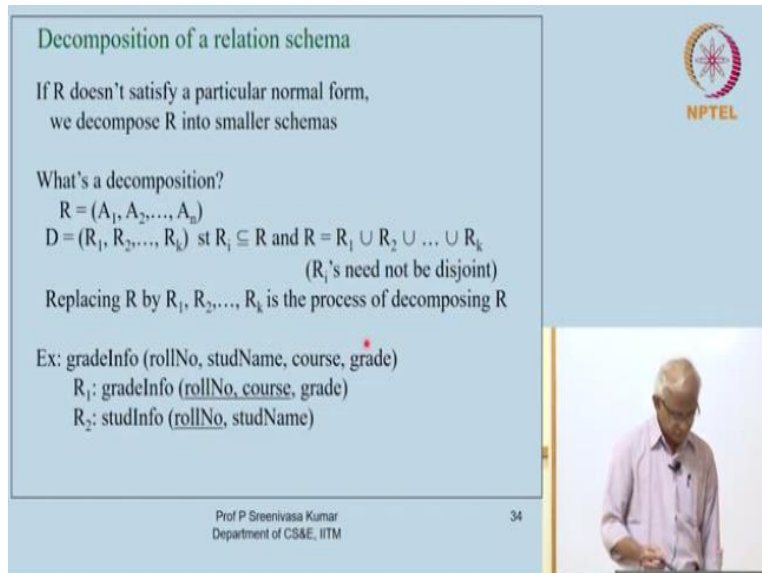


Database Systems
Dr. Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology - Madras

Lecture – 23
Properties of Decompositions

(Refer Slide Time: 00:23)



Decomposition of a relation schema

If R doesn't satisfy a particular normal form,
we decompose R into smaller schemas

What's a decomposition?
 $R = (A_1, A_2, \dots, A_n)$
 $D = (R_1, R_2, \dots, R_k)$ st $R_i \subseteq R$ and $R = R_1 \cup R_2 \cup \dots \cup R_k$
(R_i 's need not be disjoint)

Replacing R by R_1, R_2, \dots, R_k is the process of decomposing R

Ex: gradeInfo (rollNo, studName, course, grade)
 R_1 : gradeInfo (rollNo, course, grade)
 R_2 : studInfo (rollNo, studName)

NPTEL

Prof P Sreenivasa Kumar
Department of CS&E, IITM

34

Okay, so last class, actually happened to be a crucial lecture, so we discussed all the normal forms in the last lecture, many of you could not attend, I hope you have check them the slides, so we discussed what are called 2NF, 3NF Boyce Codd normal form in the last lecture, definitions, examples and all that. Now, a typical situation that we have seen occurring you know in the last lecture was that we find certain relation to be not satisfying a particular normal form.

And then what we do is; we decompose that relation, we kind of; we distribute the attributes in the relation into 2 different relations and then we see that the same information is you know, the presenter here and each of those individual relations are satisfying the normal form that we are looking for like for example, if we does not satisfy Codd normal form, we have decompose that relation into 2 and then we have seen that both of them now satisfy the Codd normal form.

Now, this process of you know taking a relation and then decomposing it is a phenomenon that is something that we have done several time in the last lecture, in each time when we find that a

relation instance, a relation does not satisfy the normal form. So, let us try to make that a little bit more formal and then actually, study certain properties of these decompositions that these decompositions are important.

So, we will realise during this course of this lecture that you know, all possible you know taking arbitrarily cut of these relations and then you know bringing them into 2 relations will not work, we have to look at certain properties before we can do that. Okay, so let us see what is the decomposition in the first place, so if R is a relation scheme, so it has some N number of attributes, if we replace R by this k number of relations, R_1 through R_k , where each of this R_i is a subset of R .

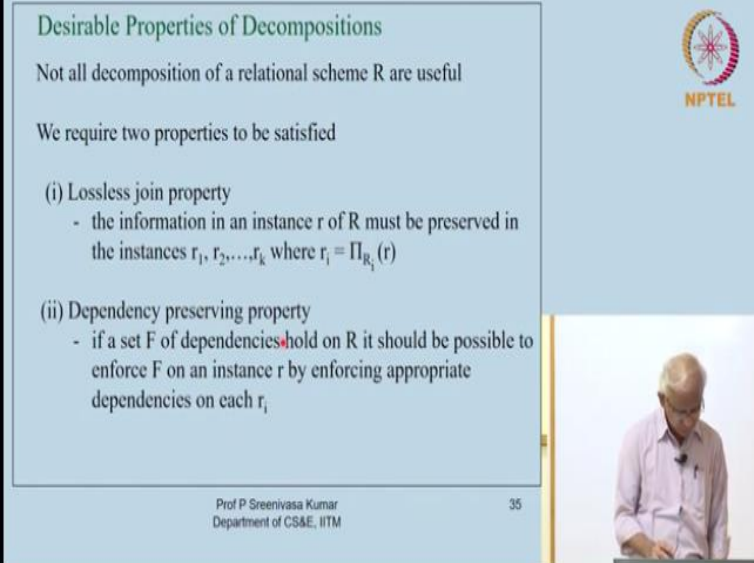
And they collectively have all the attributes of R , okay, collectively have all the attributes of R , then such a replacement of R through by this R_1 through R_k is what is called a decomposition, okay and of course this R_i 's need not be disjoint because they would not typically be disjoint okay, so you will need some attribute to be in more than 1 relation, so that you know it serves the purpose of a foreign key, primary key reference, okay.

So, such a thing is what is called a decomposition; a decomposition is nothing but taking a relations scheme and replacing due to it R_1 through R_k and we call this as D , the decomposition and these R_1 through R_k as such that R_i is a subset of R and they collectively give that our set of attribute form (\cup) (04:12). For example, in the last lecture we were looking at these relations called grade info, which had roll numbers, student names, course and grade.

And this was satisfying Codd normal form but it still had redundancy in that, so we went for a stricter normal form for that which is called the Boyce Codd normal form. So, during that thing we have decomposed this into these 2 relations; roll number, course, grade and roll number, student name, so it is an example, so we replace this R by this R_1 and R_2 , this is the R and we replaced it by the attributes of in the 2 relations that will mainly its back the entire set of attributes, okay.

Now, let us look at; can we do this arbitrary, can we you know, are there any problems in this, is there any systematic way in which this has to be done, these are the various questions that come up in the context of these decompositions.

(Refer Slide Time: 05:35)



The slide is titled "Desirable Properties of Decompositions" and contains the following text:

Not all decomposition of a relational scheme R are useful

We require two properties to be satisfied

- (i) Lossless join property
 - the information in an instance r of R must be preserved in the instances r_1, r_2, \dots, r_k where $r_i = \Pi_{R_i}(r)$
- (ii) Dependency preserving property
 - if a set F of dependencies hold on R it should be possible to enforce F on an instance r by enforcing appropriate dependencies on each r_i

At the bottom of the slide, it says "Prof P Sreenivasa Kumar, Department of CS&E, IITM" and "35". In the top right corner, there is an NPTEL logo. A small video inset in the bottom right shows a man in a light purple shirt speaking.

So it turns out that not all decompositions of a relations scheme are going to be useful, we require that these decompositions satisfy certain important properties, so we call there are 2 properties; one of them is called lossless joint property, the other one is called dependency preserving property. So, basically what we here require is that whatever the information that was present in R the; any instance R from the scheme, original scheme. Now, if you decompose the relation scheme R into say R1 and R2, what will happen to the data?

Obviously, you have to split the data also, obviously, you have to take the data, project it onto R1, project it to R2 and then create instances of data instances for these 2 schemes R1 and R2 that we decomposed from R, right, so data has to be also created, the original data which is the instance R has to be you know use in order to derive instances r1, r2. So, how do you derive this r1, r2, rk, naturally the both natural way of doing that is to take the projection of the original instance r on to these Ri, right.

So that is what we; so whatever be the information that was present in R should now be present in r1 through rk, okay. So, what do we mean by it should be present in; it will actually will be

present right because any way you are taking projection of R and then getting these r_1 through r_k , so wherever was the tuples that were in R will automatically you know, from segments of those tuples will come into r_1 through r_k .

So, each of these tuples that is there in R now will get segmented in something actually, right. So, a few values of them will go into r_1 and they may be some overlap and then some other segment will go into r_2 like that, the data will go into all these; so, we will see exactly in the next slide actually, whether they can be actually some kind of a loss in this process, then this is one thing.

The other thing is that supposing there was a set of functional dependencies that was holding on the relation scheme R okay, how do we appropriately you know translate those functional dependencies, if each of these R_i 's, so that you know enforcing these functional dependencies and each of these instances r_i ; each of these decomposed relations r_i is equivalent to enforcing the functional dependencies on the original thing; original relation scheme, how do we get possible, that is another thing.

So, if a set of functional dependencies hold on R , it should be possible for us to enforce this set of functional dependencies, what do we mean by enforcing; basically, if we want to see the relation scheme satisfies those functional dependencies, whether it is satisfying or not, we want to check that. So checking that should be possible for us to you know by enforcing some appropriate derived functional dependencies on each of these r_i 's, okay. So, these are the 2 properties that we will look for, so we will go into a bit more detail into these things now, okay.

(Refer Slide Time: 10:03)

Lossless join property

F – set of FDs that hold on R
 R – decomposed into R_1, R_2, \dots, R_k
 Decomposition is *lossless* wrt F if
 for every relation instance r on R satisfying F,
 $r = \Pi_{R_1}(r) * \Pi_{R_2}(r) * \dots * \Pi_{R_k}(r)$

Lossless joins are also called non-additive joins

Original info is distorted



$R = (A, B, C); R_1 = (A, B); R_2 = (B, C)$

r:	A	B	C	r ₁ :	A	B	r ₂ :	B	C	r ₁ *r ₂ :	A	B	C
	a ₁	b ₁	c ₁	a ₁	b ₁	b ₁	c ₁	a ₁	b ₁	c ₁	a ₁	b ₁	c ₁
	a ₂	b ₂	c ₂	a ₂	b ₂	b ₂	c ₂	a ₂	b ₂	c ₂	a ₂	b ₂	c ₂
	a ₃	b ₁	c ₃	a ₃	b ₁	b ₁	c ₃	a ₃	b ₁	c ₁	a ₃	b ₁	c ₁
											a ₁	b ₁	c ₁

Lossy join Spurious tuples

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

36

So, in this context first thing is we will elaborate on the losslessness property that means there should not be any loss in information, okay. So, let me said that up a little bit more formally now, let us say we have a set F or the functional dependencies that hold on R and R is decomposed into R1 through Rk. Now, this particular decomposition is called lossless with respect to the set of functional dependencies, when in any instance; every instance r, on these R, this scheme that satisfies this functional dependencies.

What happens here is that if you take r and project it into R1, project it into R2, you get little r1, little r2, little r3, little rk etc., you do a natural join among them, you should get back r. Why natural join; because attributes are all; same attributes are you know, there are some common attributes; common attributes are there, okay, r has some say A1 to A10 you know, so r1 through rk will have basically, this A1 through A10 right but then some subsets will be in R1, some subsets will be in R2 etc.

So, it is possible for us to do a natural join because there are some common attribute, so you project the instance, for any instance, you take any instance on R; on R and then you get back these instances by projecting them onto R1, R2, R3, Rk and if we join them back, you should get r because in any way originally, this information came from r, so if we join them you should get back r. If this is the case, then we call it lossless, this looks fair, right.

This kind of a restriction looks fair because basically, we had some data and now we are saying that this scheme is going to be split into 2 schemes, so we project the data into 2, we should join back, if we join, we should get back the original r , okay, so that is fair, right that is what we will ask for in this property. Now, one subtle thing that we have to notice here is that if there is a tuple in the original relation, right and let us say we have split it into 2, let us just for the $((13:11))$, consider 2 of them and so there are some common attribute let us say.

So, we split it into 2, right, 2 tuples have come. Now, obviously if you do a join because these 2 tuples exist, you will always get back to your original tuple, do you agree with that; you take a tuple in R , okay, project it into R_1 , project it onto R_2 , you get 2 sub tuples, right, so if the same tuple became into 2 different tuples now, one on R_1 , the other one on R_2 . So, suppose you do a join on little r_1 , little r_2 , because this particular 2 tuples exist in those individual relations, you will get back your original tuples, here these 2 things joined where they have the same common value any way.

They have some common attribute, let us say they have some attribute, the common attribute value will go into left hand side tuple also into the right hand side tuple and so when we do a equijoin, these 2 tuples will agree on the common attribute and so you will get back to your original tuple, is that clear, I can repeat if you want. See, just consider 1 tuple in the original relation, we are splitting that; we are assuming that the original relation is decomposed into 2 relations; R_1 and R_2 .

So, if you take the tuple, when you project the data, one part of the tuple will go into R_1 , another part of the tuple will go into R_2 and because these 2 tuples exist in R_1 and R_2 , when we do a natural join between them, we will always get back to your original tuple, okay. So, what I am assumed here is that or telling you here is that this will always contain r , the trouble is it may not be exactly r , it may contain something more than r that is the trouble.

By containing more than what is there in r , it is actually correcting the information, I will now show you an example, where such a thing happens and those tuples; those extra tuples uninvited you know, guess for us something like that is a trouble for us actually, okay. Here is an example,

just look at them, just look at this, focus on this example, we have A, B, C and we have decomposed it into AB and BC, these are the instance a1, b2, c1 and all that, so I have taken them, projected them here.

Now you do a join these 2, now do a join here, so a1, b1 joins the b1, c1 and we will get a1, b1, c1, one of the original tuples but unfortunately, a1 b1 also joins with b1 c3 and then you will get a1 b1 c3 which is actually not there in the original data, okay. So, it is possible that we will actually get some spurious tuples; these tuples are called spurious tuples, so some extra tuples will come into the picture.

And because of this we will call this as a; this particular decomposition is called a lossy decomposition, the original information is distorted and so we will actually call it as a lossy decomposition, even though we are actually having additional tuples, okay. So, those decompositions that do not have any spurious tuples are what are called loss less join; lossless decomposition.

And in the easier Nomenclature could be that they are called non additive decomposition, they do not add spurious tuples, so they are called non additive decomposition, okay, so, this is what is called as loss less join property for decompositions. Now, this is truly an undesirable kind of situation, we do not want this, whenever we do a decomposition, we would like the information to be preserved in the decomposed relation.

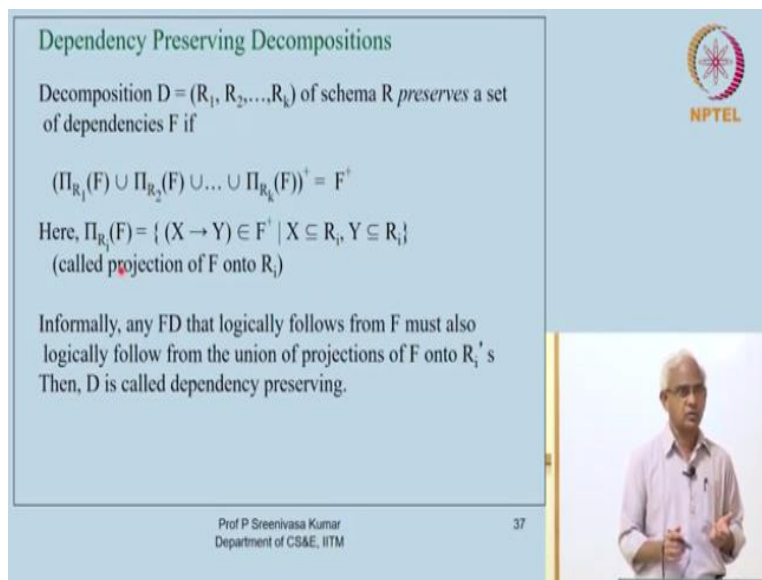
So, when you take the decomposed relations, okay join them, do a natural join, you must get back exactly what is there in R, you should not get additional tuples, there should not be any additional tuples. If there are additional tuples, then we say that there is actually a loss of information there, so we call such a decomposition as a lossy decomposition, okay. Now, you can actually, if we carefully watch this, you know this B is a common attribute for AB and BC.

And B neither determines A nor determine C from the functional dependency point of view, B does not determine C, B does not determine A and because of that if it has certain common some values, the other values can be anything, you know and because of that you can see that when we

do a join, you will get some spurious tuples okay, so there is some issue involved in this, so people who have studied this property and then said that okay, this kind of phenomenon is happening.

And we do not want this kind of this thing to happen whenever we do decompositions, so we have to be careful about this decompositions, we should always put this conditions in that we must get loss less decompositions, our decompositions that we are trying to get in order to achieve certain normal form, it will always be loss less, so that is the kind of, okay.

(Refer Slide Time: 20:59)



The slide is titled "Dependency Preserving Decompositions". It contains the following text:

Decomposition $D = (R_1, R_2, \dots, R_k)$ of schema R preserves a set of dependencies F if

$$(\Pi_{R_1}(F) \cup \Pi_{R_2}(F) \cup \dots \cup \Pi_{R_k}(F))^+ = F^+$$

Here, $\Pi_{R_i}(F) = \{ (X \rightarrow Y) \in F^+ \mid X \subseteq R_i, Y \subseteq R_i \}$
(called projection of F onto R_i)

Informally, any FD that logically follows from F must also logically follow from the union of projections of F onto R_i 's. Then, D is called dependency preserving.

At the bottom left, it says "Prof P Sreenivasa Kumar, Department of CS&E, IITM". At the bottom right, there is a small video inset showing a man speaking, and the number "37". The NPTEL logo is in the top right corner.

The other one property, the desirable property for decompositions is called dependency preserving property, okay. So, here also let me now, define what exactly are we talking about, let us take a decomposition D which has some A number of components, you can now call it probably like that. It is said to preserve a set of functional dependencies here, if the union of $\pi_{R_1}F, \pi_{R_2}F, \pi_{R_k}F$, the closure of that union is exactly same as the closure of the F , okay.

What is this $\pi_{R_1}F$? The $\pi_{R_1}F$ normally, we use π for projections; data projections but here we are kind of slightly misusing the notation because kind of conveniently use it, we are now applying it on functional dependency, set of functional dependency, okay, so it is defined like this. The set of all functional dependencies that are there in the closure whose left hand side and right hand side are part of this R_i , okay, R_i is a attribute; subset of the attributes of R , okay.

So, when you take the set of all functional dependencies that are supposed to hold on the original scheme R that is F closure, it makes sense to take those FD's in that closure you know, which contain the attributes that are there in R_i and then put them separately as those applicable for R_i and that is what we are doing here. So, $\pi_{R_i} F$ here, okay we can call it as F under script I , if you want, is the set of all those functional dependencies in F plus, whose left hand side and right hand side are both part of R_i .

They do not contain any other attribute other than what is there in R_i , so there in some schemes applicable to R_i only, so okay, so that is what we call them as the functional dependencies that are in some schemes, applicable to a particular scheme R_i ; R_i is a subset of R , right. So, we can call it as a projection of F onto R_i because normally, we talk about projection only for data but now, we are projecting the functional dependencies onto a sub scheme are called R_i , okay.

Now, the decomposition is said to preserve the functional dependencies, if you take these you know projections, take a union of all these projections and then take their closure, if that is equal to the F closure that means, all functional dependencies are available either directly or in each of these you know F_i 's or they can be derive from these F_i 's. If that is the case, then we call this decomposition as a dependency preserving decomposition.

So, I have written it here that any FD that logically follows from F must also logically follow from the union of the projections of F onto each of these R_i 's, we will call $(\cup F_i)$ (25:00), this is also fair thing to ask for, right you have a bunch of dependencies and the relation instance is supposed to satisfy all those functional dependencies at all point of time and so the RDBMS is actually fairly responsible for ensuring that happen.

And so if now you take a particular relation, then split it into 2 relations, then the RDBMS should have a way of enforcing them, so what the decomposition if it satisfies this dependency preserving property, then what it can do is that I can take the projection of F onto R_i , R_1 and R_2 and enforce them, it is equivalent to enforce in the relational set of function, okay, so that is what we want, so these are the 2 interesting properties of decompositions that we are looking for.

So, we are not interested in the arbitrary decomposition of a relation R, we are interested in those decompositions that satisfy these 2 properties; one is called the loss less joint property, the other one is called the dependency preserving property. Do you have any questions, please raise them now, because we will then discuss specific algorithms for; actually checking for the loss lessness, we now need a method of checking, whether a decomposition is indeed loss less with respect to a set of functional dependency, okay you need a method to do that.

(Refer Slide Time: 26:53)

An example

Schema $R = (A, B, C)$
 FDs $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

Decomposition $D = (R_1 = \{A, B\}, R_2 = \{B, C\})$
 $\Pi_{R_1}(F) = \{A \rightarrow B, B \rightarrow A\}$
 $\Pi_{R_2}(F) = \{B \rightarrow C, C \rightarrow B\}$

$(\Pi_{R_1}(F) \cup \Pi_{R_2}(F))^+ = \{A \rightarrow B, B \rightarrow A, B \rightarrow C, C \rightarrow B, A \rightarrow C, C \rightarrow A\} = F^+$

Hence Dependency preserving

Prof P Sreenivasa Kumar
 Department of CS&E, IITM


38

Okay, I have an example here which illustrates dependency preserving, just have a look at the simple scheme A, B, C and we have FDs AB; A determines B, B determines C and C determines A (()) (26:53) is a cyclic, so the decomposition R1, AB and BC now, if we take pi R1 of F is all those FD's in F or F closure that are applicable to the scheme AB, so obviously A determines B is applicable, B determine C is not applicable because C is not there here, this is also not applicable.

But then because of B determine C and C determines A, we have B determines A and so that is actually applicable, right. So, A determine B, B determines A will connect here, in a similar way B determines C, C determines will be connected here and if you take the union of these 2 things and take a closure of that, we will see that it is actually equal to A, not equal to this, it is equal to

the closure and so this particular decomposition is loss; I mean dependency preserving with respect to this set of functional dependency for this relation R.

(Refer Slide Time: 28:52)



Testing for lossless decomposition property(1/6)

R – given schema with attributes A_1, A_2, \dots, A_n
F – given set of FDs
D – $\{R_1, R_2, \dots, R_m\}$ given decomposition of R


Is D a lossless decomposition?

Create an $m \times n$ matrix S with columns labeled as A_1, A_2, \dots, A_n
and rows labeled as R_1, R_2, \dots, R_m

Initialize the matrix as follows:
set $S(i,j)$ as symbol b_{ij} for all i,j .
if A_j is in the scheme R_i , then set $S(i,j)$ as symbol a_j , for all i,j

Prof P Sreenivasa Kumar
Department of CS&E, IITM

39



It turns out that okay, we have an algorithm that can check for whether a decomposition is indeed loss less or not, okay, so given some scheme R, with attributes A1 to An and given some functional dependency FD's that is where the decomposition R1 through Rm; we are going to answer this question; if D is a loss less decomposition, okay, so here I am going to present a intriguing looking algorithm but will not give any clue for this, okay, it is an interesting algorithm.


So, let me present that algorithm, this algorithm works on a small matrix actually, we create a m by n matrix, where m is the number of components that are there in the decomposition and n is the total number of attributes that are present in the original scheme, we create an m by n matrix called S, and the column means are all these attributes, A1 through An, and the row actually, I have labelled by these relations, R1, R2, Rm the components of a decomposition, okay and it is kind of initialise like this.

The ith row, jth column is initialise to S symbol called b_{ij} uniformly, so all b symbols, but b symbol is distinct from other because it has a subscript ij; b_{ij} , standing for the ith row, jth column. Now, so some of these attributes are part of these schemes; R1 through Rm, so that

information is captured like this. If some A_j , which is a column right, if A_j is in the scheme R_i , schemes are in the rows, okay, scheme R_i .

Then, what we are going to do is to set the i th row, j th column into a symbol called a_j , so this does not have the; notice that this does not have 2 subscripts, just has 1 subscript, the column number so, this is how we visualise this matrix.

(Refer Slide Time: 32:10)



Testing for lossless decomposition property(2/6)


After S is initialized, we carry out the following process on it:

```
repeat
  for each functional dependency  $U \rightarrow V$  in  $F$  do
    for all rows in  $S$  which agree on  $U$ -attributes do
      make the symbols in each  $V$ -attribute column
      the same in all the rows as follows:
        if any of the rows has an "a" symbol for the column
          set the other rows to the same "a" symbol in the column
        else // if no "a" symbol exists in any of the rows
          choose one of the "b" symbols that appears
          in one of the rows for the  $V$ -attribute and
          set the other rows to that "b" symbol in the column
until no changes to  $S$ 
```

At the end, if there exists a row with all "a" symbols then D is lossless otherwise D is a lossy decomposition

Prof P Sreenivasa Kumar
Department of CS&E, IITM

40



And then we do something interesting with this matrix. We repeat certain; we actually what we do take this relation, take this matrix as some kind of a instance on the scheme R and try to enforce all the functional dependencies that are given to us, ensure that this particular data actually satisfies those functional data's. So, how do we do that; is that for each of this functional dependencies U determines V in the F , what we will do is; for all rows in this matrix that agree on the left hand side attributes; U attributes, okay, make the symbols in V , right hand side V attribute columns.

For each V attribute column, okay, make the values same in all those ways, let us say there are some 3 rows, all of them are agree on the U attributes, okay. So, take one of these V attributes and then ensure that in that column, the same value is existing in all those rows. If you do that then basically, we have ensured that U determines V is enforce on that matrix, okay, so, let me elaborate that actually, how exactly we will do that, there is a method of doing that.

So, if any rows; if any of the rows has the a symbol, what are these rows; these rows are the ones that are having the same values for the U attribute, okay. Now, we are focussing on one particular V attribute, the right hand side of it, in that we will look at all the rows, okay. If any of those 2 rows have, if any of the rows actually have an a symbol, we give priority to the a symbols if any of them have an a symbol then we put that a symbol in all the rows, of the column, okay, that will make it same.

So, if any of the rows has an a symbol for that column, set the other rows also to the same a symbol, otherwise that means, it does not have that particular column of that V attribute, does not have any a symbol at all, so it has some only this bij's, then choose one of them, choose some these symbol arbitrary and then make everybody else equal to that V symbol, so there is some non-determines here, it does not matter with these symbol is used, take them and then do that.

And you do this for there are 2 kinds of things, for each of the attributes V you do this and then go back here and then take up another functional dependency UV, you do the same thing, repeat the same thing, repeat until no changes occur will the matrix does, so this is the algorithm, this almost looks like some kind of you know magic kind of algorithm and we; what do you do at the end of this thing; is that at the end, if there exist a row with all the a symbols, then the decompositions is loss less, otherwise the decomposition is lossy, okay.

I urge you to take that example and then you know trying this out actually or I will be giving some anyway, I will going to give you an illustration of this particular. So, at the end of all these changes to the matrix, what we are doing, going to check is that is there does exist a row with all a symbols, if that is the case, then D is loss less, otherwise D is lossy.

(Refer Slide Time: 37:03)

Testing for lossless decomposition property(3/6)

$R = (\text{rollNo}, \text{name}, \text{advisor}, \text{advisorName}, \text{course}, \text{grade})$

FD's = { $\text{rollNo} \rightarrow \text{name}$; $\text{rollNo} \rightarrow \text{advisor}$; $\text{advisor} \rightarrow \text{advisorName}$
 $\text{rollNo}, \text{course} \rightarrow \text{grade}$ }



D : { $R_1 = (\text{rollNo}, \text{name}, \text{advisor})$, $R_2 = (\text{advisor}, \text{advisorName})$,
 $R_3 = (\text{rollNo}, \text{course}, \text{grade})$ }

Matrix S : (Initial values)

	rollNo	name	advisor	advisor Name	course	grade
R_1	a_1	a_2	a_3	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	b_{25}	b_{26}
R_3	a_1	b_{32}	b_{33}	b_{34}	a_5	a_6

Prof P Sreenivasa Kumar
Department of CS&E, IITM

41

So, here is an example, study this carefully, this is some kind of familiar attributes for you, we have roll number, name of the student, advisor; advisor means advisor ID, then advisor name, course; course means course ID, then grade and the functional dependencies are roll number determines name obviously, roll number determines advisor, advisor determines advisor name and roll number course determine the grade, these are look very appropriate for this relation.

Now, we are going to do this decomposition like this roll number, name, advisor; advisor, advisor name; roll number, course and grade, see this particular relation you can see that it has information about different aspects mixed up into one relation. So, if you check based on these functional dependencies, if you check the normal forms, it will not be in an appropriate, it would not be in you know, this 3NF etc., okay, so it has trouble.

So, let us consider this decomposition, which is actually pretty logical decomposition of this because R_1 has information about students, R_2 has information about advisors and R_3 has information about students and courses, okay so this is the neat decomposition of the original scheme. Now, let us just use this to illustrate this algorithm; the algorithm has the matrix where all the original you know attributes were all there as columns.

And then these 3 things R_1 , R_2 , R_3 are the rows here, initially we will fill it up with all b symbols; b_{14} , b_{15} , b_{16} like that but then some, if some attribute is present in the; in a particular

relation, we make them into a symbol, that is what the initial relation will be but then that a symbol will be distinct because we will use the column name for that, so a1 will be used here, a2 will be used here, a3 will be used here, put a3 in all those places where these particular attribute occurs.

(Refer Slide Time: 40:02)

Testing for lossless decomposition property(4/6)


$R = (\text{rollNo}, \text{name}, \text{advisor}, \text{advisorDept}, \text{course}, \text{grade})$


$FD's = \{ \text{rollNo} \rightarrow \text{name}; \text{rollNo} \rightarrow \text{advisor}; \text{advisor} \rightarrow \text{advisorName}$
 $\text{rollNo}, \text{course} \rightarrow \text{grade} \}$

$D : \{ R_1 = (\text{rollNo}, \text{name}, \text{advisor}), R_2 = (\text{advisor}, \text{advisorName}),$
 $R_3 = (\text{rollNo}, \text{course}, \text{grade}) \}$

Matrix S : (After enforcing $\text{rollNo} \rightarrow \text{name}$ & $\text{rollNo} \rightarrow \text{advisor}$)

	rollNo	name	advisor	advisor Name	course	grade
R_1	a_1	a_2	a_3	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	b_{25}	b_{26}
R_3	a_1	$b_{32}a_2$	$b_{33}a_3$	b_{34}	a_5	a_6





Prof P Sreenivasa Kumar
Department of CS&E, IITM

42

Now, let us start you know, enforcing this functional dependencies, after enforcing roll number determines name and roll number determines advisor, what happens here is the; so there are 2 row that agree on roll numbers, a1, a1, so originally it had b32, now we focus on the right hand side attribute which is name, and then in that column, we have one a attribute value; a symbol, so we will make the other one also equal to the same a symbol and then ensure that these 2 are same.

(Refer Slide Time: 40:47)

Testing for lossless decomposition property(5/6)

$R = (\text{rollNo}, \text{name}, \text{advisor}, \text{advisorDept}, \text{course}, \text{grade})$

FD' s = {rollNo \rightarrow name; rollNo \rightarrow advisor; advisor \rightarrow advisorName
rollNo, course \rightarrow grade}

D : { $R_1 = (\text{rollNo}, \text{name}, \text{advisor})$, $R_2 = (\text{advisor}, \text{advisorName})$,
 $R_3 = (\text{rollNo}, \text{course}, \text{grade})$ }



Matrix S : (After enforcing advisor \rightarrow advisorName)

	rollNo	name	advisor	advisor Name	course	grade
R_1	a_1	a_2	a_3	$b_{14}a_4$	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	b_{25}	b_{26}
R_3	a_1	$b_{32}a_2$	$b_{33}a_3$	$b_{34}a_4$	a_5	a_6

No more changes. Third row with all a symbols. So a lossless join.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

43

And same is the case with the advisor, we had a_{32} , so we are going to face and then we will do one more; advisor determines advisor name, so now in the current matrix, we find that there are 2 rows that agree on advisor a_3 , in fact all of them agree on advisor and so we do a enforcement here, where and one of them had a relatively a symbol and so we replace the other b symbols with that a symbols, so now suddenly we find that last row has all a symbols; $a_1, a_2, a_3, a_4, a_5, a_6$ and you can also see that there is no more changes that can occur in the matrix and so it actually a loss less.

This kind of a looks like magic right, I mean there is no; I have not presented any particular logic has to why this algorithm should work and I am not going to present actually, I leave it to you kind of try and figure out by consulting some foundation books on relational databases in particular; the proof of this particular algorithm makes use of a variant of tuple relational calculus called domain relational calculus and this present; the proof is present in Jack Wellman's principles of data base system book.

So, I am not going to present the proof, I leave it as something which will intrigue you to go and figure out why this book, okay. So, this is how, you should you can test for lossless decompositions of properties.


(Refer Slide Time: 43:09)


Testing for lossless decomposition property(6/6)

R – given schema. F – given set of FDs

The decomposition of R into R_1, R_2 is lossless wrt F if and only if either $R_1 \cap R_2 \rightarrow (R_1 - R_2)$ belongs to F^+ or $R_1 \cap R_2 \rightarrow (R_2 - R_1)$ belongs to F^+

Example:
 gradeInfo (rollNo, studName, course, grade)
 with FDs = {rollNo, course \rightarrow grade; studName, course \rightarrow grade;
 rollNo \rightarrow studName; studName \rightarrow rollNo}
 decomposed into
 grades (rollNo, course, grade) and studInfo (rollNo, studName)
 is lossless because
 rollNo \rightarrow studName





Prof P Sreenivasa Kumar
Department of CS&E, IITM

44

So, we will stop here, let me just conclude by saying that this particular algorithm is needed when you have you know more than 2 number of components in the decomposition, in there exactly 2 given scheme here and then given set of functional dependencies, the decomposition R into R1, R2, just 2 of them is loss less with respect to a; this is a simplest check that we can do, is that R1 intersection R2 determines R1 - R2 or R1 intersection R2 determines R2 - R1.

If this happens, if any one of these things belongs to a plus, then such a decomposition is loss less, this is another interesting property of this decompositions, so this is easier to check, you do not have to go to an elaborate algorithm for doing this. So, for example our decomposition of grade info into roll number course, grade and roll number student name is in the loss less because the common attribute that is there between these 2 things is roll number.

And roll number determines student name which is R2 - R1 and I urge you to check this condition on the example that I have gave you which has A, B, C and then it has spurious tuples, okay. So, in summary what we have done today is to observe that there are some interesting properties that we desire for decompositions, these are called lossless join property and dependency preserving property and we have defined them formally and then we have looked at an algorithm for checking lossless dependency property for a decomposition, okay, good, so we will stop here.