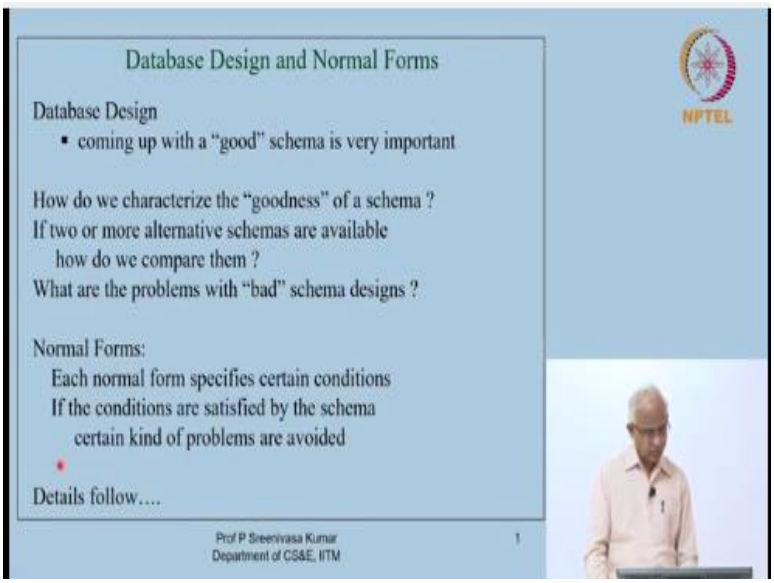
Database Systems
Dr. Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology-Madras

Lecture-19
Normal forms-Introduction

So we are going to start of a new module today.

(Refer Slide Time: 00:23)



This is to do with database design and normal forms. Now you have been studying various database schemas and posing queries on them and learning how to make use of schemas. But then you also have an exercise problem where you are, you know, designing a information system for some domain, which you have chosen. So in this context, you would see that coming up with an design is very important.

The design is the complete schema for the information system. And there will be certain choices there, when you are coming up with a design, you would have to decide about you know, what attributes should constitute one relation, should this attribute be in this relation or some other relation right. So there are various kinds of choices that come up there. And in this context we would come up with this question as to how do we know whether the schema that we have designed for the implementation system is good.

And if we have some alternate schemes for the database, then which is better ok. So these are the various kinds of questions that we will come to us, when we are looking at alternate designs for information systems, the schema lines. Now so in this module what we are going to look at is certain principles that have been developed in kind of evaluating this database designs ok.

So one of the important factors that kind of affect the design and also performance of the information system has been recognized as the issue of redundant storage of a piece of data. So by this what we mean is, if there is some one piece of data, so it of course, is belongs to a certain attribute. So if this attribute you know occurs in slightly different form in multiple design multiple relations in the design.

Then the corresponding value, which is supposed to be same in all those places will also occur redundantly in 2, 3 places right. So if a piece of data occurs redundantly unnecessarily in multiple places, then we call that as a data redundancy. So this kind of data redundancy is seem to you know kind of affect our performance also performance of the information system also. So you actually might wonder, how does that affect the performance because the storage is notice not a big major issue.

I mean you have lots of storage and you know, disks and external storage is secondary storages available abundantly. But the main issue here is that when you consider the ultimately the purpose for which the information system has been build, you see that it has to support transaction processing, it has to support everyday operations of the enterprise for which it has been designed. And in these transactions, these application programs that are invoked by end users.

And they run 1000s of times per day, you know organize this database and they will be accessing the data. So imagine a particular transaction, trying to update a value for a particular update some value which is redundantly occurring in multiple places. So if it is not updated in all those places where it is occurring, then it will lead to inconsistency in the database, so we do not want that to happen.

So we will require the application to actually update it in all the places where it has been stored. Now if you try to update it in some 2, 3 different places in which it has been stored, all those will be in different files on the disk system and then you will have to get access to them and then actually modify all of them. So we will later on see how do you access disk files and then carry out this operations.

But at this stage it suffices to remember that this operation is a costly operation compared to memory operations, accessing disks, file and then updating. So if you have to do at multiple places, it will add up to the time that the transaction takes in order to complete the task. Now if the amount of time the transaction takes to complete the task is high, then the number of transactions that the whole system can handle per unit of time, which is called the throughput of the system will obviously get affected, it will become lower.

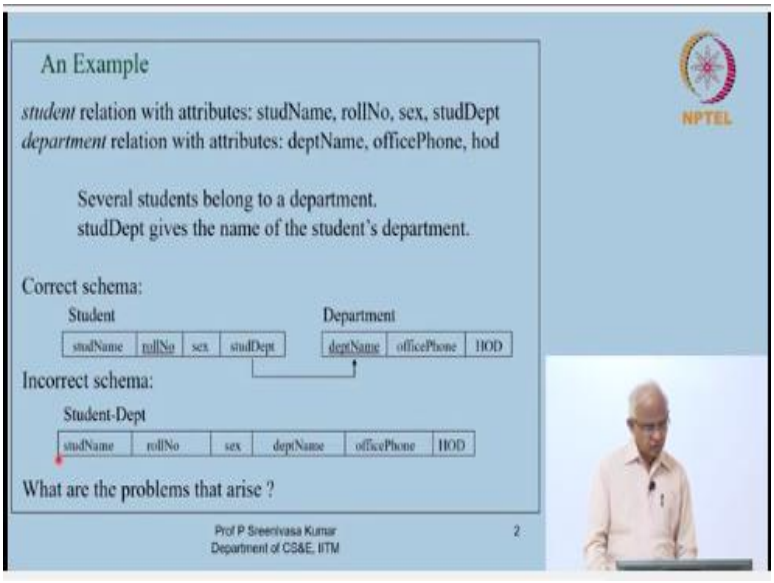
So and if the throughput becomes lower, then the response time of the system in the presence of 1000s of users for the system will also go down. So, all these are the various implications of having redundant data. So how does redundancy occur and in some simple kind of designs, which contain just about say some 1015 schemas or something like that some 20, 30 attributes you may be able to figure out many things.

But in practice these database designs can tend to have huge number of relations and then they involves some 100 to 200 attributes or something like that. In which case it kind of becomes difficult to kind of you know, figure out what is happening in the scheme. So are there any theoretical tools that we can make use of in studying this properties of these schemas. And then make use of them in comparing our schemes or if we already put in a particular you know design into operation.

And then we are now finding some kind of efficiency issues, can be kind of look back and analyze the design and then see where exactly our problems arise. So with all these goals in mind a certain kind of theory called the dependency theory has been developed by researchers in 80s and 90s. So in this module, I will give you an introduction to this theory will tell you the fundamental principles that are there in this particular theory.

So these are the questions how do we characterize the goodness of a scheme and how do we compare them. So in this process, what we will do is introduce what are called normal forms, so normal forms are some kind of conditions that are laid down. So and with a guarantee that if these conditions are satisfied by the scheme, the database and design you have, then certain kind of problems do not occur it is the guarantee that the normal forms give you, so we will now go into the details.

(Refer Slide Time: 08:55)



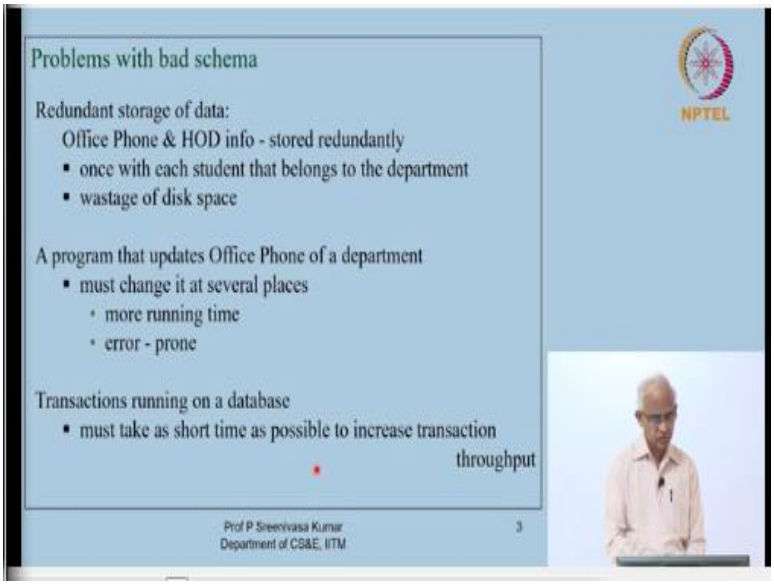
So let us start with a as usual, let us start with the example to kind of get some idea of what the things that we are talking about. So I am going to contrast a correct scheme with an incorrect scheme and then see what are the kind of issues that come up here. So let us say ok these relations are not necessarily the same as what you have seen earlier. But I am going to define them here, student relation as student name, roll number, sex and student department just for simplicity.

And roll number is the key and department relation has a department name, office phone and hod, hod is an employee, so employee ID of the hod will be there. Several students belong to a department and this student department gives the name of the students department and it is a foreign key referring to the department name and department name is the key here in the department relations.

So this is a good design, it is a correct design, so we have put student name, roll number, sex and student department here. And made it is a foreign key that refers to the department name and office phone and hod. So what if somebody you know, comes up with this following scheme, student department kind of so I do not want to many relations to handle. So let me put all the information in one relation, so let me put all this in student name, roll number, sex, department name, office phone, HOD put everything together.

So you have some intuitive feeling as to this scheme is not that good actually for some reason, you may develop some intuitive feeling that this is not good. Let us see why what are the actual problems that come arise.

(Refer Slide Time: 10:58)



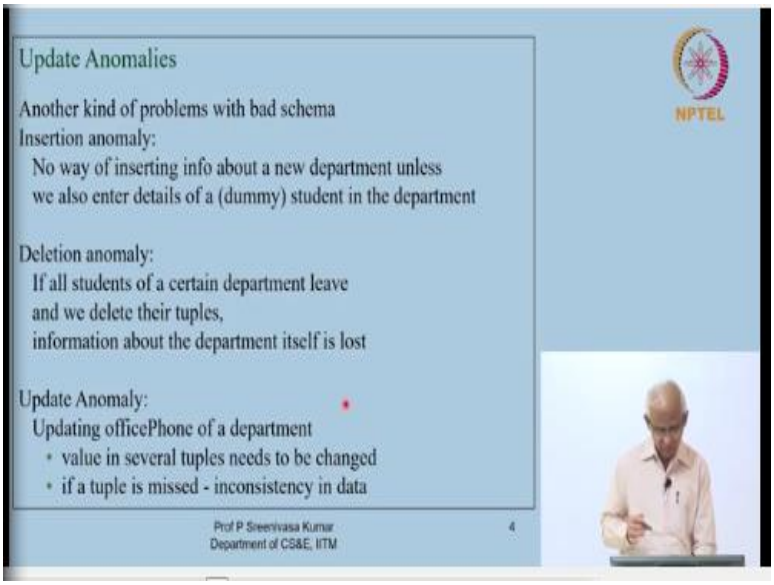
First thing is there is redundant storage of data, where is this redundant storage of data. Now several students belong to a particular department right. So some 100 students belong to 1 department or 200 students belong to 1 department. And you can see that in this scheme the department information as name and what is the phone number of the department and who is the HOD of the department is kind of tagged along with all the students that belong to that particular department.

Because there is no other way right, so if you say a student name, roll number, sex and department name and office phone. So we have dropped off the student department which served as a foreign key and then refer to this right, we dropped off that. So we directly want to give all the information about the student's department in the scheme itself. So now you can see that the office phone and HOD information and the department name information is stored redundantly.

Department name of course was there earlier also in the scheme, but definitely office phone and HOD information is stored redundantly once along with each of the students belongs to the department and it is a wastage of the storage space. So now a program that kind of that is required to update the office phone of a department. Obviously has to change the value at all these places if the 100 students belong to the department in all those 100 tuples it has to change the place.

And the main issue is this learning time because you have to change it. Now the issue with running time is the transactions that are running on the database, we would expect them to finish fast. So that the throughput will be high for the system. So if a lot of work is being done by transactions then it will affect the throughput of the system, ok. So not only this issue there are a lot of other issues with that scheme.

(Refer Slide Time: 13:19)



These are slightly different kind of problems not necessarily with the redundancy, but look at here this is what is called a insertion anomaly. There is no way of inserting information about some new department in this in this scheme unless you know, we also enter some details of some dummy student because unless the student information exists in that scheme, the department information cannot exist, right.

So you have to put some dummy student information and then put the new students new departments information. So you are kind of force to introduce some details of some non existing dummy student in order to introduce information about a department. All this because we have chosen to combine information about students and departments together into one scheme. You are able to see that and you can imagine similar set of problems that come up that arise there.

This is called deletion anomaly, if for some reason all students have a certain department leave, then how do you handle the issue you will be dropping the student rows if the student leaves the institute for some reason you will drop off that row. So if for some reason all these students in a particular department leave. Then when the last student leaves, then you are actually dropping off that row and so information about the department itself is gone.

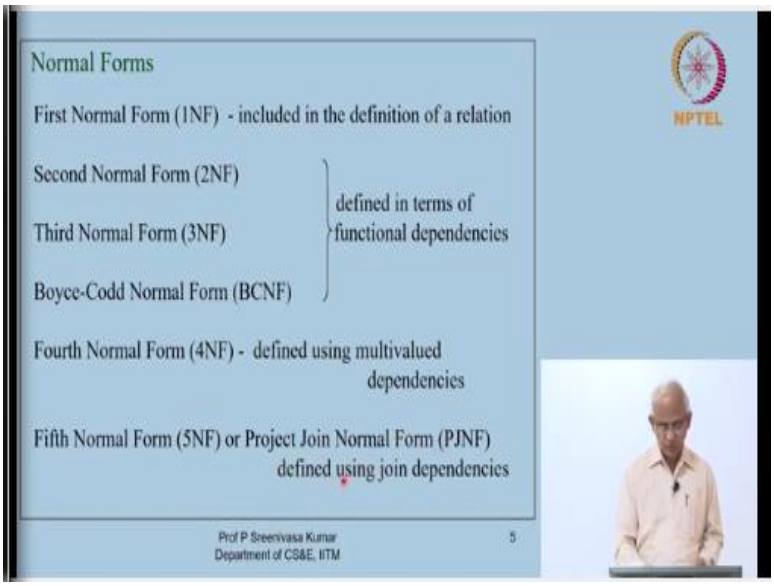
So that we did not actually plan for, so we want to keep the department alive hoping that some students will join the department at some point of time, right. So we did not expect that, but it will happen in the scheme. So if students leave then at certain point of time, the information about the entire department might get lost. This we have already seen, this is called update anomaly, updating office phone of a department in value in several tuples has to be changed.

And the issue here is that if for some reason you missed out one tuple then or a few tuples to you know update this thing. Then some tuples will say office phone is this, some other couples will say office phone is this and that is obviously a data in inconsistency and we do not want to have that in the database ok. So these are the various problems that arise with you know bad schemes.

Now, the question is how do we in this case it was kind of obvious that saying that ok, you had 2 you know logically different kind of entities the student and the departments. And what you have

done is to somehow you know forcibly put the details about both these things into one scheme and then you got into trouble ok. But in a database that may have you know several 100 attributes, how do we kind of detect this, how do we handle this that is the one question.

(Refer Slide Time: 16:47)



So we need some theoretical tools for characterizing the issue first, and then detecting and then removing that issue. So in this process we will introduce what are called normal forms. So there are various kinds of normal forms they are written like this first normal form, second normal form, third normal form and a stricter version of third normal form called Boyce-Codd normal form etc.

So the first normal form is actually included in the definition of a relation, what it actually says the first normal form is that. The value stored in a column row in a single cell of the relation a single cell of the table has to be an atomic value. This we have seen as the fundamental property of the relational model itself. In the relational model, we said that it is a relation is a set of tuples and a tuple has components.

And each component is a exactly 1 atomic value, we do not allow sets, lists or anything like that in that component, this is a fundamental property of relational database. So the first normal form basically says, restates that and it is included in the definition of a relation. And the other normal

forms like the second normal form, third normal form, Boyce-Codd normal form, these are all defined in terms of a notion called functional dependencies.

So this is a important central kind of concept for discussing normal forms and the various kind of problems that arise due to bad schemes. So a notion called functional dependencies is used and then we will define this normal form. So we will spend considerable amount of time in this module to kind of understand this central important issue of functional dependencies and then how do we you know handle a bunch of functional dependencies etc.

So immediately after this slide we will start looking at functional dependencies. Then there are these other normal forms ok. So as the normal forms are actually listed in there you know strictness in the order of the strictness. So the second normal form is less strict compared to third normal form, third normal form is stricter than second normal form. Boyce-Codd normal form is even stricter than third normal form, what do we mean by stricter is that.

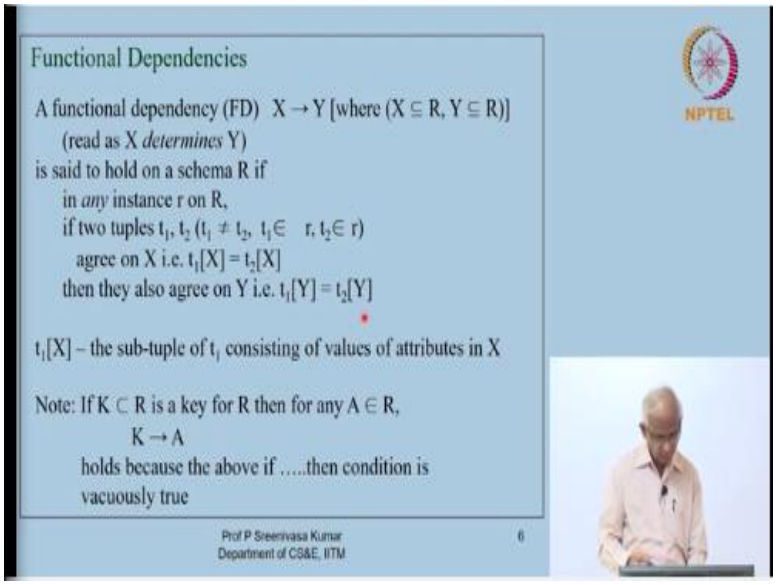
As I told you that these normal forms are basically certain kind of conditions, so if these conditions to be satisfied by the schemes and database schemes. So when you lay down if you say that this particular this is the BCNF can be thought of as a set, set of all database schemes that satisfy these conditions is the class called BCNF. And that class is smaller than the class of 3NF, that is a set of all database schemes that satisfy certain other conditions.

So we will see them see these details as we go along. So that is why we call this Boyce-Codd normal form as a stricter normal form compared to third normal form. And fourth normal form is a stricter normal form compared to Boyce-Codd normal form. And then finally what we have what is called fifth normal form or project join normal form which is defined using. So the fourth normal form is defined using what are called multi valued dependencies, which are a generalization of functional dependencies.

And then project fifth normal form is actually defined using what are called joint dependencies ok. So we will start looking at this notion of dependencies, we will start off with functional dependencies. And hence actually we will spend a lot of time on functional dependencies little

less time on multi valued dependencies. And almost will just give an introduction to project number forms using joint dependencies ok.

(Refer Slide Time: 21:27)



So let us start with this notion of functional dependencies is a very central notion for this theory of dependencies, understood very difficult to understand this actually it is pretty easy. So a functional dependency FD it is called FD functional dependency. And it has a left hand side and right hand side and an arrow. So do not read this as implies, it is to be read as determinants x determines y functional dependency x determines y where x is actually a subset of attributes of R, y is also a subset of attributes of R, R is the scheme, R is the relation scheme that we have.

So relation scheme R if you take 2 subsets x and y then we say that a functional dependency x determines y is set to hold on this particular scheme. Now the definition is that if any instance R on the scheme capital R, if you take 2 tuples t 1, t 2 which are not equal to each other t 1 t 2. And if those tuples agree on x values that means the corresponding values are same in both tuples for all of these attributes in X ok you can also write it like this t 1 X = t 2 X, where what is t 1 X, t 1 X is the sub tuple of t 1 consisting of the values of attributes in X ok, X is some bunch of attributes.

So you take the sub tuple corresponding to these attributes and then of t 1 you will get some tuple right. So if t 1 X is same as t 2 X then it should be the case that on Y also this 2 tuples

agree t 1 and t 2 agree on Y also. So to kind of summarize this, what X functional dependency X determines Y is said to hold on scheme R. If t 1 X = t 2 X implies logically implies t 1 X = t 2 Y, that is all.

If they agree on X attributes they should agree on Y attributes we will give you examples of ok. So here is one interesting note suppose K, some set of attributes of R is a key for the relation R. Then you will notice that the key functionally determines or determines A for any attribute of a scheme R ok. So if you now see apply this definition and then see this I will pause here for you to understand this particular thing.

You try to apply this definition for this K determines A where K is some key actually ok, K determines A. You will notice this to be true, why is it true, on any instance of data on the scheme R what we are saying here is that if 2 tuples agree on values of K, then they will agree on values of A. But then the property of K is that it is a key, right it is a key.

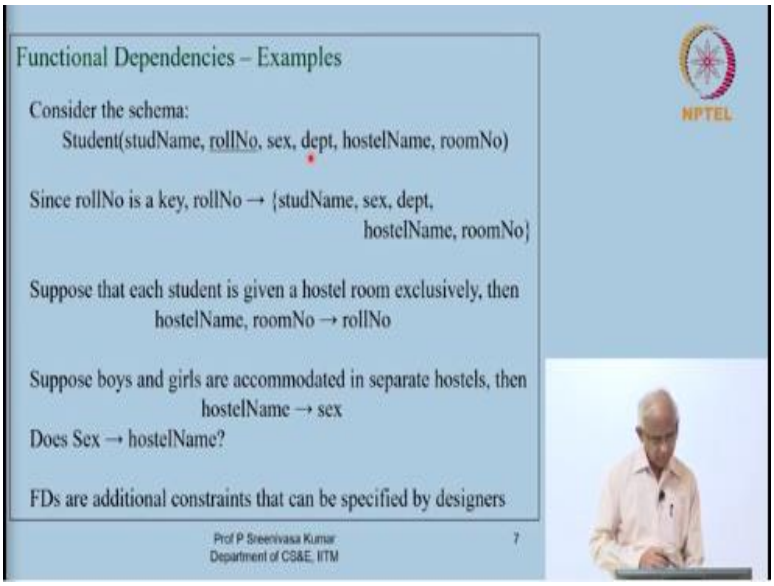
So can 2 tuples in the data instance ever agree on k can 2 tuples have exactly same values for the attributes in K right, they cannot have. Because it is a key no 2 tuples in the instance can have same values for the attributes of K. And so if you see this condition thing that if 2 tuples agree on X, then they should agree on Y. So if you read that as an implication, then that implication is actually vacuously true in this case, the implication is vacuously true.

Because there are no 2 tuples that agree on the attributes of the relation attributes of the key, right. So because the definition of key there are no 2 tuples in the data instance that agree on attributes of K. And so this implication always holds good because the antecedent of this implication which says that if 2 tuples agree on X attributes. Then they should agree the antecedent says that they should agree on X and that antecedent never holds it is always false.

And so the implication is always true, and so you can see that if K is a key K determines A, for any attribute A. So this gives you a simple example of a functional dependency straightaway. If you know the key, then key determines the A and then actually any attribute of the thing. So it

fits the definition, are there any questions here in understanding this definition of the functional dependency, I will give you more examples and we will see if we have any questions.

(Refer Slide Time: 27:59)



Let us take a slightly different scheme here, I am actually in this module Be careful that I am you know changing the schemas as per the need of the you know illustrations that I have to give. And so there are in different slides, so student. Let us look at this particular scheme which says student name, roll number which is a key sex, the gender of the student, department, the department to which the student belongs to.

And then hostel name, let us assume that the students reside in a residential campus. So they have hostels and each hostel has a name, hostel name. And let us assume that room numbers rooms in the hostel or all numbered from you know 001 to some whatever number that is needed. So it is just numeral number some actually you can think of it is a string also room number is a just a number.

In particular I am emphasizing this because it is possible for you to you know make these room number values. In such a way that it kind of actually also has some numeric some characters inside that which will indicate what is the hostel and things like that. Let us assume that such a thing is not done ok. So the domain of room number, you can make it a string and then you know put some characters which will indicate as to what is the hostel in which that room is there.

And then which case then the room number kind of becomes unique, right, it can uniquely identify any room in any hostel of the campus. So let us assume that such a thing is not done just for illustration purpose. So room numbers are simple numbers ok, room numbers are numbers then hostel name, so. Now you can see in this setting let us try and identify some functional dependencies.

So obviously since roll number is a key, roll number determines everything else. So roll number determines student, roll number determines sex, roll number determines department etc. So roll number, now the way we write these functional dependencies we should write them in such a way that the left hand side is a set right hand side is also a set ok set of attributes.

Now if for some reason the attribute is alone then we take the liberty of not putting the set will simply write it like that ok. So now roll number determines all these attributes, this is obviously true. Because in any instance of this particular relation there would not be any 2 tuples which agree on roll number and so they do not have to definition of functional dependencies, the implication that is mentioned in the functional dependencies is vacuously.

And so such a situation of 2 tuples having same roll number will not occur or even if in case by mistake some data tuple comes in inserted into the instance in with the same roll number but some different response. Then you can actually you know this system will catch that because you also stated that roll number is a key. And so such a issue does not arise, anyway good.

So this is one some kind of easy functional dependency that we can identify here. Now let us make an assumption here that the each student is given a hostel room exclusively ok. So everybody is given a room exclusively, so no 2 people are going to be put in a room. Now if this assumption holds good, sometimes it holds good probably not always ok. Let us assume that in our institute that in a second university that assumption holds good or in the domain our assumption holds good.

So if each student is given a hostel room exclusively, then the hostel name, room number together determine roll number right. So if again what you will see here is that since there are no 2 students that are going to have same hostel name and room number. In fact hostel name, room number becomes a key for this particular thing becomes a key, you understand that right. Since we are making this assumption that each student is given a hospital room exclusively.

Hostel name, room number can together identify a unique student**.** And so hostel name, room number together actually becomes a key and since again it is a key. So hostel name, room number determines any attribute in particular roll number. So again this example also is you know following from the principle that the key determines all other attributes, so this became a key but ok.

Now let us look at even a little different kind of an example. Let us make this assumption that boys and girls are accommodated in separate hostels which is usually the case right. So let us assume that boys and girls are accommodated in different hostels. Then you can see that now hostel name determines sex, hostel name is by itself not a key ok, hostel name is not a key, right there are 100s people who are staying in the same hostel.

But now you tell me the hostel name, I will tell you whether the student is a boy or girl right that is what I am saying here. So again this comes from the assumption that the boys and girls are accommodated in separate hostels. So in the domain it is the case like this and so hostel name determines sex, that means if there are 2 rows in this particular relation which have the same hostel name say Krishna or something like that.

Then you will find that those 2 people are boys, assuming that Krishna hostel is reserved for boys. Then now we have multiple hostels for girls right, I think there used to be a case where there is only one hostel for girls you know campus a long time back. Anyway so those names of hostels which accommodate girls for example. So the moment you give any so if 2 rows have 2 values like (()) (36:04) I think is one of the hostels for girls right.

So then you cannot find those 2 tuples having different sex values, one cannot be male one cannot be female. Now the other way around is the other way around true does sex determine hostel. This is obviously not true, why, given that some person is a boy there are multiple hostels in which the boy could be staying. So you cannot determine what hostel the student is staying. Same is the case, it used to be the case that ok, we have only one hostel for girls but now I think that assumption is also not true right. So given sex value, gender value you cannot determine the hostel name.

Because there are multiple students I mean multiple hostels in which boys are accommodated again multiple hostels in which girls are accommodated. So now you see that this is an example of you know a functional dependency that holds between 2 attributes of a relation. So this is kind of giving you some additional information about this student which is actually domain dependent.

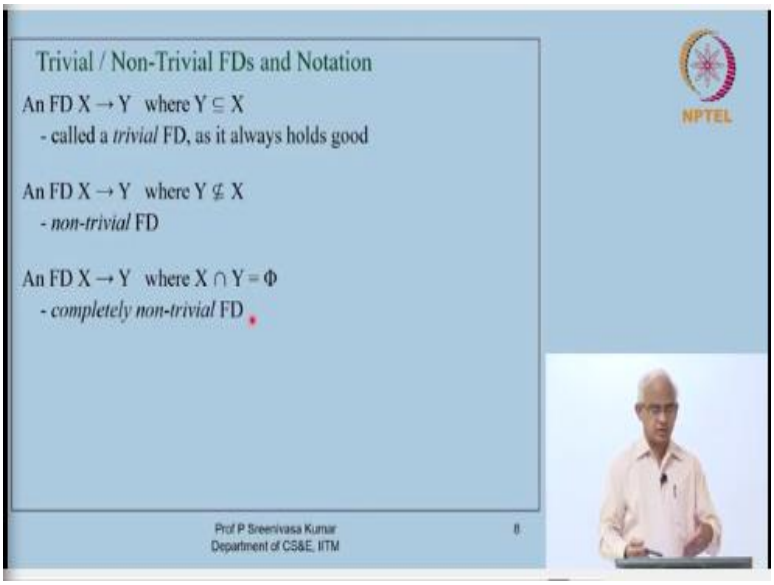
I mean it is in this domain in this you know while modeling this we will be able to figure out that certain assumptions hold good in this model. And so certain functional dependencies hold good on the relation ok. So this actually brings us to the question as to who gives this functional dependencies, how do we find this functional dependencies ok. So these functional dependencies have to be given or have to be figured out by the designers of scheme who are studying the domain.

And then finding out what are the assumptions that are valid in the domain, what assumptions are not valid in the domain or knowledge in the domain. So, based on that we will have to come up with this functional dependencies ok. So you cannot just like a key can never be you know determined by just looking at the instance of a relation right. By looking at the instance data that is there in the relation you cannot determine what is a key right.

So in a similar way here by just looking at data you may not be able to for sure determine what are functional dependencies. Of course you may do some analysis and then figure out then saying that most of the data seems to be you know giving rise to this kind of a thing when is this really true in the domain, you may pose it to the designers saying that. But it is ultimately the

designers who understand the domain have to decide as to what are these functional dependencies that hold between the ok.

(Refer Slide Time: 39:40)



So now let me so I encourage you to in your schema design that you are coming up, examine the assumptions that you have made or what are the you know appropriate assumptions for you domain, what is the domain knowledge. And based on that try and figure out what are the functional dependencies that hold between attributes in your scheme and then make a note of them.

Now a few things that we have to you know some corner cases we have to fix is that. If a functional dependency X determines Y is there then there are various cases that you know might arise what if Y is a subset of X itself, is that a functional dependency well. It is a functional dependency but it always holds good. If y is a subset of X if the right hand side is a subset of X then if 2 tuples agree on X they will always obviously agree on Y also.

Because Y is a subset of X, so it always holds good, so that is such a kind of a dependencies what is called a trivial functional dependency. Because it always holds good in all schemes in all data ok. Now if Y is not a subset of X then we will call such a functional dependencies and non trivial functional dependency. So whether it holds or not it depends on the underlying domain, so it is a non trivial functional dependency.

So this what this says allows you is to have if Y is not a subset of X, Y might have some you know attributes in common with X, Y index may have some common attributes ok, in this case. So if Y index do not have any common attributes at all then we call that as a completely non trivial functional dependency. And these non trivial functional dependencies whether they hold or not depends on the domain actually.

So maybe we will stop here for today and I will give you further examples of functional dependencies and how functional dependencies are going to be used in defining the normal forms for in the next lecture.