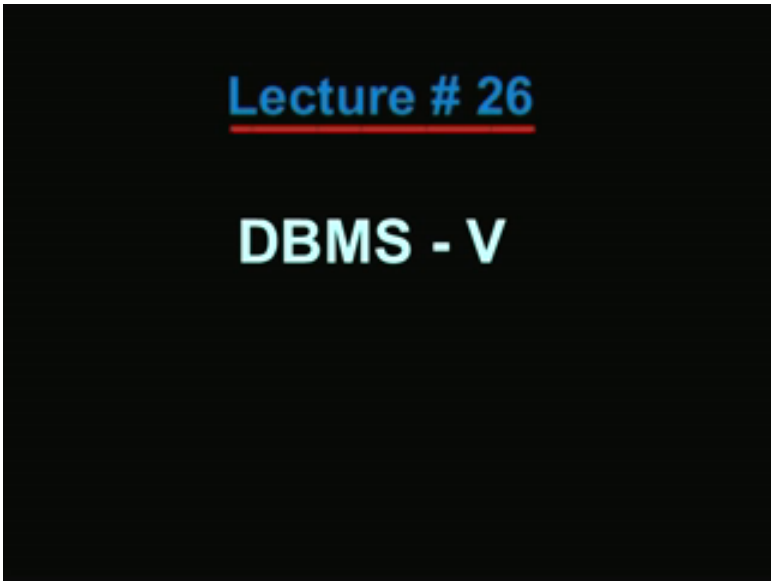
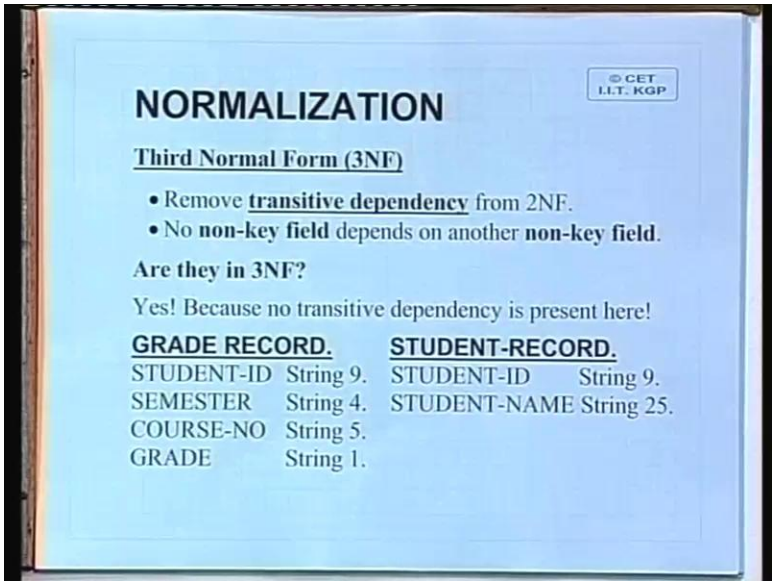
Management Information System
Prof. B. Mahanty
Department of Industrial Engineering & Management
Indian Institute of Technology, Kharagpur

Lecture - 26
DBMS – V

(Refer Slide Time: 00:51)



(Refer Slide Time: 01:09)



In our last class, we were discussing about the normalization of the DBMS. That is called, we said that whenever we are designing a database structure, essentially there are two approaches: the first approach is entity relationship diagramming and the second approach is through normalization. What I said is that normalization again can be understood from two aspects. What are these two aspects? First, a layman's view and second, an algebraic view.

So, we are in the first one - that is the layman's view. What I said was that to begin with, we have the first normal form. First normal form is nothing but when you remove the repeating groups, basically no arrays exist there and we have a set of attributes. Then we have the first normal form. Then we remove from the first normal form the non-full dependencies or the partial dependencies. I have given you some examples; and then we have the second normal form.

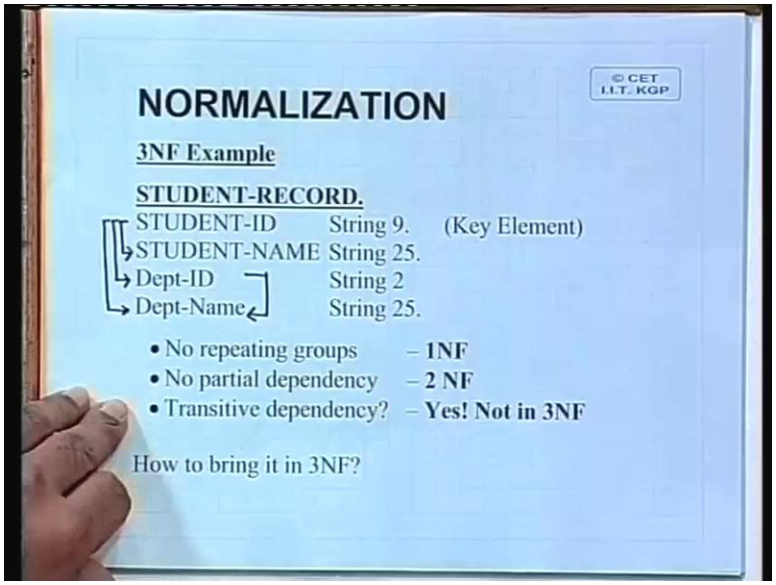
So if you recall then we took an example of students grade records and we found that if we have student name here, since the student name would depend only on student id and other things - that is grade; it depends on student id, semester as well as course number. So, there is non-full dependency. This is what we have seen in our previous lecture. So, that is why we had to have two records. That is grade record and student record and we say that this is in second normal form.

Now, is it in third normal form? That is the question today. To obtain third normal forms from second normal form, we need to remove what is known as the transitive dependency. Now, what is the transitive dependency? That is, no non-key field should depend on another non-key field. Say for example, if we have in the second student record the student id and the student name and suppose certain other attributes, we see that student id is the key, primary key here, because this uniquely determines all the other attributes.

The student name is dependent on student id; student address is dependent on student id, right? All these would be true but is there any dependence between the...I mean when we talk about dependence, you must always understand that it must identify an entity. See, there is...we must differentiate between the original dependency versus the derived dependency. We will discuss about them mathematically later. At this point, we must understand...because it is easy to get confused that a student name determines student address. So, there is transitive dependency.

So, please do not think in that line; always think of the key identifiers. For example, student id is a student identifier. So anything that is an attribute of student should be naturally dependent on student id. So, there is no transitive dependency there; but suppose you have something like hostel number, hall number. The hall number determines another entity. So, you should be careful whether there is a transitive dependency or not.

(Refer Slide Time: 05:16)

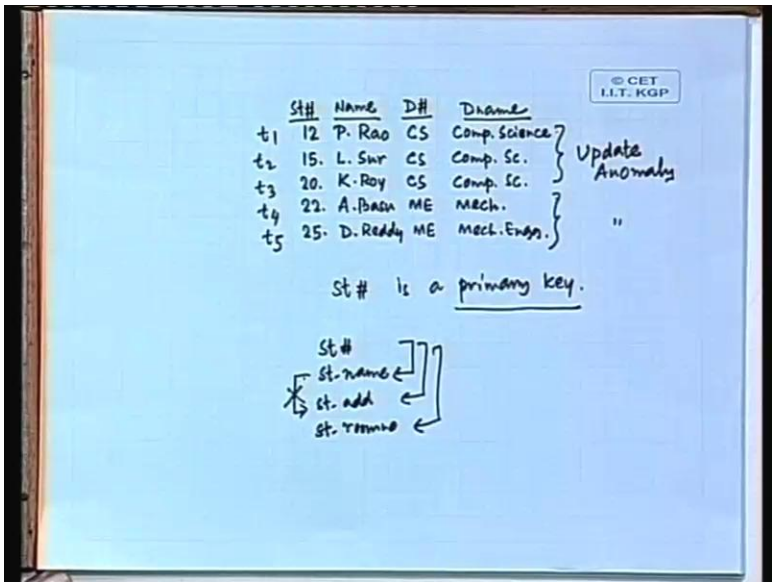


So that one more example you can see here for 3NF. You see, think of a student record where we have the student id. Now, student id, student name, department id, department name. Now tell me, is the student id a primary key in this case?

Some of you are saying there are two primary keys. If I make department id as the primary key alright then can there be two tuples where department name is same, department id is same, but student id is different? Possible. In a department, there are many students, alright? For all those students, department id will be same. So, to say that if student id, if the department id agrees then other attributes cannot disagree, will be wrong, will be wrong. So, department id cannot uniquely identify, cannot uniquely identify the records of this particular table.

So, department id is not a primary key; it is not; whereas student id is a primary key because see, if the student id is same, can there be two different department id? In other words, can a given student be part of two departments? No, right? So, if that is so, if a student is not part of two departments, that is not allowed. So we can say that student id is a unique key. If the student id is same, department name cannot be different. So, student id uniquely determines all of them. See, there is no difficulty here. The difficulty on the other hand is definitely there to some extent because the department name gets repeated.

(Refer Slide Time: 07:32)



Let us take some tuples. Say 12. You see, I have put some tuples. So, student number, student name, department number and the department name. Now, if you look at these 5 tuples, we have the student numbers are different; but if suppose the student number repeats, all the other fields has to repeat. So, student number is a primary key. So, there is definite primary key. So, that is not a problem. It is not because that we do not have primary key that is the problem. But, if you see that for many students for which department is same...for example, the first three students' department is same - that is CS; but when it comes to department name, we have to repeat the department name and when we are repeating we are writing there is a possibility, there is a chance that we may enter different things, different times. Can you see here?

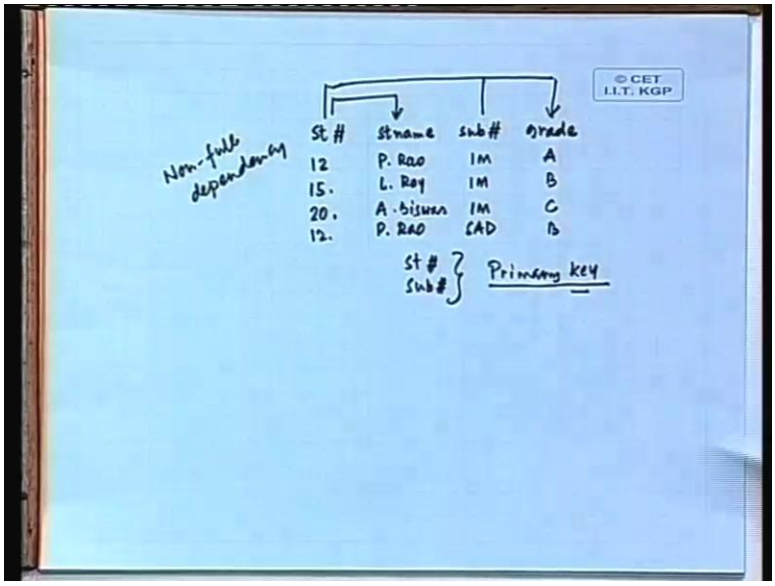
So this is where there is a chance of what is known as update anomaly. See, this is the problem. This is the problem of transitive dependency; and how to identify transitive dependency? You see, it is basically a student record. Student is an entity, department is another entity and department name is a attribute of department rather than student. So, this is how you must try to find out what is known as transitive dependency.

If on the other hand you want to find out transitive dependency within the entities of the...within the attributes of the same entity, it is slightly difficult situation. I mean you will...you will be wrong actually, alright? Say for example, if you have student name...student hash, student name, student address, then room number, etcetera, etcetera. Now you can see that these uniquely identifies each of these and they are basically nothing but attributes of student.

So, if you now try to think that...does a student address or student name determines student address? Probably yes, probably yes, because if the student names are not like same name, repeated. Suppose if they have, all have unique names, all have unique names, there is a possibility of finding this kind of transitive dependency alright but you must remember that this is not an original transitive dependency because they are between the attributes of the same entity. So, be very careful about finding transitive dependency between the attributes of the same entity. You must avoid them. If you do not, then there will be chance of getting wrong answers.

Now, this is the 3 NF example we are giving. You can see that there are no repeating groups; so it was, it was definitely in first normal form; no partial dependency - second normal form; transitive dependency – yes. There is a transitive dependency; student id although uniquely determines all of them. So, all these are true; but this is also true. Since this is also true, we have what is known as transitive dependency and therefore this is not in third normal form. Non-full dependency, right? So, let me show you.

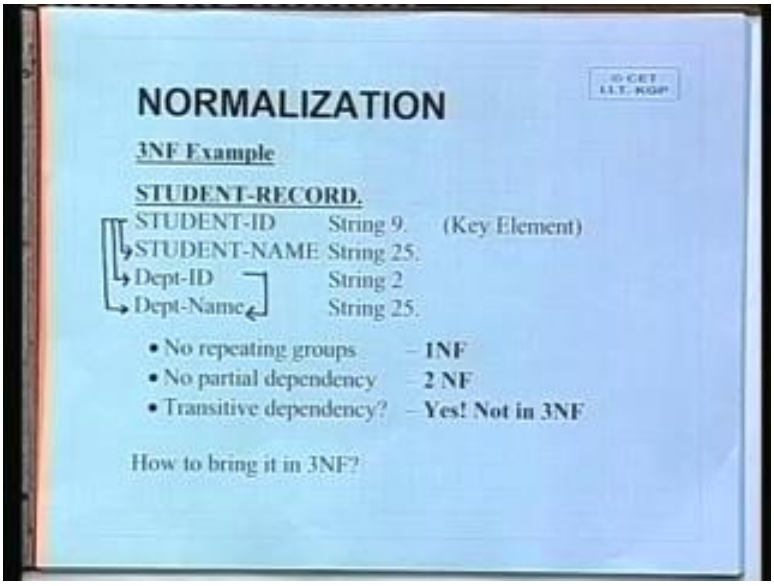
(Refer Slide Time: 13:38)



Suppose you have student hash, student name, subject hash and grade, right? So, you may have something like...you see what is happening here? This is true. The student name is uniquely identified by student roll number itself; but grade is identified from a consideration of both student number and subject number. So, you have to have a primary key this way. Student number and subject number - together will be a primary key. So, this is primary key. But what is happening? Unfortunately, the student name is dependent on one part of the primary key only. So, this is known as non-full dependency.

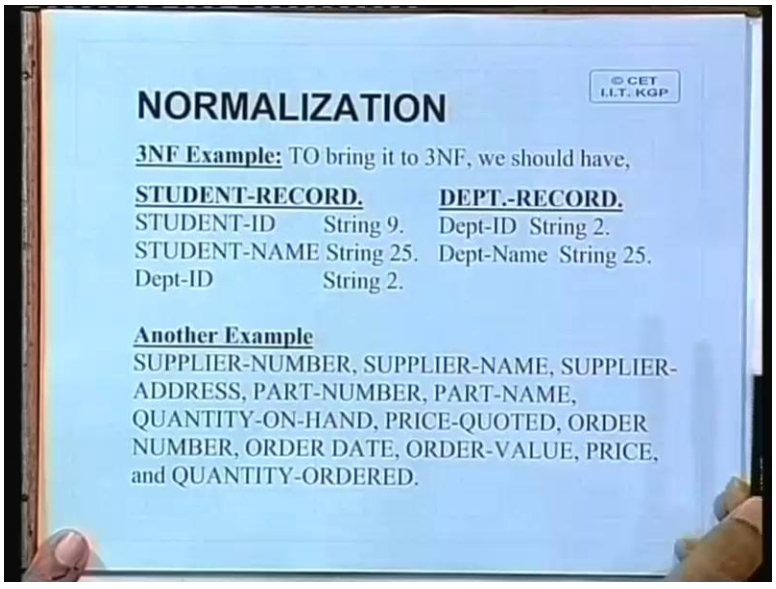
What is the problem? If suppose P. Rao takes another subject. Then P. Rao, you see being repeated. The student name is being repeated for every subject that he has taken. So, since his name is being repeated, every time he takes another subject; if we make a mistake in entering his name even once, there is an update anomaly. So, you must remember. If we have to repeat, we must repeat the key element or the hash or the, you know, the identifier, unique identifier, but not the attributes. If any time the attribute is to be repeated, there is a chance of what is known as your update anomaly or redundancy. So, this was an example of non-full dependency once again.

(Refer Slide Time: 13:38)



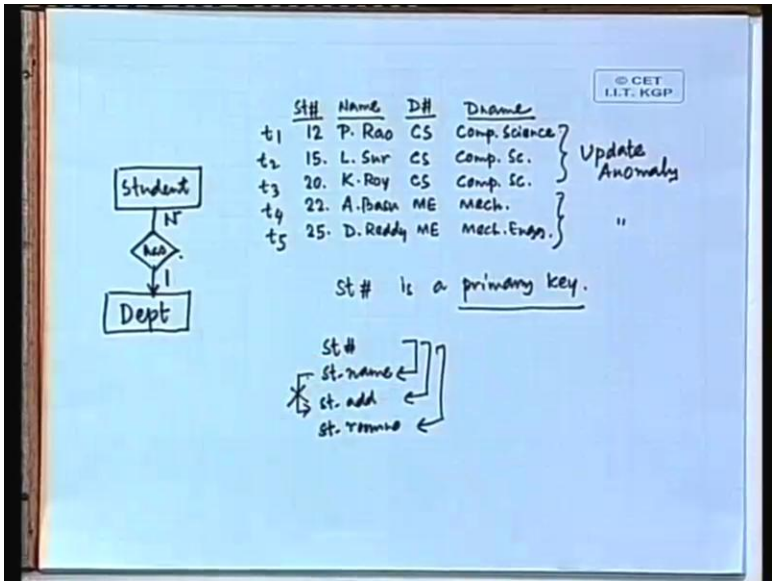
Now, coming back...so this is the example we were discussing that this is not in third normal form. So, it is to be brought into third normal form. So, how to do this?

(Refer Slide Time: 16:41)



Definitely, we have to separate out the department record and the student record. Please recall. This is a many to many relationship, this is a many to many relationship, right? Since this is a many to many relationship, so we have...Yes.

(Refer Slide Time: 17:04)



So, this is a many to many relationship. So, let us draw once again. This is student, this is department. So, department has students. Now, one department can have many students but a student should have only one department. So, this is one to many...sorry...not many to many, one to many relationship. So, one department has got many students, but one student can have only one department. So, it is a one to many relationship.

Now, because of these one to many relationships, I have said, in certain circumstances we can just have only two tables, is it not? Since the number of records of hash and number of records of student should be exactly same. So, instead of having again a third table, it is better that we have a department table and in the student table, we add department number, alright? So, that makes it a concise design.

Rather than all the time translating, whenever you have three structures, you have three different tables. So, exactly that has been done here. That is, the student id, student name, department id, this is the student record; and department record - we have department id and department name. I think you can see that in the 3NF example, we can definitely bring to third NF by suitably decomposing the relationship.

So, essentially the idea is, whenever the basic idea of normalization is that all the data elements you put it into one big table and that is basically first normal form. Then identify the unique identifiers. So basically for all the entities, you put a hash number and then you try to see, out of these hash numbers and their attributes, what are the different kinds of dependencies, that is wrongful dependencies and transitive dependencies. Then keep removing them one after another.

So, when you have removed all of them by decomposing, suitably decomposing them into different sets of tables, what you have got is a normalized database design, alright? Your entity relationship diagram should also give you the same diagram. So, this is an example of slightly bigger one; that is, we have the supplier number. You see where we have got this. We have got this from a given situation where, in an order supply kind of situation where we have a number of suppliers, their number, their name, their address, then we have different parts - like part number, part name, quantity on hand, how many we have right now in the stock, the price quoted...

So whenever you want some material, you are going to a supplier. Supplier quotes a price. Then, whenever an order is given, there is an order number, there is an order date, there is an order value; the price is related to the particular part that is being ordered along with an order and quantity ordered, how many have been ordered for a given part in a given order.

Now suppose if only this much is given to you, it should be possible for you to design a database; but in reality it is not that simple. You see, first of all you must understand the context; you must understand the context that although I said that, definitely that, wherever it may occur, as long as they mean the same thing, the database design should be unique; but you must understand it, is it not? You must understand the ordering process, you must understand what is

an order, what constitutes an order, who is a supplier, what does he supply, can one order go to multiple suppliers?

Suppose one order can go to multiple suppliers. Then the very assumption that an order is unique as far as a supplier is concerned that means one order can have only one supplier - this assumption will not hold. So, you must understand all these. That means you must understand each entity, you must understand its attributes and you must understand the relationship between the entities. That is, the relationship cardinalities, the meaning of every attribute should be very clear to you. If it is not clear, then you cannot make database design. Is it okay? So, that is how it should be done. Now, what should be done? You tell me. Now this is an example. Suppose this is what you have to carry out. How will you go ahead? What is the first thing that you must identify?

Yes. So, what is to be done after you have got a set of...now, is it in any form out of various normalized forms? Is it in any form? It is in first normal form because there are no repeating groups. So, if you simply keep them, it will be in first normal form. Does it have a primary key? Does it have a primary key? Why? They are supplier number, supplier number will uniquely determine supplier details; but what about orders and what about parts? So, all of them...So you see, this is very good. In a flat file design, the flat...

In fact, in my old company, my boss used to say, Mister Vijay Kumar - he was my boss. So, he used to always say that what is there in, let us have a flat file. He was very fond of flat files because very easy to design, alright? All you have to have is, you just keep on having all your fields put together and all the identifiers you identify and that is the primary key. That is all. Your database design is over, but you will pay heavily. You will pay heavily by redundancy update anomaly and this is no design at all, alright?

Then, if that is so simple, then there is no need to study relational DBMS at all. Fine. So, as a first step, you have to identify the order number, the part number and the supplier number as unique identifiers of certain sets of attributes. In fact, if you put them all together, they will become a unique key for the thing in the first normal form; but you can find there are many, what you say, non-full dependencies. What are the different non-full dependencies? Here, supplier

name, supplier address, depends only on supplier. Then part name, quantity on hand - depends only on part number; then order date, order value - depends only on order number.

So this gives us basically three more: price, quoted price and quantity ordered. Can you tell me, on which price quoted depends? Price quoted will be dependent on the supplier number and the part number. Is it okay? Sir, it also depends on quantity how? You see, again you are confused. We will come to that. Then, on which price will depend? Price...

See, price quoted and price are two different things. Every supplier quotes a price, alright? Every supplier quotes a price and this price will ultimately be dependent on the...that is the price quoted, the final price, final price of a part will be on the order number and the order and the particular part. Actually, somebody asked me that. Why not price quoted? Depends on quantity. Actually, many of us...we have seen in some sort of mathematics that the price versus quantity is not a unique...you know, there is a straight line. It is not a straight line. We can see that sometimes when quantity is higher we get discounts all right so all these.

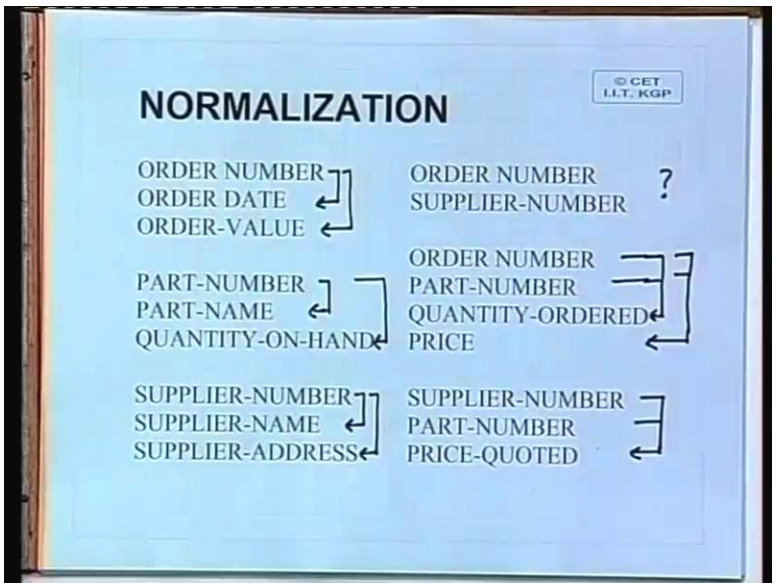
So you know, price is not necessarily a unique item. It may be having lot of if, then, else... It can have a decision kind of a thing, a decision logic built into the price. There are many components of price. So, if you want to really deal price in that manner, it will be complicated. It will not be that easy to put all components of price within the database. So, it is ideal you can have two approaches; if you really want to put it, you can make price.

See, if that is so, then you have to make price as a different table itself, but in a simple sense what you can do? Instead of trying to get price per unit, you can just give price for the quantity being asked, for alright? Suppose we are asking for in the tender or in the quotation for 20 numbers. What is the price given for these 20 numbers, alright? It is then all inclusive price - you can say in that manner and on which can you tell me quantity order depends on quantity ordered, Quantity, Part number and order number. Fine. So, you can see that a large number of attributes are dependent on single identifier whereas certain others like price quoted like price, like quantity ordered, they depend on two identifiers. Sir, one question why does price depend You tell me. Price depends on which identifier part number and part number and order number order number, right?

Now, some people may be tempted to say supplier number when it comes to price; but this is...see, every supplier has quoted certain prices, alright? Now, after making a comparison, we have come up with a unique price which is probably the lowest, may not be lowest. Depends on the criteria by which the decision for the supplier has been made and sometimes even if whatever price is asked by the supplier, whatever price is quoted by the supplier, may not have been agreed to by the ordering agency.

It is not always that price quoted and price they are exactly same; may not be, may not be. See, sometimes it is said, that is 4% sales tax extra, right? Now suppose the organization in concern has a special status for which they do not have to pay sales tax. So, the price quoted and the price of the order, they may differ. They may actually differ, right? So, these are some of the things on which you may have the price quoted for the supplier who is chosen. The price may not be the same, fine. So, this price is actually dependent by order and the price, right?

(Refer Slide Time: 30:52)



So, this has been put. So as a part of normalization, you can group them in this manner. You can see here that all these have been grouped and when this grouping has been made, we can order number depends identifies order date and order value. Then, part number identifies part name, part number identifies quantity on hand, supplier number identifies supplier name, supplier number identifies supplier address. What you see? This is now definitely in second normal form.

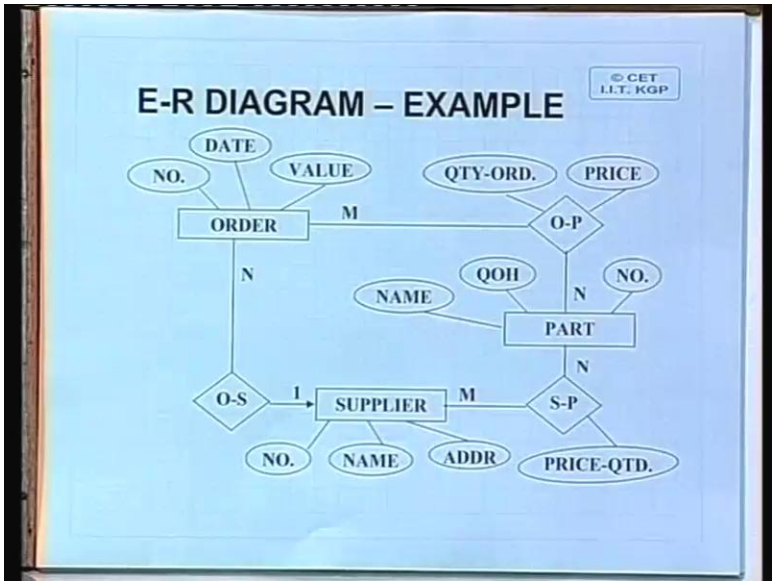
Then, order number and part number, together quantity ordered and they together identifies price; and supplier number and part number, together they identify price quoted. And what about this order number, supplier number? Order number and supplier number - we have not identified any attribute. Now, is it really required? Basically, how to relate order versus supplier, which order is given to which supplier, that is also to be identified.

Now, from a normalization point of view, is it in second normal form? You see, basically what we have done? We have been able to decompose the table into a number of tables; but...so this is the one set of normalization that has been carried out. Now tell me, is it in second normal form? Yes or no? This is definitely in second normal form because there are no non-full dependency. Now, is it in third normal form? Yes or no? It is not in third normal form because...why? See, for something to be in third normal form, you have to have non-full… You have to have transitive dependencies to be removed. Is there any transitive dependency? Yes sir. Some people are saying, yes. So, where do you find transitive dependency?

Supplier number, part number and price quoted. Why? Why you see this? There are only three fields. Now, out of these, supplier number and part number together are primary key which I definitely, uniquely identified, price quoted, alright? So, where is the other one? Is there any other dependency derived defined here? Yes or no? No. So, there is no other they are dependency defined here only one dependency defined here.

So therefore, that is no transitive dependency. So there is in fact no transitive dependency anywhere and we have this one in the third normal form. So, this is a 3 NF design of the database, alright? By suitably differentiating the...suitably differentiating the various dependencies into separate groups and identifying that there are no non-full dependencies or there are no transitive dependencies, therefore we can say that this is in 3 NF, right? So, this is a third normal form design. Now, let us look at it. Let us find out the entity relationship diagram for the same one.

(Refer Slide Time: 35:28)

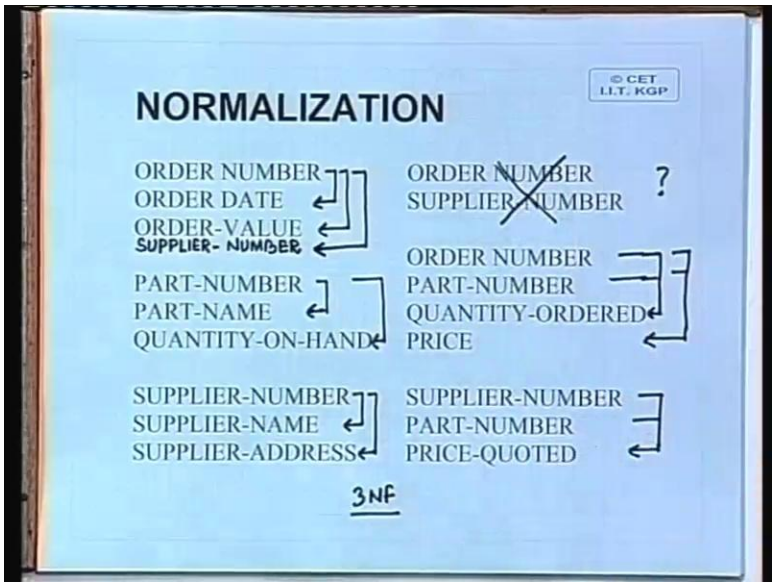


You see, this is the entity relationship diagram for the same table and we have here the order which has a number, which has a date, which has a value. The part which has a number, which has a name, which has a quantity on hand, the supplier which have a number, name, address. So, these are the three basic entities: order, part and supplier. Then we could relate them by order; part, supplier part and order supplier. So, this a three-way relationship. So, three entities - each are connected to one another. Order part - most important – one, it is a many to many because one order can have many parts and one part can also have many orders, right?

So because of these, we can see that the order part again has got the no, two attributes price and quantity ordered; definitely two more attributes are there. That is order number and part number, which is understood. So it is not drawn, right? So, order part has got 4 attributes: quantity ordered, price, order number and part number. But since it is a relationship between these two, it is automatically understood that order number and part number are its attributes. So, they are not shown separately. Supplier part – again, many to many, one supplier can have many part and one part can have many suppliers.

So, this is again many to many and we have the price quoted; but between order and supplier, order supplier. So, one supplier...See, one order can have only one supplier alright but is it the reverse? It is the other way around, is it not? See, a supplier can have many orders. It is correct. So, one order can have only one supplier. So, an order is connected to only one supplier, but a supplier can have many orders. So, is it...it is correct. So, we have a 'one to many' relationship there. Now, since I have said, you know, you can reduce that order supplier.

(Refer Slide Time: 38:21)

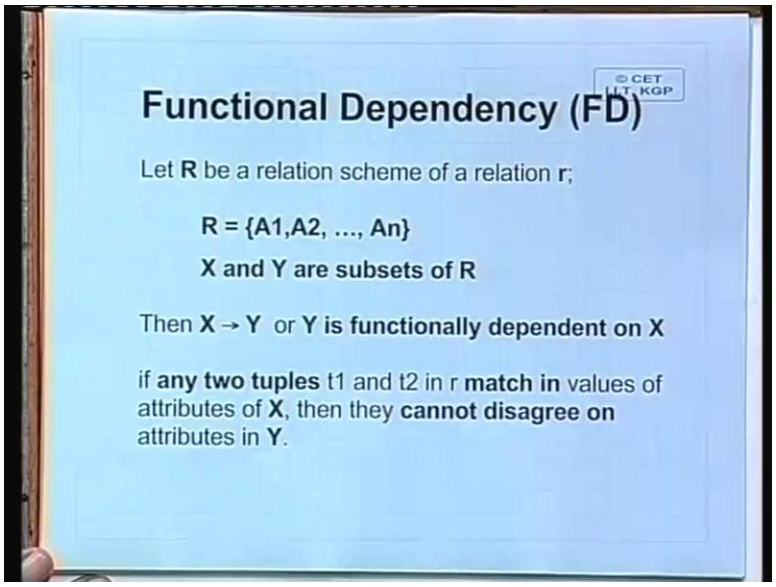


You can definitely make a small change in this. We can add the supplier number in this table. So, we can add supplier number in the first table itself and we can remove this, alright? So, this is going to be our database design; that is order number, order date, order value, supplier number, part number, part name, quantity on hand, supplier number, supplier name, supplier address, order number, part number, quantity ordered, price, supplier number, part number, price quoted, alright? So, this is a first step. This is like a first step towards a design of schema.

Next, what you have to do? You have to identify for each element the picture, alright? String, what is the size, what is the primary key, what are the different integrity checks and controls. So, before you can design an actual schema, lot of other things has to be done; but this is the first step towards it and this is the corresponding entity relationship diagram.

So from this example, I think we are now fairly clear about what should be done when we have to design a database from a layman's point of view, alright? From a layman's point of view, a simple procedural will be that identify all the elements, identify the entities, identify the key elements, the primary keys of each entity. Then separately draw an entity relationship diagram and also try to carry out normalization by removing stage by stage non-full dependency and transitive dependency. Fine. So, that is the thing. Now, let us go ahead and try to look at it from an algebraic point of view.

(Refer Slide Time: 40:50)

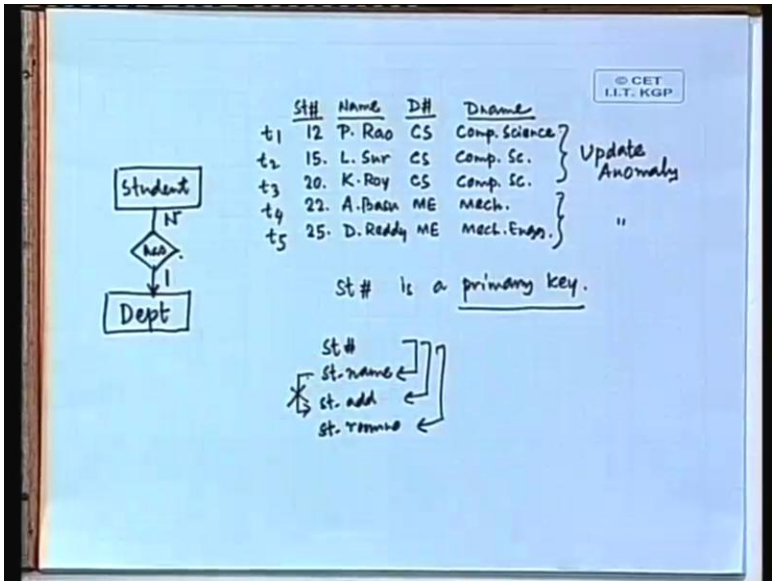


The first one, the first concept is I have already defined that is known as the functional dependency. The...see, for all our discussions, we shall differentiate between a relation r and a relationships schema R, capital R. The capital R and small r are different because you see, every, every database definition will have a relationship, relation scheme, capital R, alright? But it may have different relations because at a given point of time, what data will be stored in the relation will be dependent on, will be dependent on what are the values there available at that time.

So a given relationship schema, scheme can have a number of relations depending on what data is stored there, what are the different tuples of different tables at a given point of time. So, R will be composed of the scheme definition, will be composed of a number of attributes A 1, A 2 to A n and X and Y are subsets of R. Then we say that X functionally determines Y or Y is

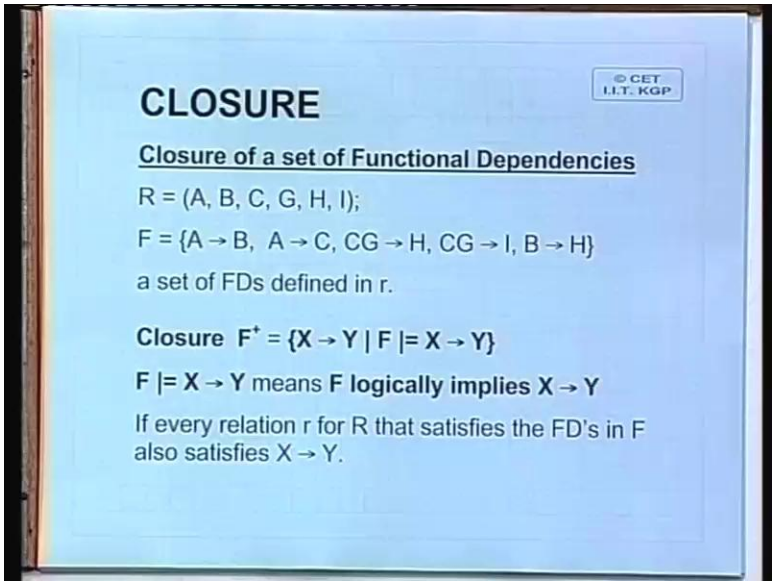
functionally dependent on X. If any two tuples t 1 and t 2 in r match in values of attributes of X, then they cannot disagree on attributes in Y. So, this is what I was telling you in many different ways.

(Refer Slide Time: 42:47)



So this example, once again, you see that there are number of tuples - student hash, etcetera, etcetera. Now out of these different tuples, if due to whatever reason we repeat the student number, say for example if you think that department hash, whether it is a primary key or not in 2 tuples t 1 and t 2 department hash is matching, they are both CS. However, the student hash is different. So, department hash does not uniquely determine or student hash is not functionally dependent on department hash. So, this is how you define what is known as functional dependency.

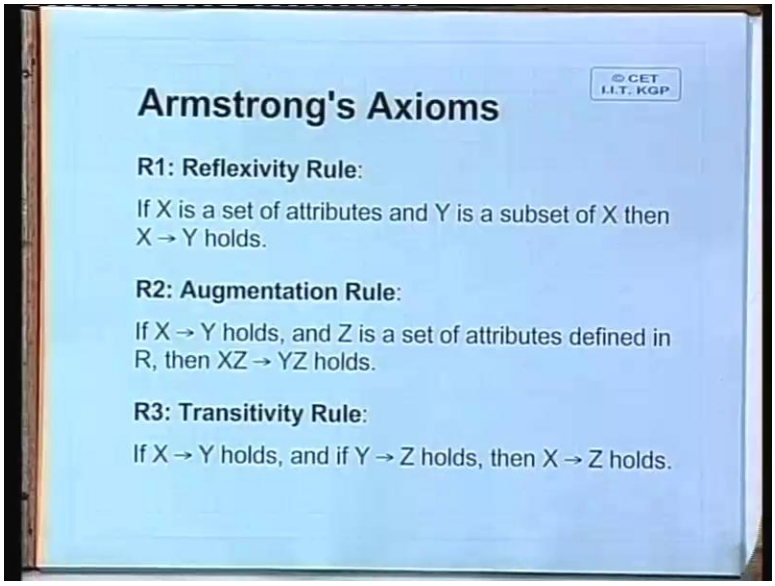
(Refer Slide Time: 43:41)



The...with regard to functional dependency, there is another property which is related to closure, you know, related to functional dependency. That is called closure; so closure of a set of a functional dependency. So you see, it is something like this that you have. Let us say in a relationship schema, relation schema, there is a set of functional dependencies which are defined.

So, this is like...this we have A, B, C, G, H, I; these are 1, 2, 3, 4, 5, 6 attributes which define a relationship schema and these are a set of functional dependencies. That is A determines B, A determines C; CG determines H, CG determines I; and B determines H. These are a set of functional dependencies defined in r. Then the closure or F plus means all the...you see, X X determines Y, F implies X determines Y. That means it includes all the relations, all the... I mean functional dependencies which are logically implied by all this, okay. I think, before we understand closure, may be we discuss Armstrong's Axioms so that our understanding will be slightly clear, more clear.

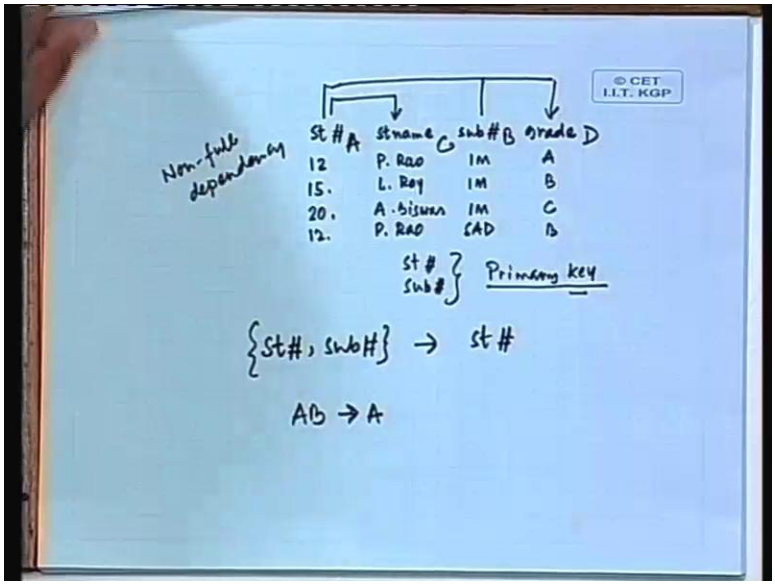
(Refer Slide Time: 45:28)



See, Armstrong's Axioms are basically a set of 6 rules, a set of 6 rules. Out of that, 3 are fundamental and 3 are derived which actually allows you to derive or to obtain or imply further functional dependencies, once certain functional dependencies are available. So, there are basically 3 fundamental rules: the reflexivity rule, the augmentation rule and the transitivity rule, right? The first rule, that is the reflexivity rule says that X is a set of attributes and Y is a sub set of X. Then X determines Y holds or Y is functionally dependent on X. Now you must understand which is not written here is that X and Y, X and Y are all defined in r. You see, X and Y are not coming from two different schema; they are basically from the same schema, has to be from the same scheme and then they are two different attributes.

Now, this is very obvious. Actually, the reflexivity rule...that is, you see...if we have a particular key that is supplier number and part number, a concatenation of supplier number and part number. Now, does supplier number depends on them, it or not see?

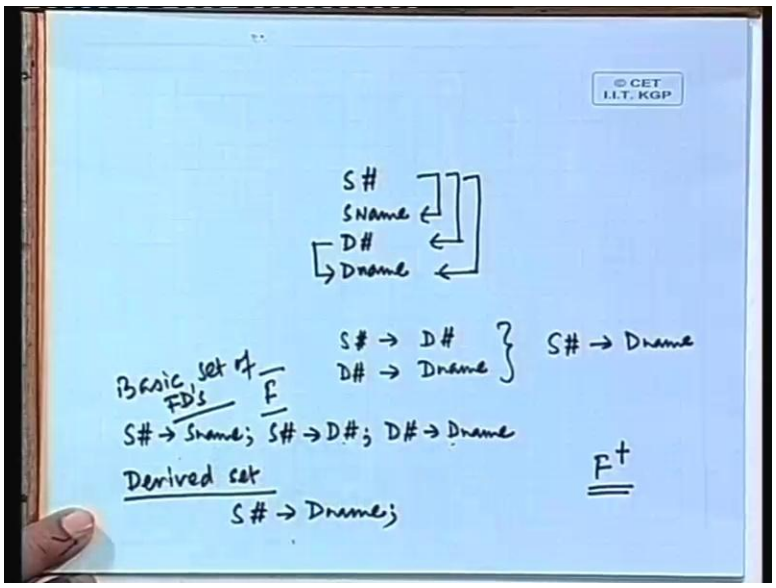
(Refer Slide Time: 47:39)



What I mean is, take this example where we have the student number and subject number; together determines grade. But can we say this? Can we say this in to for to make things simple? Suppose we call this A, call this B, call this C and call this D. Then...so, does AB functionally determines A or A is functionally dependent on AB? These obvious because in two tuples, if AB agrees, A has to agree because A is nothing but a part of AB only, alright? So, this is known as reflexivity rule. The other one, the augmentation rule says that if X determines Y holds and Z is another set of attributes defined in R, XYZ all are defined in R, then XZ, then YZ is functionally dependent on XZ.

So, this is augmentation. That means if you add the same thing on both sides, it is still true. The third one transitivity rule - if Y is dependent on X functionally and Z is functionally dependent on Y, then Z is also functionally dependent on X. That is known as transitive rule. So you see, the...it is easy to get baffled sometimes because of the transitivity rule; particularly, let me bring out the example where we were discussing about the third normal form. You see, look at here. See, in fact, let me take another example.

(Refer Slide Time: 49:52)



See, we have the student number, we have the student name. Now, this is a true transitivity. This is a true transitivity, is it not? You see, now even if this is not true, you see S number determines D number and D number determines D name. So these two, you can actually imply this automatically, is it not? So, it is not necessary that we have to have again separately student number to D name as a separate relation.

So in this case you can see, the basic set of dependency will be this. So, this is the basic set of the FDs. What is the derived set? Derived set will be S hash to D name. So, this is also in the part of derived set. Any other Can you derive any other? Can you derive any other out of this relationship? Can you derive any other? If these three are my basic sets, can you say S name functionally determines D name? Yes or no? No Sir.

No, we cannot. So, we cannot have any more dependencies to be put further, alright? In fact, can you derive any more dependencies here? We cannot. Since we cannot, then you see one two three four that will be the part of the closure I think now you understand closure the closure is all implied dependencies so by using 3 fundamental Armstrong's Axioms and 3 derived Armstrong's Axioms which I have not discussed so using these six Armstrong's Axioms you can keep on identify new FD's the complete set of those FD's is known as the closure. So F is the basic set, so this is like F you see this is F but F plus will have which is the closure of the F

which will have this this this as well as this. So a set of basic FD's and set of derived FD's the complete set of FD's that are available should be part of F plus, so have you understood what is F plus okay so we stop here and in our next class we will continue from here.