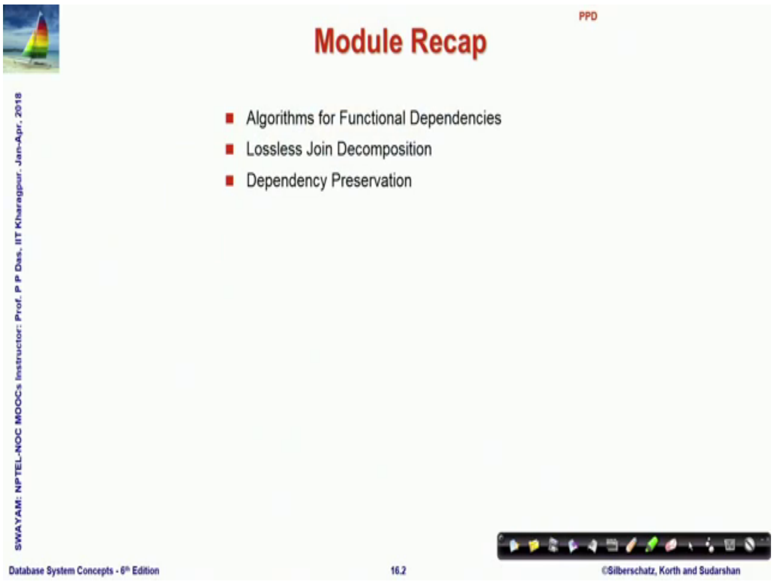
Database Management System
Prof. Partha Pratim Das
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture - 19
Relational Database Design (Contd.)

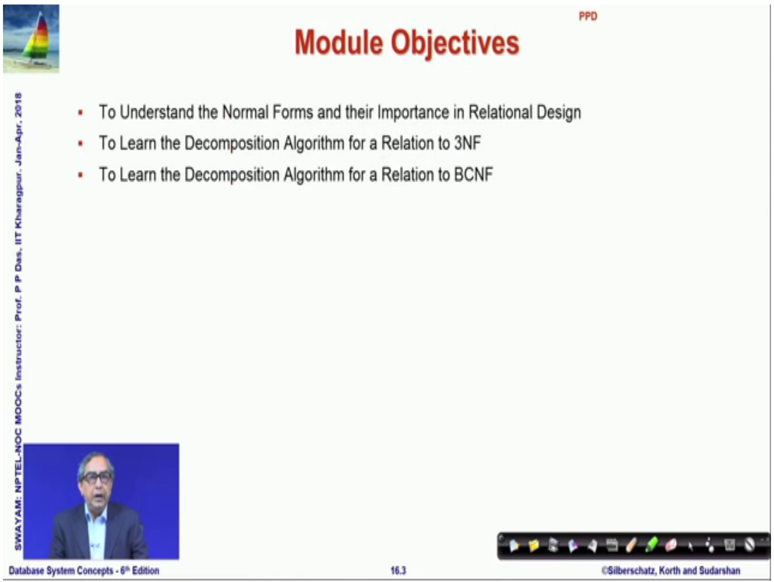
Welcome to module 19 of Database Management Systems; we have been discussing relational database design and this is the fourth part; fourth module in that series.

(Refer Slide Time: 00:38)



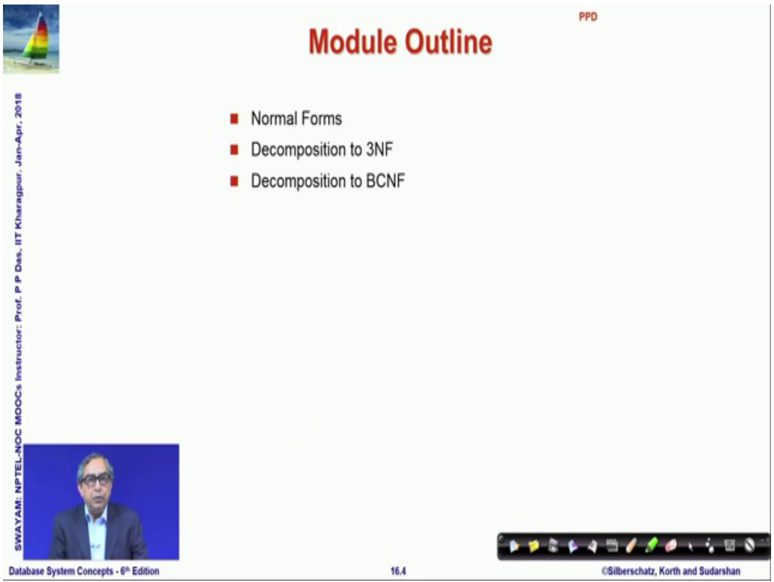
In the last module, we have discussed about algorithms for functional dependencies lossless joint decomposition and dependency preservation. So, based on this foundational algorithms and concepts.

(Refer Slide Time: 00:51)



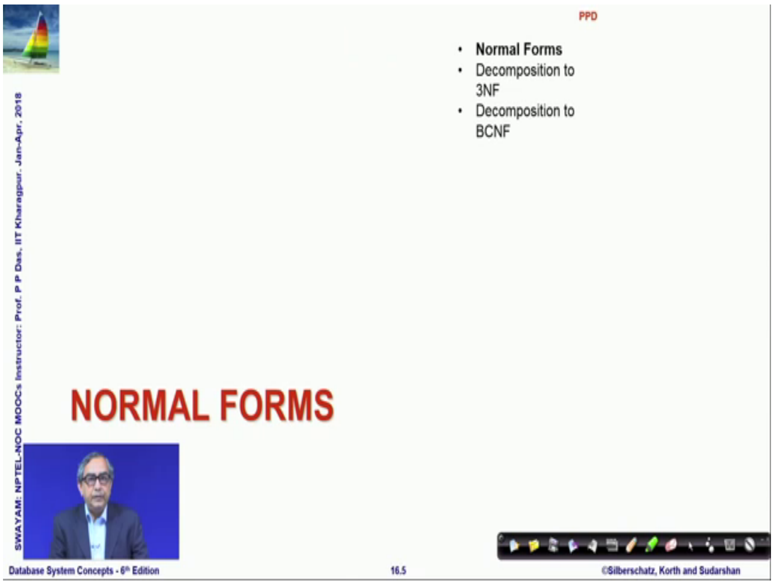
We will in today's module get into understanding the core design aspects of relational databases; that is a normal forms and how important they are in terms of the relational design. We would specifically learn about decomposition of a relational schema into the third normal form and into Boyce Codd BCNF form.

(Refer Slide Time: 01:20)



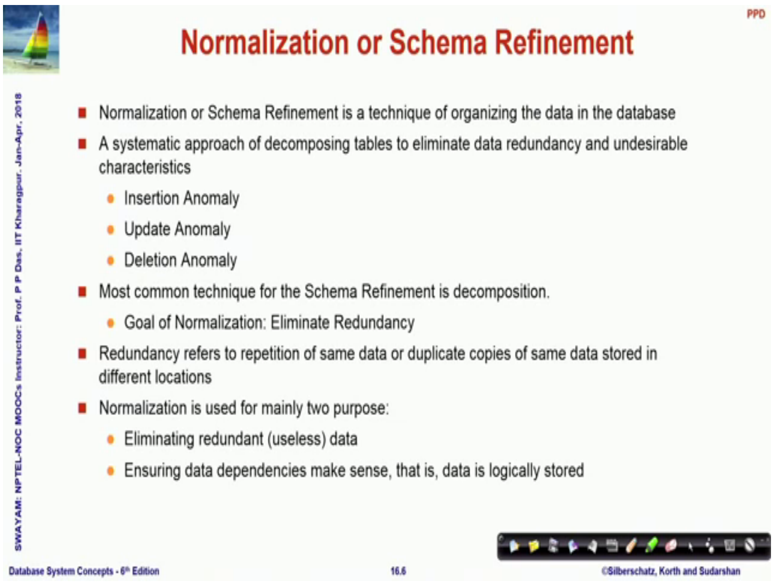
So, our topics will be the three normal forms decomposition of 3 NF and into BCNF.

(Refer Slide Time: 01:27)



So, starting with the normal forms.

(Refer Slide Time: 01:29)

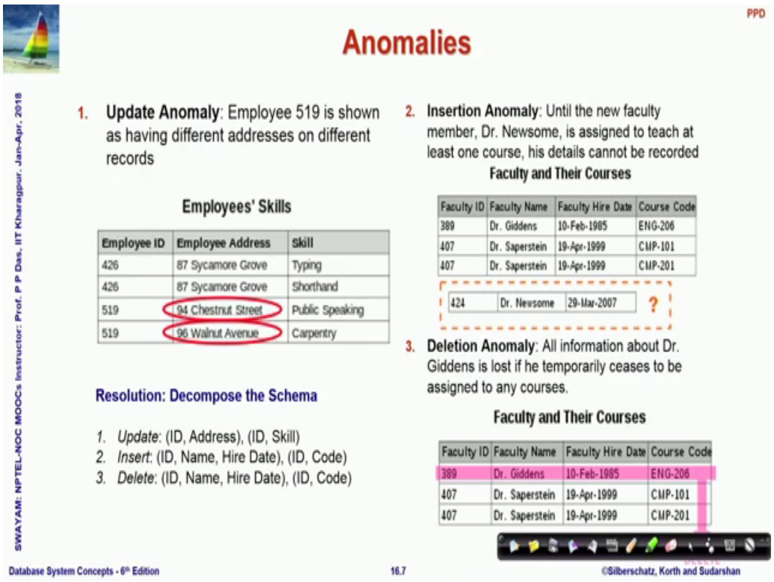


So, normal forms or normalization of a schema is a technique of refinement to organize the data in the database. So, the question naturally arises as to why do we need to do this refinement after we have done a design based on possibly the E-R diagram based approach that we had talked of we had identified the entities and we had identified the attributes for the entities their relationships; then why do we need to normalize?

The answer to this question lies in the fact that a design for a relational schema may give rise to a variety of anomalies in terms of the data. These are typically three anomalies which concerns us most the insertion, the update and the deletion anomaly. So, the anomaly is happen when there is redundancy in the data in terms of the schema. And whether there will be redundant data and how much what kind of redundant data would be there depends on the design of the database schema depends on the design of the normal form that we are using for it.

But if we have redundancy then there is potential for anomalies and therefore, we want to reduce the redundancy and get rid of this anomaly.

(Refer Slide Time: 03:00)



So, we will quickly take a look into the anomalies that are that we are talking of first one is called an update anomaly. So, we are showing you a snapshot of an instance of a database which has three attributes and you can look at the row having two entries the last two rows for employee code 519 and there are two different addresses in these two different rows. So, if we know that the employee will have a unique address or in other words if employee ID would determine the employee address functionally determine that employee address then this situation is not possible.

So, but when we try to update then it is for example, the employees address has changed. And while making that change this change will need to be incorporated in all the records having the same ID. And if because of some coding error or something we miss out to update any of the address fields then we will have a difficulty and that difficulty is having inconsistent address data as in this case..

So, this is known as update anomaly similarly I could have an insertion anomaly which I am illustrating here in terms of another database schema which has four attributes. And we have faculty ID name the hiring date and the course name naturally given the faculty ID the faculty name and hire date should be unique. Now suppose a new faculty joins and as soon as the faculty joins he or she may not have an assigned course.

So if we want to enter that record here we will not be able to do that because we do not have any value for the course code. So, either we use a null value or we cannot actually enter this value; this kind of situation is known as a insertion anomaly. Similarly I could have a deletion anomaly in the in the same table we are showing that in the table the first highlighted row; the for faculty ID 389 if that faculty stops taking any course for the time being..

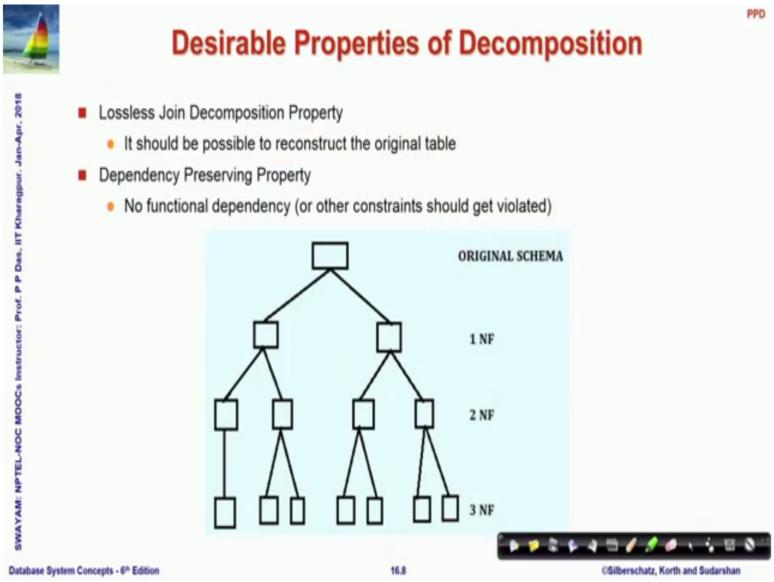
So, the association between 389 and the corresponding course code will be removed and once you remove that you remove this whole record in the process you actually lose the whole of the faculty information the ID, name and hire date. So, these are difficulties in these relational schemas and that lead to a whole lot of problems.

So, the resolution for this lie in terms of decomposing the schema that instead of having one relation, I will decompose this set of attributes into multiple different relations. So, for example, the update anomaly can be removed if we have two different tables; one that maintains ID with address and one that maintains ID with skill. So, in that case what will happen if the for every ID the address will not be repeated..

So, if the address is updated; it will be updated only at one place and it will not feature in the other table. Similarly to avoid insert or delete anomaly the other table schema can be split into ID name and hire date as one table and ID and code rows code as another table. And you can you can easily understand that if this is split in this way then you cannot have an insert anomaly because you can insert a new faculty without assigning a course to him because that will feature in as a separate record in a different table similarly in the same way the deletion anomaly also disappears.

So, these anomalies are resultant of the redundant data that we are having and can be removed by taking care of the process of decomposition.

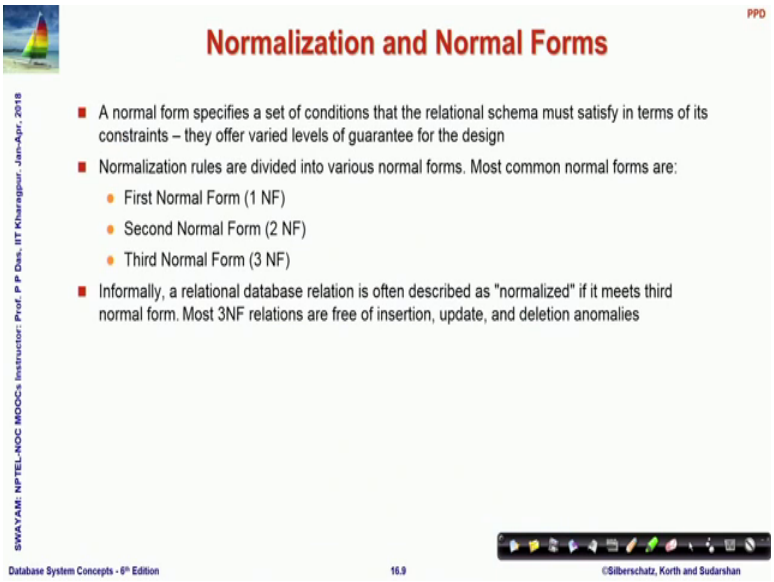
(Refer Slide Time: 07:03)



Now, when we decompose then we would desire certain properties to be hold held and we talked about this loosely earlier as well. We would require the lossless join decomposition property that it should be possible to take any instance of the two or more decomposed relations and join them by natural join using common set of attributes and get back the original instance of the relation if that does not happen then the relationship is lossy we have discussed it at length in the last module. At the same time we would want that all functional dependencies that hold must be; can must be testable in the decomposed set of relation.

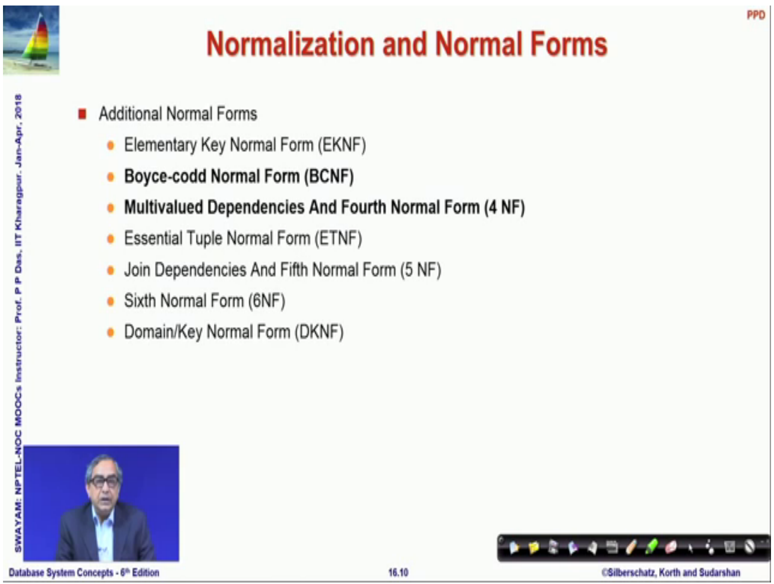
So, all functional dependencies when they are projected in terms of the decomposed set of relations; they must be testable within them. So, that to test for a dependency I do not need to carry out a join this is a point we discussed in the last module as well. So, based on that once you start with the original schema, you can check for what are the different possibilities or sources of redundancy define constraints based on that and step by step; you could convert a schema into a one normal form have more constraints put onto it convert it into two normal form have further constraints decompose it into third normal form and so, on.

(Refer Slide Time: 08:34)



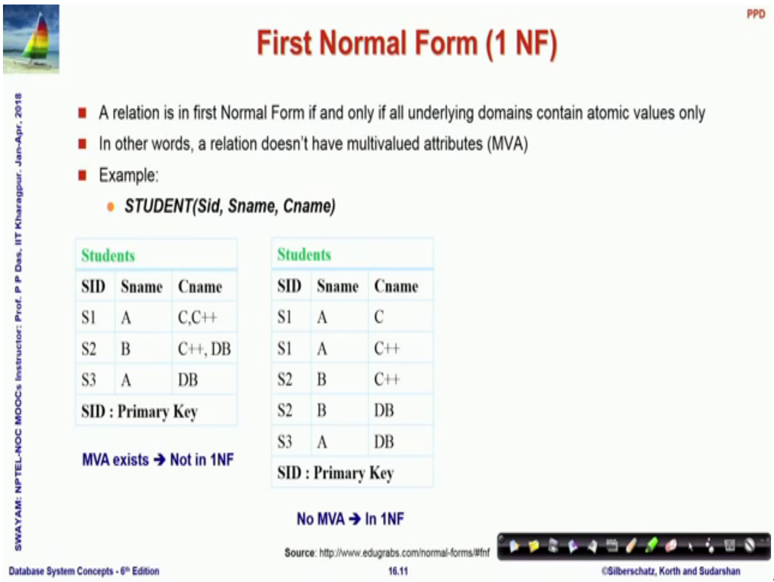
So, normalization is a process through which we do this kind of decomposition and make sure that once a relational schema is expressed in terms of a normal form; it satisfies a given set of properties that that normal form should adhere to. And the common normal forms are 1 NF, 2 NF and 3 NF and loosely speaking when we say if a database schema is normalized; we normal usually mean that it is in the 3 NF form a third normal form. And most third number form relations are free of insert, delete or update anomalies. So, that they are a good positive in the design.

(Refer Slide Time: 09:12)



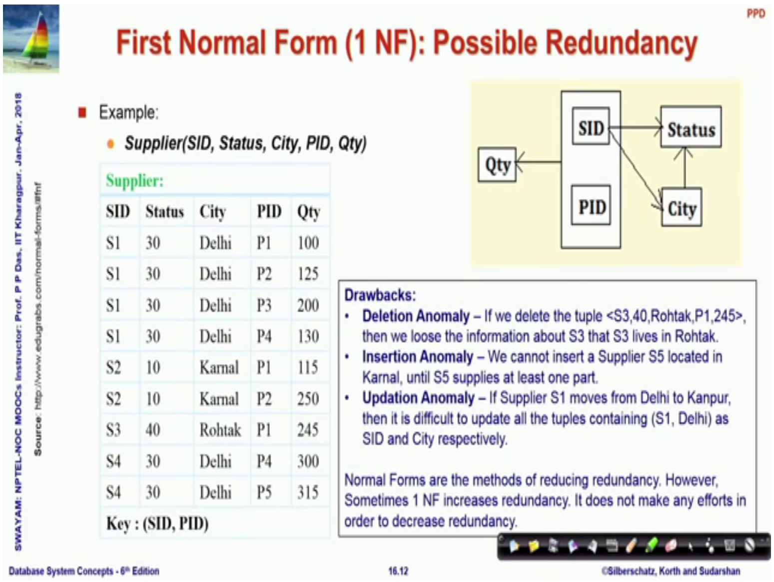
Of course, these are not the only normal forms as you can see there is a whole lot of lists of variety of normal forms; we will not study all of them we will study further in the next module the other two highlighted ones.

(Refer Slide Time: 09:26)



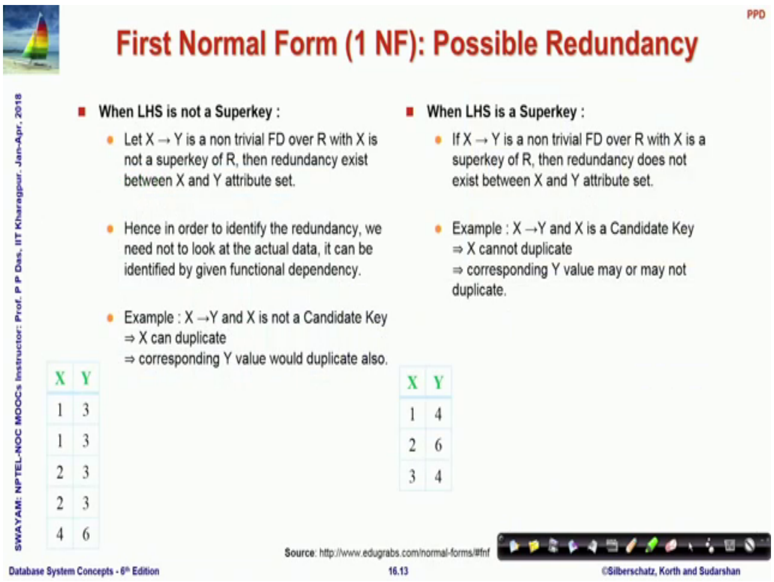
But first let us get started with the first normal form which we had talked about earlier as well; that first normal form is one where the multivalued attributes are not allowed. So, if you think about a think about a relationship where you have a student relationship between student the her name and the courses taken by the student then since the students take multiple courses; the C name in this case can take multiple values. So, we do not allow that we expand them into different rows and that once we have done that we say that relation is in the one normal form.

(Refer Slide Time: 10:02)



But one normal form may give rise to a variety of different redundancies and therefore, anomalies. So, this is another instance; in fact, the earlier instances that you saw all of them were also in one normal form, but they had deletion insertion and update anomaly. So, here is another example where we are illustrating that.

(Refer Slide Time: 10:26)

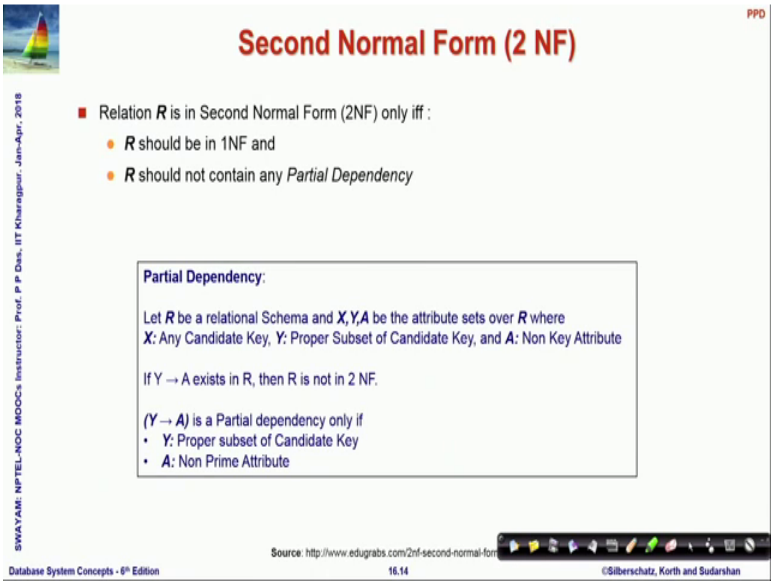


So, it is a possible that if I have a functional dependency X determining Y which is nontrivial functional dependency over the set of attributes and X is not a super key; then there exists a redundancy between X and Y attribute set. So, on the left the we have shown an instance of this relationship on only on the X and Y attributes and you can see since X is not a key; I can have two rows having the value one in X.

And since the value is 1 in X; the value Y will be same for these two rows and we have redundancy of that please all. Please remember that X is not a super key; so, there are other attributes which actually form the super key and therefore, such instances are possible.

Whereas if you look at the right column where the left hand side X is a super key then such instances will not happen.

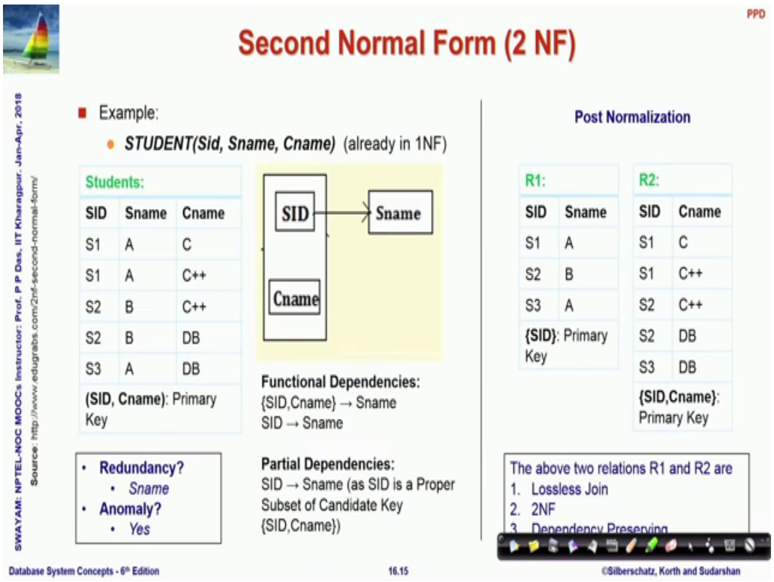
(Refer Slide Time: 11:22)



Moving on the second normal form which is obviously, a relation is in second normal form if it is in first normal form and it does not have any partial dependency. So, what is the partial dependency? I have given the definition here partial dependency why determining A if that that can hold in the set of functional dependency then if I have that Y is a proper subset of a candidate key and A is a nonprime attribute in nonprime attribute is one which one nonprime attribute we defined in the last module is an attribute which does not feature in any of the candidate keys.

So, if Y is a proper subset of a candidate key which functionally determines a nonprime attribute; then this is known as a partial dependency and if there is partial dependency then the relationship is not in second normal form. So, second normal form will require that the relation is in 1 NF and there is no partial dependency.

(Refer Slide Time: 12:25)



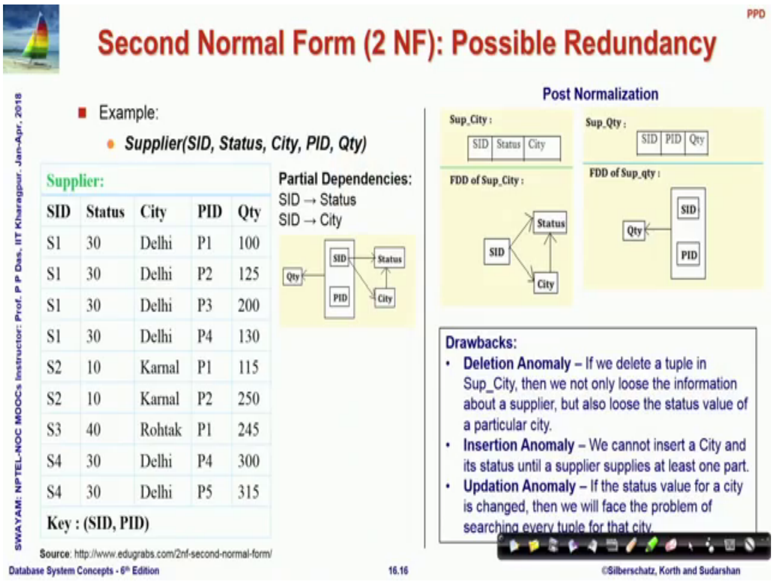
So, here I were showing an example where on the left you can see that SID and C name together forms a key and SID determines S name. So, SID and C name together also determines S name naturally SID determining S name is a partial dependency because the left hand side SID is a proper subset of the candidate key SID C name. And S name is not featuring in any candidate key. So, S name is actually a nonprime attribute and the result of that as you can see in the first two rows or in the third and fourth row you can see that S name is repeated.

So, there is redundancy and therefore, consequently we will have anomalies that we have talked of, but we can normalize we can decompose this into two separate relations R1 and R2 as I am showing on the right; where you associate SID and S name in one table and SID and C name in other table. Naturally then the dependency that the partial dependency that you had disappears because SID determining S name in R1; now becomes is not a partial dependency because in that table SID becomes a primary key. So, it does not qualify as a partial dependency..

So, R1 and R2 both are in second normal form and you will get rid of the redundancy that you saw and this decomposition is ensures that it has a list lossless join incidentally;

this is we have not guaranteed that it is in second normal form and it has also the dependency preservation.

(Refer Slide Time: 14:04)

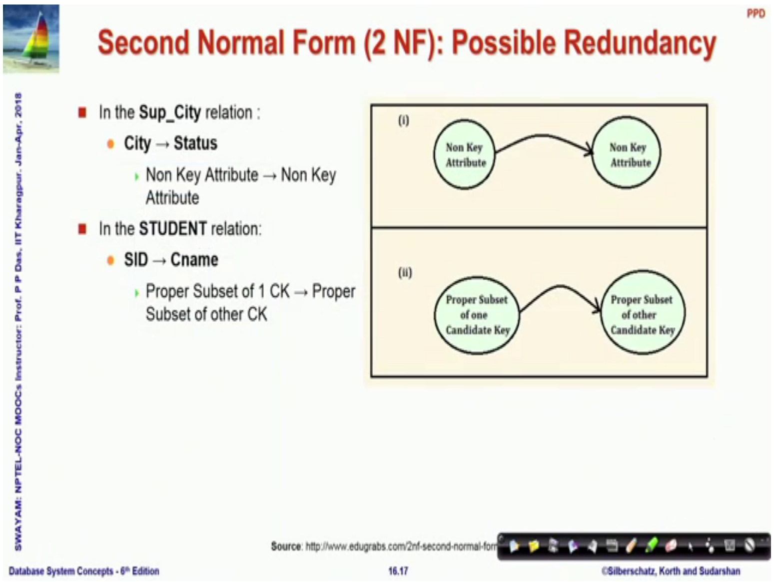


But it is possible again in second normal form a relation could be in second normal form yet it could have some possible redundancies. So, there is a design instance that I am showing with the supplier ID, SID the status key which are functionally determined by SID and the product and quantity values..

So, that in the table supplier SID and PID together form say key whereas, and as that happens you can clearly see that there is a lot of redundancy that you can see in terms of the status happening and which will cause you different anomalies to occur. So, if I normalize in the second normal form on the right then I will have a supplier city say with the three attributes SID status and city and another supplier quantity which has SID PID and quantity naturally in this there is no partial dependency anymore.

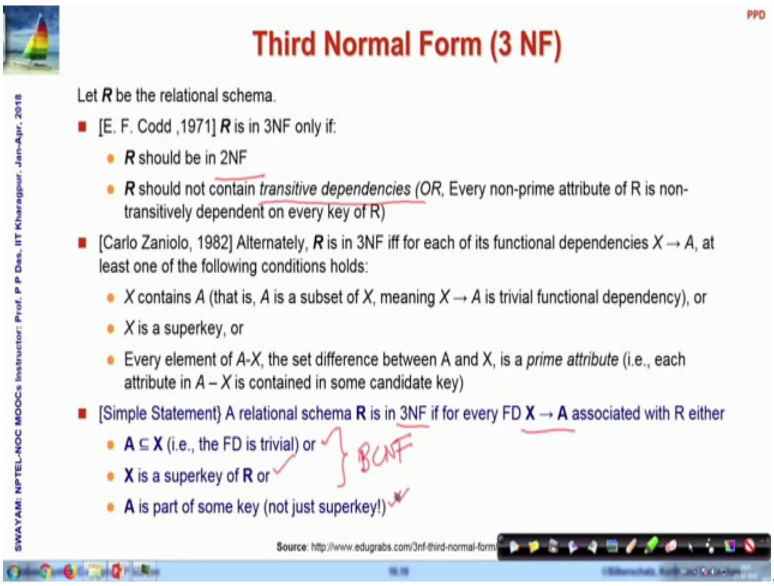
Earlier we had SID determining status as a partial dependency because SID is a proper was a proper subset of the primary key which is SID CID, but after I normalize this dependency does not exist, but yet there will be redundancy in this relationship and there the status will continue to be redundant.

(Refer Slide Time: 15:30)



And for that reason we have to move on to the next type of normal form. So, this I am just explaining here as to what are the possible redundancy sources of possible redundancy that you can have in 2 NF.

(Refer Slide Time: 15:43)



In the 3 NF; third normal form what you define is your relation first of all has to be in 2 NF. So, we are looking at the first definition these are there are three forms of definitions given all of them are actually equivalent, you do not have to worry about why and how they are equivalent slowly you will start understanding.

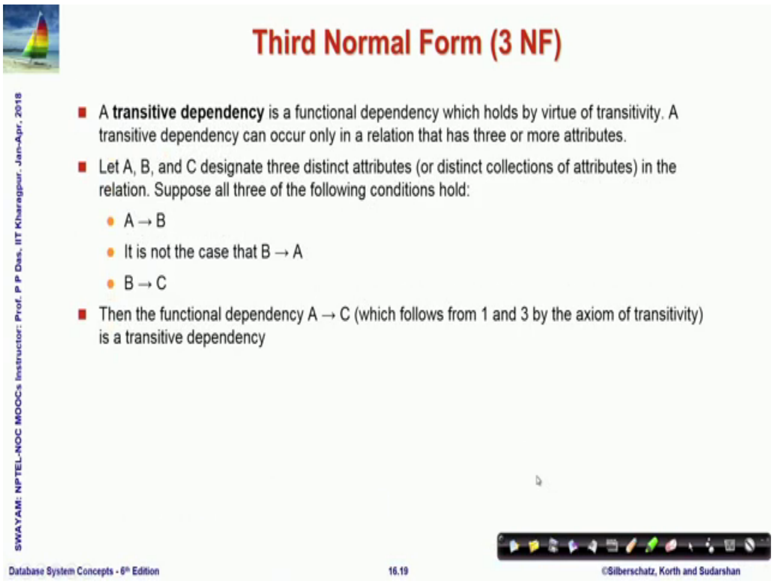
But we take it in three different forms because each form of the definition allow us to understand certain aspect of the three normal form. So, the first thing which is true for everything is it has to be in the second normal form and it should not contain any transitive dependency which means that I should not if I have X determining Y and Y determining Z; then I should not have X determining Z which can be inferred transitively as you know through the angstrom axiom.

Alternatively there was an alternate definition given later on by Zaniolo and I have stated a simpler simplified version of that at the bottom. So, we will say that a relational schema is in 3 NF if for every functional dependency X determining a that holds on this schema either it is a trivial dependency which is X is a A is a subset of X or X is a super key.

So, this is kind of the condition also as you had seen earlier this also is a condition to be in Boyce Codd normal form. So, you can easily understand the 3 NF is a any relation which is in 3 NF is also in the Boyce Codd normal form, but we add a fourth third condition where you say that we will say this is in 3 NF; even if the first two conditions are not satisfied, but a is a part of some key just note the wording is a part of some key not just the super key..

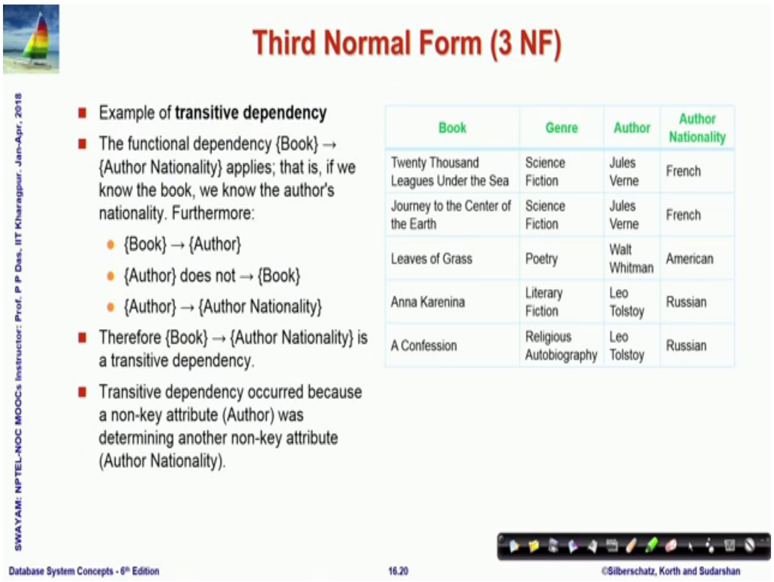
So, if A is a part of some key then and the first two conditions are also not are not satisfied even then we will say that the relation is in third normal form. So, to check for a relation to be in third normal form; we will actually check for whether any one of the three conditions hold.

(Refer Slide Time: 18:00)



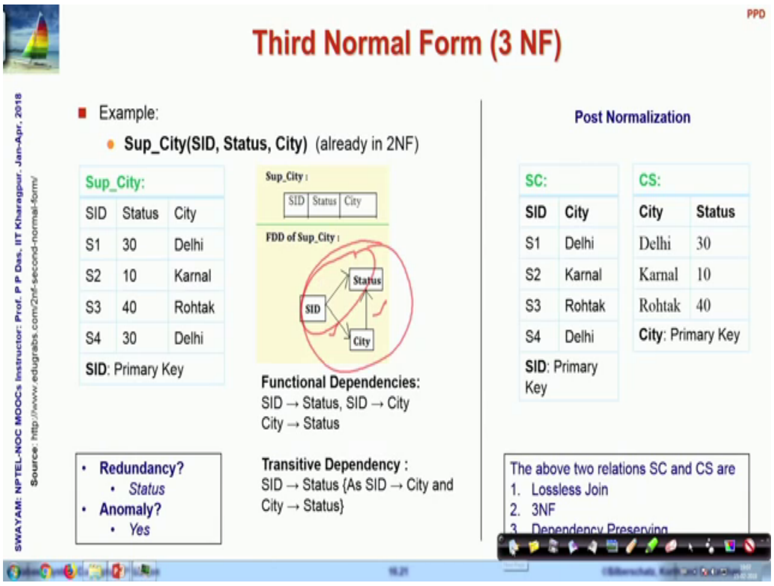
So, this is a definition of transitive dependency which I have just loosely told you. So, I will skip over this.

(Refer Slide Time: 18:09)



There is given another example of a very different kind of a relationship book genre author and author nationality as you can understand. Given the book you know the author there is a functional dependency given the author do you know the author nationality and the, but author does not actually determine the book because the author may have written multiple books. But given that book determines author and author determines author nationality we have that book determines author nationality and therefore, we have redundancy possibility of redundancy in here which is a transitive redundance due to this transitive dependency that we have.

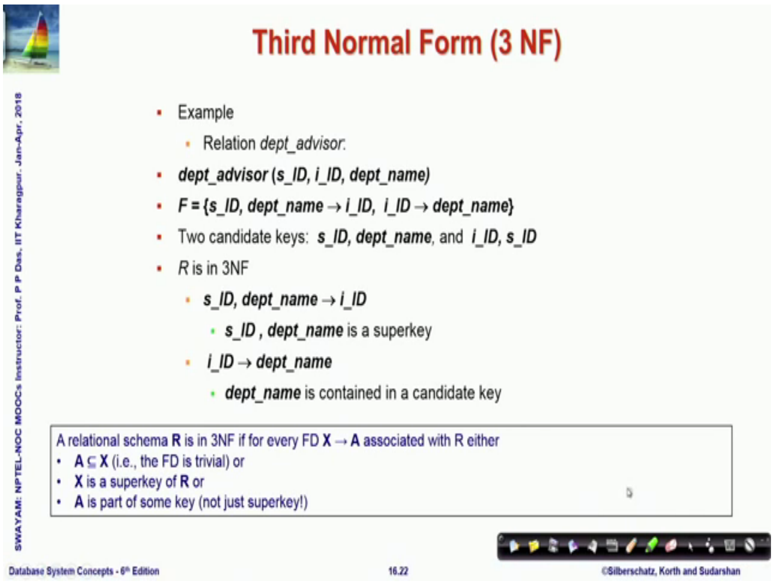
(Refer Slide Time: 18:48)



So, here is a the earlier example where you can as you can see clearly in this diagram you can if you note this diagram you can see that SID determines city and city determines status. So, this is it this is the transitive dependency that SID determines status..

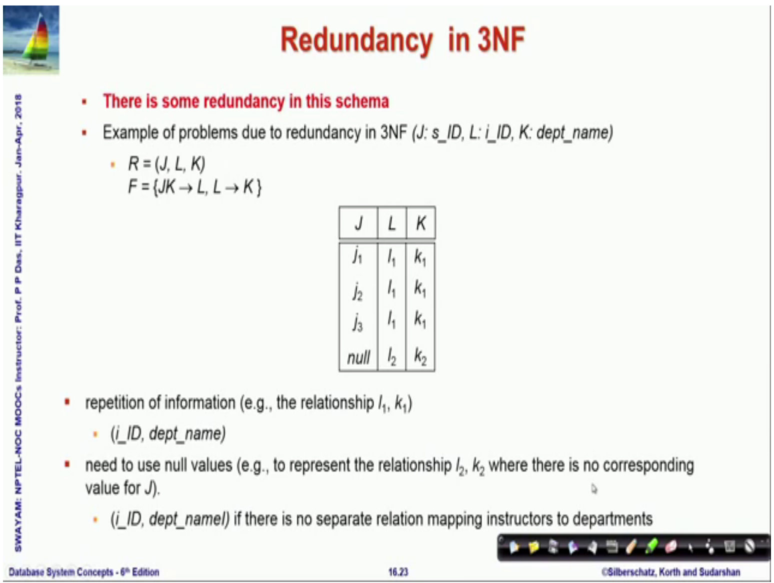
So, if that happens and status becomes redundant and therefore, there could be anomalies. And we can easily normalize by making them into SID and city and city and status. And in that naturally that that redundancy goes away because you have no more the transitive dependency in the relationship; you only have SID determining the city which is a primary key in S C and city determining status which is the primary key in the C S.

(Refer Slide Time: 19:50)



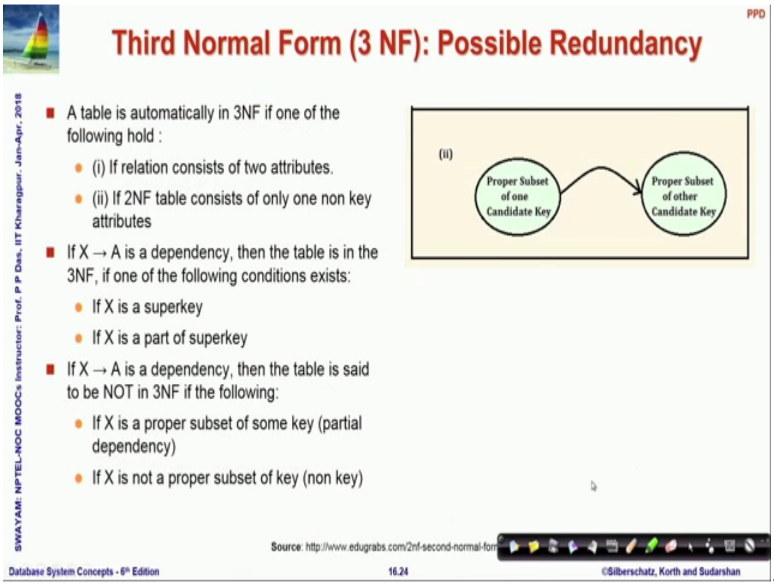
So, there are these are other examples that that you can go through where we have I have taken the example of a student ID I ID and the department name and shown that what kind of problems, you might get into in this. In this case you can see that the relationship actually is in the there because there are two candidate keys and. So, this SID department name is a super key and this relationship is in the third normal form. Because IID determining department name is contained in a candidate key. So, that is the it is a it is in 3 NF due to the third condition that we have had shown.

(Refer Slide Time: 20:41)



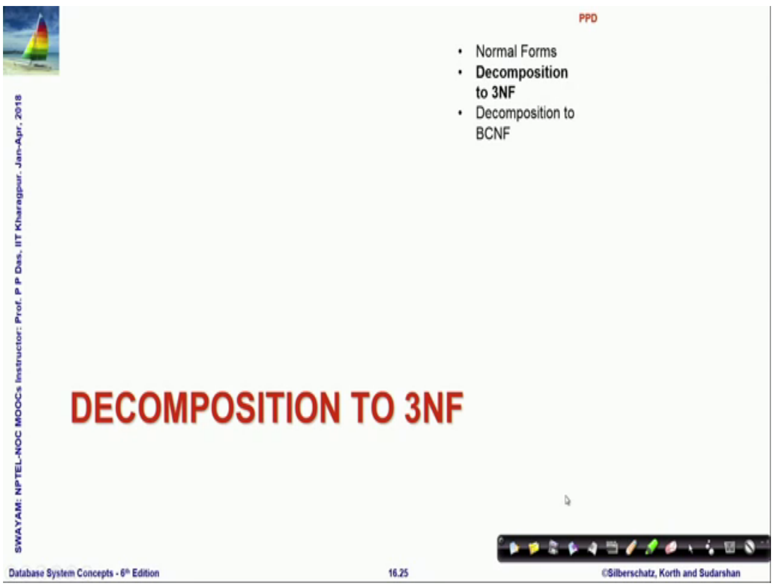
So, when you, but this is a where you can there is some redundancy in this schema that you can observe. So, this is just constructed and you have been because of this redundancy you have been able to we have had to use null values in this case.

(Refer Slide Time: 21:02)



So, in a third normal form there is possible redundancy coming in and these are the different cases that we have to check through.

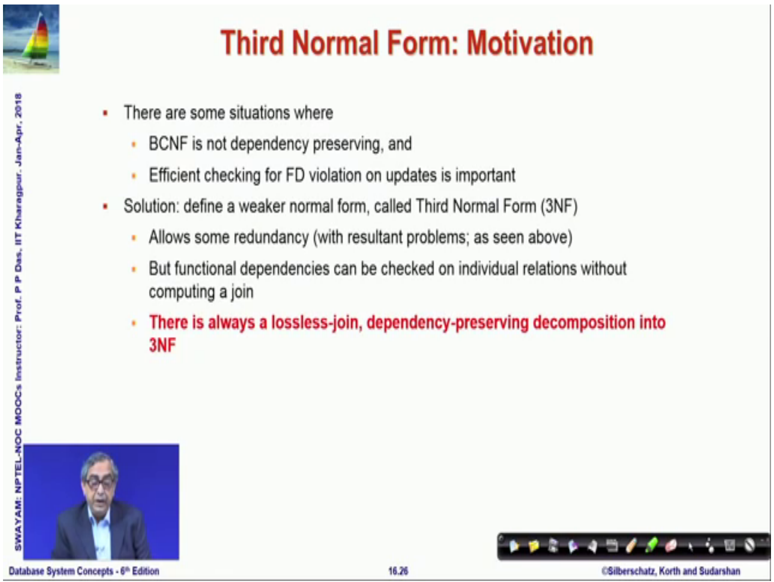
(Refer Slide Time: 21:13)



So, next what ; so, we have seen the different normal forms first normal form no multivalued attribute then the second normal form no partial dependency then the third normal form where you do not have any transitive dependency. So, all these are cascading definitions. So, in third normal form you have low multivalued attribute, no partial dependency and no transitive dependency.

So, now what will take a look into is how if I am given a relational schema and if it is violating any one or more of this condition. So, that the schema is not in the three normal form third normal form then how can we decompose it into the third normal form?

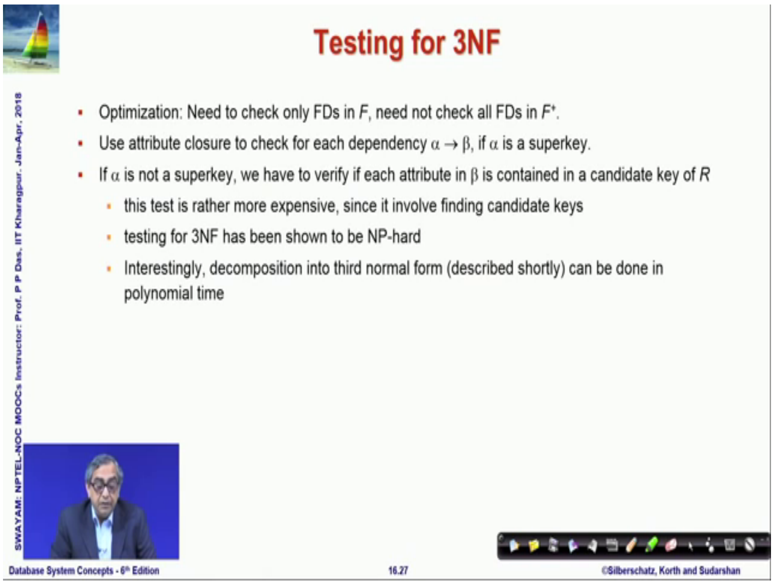
(Refer Slide Time: 21:55)



So, the question naturally is certainly is can it always be done is the basic question that can I always decompose a schema into third normal form the answer is yes you can and that is always a lossless join and dependency preserving decomposition into third normal form which is of great value.

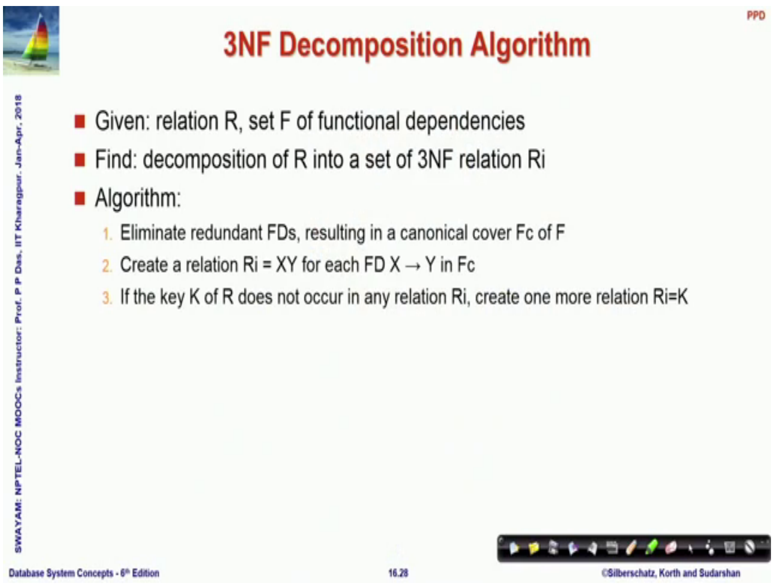
Because that is we said is that desirable properties of our decomposition and if you recall our discussions in the earlier part of the relational design modulesm, then you would recall that Boyce Codd normal form also we had discussed at the early stages. And that gives you a decomposition which is lossless join, but it does not guarantee preservation of the dependencies with third normal form does that.

(Refer Slide Time: 22:49)



So, naturally there are different algorithms first the question is can you test if a relationship is in third normal form; I will not go into the details of that and the computer science result here is testing for third normal form is an NP hard problem. So, there is no known polynomial time algorithm for that, but the interesting thing is the actually that decomposition can be done in very simply in polynomial time.

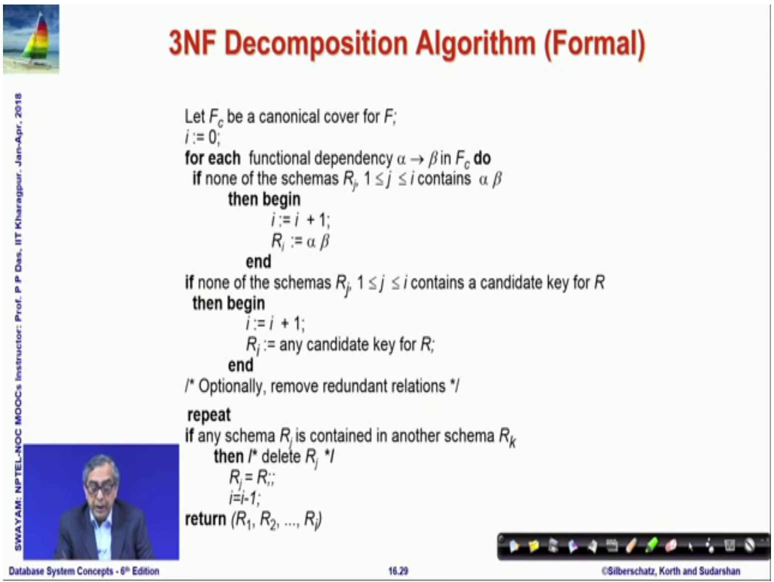
(Refer Slide Time: 23:17)



So, what do you have what is the decomposition algorithm very written in very simple terms you want to you have given a relation R and a set of functional dependencies that hold on you. So, you first compute a canonical cover you know what is a canonical cover. So, you compute a canonical covers you eliminate extraneous attributes eliminate redundant FDs and you have the canonical cover F c from F then you create for every functional dependency X determining Y that exists in the canonical cover.

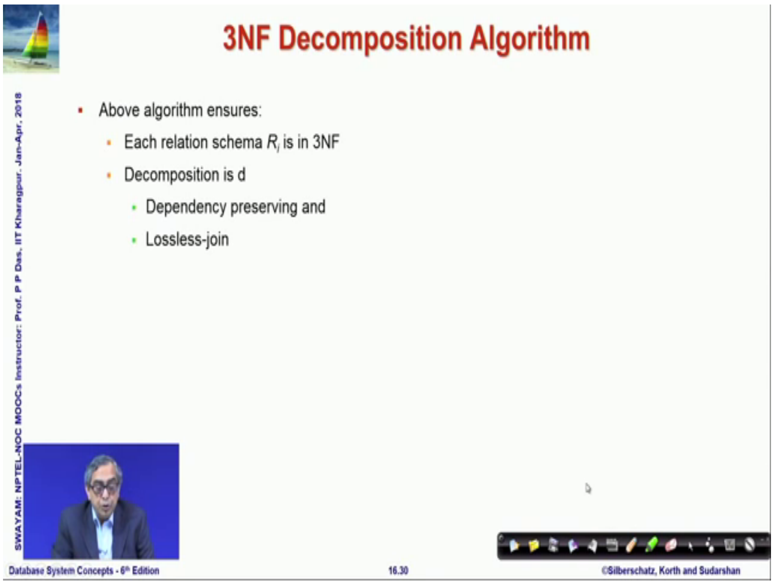
You compute you make a relation say the ith relation taking union of X and Y. So, you call it the relation X Y and you do that for all the functional dependencies in the cover. And after that if you find that the key does not occur in any one of these decomposed relations as generated, then you generate one separate relation to represent the key.

(Refer Slide Time: 24:19)



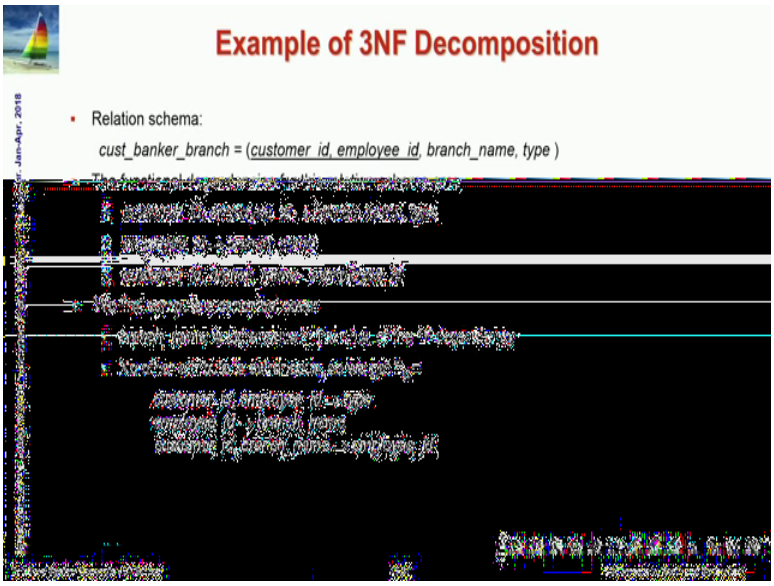
That is a very simple algorithm and I just wrote it in simple hand. So, that you can understand it easily, but here is the formal algorithm. So, if you are interested to rigor I mean in the in the rigor of how 3 NF decomposition will happen here is the algorithm, but I will not go through these in steps.

(Refer Slide Time: 24:37)



So, that ensures that each relation R i that I have decomposed and generated is actually in third normal form and this decomposition is dependency preserving and is lossless join we are not proving that but we are just using that result.

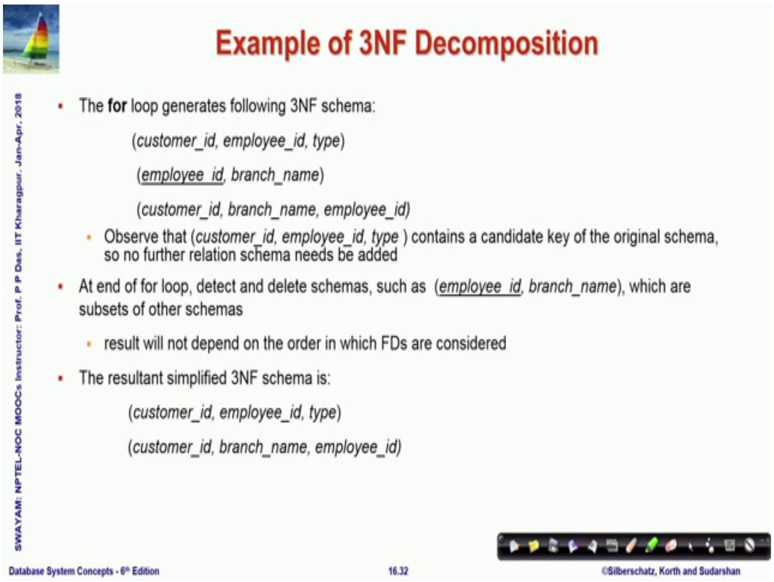
(Refer Slide Time: 24:54)



So, here is an example of a schema; so, we have a customer banker branch. So, these are the four attributes and these are the different functional dependencies that exist. Now naturally given this first thing you will have to do is first thing you have to do is to look at the different to look at taking the canonical cover the minimal cover.

So, if you compute try to compute the minimal cover; you will find that branch name actually is extraneous in the first dependency. So, you can remove that and there is nothing else.

(Refer Slide Time: 25:36)

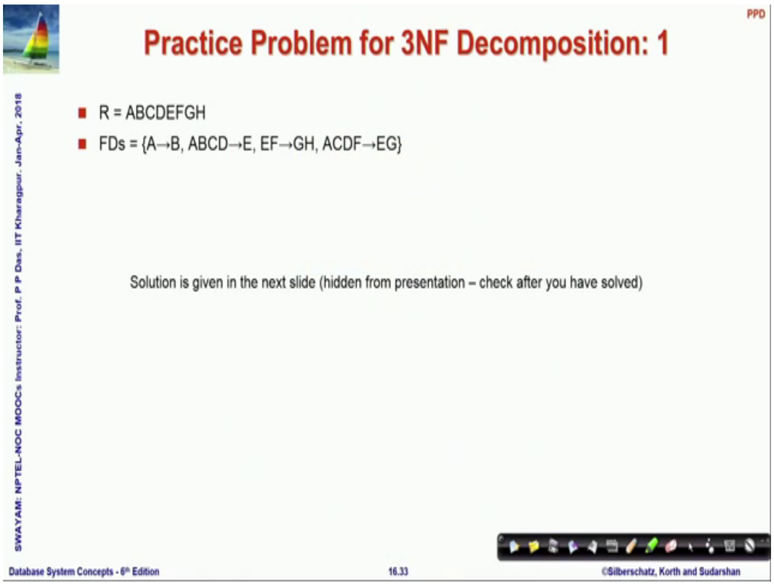


So, your canonical cover turns out to be this set of dependencies and then you go over and for each one of them. So, you take each one the first one is customer ID employee ID determines type. So, for that you generate a schema customer ID, employee ID and type again you take the second functional dependency employee ID determines branch name. So, create employee ID and branch name as a different schema and in this way you will generate three decomposed schema in the third normal form.

Now, once you have done that then you find that your if you look into the original key it was customer ID and employee ID and you find that here in the third second and the third you already have that. So, you do not need to add a separate relation for accommodating the key and also the third relation. So, we can now declare that no further key needs to be added and we have the final 3 NF decomposition..

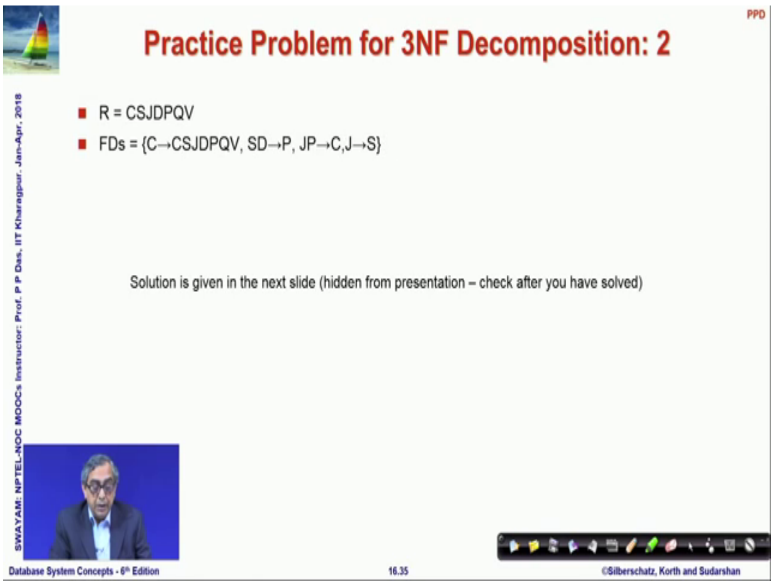
So, at the end of the fault detect and delete. So, this is this is a stated in terms of the detailed algorithm, but this is you can say that the employee ID and branch name the second relation in the decomposition is actually a subset of the third relation. So, you can remove that as well. So, you will be left with only two relations in this decompose schema which both of which are in third normal form and this decomposition is guaranteed you lossless join and dependency preservation.

(Refer Slide Time: 27:17)



So, I have given some practice problems for you I have also given the solution, but the solution is not in the current run of the presentation; you will get see them in the presentation as hidden slides. So, you first try solving them and once you have solved them then you look at the solution in the slide.

(Refer Slide Time: 27:37)

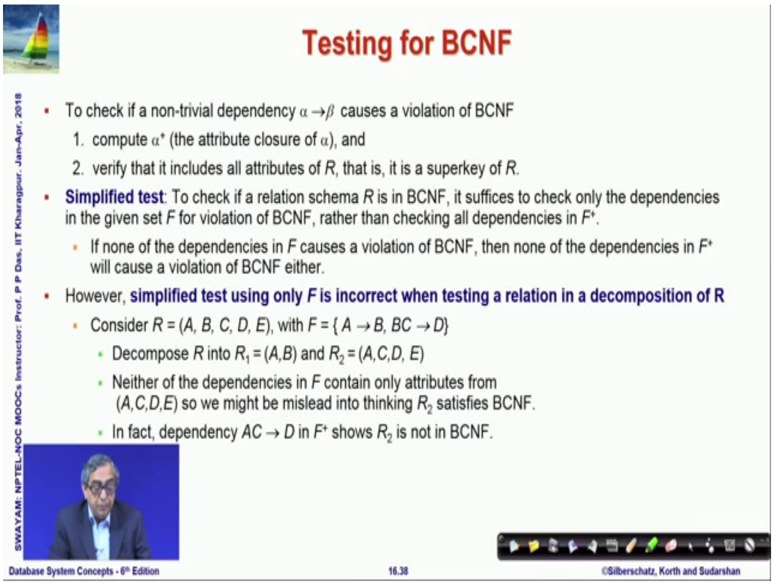


So, there are two problems; so, this is a second one and you can solve them in that way.

(Refer Slide Time: 27:40)

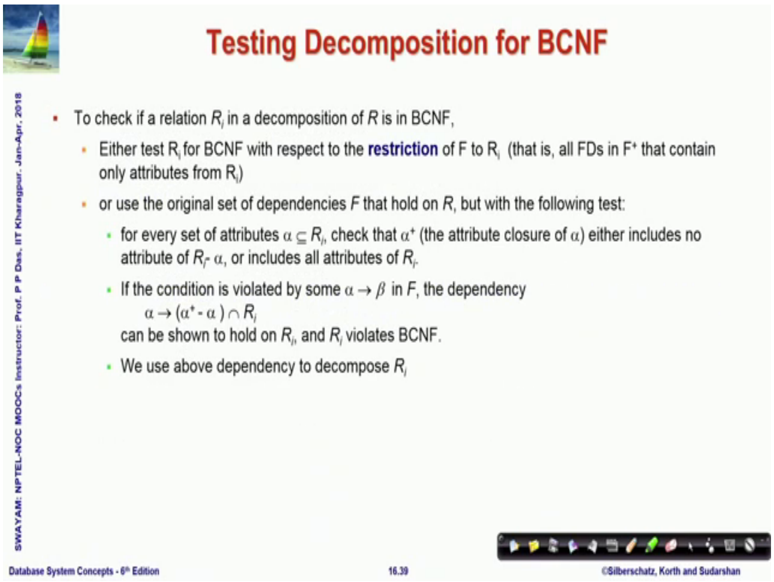
Next is the we will quickly recap on the decomposition of BCNF Boyce Codd normal form which we had seen earlier.

(Refer Slide Time: 27:49)



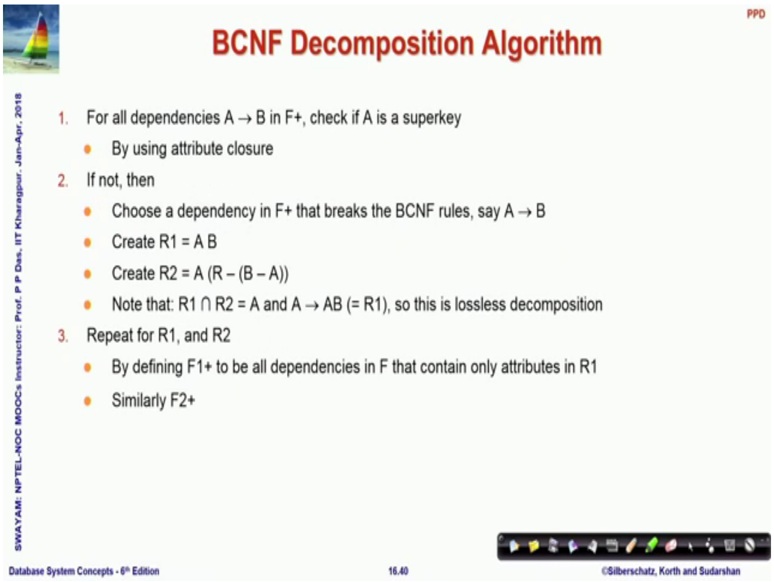
And we know that the Boyce Codd normal form guarantees that there will have be every dependency that exists must be either trivial or the left hand side must be a super key. So, using the algorithms, you can test for the Boyce Codd normal form which is described here I am not going through in steps.

(Refer Slide Time: 28:08)



And here is the more detailed formal algorithm to find determine whether a Boyce Codd normal form is in a decomposed form is in Boyce Codd.

(Refer Slide Time: 28:18)

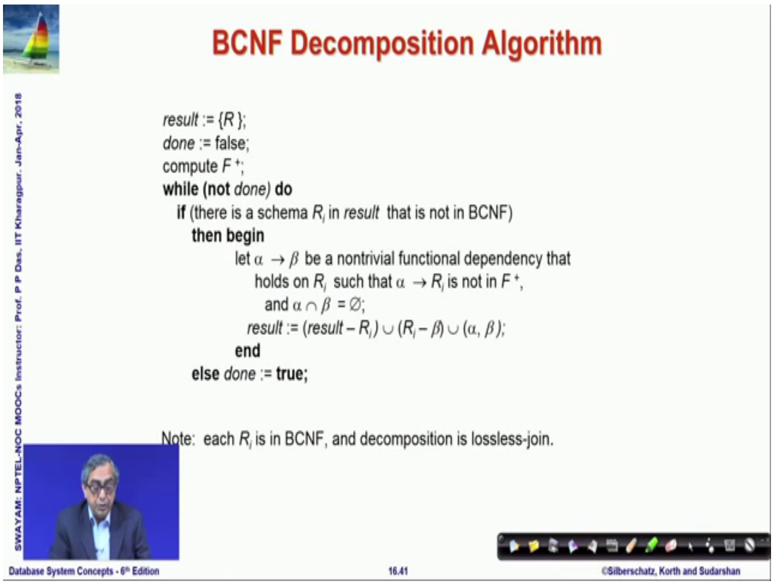


So, I will just quickly recap on the algorithm to do that naturally for all dependencies you first determine the super key and check if A determining B is a super key or not if it and that you can easily do using attribute cover. If it is not a super key then you choose a dependency A determining B which violates and you form by Boyce Codd goes in every step it decomposes one relation into two separate relation..

So, one that you take by taking union of the attributes of A and B and the other where you take out B minus A; these attributes this difference attributes you take out from R and then you add A and make the other relationship. Naturally in between these two A is a common attribute and since and that will determine A B because A determines B..

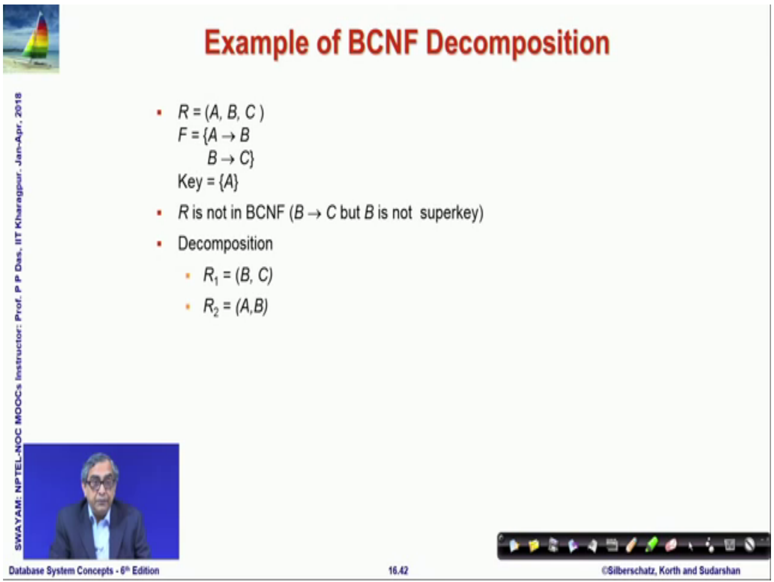
So, A will determine A B that is whole of R1. So, naturally the lossless join is guaranteed and you repeat that keep on doing that for the resultant relations that you have got. keep on decomposing them till you finally, close and you have no more violating dependency and you will have a decomposition into Boyce Codd normal form.

(Refer Slide Time: 29:39)



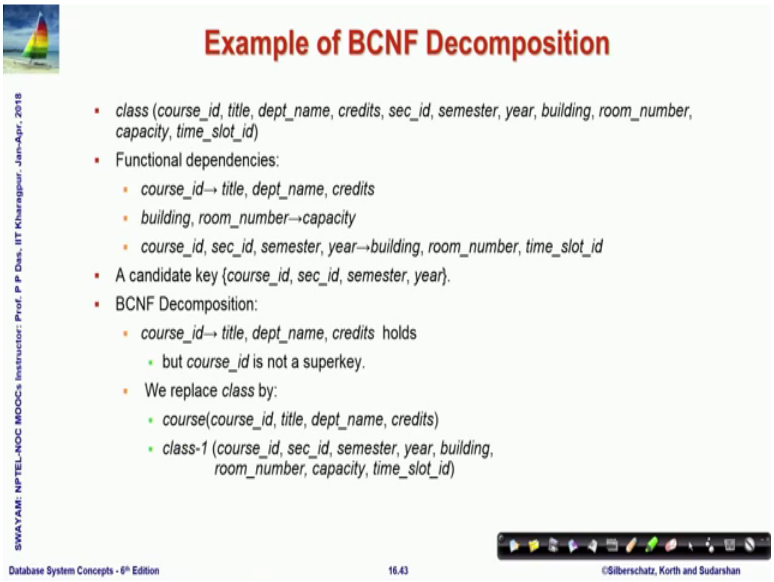
Here is the formal algorithm again for you to go by steps if you are interested.

(Refer Slide Time: 29:46)



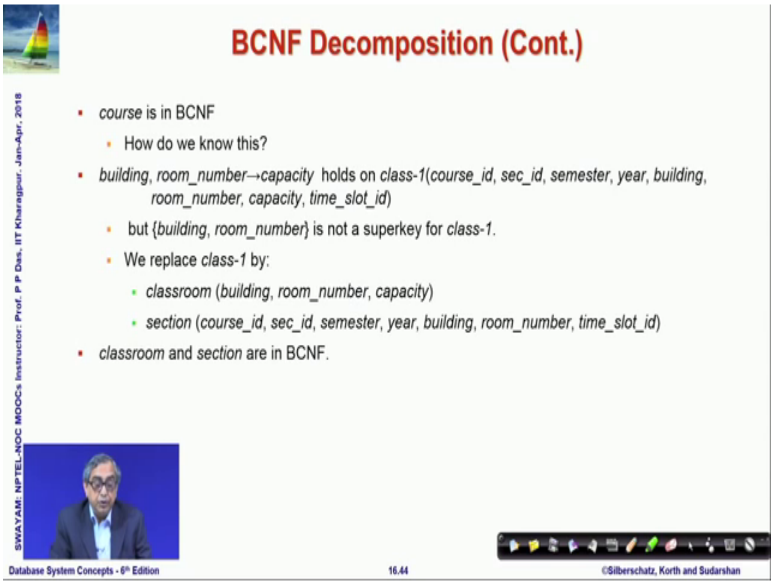
Otherwise you know how to do this; again I have shown another example here which is showing that how to decompose in BCNF. So, you should practice this that is why I have work them out in steps here. So, here A determines B; B determine C naturally A is the key R is not in BCNF because B determining C is a functional dependency where B is not a super key.

(Refer Slide Time: 30:15)



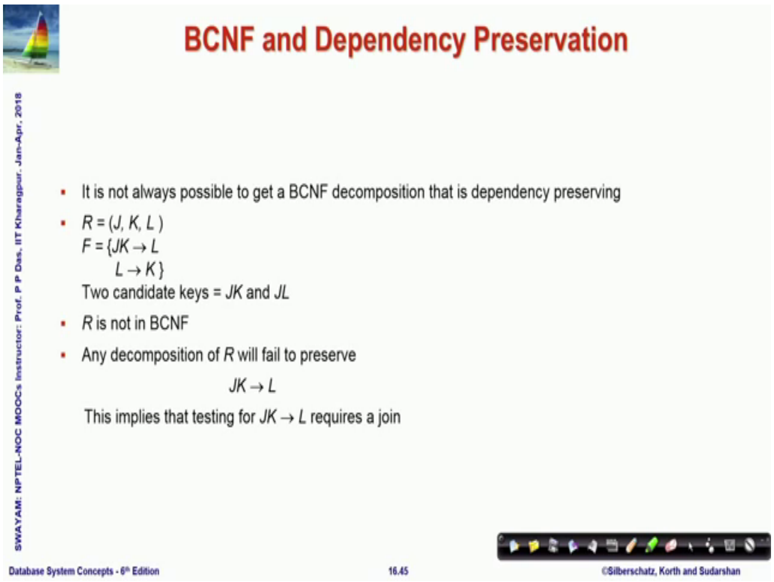
So, you can decompose them in terms of. So, you can decompose in terms of B C as one relation and A B as another relation. Here is another example a more detailed one of a class relationship which has a whole set of attributes and these functional dependencies and based on that the candidate key is course ID, section ID, semester and year and you can proceed with the BCNF decomposition; taking the first functional dependency that holds, but the left hand side the course ID is not a super key. So, you will replace it by a one relation; which is say new course relation and a new class relation which is the remaining attributes.

(Refer Slide Time: 31:06)



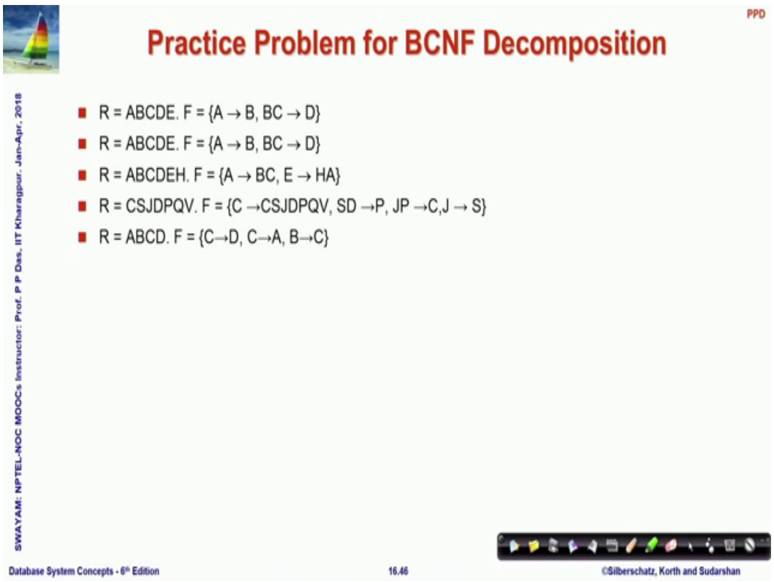
And then you get convinced that course is in BCNF, but the other one the class is not because building and room number determines capacity where building room number together is not a super key. So, you split it again and you replace class 1 in terms of 2 new relations class room and section and both of them are in BCNF and you are done with this.

(Refer Slide Time: 31:31)



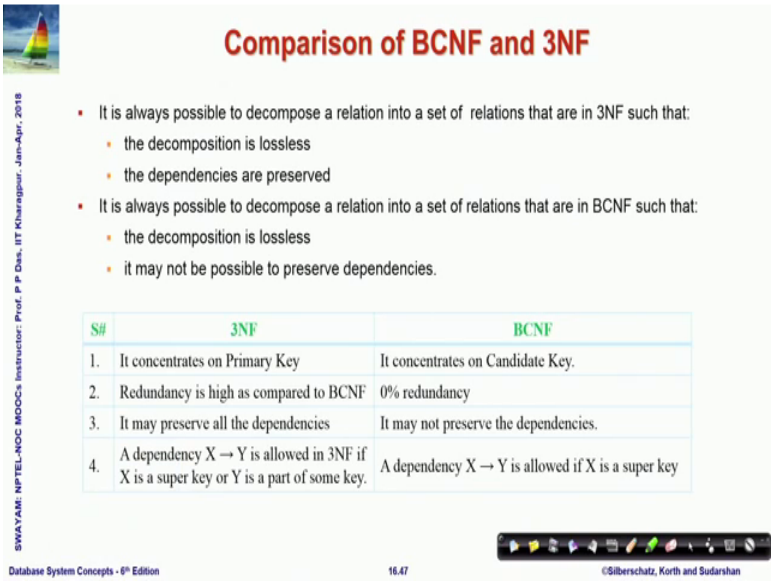
But BCNF as I would again warning you BCNF does not preserve dependence it gives you lossless join, but it does not preserve the dependencies. So, it is not always possible to decompose into BCNF with dependency preservation. So, here is an example which we saw little earlier and there are two candidate keys R is not in BCNF, you can clearly see and any decomposition will fail JK determining L and that will require a join. So, this will not preserve the dependencies in terms of the decomposition.

(Refer Slide Time: 32:06)



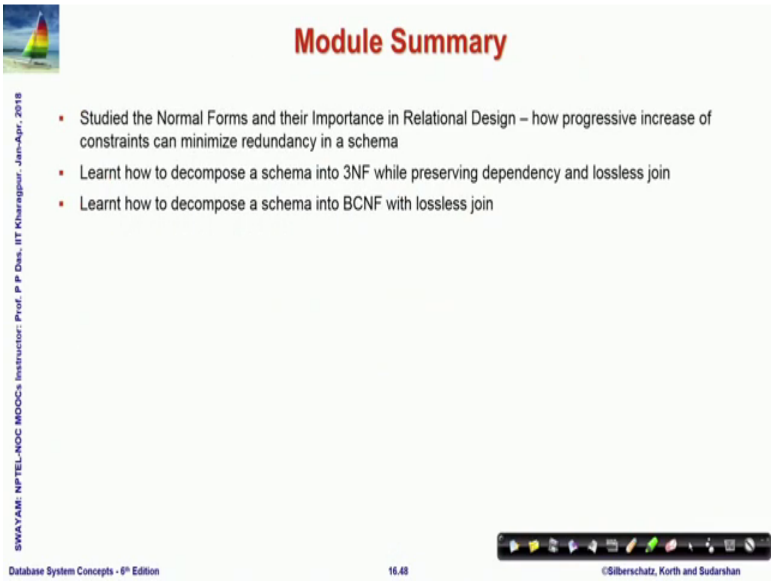
Again I have given a set of practice problems here which we you should try and get confident in terms of the Boyce Codd from normal form normalization.

(Refer Slide Time: 32:19)



Now, it is always possible to decompose a relation into a set of relation in 3 NF; if the decomposition is lossless and the dependencies are preserved. Whereas, in case of BCNF it is not possible; so, here is a table which summarizes the relative comparison between Boyce Codd and third normal form because they are the common once Boyce Codd naturally is more strict it gives you lesser dependent lesser redundancies, but it cannot guarantee that your dependencies will be preserved. So, more often we will accept 3 NF as an acceptable normalized decomposition with some redundancy still existing it is possible and we cannot get rid of them.

(Refer Slide Time: 33:09)



So, we have studied about the normal forms and their importance and how progressively we can increase the constraints to minimize redundancy in the schema and learned how to decompose a schema into third normal form and also in the Boyce Codd normal form.