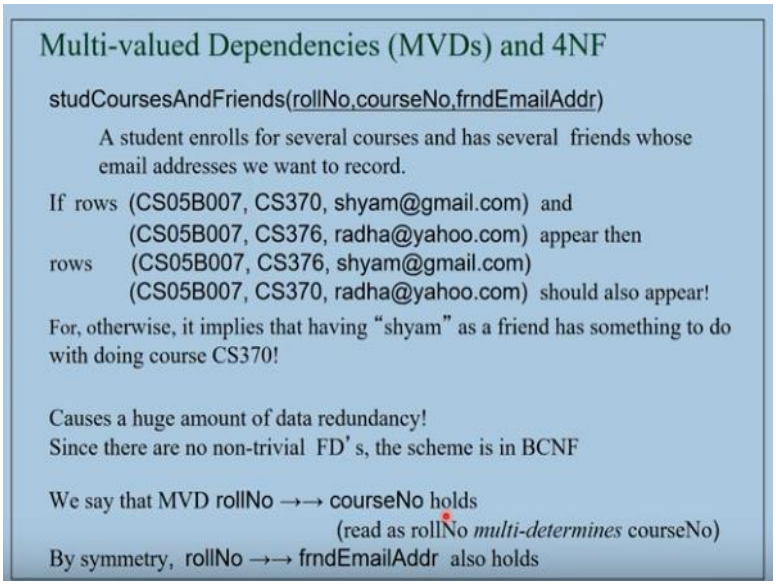
Introduction to Database Systems
Dr. Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology – Madras

Lecture – 24
Normal Forms – 4NF, 5NF

(Refer Slide Time: 00:18)



So in today's class, we are going to complete our discussion on normal forms. So, let me introduce what are called multivalued dependencies, and then define the fourth normal form. Look at this example relation, which is really interesting. For some reason, I have combined students with their friends and courses. For some reason, let us say we have deliberately made up this relation.

So, again is that the student enrolls for several courses and obviously he she will have several friends. So, we want to keep track of this information. So, the friend is represented by friend email address. Okay. Now, obviously the key for this relation is all the three attributes together. Okay, and do there exist any nontrivial function dependences here. Does roll number determine anything course or friend, in general, not because I have multiple courses (()) (02:16) have multiple friends and all that.

So, likewise you can see that course does not determine anything and friend email address also does not determine anything. So, there are no functional dependencies among the three attributes that we have here and since there are no nontrivial functional dependencies, the

relation actually is in Boyce-Codd Normal Form. The highest normal form that we know right now is the Boyce-Codd Normal Form and it is in Boyce-Codd Normal Form because what Boyce-Codd Normal Form says is that for every nontrivial function dependency X determines A there should be a super key.

Now, there are no nontrivial functional dependencies at all. But now look at the interesting consequences of having to store (()) (03:15) this particular relation scheme. Supposing, I have details about a particular student whose roll number is given here, and he is doing two courses. Let us say he has couple of friends, Radha and Shyam. Now, I should not be storing these two tuples that these are all the same students anyway.

This student with this course 370, Shyam, and 376 with Radha. So, this would imply that by doing this course has something to do with having a friend by the name Shyam, and doing this course has something to do with having a friend called Radha. So, we very well know that these two things are independent things. So, if we do not want to convey any such associations, then it is imperative for us to kind of store if at all I have these two tuples, I should actually swap these two values and store the other two tuples also.

Otherwise, I would be conveying some kind of an association between the courses and the friends, which is not really there, in reality. So, in general, you can see that if I have registered for some half-a-dozen courses and I have half-a-dozen friends or more friends, probably having a popular guy, so then we will be having some 60 tuples here. A huge amount of data redundancy exist here.

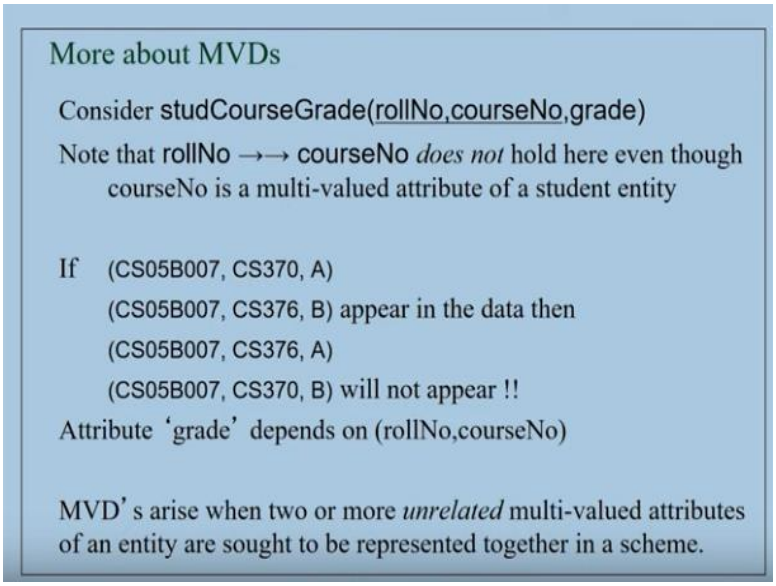
The relation has huge amount of data redundancy, but all the missionary that we have set up so far is not able to detect that, because there is no functional dependencies. Okay. This is the kind of situation that is detected by what are called multivalued dependencies. So, before we formally define them, let me show you some more examples. Actually, I will define them in formally like this now, that whenever we have two tuples of this kind, we will force to have the other two tuples where I swap the values of these, third-column values.

So, by swapping, I delete Shyam here and put Radha, and then that tuple should be there. So CS370 Radha is there. In a similar way, I take out Radha here and then put Shyam here, and CS376 Shyam should be there. That is what I am swapping basically. So, in this kind of

peculiar situation, and you can kind of guess that this kind of situation has come up because these are two independent multivalued attributes of the student and we have sort to put them together into one relation.

This kind of messy situation has arisen. For notational, we will say that multivalued dependency, roll number multi-determine scores exists and so by symmetry, we can also see that roll number, multi-determines friend emails also kind of holds. I will tell a formal definition in a short while.

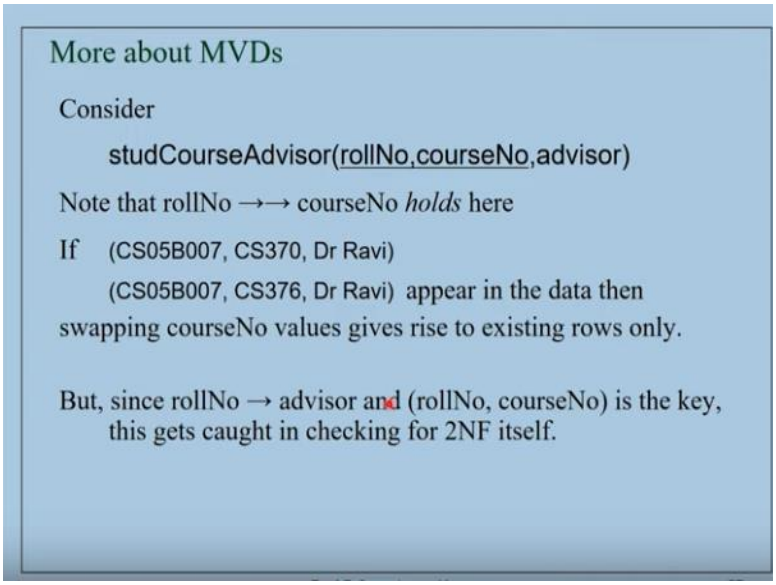
(Refer Slide Time: 07:20)



Now in this, let us look at this slightly different relation scheme. Student course grade. I have roll number, course number, but then grade. Actually, the grade depends on roll number and course number, right. So even though the course number is, in fact; a multivalued attribute of roll number. Just because having a multivalued attribute, it does not mean that we have a multivalued dependency. I want to illustrate that.

So, in this case, let us say that the same student, so you have CS370 and grade A, and CS376 and grade B, and if these two things appear, it does not mean that the swapped tuple should appear, because they are actually dependent, right. So, this grade in fact dependent on roll number and course number. So, it is not the single multivalued attribute that is causing tuple, it is the existence of two independent multivalued attributes being put together into relation, that is what is causing tuples. So, MVD's arise when two or more unrelated multivalued attributes of an entity are being represented in a particular relation.

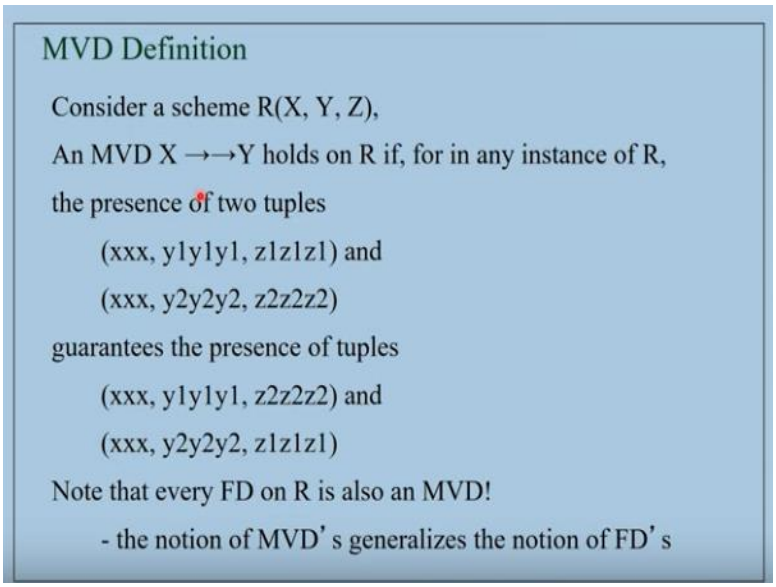
(Refer Slide Time: 08:49)



Now, there is another example. In this case, the roll number, course number, and advisor. Suppose, again we consider these two cases, if you swap the third-column values here, it represents the same set of tuples anyway. So, swapping either of them course number or this one, will result in the same existing rows only. So, if the definition is asking for swapped rows should exist, in fact, they do exist.

So, in this case it actually holds, but then this kind of situation, would have been caught early in our normalization process because roll number determines advisor, and so it is actually a partial dependency here and this would not have come here, a situation where we have to analyze for multivalued dependencies. Okay.

(Refer Slide Time: 10:04)



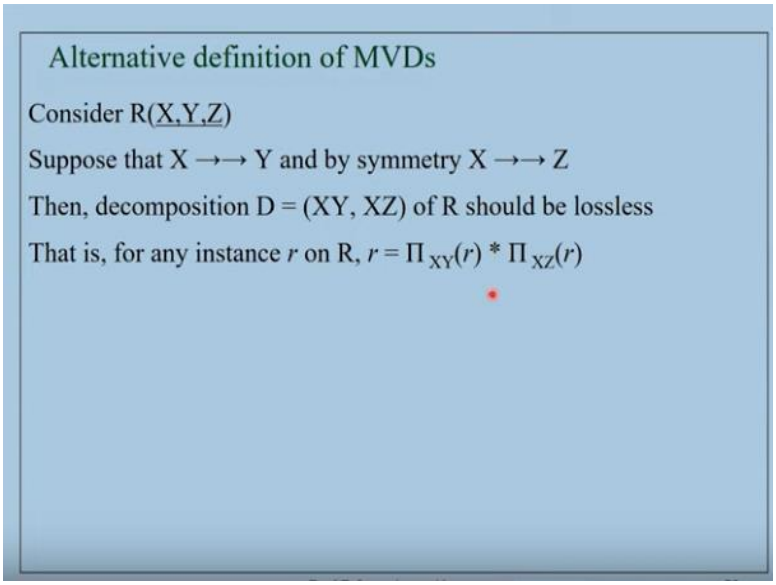
So, here is a formal definition for multivalued dependencies. So, consider a relation scheme R X, Y, Z. So, MVD X multi-determines Y holds on R, is set to hold on R if, for any instance of R, the presence of these two tuples, having xxx, y1y1y1, like this, that means these two values are same. So, these are two are not necessarily the same. So, the presence of these two tuples guarantees the presence of the other tuples, where how do you obtain these tuples. We basically swap the values of these two tuples.

These two values here. So, take out this y1y1y1, and then put it here. So, you get xxx, y1y1y1, and z2z2z2, and in a similar way, you take out this value and put it here, you get this. So, swap the two. If the presence of these two tuples always guarantees the presence of these two tuples, then we say that MVD X determines Y holds on that relation and there is a symmetry in this definition. So, if X determines and Y holds, then X multi-determines Z also holds actually.

Now, notice one more thing here that, supposing there is a FD, okay. In fact, FD, functional dependency can even be thought of multivalued dependency. So, I claiming that the multivalued dependency is actually generalizes the notion of functional dependencies. Supposing, there was actually just a normal functional dependency, X determines Y. If X determines Y actually holds, then you have two values here, these two values should have actually seen.

So, this also will be y1y1y1, okay, and the swapping of these two will result in the same tuples anyway. Okay. So, the definition kind of holds even though if you think of it as FD. So MVD actually further generalizes the notion of functional dependencies. So, in the functional dependency on the right hand side, you had single value, but now, you actually have in some sense a set up values. So, okay that is the definition.

(Refer Slide Time: 13:17)

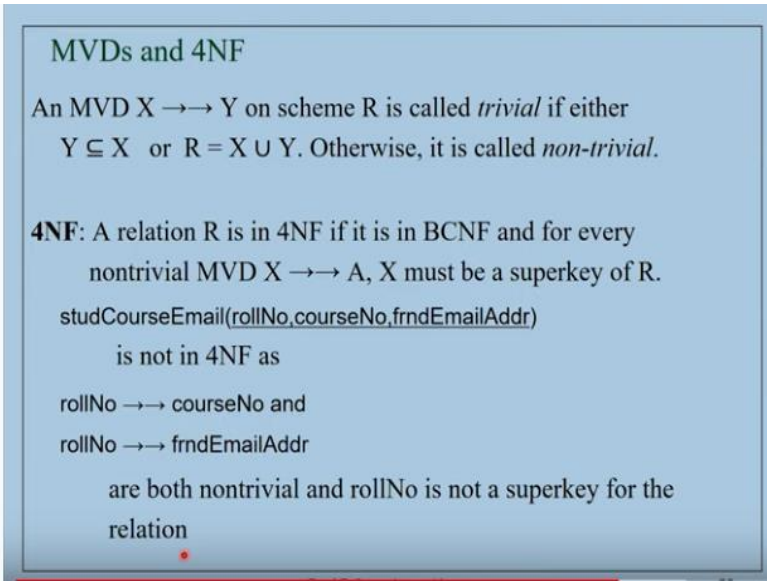


So, here is an alternate definition of the multivalued dependencies in terms of joins. Okay, let us say again, consider some relation like this, X, Y, Z. Suppose X multi-determines Y, then by symmetry, we know that X multi-determines Z. Now, you can actually say that the X multi-determines Y exists in terms of this de-composition actually. Okay, if you consider this decomposition XY, XZ, of R.

So, we can now say that X multi-determines Y, if for any instance R, r is actually equal to the projection of the original relation onto XY, naturally join with the projection of relation onto XZ will be equal to r. So, if you do a projection here, for example, the roll number, course number, friend email address, that example. If you do a projection here on roll number and course number, you will get for every student, all the courses he or she is doing and here you will get every student all the friends he or she has.

And the key here will be XZ and the key here will be XY. Now, do a natural join, the only common attribute is roll number, and so every course should get combined with every friend and you have half-a-dozen courses and 10 friends, you will get 60 tuples. So, that is what we are seeing. That is what the definition also said and we are putting it in another way saying that if we project the relation onto XY and XZ, do a natural join, it should be equal to r. So, here is the alternate way of defining existence of a multivalued dependency. Okay.

(Refer Slide Time: 15:42)



So, now we are ready to define the fourth normal form. Before that, we will also define when you call MVD as a trivial MVD. MVD X multi-determines Y on scheme R is called trivial, if Y on the right hand side is a subset of the left hand side or the union of this X and Y is actually equal to whole of R. Otherwise, it is called non-trivial. Now, for fourth normal form, you remember look at this numbering.

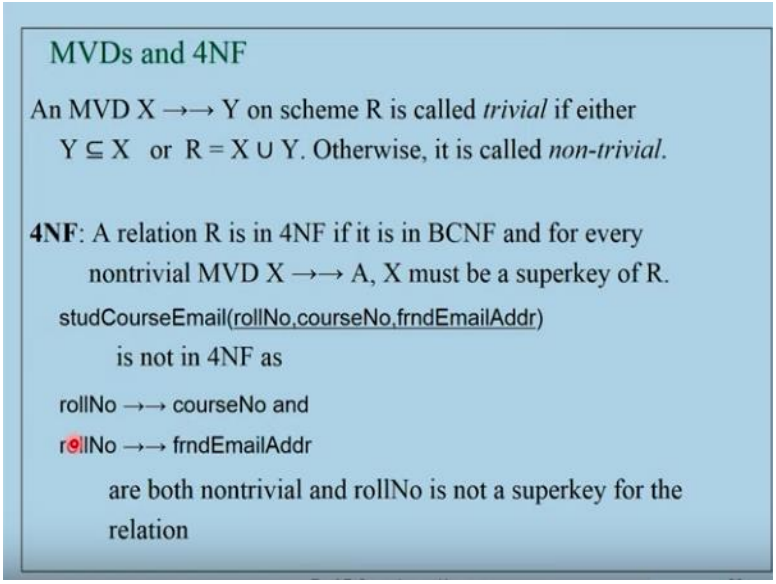
First normal form is already there in the relation definition itself. Second normal form we defined. Third normal form we have defined where we gave the concession for primary attributes and then naturally BCNF should have been called fourth normal form if your vehicle is following the sequence number, but then actually the fourth normal form is defined in terms of multivalued dependencies.

Later, we discovered that we need something stricter than third normal form, which is possible, so Boyce-Codd Normal Form came in between subsets. So, that is why the numbering is different. Okay, fourth normal form, a relation scheme is in fourth normal form if it is in Boyce-Codd Normal Form and for every nontrivial MVD now, X multi-determines A, X must be a super key of R.

Now, this student course of friend e-mail address, that relation is not in fourth normal form, because roll number multi-determines course, and roll number is not a super key, the key is the entire set of all the three attributes. Together is the key. So, roll number is not a super key and the union of these two things is not entire relation and so it is a nontrivial MVD, and so this does not satisfy our definition that the left hand side should be a super key.

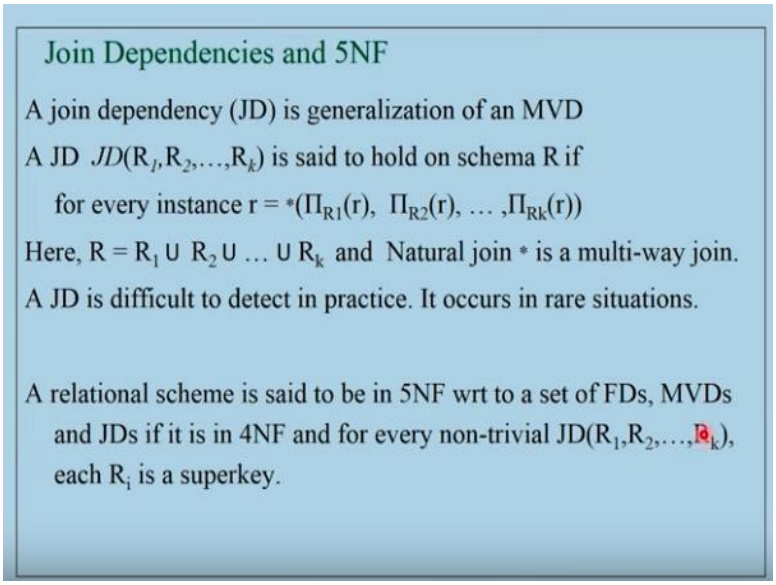
So, what we now suggest of course is to decompose (()) (18:12) into roll number, course number, and then roll number. Of course, what I am trying to say here is interesting. Given to you, you know, (()) (18:22) we will never naturally design this kind of a relation, but if this relation came into the picture inadvertently into your design, then, you know, the missionary up to using functional dependencies cannot detect any problem with this that is why there is a canonical need for having multivalued dependency and so we define that and then say that such kind of things are not satisfying fourth normal form and they should be decomposed.

(Refer Slide Time: 19:00)



Okay. So, let us move on. We have considered fourth normal form and multivalued dependencies and fourth normal form so far. Now, a natural question that comes up as a follow-up of the definition of fourth normal form is that, recall, the fourth normal form, you know, is defined saying that something is equal to the natural join of two relations. Okay. So, now naturally that a question that comes up is why two, why not more.

(Refer Slide Time: 19:46)



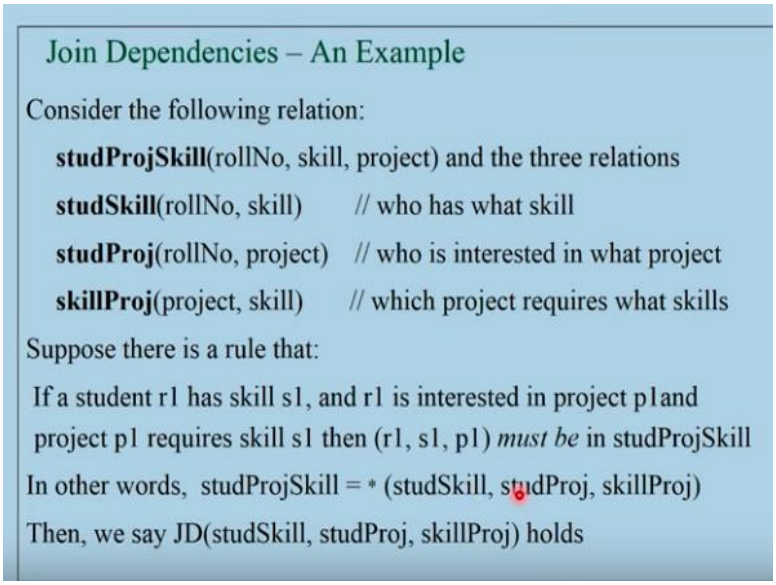
So, it is in this context we define what is called the join dependency and the corresponding normal form called the fifth normal form. So, join dependency is defined like this. This is the generalization of the multivalued dependency and it is written like this JD R1, R2, Rk where R1, R2, and Rk are all individual schemes. So, JD R1, R2, Rk is set to hold on schema R if for every instance of that particular relation r, where this instance r is on the schema R, which is the union of this R1, R2, and Rk.

So, for every instance of r, it so happens that if you project r onto R1, R2, Rk, and then do a multi-way natural join, you get back your r. So, it is written as r is equal to multi-way join of pi R1r, pi R2r, etc pi RKr and please read this R1 as R subscript 1. It is same as the R subscript 1, R subscript 2 like that. Okay. This kind of a condition is indeed a very difficult in practice to kind of find out that it is occurring in the data.

So, in some rare situations, this kind of a condition might hold and in such situations, we will define the normal form. So, a relation scheme is said to be in the fifth normal form, with respect to a set of functional dependencies, multivalued dependencies and join dependencies, if it is already in fourth normal form.

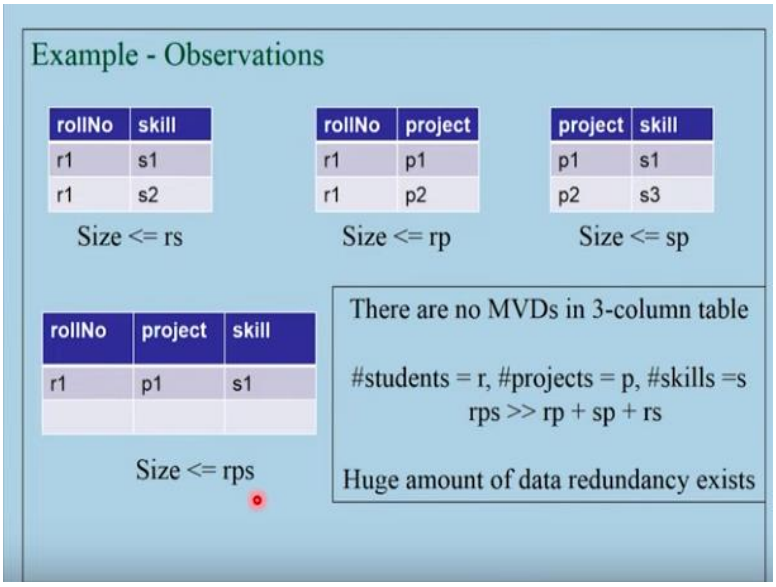
We will define the normal form and for every non-trivial JD R1, R2, Rk, each of the Ri is a superkey. Okay. So that is how we define the fifth normal form with respect to a bunch of functional dependencies, MVDs, and JDs, if the relation scheme is already in fourth normal form and for every non-trivial JD, Ri is a super key.

(Refer Slide Time: 22:34)



Now, let me give you an example. So, consider this particular example situation. So we have a relation called student project skill with three attributes, roll number, skill, and project, and then three relations called student skill with roll number and skill, which basically records as to who has what skill, and then student project indicating that who is interested in what project. So, roll number and some project id kind of thing, and then finally one more thing we have skill project, which basically records that what project requires what kind of skill. Now, these are the three individual relations that result that can be considered within this context.

(Refer Slide Time: 24:39)



Now, let us say there is a particular rule or situation that prevails, which says something like this that if a student r1 has a skill s1, and the student r1 is indeed interested in project p1, and project requires this skill s1, then it should be the case that r1, s1, p1 must appear in our student project skill relation. Okay. So this is the prevailing condition there. So, in such

situation, you can actually see that this student project skill relation is the multi-way natural join of these projections called student skill, student project, and skill project.

So, in such situations we say that this join dependency holds. Now, if you look at actual data, like for example this roll number, skill, roll number, project, and project skill individual relations, let us say there are some r number of students, and p number of projects, and s number of distinct skills, and we also noticed that in this particular three-column table, there are actually know multivalued dependencies.

Okay, if you focus on this, the maximum size of this can be utmost r times s. So, every student may not have every skill. So, typically it might be much smaller than r times s, but the maximum is rs. In this similar way, every student may not be interested in every project, and so this will be less. Similarly, all projects need not require all skills, so project skill table will be having some size that is less than s times p.

But, then if you focus on this for the appropriate number of this chosen appropriately rps, you can see that the size is upper bounded by rps, but then rps is typically much larger compared to rp + sp + rs and because of this requirement that if a student has a particular skill, and is interested in a particular project and that project requires that skill, then that combination must exist here, r1p1s1 must exist here.

That is the condition under which we are operating. It is highly likely that this will have a huge number of redundant tuples. Okay. There is a huge amount of data redundancy here, and so we say that the join dependency exists here, and we are advised to avoid this kind of design. Okay. So, this join dependencies are very difficult to detect in practice. It might actually not occur. So most of the time, we will focus on obtaining designs that are into fourth normal form.

(Refer Slide Time: 27:28)

So, moving on to kind of close this module on discussion about normal forms, we consider two kind of approaches, for the overall relational database design. These approaches are Bottom-up design and Top-down design. So, let us see what they are and the theory that we discussed is actually applicable for both this kind of situations. So, this bottom-up approach is what is called as Synthesis Approach.

So, in this Synthesis Approach, what we do is we identify all the attributes of interest in the domain that we are modeling or in the database that we are modeling and keep all these attributes in a big universal relation. Then by doing a careful study of the domain, we have to determine all the functional dependencies, all the multivalued dependencies, if there are there JDs etc, all of them we need to determine and write down.

Then, we can use the algorithms that are discussed in this particular module in order to decompose this universal relation into those individual relations that satisfy this highest normal form, may be fourth normal form etc. and then actually obtain a relational database design. The downside of this approach is that it is kind of very difficult to obtain all the functional dependencies in a very large database with 100s of attributes and as you would have noticed while we are discussing algorithms, the algorithms are actually nondeterministic.

So that means there are certain choices in the algorithms, which can be done in multiple ways and so may result in multiple designs. So, this approach is not really that popular in practice, so in practice what we use is what is called the top-down approach or the analysis approach

where we basically do a modeling based on certain conceptual level data models like the entity relationship model that we have discussed in one of our course and then generate a relational database design by representing these entities, relationships as individual relations, which basically group these attributes that are belonging naturally to each other.

On those smaller relations, we determine the functional dependencies, MVDs, that are applicable among those attributes and then carry out an analysis of the relations individually and also sometimes collectively to ensure that they are meeting the normal forms requirements, and if necessary we will carry out de-compensation to obtain desirable properties.

So, this approach is much more popular and of course, the theoretical observations that we made about the normal forms, and then the dependencies that exist and things like that. They are kind of applicable in both approaches. So, it all depends on how actually we make use of them in practice. So, with these observations, let us now close this particular module, which is a module on relational design normal forms. Okay. Thank you.