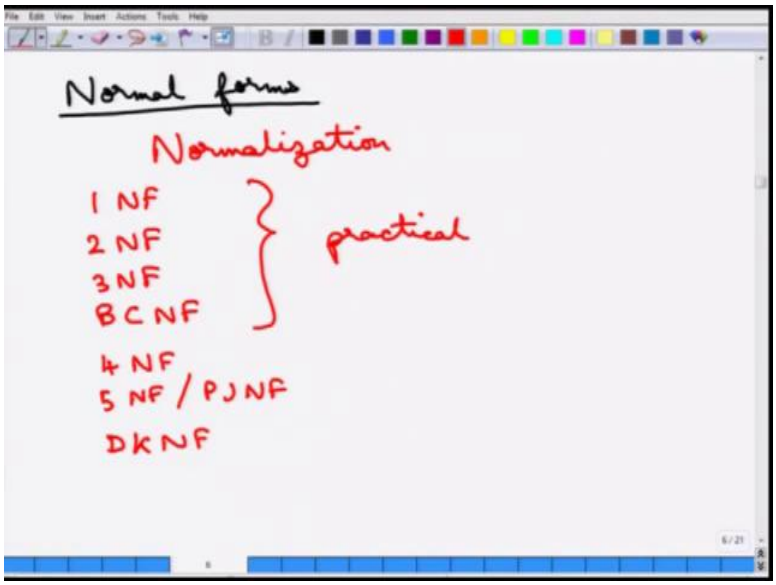
Fundamentals of Database Systems
Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 13
Normalization Theory: 1NF and 2NF

We will start off with these normal forms.

(Refer Slide Time: 00:12)



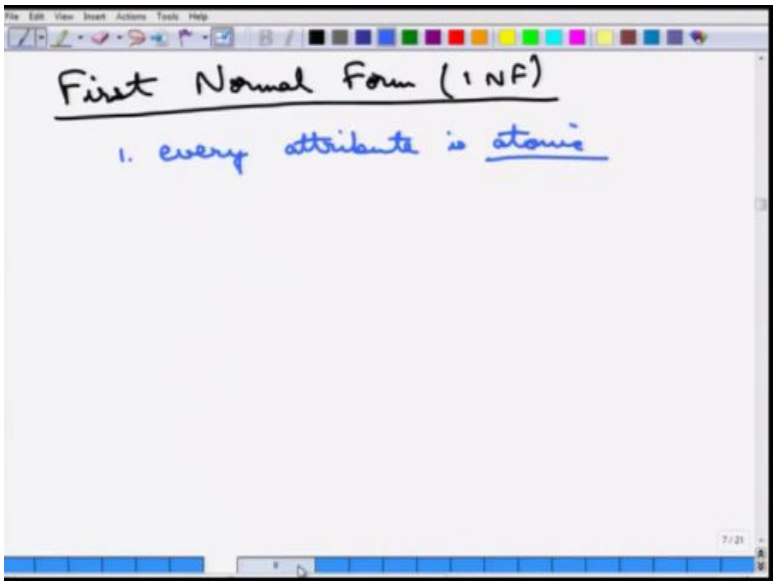
So, the process of given relation, given a schema of a database so; that means, multiple relations etcetera. The process of actually designing what the schemas are and in general taking all the attributes of all the entities of the database together and breaking it up into smaller relations is called normalization. So, that process is called normalization and this whole theory of how to normalize etcetera is called the normalization theory and keys and f d's as we told earlier determines which normal form relation is in.

So, there are different normal forms, there is this 1 NF or first Normal Form then 2 NF or second normal form then this is 3 NF third normal form, there is something called a BCNF Boyce-Codd Normal Form. Up to this is something which is mostly practical and database designers tries to achieve up to this, then there are some higher order normal forms, which are also useful in some cases.

But, mostly they are not attempted which are called fourth normal form, fifth normal form which is also sometimes called project join normal form. And finally, there is a

domain key normal form which is the theoretical maximum that one can reach. So, we will go over each of these next.

(Refer Slide Time: 01:40)

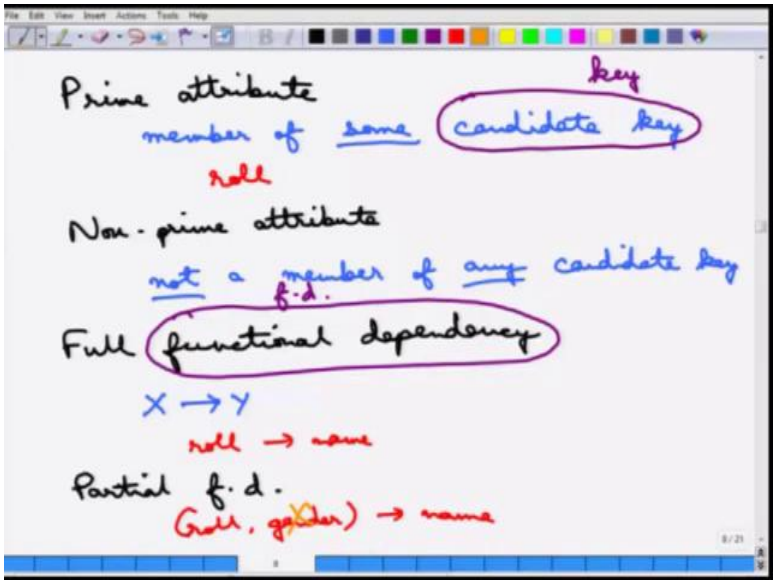


So, we start off with the first normal form or the 1 NF, so a relation is in 1 NF if that following condition holds is that every attribute is atomic. As soon as this is achieved, then the relation is said to be in the first normal form and this seems to be very trivial. But, in some cases we can see that a relation may not be in first normal form, for example we saw that the example of a non atomic attribute earlier which is a name, a name can be most cases can be broken down into a first name and the last name.

So, if a relation contains name, it may not be considered to be in 1 NF, now the point of 1 NF is that there is no way to actually argue about it. Because, one can say that together this make a name and whether it makes sense to break it up for the particular application, it is not clear. So, generally when we study normalization theory, we will just assume that relations to be in normal form. So, again where... So, we will in the following couple of minutes etcetera, we will see examples of relations there we will use the name and we will assume that the name is an atomic attribute.

So, we will just assume that things are in first normal form unless so much said. So, this is one thing to remember, that everything is in first normal form to start with. If it is not in first normal form, you can always break it down into this first name and last name etcetera. So, it is not a very hard thing to make a relation into first normal form.

(Refer Slide Time: 03:33)



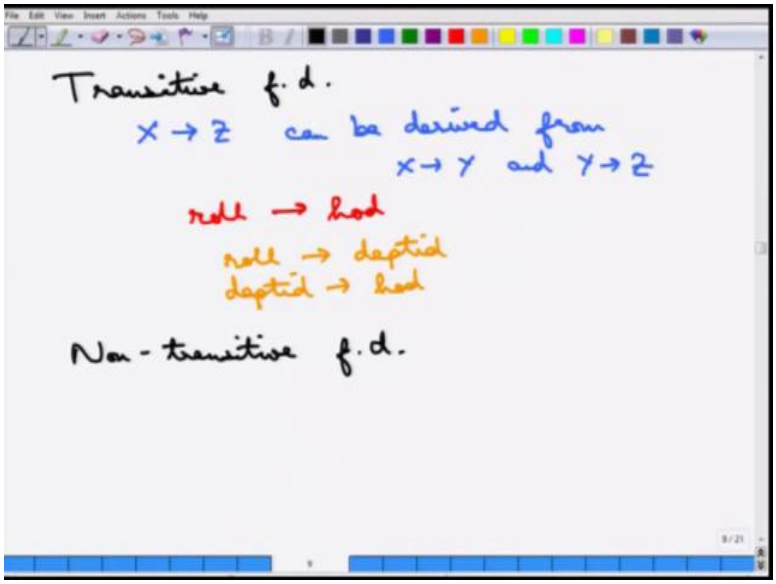
Next, we move on to something that is required for the other normal forms, first is called a prime attribute. So, this is a prime attribute is something it show up an attribute that is a member of some candidate key is called a prime attribute, this is important it is some candidate key. So, an example may be the roll number of a student, so roll is a member of some candidate key. So, it is a prime attribute and analogously one can define the non prime attribute, so any attribute that is not prime is a non prime attribute.

So, the equivalent definition it is not a member of any candidate key, so just to ensure that this is good, not a member of any candidate key. So, let me also use this notation, so whenever we talk about candidate key from henceforth, I will just use the word key to determine there it is. So, if I do not mention the context etcetera, if I just mention the word key then it actually means a candidate key. So, prime attribute and non prime attribute should be easy to understand, then the next definition that we will require is something called the full functional dependency.

So, a full functional dependency will be defined, but again we will use instead of a functional dependency I may write down as f d, which is the same as same functional dependency. A full functional dependency is something if the functional dependency does not hold when any attribute from the X is removed. So, if X goes to Y if this is the dependency, this f d is called full functional dependency, if X cannot be reduced any further and it is a partial dependency otherwise.

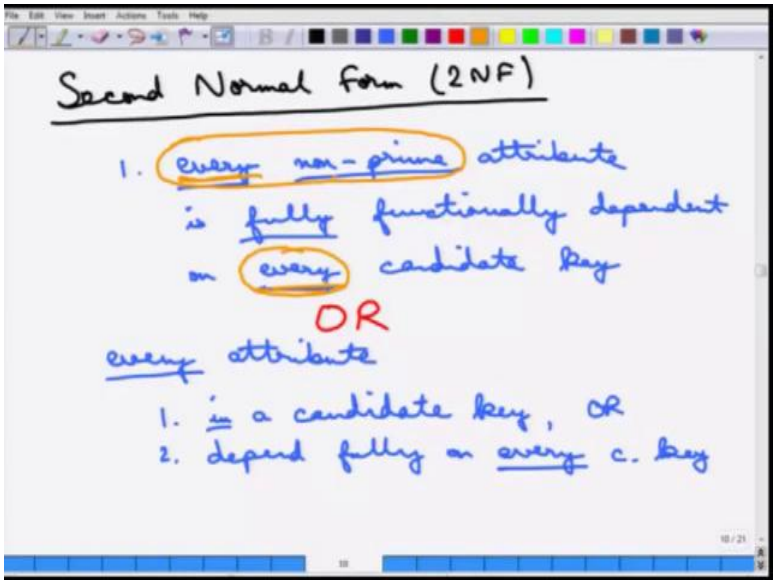
So, an example of full functional dependency may be roll determines name I mean of course, if the left side is only one then nothing can be done about it, it is always full. But, we will see examples of some more interesting things and it is a partial functional dependency, if the left side is reduced and it is still a functional dependency and here is one example is that if you say roll comma gender together determines name, then of course, this is a partial functional dependency. Because, one can get rid of the gender and the still functional dependency holds, so that is a partial functional dependency.

(Refer Slide Time: 06:06)



Then, we will talk about something called a transitive functional dependency. So, suppose X determines Z if this can be derived from two functional dependencies can be derived from X goes to Y and Y determines Z. An example of this is suppose the roll number of a student to the head of the department of that student. Now, this is a transitive functional dependency, because the roll number essentially can say what is the department of that student and the department id can say, who is the head of the department of that department, so this is a transitive functional dependency. And of course, it is not transitive otherwise, so if f d X to Z cannot be broken down X to Y and Y to Z then it is called a non transitive functional dependency.

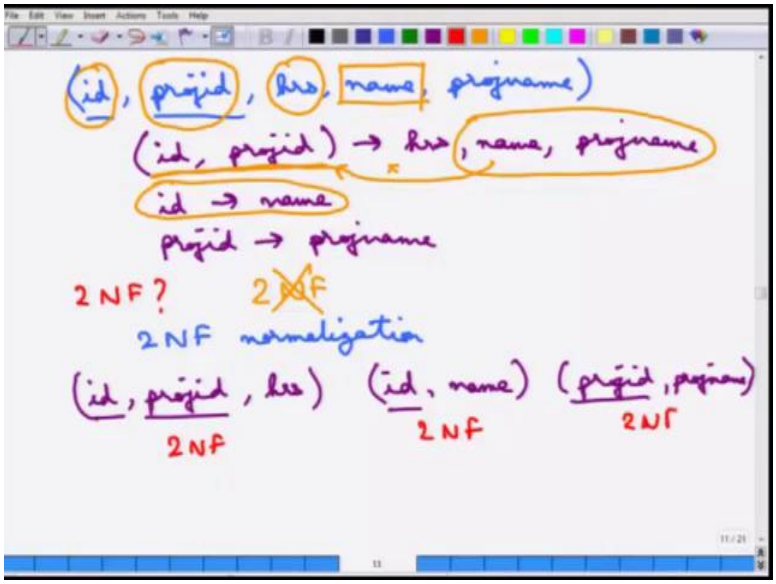
(Refer Slide Time: 07:15)



So, having this definition handy with us we will next define what is called a second Normal Form or 2 NF. So, a relation is said to be in second normal form, if every non prime attribute is fully functionally dependent on every candidate key then there is an alternative definition or every attribute must be in a candidate key or depend fully on every candidate key. So, these two are equivalent definitions as one can understand, let us take the second definition.

The second definition says that every attribute is in a candidate key; that means, it is a prime attribute if not; that means, it is a non prime attribute, then it is fully functionally dependent on every candidate key. So, this is a important part is that this is every candidate key and also one while testing this has to be every non prime attribute, these two are important things this is every here and every there as well. So, second normal form what does it mean, so here is an example.

(Refer Slide Time: 09:18)



Suppose, we consider the following schema, id, project id, hours, name, project name. So, this is the schema that we will consider and we will try to argue whether this is in second normal form or not. Now, one thing one must remember is that to determine whether a relation is in the second normal form or not, the functional dependencies must be specified, so this is part of the question. So, one cannot answer whether what type of relation it is in, whether in which normal form it is unless the functional determines functional dependencies are given.

So, there are two ways of giving the functional dependencies, first is generally what is being done is the primary or the candidate keys are underlined. So, this essentially means that this translates to the following functional dependency that id, project id together is the candidate key. So, this determines everything else, so this determines for example, hours then this determines name and project name, etcetera, but in addition there may be other functional dependencies that are specified for example, one may say that id determines name and project id determines project name.

So, these are the three functional dependencies that is given and now, one can see that essentially this is redundant, because if id determines name of course, id, project id determines the name as well. So, the question is this relation in 2 NF that is the question that we will try to understand, how do we go about answering that question. So, we test each attribute and see whether it satisfies the condition of the 2 NF.

So, first of all let us check up this attribute, now this is in a candidate key, so this is not a non prime attribute. So, nothing to be done it is already satisfying project id, project id is again in a candidate key, so that is fine. Now, let us say take hours, so hours does it depend fully on every candidate key. So, what is the only candidate key? The only candidate key here is id and project id. So, hours dependent completely on id and project id that means, that neither id by itself nor project id by itself determines hours.

Now, let us consider name, now what happens with name is, name is a non prime attribute and it is not determine fully by id and project id, because there is this partial id to name. So; that means, name is not fully determined so; that means, name fails the 2 NF test and the answer is that this is the relationship is not in 2 NF. Now, one may argue that, that is fine what how is this 2 NF useful, so 2 NF essentially says that, that is some problem way this scheme determined.

What is the problem? Now, you see that name is determined only by id, so which means that the id and project id together the key is redundant, it does not require that. So, another way of saying is that suppose the name of a person is changed then... So, corresponding to the id the name is changed now the person is working in many different projects, supposedly and all of those tuples needs to be modified. So, that is the problem and why are those all of those tuples need to be modified.

Because, just by changing the name one only has to argue about the id or vice versa what the project id is something that is also part of the candidate key of this relation. So, the project id is in a sense redundant when we are arguing about the name. And why is it redundant? Because, name is not fully dependent on id comma project id that is why 2 NF is useful and that is why it makes sense to determine whether a relationship is in 2 NF or not.

Now, suppose we want to make this relationship into to 2 NF, so that this process is called a 2 NF normalization, so we understand that above schema is not in 2 NF. So, how do we 2 NF normalize it, the way to do it 2 NF normalization is that we break it up into the following tables. So, the first table contains id, project id and hours, this is the first table, the second table is, so this key for the thing is id and project id.

So, second one is id and name and the third one is project id and project name, one can intuitively see that this is a better design, the reason is what we have been arguing about.

So, name depends only on id, so why not break it up into a separate table. Similarly, project name depends only on project id, so why not break it up into a separate table and now what happens is that if we go about to the previous example, if one changes the name there is only one place that this whole information needs to be changed in the database, the tuple corresponding to that particular id to name.

Similarly, if the name of a project is changed there is only one place that it needs to be touched, the other tables are not touched. So, there is no modification anomaly etcetera, formally we can also test if this is a better design than the other one. Because, now we can test whether each of these three relations are in 2 NF and if you go about testing it you will find that all of this r actually in 2 NF. So, this is in 2 NF, this is also in 2 NF and this is also in 2 NF. So, we have argued both informally and formally that the design where you break this up into these three tables is a better design.