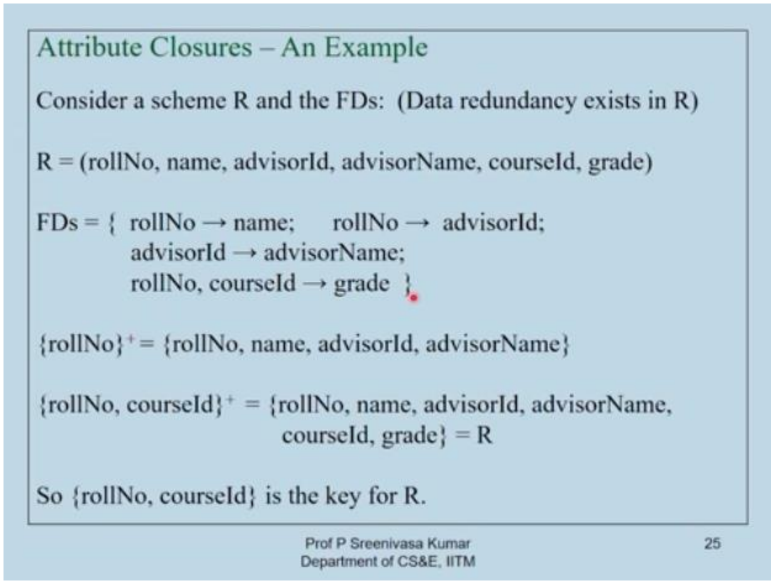
Database Systems
Prof. Sreenivasa Kumar
Department of Computer Science and Engineering
Indian Institute of Technology – Madras

Lecture 22
Normal forms - 2NF, 3NF, BCNF

So towards the end of last class, we were also looking at how to compute the attribute closure, right. So I think I just looked at some examples, but we were in a very hurry to locate the examples. So I will now let you see the examples once more.

(Refer Slide Time: 00:36)



Attribute Closures – An Example

Consider a scheme R and the FDs: (Data redundancy exists in R)

R = (rollNo, name, advisorId, advisorName, courseId, grade)

FDs = { rollNo → name; rollNo → advisorId;
advisorId → advisorName;
rollNo, courseId → grade }

$\{rollNo\}^+ = \{rollNo, name, advisorId, advisorName\}$

$\{rollNo, courseId\}^+ = \{rollNo, name, advisorId, advisorName, courseId, grade\} = R$

So {rollNo, courseId} is the key for R.

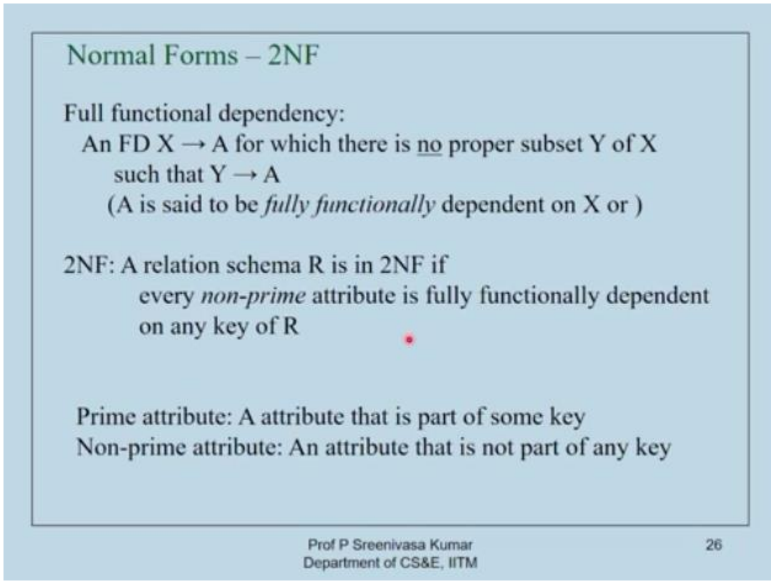
Prof P Sreenivasa Kumar
Department of CS&E, IITM 25

So given some R set of attributes or the scheme and the functional dependencies like this, we can calculate the attribute closure by given attribute set, then you include that into the sets, successive sets. We will include that first and then since this is the left hand sides of some FD, then we will start including the right hand sides into the set and now that advisor ID has come, we will include advisor name and it stops there. You can see that.

Since course ID is not there, you cannot use this FD and then get grade into this. So roll number in particular is not a key for this scheme. It is kind of obvious now, because grade does not, it does not determine grade. The key determines all the attributes. So now, if you take roll number course ID, so obviously since your roll number is there, all these attributes will come into picture and since course ID is also there, you can now use this and then so you get this.

So that will become equal to R and we have, so this is one way of getting hold of the keys of a relation, is actually to focus on those attribute sets whose closure is the same R, given that you know the functional dependence.

(Refer Slide Time: 02:31)



Normal Forms – 2NF

Full functional dependency:
An FD X → A for which there is no proper subset Y of X such that Y → A
(A is said to be *fully functionally* dependent on X or)

2NF: A relation schema R is in 2NF if
every *non-prime* attribute is fully functionally dependent on any key of R

Prime attribute: A attribute that is part of some key
Non-prime attribute: An attribute that is not part of any key

Prof P Sreenivasa Kumar
Department of CS&E, IITM 26

So let us now in this lecture, actually want to talk to you about normal forms, okay. So at the beginning of this module, I told you that normal forms are basically certain kind of constraints and in order to, you know express these constraints, we need some theoretical tools and that is why we started looking at functional dependencies. The definition of functional dependencies and all that and then how to derive new functional dependencies, all that we have looked at.

And the idea behind normal forms is basically that they give us in terms of these functional dependencies. We can express certain constraints and if the schema satisfies those constraints, then certain kind of problems are not present with that scheme, okay. So we will progressively look at more and more restrictive normal forms and as we look at normal forms, we will see what kind of problems it avoids in each of these normal forms and then see that what problems all persists and how do we handle them.

So that is how we will present these normal forms, okay. So in general, we will assume that there is a scheme. There is a relation scheme and a set of functional dependencies, all functional dependencies on that particular scheme are known and that is there in the background. So we will represent that as F. Now in order to define second normal form, remember the first normal form is nothing but the assumption that all the values and all the cells of the table are atomic pieces of data.

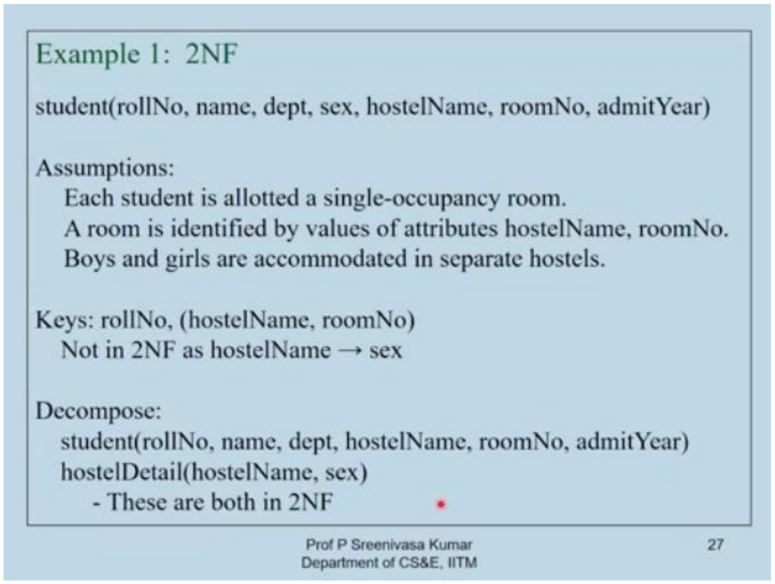
So that is built into the definition of relation database itself. So that is why we do not discuss the first normal form. All relations are in the first normal form by definition. Now in order to define second normal form, we need what is called full functional dependency. So there is an FDX determines A for which there is no proper subset Y of that X, such that Y itself determines A. Then, we call this as a full functional dependency, okay.

Otherwise, we will say that A is, if this is not the case and there exists a Y, such that Y determines A, then we say that A is partially dependent on X, because A is actually dependent only on Y, which is a proper subset of X. So we define partial dependency or full form appropriately. So this A is said to be fully functionally dependent on X. Now it is the normal form definition says like this.

Goes like this, so a relation scheme R with respect to a set of functional dependencies is said to be in second normal form, if every non-prime attribute is fully functionally dependent on any key of R. This is important. Now what is non-prime. That is defined here. An attribute that is part of some key, a relation may have several keys. All of them are called candidate keys, right. So an attribute that is part of some key is called a prime attribute.

And an attribute is not part of any of the keys is called a non-prime attribute. So for some technical reason, right now, we will give some concession for prime attributes and then impose a condition on non-prime attributes saying that, every non-prime attribute is fully functionally dependent on any key of R, okay. So then, you call a relation to be in second normal form. Okay, now let us look at some examples and then see if the second normal form is actually violated, what kind of data redundancy exist and how do we get rid of that.

(Refer Slide Time: 07:24)



Example 1: 2NF

student(rollNo, name, dept, sex, hostelName, roomNo, admitYear)

Assumptions:
Each student is allotted a single-occupancy room.
A room is identified by values of attributes hostelName, roomNo.
Boys and girls are accommodated in separate hostels.

Keys: rollNo, (hostelName, roomNo)
Not in 2NF as hostelName → sex

Decompose:
student(rollNo, name, dept, hostelName, roomNo, admitYear)
hostelDetail(hostelName, sex)
- These are both in 2NF

Prof P Sreenivasa Kumar
Department of CS&E, IITM 27

I am giving you some example, so that you know there is continuity between what we are discussing, so let us look at this example. I have put in attributes, which are different from the usual student relation that you had earlier, just for example sake, for illustrating points here. So roll number, name, department, sex, hostel name, room number and admit year. Let us say, these are the attributes in some relation for students.

And there are assumptions, which will dictate the functional dependencies that are present. So each student is allotted a single occupancy room, okay and each room is identified by values of these attributes, hostel name and room number and boys and girls are accommodated in separate hostels. These are the assumptions. So with that, we will actually get two keys for this. Roll number is a key, no two rows in this particular table will have the same roll number.

Hostel name, room number also is a key. That is because each student is allotted a single occupancy and because of this policy that boys and girls are accommodated in separate hostels, there is an FD called hostel name determine sex. That we have also seen earlier. So given the hostel name, we can uniquely identify if that is a boy or girl. Now this functional dependency actually is such that, this sex attribute does not fully functionally dependent on the key.

The key is hostel name, room number. So on this key, you can see that there is only part of the key determines the set. So this relation is not in second normal form and in general if we see some, you know, relation and it is not in a particular normal form, one technique that we will use is to break that relation. We call it decomposition. It decomposes that relation and then bring in two relations into picture, two or more impact.

So in this case, since this attribute sex is causing trouble for us, we will actually isolate that. We will isolate that. So we will bring in new relation, okay, called hostel detail, where we put this hostel name sex, okay. Now this particular FD is now confined into this particular relation. Hostel name determines sex and hostel name is obviously key for that and so this FD is now a normal FD. There is no troublesome FD. That is not a partial dependency or anything.

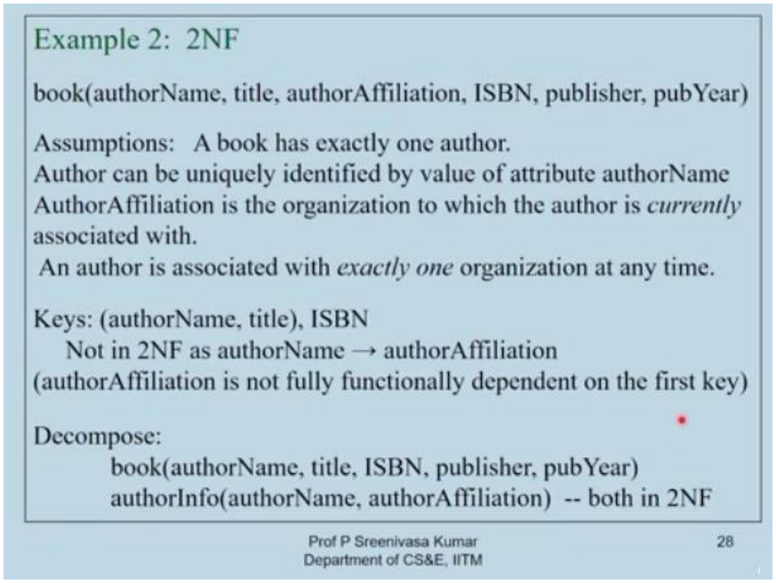
The key here is hostel name. Now here, both keys are there, hostel name, room number and roll number are keys, but there are no functional dependencies, which are causing problems for us. There are no non-prime attributes. What are the prime attributes here? Roll number, hostel name, room number, these are all prime attributes. The non-prime attributes are name, department and admit year. So name is fully functionally dependent on both roll number as well as hostel name.

And so is the case with department and admit year, okay. So now, this relation is actually in second normal form. This relation is not in second normal form and we have done a decomposition and then ensured that the trouble making functional dependency is actually in a new relation and with that we have been able to get two relations that have the same information and both of them are second normal form, okay.

Now, actually there is only a small amount of redundancy of data here, that we just had one, you know, probably one bite of information extra, you know, but that is not, we do not need to bother about how much and all, but the theory tells us that there is redundancy, because you are going to, you know, keep this general information with all the students, even though it actually can be figured out from the hostel name.

But in general, it can be lot more information, so this is only an example. So that kind of redundancy is what is being detected by this second normal form in a relation and we are now saying that we can avoid that redundancy by decomposing this relation into two relations. To given another example:

(Refer Slide Time: 13:30)



Example 2: 2NF

book(authorName, title, authorAffiliation, ISBN, publisher, pubYear)

Assumptions: A book has exactly one author.
Author can be uniquely identified by value of attribute authorName
AuthorAffiliation is the organization to which the author is *currently* associated with.
An author is associated with *exactly one* organization at any time.

Keys: (authorName, title), ISBN
Not in 2NF as authorName $\rightarrow$ authorAffiliation
(authorAffiliation is not fully functionally dependent on the first key)

Decompose:
book(authorName, title, ISBN, publisher, pubYear)
authorInfo(authorName, authorAffiliation) -- both in 2NF

Prof P Sreenivasa Kumar
Department of CS&E, IITM 28

Look at this. We have information about books, author name, title, author affiliation. Author affiliation is an institution to which the author is currently associated with. Then, this ISBN is the International Standard Book Number, right. So this is another attribute of books and it is a key actually and then publisher and publishing year, okay. So we have made some simplifying assumptions here that the book has exactly one author.

And author can actually be uniquely identified by the value of the author name. We could actually replace it by some aadhar number or something that you go on, but let us make a simplifying assumption saying that author name can uniquely identify a person, okay. An author is associated with exactly one organization at any point of time. So with these assumptions, what are the keys for this relation?

Obviously, ISBN is a key. ISBN is book identification number. So that is a key. What else is a key? Is author name a key? Is author name a key for this relation? A person, you know, a person might write multiple books and so author name together with the title will be one of the keys actually. So now, you can see exactly where the trouble comes. So author name, title is one key, ISBN is one of the key. So this author affiliation is an attribute that is dependent only on author.

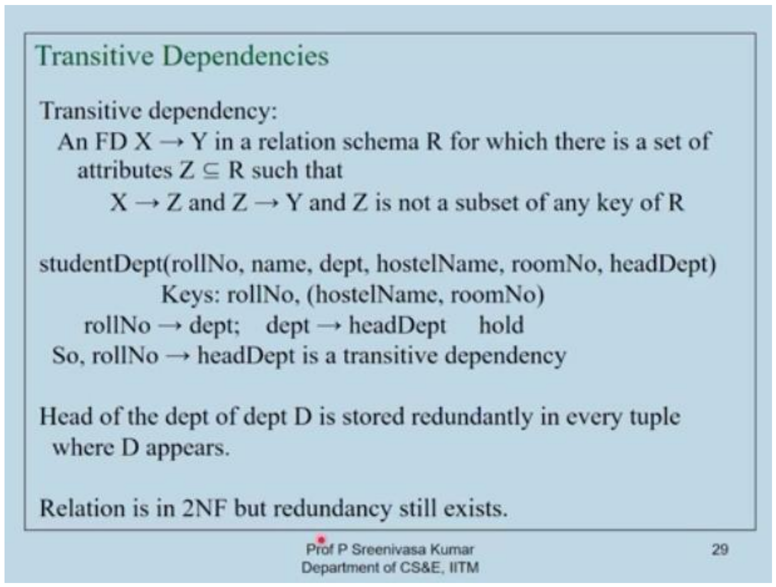
It does not matter, how many books he might write, right. Affiliation is dependent on the author and so this is the attribute that is not fully functionally dependent on the first key and so again, there is a violation of second normal form here and what we say is that we decompose and then get rid of this author affiliation and put it in a new relation. Again, author name, title together and ISBN are keys here.

And then this non-prime attributes publisher and publish year are fully functionally dependent on all these in this relation and this is the original key and that is also functional. So actually, with the question of violating second normal form does not even come into the picture, if you do not have a key that has multiple attributes, right. Unless a key has multiple attributes, there is no scope for this kind of a partial dependence.

So the first thing, you have to check is whether, if you are checking for second normal form, are there any keys that have multiple attributes. If there is no key with multiple attributes, then there is no scope for violating second normal form at all anyway. Okay, so this is what and this also, you know, for all the books that a particular person writes, we were keeping the author affiliation data unnecessarily and so now that has been actually shifted to another relation.

And so that redundancy of data has been avoided. So if a relation scheme violates a normal form, usually there will be redundancy. So in this case, there is redundancy, because that author affiliation is told multiple times, if a particular person writes some 10 books and he is in affiliation with it 10 times. So let us move on.

(Refer Slide Time: 18:30)


Transitive Dependencies

Transitive dependency:
An FD $X \rightarrow Y$ in a relation schema R for which there is a set of attributes $Z \subseteq R$ such that
$X \rightarrow Z$ and $Z \rightarrow Y$ and Z is not a subset of any key of R

studentDept(rollNo, name, dept, hostelName, roomNo, headDept)
Keys: rollNo, (hostelName, roomNo)
rollNo → dept; dept → headDept hold
So, rollNo → headDept is a transitive dependency

Head of the dept of dept D is stored redundantly in every tuple where D appears.

Relation is in 2NF but redundancy still exists.

Prof P Sreenivasa Kumar
Department of CS&E, IITM 29

So in order to talk about further dependencies, further normal forms, we would like to introduce these dependencies called transitive dependencies. Do not confuse this with transitive inference rule. That is different. Here, we call some FDX determines Y in a relation scheme and transitive kind of dependency, if there exists some set of attributes Z, such that Z is not a subset of any key of R. It is a little technical condition imposed on Z that Z is not a subset of any key of R.

So all Z attributes are non-prime attributes. Okay, it may contain some prime attribute, but it is not part of any key, fully, it is not a subset of a key. That is how we are concerned about, such and what happens is, we get this X determines Z and Z determines Y exist in the scheme. So in the scheme X determines Z, Z determines Y exist and so naturally X determines Y also exists, okay. But then, that Z is something which is not a subset of any key of R.

If this kind of three functional dependencies exist in the relation, we call that as a transitive dependency. So here is an example, where again I slightly change the example. I think I have now got in department and head of the department into picture. So student department, we will call it student department, roll number, name, department, hostel name, room number, head of the department. So the keys are roll number and hostel name, room number.

And the same assumptions hold good and each student belongs to exactly one department and so roll number determines department. Each department has exactly one HOD, so department determines head of the department and so here is instance of transitive dependency. Roll number determines department. Department determines head of the department and so roll number determines head of the department, okay. Now is this relation in the second normal form first?

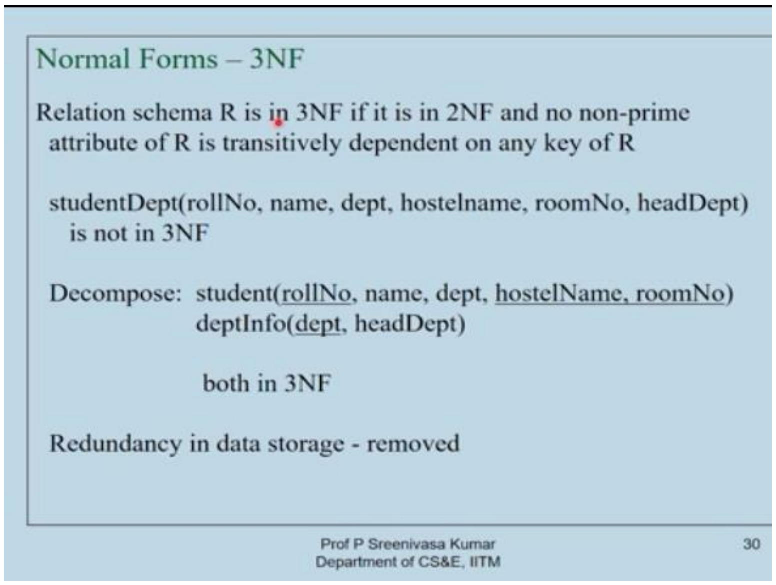
Okay, so let us come to that question a bit later. So right now, where is the redundancy? Where is the redundancy, you can see the redundancy here, right. Since roll number, name, department, hostel name, room number that much is actually okay, but now you are storing head of the department along with every student. So if there are 100 students in Computer Science Department or 500 students in Computer Science Department, with all of them you are also storing the information about the head of the department.

That is where redundancy exists, but can this redundancy be detected using second normal form? Is this relation violating second normal form? Is this scheme violating second normal form? So especially, if you look at this head of the department, it is dependent on department, but you know department is not a key, not part of any key or anything like that. So there is no question of partial, you know, no violation of too enough restriction here, which basically said that no non-prime attribute should be fully functionally dependent on all keys.

That is what taken in our functions. So here, non-prime attributes are name, department and head of the department. They are all, you know, fully functionally dependent on the keys. So even though the relation is in second normal form, so there is redundancy and that redundancy can only be detected because of, you know, if we detect presence of this transitive dependencies and then make a rule about them.

So redundancy still exists and the relation is in second normal form. So to kind of plug this, what we do is now, define third normal form.

(Refer Slide Time: 23:54)


Normal Forms – 3NF

Relation schema R is in 3NF if it is in 2NF and no non-prime attribute of R is transitively dependent on any key of R

studentDept(rollNo, name, dept, hostelname, roomNo, headDept) is not in 3NF

Decompose: student(rollNo, name, dept, hostelName, roomNo)
deptInfo(dept, headDept)

both in 3NF

Redundancy in data storage - removed

Prof P Sreenivasa Kumar
Department of CS&E, IITM 30

So relation scheme R is in third normal form. If it is in second normal form and in addition, no non-prime attribute of R is transitively dependent on any key of R. This is the new constrain that we impose. In addition to second normal form constraint, we now say that no non-prime attribute should be transitively dependent on any of the keys of R. In the previous example, we had head of the department is transitively dependent on the key roll number.

And that is why it is causing problems for this. So this is not in third normal form, because head of the department is dependent on department, department is dependent on roll number and roll number is a key, so roll number determines department, department determines head of the department, that is a transitive dependency and so it is violating the third normal form. So again, the remedy is similar.

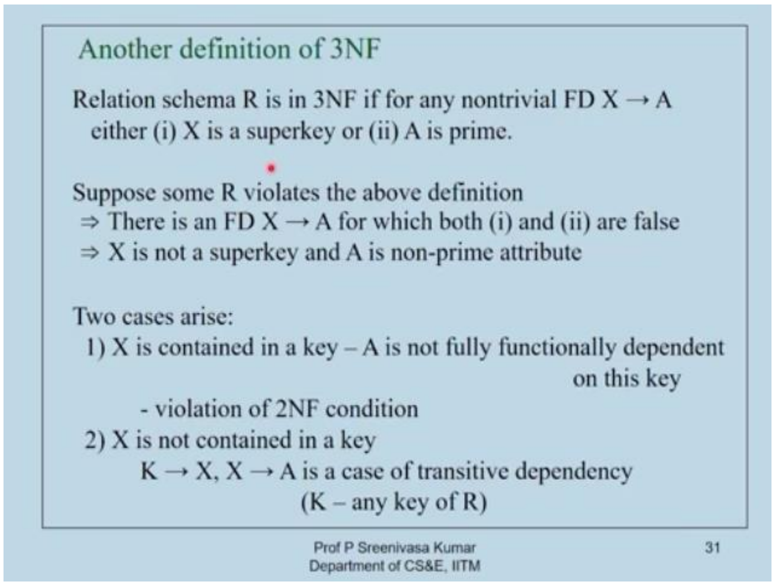
The remedy is that take out the department and head of the department and then put it in a separate relation. So department info is a new relation that we create, put department and head of the department information there and the rest of the information will be given. Now you can see that both of them, so the troublemaking transitive dependency has been removed from each of this. So we will independently look at dependencies, functional dependencies in this relation, student relation and department information.

And there is no transitive dependency now and so both of them again turn out and that redundancy that was existing in the original relation is now removed and so we have a better design. So in general these normal forms give us some kind of guidelines, so that when we have relational design and we have the functional dependencies, we can evaluate are there any issues with our relational schemes and if so, we can improve the design and then replace certain relations by decomposed relations.

But this is not a very, you know, hard and fast rule also actually. Sometimes, depending on the kind of performance considerations, we may like to keep certain attributes like for example, the general attribute in the student. It logically belongs to the student relation with a little bit of redundancy exists. We might, for performance reasons, we might decide not to decompose. So takes these things as guidelines.

But by and large, we will try to go for the strictest normal form that is possible. There is one more normal form that is stricter than 3NF that is called Boyce-Codd normal form. So we will try to go up to Boyce-Codd normal form.

(Refer Slide Time: 28:01)



Another definition of 3NF

Relation schema R is in 3NF if for any nontrivial FD $X \rightarrow A$ either (i) X is a superkey or (ii) A is prime.

Suppose some R violates the above definition
$\Rightarrow$ There is an FD $X \rightarrow A$ for which both (i) and (ii) are false
$\Rightarrow$ X is not a superkey and A is non-prime attribute

Two cases arise:
1) X is contained in a key – A is not fully functionally dependent on this key
- violation of 2NF condition
2) X is not contained in a key
$K \rightarrow X$, $X \rightarrow A$ is a case of transitive dependency
(K – any key of R)

Prof P Sreenivasa Kumar
Department of CS&E, IITM
31

Now before we proceed further case and alternate definition for third normal form. The alternate definition is like this. The relation scheme R is in third normal form if for any nontrivial functional dependency X determines A, either X is a superkey or A is prime. This is a very nice definition and it is somewhat easier to remember also. Now look at this definition. It does not even talk about 2NF. So it combines everything into one nice compact definition.

What it says is that, all that you have to do is to remember this notation about these letters, X determines A. That means, X is a set and A is a single attribute. So for every nontrivial function dependency of this kind, which has left hand side set of attributes and the right hand side a single attribute, either X is a superkey or A is prime. Recall the definition superkey. What is a superkey? A superkey is something which contains a key, fine.

Now let us look at this a little bit careful. Suppose some R violates this definition, some scheme violates this definition, that means there is an FD X determines A, for which both 1 and 2 are false, right. 1 is X is a superkey 2 is A is prime. If one of them is true, then this is true, right. If both of them are false, then it is violating the definition. So suppose R violates this definition, then there is an FD X determines A, for which both 1 and 2 are false.

That means X is not a superkey and A is a non-prime attribute. Now if you observe the situation carefully, then two cases might arise actually. That if you focus on X the left hand side of the functional dependency, either X is contained in a key or X is not contained in a key. These are disjoint cases. So let us look at each of these cases. Suppose X is contained in a key, then it is clearly, so this is a key K and X is contained inside that and X determines A.

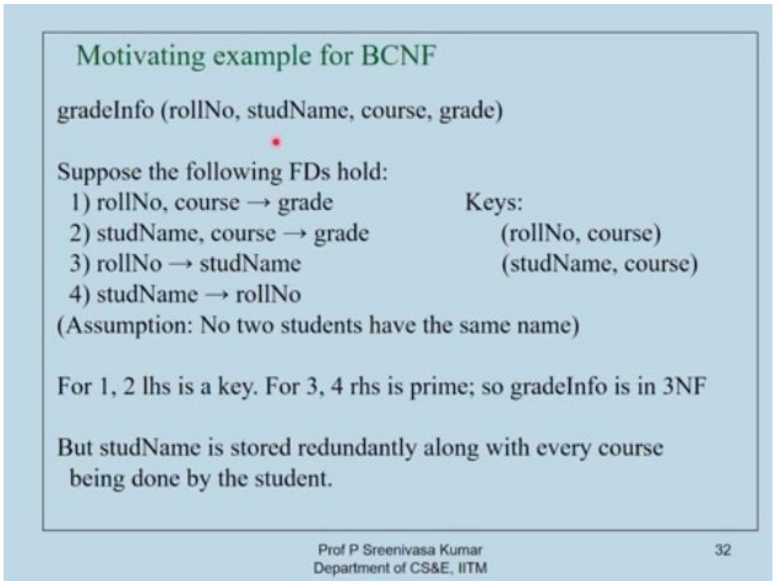
So what does that mean? There is a partial dependency. The A is dependent on X, which is part of a key. So A is not fully functionally dependent on that particular key. So this violates the 2NF condition, okay. So obviously the relation cannot be in 3NF also, because it violates the 2NF itself. So supposing, X is not contained in a key, supposing the other case, X is not contained in a key, then you consider some key K for this relation, obviously K determines X.

Because K determines every subset of attributes, K is a key. So K determines X and X determines A, okay and X is not contained in a key. In the transitive dependency, we exactly put this particular constraint thing that X should not be part of it. So this is an example of a transitive dependency now, K determines X, X determines A. So this is a case of transitive dependency and K can be any key here, because every key determines X and X determines A is given to us.

So basically, this definition in some sense captures the essence of 3NF and it is easier to remember and also the advantage of this definition is that, when you go to a stricter normal form, BCNF normal form, all that we will be actually doing is to simply drop this concession that we are giving for prime attributes. We will simply remove this. That will give the Boyce-Codd normal form.

Boyce-Codd normal form is relation scheme R is said to be in Boyce-Codd normal form, for every nontrivial functional dependency X determines A, X is a superkey. So we will look at Boyce-Codd normal form. Why do we need that?

(Refer Slide Time: 33:42)



Motivating example for BCNF

gradeInfo (rollNo, studName, course, grade)

Suppose the following FDs hold:

1) rollNo, course → grade
2) studName, course → grade
3) rollNo → studName
4) studName → rollNo

Keys:
(rollNo, course)
(studName, course)

(Assumption: No two students have the same name)

For 1, 2 lhs is a key. For 3, 4 rhs is prime; so gradeInfo is in 3NF

But studName is stored redundantly along with every course being done by the student.

Prof P Sreenivasa Kumar
Department of CS&E, IITM
32

Here is a motivating example for that. Let us again, this example has been created to illustrate the point. So I am making a very important assumption here that student names are unique. In this example alone, we are making that assumption, no 2 students have the same name, an important assumption we are making here. So roll number, student name, course, grade, okay. What are the keys? So this relation is obviously for keeping the grade information.

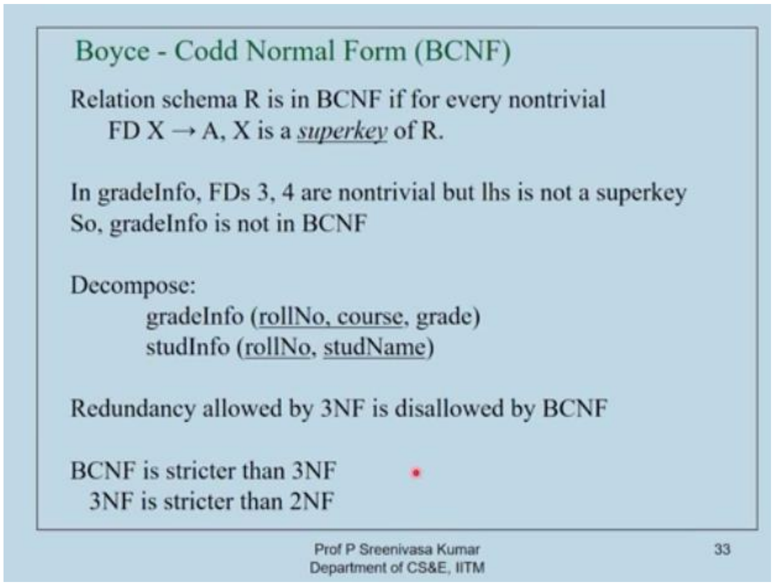
So student whose roll number is this, whose main is this, has done some course and got some grade. That is information in each of the row. So now the keys are obviously, each student does several courses. So the keys are roll number and course ID and student name and course ID. These 2 are the possible keys for the relation. So keys are roll number, course, student name and course and so because these are all keys, all attributes are prime, except grade.

Grade is the only non-prime attribute and look at the functional dependencies, roll number, course determines grade. Student name, course determines grade, roll number determines student name, student name determines roll number, okay. Now apply the third normal form definition for each of these functional dependencies. Everybody qualifies the condition. What does the third normal form condition says that for every nontrivial functional dependency, all of them are nontrivial, X determines A, either X is a superkey, which is the case in this case.

First one, it is a superkey. It itself is a key. This also itself is a key and so either X is a superkey or A is, that is right hand side is called. Now this is not a superkey. This does not even contain a key, but then the right hand side is a prime attribute. So for both of these things that concession we gave for prime attributes will cover them. So all the functional dependencies actually satisfy the third normal form condition. But it is obvious that there is redundancy in this table.

It is obvious that there is redundancy, because we are storing name of the student or whatever it is, one of them is redundant; either you should have roll number or you should have name. You need not have both and we are not able to catch it. This is the third normal form, mainly because we have given some concession from prime attributes. So student name is stored redundantly along with every course being done by the student. So these kind of examples were such kind of things happen, that there are attributes of this kind.

(Refer Slide Time: 37:29)



Boyce - Codd Normal Form (BCNF)

Relation schema R is in BCNF if for every nontrivial FD X → A, X is a *superkey* of R.

In gradeInfo, FDs 3, 4 are nontrivial but lhs is not a superkey
So, gradeInfo is not in BCNF

Decompose:
gradeInfo (rollNo, course, grade)
studInfo (rollNo, studName)

Redundancy allowed by 3NF is disallowed by BCNF

BCNF is stricter than 3NF
3NF is stricter than 2NF

Prof P Sreenivasa Kumar
Department of CS&E, IITM
33

We have a new normal form called the Boyce-Codd normal form. The Boyce-Codd normal form, what it says is that; the definition is this: we just drop the R in the third normal form. Relation scheme R is in Boyce-Codd normal form, if for every nontrivial functional dependency X determines A, X is a superkey of R. So now all those FDs 3 and 4 do violate this rule and so we decompose the relation and then get grade info, roll number, course grade, and student info roll number, student name.

In this, both of them are keys. The roll number is a key. Student name also is a key. These are all assumptions, okay. Now this is in third normal form. This is also in third normal form. So Boyce-Codd normal form further removes redundancies and so we normally try and go up to Boyce-Codd normal form, but one of the questions that might crop up right now, is that why are you boring us with all these definitions?

If Boyce-Codd normal form is what you actually want, why do not you simply define that Boyce-Codd normal form right in the beginning and be done with the class, right. This is the strictest normal form is what you say and we should look for this Boyce-Codd normal form, why do not you first define that and then be done with that. Obviously, second normal form is not good, because it is allowing redundancy.

Third normal form is also not good, it is allowing some more redundancy. Boyce-Codd normal form is not allowing any redundancy, that is good. So we should have that. So actually there is a reason for why we have to still study third normal form. Definitely, we will not stop at second normal form. So depending on the actual situation in the design, we either go up to third normal form and stop there or go all the way upto Boyce-Codd normal form and there are some technical reasons why we may not be able to go to Boyce-Codd normal form in all situations.

So that is the reason why we still study the third normal form. So that, I will do in the next class. In the next couple of lectures, we will introduce certain important properties of these, we have been solving the problems that arise in these relation schemes by doing some decompositions. So what are some properties of this decomposition, that we will look at and then actually from those principles, it becomes clear to us that third normal form is still required, okay.