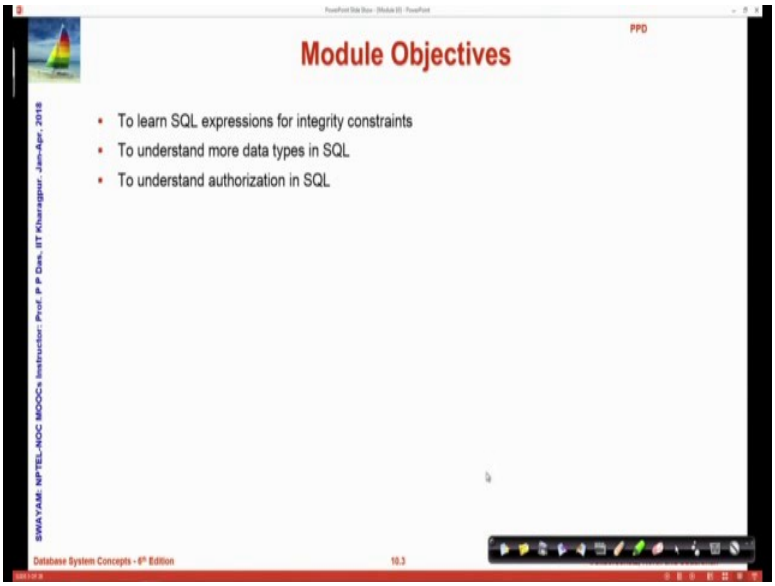
Lecture – 10
Intermediate SQL/2

(Refer Slide Time: 07:37)



Welcome to module ten of database management systems. We have been discussing about intermediate level features in SQL; and this is a second and closing module on that. We have in the last module; talked about join expressions, views and transaction in a bit.

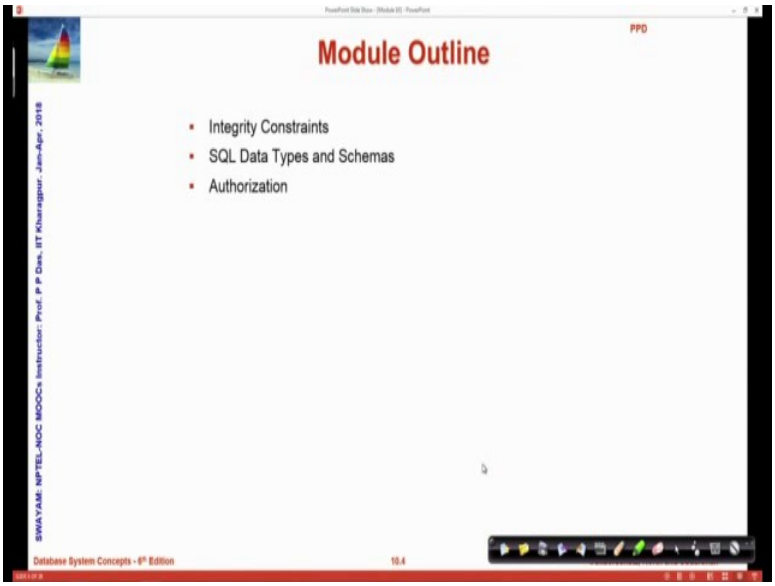
(Refer Slide Time: 07:46)



In this module, we will try to learn SQL expressions that are responsible for maintaining in the integrity of the database. We have talked about integrity a little. We will now see how explicitly integrity can be checked and how different kinds of integrity can be ensured through SQL. We will also talk about more data types.

We have seen the basic primitive data types, and we had promised that we will talk more about the data types including user defined data types here. And finally, we will talk about a very important aspect of authorisation as to who can do what in a database in through SQL.

(Refer Slide Time: 08:27)

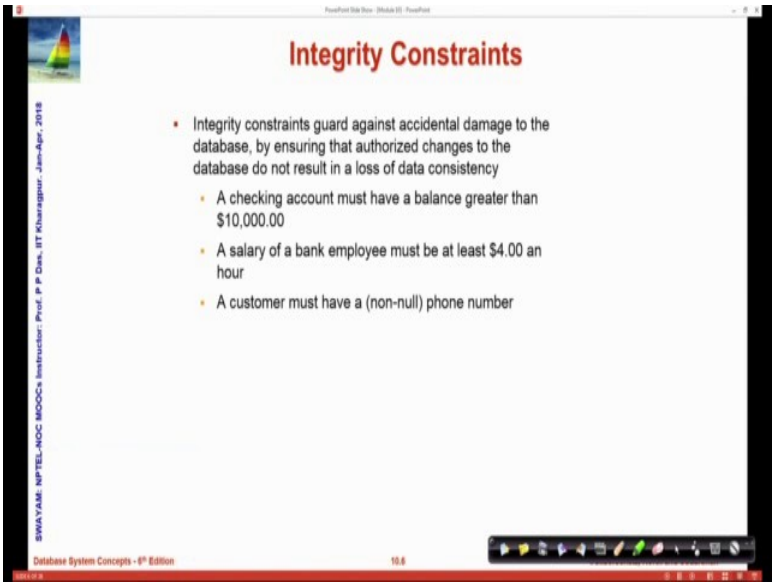


So, this is a module out line.

(Refer Slide Time: 08:30)

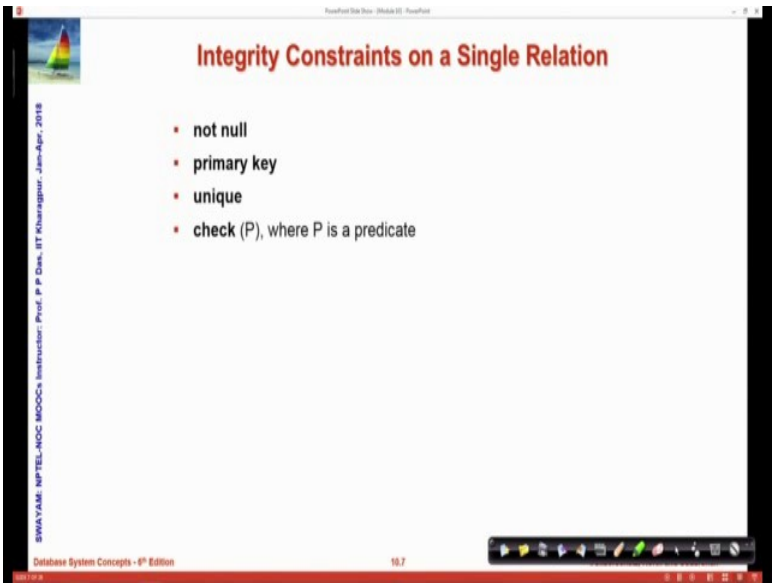
We start with integrity constraints.

(Refer Slide Time: 08:31)



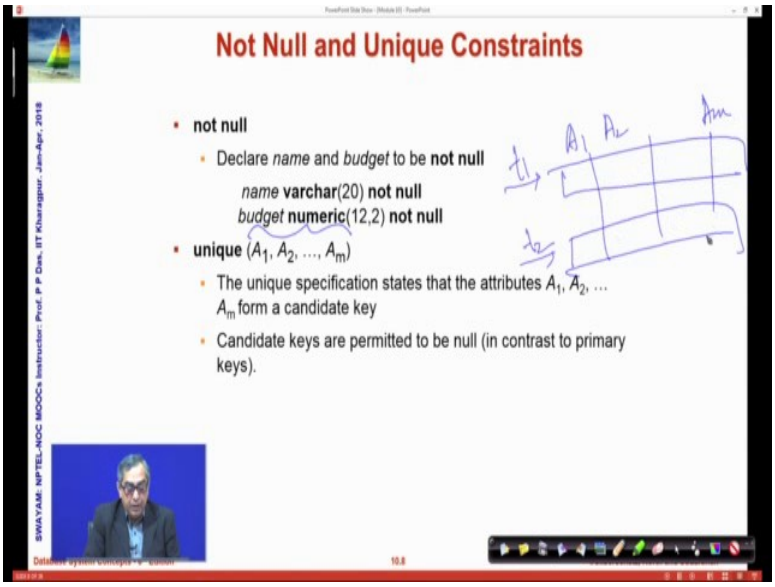
Integrity constraints guard against accidental damage to the database that is there are certain real world facts that must be ensured in the database all the time. For example, in bank accounts, we have a minimum balance that need to be maintained, for a particular customer we might want that the customer's phone number must be present. We may have certain age bar in terms of entering into certain memberships or certain employment and so on. All these kinds of real world constraints need to be represented and maintained in the database and that is the purpose of the integrity constraint.

(Refer Slide Time: 09:16)



And we will first look at the issues of integrity constraint for a single relation, and we have seen the use of NOT NULL and PRIMARY KEY. We will also talk about what is UNIQUE and we will see how actually general constraints can be checked in terms of a CHECK clause with a predicate P.

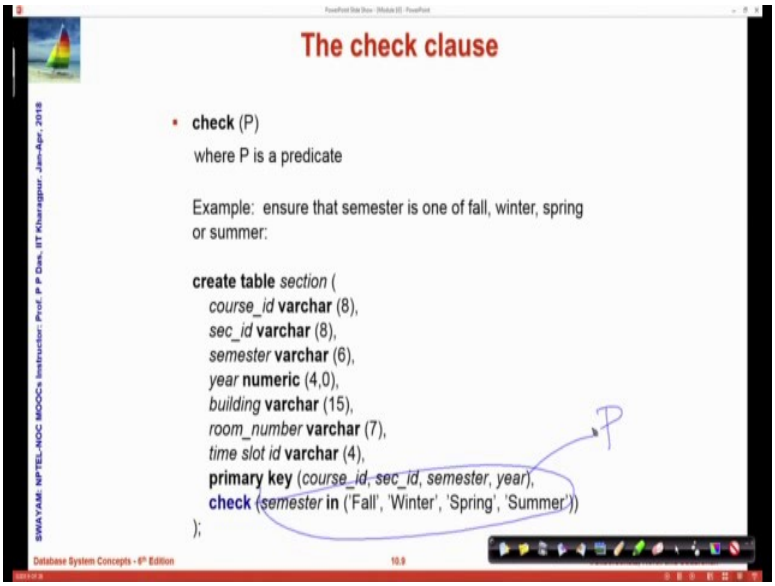
(Refer Slide Time: 09:43)



So, NOT NULL. we had seen this before we can while creating the database table. Create table we can specify a field to be NOT NULL and then in that case null values will not be allowed in those fields**.** So, they will must be some value given. You can say one or more attributes to be UNIQUE. If you specify them to be UNIQUE; that means, that in any instance of the table in future that cannot be two tuples which match in all of those attribute.

So, if you say A_1 to A_m are UNIQUE; that means, that if we have two different tuples in the table anytime in future. Two different rows in the table t1 and t2 then across A_1, A_2, A_m they must differ in at least one attribute value. So, uniqueness is a basic requirement for being a CANDIDATE KEY, but they are, but still permitted to be null which in contrast to what is the true for PRIMARY KEY already you know that PRIMARY KEY values cannot be null, but uniqueness allows a null value, but they have to be different.

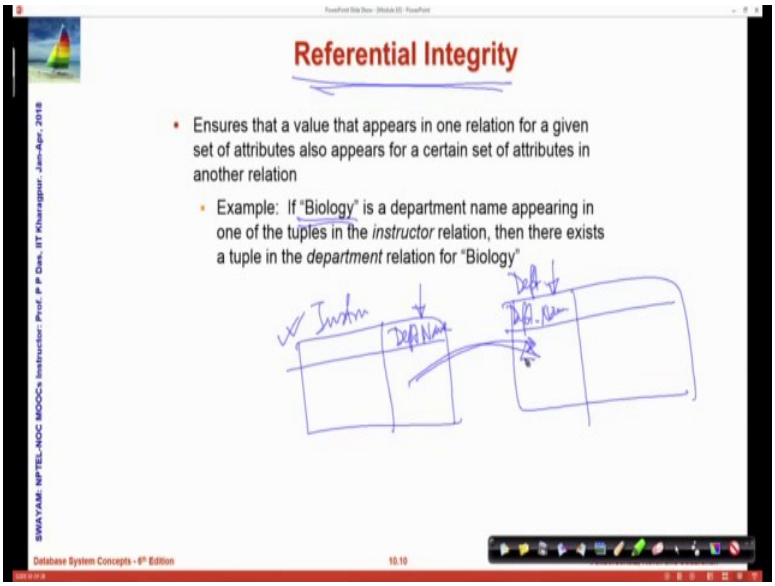
(Refer Slide Time: 10:58)



The CHECK clause is where you say CHECK and then you put a predicate. So, the idea is like this, suppose I know that I have specified and attribute called semester. It is a varchar 6 which means that it can have a string maximum of length 6, but naturally I can write anything there I can write morning in that field. I can write welcome in that field and so on, but those are not valid names of semester. So, I want that in my design semester must have any of these values only.

So, we say semester in. So, I have listed the values that are allowed in is a set membership. So, it says the semester in so which means the value of the semester is be one of this fall. And this whole thing now becomes the predicate P on which I give a check which means that, whenever I am creating once, you have created the table. when I want to insert or update the values in the table, the records in the table; the value of semester has to be always within this. Otherwise, the CHECK integrity constraint will fail, and the update or insert will not happen and an exception will be raised. This is the basic idea of CHECK constraints.

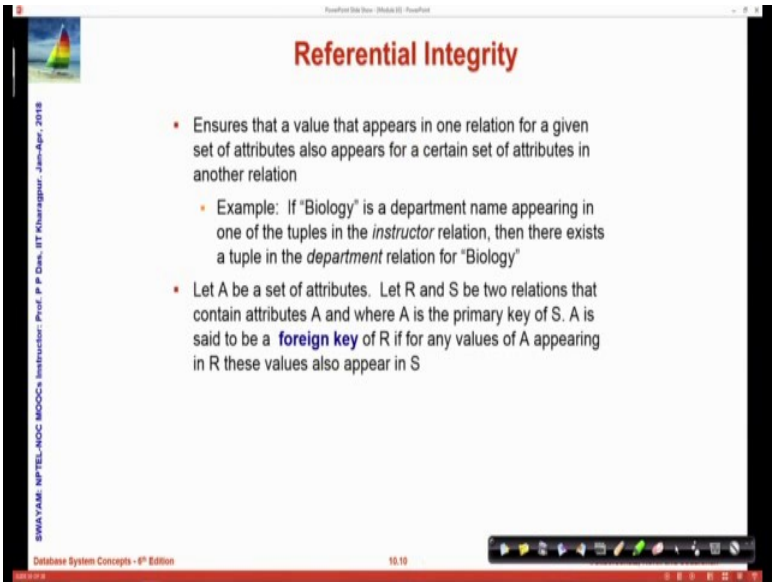
(Refer Slide Time: 12:25)



Now, let us move on to more involved integrity CHECK which goes beyond one table. So, let us suppose that we are talking about the instructor table we have the instructor table. An instructor table has a department name. Similarly, we have a department table which naturally has a department name. Now, we know that this is the key in the department table and therefore, here it is the foreign key.

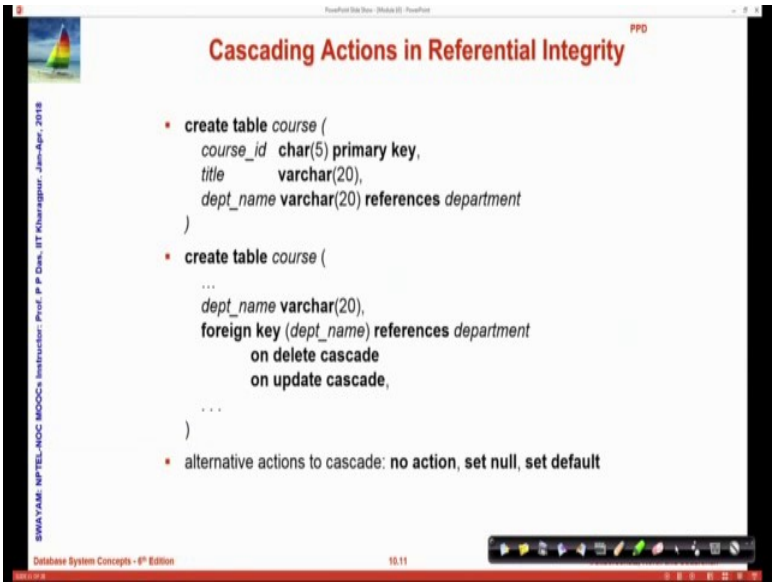
Now, while we are inserting records in the instructor table how do we guarantee that the record that we insert has a corresponding entry in the foreign key table that is difference table. So, it is when we are inserting and in a faculty in the saying that the faculty belongs to biology department there needs to be a biology entry in the department table as well. So, this is known as a referential integrity that is once you refer from one table to the other that reference must also be a valid one; otherwise, all your computations will go wrong.

(Refer Slide Time: 13:48)



So, this is a for saying it, formally there are two relations and one relation as a PRIMARY KEY which is used in the other relation as a foreign key then there is a referential integrity that needs to be maintained.

(Refer Slide Time: 14:02)



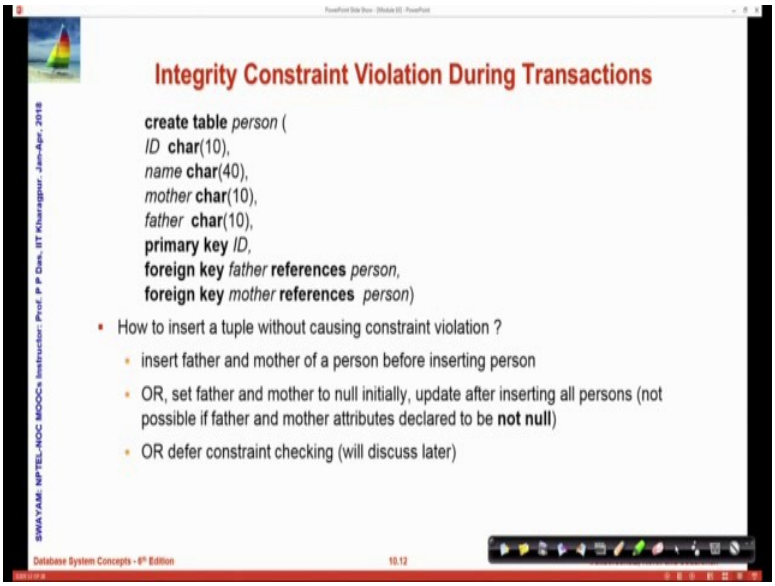
So, here we are just showing the effect of that. So, we have created a table this is the first one is what you have seen earlier creating the table course and that table course needs the name of the department. So, we are specifying that it references the department table. Now, if it references the department table, it must ensure the referential integrity. So, this

just says that this refers to the department table, but I can be more specific to say what will happen if the integrity gets violated. For example, I have created this and the course table has an entry, which has a department name say biology. Naturally, biology department should have the entry in the department table with this department name biology for this to be valid.

Now, say for some reason the biology department is abolished and that particular record from the department table is removed, naturally, the course which is referring to biology in terms of its department name that particular record will become invalid. So, we can say that on delete, what you should be doing, one most common action that we specify in referential integrity is cascade that if the referred entity is deleted then the referring entity should also be deleted. So, if you delete the biology entry from the department table, then all courses which have biology through references to department as their field value should also get deleted.

Similar thing can be there on update also. For example, biology department say tomorrow changes the name to bioscience. Now, if I have a referential integrity put on the course table as on update cascade, then as I change the bioscience the name to bioscience. All records in the table course, which had the department name as biology will necessarily get updated. So, this is the way to maintain referential integrity. Cascading is one of the most common way to handle this, but there could be other ways to take action also that could be no action that you say ok, I do not care. Let that happen in that case, because of the violation that could be some exceptions thrown or even say that if this happens then I will set that field to null or I will set that field to some default value and so on.

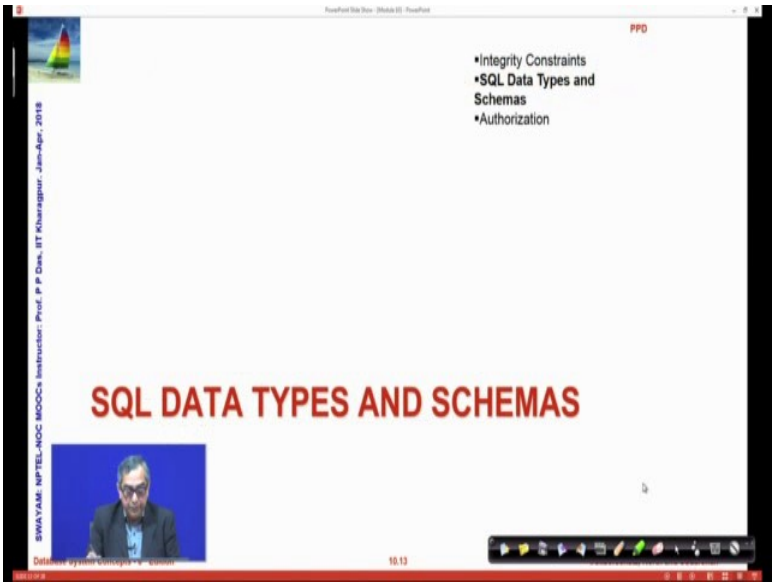
(Refer Slide Time: 17:01)



So, this is how the referential integrity has to be handled. There could be integrity violation during transactions also. This is an example of a self-referential table which of persons which where every person's entry needs the name of the mother and the father which are also entries in this table. So, necessarily if you are entering a person record you need this feels to be populated and that can be populated only if those records already exists.

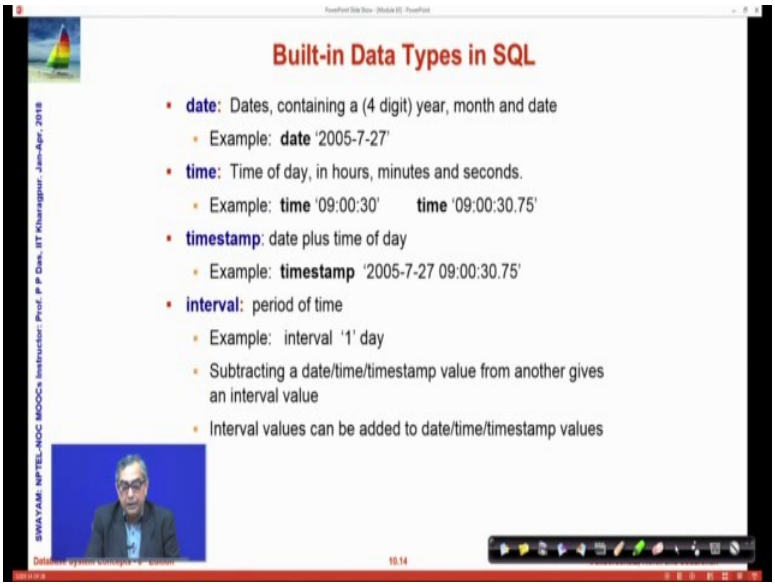
So, there is some order in which you have to enter the records or you have to set them as null and then update them in future and or some ways to say that well do not check this integrity now. We will talk about this integrity at a later point of time. So, these are the issues that necessarily you will have to be addressed.

(Refer Slide Time: 17:54)



Let us move on and look at the SQL data types and schemas.

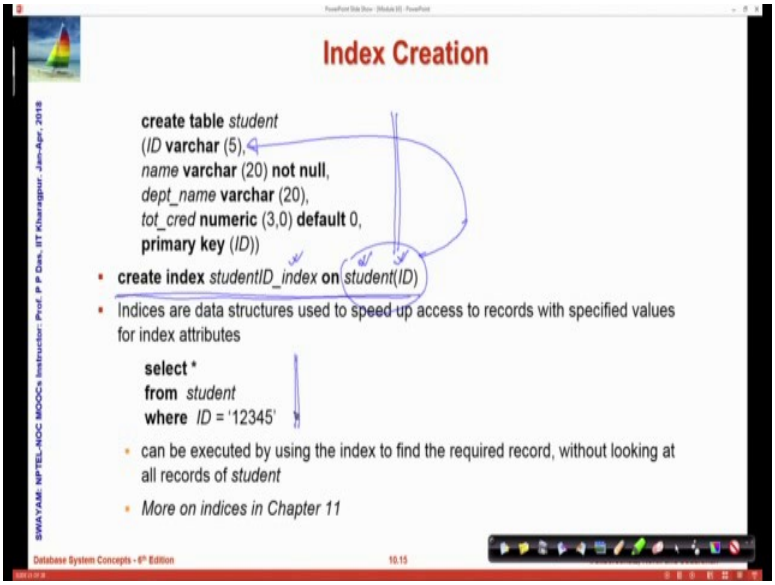
(Refer Slide Time: 18:02)



So, in addition to the data types like char, varchar, int and all that you have an explicit date data type which gives you a year, month, date kind of format with a four-digit year. because date is very frequently required, you have a time type to give you hour, minute, second time format. You have a timestamp, which is date and time together and you have what is known as interval where you can do a date or time difference between two different dates, two different time, two different time stamps and so on. So, these are the

common added built in types which makes it very easy to handle the temporal aspects in SQL queries.

(Refer Slide Time: 18:54)



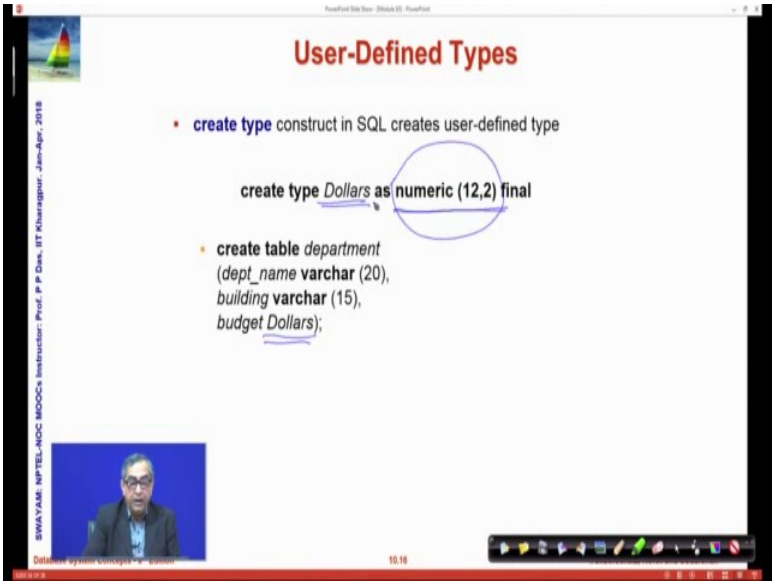
In addition, the next that you can do is you can create an index. So, let us look at this. So, this create table definition you understand well by now. You can do this, I can say create index and give a name for the index and specify which field on which the index should be created. So, here we are saying that the index should happen here. This is the name of the relation; this is the name of the attribute name of the field. Now, this does not change any data neither does it change any schema, but it creates certain additional structure so that it becomes easier to search this particular table using IDs.

So, if I have a query like this that I am trying to find out all information about a particular student then as we have said that by default the different entries the rows of a relation are unordered. So, the only way to find out this particular row and in fact, whether it actually exist would be to go over all the relations one by one. But if we index it, then it creates some kind of an efficient data structure through which it can be searched out very efficiently very easily. with a later module we will talk about indexing. But just to give you the idea that this is similar to finding out a value in an unordered array if you are thinking of C.

In contrast, we all know that this can be done, but takes a whole lot of time it takes order in time. But I could keep those numbers in terms of some binary search tree, balanced

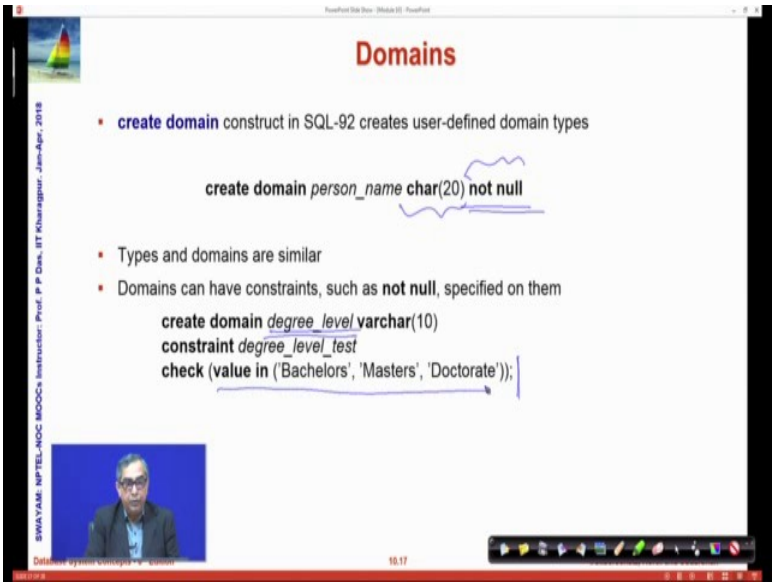
binary search tree like red black tree or two three four tree kind of, where the search can be conducted in a login time. Or I could keep it in terms of some efficient hashing mechanism where the search could happen in terms of an ordered one time also. So, indexing has a lot of importance and we will talk about that more, but this is how you create index in SQL.

(Refer Slide Time: 21:18)



You can have user defined types, you can say create type and use some specific. you know sub types of a type as and give it a name. So, you send numeric 12, 2. So, which is the 12 digit number with two decimal places of precision. you can call it dollar and then use that as a type name. So, type name doing this helps in to visit make sure that wherever you actually have to conceptually refer to dollars you are talking about dollars it is easier to understand and you are making sure that everywhere the same numeric precision is used.

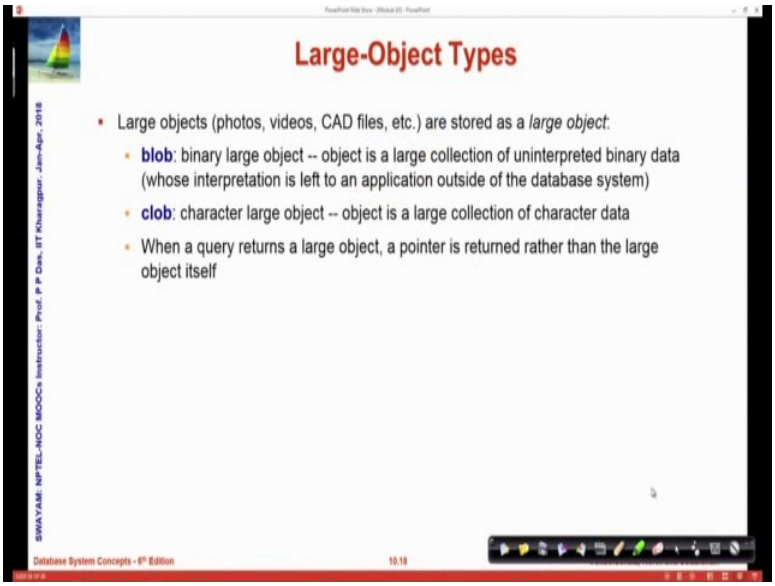
(Refer Slide Time: 22:02)



You can also actually go further and create domains which is very similar to create type. But domains are more powerful in the sense that, in a domain, you can also add constraints like NOT NULL and you say that this is person name, say that this once you have set that this person name is 20 characters long and it cannot be null then you do not specifically have to every time.

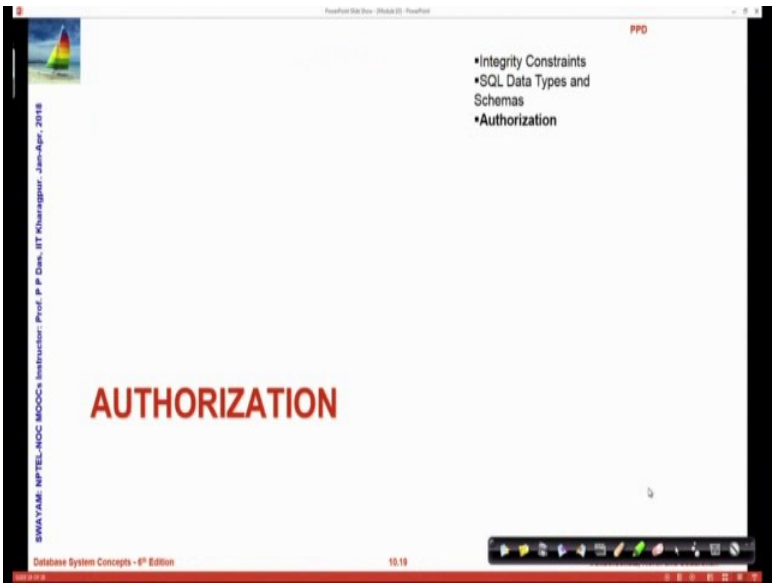
You define a field based on this domain type you do not have to specifically say that it is NOT NULL. You could also create a specific constraint in terms of the CHECK clause and make it easier. So, now if you say degree level you do not have to put CHECK clause explicitly in the SQL query, because it is already specified in the created domain.

(Refer Slide Time: 22:52)



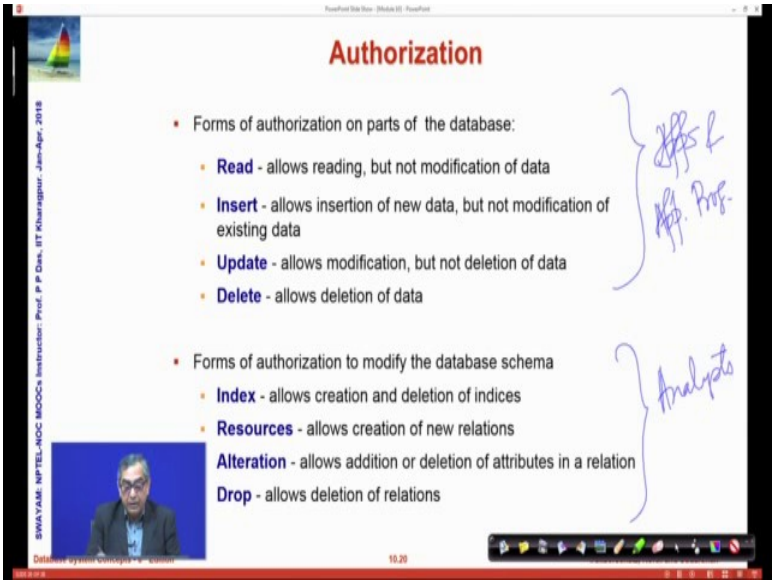
SQL supports certain large-objects which are either called blobs if they are binary, or called clob if they are character objects. The only the major difference in terms of the large object types are they are not stored as a part of the table. They stored elsewhere and you actually maintain a kind of a reference a pointer to that large object. So, this is very useful in terms of handling photos, videos and no big binary files, character files also.

(Refer Slide Time: 23:21)



Let us move to authorization.

(Refer Slide Time: 23:29)



Next, authorization is the process by which you restrict different users to be able to do different kind of operations. You recall in the early modules on database overview. We mentioned that there could be several types of users for a database. There could be absolutely application users who may necessarily do not feature as a part of the database development. But there could be application developers expectedly most of you would become application developers or there could be intermediate higher level of analyst who design databases, design constraints, decide on indices and so on and that could be database administrator.

And also in terms of different application programs and programmers there is a need to separate out who can access which part of the database. For example, if you look at a add a banking system, then while I am, by net banking application is accessing different information about my account, one part I need to ensure that I can only access my account.

And also what the database system needs to ensure is that a net banking application should in no way be able to access the information about the specific employees, because, in the same database information about the bank employees will also be there. It should not be possible for possible to access information about different physical information about the branches as to where, how many square feet of area that branch has and so on and so forth.

So, we need to put variety of restrictions and as we will see that authorisation or this process of restricting or allowing different access and different authority to operate is decided based on two different factors. One is what you want to do, and two is who wants to do that. So, what and who, so we identify different operations or different operations on certain tables or operations on certain attributes as what needs to be done. And on the other side, we will identify who in terms of specific individual user IDs or groups of user IDs or roles that exist.

So, here we will just try to show you how we can do that in SQL. So, the first part of the authorisation is being able to do different things with the database that means, the instances of the database. So, there are authorisations to read insert, update and delete. So, read is where you can access the data, but you cannot modify; insert is when you can add new data, but you do not insert rights authorisation, you cannot update an existing data, you can only insert data. You can have update rights, where we can change make modifications, but you may not be, you are not allowed to delete data, and you can that would be delete right where it allows you to delete data and my new this authorisations or not these are all independent authorisation.

So, you may have I mean certain authorisations may need certain other authorisations to be present. For example, if you are updating naturally evenly to read, but it is these are all independent authorisations, and you may have one or more of them to be able to do the appropriate actions. Similar set of another set of authorisations will exist, if you want to for those want to modify the database schema, naturally, this is primarily for the applications and application programmers, and this primarily would be for the analysts that you can index the different table you can do.

You can have authorisation for resources which mean you can create new relations, create new schemas, you can alter schemas, you can drop schemas and so on. So, these are the different kinds of authorisation that are possible.

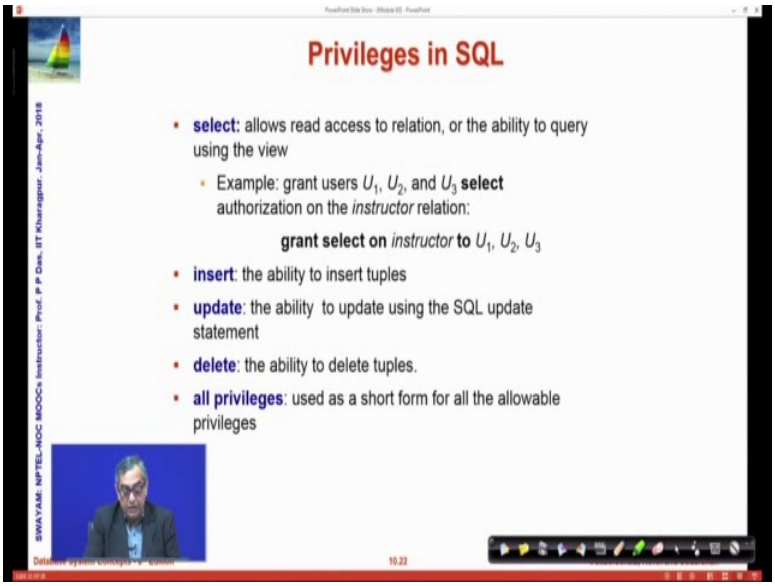
(Refer Slide Time: 28:01)

So, let us see how it works the authorisation is specified in terms of a statement called grant. So, you grant an authorisation to a privilege list and on certain relation to a group of users. So, grant what kind of authorisation you are granting that is the privilege less list on what relation on view you are granting that is on condition and to whom are you granting those. So, user list could be a specific user ID or you could say public which in this case everybody will have that or this could be a role which will see, what a role is.

Granting a privilege on a view does not imply granting any privileges on the underline relation, please mind this one, because, you have seen that a view can be formed from multiple different relations. So, if somebody has been granted a particular privilege say read privilege on a view, then it does not mean that the corresponding underline relationship, you been granted a read privilege on faculty relation that we faculty view that we did that does not mean that the user will automatically get a read privilege on the underline in structure relation. So, that has to be kept in mind.

The grantor of the privilege must already hold the privilege that is you cannot grant. Naturally, grant will be done also by somebody in some of the users you may be, so that user who is granting must also have the privilege, same privilege on the specific item. So, you cannot grant privilege on something, some relation or view on which you yourself do not have that or it has to be the database administrative who naturally has privilege for everything.

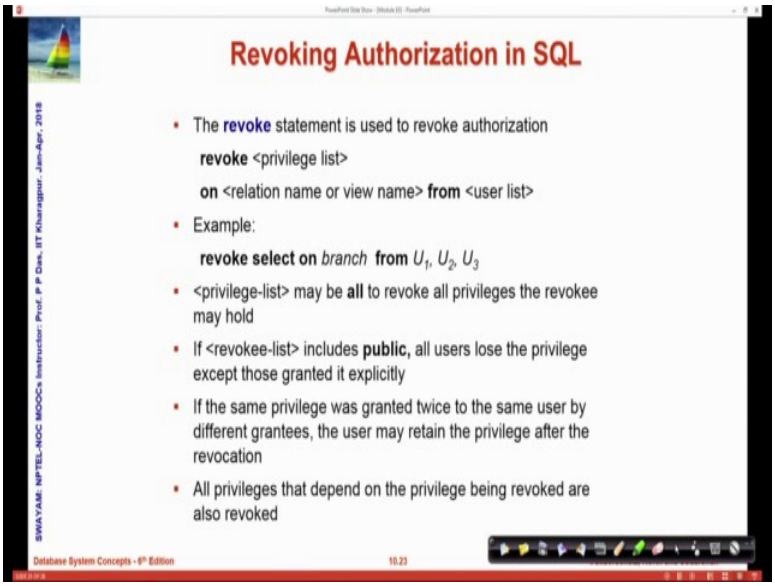
(Refer Slide Time: 30:13)



The privileges on SQL there is a SELECT privilege, which is so you know this is the privilege list. SELECT privilege which basically means read access. So, it is saying SELECT privileges is read access on instructor and these are the different users. So, this is how typically, you can have insert to ability to insert tuples, the update privilege this should be easy to understand now delete privilege.

So, only thing is read is a called SELECT here, so these are the different privileges that you have in a SQL. And you have one all in compassing privilege which is called all privileges so as a short form of allowing all these allowable privileges.

(Refer Slide Time: 31:00)

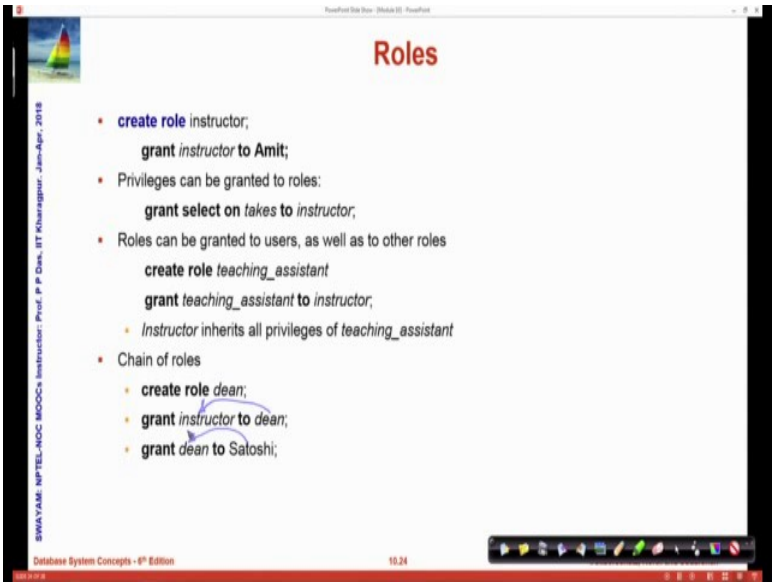


Certainly, if you can grant an authorization, then needs to be a reverse process that is if we want to withdraw authorization of certain privileges on certain items for certain users, so that is known as a REVOKE statement. So, we can revoke it the structure looks exactly similar to the grant. So, you REVOKE a privilege list, SELECT, insert this kind of on certain relation and view from a set of users. So, you can say that REVOKE SELECT ON branch from this. So, once this is done, then U_1, U_2 and U_3 will not be able to read the branch relation or the branch view. The privilege list may be all to REVOKE all privileges. So, instead of revoking SELECT, insert separately, you can just say all and revoke all of that.

The list of revoking can include public which means all users lose that privilege and or though those were granted. If the same privileges granted twice, now it is possible that a user gets privilege granted to him or her by two different granting authorities, then if ones is one of them is revoked the other will still continue to remain; So, every privilege that is granted needs to be explicitly revoked that is a basic meaning.

So, all privileges that depend on the privilege being revoked are also revoked. So, some privilege which is dependent on some other privilege, if you REVOKE the update privilege then this SELECT privilege will remain. But, if you REVOKE the SELECT privilege then if you also had the update privilege that will certainly get revoked, because if you cannot read then naturally you cannot change.

(Refer Slide Time: 33:01)



The SQL also allows you to create certain roles. Roles are kind of like virtual use that so, we all say that we all play certain role. So, I have an entity as an individual say I may be user called PPD, but I have a role as an instructor, I have a role as say the head of the department, I have a role as a chairman of committee and so on. So, often times it becomes easier to grant privileges to different roles.

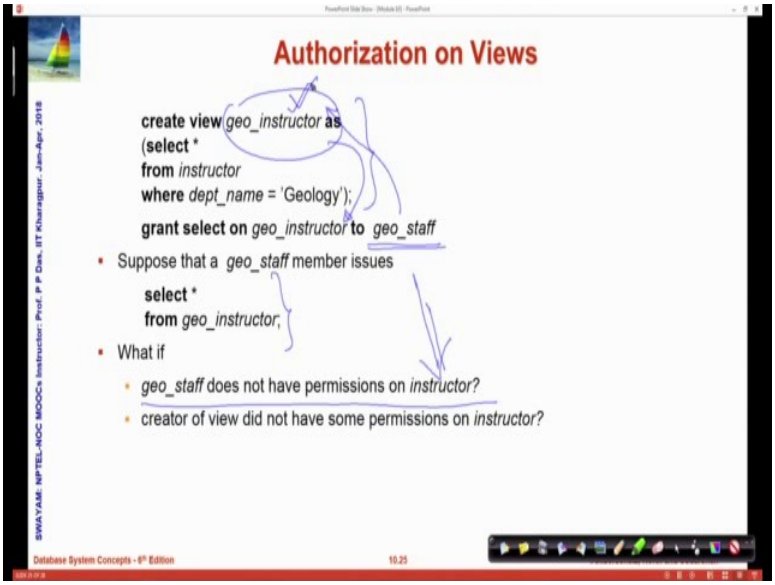
You do not really care immediately about who that individual could be who that particular user could be who has that privilege, whoever plays that role gets that privilege, whoever becomes the director of IIT Kharagpur has the privilege to appoint faculty members it is of that kind. It does not specifically. So, role is of that kind of a concept. So, you CREATE ROLE we are saying that role is the role instructor is created.

And then you are saying that you grant instructor to Amit which says that Amit now plays the role of instructor. So, any privilege that the instructor role has Amit will enjoy that. So, let us see more of this the privileges can be grant to roles. So, earlier we said it could be public, it could be users, but now you are saying that it could be two roles. So, here this role was creates and the privilege is being granted to that. And since Amit plays that role it will mean that Amit with this GRANT SELECT ON takes to instructor, Amit will actually get a privilege of SELECT on takes relation that is the kind of derived structure that roles give you roles can be granted to users as well as to other roles.

So, roles are becoming like virtual users. So, you can CREATE ROLE teaching assistant and grant teaching assistant to instructor, which means that if you do that you are granting this. So, it means that any privilege the teaching assistant will have, the instructor will get those privileges, because, you have made in instructor to also play the role teaching assistant mind you instructor itself is virtual entity. So, if Amit is an instructor by this, then Amit plays this role and this role plays teaching assistant role and this teaching assistant role has certain privileges, so naturally through this chain process Amit will get those privileges. So, in an instructor inherits all privileges of teaching assistant.

So, this is what exactly, what I was talking of you can have a chain of roles create a role dean new one, you have created, then grant instructor to dean grant dean to Satoshi. So, which means; that once you grant dean to instructor; so anybody who plays the dean's role will get all privileges of instructor. Here you are saying that Satoshi is going to play the dean role. So, the Satoshi in terms of chaining gets all the privileges that instructor has.

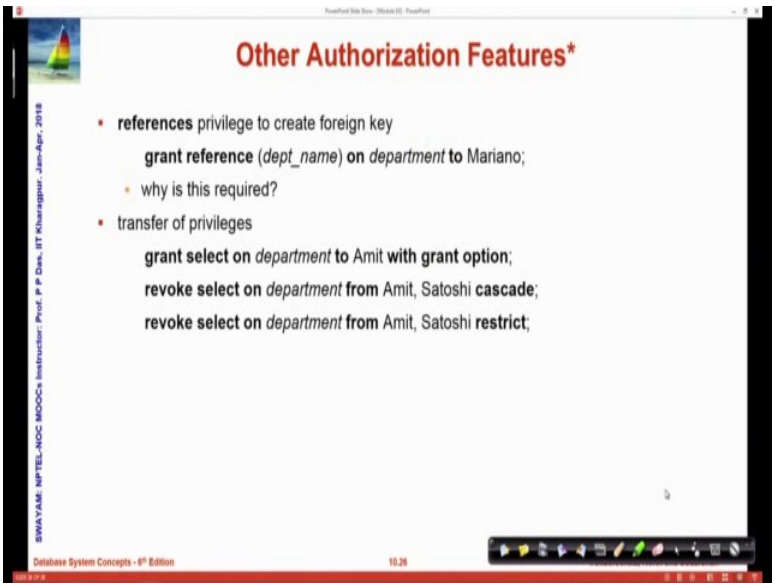
(Refer Slide Time: 36:41)



So, once this has been done, then you can have authorization on views as well. So, you have created a view here. So, this is the view created the geo instructor the Geology instructor. And on that view particularly you have given the privilege to the geo staff. So, a geo staff member would be able to access this view. And if this query is fired by a geo

staff member, which I am assuming is a role then this view will get executed and the results of all instructors in the department geology will be update. But, what if the geo staff does not have permission on instructor, does not matter that is the beauty of the whole thing. The geo staff may not have permission to do SELECT on instructor, but the geo staff has permission to SELECT on geo instructor. So, the geo staff will be able to execute this view, but the geo staff will not be able to do a SELECT on from the instructor database instructor table.

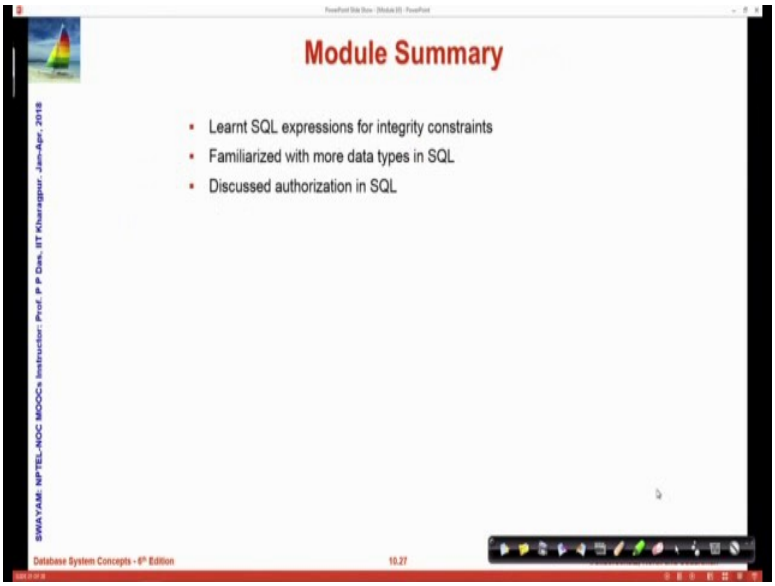
(Refer Slide Time: 38:16)



There are several other authorization features, the references a privilege to create foreign key. So, we talked about basic read, write, data manipulation privileges, but there could be other privileges like whether you can create a foreign key, whether you can transfer of privilege. So, whether you can give one privilege to another, so whether you can cascade, whether you can restrict and so on.

So, transfer of privileges is also privilege. I am naturally will have to think of this is an authorization. So, actually what we can authorize is also a privilege that needs to be authorized. So, these are the derived privileges that I have.

(Refer Slide Time: 39:00)



So, in summary, we have learnt about SQL expressions to deal with integrity constraints. We are familiarized with more data types particularly user defined types and domains creation of index and we have discussed about authorization in SQL. Each one of them particularly authorization has lot more details, but at this intermediate level we just wanted to get a basic idea about authorization to be able to deal with that.

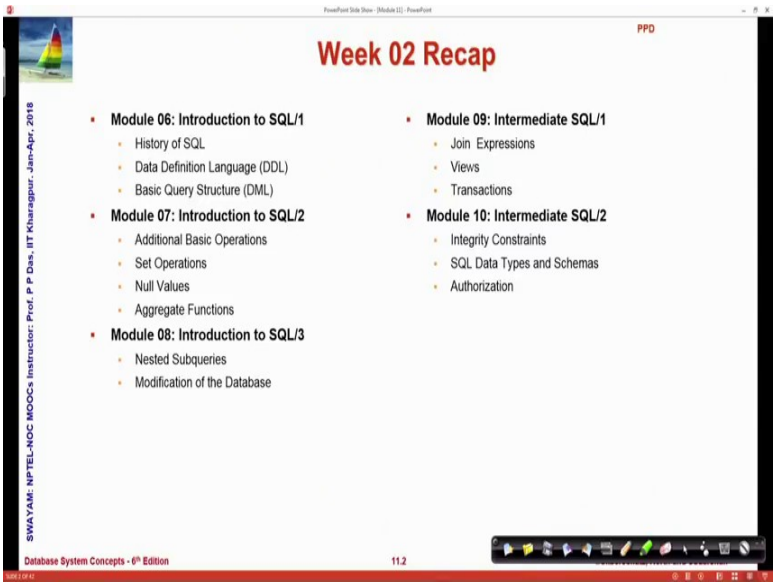
So, with this we close our discussion on the intermediate level SQL features. In the next module that we start next week, we will talk about some of the advanced SQL features.

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

Lecture - 11
Advanced SQL

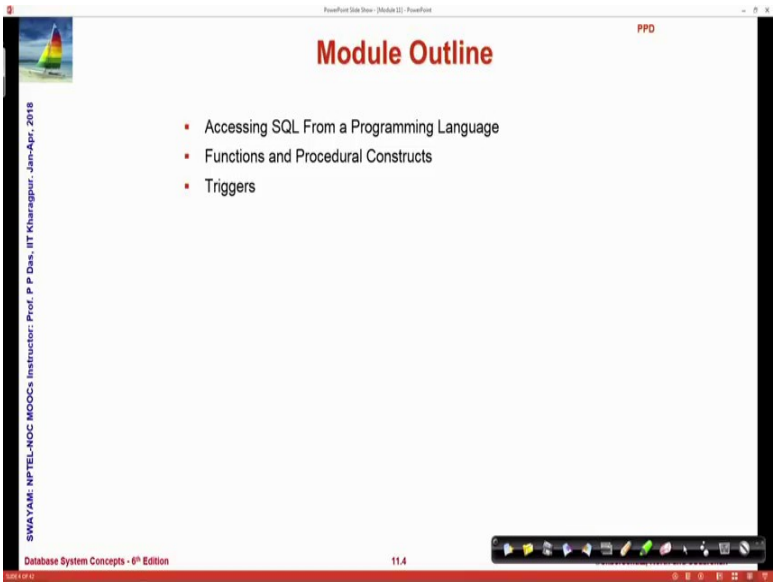
Welcome to module eleven of database management system. This will be on advanced SQL.

(Refer Slide Time: 00:31)



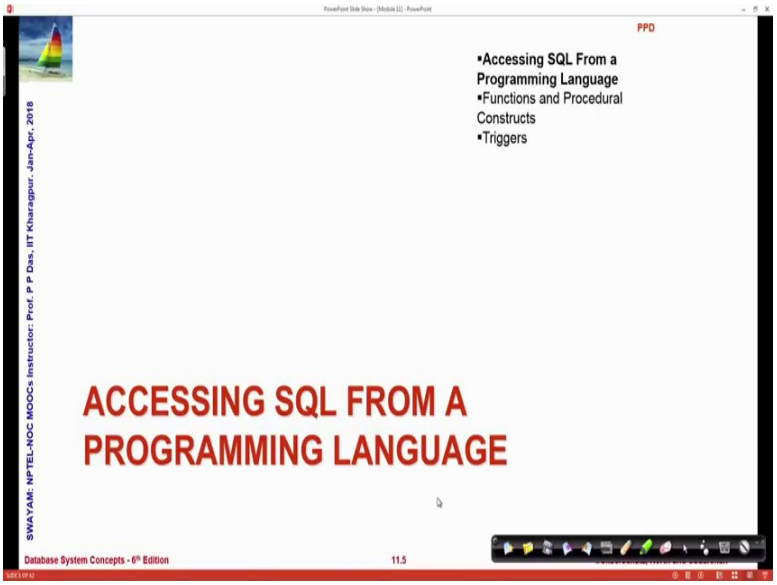
Before we start let me quickly recap what we did last week in the five modules. Last week we totally spent on discussing first the introductory features of SQL how to create data and how to write basic queries, and we studied about all different kinds of SQL operations at theoretic operation, handling of null values aggregation, nested queries and so on. And then we did an intermediate level of SQL query formation in terms of joint expression, views and integrities different kinds of SQL data types and importantly authorization.

(Refer Slide Time: 01:24)



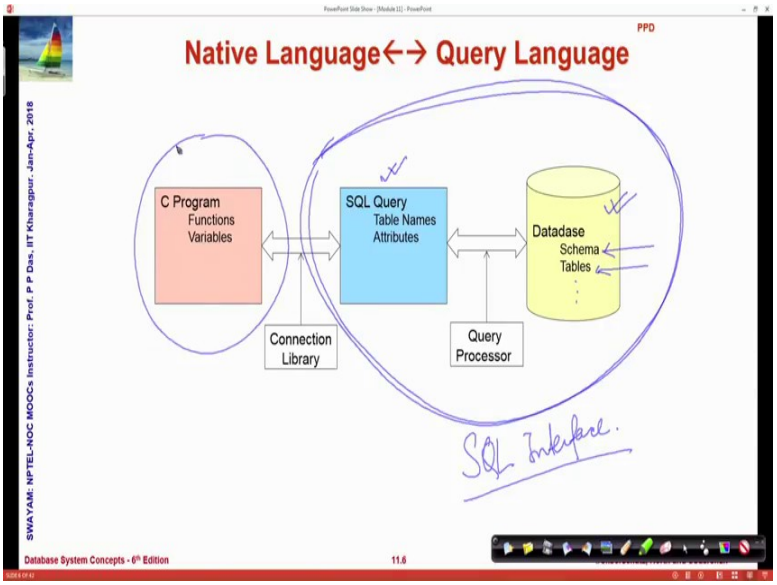
In the context, in this context, we now take up some more of the SQL features which are somewhat advanced. And we will try to understand how SQL can be used from a programming language, and familiarize with functions and procedures in SQL. This will violate some of the basic premises that we started with in saying that SQL is a declarative language only because as you understand functions procedure as procedural language features we will see how to handle those, and we will take a look into another important feature of triggers.

(Refer Slide Time: 02:07)



First with accessing SQL so, this is the module outline accessing SQL from a programming language.

(Refer Slide Time: 02:11)



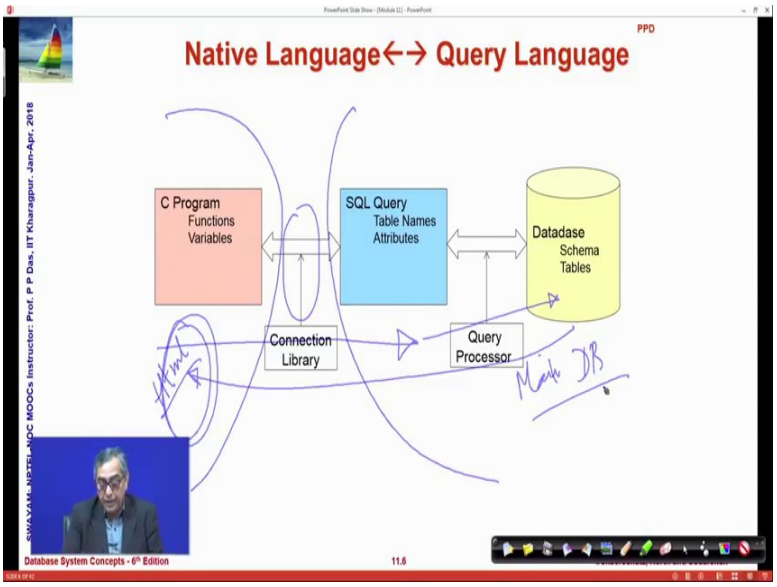
So, this is just kind of an abstract view that you can think of that what we have been doing so far naturally there is a database which has primarily two things schema and tables. Of course, there are many other things there are index, there is authorization that triggers and all that, but there is a database which stores everything.

And so far we have been dealing with only this part of the information that I can write certain SQL query to define table, using table names, attributes and certain logic and that goes through certain query processor works on the database to get me either create the desired effect. Either that is creating table instances or creating index or extracting certain relation values, defining some views and so on. So, all these have to happen through an SQL interface. So, this has to happen through an SQL interface.

So, whatever system we are using whether you are using mySQL, Postgres, Oracle, Sybase SQL server whatever you using that has some interface through which these SQL queries can be executed, so that is kind of a standalone complete system. But, in general, we will have lot of more requirements in terms of the application. For example, the application might require some graphics, SQL does not have support for graphics; application might require certain numerical algorithms to be executed might require some geometric computations to be done poly intersection of polygons to be computed

and so on and so forth. It might require some network programming and so on. So, there is a need to do all these and certainly these are best done in terms of certain native language that could be C++, java, C#, visual basic, python any of the native languages which has support for a variety of different tasks.

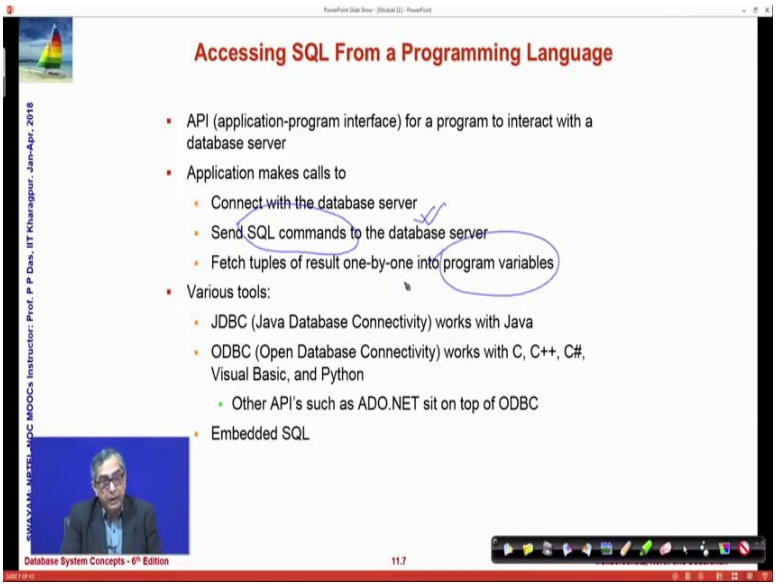
(Refer Slide Time: 04:20)



So, what is critical is can we make a bridge between these two that is can we take the advantages of the SQL domain and the advantages of the normal imperative procedural programming domain together. That is I can do some graphic representation and then certainly go to the database extract some information and then present it in the graphics and vice versa. For example, whenever we access say gmail, we get to see them in terms of an html presentation which is some kind of a graphics rendering, but the all the mail entry certainly come from some database.

So, somewhere there is a connection by which when I say get my inbox mails and somewhere the information goes over from this to the SQL query up to the database and the result is brought back to me. I see that in the html page here, but here there must be certain mail database, where all these information exists. So, what I am trying to come at is it is critical that the application programs or you know high level programming normal programming languages native languages should be able to interface with SQL. And what we discuss here is two different mechanisms for this interfacing. The first one is using a connection library and this is what I will specifically show you.

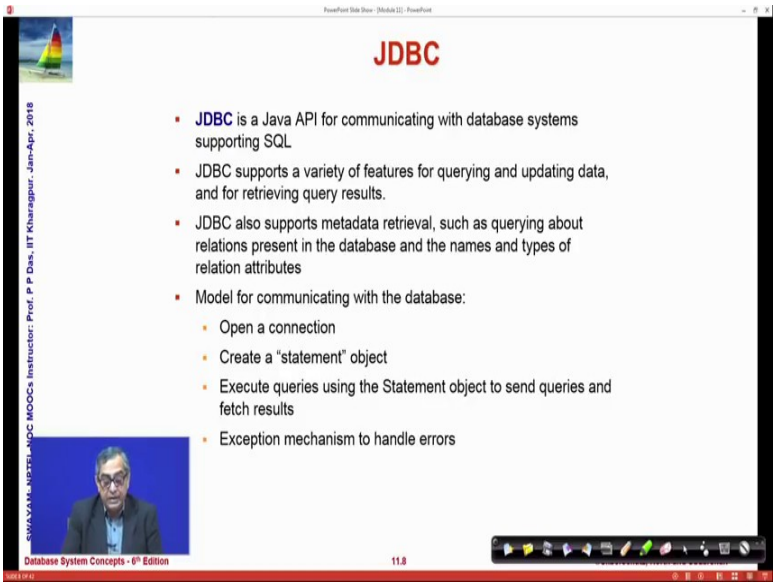
(Refer Slide Time: 05:50)



So, for if you are using a connection library then you have a set of APIs application programming interfaces these are basically functions in that library. So, that with that application can connect to the database because certainly the database is somewhere else is a different server. And send an SQL command to the database server so that you can say that this is what I want. And then a result is computed the result of that computation the table the can be brought back those tuples can be brought back to the application program. Mind you when we are here when you are sending the information in terms of the database server, we are talking SQL command, we are talking about attributes tables of the SQL space.

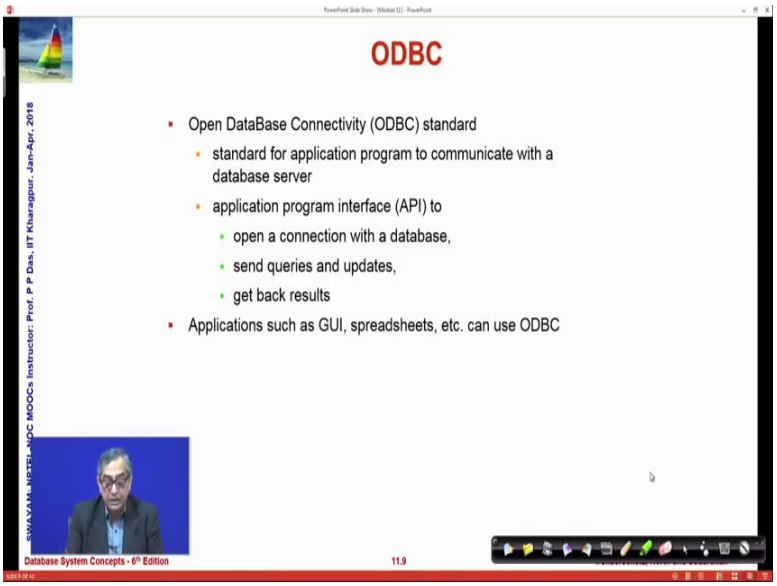
Whereas, when I want it in the program I want it in terms of program variables. So, there has to be certain correspondence made between them. There are a variety of tools available which allow you to do this JDBC is common very commonly known which is specific for java. We have an open database connectivity APIs which has different versions for different languages these are the common languages that it is used with. And as we will see later on that there is another mechanism for doing the same thing called the embedded SQL, we will come to that later.

(Refer Slide Time: 07:15)



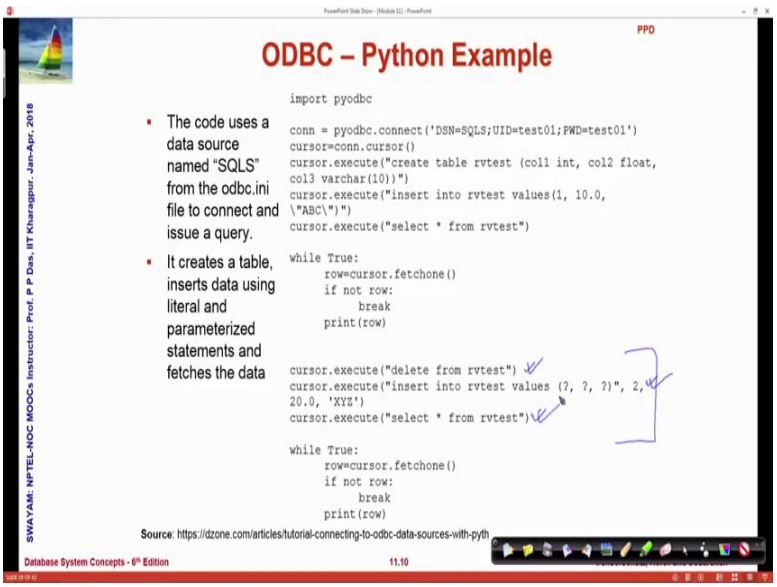
So, JDBC is a java API, I will not go into details here if you know java you should be able to quickly look up. So, it is a it communicates with the database by opening a connection creating what java calls a statement object and executes the query using the statement objects and that is used to send the query as well as to get back the result. So, java is object oriented as you know so statement object is used as a as an encapsulation which travels between the java program and the SQL query processor. And since the query gets back the result since the query gets back there is that kind of there is exception mechanism to handle errors which is common for java.

(Refer Slide Time: 08:00)



What you the see in contrast does a similar thing, but since it is to cater for different languages it has got a softer model it has got less powerful model. It is a standard application program to communicate with database server. So, again it has to open a connection to the database, send queries and updates and get back results. So, these are these are the three basic things, these three other three basic things that certainly needs to be done if I want to easily work across the application programming domain and the database programming domain. So, applications such as GUI, spreadsheet, etcetera can use ODBC.

(Refer Slide Time: 08:39)



So, yeah I am just quickly showing you an example we will talk about application programming mode in a in a later module. So, this is a python example. So, python for this the ODBC for python is known as pyodbc library. So, you need to import that. And then using that you can connect. So, if you have to connect you have to say which database who is the user, what is the password, because authentication needs to happen. So, SQLS is a database here and with this user with this password is connecting you that is successful you get a conn object and on the conn object you have something what is known as a cursor. Cursor is nothing but if you think about it is it is kind of a pointer. So, it can be used to point to either a row or a whole query or a table.

So, you get back a cursor object. So, then this is if you look into this part, this part is nothing but a pure SQL query. So, you take that as a string and pass it on to the execute

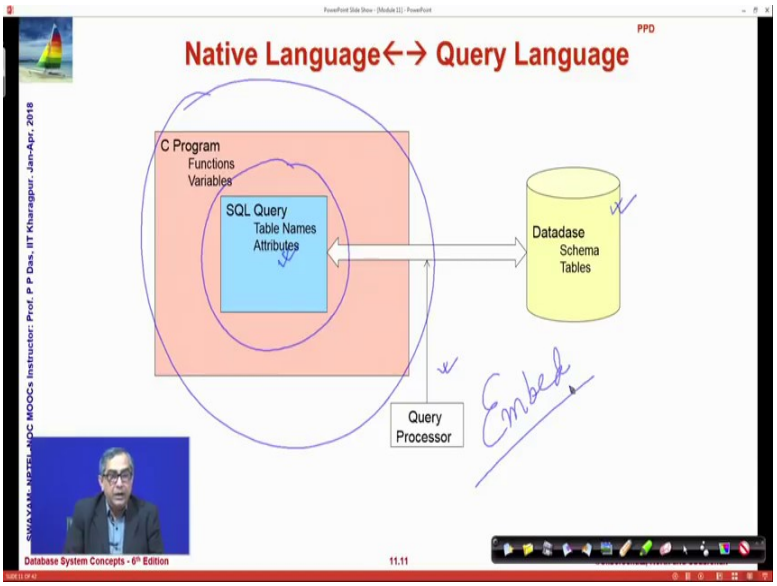
method of cursor. So, what it does is, so this is this has done the first task which is connecting to the database. Now, you are basically putting the query to the database in terms of doing saying that this execute. Similarly, so if that will get executed, so the table is created naturally there is no result to get if you have created a table. Now, you do an insert. So, again this particular record, you can see that within this double quote, this whole insert syntax you take that as a string and just give it to execute as a parameter.

In the third, that you do is do a select. So, you have done this we have inserted a record, created a table inserted a record in that and then now you are doing a select. So, certainly we expect one record to come back. So, these are the SQL executions and then your get back result. So, cursor has another method which is known as fetch one. So, what it will do that if your result table has multiple entries, then the cursor will if will be will start with the first row and fetch one will bring the whole of this first row as a row vector.

So, it will bring in as all the components as one as a python string. And then you check whether it is empty or not, if it is empty then the I mean there is nothing to bring back, so you are done. So, you break otherwise you simply print the row, so you are printing there. And then you go back again this is while true. So, what happens is the cursor will advance to the next row and get you the next row. Again you do the same thing go back it will come back to the same, but once you get an empty result, you know that there is nothing more to proceed and you break.

So, this is what is illustrated then there are some more this particular record is deleted, then again another record is inserted. And another select is done. So, this is how a native program here in this case python can interact with the database and do any of the SQL tasks that we were doing earlier with SQL interface, now we can be done from the python, so that is a basic ODBC mechanism. I particularly chose python to just give you a different flavour, you can we will have assignments on doing it for C and using jdbc on java as well.

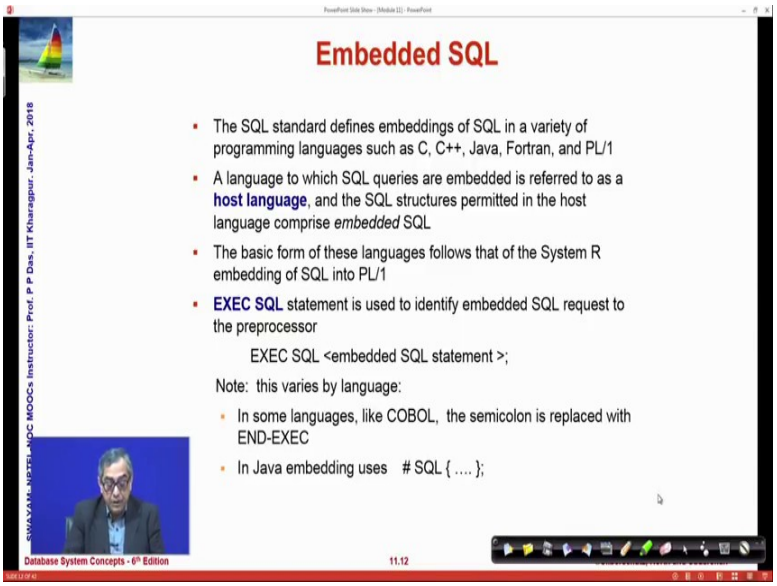
(Refer Slide Time: 12:19)



Now, I am come I will come back to the same interaction issue, but this time you can see that I am using a different diagram. The database is the same; this part is the same; the query processing is same, but instead of having a connection library, now I have put the SQL query itself as a part of the native program, the C program. So, this is what is called embedding.

So, you say I embed I put the SQL as a part of C, but naturally SQL is not C. So, we need to put certain additional syntax in C, so that I can directly write SQL as a part of the C program. I am talking about C, again just as an example if this is true for other several other languages which can be used as used for embedding SQL within them.

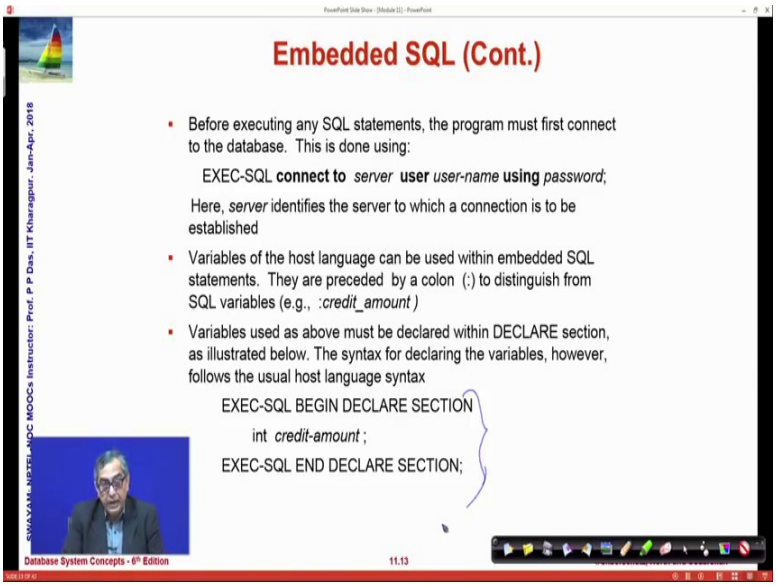
(Refer Slide Time: 13:09)



So, this is called the embedded SQL, it works for C, C++ , java fortran etcetera. And the language native language in which your embedding is called the is known as a host language. And basic form of these languages allow that the are come from the System R. So, what is important is this particular statement EXEC-SQL. This EXEC-SQL written inside the body of a C program will tell the C compiler that this part is not C; this part is actually embedded SQL.

And it will be treated differently; it will be compiled by the SQL compiler within the C compiler, so that is the basic structure, so EXEC-SQL. And then you put the pure SQL statement the embedded SQL statement. So, let us go forward and see some of this, there are different syntax for different native language embedding.

(Refer Slide Time: 14:11)

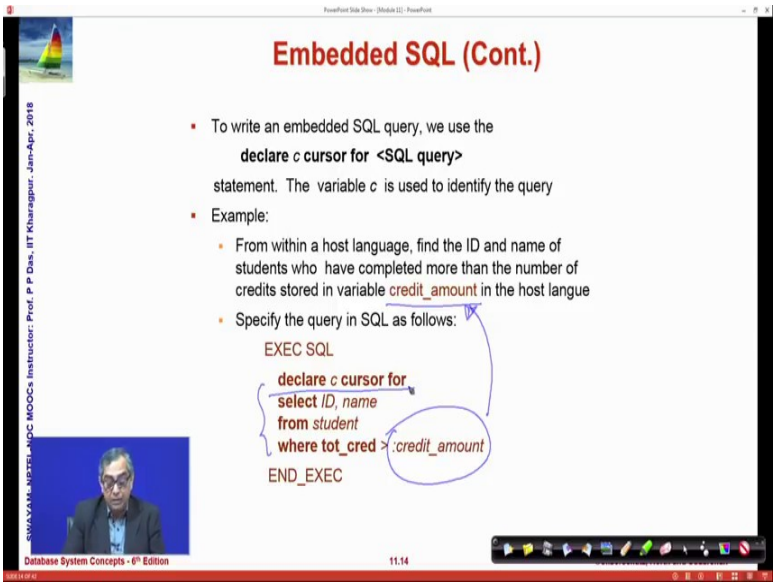


So, to be able to connect you say EXEC-SQL and connect to server username using password. So, we saw similar things in terms of ODBC based connection these three information needs to be specified which database server, who is the user, what is the password. So, here you say that in this form. And this particular statement you can write as a part of the C program. Now, naturally the question here is in the in the earlier case when we are using the connection library, your results came back through the cursor which you then could deal because the cursors are necessarily objects in your native language. So, in python the cursor was a particular object in the pyodbc library.

But now you have embedded the SQL in terms of your c program. So, naturally the results or whatever you are doing in C which needs to have a communication with the SQL statement need to be differentiated from the SQL names themselves. So, any native language variable that needs to be treated in SQL will be marked with a colon preceding it. So, credit amount of this a colon credit amount, so it becomes a SQL variable provided credit amount itself is a C variable, so that is the basic you know connection mechanics. There are far more details in that, but I am just giving you the glimpse.

Now, any region where you write the SQL, you can write it as exact SQL begin declare section, and END declare section this is where you specify what are the different what are the different C variables C declarations that need to be used for this SQL definition. I will just show you an example soon so that.

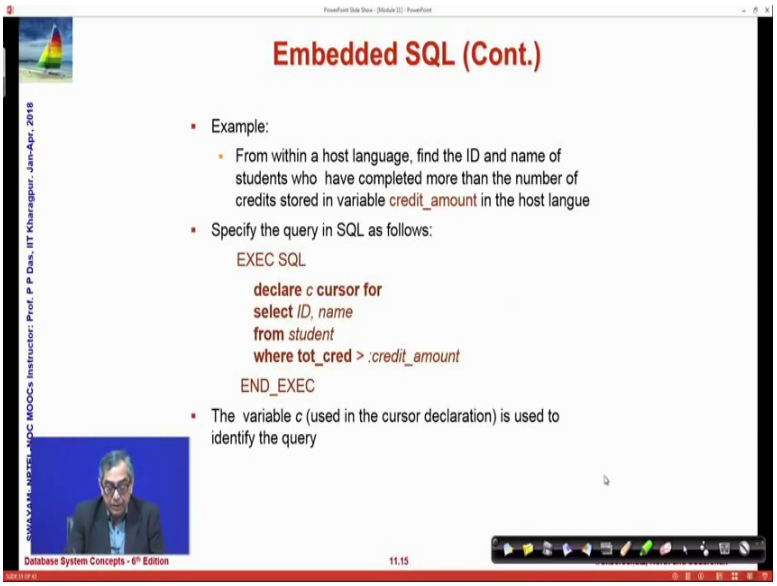
(Refer Slide Time: 16:15)



Similar to the ODBC style, you have a you will declare a cursor. So, this is to write the embedded SQL, you use declared c cursor for such and such SQL queries, so then that C, variable C will become your handle in the in the C language to be able to answer handle the query results. So, here is an example. So, you can see that credit amount is a variable in the host language.

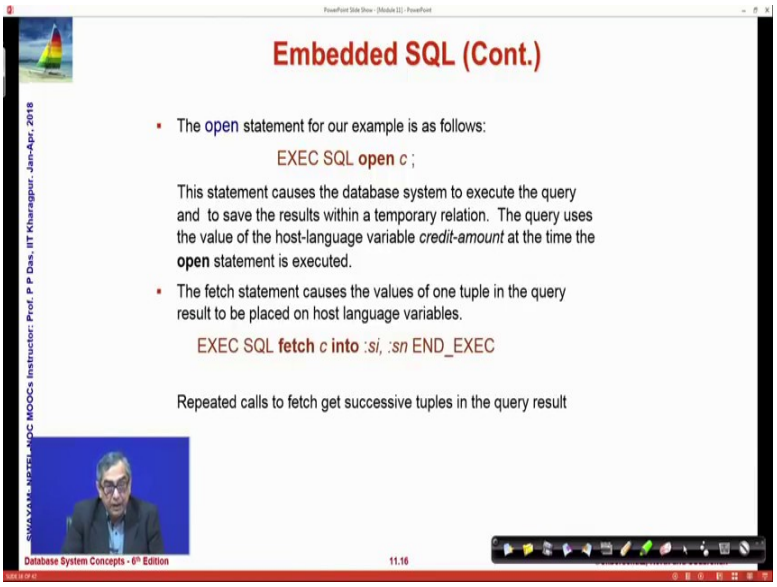
So, you are using it in SQL with colon credit amount which says that it is this host language variable. So, that you can set a particular credit amount and go with that. And this is the query, and you have set a cursor on that which you can make use of in the exact SQL.

(Refer Slide Time: 17:14)



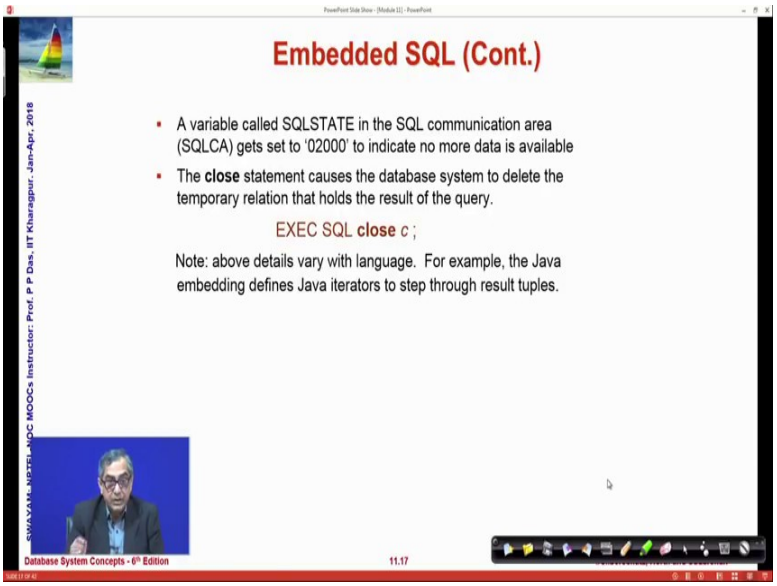
So, let us this is the example continued.

(Refer Slide Time: 17:18)



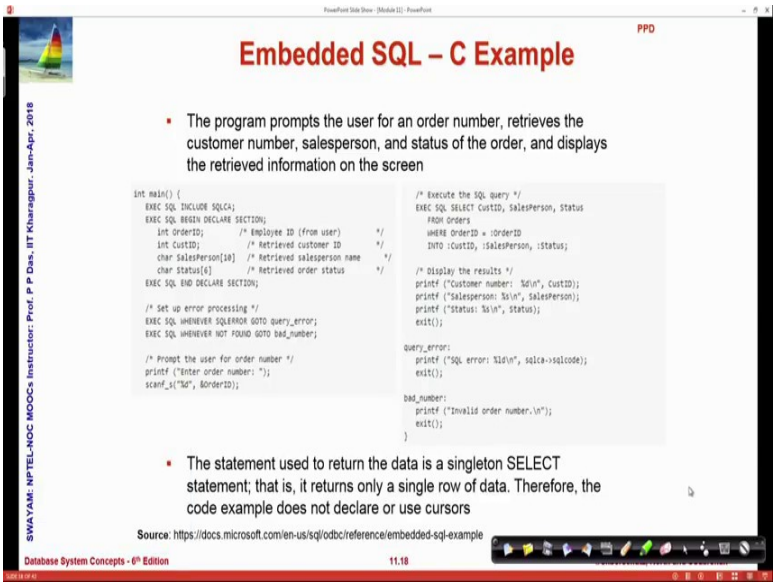
Let us look at other features. You can once you have set a query you can actually execute that by the open statement. So, you say EXEC SQL **open** c, the cursor. So, that will execute the query that you have associated with that cursor. And then once that has been done, then you can fetch the results into that cursor one by one; one tuple at a time.

(Refer Slide Time: 17:47)



Once you are done with all that then you simply close.

(Refer Slide Time: 17:52)



So, let us look at a example. This is a program which will which prompts the user for an order number, and retrieves the customer number I mean given an order number it will retrieve the customer number, salesperson, status of the order and it will display that as the retrieved information on the screen. So, here is a C program. It starts on here with the main. So, you can see that this says that EXEC-SQL **INCLUDE** SQLCA, SQLCA is the communication area. So, there is a there is a exchanges going on between the c program

and SQL program. So, the area that is used by both for this transaction is known as SQLCA.

Then you have the declare section. So, you are saying these are SQL exec declaration. So, these are, but within that what you have are pure c declaration, but all the declarations of c that are put within this can be used in SQL query with a colon at the beginning. So, that the value can be exchanged to the SQLCA then in the next you are specifying what will happen if you have an error.

So, you say SQL look at this EXEC-SQL whenever SQL error go to query error. So, it will go to this level. If it is not found that is no result is there it will go to this. So, you by making sure that if there is some error that happens in the SQL part, what will happen in your C program. And then subsequently you have simple C program which reads the order number, and after having read that you are doing this.

So, what does it do, EXEC-SQL is to say that this is embedded; this is the select query starts, the fields, the relation, the condition. And then there are three attributes. So, you are setting an association with three variables in the C program. So, if I want to the result of this select will be a table of three attributes three columns, so we are giving a name to each one of these three attributes in terms of our C program.

So, there is a cust id. So, I say the cust id attribute in SQL is colon cust id here which is basically cust id variable in the C program. Let me clean up again. So, this is cust id; this is in SQL; this is my C program, and this is my C program variable in SQL which corresponds to this its similarly order based. Similarly, this is in SQL attribute this is a array of character here. And this is a correspondence; this is a correspondence.

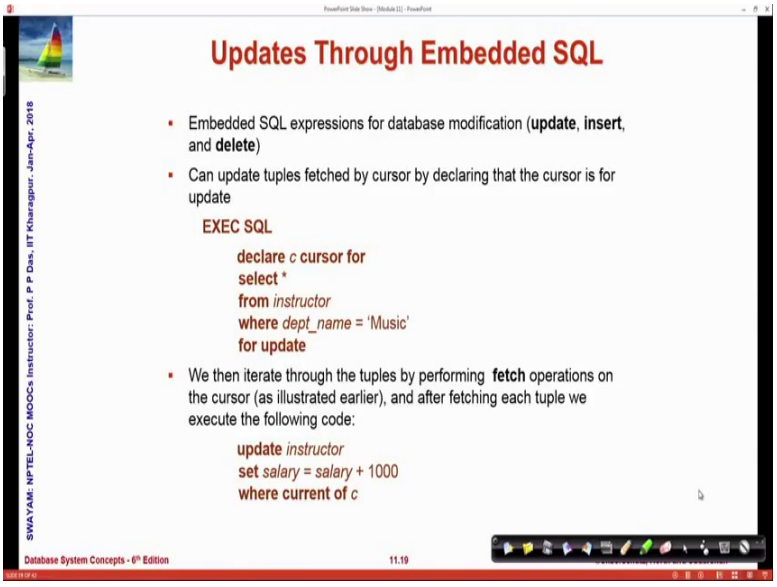
So, now, you can easily see that once this has been done I will be able to get all these values here, normally the program would be longer the you will have to iterate over the cursor as you did in case in the ODBC case. But in this case since we are using one order number we know that there will be only one record. So, we have not shown the iteration on the cursor.

So, once you have got this you are you are using those values to print out the result. And in case this query has got into some problem because of SQL, then it will automatically jump to SQL query this particular level. If it has got a bad number which means that no

such record was found, it was a null table then it will immediately take you to this. So, this is how the embedded SQL worked.

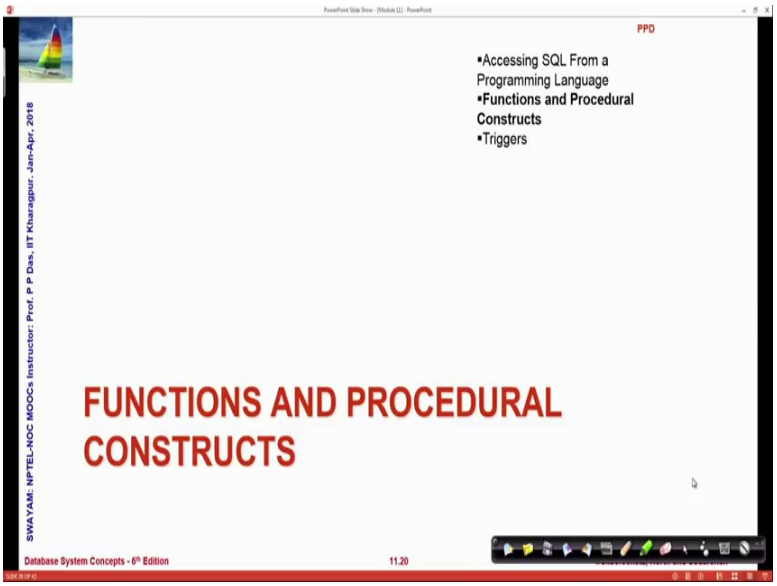
So, there are these are two different styles the ODBC style and the embedding style are two different styles. If you have if you depending on the preference you use that earlier days people used to use more of embedding, I believe that now the preference is more for the ODBC kind of connection oriented system ODBC JDBC kind of because they are certainly more programmer friendly this.

(Refer Slide Time: 22:05)



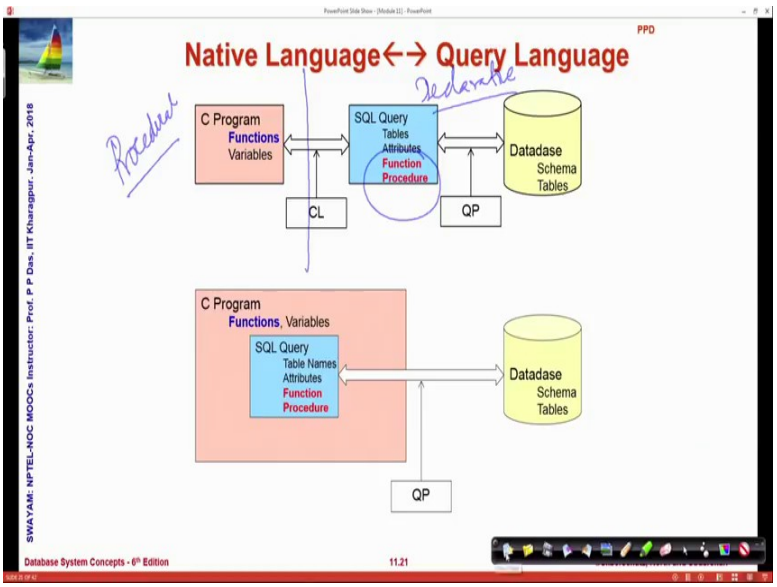
You can do updates through embedded SQL as well. So, I will not go through the details you can just go through this and understand that.

(Refer Slide Time: 22:15)



Let me move onto the next advanced part of SQL which is function and procedural construct.

(Refer Slide Time: 22:27)



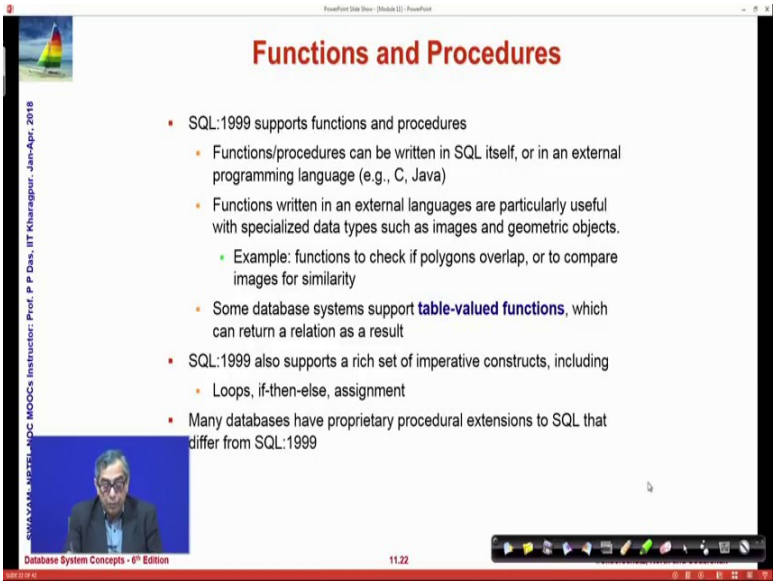
Now mind you again this is these are the two models that we have I have already shown you. The two models in which the application program the native language program and the query language can interact the ODBC mechanism and the Embedded mechanism. But what we are now empowering is so far our basic premise was that this site is a procedural I discussed this at the very beginning. And this side is declarative. So, in SQL

you do not say that how you find out a result, you say what you want as a result, these are more like predicates. And in C program, you cannot specify what you want as a result you would rather say that do step one, step two, step three and you will get this result.

So, C program is all full of functions again I am talking about C as a placeholder it is true for most of the programming languages we use otherwise. So, there are procedural languages. So, procedures in C are functions; whereas, in SQL you had select from where kind of conditional clause.

Now, what we are saying that SQL also in later version have allowed certain functions and procedures, which can be part of SQL. It also has allowed certain imperative constructs like case like loop like while, repeat those kind of to make certain kind of procedural programming easier in SQL. And naturally these again can be used in conjunction with the connection oriented applications with the native or embedded oriented mechanisms. So, we will just take a quick look into some of these function and procedural features.

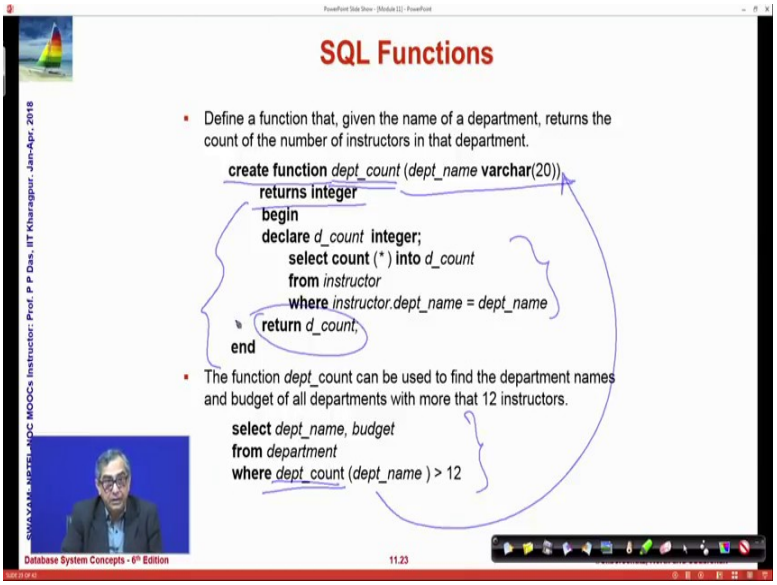
(Refer Slide Time: 24:15)



So, this started coming in from SQL 1999. And you can have functions and procedure written in SQL itself and function and procedures written external language or the host language also. So, both of them are available. What is interesting is some database systems support functions which are table valued. Functions, we always know functions

written objects only, but in SQL you can have functions which have table valued which written new functions. So, and certain imperative constructs have come in.

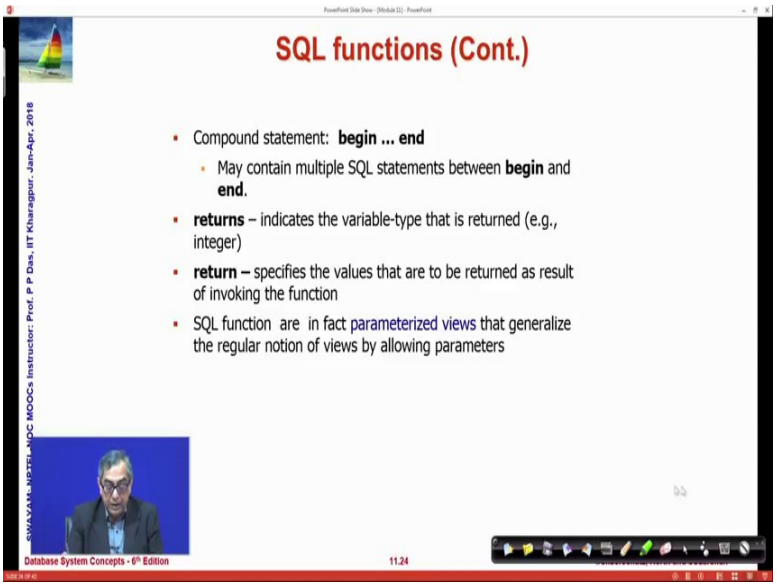
(Refer Slide Time: 24:56)



So, there are several databases have their proprietary constructs and all that also. So, at this level of the course since we are not focusing particularly on any specific database systems, we will not talk about those. We can do enough in terms of the standard SQL itself. So, this is how you define a function which looks very similar to the SQL definition, create function, give it a name certainly here are the parameters. And here is a return type. So, if you are familiar with C, C++, Java you I mean it is just that the syntax is different, but the elements are same.

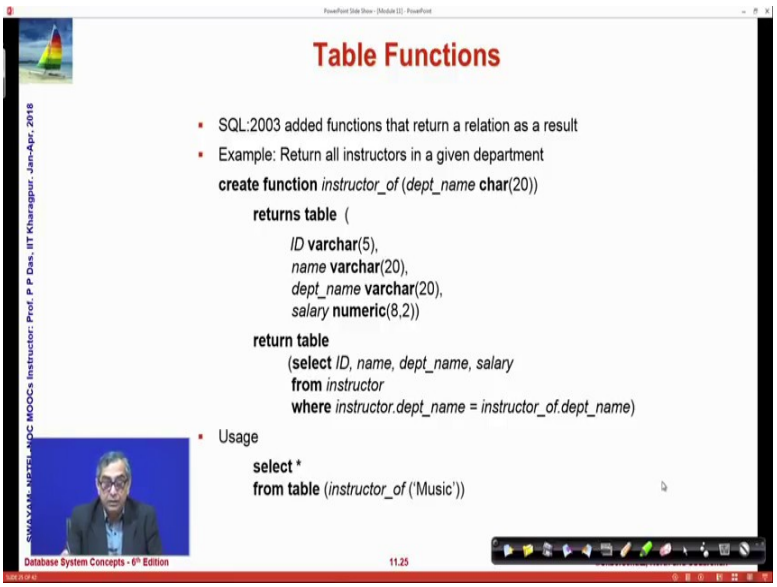
There is a begin end in the scope. And this is the pure SQL that you are writing here. So, this is a function which is not a function in C, this is a function in SQL. So, this you can write this function in SQL. And once you have written that function then you can actually use that. So, if you look at the function is department name, then separately I am writing a query, I am using this department name as function. So, whenever I whenever this query will get executed, this function will be called, and the corresponding values will get returned. So, this is the basic mechanism ok.

(Refer Slide Time: 26:16)



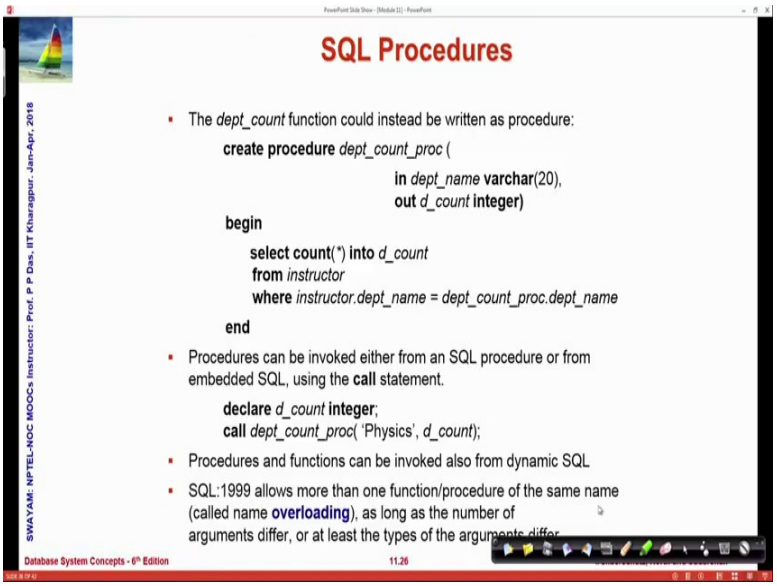
So, so there are SQL functions have several details which you can go through it has returned naturally.

(Refer Slide Time: 26:23)



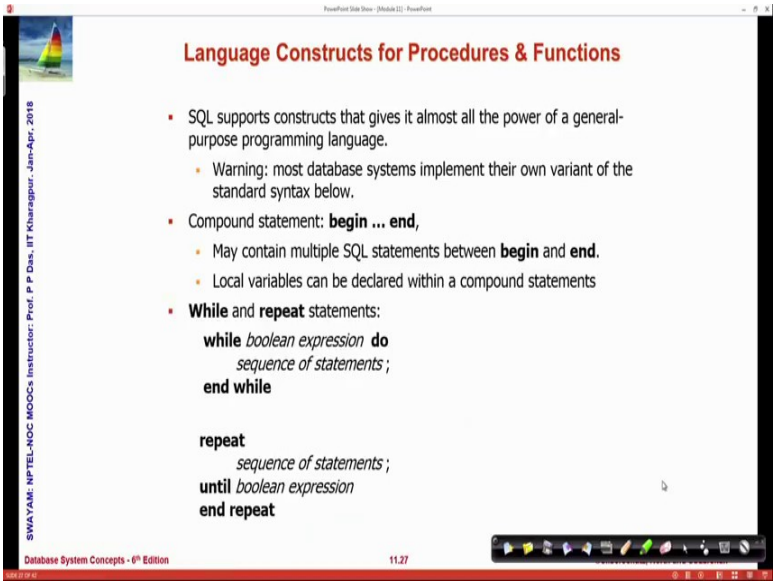
And you can have table functions where it can return table. So, the syntax is given again its clear to see what will happen if you return a table which is computed from the select from where query that you have within the function.

(Refer Slide Time: 26:37)



I can have SQL procedures which just performs some action, but does not have a have a return value so to say. And you can use them they can declare and call those procedures explicitly. Procedures can be functions and procedures can be overloaded as well if you are on SQL 99.

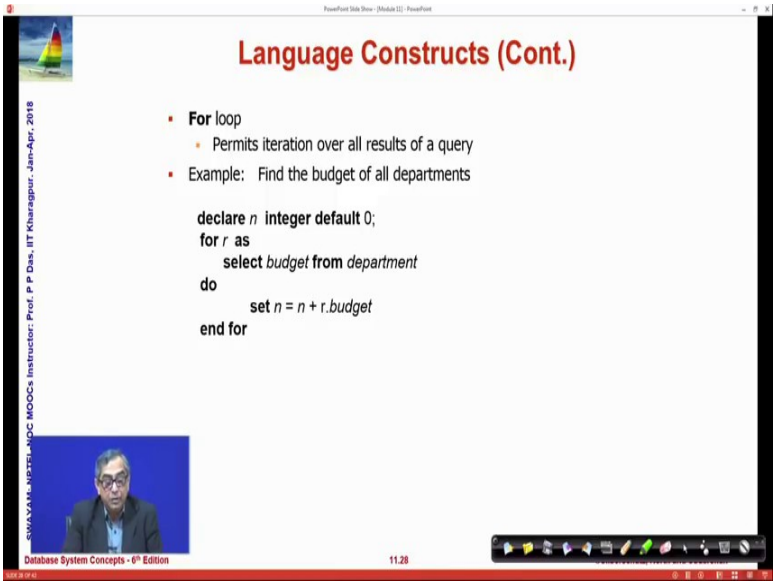
(Refer Slide Time: 27:08)



Now, there are several language constructs as I mentioned SQL does allow while repeat. So, I am I am just covering them in terms of completeness it is not that these are frequently used features or I recommend that you do lot of them right here. If you need

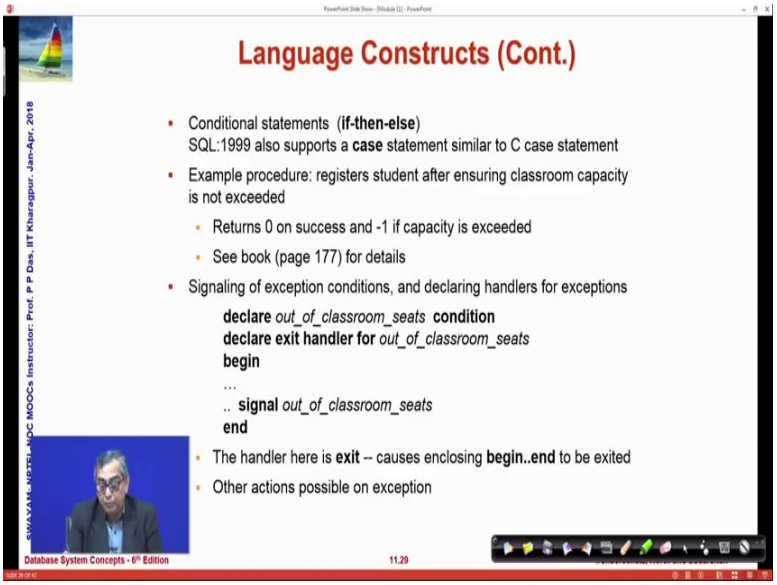
to do procedural thing its always better to do them outside, but in some cases it may be easier to code a query if you can write a while, repeat kind of loop.

(Refer Slide Time: 27:37)



You can write a for loop which iterates over the records of a table which is certainly very convenient. So, if you want to do something over the records of a table compute something that the for loop will become easier.

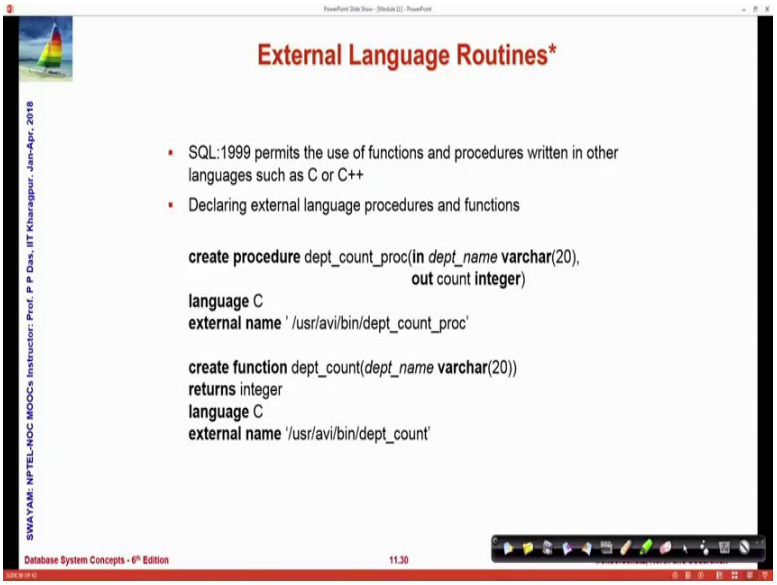
(Refer Slide Time: 27:50)



You have a conditional statement case actually we have seen the case already. So, which is very easy in terms of coding many of the features so, here are some of them then you

have exceptions. So, I am not going through these, these in depth, but this is just to making you aware that while SQL continues to be predominantly a declarative language, it does have quite a bit of procedural support which in an appropriate time can be used if required. So, you can look up the manual for that. And there are a whole lot of things you can do with the external language routines.

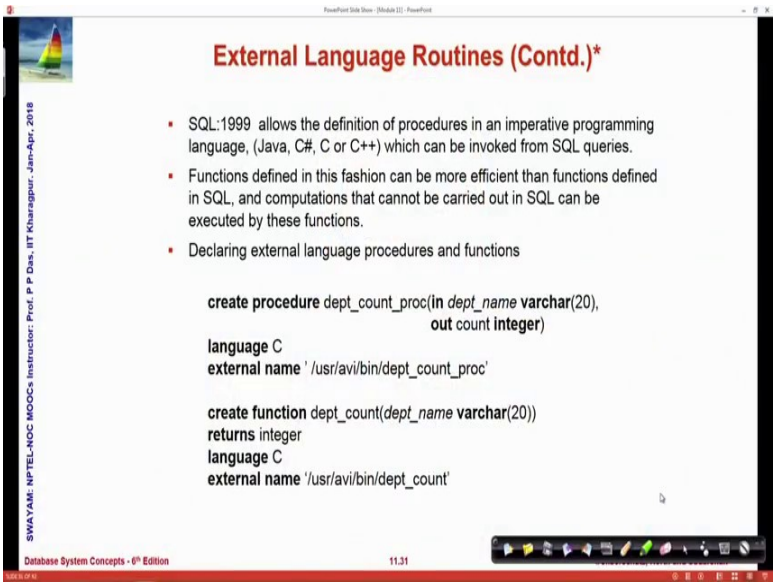
(Refer Slide Time: 28:30)



That is can write a function in C and actually call it from SQL, this is just doing the other way round. Earlier in terms of embedding or in terms of ODBC, a C function was executing a query. Now, you are doing the reverse you are saying that I can write a SQL query which uses a C function that already exist. So, there are external ways of binding, I will not go through these slides in depth, because again the you will have to come a certain way before you can actually start using such features.

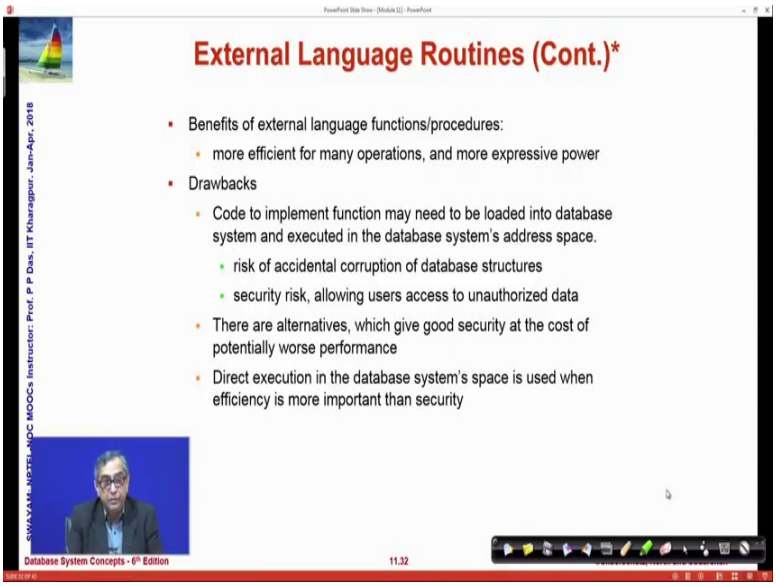
But get to know that there could be as from SQL you can use any external language library; and if some library is good for certain computation which is needed for your query which is a non database kind of computation then you can make use of this external language routines.

(Refer Slide Time: 29:23)



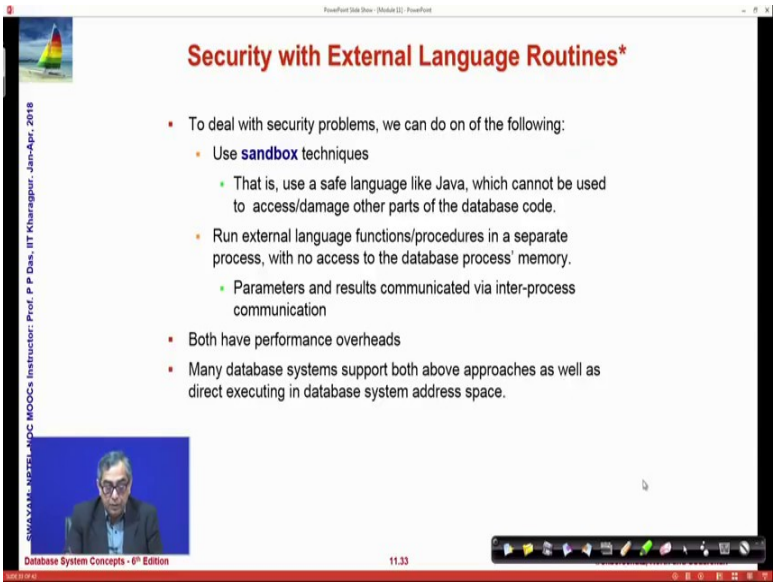
So, I have put together all the basic features that external language routines will need.

(Refer Slide Time: 29:30)



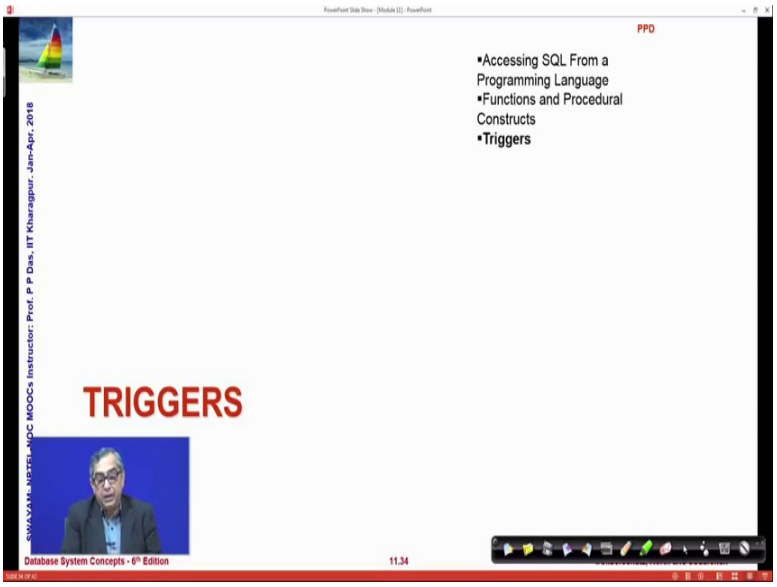
But again I would tell you that there are benefits, but there are lot of drawbacks in using them. And I would not recommend that you frequently use these features. You should primarily restrict to SQL programming, and then anything that you need to do in the native, you should go to choose your language and do that.

(Refer Slide Time: 29:48)



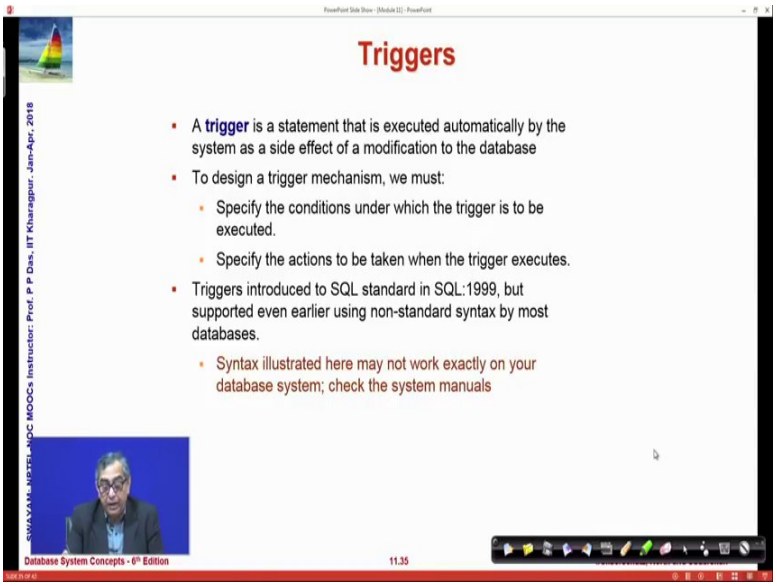
There are security issues also that you will need to understand here.

(Refer Slide Time: 29:58)



Finally, before we close I will just mention a another feature called triggers, triggers are very important.

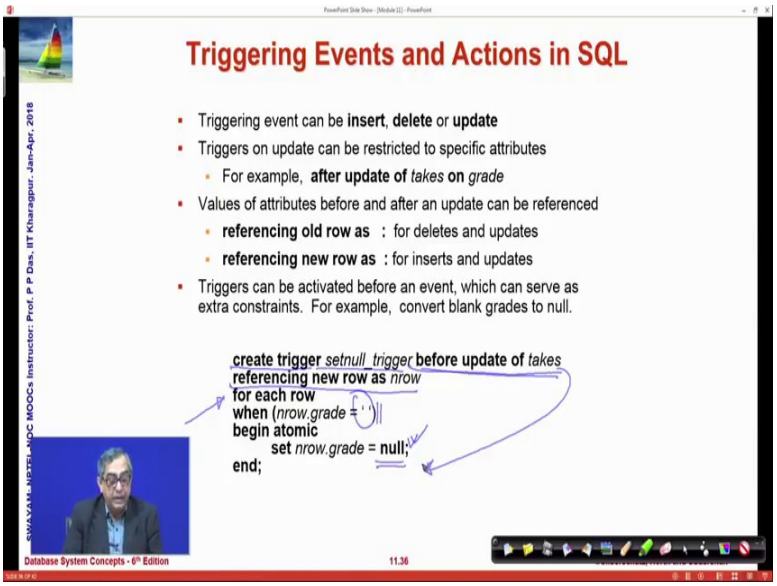
(Refer Slide Time: 30:04)



A trigger is a statement that is executed automatically when something happens in the database. So, you want that well things are happening. And well I want to know if a particular value has exceeded a certain level or if something has become null or some violatory things are happening and so on. So, how do you know that because a database is being accessed by hundreds and thousands of people and with hundreds of tables and millions of records.

So, triggers are a mechanism by which you can set that under this condition, I want a trigger, I want something specifically to happen. So, again they were introduced in 99, but earlier also triggers were there, but in 99 standard they became formal earlier they were somewhat you know differently structured. So, you might find that the system that you are using for practice the trigger in that may have a different format and semantics then the what we are discussing here.

(Refer Slide Time: 31:12)



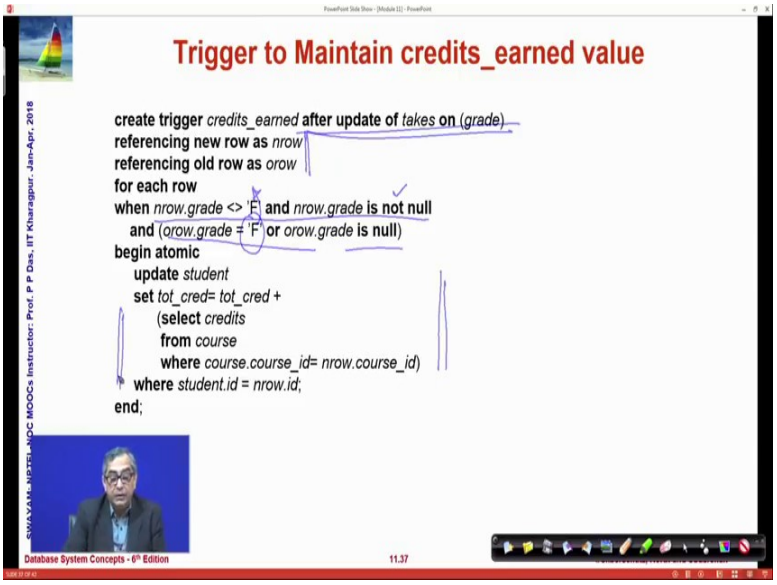
So, the most common triggering events are insert, delete, update. So, if something some update is happening, so I can say that after this update, I want such and such things to happen. Or I can during the update I can one that well I am doing an update. So, there is an old value which is typically referred to as old row, the row that is getting updated. And there is a new row the new set of values that are getting created. So, I might want between the old row and the new row that certain things happen.

So, I can say that well just look into this. So, again the syntax is all similar. Create trigger, trigger there is a name of the trigger and this is the condition. Before update of takes, takes is a relation **referencing new row as nrow**. So, this is the new value that we set. Now, what are you saying, saying that for each row what you do when n grid is blank n row dot grade is blank that is if this is if you are updating and you have got a got you are going to update, a grade value which is blank then you simply set it to null.

So, it is possible that the grade that has come in and grades are characters and what has come in from the input and is going to get updated is a show a certain grade to be now because it may not have been decided. Now, you do not want those blank values to be present. You want because blank cannot be checked, we have we have checkers for null and so on.

So, you want to set that to null. So, trigger can make this thing happened because otherwise how will you know what value is actually getting changed. So, there could be several ways trigger can be used.

(Refer Slide Time: 33:23)

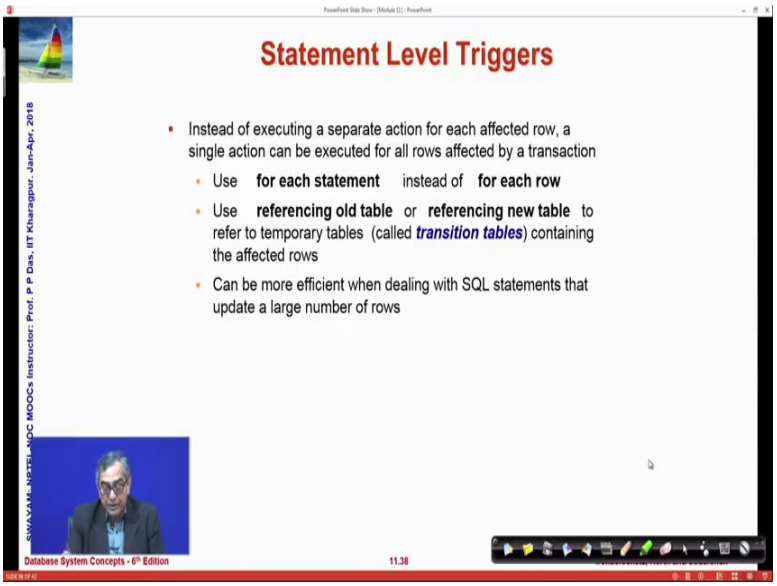


For example, this is showing one that as you change the grades then certainly based on the grades credit earned value is computed. So, as a greatest change if it is now once grades have been entered say for 200 students. Now, after reviews grades for three of them are getting changed. So, how do you know that for those students, the change of the grade may impact the computation of the on credits.

So, you would like to update that on credits or it may or may not be required. So, the trigger will tell you that after update. So, whenever the update happens you take the old and the new value n row and o row, and then you are putting some conditions that if that new row is grade is f, and is not null; old row is grade or it is not null, then you try to do this.

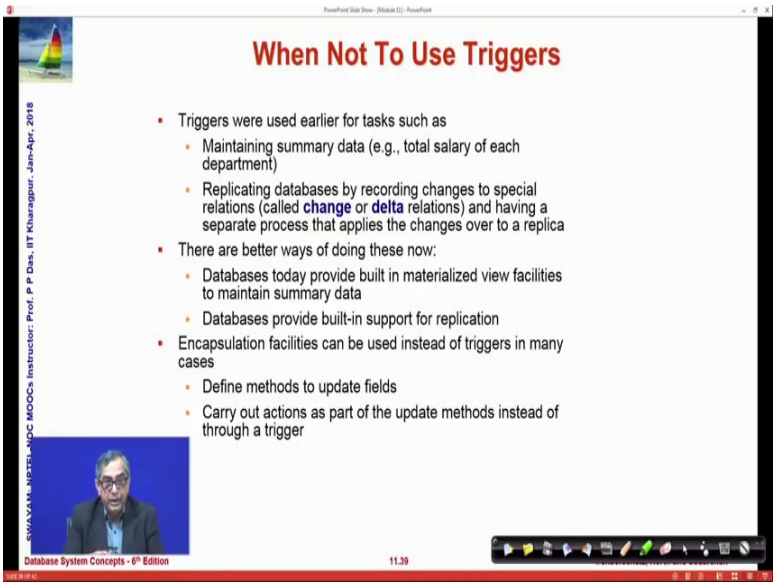
So, if it was failure, and if it continues to be failure, new grade is not f; if it is f then you do not have to do anything; if it is null it do not have to do anything. But if it is if it was f or it was null, and now it has become a different grade then certainly the computation to update the credits earned is required. So, and the trigger gives you the right point when you can do this because otherwise you will not know in terms of millions of updates happening when this particular thing is going on.

(Refer Slide Time: 35:08)



Triggers can be on statements as well you can decide leave for your reading.

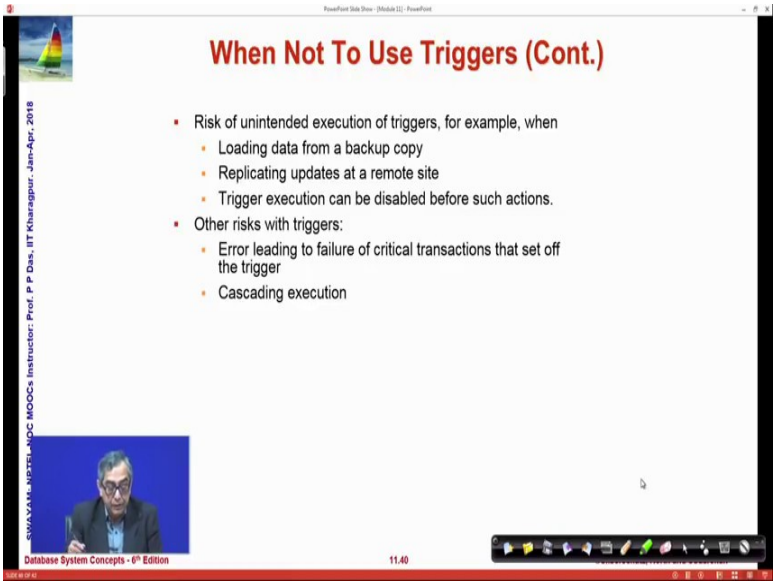
(Refer Slide Time: 35:12)



But you have to be careful that triggers sounds very interesting and what happens particularly with the early stage of programming people get overboard with triggers and start using them severely, but triggers do have a lot of overhead. So, you should not many of the things that triggers can do, can be done through other means for example, by materialized, views and so on. So, as we go along we will mention that these are the problems that need to solve get solved by triggers; otherwise normally you should think

twice before you actually use a triggers. So, there could be different other ways of solving the same problem.

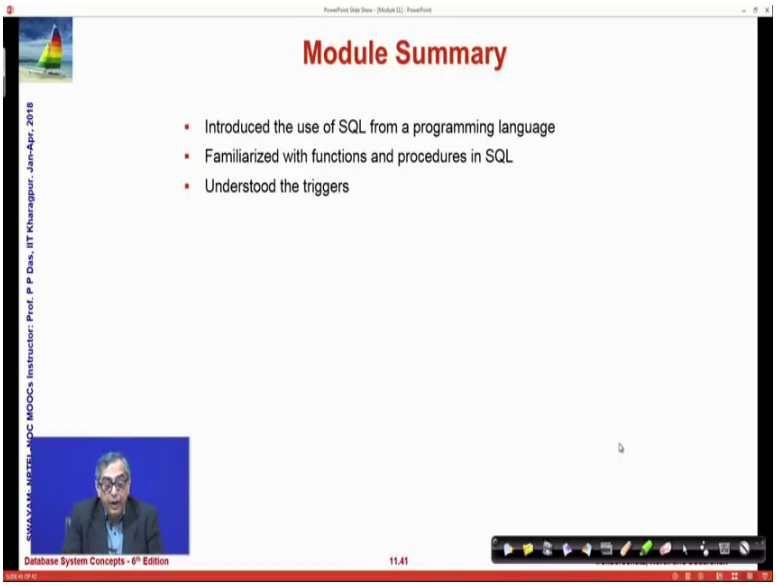
(Refer Slide Time: 35:55)



And because you have to keep in mind that triggers are expensive because once you have triggers and actually internally database for every update. If you have an update trigger on a field or a relation, then with every transaction with every change the database has to check if your trigger is true or not and so therefore, there is a cost to that. The other thing is there are triggers are for the live execution.

So, if you have offline for example, you are loading the data from a backup copy or you are replicating your database at a remote site and so on, then you have to put off the trigger; otherwise you know falsely the triggers will start happening and that may have a catastrophic effect that might trigger of different alarms and all that. So, you have to be careful with triggers in that manner so, cascading executions and all those.

(Refer Slide Time: 36:46)



So, to summarize we have and this kind of closes our direct discussion on SQL. So, we have introduced the use of the very important aspect the use of SQL from a programming language, the interface between the native language and query language boundary. And that is something which will be extremely useful for application programming. And we are familiarized with you know the imperative extensions of SQL, the procedural extensions of SQL in terms of functions and procedures that you can directly write in SQL. And we have just introduced concept of triggers, so that you can sniff what is going on in your database.

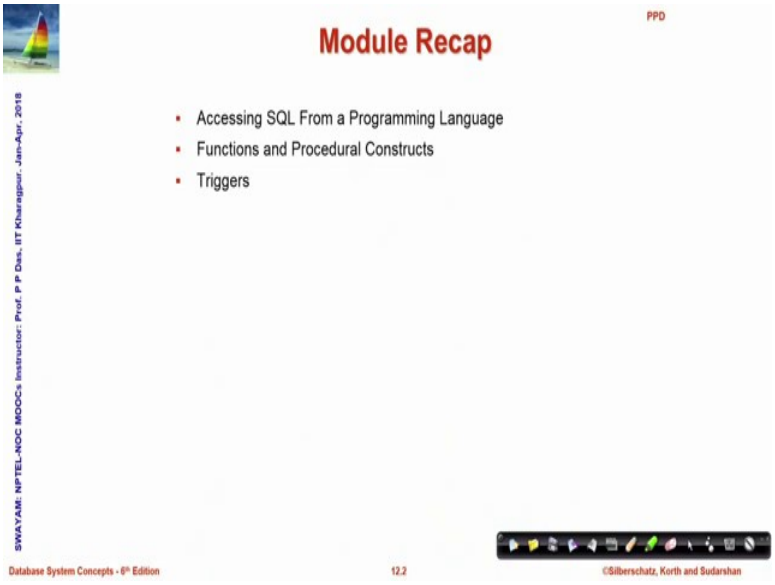
So, there is the quite a few other features of SQL as well majority of them are advanced features dealing with olap and several others, which I chose to skip at this level of the course. So, this will close our discussion on SQL. And next we will move onto the design of the database looking into the algebra and the modelling.

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

Lecture - 12
Formal Relational Query Languages

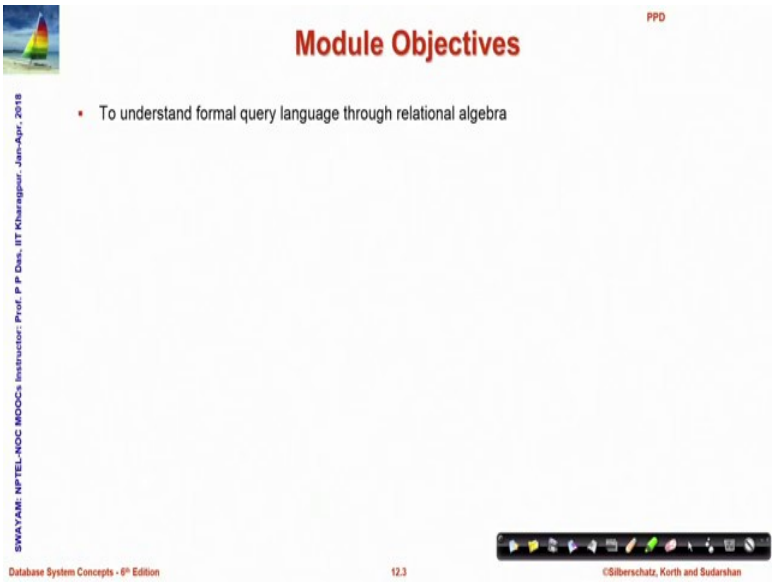
Welcome to module twelve of database management systems, in this module we will talk about the formal relational query languages. In the last couple of modules we have discussed about SQL at length introducing it dealing with the intermediate level of SQL features, and then exposing to some of the advanced features as well. The foundational mathematical model of SQL the query languages are to be discussed in this present module.

(Refer Slide Time: 01:04)



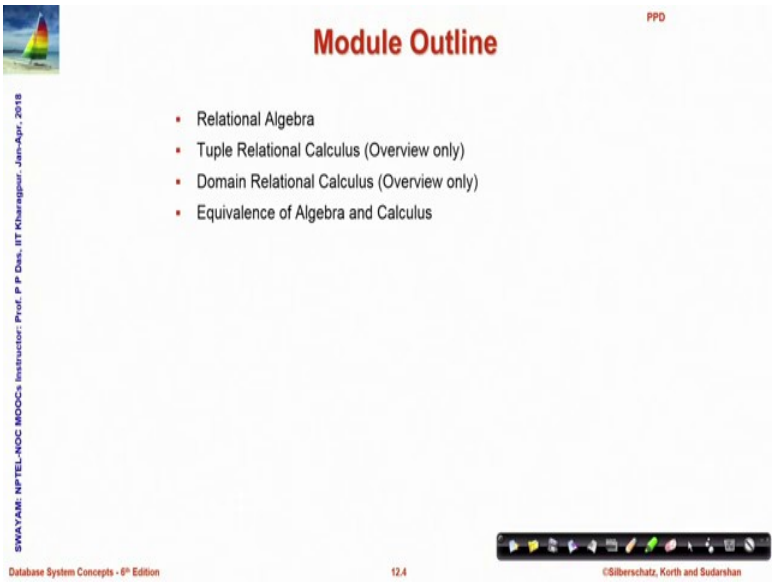
So, this is what we had done in the last module.

(Refer Slide Time: 01:10)



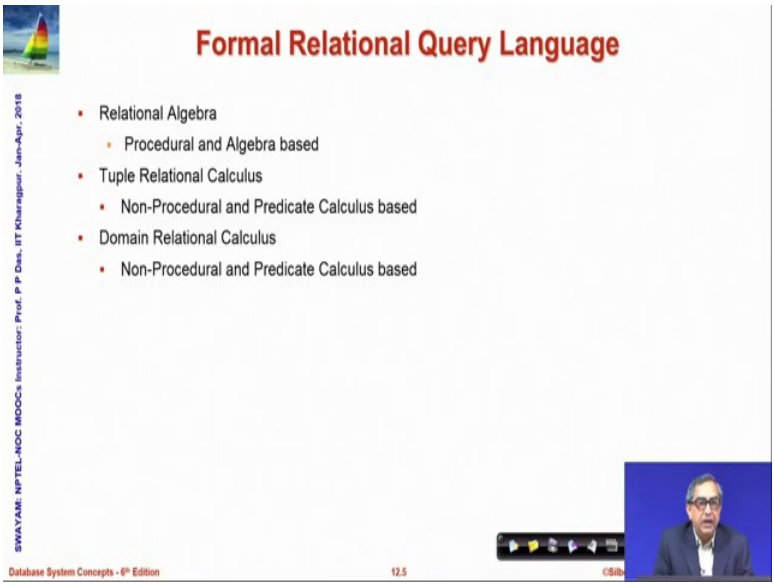
In the current 1 we will work to understand the formal query languages.

(Refer Slide Time: 01:19)



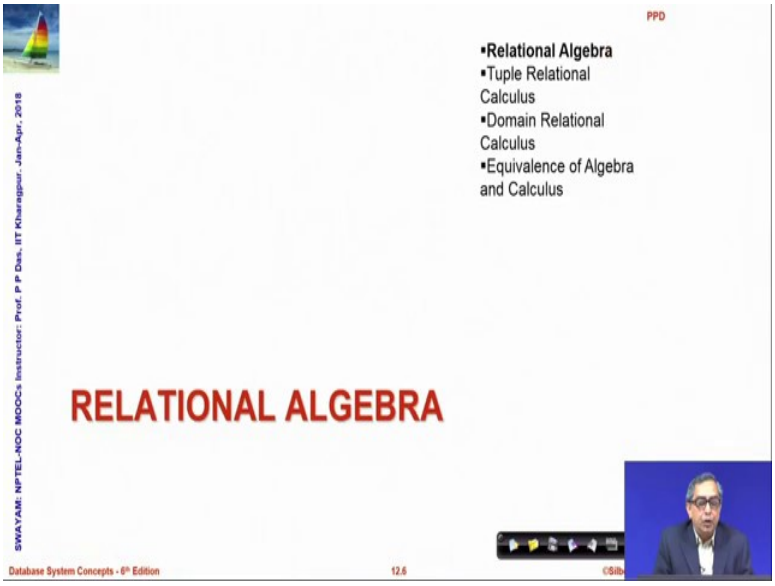
Primarily through relational algebra, and then we will also take a look into some of the calculus aspects tuple relational calculus, and domain relational calculus and we will show by example the equivalence between the algebra and the two calculus.

(Refer Slide Time: 01:40)



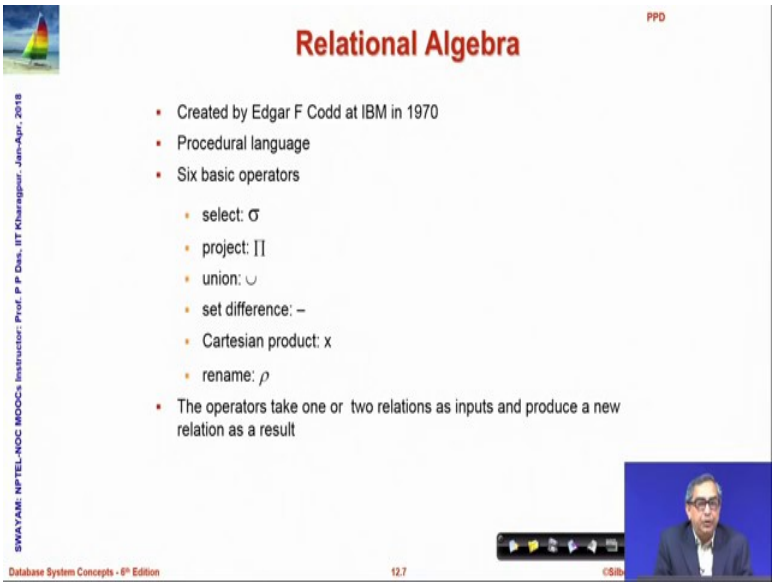
So, formal relational query languages are of 3 types 1 is known as relational algebra this is procedural in nature. So, we specify what operations need to be done to achieve the result and the whole formulation is based on set algebra. The second formal query language is tuple relational calculus which is non procedural and is based on predicate calculus. The third one the domain relational calculus is a minor variant of the tuple relational calculus is and is also non procedural and predicate calculus based.

(Refer Slide Time: 02:35)



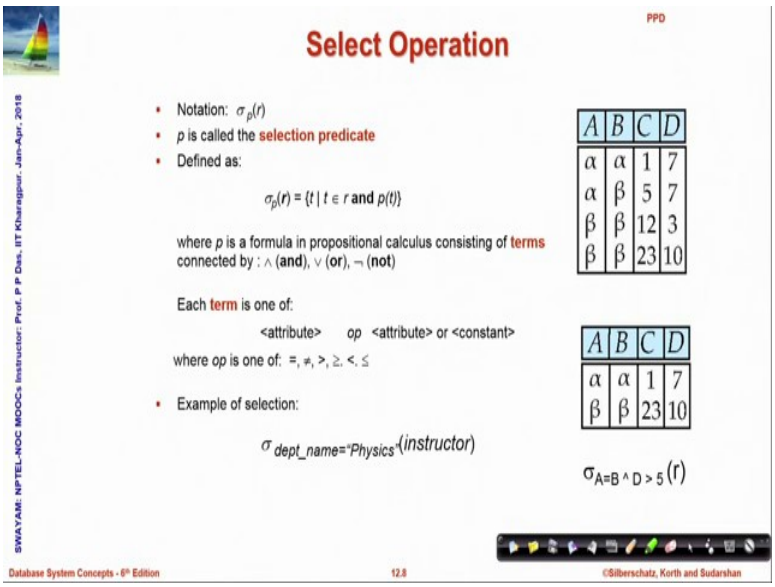
So, we start with the relational algebra in the relational algebra.

(Refer Slide Time: 02:40)



It was created by Edgar F Codd at IBM in 1970. So, you can see that it is quite an old formulation it is a procedural language it has six operators we have taken a quick view of these earlier in this module we will look at them at length. The select(**σ**) project(**Π**) union(**U**) set difference(-) Cartesian product(×) and rename(**P**), we will also look at few derived operations like intersection and division which can be expressed in terms of these basic operators. And each one of these operators can take one or two relations as input and they produce one relation as a result.

(Refer Slide Time: 03:35)



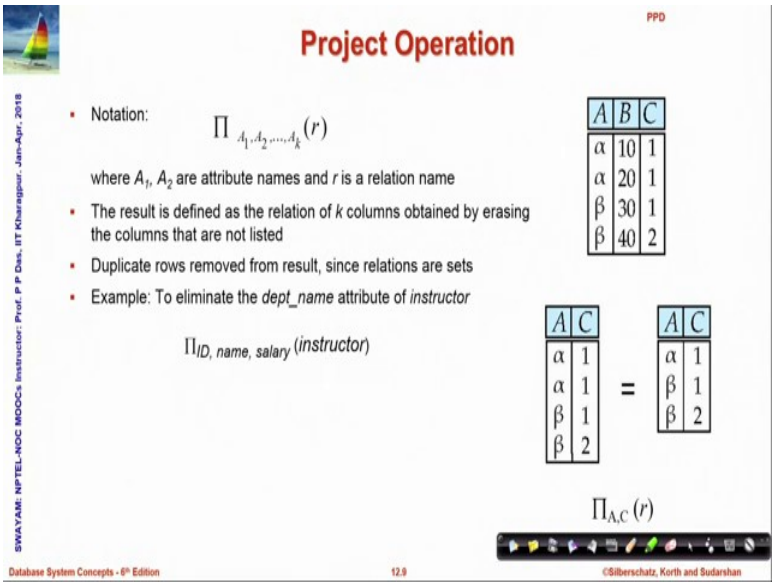
So, we start with the select operation which you know has a notation σ_p where p is a predicate it is called the selection predicate and within parentheses we have a relation r (r) on which this predicate applies. So, it is defined as a set where you collect all the tuples all the rows designated by t and you specify that **t € r**. So, it already exists in that relation and it satisfies the particular selection predicate.

So, any tuple that satisfies this predicate is included in the result any that does not satisfy is excluded from the result, p here particularly is a propositional calculus formula or expression. Where we have different terms that are connected by conjunction (^)or and disjunction (**v)** or negation (¬) or not, and each term by itself could be something like this it is an attribute operator and an attribute where operators are different comparisons operators one of the any six or a term could be an attribute operator a constant.

So, given that we can write any expression, which is a predicate and applying that we can select the tuples from the relation r which satisfy this predicate. So, here we show a simple example instructor is a relation, department_name is an attribute within, course physics is a constant or literal. So, this selection show will select all the tuples where the attribute department name is equal to physics and all the others will be eliminated for reference I have also quoted here the example that we had shown at the time of introducing relational algebra.

So, you can see that here we have a more complex propositional term propositional formula where there are two terms, the **a =b ^ d > 5**. So, in the selection result both of these conditions must be satisfied by all the tuples which feature here. So, this is the first operation that relational algebra has the select operation.

(Refer Slide Time: 06:40)

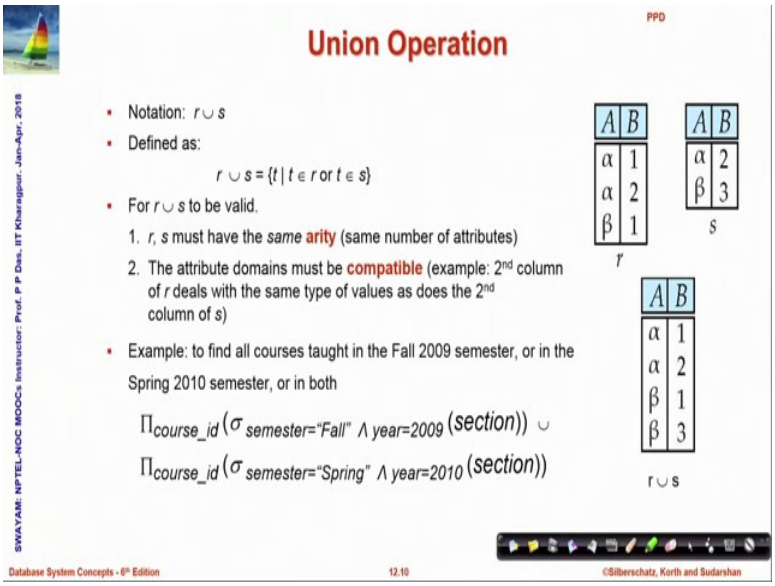


Next we move on to the second operation which is a project operation where a relation can be projected in terms of a number of attributes. So, pi(**Π**) is the notation r again is the relation and the subscript at A_1, A_2, A_k, are key attribute names k has to be at least one and these attributes will be retained in the relation. $\mathbf{\Pi}_{A1, A2, Ak}$

So, the result is defined as the relation of k columns by erasing all the columns of r which are not listed amongst this A_1 to A_k. Naturally, if you erase some columns it is possible that two rows that were distinct in those columns, but are identical in A_1 to A_k feature in the relation since every relation is a set no distinct no two copies of the same people are allowed. So, the duplicate rows will be removed from the result remind you this is in contrast to what SQL does by default where duplicates or multi sets are allowed by default here we are talking about the formal relational algebra where it is purely set theoretic.

So, duplicate rows on projection will be removed from the result. So, we have an example from the instructor relation we had seen earlier we are projecting id name and salary($\Pi_{id\_name, salary}$). So, we are removing the department name, which also exists in the same relation and as a reference you can see the example that we had seen earlier while introducing the relational algebra where projection is done from three columns A, B, C into two columns A and C and this results in at least results in two rows which are identical and therefore, in the final result one of those the duplicate one is removed.

(Refer Slide Time: 08:57)

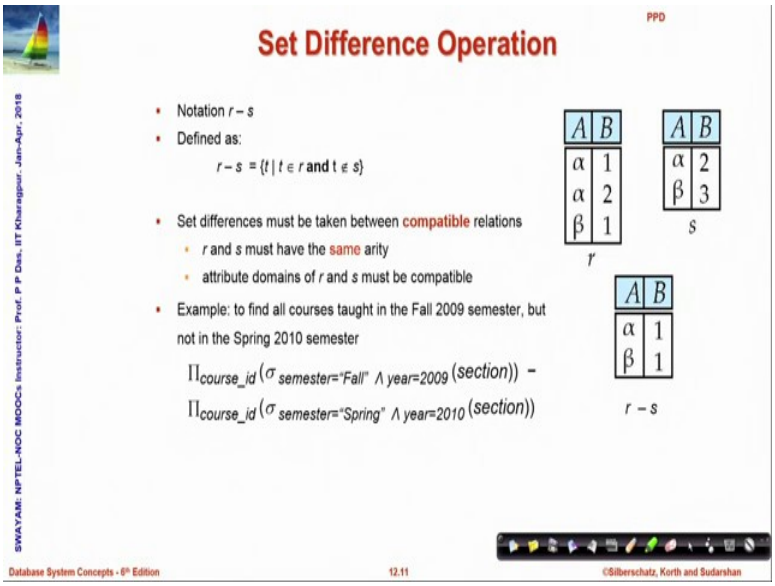


Moving on that the third operation is quite simple it is set theoretic union. So, **r U s** where r and s are two relations our set of peoples which either belong to r or belongs to s, or belongs to both, the condition is you can take union of both these relations have the same arity and the order of the attributes must satisfy that every corresponding attribute must have compatible domains. So, if we I talked about the second column of r, and if we talk about the second column of s they must be of the same type and this must hold for all columns that the union forms that is all columns of r as well as s , otherwise this operation is not defined.

So, as an example we show that to find all courses taught in fall 2009 semester, this is a this is the query where we do a selection to find all tuples which are taught, which represent courses taught in fall 2009 semester, from the section relation we do a projection to get the ids of those courses only $\Pi_{course\_id}(\sigma_{semester="Fall"\wedge\ year=2009}(section))$. And the second row tells you the courses that are taught in the spring 2010 semester $\Pi_{course\_id}(\sigma_{semester="Spring"\wedge\ year=2010}(section))$, and we do a union to get courses that are taught either in fall 2009 semester, or in spring 2010 semester, or both. This is how the union is performed and this is the earlier example repeated here.

$\Pi_{course\_id}(\sigma_{semester="Fall"\wedge\ year=2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester="Spring"\wedge\ year=2010}(section))$

(Refer Slide Time: 10:54)

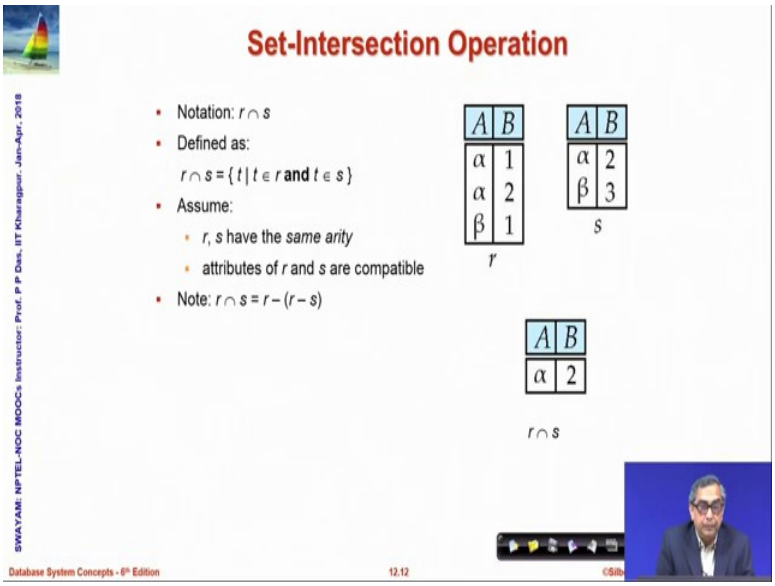


set difference is again just simple difference of sets r minus s where a tuple belongs to r and it does not belong to s. So, you remove all the tuples belonging to s that exist in r to get r - s again they must have the compatibility of having the same arity and attribute corresponding attribute domains must be compatible , this is an example to show to find all courses taught in fall 2009, but not in spring 2010.

$\Pi_{course\_id}(\sigma_{semester="Fall"\wedge year=2009}(section)) - \Pi_{course\_id}(\sigma_{semester="Spring"\wedge year=2010}(section))$

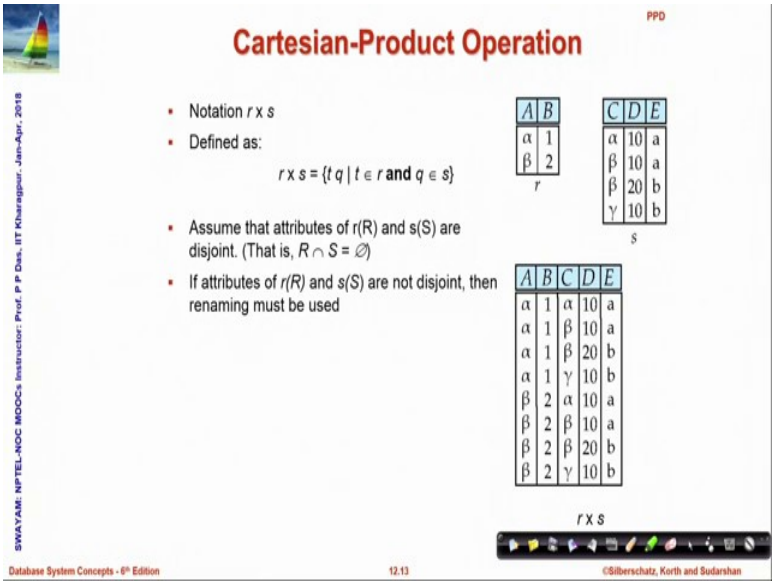
So, as opposed to union in the last slide you do a set difference to get this result. So, this is how you can use set theoretic operation to get different relational results this is also the result from the earlier example.

(Refer Slide Time: 11:49)



Set intersection can be supported is supported, but it is not a basic operation because as it is defined by all tuples which belong to both r and s. It can actually be computed by applying set difference twice **r-(r-s)** and certainly for set intersection also the same assumption about arity and compatibility of types hold and this is the earlier example.

(Refer Slide Time: 12:20)

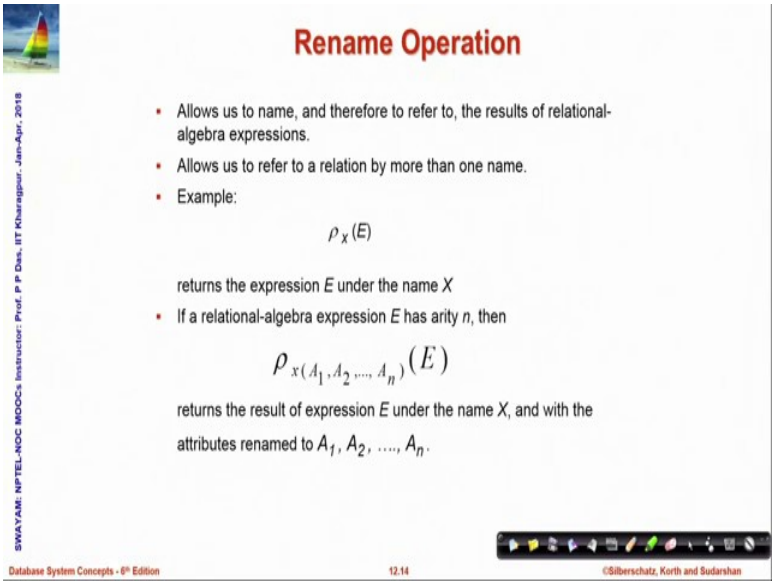


Next is Cartesian product, where we take two relations and for the Cartesian product we make we juxtapose one relation with the other. So, t is a relation from is a tuple from r(**t ∈ r**), q is a tuple from s (**q € s**) and we put them side by side to get a tq in the Cartesian

product **r** × **s**, which basically means that you compute all possible combinations of pupils from r and of s. It is assumed that the attributes of r and s are disjoint that is a schema of r intersection schema of s is null. **R ∩ S = Φ**

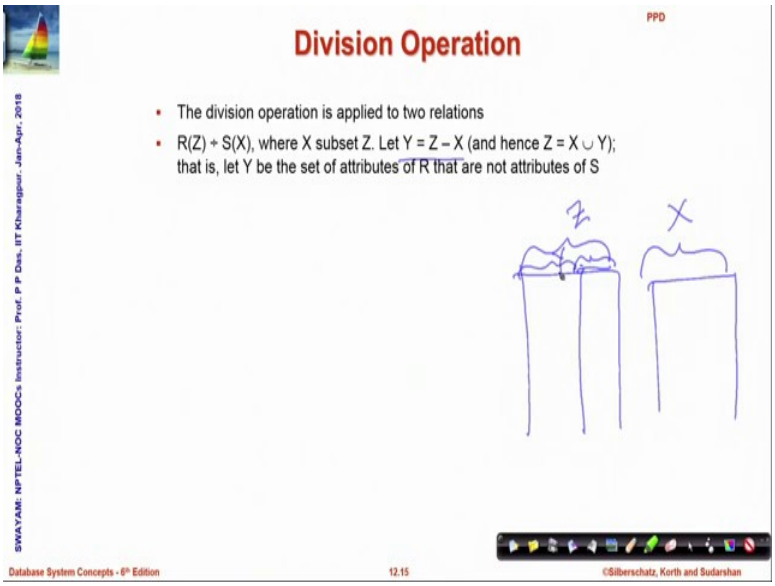
If the attributes are not disjoint then we must use renaming which we will soon see, and here is an example that we had shown earlier of r and s computing r process, Cartesian product is a very useful operation particularly for computing join as we have seen in sql already.

(Refer Slide Time: 13:36)



Rename operation basically allows you to rename some expression attribute into another. So, the operator is **P** and you have an expression to which you give the name x and that is how the renaming of you can have multiple attributes of x as well.

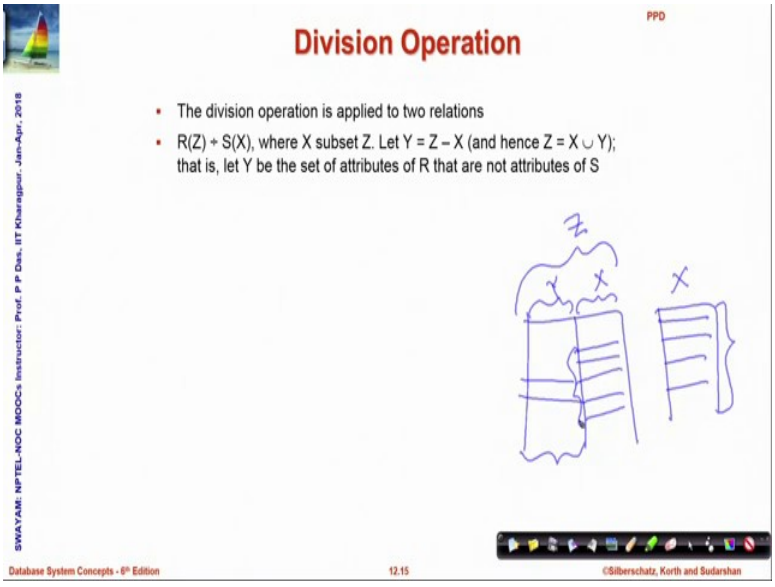
(Refer Slide Time: 13:58)



Division is a is another operator in relational algebra that can be applied between two relations, but it is a derived operation. So, it says that if I have, so let me just show you by, by a little bit of sketch that if I have two relations which are which has a set of attributes z and s which is a set of attribute x, such that actually the set z is a superset of x. So, z has more attribute the relation r has more attributes.

So, if you take the difference of attributes between z and x and call it y, Y=Z-X then we are interested what happens on these remaining set of attributes y..

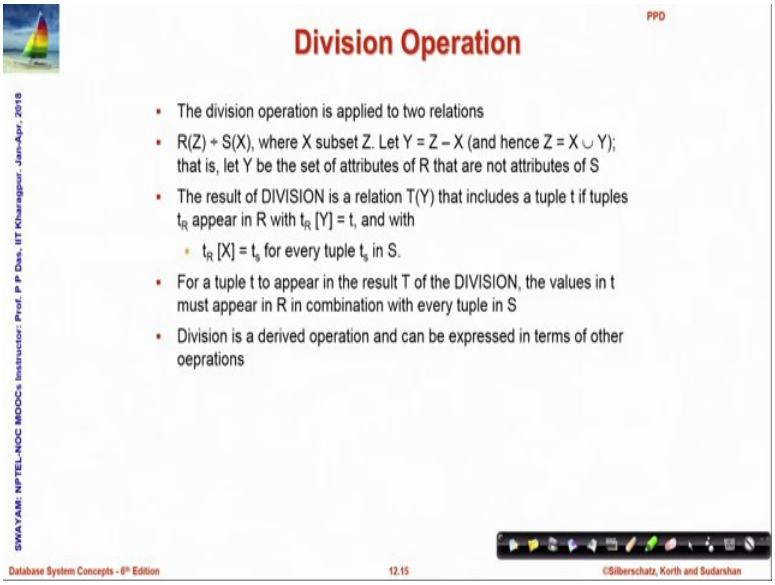
(Refer Slide Time: 15:02)



So, this is what, we have this is my X set of attributes this is where X occurs this difference is Y this whole set is Z. Now in this what we want is we want to in the output we want a relation having only the y attribute such that for every tuple in that relation if I consider all the tuples of s then their cross product must be a part of r.

So, for every tuple here if there are say four tuples here, a tuple here must have all these four tuples along with it in the result. If it does not have any one or more of them then that tuple will not feature in the final result.

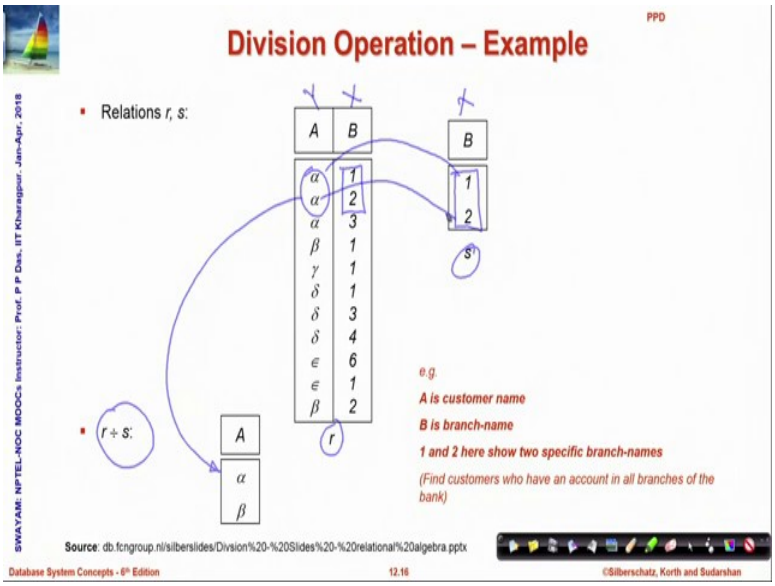
(Refer Slide Time: 15:58)



So, the result of a division is a relation T(Y)that include tuple t if tuples tr that is the part of the tuple the tuple that appear in r and on that on the y part the difference part it matches. So, that you have that t_r [**X**] = t_s where t_s actually exist in s.

This must happen for all tuples in s, so division is a very good interesting operator which is often required for coding different queries. So, it is a derived operation.

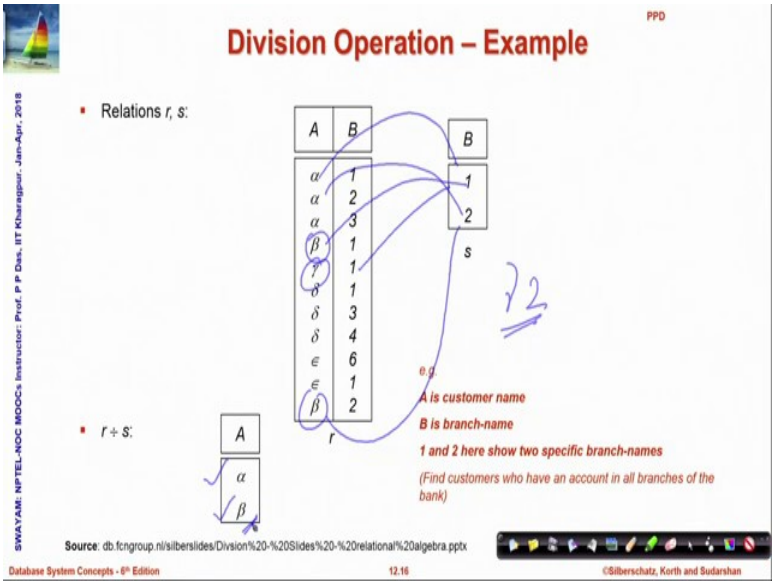
(Refer Slide Time: 16:44)



Let us take an example, let us say this is this is the relation r and this is a relation s and I am trying to compute r ÷ s.

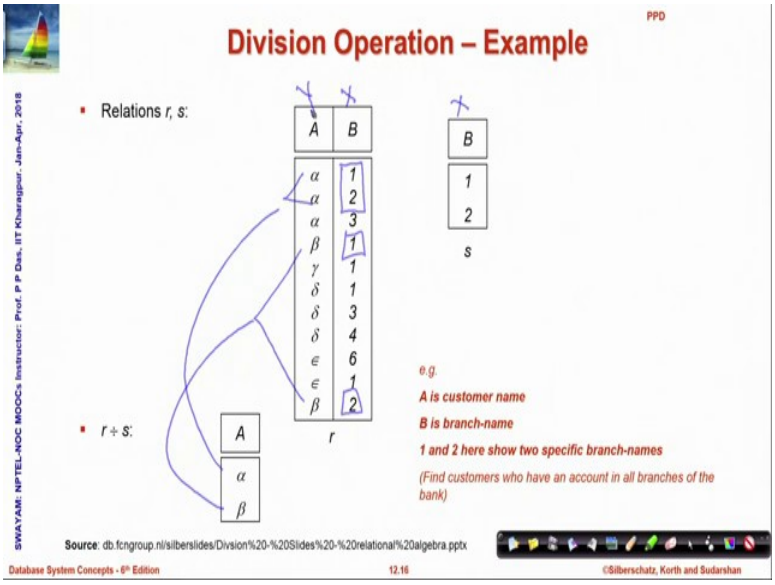
So, what I want is over the attributes of this is therefore, X this is x this attribute y attribute set Y. So, all over X all the values that I have, I must have those values in the relation r, if I do then the attribute the particular tuple matching on the attribute Y goes onto the result. So, **α** goes onto the result because you have both α a, **α 1** as well as **α 2** in the set r, in the relation r.

(Refer Slide Time: 17:44)



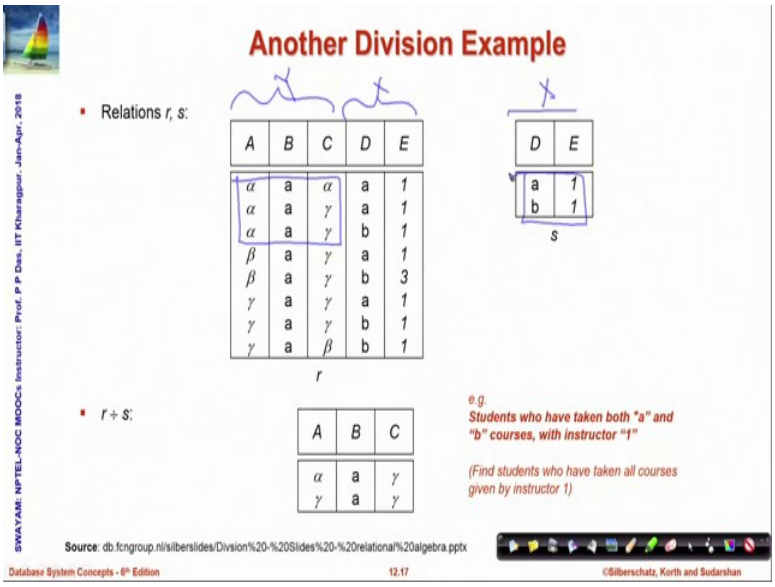
β1 is there and β2 is also there, so β also goes into the relation α goes in because 1 is there 2 is also there, but γ will not go in because I have γ 1, but I do not have a tuple γ 2 in r if I had γ 2 in r that will go in the result.

(Refer Slide Time: 18:13)



So, if I can say it again the whole of the relation s must happen over the X attributes of r, consider these two together. If that happens then the attributes on y the tuples would be chosen and that is how we get the result having α and β.

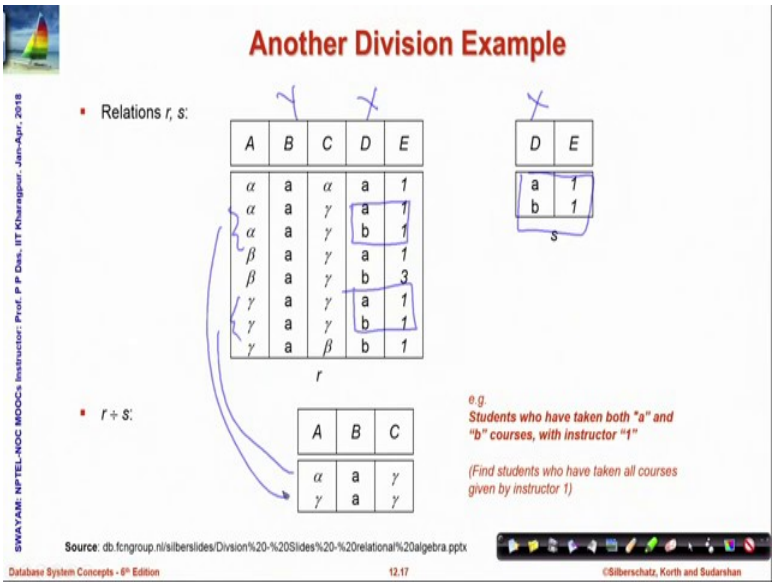
(Refer Slide Time: 16:47)



Let us look at one more example, so this is got r has five attributes this has X as two. So, this is this is two attributes X, these are three attributes Y and what I have to look for is those tuples in r where the values over Y would be same and I should be able to get the whole table of X over whole table of the relation S over the X attributes. So, if we look at here this is a 1, b 1, a 1, b 1, here these are identical.

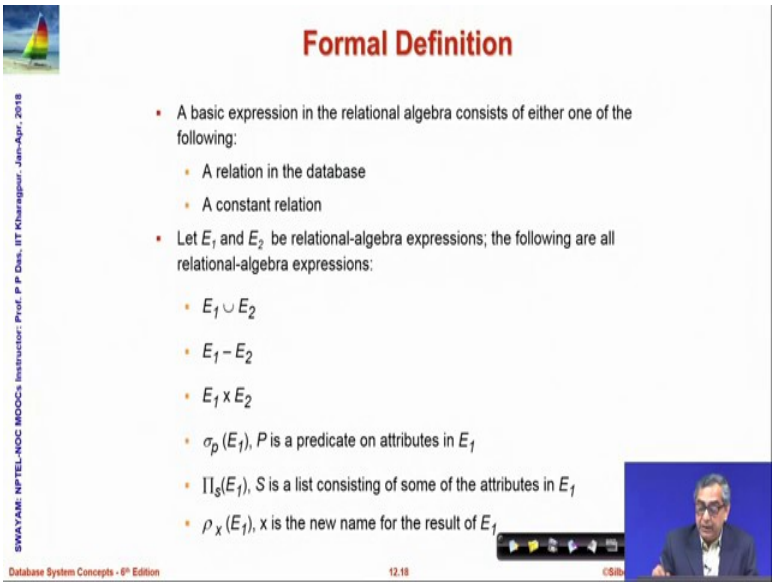
So, this particular tuple will go into the result if I look in here this tuple will go into the result, but if I consider this tuple β a γ which has a 1 over d, but β a γ does not have b 1 it has b 3, so it will not go into the result. So, if you conceptually look at that is the reason this is called a division. So, you get this here you get this here. So, this is like the way we divide that this is the whole and wherever it goes in if the tuples that are identical on the wide set of attributes we will collect them into the final result.

(Refer Slide Time: 20:30)



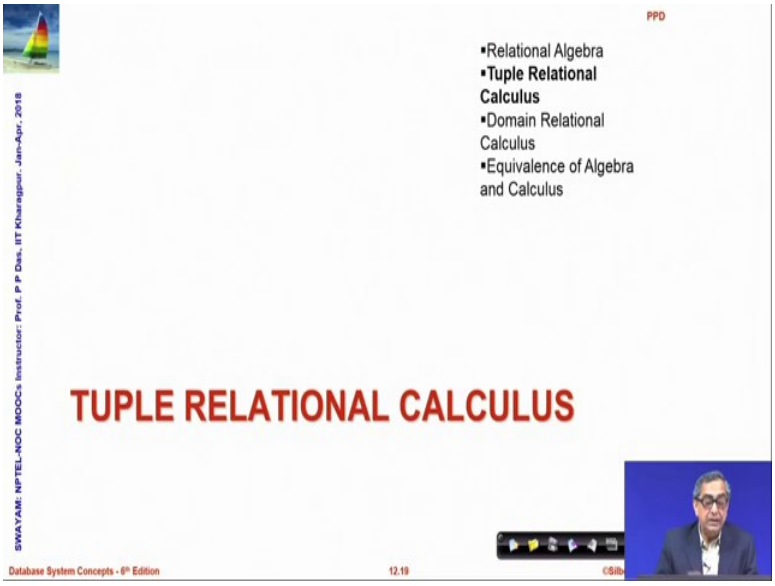
So, this is the division operation which by which we can compute the students who have taken both a and b courses instructor 1 will be found out from this division operation.

(Refer Slide Time: 20:51)



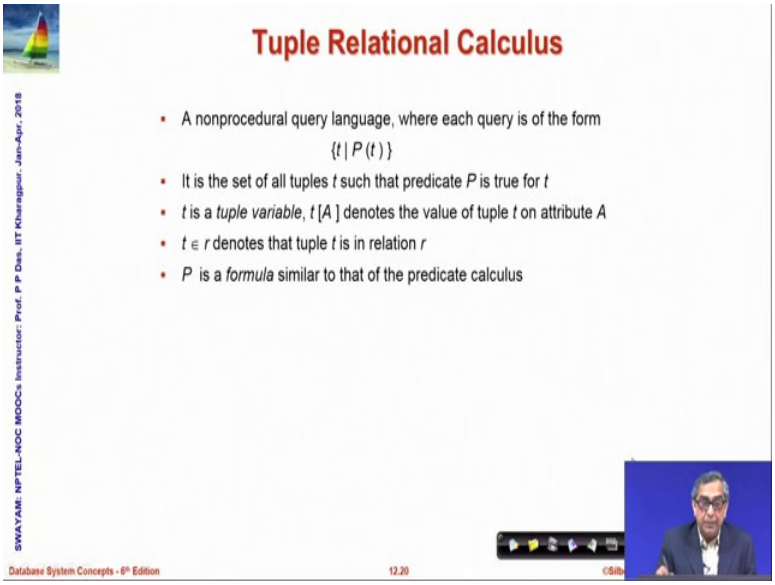
so formally speaking a basic expression in relational algebra consists either of a relation in the database, which is a instance or a constant relation which does not change which is given. And then we have six operations of union set difference cross product, selection projection, and renaming that can give us all sorts of different relational algebra formula and also the derived operations and whatever we have seen of sql can be expressed in terms of this relational algebra formula.

(Refer Slide Time: 21:30)



Now, relational algebra is not something totally unique the same thing can be done in terms of other formulations also.

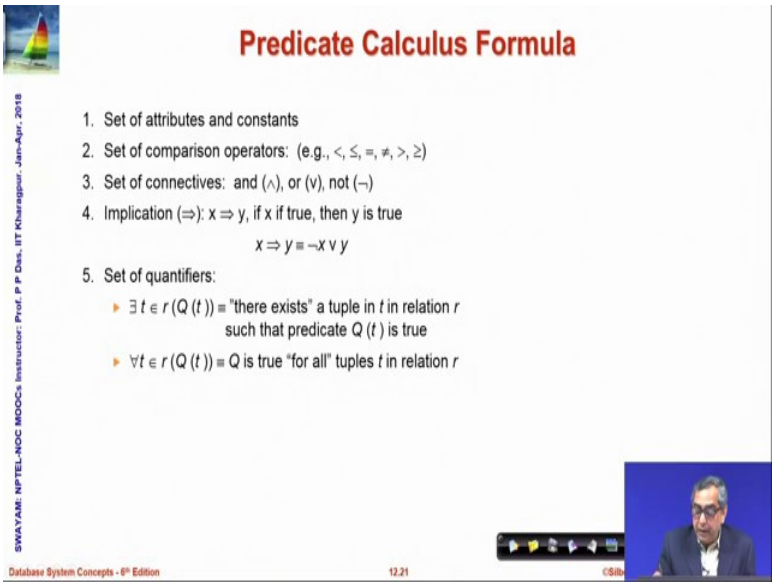
(Refer Slide Time: 21:41)



A second formulation which is also used is known as tuple relational calculus, which is non-procedural relational algebra was procedural because you are actually doing the explaining what the operations or you are detailing out what the operations are in tuple relational calculus you are specify what the condition is you are specifying what this condition is.

So, those tuples which satisfy this condition form the relation, so p is a predicate. So, whatever t satisfies the predicate are included and if a is an attribute then t [A] will denote the value of the tuple on attribute A, A could be a single attribute it could be A set of attributes also and t € r, P is as I said it is a its a predicate calculus formula.

(Refer Slide Time: 22:43)

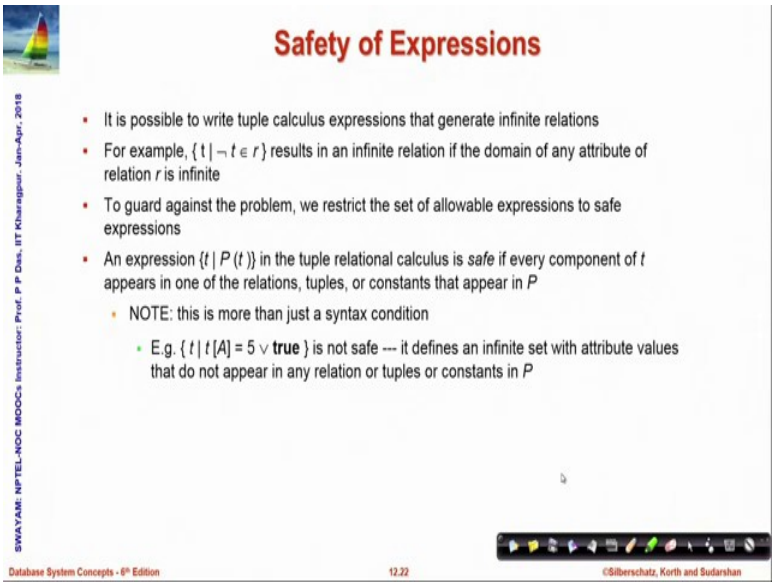


So, it could be a set of attributes or constants this I am just included for your help if in case you have become rusted with predicate calculus you can refer to a predicate calculus as a set of attributes and constant. It has set of comparison operators the six of them, there are a set of connectives these are all same as the propositional calculus there is ⇒ which says if x is true then y is true if x is false then the whole thing is true vacuously.

And what makes it primarily predicate calculus is a fact that it has existential quantifier, which says that the formula there exists t ∈ r Q(t) holds if I can find at least one tuple t which satisfies Q(t).

Similarly, there is a universal quantifier where I will say that for all t € r, Q(t) is true if for all tuples of r, t satisfies Q(t). So, this in tuple relational calculus all conditions all predicates are formula of this kind and with that we can represent any.

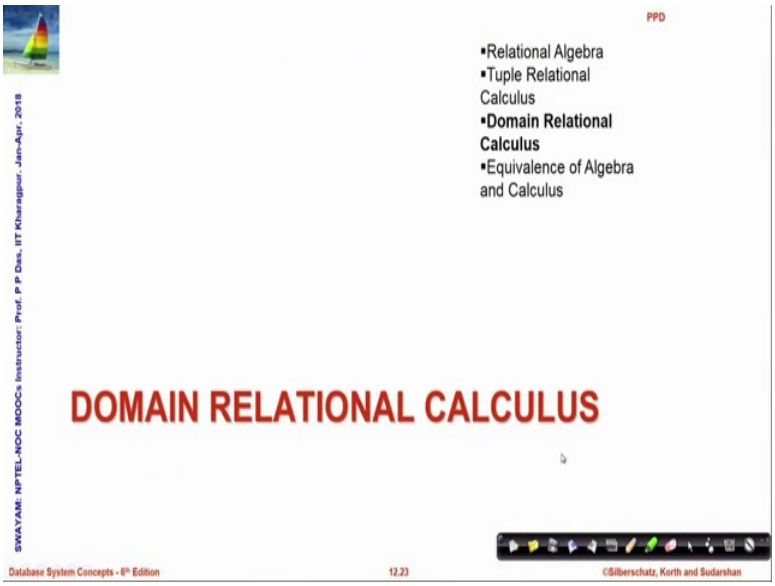
(Refer Slide Time: 24:07)



Relational set in full there is a word of caution because it is possible to write tuple relational calculus expression that can potentially generate infinite relations. Now, infinite relations are naturally not representable for example, if I write simply this that r is a relation and I write this predicate that ¬ t € r, which is basically complement set of r now a complement set of r potentially may be infinite if the domain is infinite..

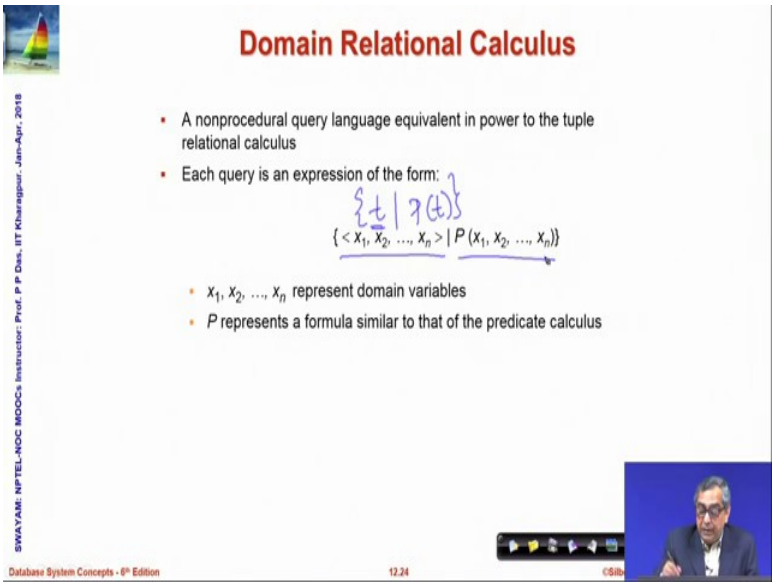
So, such expressions tuple relational expressions are not acceptable as a part of the design. So, whenever we want to do this we would like to guard this by putting some additional condition and we have to make sure that any expression that we have in tuple relational calculus is a safe expression, in the sense that it does give me finite number of tuples in the relation.

(Refer Slide Time: 25:14)



A third formalism that exists that is used is known as domain relational calculus.

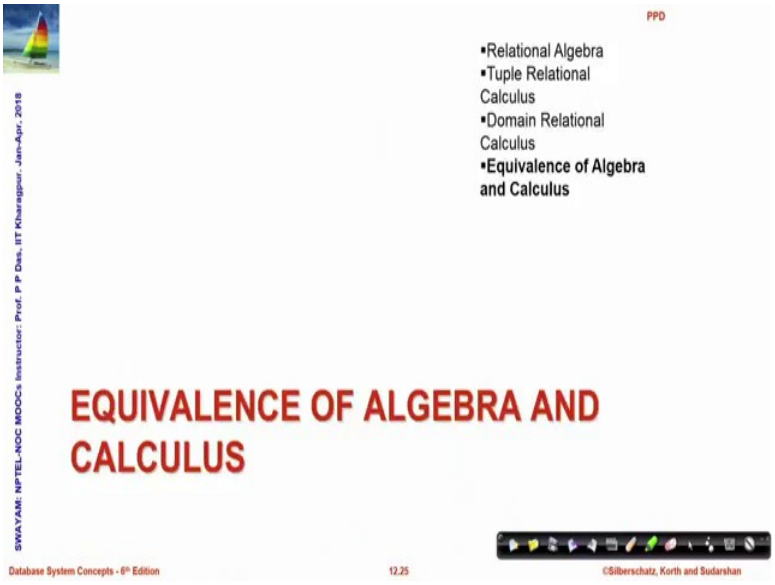
(Refer Slide Time: 25:25)



Which is also non procedural and equivalent in power to tuple relational calculus again, which is very similar to tuple relational calculus the only difference being if you just recall tuple relational calculus. We are writing collection of tuples t such that P(t) that is the predicate P is satisfied by t here, instead of writing a tuple variable t we write expand it out in terms of all its components..

So, we write the values of the different components of t over different n attributes and write it as a n tuple and so here instead of having one variable t we have n variable $<x_1, x_2, x_3 \ldots. x_n>$ and therefore, the predicate is formed of this n variables where x_1 to x_n are represent the different domain values, domain variables and that leads to the reason for the name domain relational calculus.

(Refer Slide Time: 26:28)

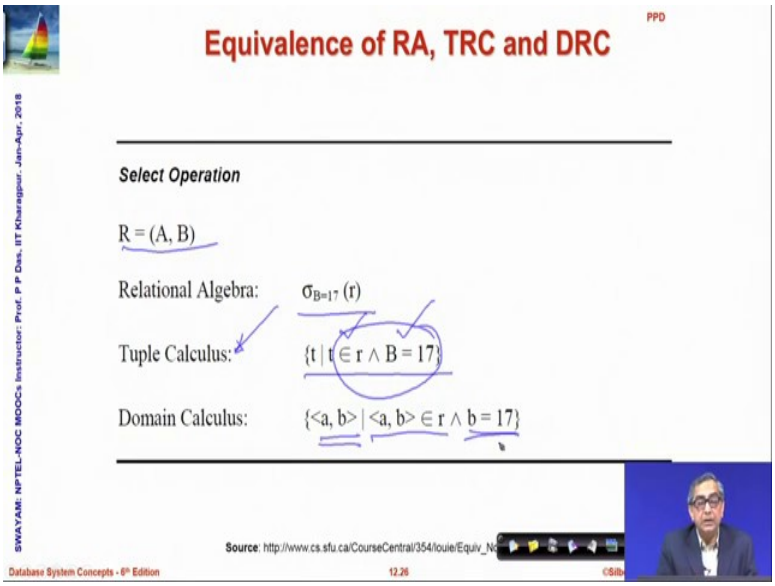


Now, of the three formalisms that we have seen we will not go into direct mathematical proofs, but in the next couple of slides, I just show that they are equivalent in nature. What means that if I can write an expression in relational algebra then it is possible to write an equivalent expression in tuple relational calculus and in domain relational calculus and vice versa..

So, if I can write an expression in any one of these formalisms then there are equivalent expressions in the other two formalisms as well which is probably very easy to see between, tuple relational calculus and domain relational calculus because 1 is just representing the whole tuple as a single variable whereas, the other is representing it in terms of n domain variables.

So, their equivalence is pretty much very similar the fact that your predicate calculus formula has to change, but it is not, so obvious for the equivalence between relational algebra and the calculi.

(Refer Slide Time: 27:41)

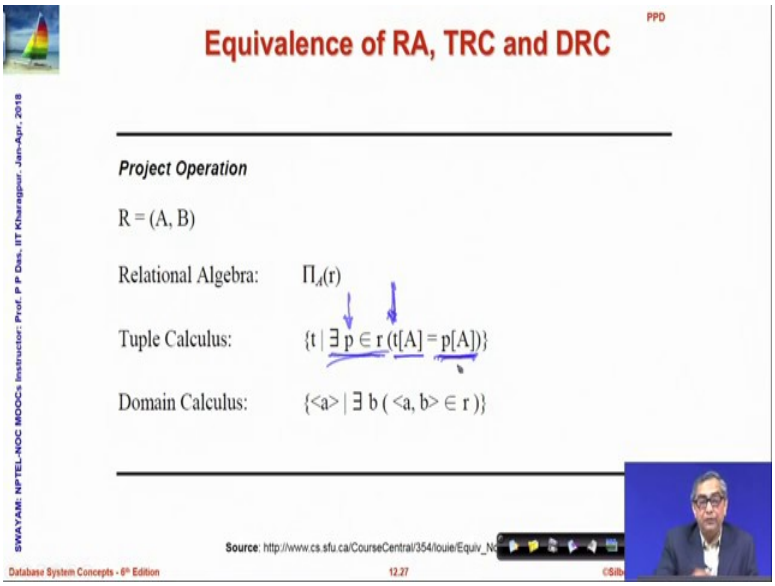


So, we just show a few examples of the basic operations for example, say select operation. So, I am just not showing the proof, I am just giving you some example cases to show through a relation r has two attributes A, B this is what you wanted to write in relational algebra that you want to collect all tuples where b =17.

Naturally in tuple relational calculus you can easily write the first condition is you are doing it on r. So, t must belong to r and your condition is B should be 17 {**t | t € r ^ B = 17**}. So, this predicate will represent the same set or the same relation as in tuple calculus in domain calculus there are two components a and b. So, you have to say component taken together must belong to r and the component b must be equal to 17. **{<a , b>| <a , b>€r ^ b=17 }**

So, you can see that it is pretty straightforward to see the equivalence between a relational algebra expression involving select and the corresponding tuple calculus or domain calculus expressions this is through an example, but you can certainly easily generalize this as a proof.

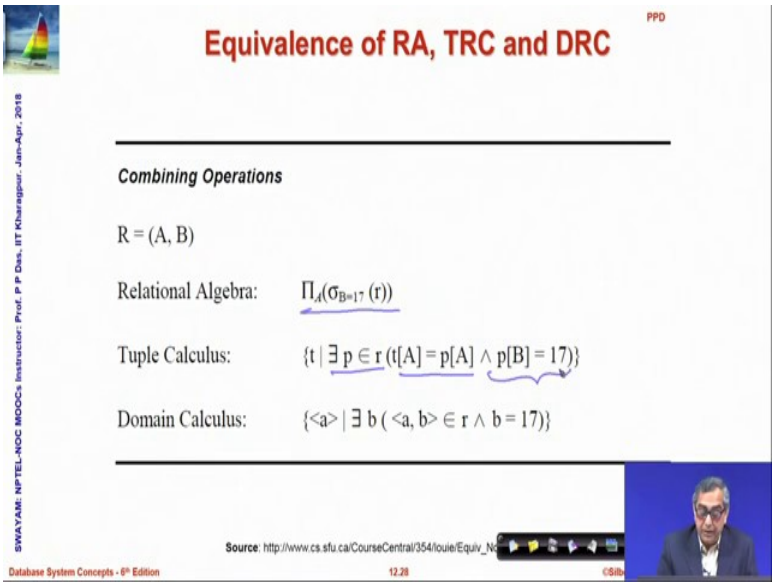
(Refer Slide Time: 28:49)



Similarly, for projection if we do a projection on a then all that we are trying to do is we are trying to create a new relation where only the a attribute exists. So, in the tuple t the a attribute exists and if I have projected and got this tuple t then in my original relation r there must be some tuple p such that on the attribute a they match they are same.

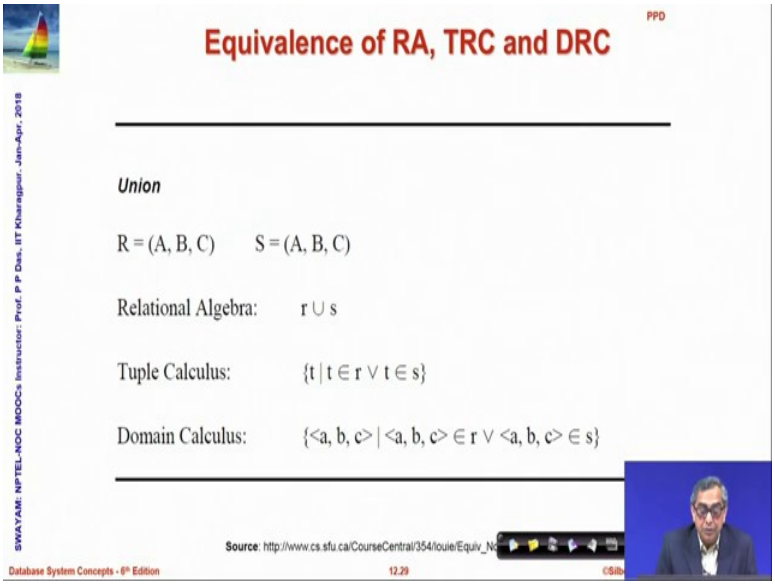
So, it is the same thing in relational algebra we said that keep a and erase everything else here we are saying that if we have been able to get a tuple t which has a value t a then there must be a tuple p in r, which has the same value over the same attribute. So, this condition is equivalent representative of the projection, and the same thing can be written in domain calculus you can go through it carefully and convince yourself.

(Refer Slide Time: 29:59)



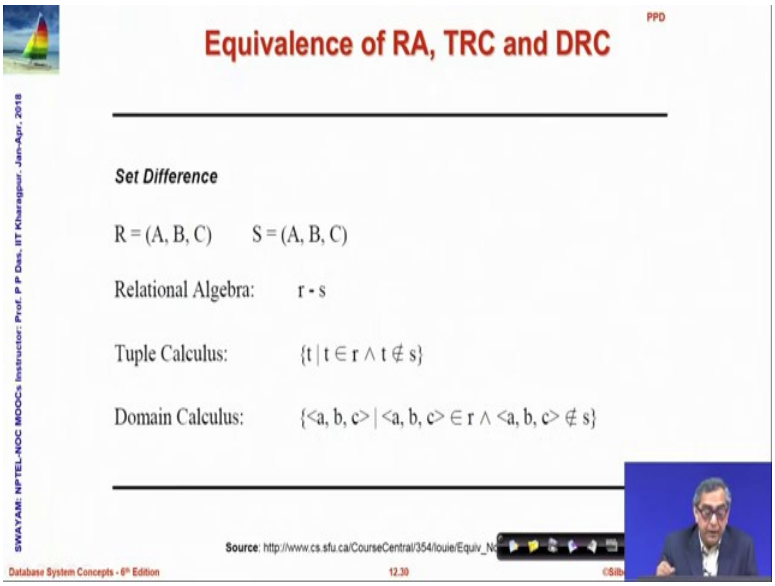
You can combine this as in the relational algebra as well as in tuple calculus. So, here you apply one relation one operation and then the other one selection then projection here you are combining this is part of projection this is also part of projection, but this condition has come from the selection and get a total predicate calculus predicate which will give you the tuple calculus expression for this combined expression of relational algebra , domain calculus will certainly happen in a similar manner.

(Refer Slide Time: 30:34)



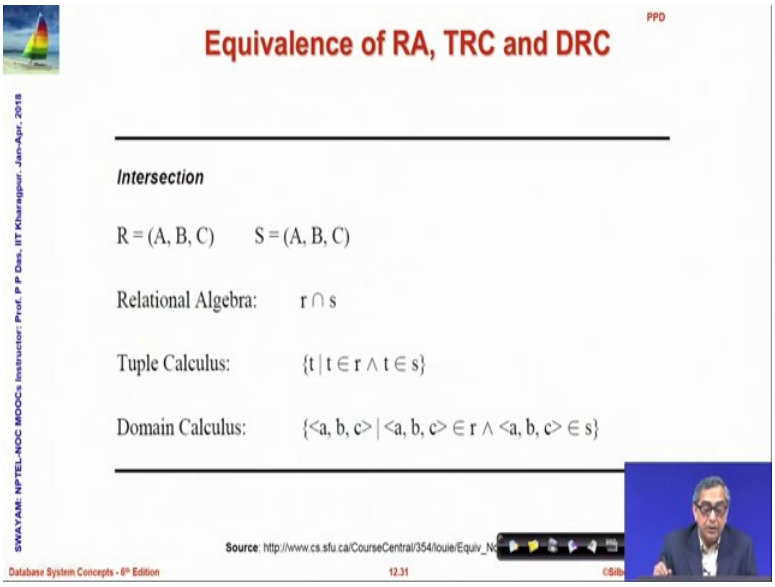
Union certainly straightforward, so you can do it yourself.

(Refer Slide Time: 30:40)



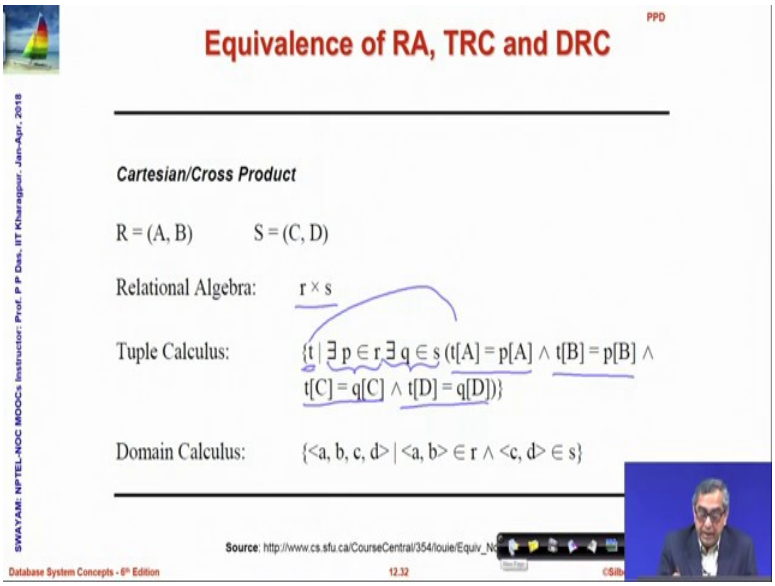
Set difference is again very straight forward because that is. In fact, in set theoretically whatever operations we have their relational algebraic definition itself is a tuple calculus formula you can expand them out and write in the domain calculus as well.

(Refer Slide Time: 30:57)



Intersection plays out in the same way tuples that belong to both r and s.

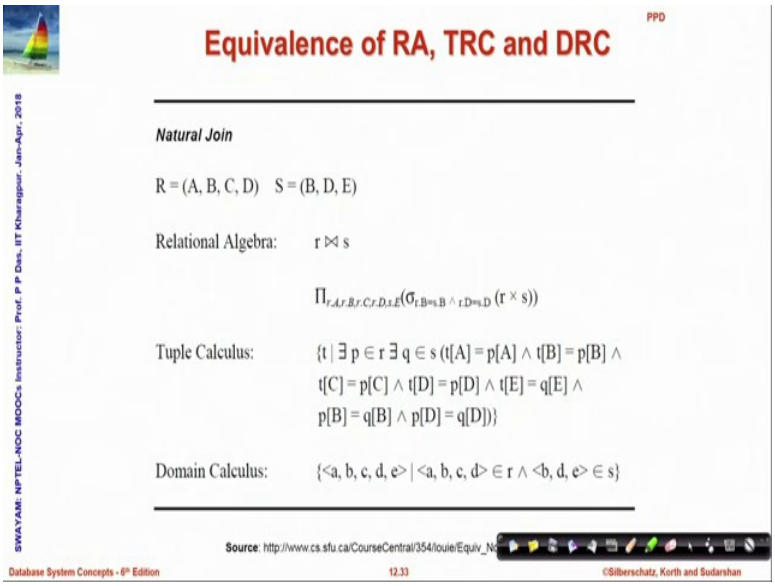
(Refer Slide Time: 31:04)



Cartesian product is a little bit more involved because all that you are saying here is if I have a Cartesian product then if that product has a tuple t. Then there must be a tuple p in the relation r (p € r)the left relation there must be a tuple q in the in s (q € s) the right relation. So, that the final tuple t matches p on the a attributes the b attributes that is attributes of relation r and the components of t matches the tuple q in the attributes of s.

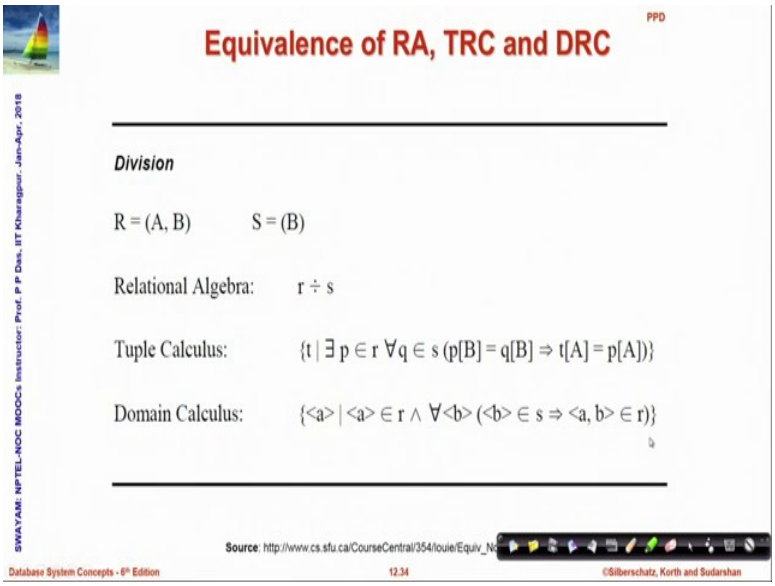
If all these conditions happen together then naturally this tuple t is a valid tuple for the Cartesian product. So, you could take examples and work this out and convince yourself that these are really equivalent.

(Refer Slide Time: 31:58)



We can define a natural join in a similar way, which I will leave it as an exercise for you to convince yourself that this relational algebra expression for natural join indeed has similar equivalents in tuple and domain calculi.

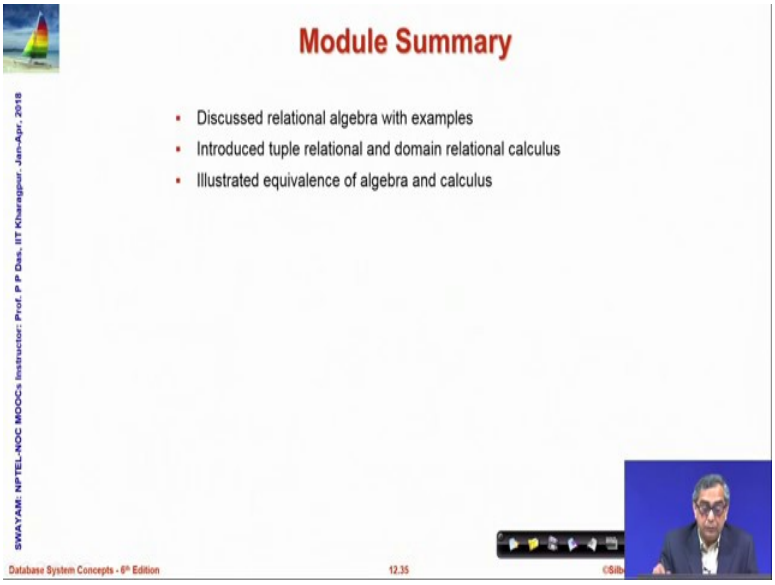
(Refer Slide Time: 32:17)



Division we just showed as a derived operation, we have not showed how in relational algebra you can write division using the other operations. I will leave that as an exercise as well, but here what I show is in tuple calculus how you can write division using the quantifiers.

Here you can see that here for the first time we do need to use the universal quantifier to make sure that while I divide that the whole of the table of s must be available against the part of the tuple part of the y attributes as we said that will be collected in the result.

(Refer Slide Time: 33:02)



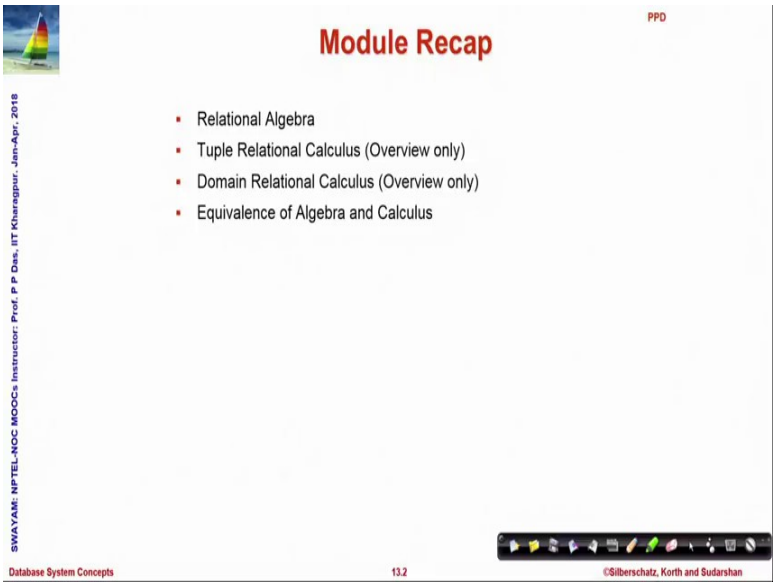
So, to summarize we have discussed primarily the relational algebra with some examples, we have introduced the tuple relational and domain relational calculus and through a set of examples. We have shown that we have illustrated that the algebra and the calculi are equivalent and I would request you to work out more examples to understand the equivalence, or if you are really enthused please try out proving their equivalence formally as well.

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

Lecture – 13
Entity-Relationship Model/1

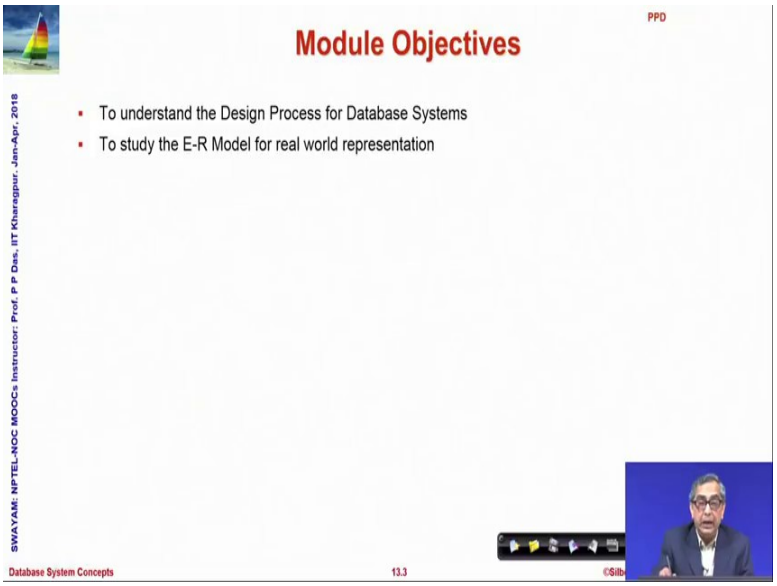
Welcome to module 13 of Database Management Systems. In this module and the next 2 we will discuss about Entity Relationship Model.

(Refer Slide Time: 00:34)



So, far we have had a good look into the SQL language, the query language and it is formal basis in terms of relational algebra and calculi.

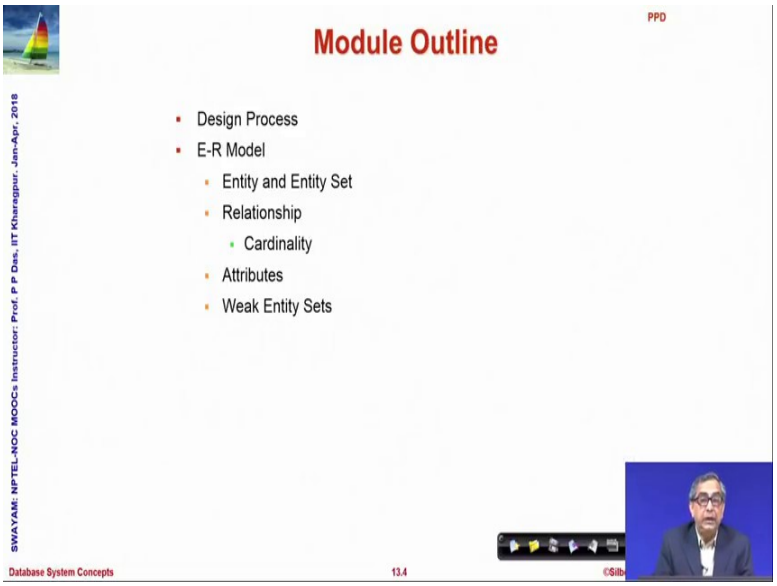
(Refer Slide Time: 00:44)



In this module we will un try to understand the design process for database systems, because so far whatever we have done we have assumed that the schema is known to us, that some instance is given to us and then we have tried to extract different query information from the relation; but now we will look into how do we model the real world and actually get into the design process.

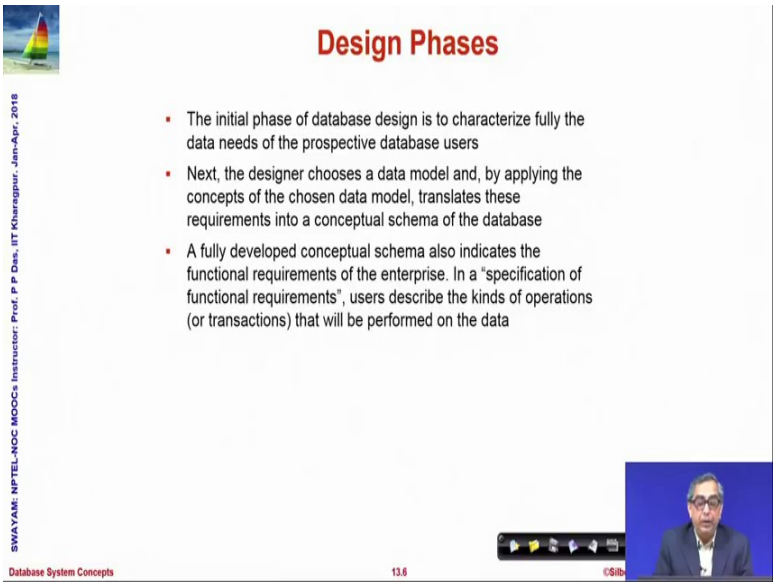
So, after an overview of the design process we would study entity relationship model, which is used to represent the real world whatever exists in the real world that will have to be represented for our use and final representation in terms of different relations.

(Refer Slide Time: 01:32)



The design process at an abstract level.

(Refer Slide Time: 01:43)



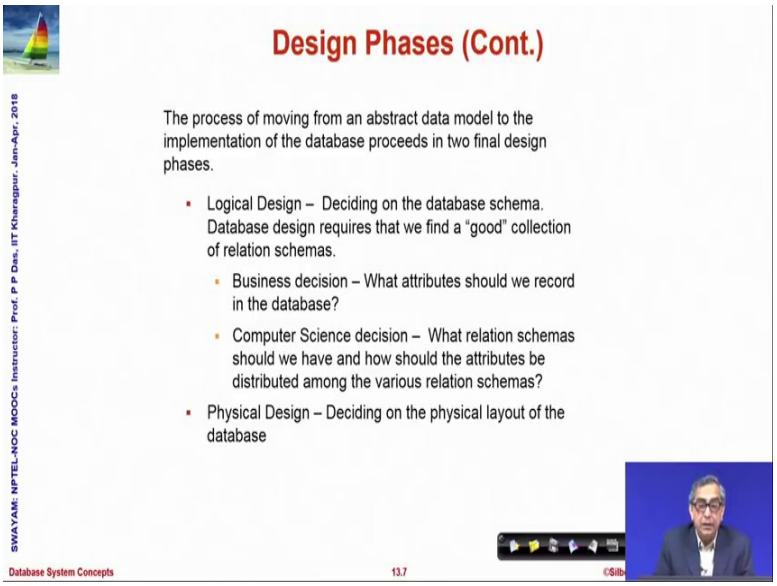
The initial phase of database design certainly has to characterize what data is required to be maintained for an enterprise. So, whether I am doing, if I am doing an university database naturally we will need to identify that what are the data needs the students need to be described, the instructors need to be described, the courses sections time slots grades examinations etc; but if I am trying to deal with an world which is say railway

reservation, then I will need to deal with stations trains date (Refer Time: 02:26) the different classes of coach that the train has and so on.

So, the initial phase is to characterize the data requirement next the designer has to choose a data model because, unless we can we cannot deal with a natural language or English kind of description and work towards getting a particular schema.

So, we will need to use a data model and apply the concepts of the data model that we choose and translate the requirements into what is known as a conceptual schema of the database which is not a not a very concrete 1. But, a conceptual one this is what grossly what I want to do and a fully developed conceptual schema will indicate my functional requirements, in terms of what usually is called a specification of functional requirements system requirements. If it will specify what kind of users will be involved what kinds of operations transactions will be performed and so on.

(Refer Slide Time: 03:43)

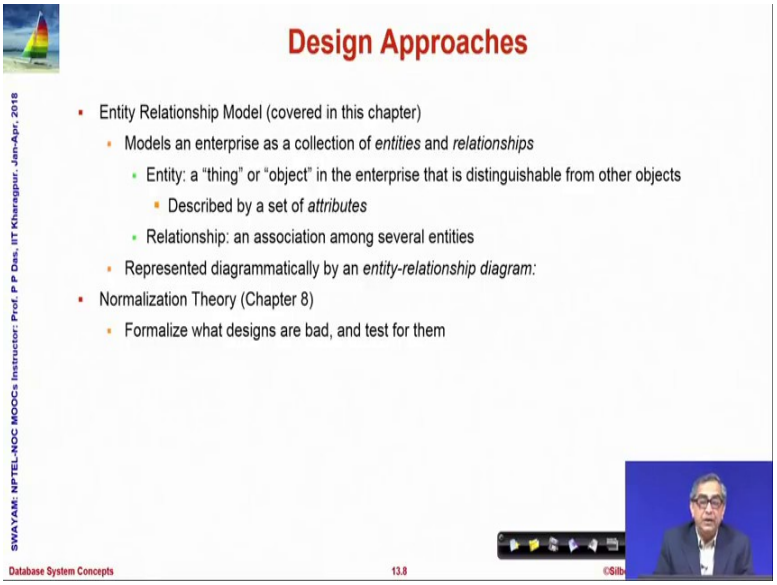


Now, once we have that kind of a conceptual model that abstract that is a conceptual more abstract data model, we will go to the next phase of the design which is finding out what is the more concrete design through a process of logical design. In the process of logical design we will first decide on the database schema, we need to decide on what is a good schema.

So, there are principles to say that what is good and what is not so good, we need to make business decisions to find out which attributes we record in the database; we need to make computer science decision as to how the relational schemas will be interrelated between themselves, how the attributes will be distributed and at a last phase we need to also decide on the physical design which will tell us what is the physical layout of the data.

So, conceptual design refined into logical design finalized with physical design is our gross process of design.

(Refer Slide Time: 05:06)



Now, in this for the conceptual design we primarily follow a model called entity relationship model; that tries to identify the collection of entities and relationships. An entity is nothing, but it is an object is a thing that is distinguishable from other objects. So, if I say that student is an entity then the student is distinguishable from another entity course both of them are distinguishable from a third entity instructor and so on.

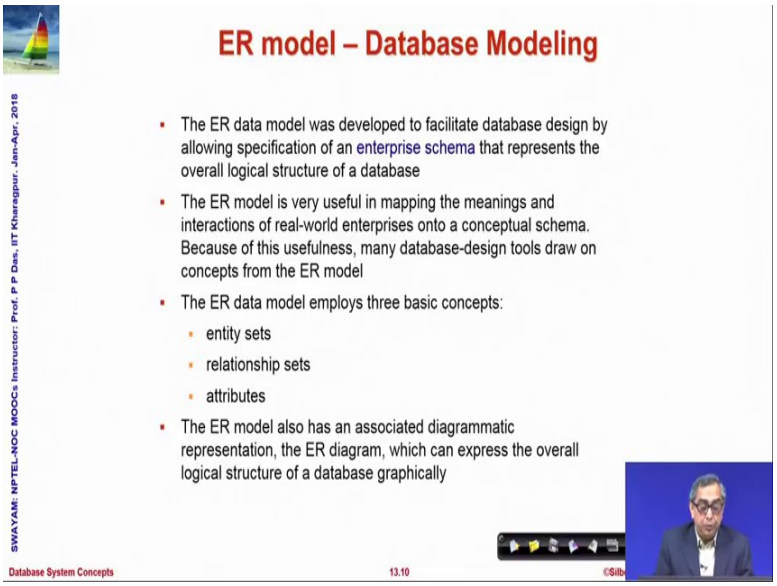
So, every entity for the purpose of distinction is described by a set of attributes or properties and these entities will have relations between them. For example, you can say that a course will be attended by students; students will be advised by instructors. So, this attended by advised by these are relationships or association between several entities and the model which represents initially diagrammatically and then in textual form this kind

of relationship is known as the entity relationship model or the entity relationship diagram.

We will then use it to get a relational set of relational schema which subsequently we normalize; the normalization is nothing but refinement of the design which improves a design to make it better in terms of correctness, in terms of ease of manipulation performance and so on. So, it basically removes bad designs from the database and converts them into good designs; we will talk about this normalization theory later in the course. Right now we are interested only in the entity relationship model, which will be used for conceptual design and then will give us the basis for the logical design in terms of the schemas.

So, let us take a deeper look into the entity relationship model and entity relationship model as I said is developed to facilitate the database design.

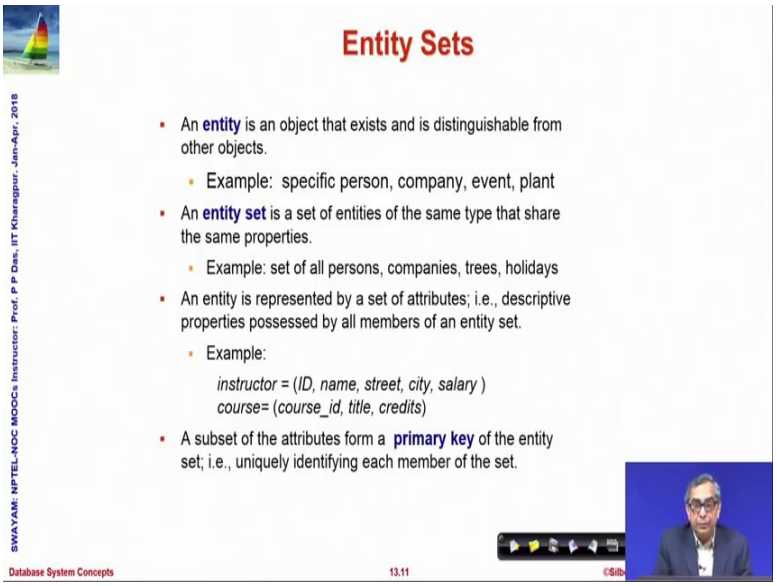
(Refer Slide Time: 07:45)



Get the overall logical structure it is useful in mapping the meaning and interactions of the real world in terms of certain diagrammatic schemas and it employs 3 basic concepts entities or entity sets we talked about entities, all entities that share the same set of properties like if student is an entity, then the collection of student is an entity set a instructor is an entity collection of instructors is an entity set.

So, all entities in an entity set will share the same set of attributes, will have relationship sets which define relationship between multiple entity sets and certainly in the process will use make use of attributes. These are the 3 key components of an ER model it also has a ER diagram as we will show soon.

(Refer Slide Time: 08:51)

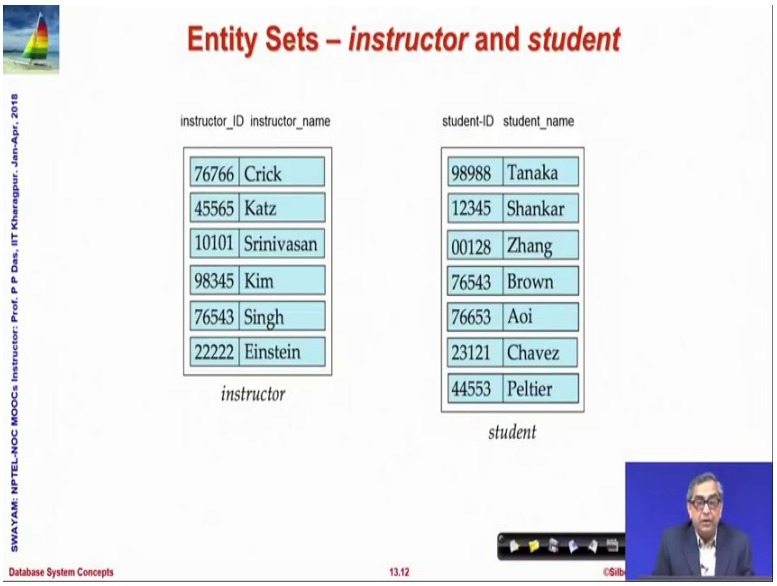


So, as already defined entity is an object that exist and is distinguishable from other objects, entity set is a set of entities of the same type that share the same properties and an entities is represented by the set of attributes or properties that describe it.

So, when we say instructive for example, if we say here these are my attributes you have already learned this in terms of studying SQL. So, it has there is 5 attributes and these 5 attributes together or the values of these 5 attributes for a particular instructor defines my entity set instructor, collection of these attributes define my entity set courses. So, these are my different entity sets that exist that can be defined.

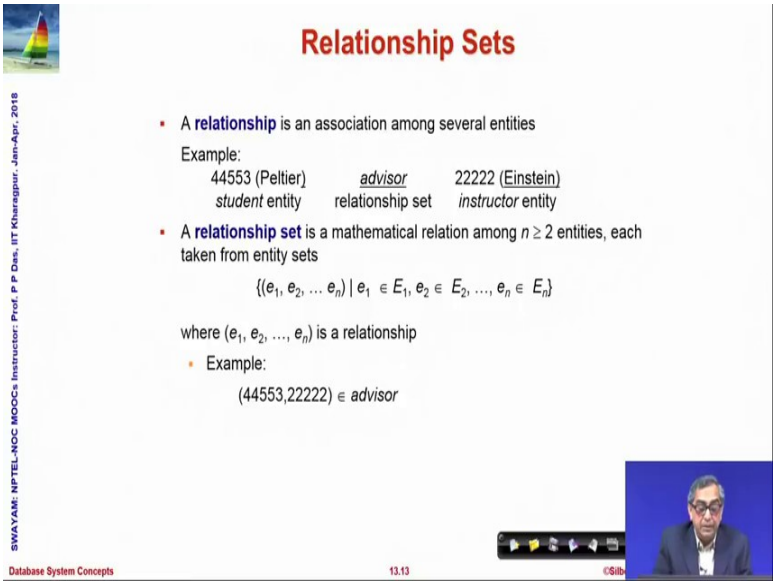
So, a subset of attributes in the entity set forms a key called the primary key, which can uniquely identify every entity in that entity set we have already been familiar with this concept of primary key the same concept continues.

(Refer Slide Time: 10:00)



So, these are examples of entity sets instructor with 2 attributes and student with 2 attributes as well.

(Refer Slide Time: 10:14)

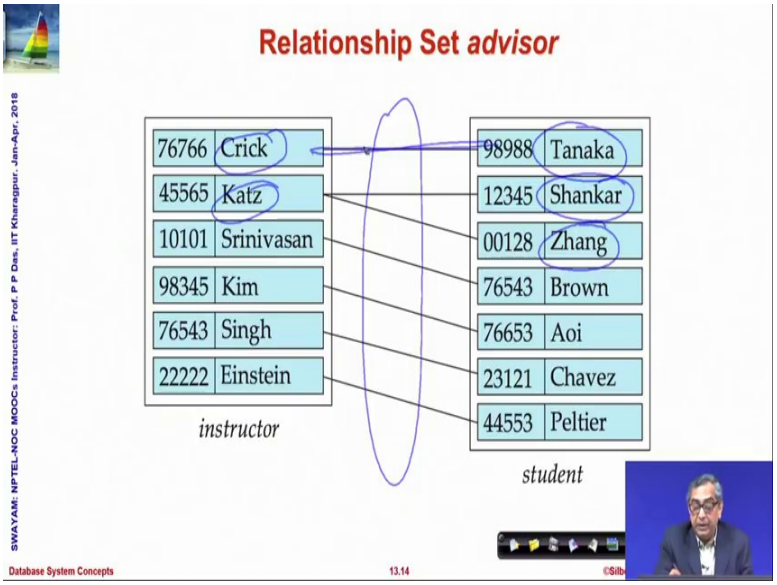


A relationship is an association among two or more entities, so here we have an entity here shown as a student this is a student entity, identified by the student id which is a primary key in the student entity set. We have an instance of an instructor entity identified by the id of the instructor Einstein, which identifies any instructor uniquely and then adviser is a relationship set which relates these 2.

So, what I we want to mean is if I say adviser relates 44553 to 2222, what I want to mean is peltier the student peltier is advised by the instructor Einstein. So, whenever we relate two or more entity sets like this we get relationships. So, a relationship is a mathematical relation Emma more than two or more entities, each taken from the entity set.

So, you can see that it can have components e 1 e 2 e n, n entity sets and each entity e 1 should belong to entity set capital E1, e2 should belong to entity set capital E2 and so on and is called a relationship we have already seen the advisor relationship as above. {(e1,e2,e3,e4….en)| e1€E1, e2€E2, e3€E3…..}

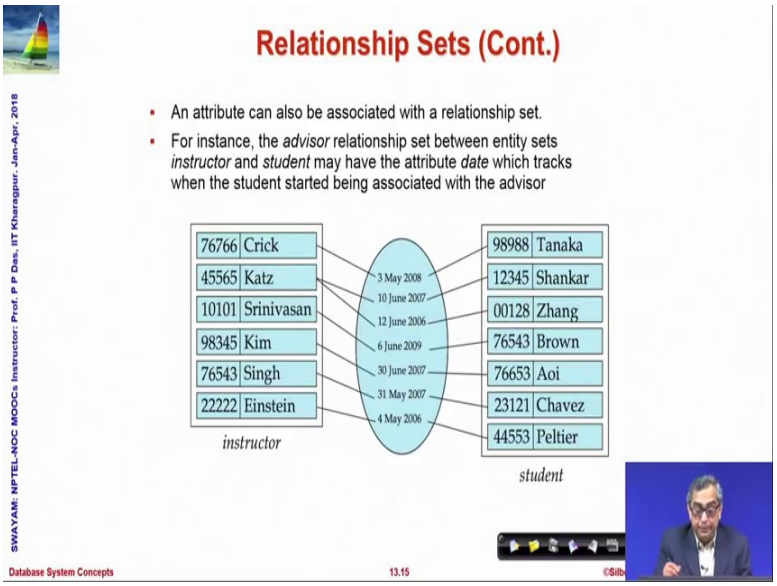
(Refer Slide Time: 11:58)



So, here what we show is a relationship advisor by these arrows these lines. So, what we are showing is this connection between these two, show that this student is advised by this instructor where as you can see. So, crick advises Tanaka where as Shankar and Zhang both are advised by Katz.

So, this group of associations between instructor and student is the gives me the relationship adviser as to who advises whom.

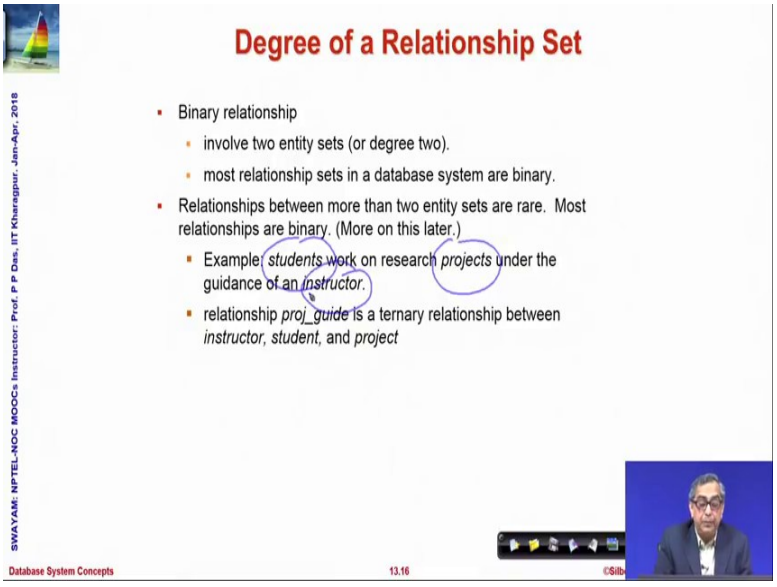
(Refer Slide Time: 12:39)



A relationship also like the entity sets the relationship also can have some additional attribute. For example, when I say that crick advises Tanaka I may associate an attribute date type attribute set third May 2018, to mean that when did this process of crick advising Tanaka started, we can it can be some other attribute also.

So, all that I am trying to highlight is attributes can be assigned to relationships as well.

(Refer Slide Time: 13:15)

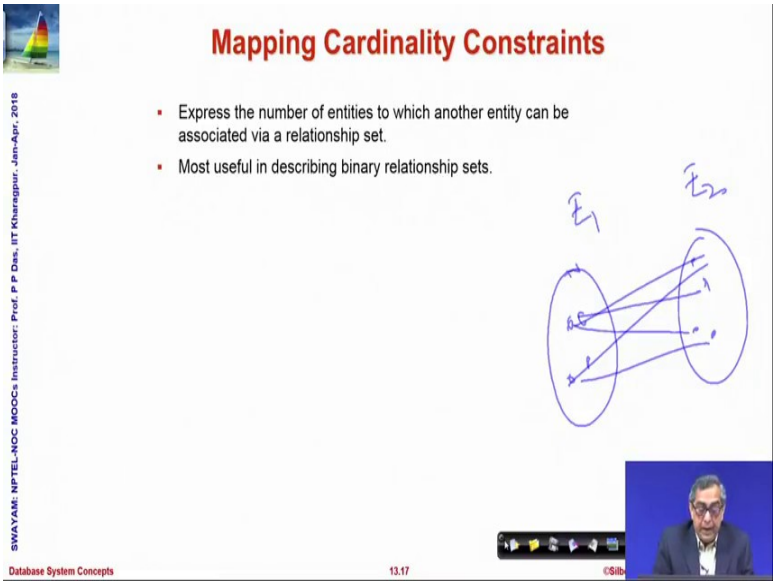


Now, how will a relationship span out, we have said that a relationship must involve two entity sets. So, primarily relationships are binary it involves two and most relationships

in most databases are binary in nature, but it could be that there are we will see later that there are possibilities of having relationships which are more than binary ternary and higher.

So, here are example students works on research projects under the guidance of an instructor. So, here we have as you can see student's research projects and instructors, so there are 3 entity sets. So, if I want to maintain a relationship of say project guidance between them then that turns out to be a ternary relationship we will talk about this more later.

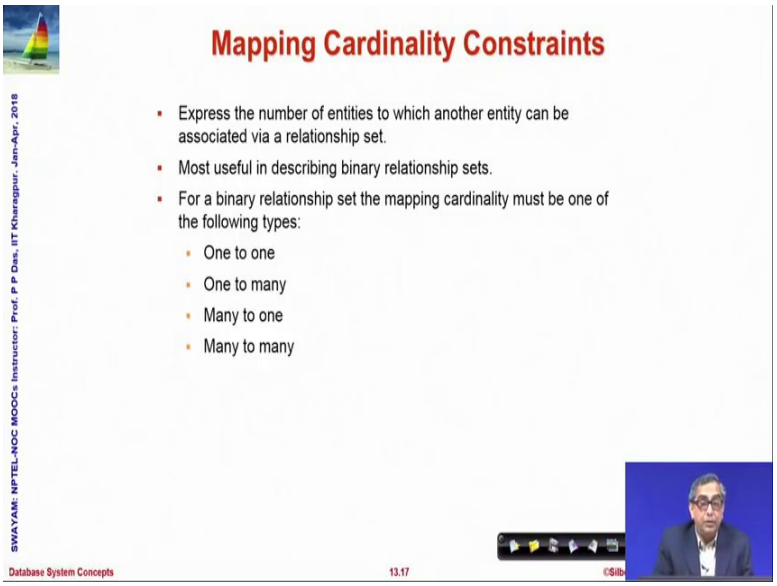
(Refer Slide Time: 14:16)



There are constraints in terms of the cardinality of the relationship, the cardinality basically talks of that when we have when I have a relation entity set E1 and identity set E2.

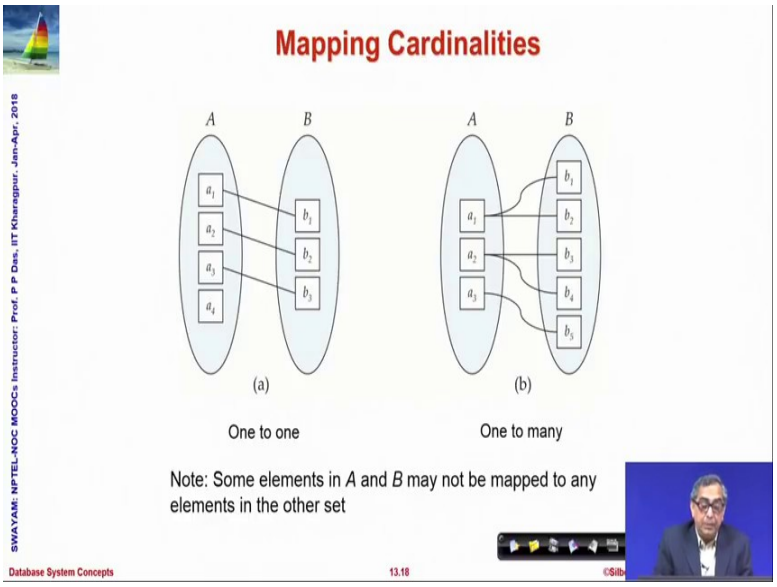
So, there are different entities in them and I have different associations between them, then the question is how many of the entity of one entity set is related to how many of the entities of the other entity set and certain types of cardinality measures are very important.

(Refer Slide Time: 14:58)



To track and we say it is whether it is 1 to 1 to many many to one or many to many.

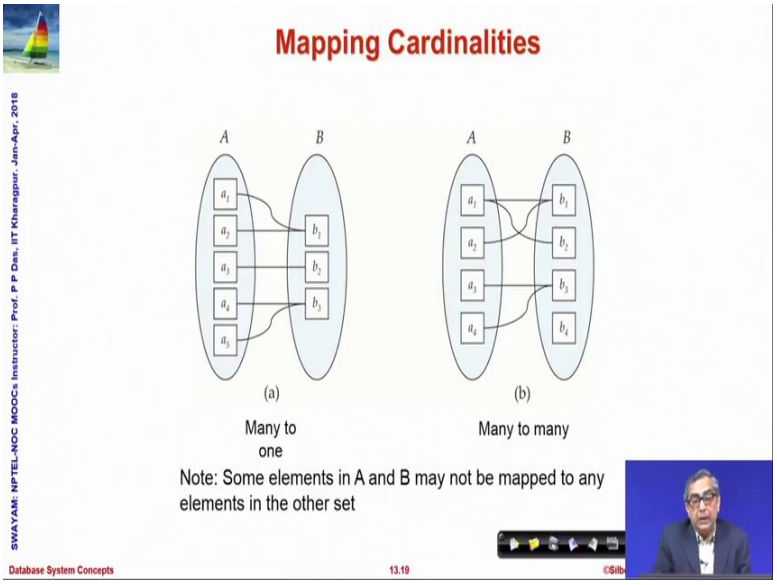
(Refer Slide Time: 15:08)



So, here the examples or the schematics, so in the first one in the diagram A you see that every entity from the entity set A relates to exactly one entity in the entity set B or you can say at most one entity in then they decide B similarly every entity in entity set B relates to exactly one entity in entity set A or at most 1 entity and entity set a if this holds then we say this relationship is one to one whereas, in diagram B you see that a 1 relates to b 1 as well as b 2 a 2 relates to b 3 as well as b4.

So, 1 entity in A relates to more than 1 entity may relate to more than 1 entity in B, but if you look from B side every entity in B is related to at most 1 entity in A then we say from A to B it is 1 to many.

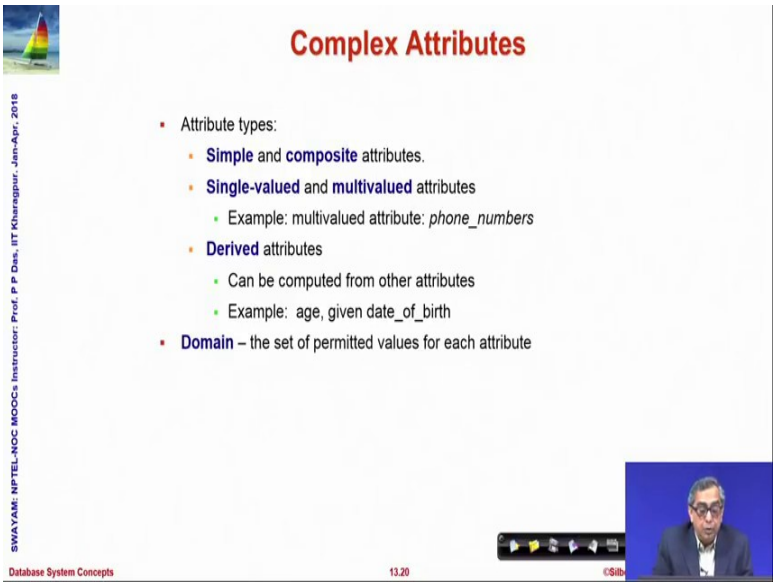
(Refer Slide Time: 16:04)



Now naturally since I can put the relations in any order as we have one to many if you look in the other direction it becomes many to one. So, many to one is from A to B many to one is where more than one entity in set A may relate to one entity inside B, but all entities in set B relates to at most one entity inside A and when there is no restriction at all that is any number of entities in set A may relate to any number of entities inside B and any number of entities in set B may relate to any number of entities in set A we say it is a many to many relation.

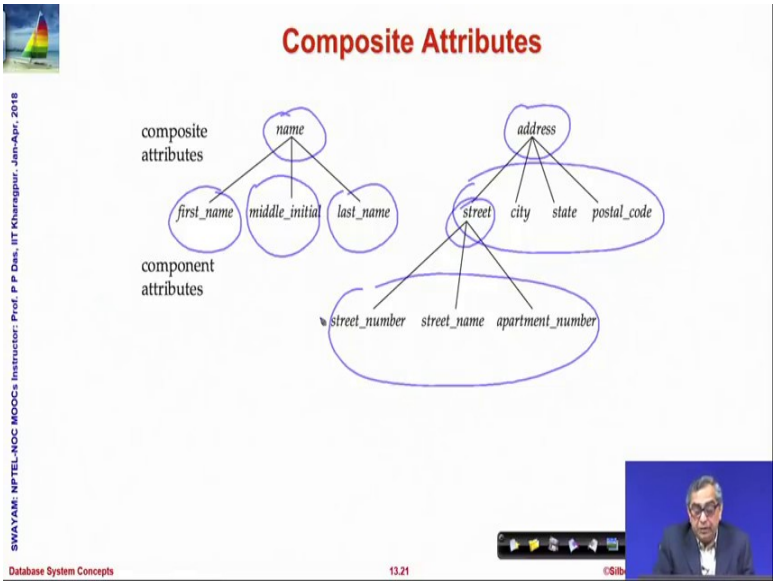
So, we have one too many one to one, we have one to many and many to one and we have many to many and it often helps in the design to be able to characterize which type of relationship we do have.

(Refer Slide Time: 16:51)



Coming to the attributes we can note that attributes are of different types, one is they could be simple or composite a simple attribute is just one single domain value like a salary number like an id like a name string and so on; whereas, a composite attribute may comprise of multiple parts.

(Refer Slide Time: 17:16)

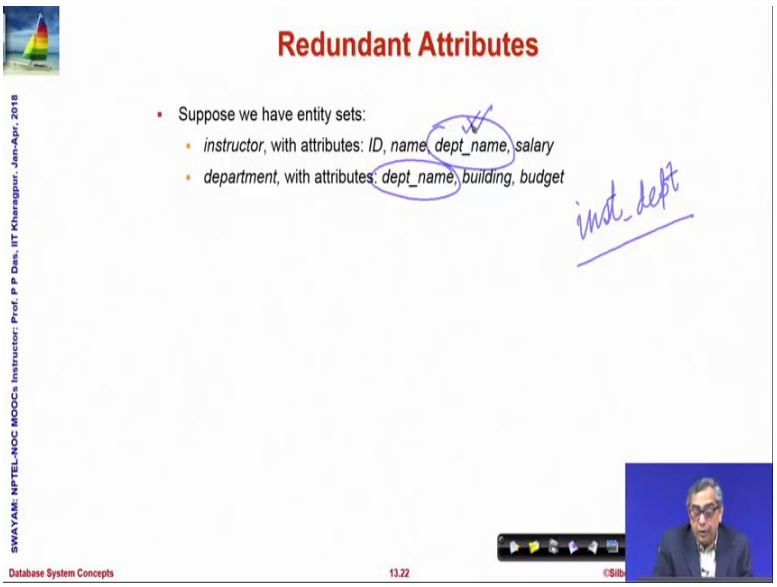


So consider this, this is a composite attribute, so name is an attribute if I think of then it has different parts, it is a first name middle name last name if I think of address it has. So, many different parts then street itself has so many different parts. So, whenever an

attribute is comprised some more of the components, when it is not a simple value then it is called a composite attribute we will see how to handle them; then some attributes may be single valued, for example a person has a has 1 name let us say, but has one address, but may have 2 or more phone numbers, the attributes which can take more than 1 value is known to be multi valued attribute.

So, we also need to specify whether certain specifying in the design whether certain attributes are single valued or multiple valued multi valued; of course, single value attributes are easy to deal with if it is multi valued we need to do some design changes. Certain attributes can be derived for example age, now I cannot keep the age of some a person in the database because, with every day the age changes. So, what we typically keep is a date of birth and the age is computed on the day when the particular query is made to find out what the age is so it is called a derived attribute and each 1 of them will have corresponding set of domains.

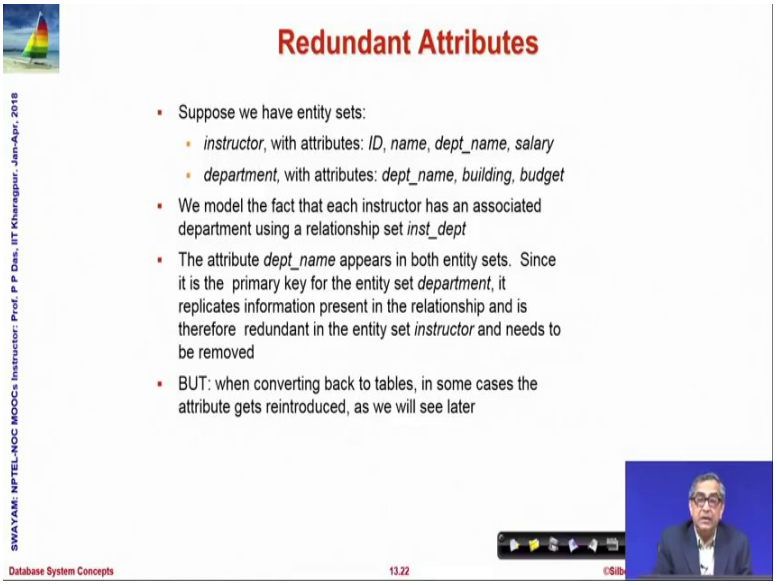
(Refer Slide Time: 18:53)



Some attributes in the design may turn out to be redundant also consider this you have already seen this this is an instructor, which has a department name along with the different attributes and certainly I have a department table, so which department relation which gives the details of the department. Now since every instructor belongs to a department, so naturally we might want to have a relation inst_department which could give the instructor and his or her department name. So, if we maintain that then this

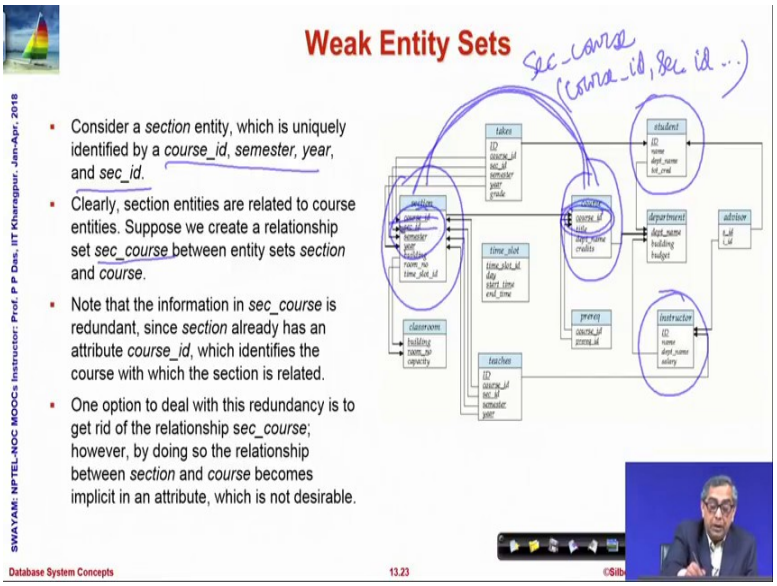
becomes a redundant attribute this is not required, because if that information is already there in this relation.

(Refer Slide Time: 19:49)



So, in several cases there is a question of whether we maintain some information in terms of a relation or we can make that directly include that directly in the entity set and get rid of that relation. So, if I have the inst_dept relation and then the attribute department name appears on both these sets, inst_dept as well as on the instructor and there is duplication replication of the data which we want to avoid. But we will see the different cases when which style of design, whether we would be better to maintain the department name as a part of the instructor relation or it would be better not to have it there and have a separate relation which maps instructor id against the department name.

(Refer Slide Time: 20:48)



Finally comes a concept of weak entity sets you need to understand this a little bit, consider the university database example. So, we have courses we have students we have instructors and we have section, a section is if a course is large then it needs to be taught in multiple sections. So, for the same course at the same semester in the same year I may have different sections, in which the students are divided and naturally there could be multiple instructors each teaching 1 section of that course and students will be distributed on the sections not on the course.

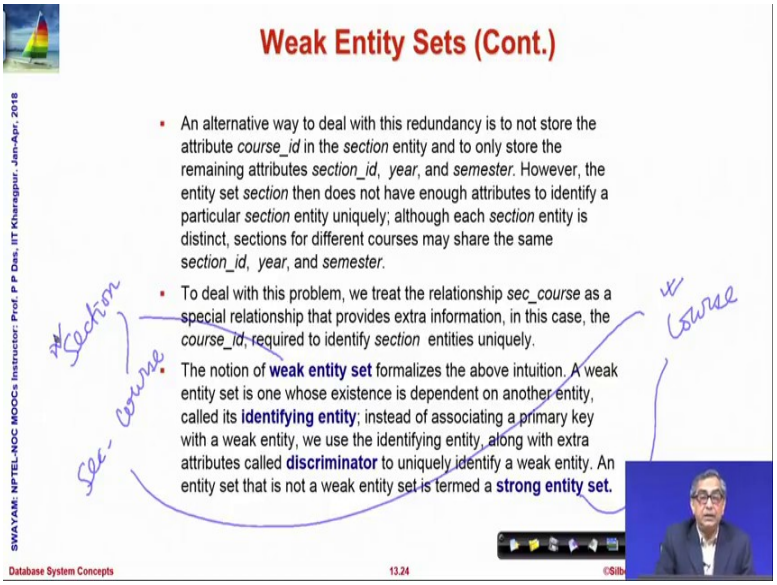
Now, consider this section entity, if you look into this then this is how what we we maintained we did a course id semester year and section id, but if you look into specifically and if you want to now know you know that there is a section and there is a course. So, you may want to relate these two section with the course and set up an entity between them. So, what will it relate it will relate the course id of the course with all of these, but the course id is already there as a part of the section right. So, you would say that well it is not required to have the course id, since it already has that and it it identifies it.

So, we can can we remove this course id from here, well if we remove the course id now we have a different problem. If you remove the course id then you have section id semester and year. But this does not uniquely represent the tuples of this relation

because, there could be 2 section as in the same semester in the same year for 2 different courses how do you distinguish them.

So, you get into a situation where the course the section gets identified uniquely provided, either you know the relationship between the section and the course in terms of the sec_course relationship or you include the primary key of course, into the relation section which we did in the design and this is not a coincidence this is something which happens regularly and is is the characteristics that specify the existence of weak entity sets.

(Refer Slide Time: 24:11)

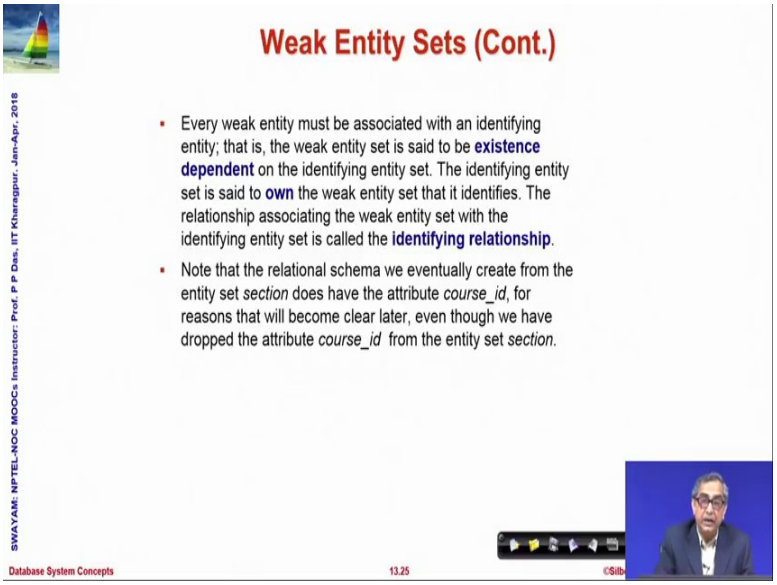


So, the weak entity set is one whose existence depends on another entity set. So, if I just say section having section id year and semester then it is not uniquely specified, until I have a relation section course which relates the section to the particular course id. When such relationships are used to identify entities of a particular entity set, then the unique side the course side which is unique is known as the identifying entity and the other attributes in this case section id year semester are known as the discriminators.

So, we have a relationship between a weak entity set which is section, we have a strong entity set which is course why is it strong because course is identified by course id itself. Section is not unless you have a set course kind of relationship set between the course and the section which specifies that well, for this course this is the section this is the year this is the semester. So, this is the identifying entity through which the entities of this set

gets specified and whenever that situation happens then we say that we have a weak entity set.

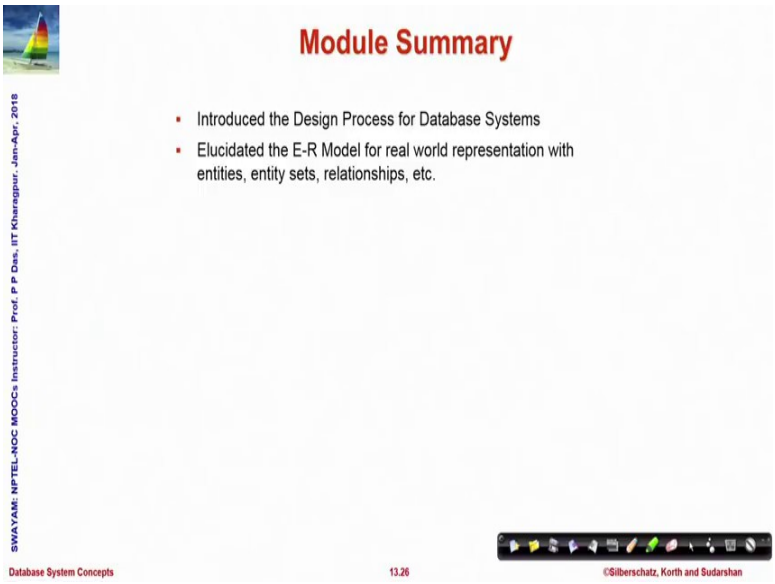
(Refer Slide Time: 26:21)



So, weak entity sets naturally cannot happen by themselves their existence dependent on identifying entity set and the identifying entity set owns the weak entity set. So, the courses in that way own the section and the identifying relationship between them is necessary to uniquely identify every entity of this weak entity set or section in our case.

So, this notion is very important for the design as we will see that the relational schema that we eventually created, in this case from the entity set section we did include course id as a part of the primary key not using the sec course kind of relationship and we will show how this design style for dealing with entity weak entity sets influences the different database designs. So, weak entity sets are critical notions that you need to be aware of need to be confident of.

(Refer Slide Time: 27:34)



So, in summary we have introduced the design process for database systems I will just quickly recap, the first stage is identifying the data items which is leading to the conceptual design, which will primarily do in terms of the entity relationship model identifying the entities the entity sets, the attributes that define the entity set, describe the entity set, the subset of attributes forming primary key that uniquely specifies every entity set, every entity in the entity set and the relationships typically binary may be non binary also, relationships that hold between the different entity sets.

So, this is the conceptual design that will lead to more detail logical design of how the relationship should be organized, what is the cardinality of that, what kind of attributes do I have, whether it is sample whether it is composite whether certain attributes are derived or not. So, all those are different aspects will have to be detailed out and we need to identify what are the weak entity sets and what are the strong entity sets, what are the identifying entities and with that we could complete the logical design and then we will need to make it in terms of it express it in terms of a relational schema.

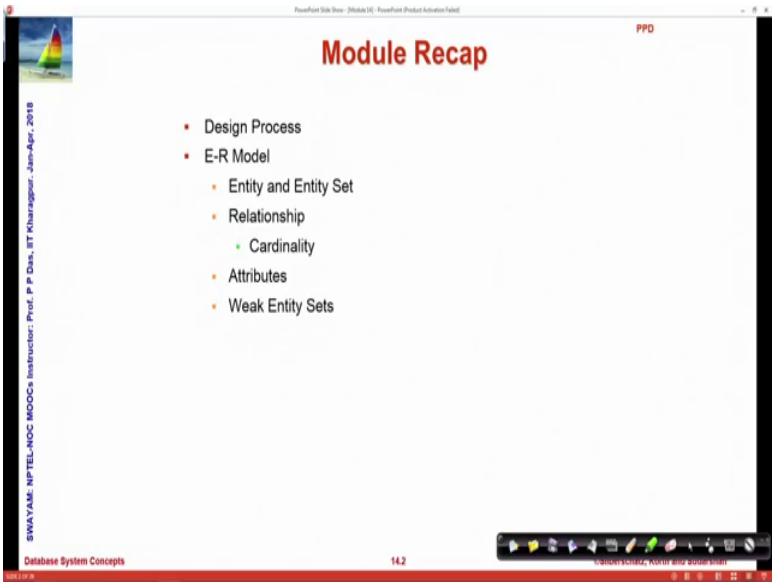
So, in this module we have just taken a look in the first part the entity relationship model and the very basic of how the conceptual design. We will go forward, in the model we have seen all the different primitives required to represent the reality represent what holds in the real world.

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

Lecture - 14
Entity-Relationship Model/2

Welcome to module 14 of database management systems. In the last module we will started our discussions on the entity relationship model.

(Refer Slide Time: 00:29)



We will continue that in this module as well, and actually conclude it in the next module. So, these are the items that we had discussed in the last module.

(Refer Slide Time: 00:39)