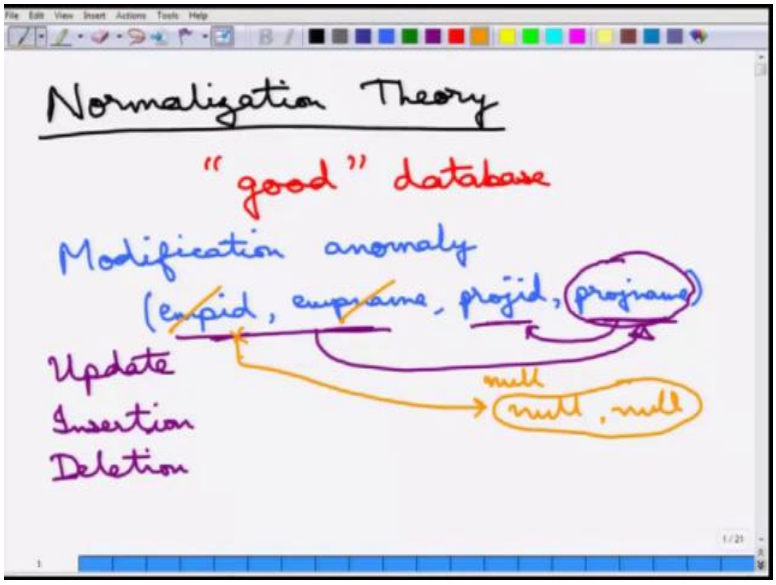
Fundamentals of Database Systems
Prof. Arnab Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 12
Normalization Theory: Motivation

Today, we will start on Normalization Theory.

(Refer Slide Time: 00:13)



So, the essentially the main question that we will try to answer is, how to design a good database, now there can be different meanings of, what a good database is and the answer can be given informally or formally. So, informally we can say the database is good when each relation or schema represents a particular entity for example; student is one kind of entity, so there is a relation for student faculty member is another kind of entity.

So, there is another entity for faculty member there is a relation for course etcetera, etcetera. Then, we can also say that there are no spurious tuple; that means there is no tuple which has got values that does not mean anything, so there is very little or almost no redundancy. Then, we can say that the null values as far as possible are minimized in the entire database schema.

So, null values actually represents missing or unknown values and it does not make sense, so the lesser number of null values it is the better it is and then, there should not be

any modification anomalies. So, this is one important part of it, so this is, so we will go over this in a little bit more detail the modification anomaly, but this is the informal way of answering, what it is.

So, what we will do is we will try to tackle this modification anomaly part first. So, this is modification anomaly and, let me explain what it means? So, for example, let us just start with the particular schema, let us say the schema is employee id, then there is an employee name, then there is a project id and a project name. So, essentially the idea of this schema is that there is an employee with a particular name, who works in a particular project id with a name of project name.

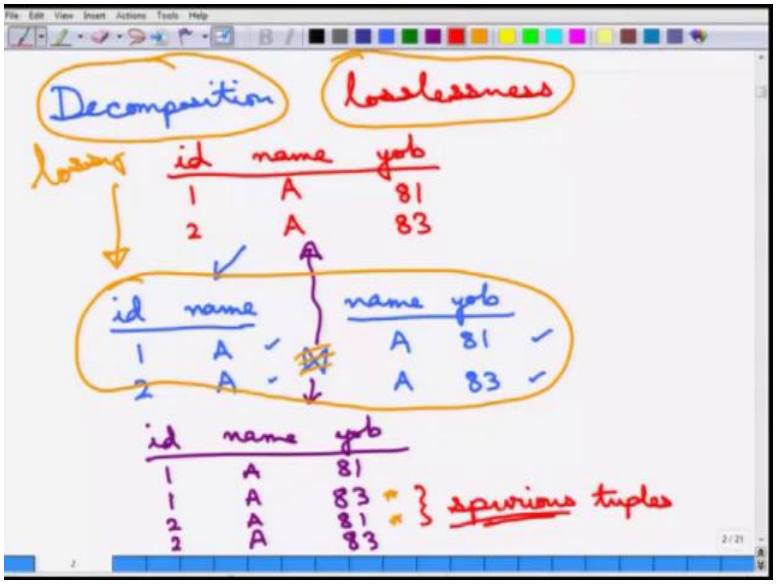
Now, is this a good schema for an employee project; it is not because of the following reason, so the first is the update anomaly. So, what happens is suppose the project name of a particular project is changed; that means, every employee id, so this is a tuple. So, every employee id, which was in the project will go and change it is corresponding projects name attribute, which is a lot of changes.

So, although only one piece of information the project name is changed, there are lots of changes in the database, this is not a good way of doing it. Then, the next one is the insertion anomaly, so the first one was update anomaly, this one is an insertion anomaly. So, now, as soon as an employee is inserted into the relation, the employee must have a corresponding project; otherwise this column becomes null and as we said nulls are not really preferred.

The other way around is also true, whenever a project is introduced, it must have some corresponding employees; otherwise this column becomes null. And, now it may not be that whenever a project is opened there are employees already assigned or it may not be that when an employee comes to an organization he or she is already assigned to some project, so these are some insertion anomalies that will happen. And then, of course, there is the third one, which is a deletion anomaly.

Now, suppose a particular project is being tried to be deleted, now as soon as a project is deleted, this for the employee which were in the project these two corresponding fields become null and it may happen that the deletion algorithm will try to get rid of this particular tuple. So, that even the employee id and employee name may be deleted. So, that, so these are the problems with the modification anomaly.

(Refer Slide Time: 04:14)



Then, the next kind of thing, that we will handle is the decomposition, when a schema is decomposed or a relation is decomposed, what are the issues that one can face. So, whenever there is decomposition, the important property in the decomposition is something called the losselessness. So, I will explain what is this with an example, suppose here is one particular schema, so what does this mean is that there is a particular employee with A id, suppose id 1 whose name is A and who had born in let us say 81.

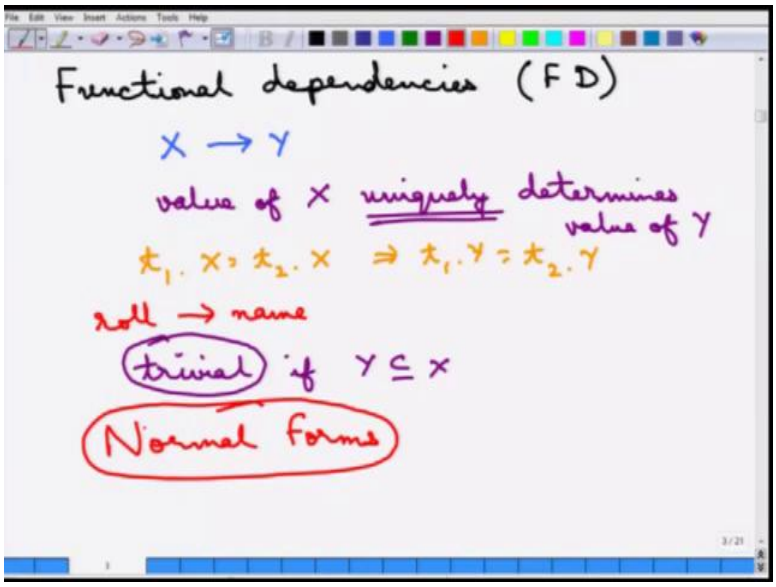
And then, there is another person B, which 2 with the same name apparently and we has born in year 83, now what may happen is that the name clash. Now, suppose we decompose this relation into two relations, which is id name and name year of birth. So, what will happen is that when we decompose, this is what the decomposition will say note that up to this point it seems that this decomposition is correct, because the information are all correct.

So, 1 A is actually an employee, so is 2 A and for, then employee A, whose name is A there is a year of birth 81 as well as year of birth 83. But, the problem is the decomposition must satisfy that when they are joined they should give back the original table; however, the join produced this wrong information in this case. So, the join produces the following thing this is again id name and dob, but it produces, even if you do a natural join I am assuming a natural join with name.

So, it produces 1 A 81 it also produces 1 A 83, it similarly produces 2 A 81 and 2 A 83. Now, you can see that there are two spurious tuples these are spurious tuples 1 A 83 are 2 A 81 these are spurious tuples these have resulted because the decomposition into these tuples were not done correctly and, so the join is not correct. So, this decomposition is a lossy decomposition, because this decomposition looses certain information.

So, it essentially violates this very important property of losselessness, so we would want the database; such that the decomposition are lossless. So, these two things are called spurious tuples we must try for a design, where there are no spurious tuples this is important this is called spurious tuples. So, the normalization theory is actually tries to say, how to design a good database in a formal manner. So, it tries to handle this problem, so modification anomaly the lossless decomposition etcetera.

(Refer Slide Time: 07:18)



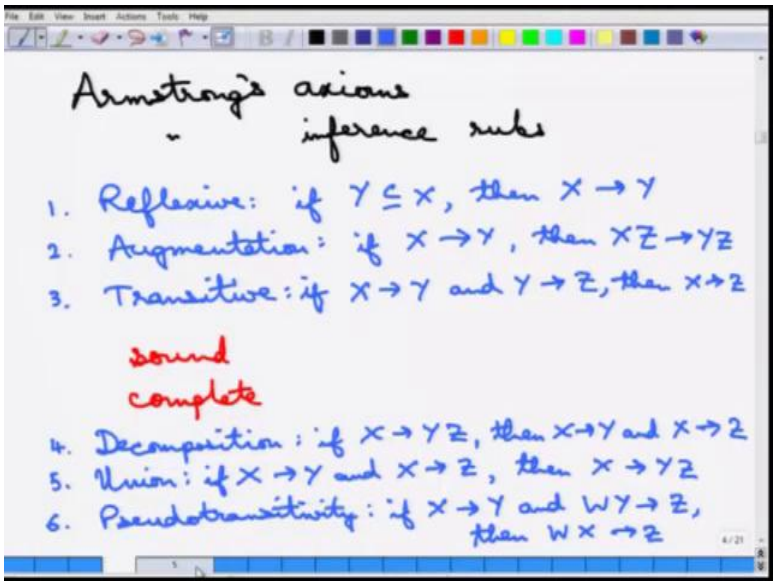
For that, before we go into the normalization theory we require the concept of something called functional dependencies. So, we will define functional dependencies, that will be used in all the definitions of normalization or we will sometimes use the abbreviation FD. So, functional dependencies are constraints that can be derived from the relation itself. So, we say X functionally determines Y by the way X and Y are sets of attributes in a particular relation, X functionally determines Y this is the notation value of X uniquely determines the value of Y.

And you can see, where the definition is coming from this is essentially X is, so suppose X is the candidate key or the super key of the relation, then the unique value of X determines the entire tuple, so Y can be any other attribute. So, here is, what it means once more the functional decomposition is that, if we say X functionally determines Y; that means, if we know the value of X the value of Y is fixed and it is an unique value.

So, again the other formal way of saying is that if for two tuples t 1 dot X is equal to t 2 dot X, because this the X value uniquely determines this implies the Y value of t 1 should be equal to the Y value of t 2 note, that it is of course, not the other way around. An example in a student database may be roll number, so it functionally determines the name. So, if one knows the roll number of a student the name is uniquely determined that is the point.

And a functional F D is called trivial it is called trivial, if Y is a subset of X, because if X is a unique name of course, Y is unique, so it is a subset this is called a trivial functional dependency this is an important part. And as we said a candidate key functionally determines every other attributes of the relation, so it is the thing. So, the functional dependencies and the keys they together define, what are called normal forms of the database, so this normal forms are, what we will be going over in more detail next, so these are normal forms, so this functional dependency and the keys depends the functional dependencies.

(Refer Slide Time: 09:40)



So, before that we go over certain axioms about the functional dependency these are called Armstrong's axioms. So, note that these are axioms, so this cannot be proved etcetera these are called sometimes called Armstrong axioms or Armstrong's inference rules. And there are three such axioms the first one is the reflexive, it essentially says that the definition of the trivial thing if Y is a subset of X, then X functionally determines Y the next one is called augmentation.

So, if X functionally determines Y, then X Z functionally determines Y Z. So, essentially X is augmented with another set of attribute Z if the left side is augmented, then the right side can be augmented and this is not very difficult to understand, why and the third one is called transitive. So, if X if X determines Y and Y determines Z, then X determines Z again this is not very hard to understand. So, if we know the value of X the value of Y is unique.

And if we know that unique value of Y again the value of Z is unique, so you can simply say that knowing X will determine the unique value of Z. Now, these rules are sound and complete in the sense that any other rule that can be derived from it will also hold and complete meaning no other rule can be outside this. So, every other rule can be derived from all of this from one or more applications, so each one or more of this.

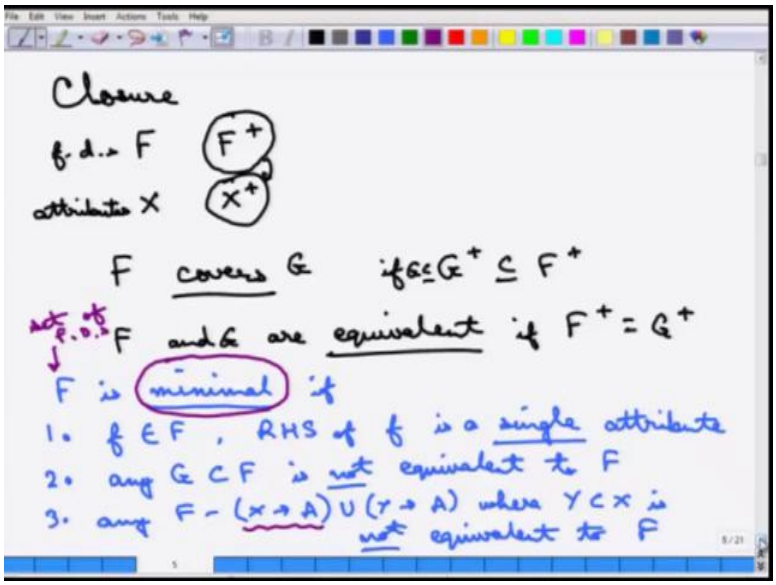
There are some other rules, which are useful, but are not actually needed, because they can be inferred from the above field, but nevertheless are very useful, first one is called decomposition. So, if X determines Y Z by the way Y Z means it is a attribute set, which is formed of Y and Z, then X determines Y and X determines Z. So, if X determines the values of Y and Z both and of course, X determines Y and of course X determines Z individually.

Fifth one is called union, which is if X determines Y and X determines Z, then X determines Y Z. Again this is easier to understand knowing X if one knows the unique value of Y and 1 knows the unique value of Z, then knowing X 1 knows the unique value of combination of Y and Z it is essentially a combination meaning it is just a concatenation of the attributes.

And the last one is pseudo transitivity if X determines Y and W Y determines Z, then one can say W X determines Z as well. Again this is not very hard to determine, because Y there is a unique value of Y for each unique value of X and, so if Y is replaced by X

the same functional dependency goes through. So, that is the idea about the functional dependency and their rules, such that Armstrong axioms etcetera.

(Refer Slide Time: 13:43)



Then, using this one can define a closure of a set functional dependency. So, if F is a set of functional dependency the F plus is the closure of it, so given if all the functional dependency rules that can be derived from F form F plus, so F plus is the closure of it. So, similarly closure of a set of attributes of X with respect to this, so this is a functional dependency's and these are attributes if X is the set of attributes with respect to F, then the set X plus is the closure if X is the set of attributes that is derived from F.

Then, X plus is the set of attributes that is derived from the closure of F, which is F plus again this assuming this. Then, there is another definition, which is called covers, for F covers G, so a set of functional dependency F covers a set of functional dependency G if everything in G can be inferred from F. So, which essentially means G plus is a subset of F plus, so if everything in G can be inferred from F, so what is the everything can be inferred from F, which is F plus and then G is of course, part of G plus, so if G is part of F plus as well, so G can be determined from this.

And if and G are equivalent F and G are equivalent if the closure of them are the same. So; that means, knowing F is same as knowing G, because the closure, so all the functional dependencies, that can be derived from F is the same as all the functional dependencies derived from G and it is exactly the same it is not a subset etcetera. So,

knowing F is essentially knowing G, so these are equivalent it can be also said that F and G are equivalent if F covers G and G covers F.

Using this there is a definition called F is minimal this is called F is minimal, so there is a minimal set of things if this is the definition is that every, so every F, in F is a single attribute in this thing, so for F, which is a F D in F the RHS of F. So, the right hand side of F consists of a single attribute is a single attribute, so that is a very important condition this is minimal, because we do not have unnecessary things in that right side.

Then, any G, which is proper subset of F is not equivalent to F; that is why this is even it must minimal meaning you cannot get rid of some functional dependency and get back the same thing this is F. And the last one is any F minus X union, where Y is a proper subset of X is not equivalent to F. So, what does the third rule means, so this is 1 2 3 the third rule means, that let us consider F and let us check out one functional dependency of the form X determines A.

Now, let us take that out and add another rule which is Y determines A, where Y is the subset of X. So, essentially X is being that this rule X determines A is being reduced to Y determines A. Then, it is not equivalent any further, so the left sides are also in some sense minimal. So, because reducing something from the left side of a rule is does not produce the same set of functional dependency.

So, this is the definition of when a set of functional dependencies is called minimal by the way just remember that this is a set of functional dependencies not of course, a single functional dependency. Now, using all of these, so we can define the normal forms and just one important thing is that every set of functional dependencies has at least one minimal set of functional dependency. So, next we will go over something called normal forms.