

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

Lecture - 17
Relational Database Design (Contd.)

Welcome to the Module 17 of Database Management Systems. From the last module, we are discussing Relational Database Design. So, this is second in the series of 5 modules which we will discuss this.

(Refer Slide Time: 00:31)

PPD

Module Recap

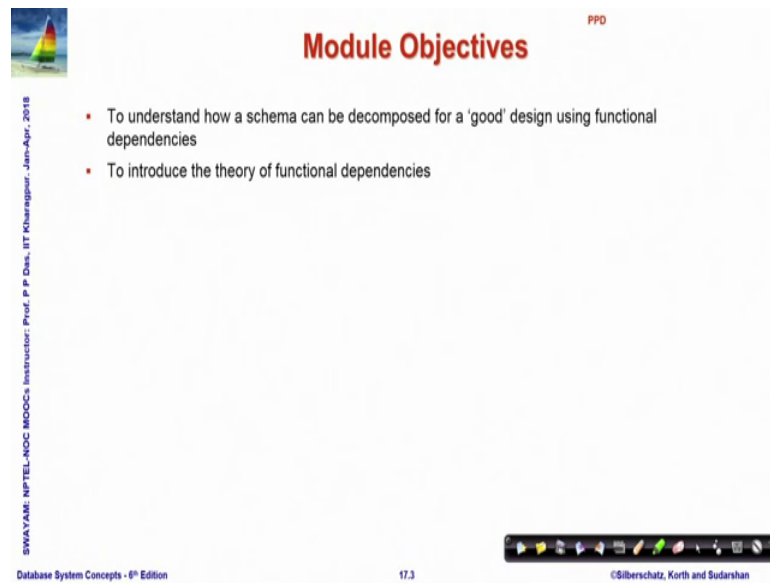
- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Functional Dependencies

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 17.2 ©Silberschatz, Korth and Sudarshan

We have already seen basic features of good relational design. We have studied about first normal form atomic domains and got introduced to functional dependencies.

(Refer Slide Time: 00:43)



PPD

Module Objectives

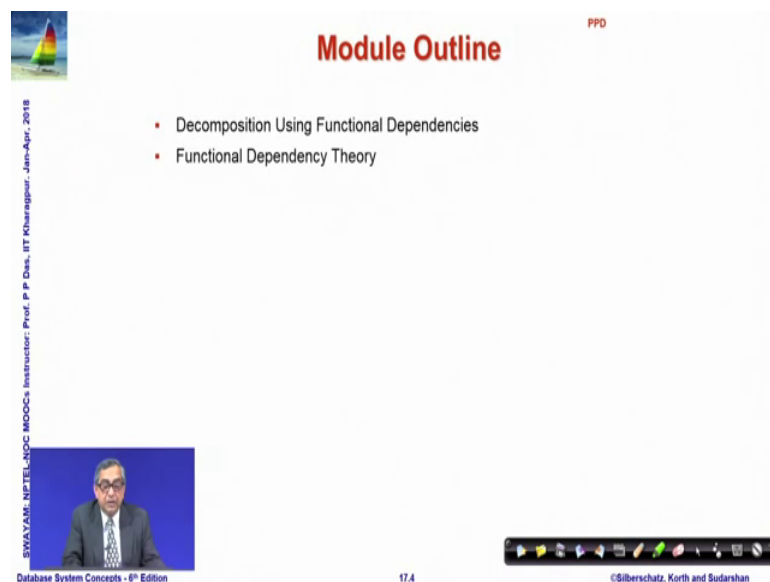
- To understand how a schema can be decomposed for a 'good' design using functional dependencies
- To introduce the theory of functional dependencies

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P P Das, IIT Khargapur, Jan-Apr, 2018

Database System Concepts - 6th Edition 17.3 ©Silberschatz, Korth and Sudarshan

So, we will develop further on that to see how decompositions into good design can be done by making use of the notion of functional dependencies and we will more formally introduce the theory of functional dependencies.

(Refer Slide Time: 00:57)



PPD

Module Outline

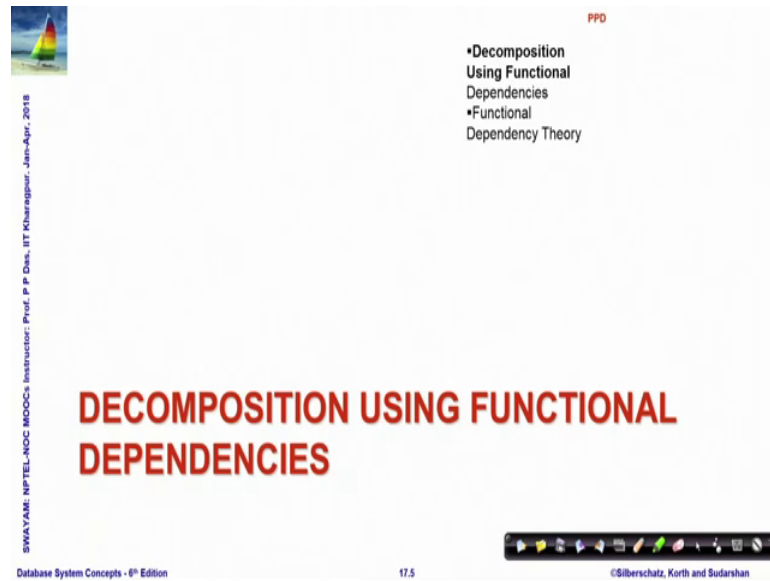
- Decomposition Using Functional Dependencies
- Functional Dependency Theory

SWAYAM: NPTEL-NOC MOCs Instructor: Prof. P P Das, IIT Khargapur, Jan-Apr, 2018

Database System Concepts - 6th Edition 17.4 ©Silberschatz, Korth and Sudarshan

So, that is all that we discuss in this.

(Refer Slide Time: 01:00)



PPD

- Decomposition Using Functional Dependencies
- Functional Dependency Theory

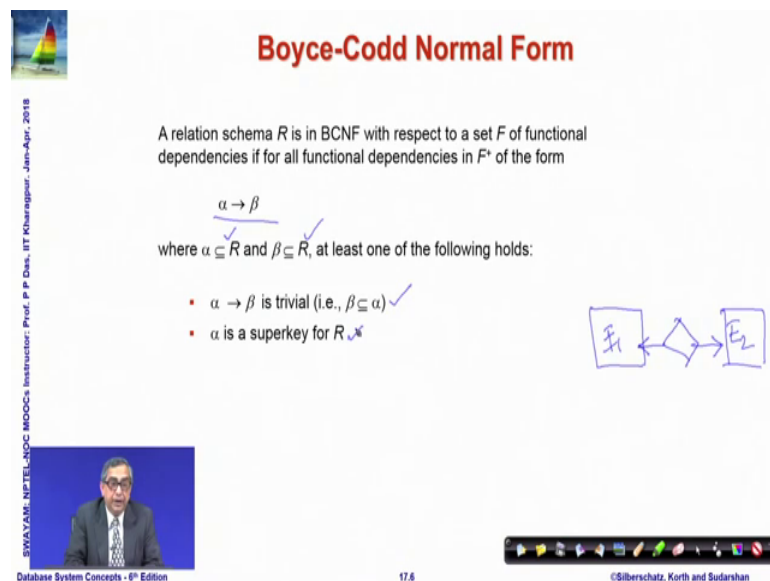
DECOMPOSITION USING FUNCTIONAL DEPENDENCIES

SWAYAM: NPTEL-NOC MBOCS Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 8th Edition 17.5 ©Silberschatz, Korth and Sudarshan

So, decomposition using functional dependencies is the first thing that we look at.

(Refer Slide Time: 01:04)




Boyce-Codd Normal Form

A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$) ✓
- α is a superkey for R . ✓



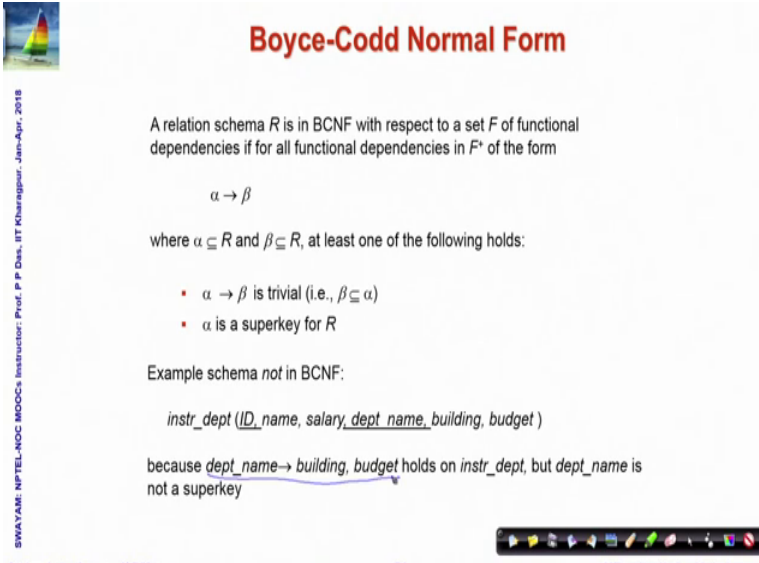
SWAYAM: NPTEL-NOC MBOCS Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 8th Edition 17.6 ©Silberschatz, Korth and Sudarshan

The first normal form of relations which were studied, we look at its Boyce-Codd Normal Form. So, normal forms are kind of set of properties which is satisfied by a relational schema and if they are satisfied, then we have certain guarantees in terms of what can or cannot happen in that relational schema design. So, Boyce-Codd is a simplest kind of beyond 1NF is a simplest kind of normal form and a relational schema is said to be in Boyce-Codd normal form if with respect to a set of functional

dependencies, all functional dependencies in the closure. So, in respect of F , we compute F^+ which is a closure and if I have a dependency $\alpha \rightarrow \beta$, then naturally α will have to be a subset of R β will have to be a subset of R , but what is important is every functional dependency in the closure set must either be trivial that is right hand side is a superset of the left hand side or the left hand side set α must be super key. So, only those kind of functional dependencies are possible. No other functional dependencies are possible. If that is satisfied by the relational schema R , then it is said to be in the Boyce-Codd normal form.

(Refer Slide Time: 02:39)



Boyce-Codd Normal Form

A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F^+ of the form

$$\alpha \rightarrow \beta$$

where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:

- $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
- α is a superkey for R

Example schema *not* in BCNF:

instr_dept (ID_name, salary, dept_name, building, budget)

because dept_name \rightarrow building, budget holds on *instr_dept*, but dept_name is not a superkey

Database System Concepts - 9th Edition 17.6 ©Silberschatz, Korth and Sudarshan

So, if we look at *instr_dept* schema of the combined relations we saw last time, then we will know that certainly this is not in Boyce-Codd Normal Form because this functional dependency holds in this schema where it is neither a trivial dependency and nor department name is a super key. So, this is not in BCNF.

(Refer Slide Time: 03:10)

Decomposing a Schema into BCNF

- Suppose we have a schema R and a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF

We decompose R into:

- $(\alpha \cup \beta)$
- $(R - (\beta - \alpha))$

In our example,

- $\alpha = \text{dept_name}$
- $\beta = \text{building, budget}$
- $\text{dept_name} \rightarrow \text{building, budget}$

inst_dept is replaced by

- $(\alpha \cup \beta) = (\text{dept_name, building, budget})$ ✓ R_1
- $\text{dept_name} \rightarrow \text{building, budget}$ ✓
- $(R - (\beta - \alpha)) = (\text{ID, name, salary, dept_name})$ ✓ R_2
- $\text{ID} \rightarrow \text{name, salary, dept_name}$ ✓

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 8th Edition 17.7 ©Silberschatz, Korth and Sudarshan

So, if a relational scheme is not in BCNF, then the question naturally is can I make it into BCNF so then that process is the process of decomposition. So, what you do? You divide the set of attributes into two or more at sets of attributes. So, here let us say that we have a relational schema which has a non-trivial dependency alpha determines beta where alpha is not a super key. So, with respect to this functional dependency, the relational schema is not in BCNF, then we can decompose R by two sets. One is alpha union beta take the union of these two attribute sets and remove beta minus alpha from R, take the difference of beta minus alpha and remove that from R. The resulting pair of relations, relational schemas will be in Boyce-Codd Normal Form with respect to this particular functional dependency.

So, let us see an example. So, if alpha is department name beta is a pair of attribute building and budget, we have department name functionally determines building budget. So, alpha determines beta and we have already seen that it does not hold. It is not satisfied by the inst department. So, you replace it by taking alpha union beta. So, alpha union beta is this set of relational this relational schema and you do R minus beta R minus difference of beta minus alpha beta minus alpha. Naturally if this is beta and alpha, then beta minus alpha is necessary building budget because if the department does not occur in beta.

So, this set is building budget and if I remove it from R which means that id, name, salary and department name are retained, but building and budget gets removed. So, I get another relational schema which has these four names and it holds the functional dependency id determinant. So, even now if I look into this schema R1 and this schema R2, there are different dependencies that hold on R1 and with respect to that dependency R1 is in BCNF because department name is the super key, is the primary key and with respect to this dependency, R2 is in BCNF because id is the key.

So, I can see that the original combined relational schema was not in BCNF with respect to this functional dependency, but when I do this decomposition, I get two schemas which are each in BCNF normal form. So, this is the basic process and we will see depending on the normal form and different notions of functional dependencies, we will see how these conversions can be done, but this is a basic approach of converting a schema into a normal form.

(Refer Slide Time: 06:24)

BCNF and Dependency Preservation

- Constraints, including functional dependencies, are costly to check in practice unless they pertain to only one relation
- If it is sufficient to test only those dependencies on each individual relation of a decomposition in order to ensure that *all* functional dependencies hold, then that decomposition is *dependency preserving*.
- Because it is not always possible to achieve both BCNF and dependency preservation, we consider a weaker normal form, known as *third normal form*.

Database System Concepts - 6th Edition 17.8 ©Silberschatz, Korth and Sudarshan

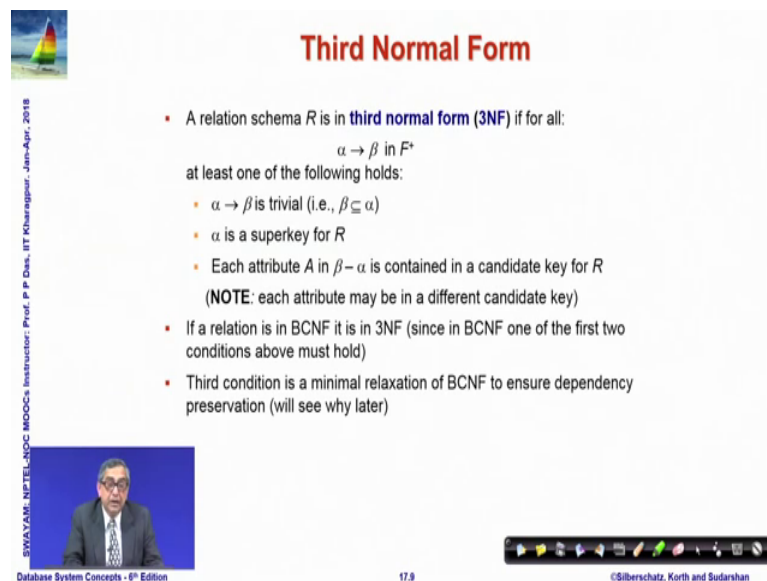
Now, the question is if the constraints including the functional dependencies if we look at, then functional dependencies will have to be checked on different instance. Now, in general it is difficult to check a functional dependency alpha determining beta if the attributes alpha and the attributes of beta or the attributes of beta are distributed between multiple relations because naturally how do I check if they are true, how do I check that two tuples which match on alpha is indeed matching on beta unless I perform a costly

join operation. So, there objective is to be able to come to designs where it is sufficient to test only those dependencies on individual relations of the decomposition and with that I must be able to ensure that all functional dependencies hold. So, it is a very interesting situation.

So, we are saying that we will decompose, get into a number of relational schema. Every schema will have a number of dependencies, functional dependencies and those functional dependencies if they involve only the attributes of that relational schema, they can be tested very easily and if these functional dependencies together mean ensure that all functional dependencies hold that is if the closure of this set of functional dependencies is same as the closure of the earlier set, the original set, then we say that the decomposition that we have achieved is dependency preserving because I can actually effectively compute.

This is dependency preserving because I can effectively compute whether every dependency is satisfied by checking on every individual relation, but the unfortunate part of the reality is that it is not always possible to achieve a Boyce-Codd Normal Form Decomposition which also preserves the dependencies. See if there are in some cases will be able to do like the example we saw just now the instructor and department, but it is not always possible. So, we usually need another weaker form, normal form which is known as a third normal form and we will subsequently look into those.

(Refer Slide Time: 09:09)



Third Normal Form

- A relation schema R is in **third normal form (3NF)** if for all:
 $\alpha \rightarrow \beta$ in F^+
at least one of the following holds:
 - $\alpha \rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$)
 - α is a superkey for R
 - Each attribute A in $\beta - \alpha$ is contained in a candidate key for R
(NOTE: each attribute may be in a different candidate key)
- If a relation is in BCNF it is in 3NF (since in BCNF one of the first two conditions above must hold)
- Third condition is a minimal relaxation of BCNF to ensure dependency preservation (will see why later)

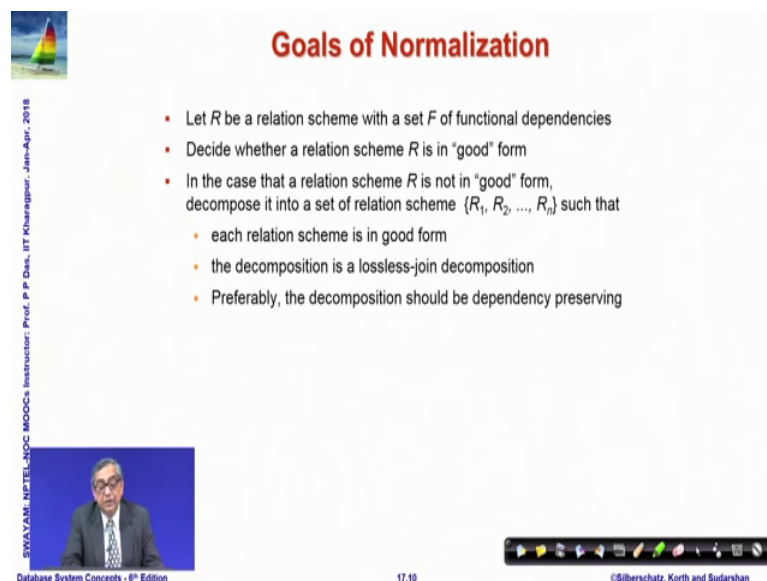
SWAYAM, NPTEL, NDC MOOCs, Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr., 2018

Database System Concepts - 9th Edition 17.9 ©Silberschatz, Korth and Sudarshan

A third normal form is again a relational schema is there and for all attribute, for all dependencies that belong to the closure of the functional dependencies, this following conditions must hold either alpha, determines beta is trivial which is a condition which BCNF or alpha is a super key of R which is also a condition that we say an effort or each attribute in beta minus alpha that is right hand side difference. The left hand side is contained in a candidate key for R. It is not very obvious as to why we need that. That will unfold slowly. This is the condition we did not have in BCNF. So, naturally you can see that based on the first two conditions, you can always say that if a relational schema is in BCNF, it necessarily is in 3NF, but not that if.

There could be some schema which is in 3NF because of the third condition where there exists a functional dependency. So, that beta minus alpha is contained in a candidate key for R, but it is not in the BCNF form and also you can note that the attributes of that are contained in beta minus alpha must be in some candidate key, not necessarily in the same candidate key. If they exist in some candidate key, then itself 3NF condition will get satisfied. So, if a relation is in BCNF, it is in 3NF. We have already seen that. So, third condition minimally relaxes BCNF to ensure that we have a dependency preservation. We will see this more later. So, I am just introducing the concept of a relaxed normal form here.

(Refer Slide Time: 11:11)



Goals of Normalization

- Let R be a relation scheme with a set F of functional dependencies
- Decide whether a relation scheme R is in "good" form
- In the case that a relation scheme R is not in "good" form, decompose it into a set of relation scheme $\{R_1, R_2, \dots, R_n\}$ such that
 - each relation scheme is in good form
 - the decomposition is a lossless-join decomposition
 - Preferably, the decomposition should be dependency preserving

SWAYAM - NPTEL - MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition

17.10

©Silberschatz, Korth and Sudarshan

So, what is a goal of this normalization is if to summarize let R be a relational scheme, F is a set of functional dependencies, we need to decide whether the relational scheme R is in a good form which means that it should not have unnecessary redundancy. It should be impossible to acquire information by doing lossless join. So, in case it is not in good form. We can convert it by decomposition into N relational schema, such that each schema is in good form. The decomposition has a lossless join, so that I can get back the original relation from this and preferably the decomposition should preserve that dependencies. So, that is what we will target henceforth.

(Refer Slide Time: 12:08)

How good is BCNF?

- There are database schemas in BCNF that do not seem to be sufficiently normalized
- Consider a relation
 - inst_info* (*ID*, *child_name*, *phone*)
 - where an instructor may have more than one phone and can have multiple children

<i>ID</i>	<i>child_name</i>	<i>phone</i>
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	William	512-555-4321

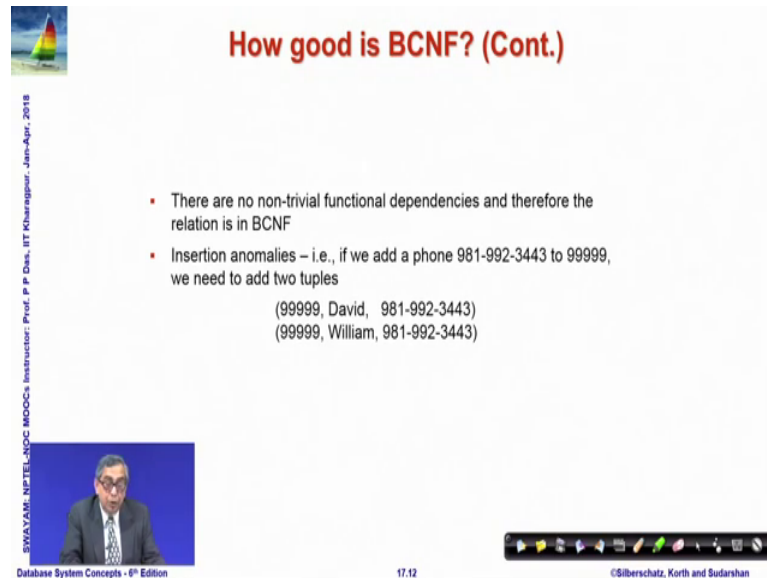
inst_info

Database System Concepts - 9th Edition 17.11 ©Silberschatz, Korth and Sudarshan

So, when we do that let us quickly evaluate as to we have seen BCNF. So, how good really BCNF is. So, if I have something in BCNF should I really be very happy always. So, let us look at a relational schema. This is an information relating the idea of a person, the name of the child and the phone number and naturally the person, the instructor may have more than one phone and may have multiple children. So, this is a possible instance that you can see though all of these belong to the same instructor. He has naturally you can see that two children and there are this is here, this is here.

So, this is here and this is here. So, there are two different phone numbers. So, naturally you have four possible combinations that you need to look at.

(Refer Slide Time: 13:08)



How good is BCNF? (Cont.)

- There are no non-trivial functional dependencies and therefore the relation is in BCNF
- Insertion anomalies – i.e., if we add a phone 981-992-3443 to 99999, we need to add two tuples
 - (99999, David, 981-992-3443)
 - (99999, William, 981-992-3443)

SWAYAM - NPTEL_NOC MOOCs Instructor: Prof. P. P. Das, IIT Khargpur, Jan-April, 2018

Database System Concepts - 9th Edition

17.12

©Silberschatz, Korth and Sudarshan

So, now there is no non-trivial functional dependency in this relation. So, since there is no non-trivial functional dependency, this relation naturally is in BCNF form because that is the existence of non trivial dependency is what makes a schema not conform to the BCNF form. So, there is no such. So, this is in BCNF form and now, if you look at, but what did we see the key thing that we saw if we just go back, the key thing that we saw that there is ample redundancy of data, the same data is entered multiple times.

So, the consequence of that could be insertion anomaly. If we want to add a phone number to the same instructor, then we need to add two tuple because the instructor also has two children. If the instructor and three children will need to add three and unless this is maintained always, then we will have difficulty.

(Refer Slide Time: 14:15)

How good is BCNF? (Cont.)

- Therefore, it is better to decompose *inst_info* into:

	<i>ID</i>	<i>child_name</i>
<i>inst_child</i>	99999	David
	99999	David
	99999	William
	99999	Willian

	<i>ID</i>	<i>phone</i>
<i>inst_phone</i>	99999	512-555-1234
	99999	512-555-4321
	99999	512-555-1234
	99999	512-555-4321

This suggests the need for higher normal forms, such as Fourth Normal Form (4NF)

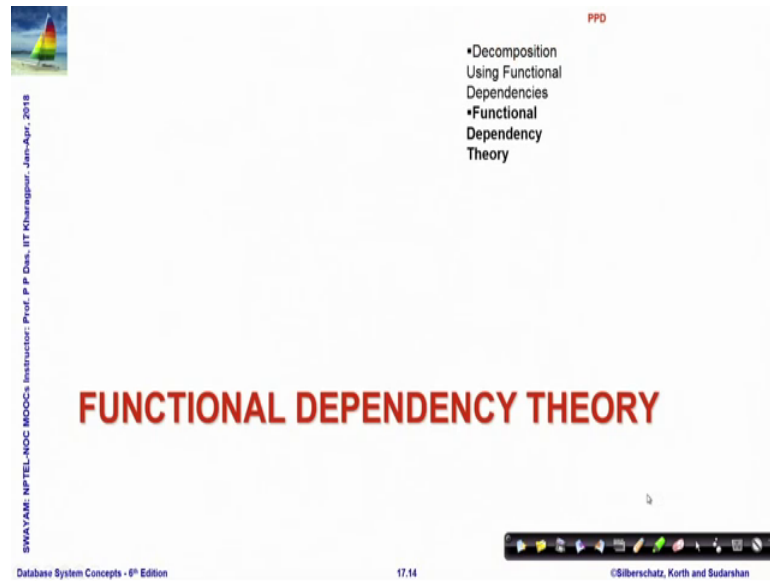
SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P.P. Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 8th Edition 17.13 ©Silberschatz, Korth and Sudarshan

So, the redundancy consequences anomaly that we are getting into, so it could be better to decompose this to say that I make this orthogonal; I keep the child information with id and I keep the phone number information in the id separately. So, if I do that, then I can decompose it in this manner and if I decompose that, this have just shown that if you are dividing that table in two parts. So, naturally these are not required, neither are these required. So, these are the entries that I get and you can convince yourself that you can actually do a lossless join to get back the information.

So, BCNF not necessarily give you good designs and we will see later on that there are other normal forms which can be used to improve on BCNF.

(Refer Slide Time: 15:05)



PPD

- Decomposition Using Functional Dependencies
- Functional Dependency Theory

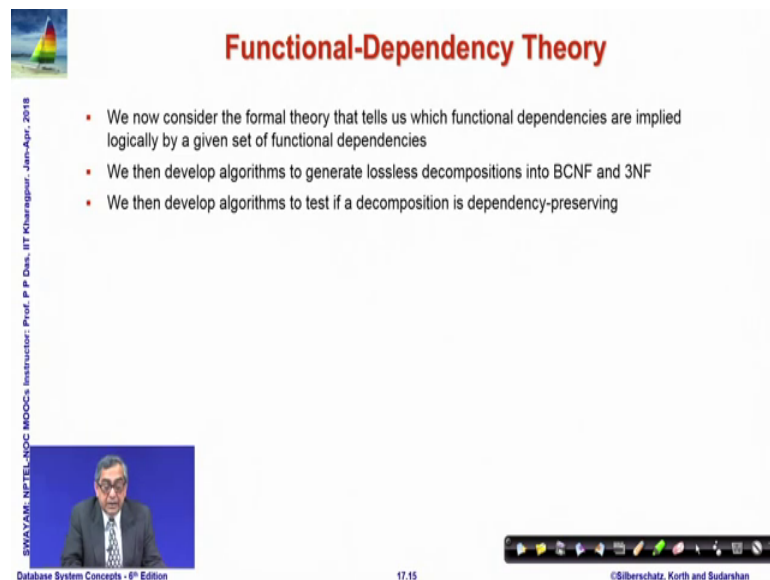
FUNCTIONAL DEPENDENCY THEORY

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 17.14 ©Silberschatz, Korth and Sudarshan

Now, let us for formally getting into how do we convert decompose relation into a third normal form and how we assess that we need to understand more of the functional dependencies.

(Refer Slide Time: 15:20)



Functional-Dependency Theory

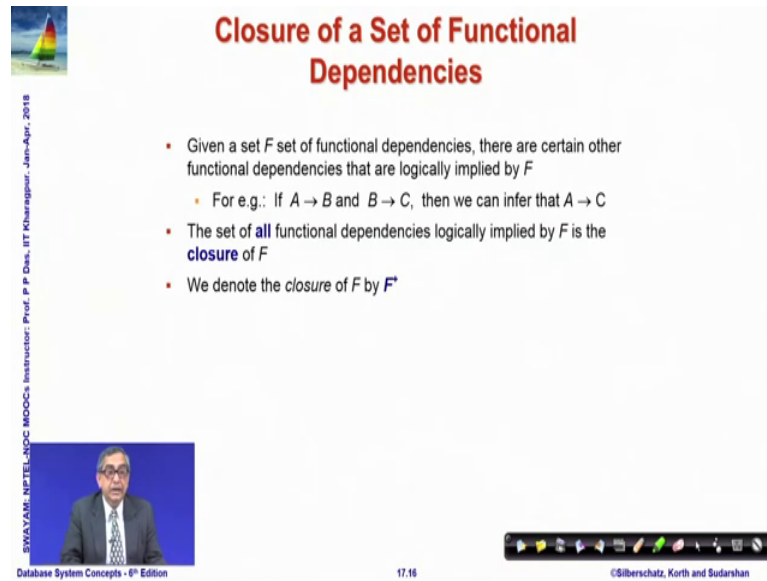
- We now consider the formal theory that tells us which functional dependencies are implied logically by a given set of functional dependencies
- We then develop algorithms to generate lossless decompositions into BCNF and 3NF
- We then develop algorithms to test if a decomposition is dependency-preserving

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 17.15 ©Silberschatz, Korth and Sudarshan

So, we will consider now a little bit of formal theory on them and then, develop algorithms that can generate lossless join decomposition into BCNF and 3 NF and we will also create algorithm to test if decomposition preserves the dependency.

(Refer Slide Time: 15:39)



The slide features a title 'Closure of a Set of Functional Dependencies' in red. On the left, there is a small image of a sailboat and a vertical text string: 'SWAYAM - NPTEL_NOC MOOCs Instructor: Prof. P P Das, IIT Khargpur, Jan-Apr, 2018'. Below this is a video thumbnail of the instructor. The main content is a bulleted list. At the bottom, there is a navigation bar with icons and the text '©Silberschatz, Korth and Sudarshan'.

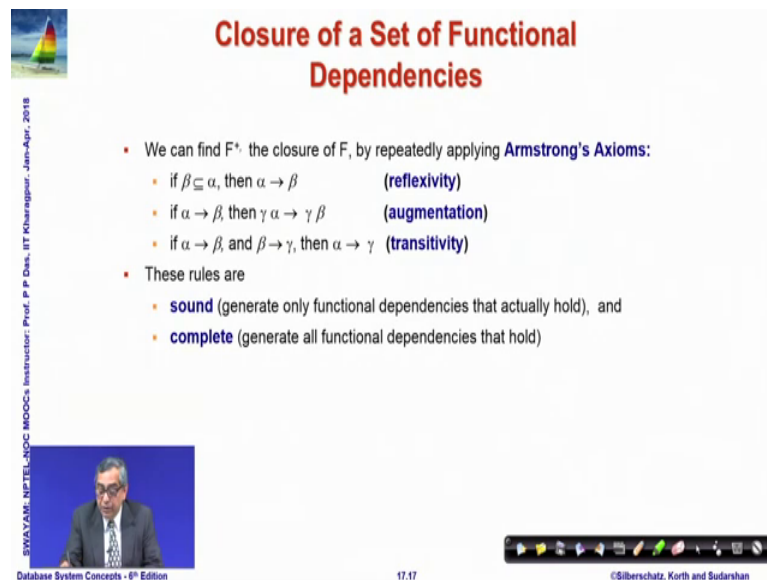
Closure of a Set of Functional Dependencies

- Given a set F set of functional dependencies, there are certain other functional dependencies that are logically implied by F
 - For e.g.: If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of **all** functional dependencies logically implied by F is the **closure** of F
- We denote the *closure* of F by F^+

Database System Concepts - 9th Edition 17.16 ©Silberschatz, Korth and Sudarshan

So, just quickly to recap we have already introduced the closure set of a functional dependencies. It is all dependencies that are logically implied by it. Now, the question certainly is how do I give a set? How do I compute this closer set?

(Refer Slide Time: 15:57)



The slide features a title 'Closure of a Set of Functional Dependencies' in red. On the left, there is a small image of a sailboat and a vertical text string: 'SWAYAM - NPTEL_NOC MOOCs Instructor: Prof. P P Das, IIT Khargpur, Jan-Apr, 2018'. Below this is a video thumbnail of the instructor. The main content is a bulleted list. At the bottom, there is a navigation bar with icons and the text '©Silberschatz, Korth and Sudarshan'.

Closure of a Set of Functional Dependencies

- We can find F^+ the closure of F , by repeatedly applying **Armstrong's Axioms**:
 - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ (**reflexivity**)
 - if $\alpha \rightarrow \beta$, then $\gamma \alpha \rightarrow \gamma \beta$ (**augmentation**)
 - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ (**transitivity**)
- These rules are
 - sound** (generate only functional dependencies that actually hold), and
 - complete** (generate all functional dependencies that hold)

Database System Concepts - 9th Edition 17.17 ©Silberschatz, Korth and Sudarshan

So, to do this we make use of three rules known by Armstrongs Axiom named after the person who first observed them. So, the first rule is reflexivity which says that if beta is a subset of alpha, then alpha determines beta. Always alpha functionally determines beta. So, this is basically reflexivity you can see is a different way of saying specifying about

trivial dependencies. Next comes important thing augmentation which says that if alpha determines beta, then gamma alpha where gamma is some set of attributes in R.

Then, gamma alpha will functionally determine gamma beta which is very easy to see because alpha determines beta means two tuples who match on alpha will necessarily match on beta. Now, if that happens and whatever is gamma if two tuples match on gamma and alpha, then certainly they will match and gamma and beta because alpha determines beta tells me that they will match on beta and gamma is the same set of attributes. So, augmentation also is easy. Then, we have transitivity which we earlier saw also if alpha determines beta and beta determines gamma, then obviously alpha determines gamma.

So, these are the foundational rules observed which can be made used to compute the closure of the set of functional dependencies. Now, these rules as they say this is more for you know understanding the theory better. These rules are sound as well as complete. Soundness mean that if I use these rules repeatedly in a set of dependencies, then it generates functional dependencies all of which actually hold. So, it will never generate a functional dependency which is not correct, which will not hold and the second it is complete which means that if I keep on using these rules, then all functional dependencies that can at all hold will eventually get generated.

(Refer Slide Time: 18:06)

Example

- $R = (A, B, C, G, H, I)$
 $F = \{ A \rightarrow B$
 $A \rightarrow C$ ✓
 $CG \rightarrow H$
 $CG \rightarrow I$ ✓
 $B \rightarrow H\}$
- Some members of F^+
 - $A \rightarrow H$
 - by transitivity from $A \rightarrow B$ and $B \rightarrow H$
 - $AG \rightarrow I$ ✓ ✓
 - by augmenting $A \rightarrow C$ with G , to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$
 - $CG \rightarrow HI$
 - by augmenting $CG \rightarrow I$ to infer $CG \rightarrow CGI$, and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P P Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 8th Edition 17.18 ©Silberschatz, Korth and Sudarshan

So, that is a very strong result and that is what leads to say the following example. So, when we are trying to compute the functional, the closure of the function set of functional dependencies here. So, there are six attributes in the set. There are six different functional dependencies and we identify some members of the closure. For example, we can see that A functionally determines B and B functionally determine H. So, transitivity clearly shows that A will functionally determine H, very clear. So, in the closure that must be there.

Similarly, we can see that A functionally determines C. Now, if we augment it with G, that is put G on both sides, then AG functionally determines CG and we know that CG functionally determines I. So, if we combine these two by transitivity, then we can get a new functional dependency which has AG functionally determines I. So, in this manner you can do the next one also and you can try to infer several other functional dependencies that can be inferred by different applications of the Armstrong's axiom, the three rules in any multiple different ways.

(Refer Slide Time: 19:36)

Procedure for Computing F^+

- To compute the closure of a set of functional dependencies F :

$F^+ = F$

repeat

- for each** functional dependency f in F^+
 - apply reflexivity and augmentation rules on f
 - add the resulting functional dependencies to F^+
- for each** pair of functional dependencies f_1 and f_2 in F^+
 - if f_1 and f_2 can be combined using transitivity
 - then** add the resulting functional dependency to F^+

until F^+ does not change any further

NOTE: We shall see an alternative procedure for this task later

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018
Database System Concepts - 6th Edition 17.18 ©Silberschatz, Korth and Sudarshan

So, to get the closure what we need to do is, now very simple is certainly we will have a repetitive algorithm to get the closure. The first algorithm will start with the set of functional dependencies that we have. So, the closure must include the given set of functional dependencies. So, F^+ must have F . So, let us start with initial value of F^+ as F , then for every functional dependency is F^+ . This is what we keep on

repeating. Look at the outer loop, every functional dependency that we have. F plus now will apply reflexivity and augmentation and add the resulting functional dependency in F plus. It is possible that the same functional dependency gets generated and added multiple times does not matter. F plus is a set. It will naturally eliminate duplicates.

Then, for each pair of functional dependencies because reflexivity and augmentation applies to one functional dependency only, but transitivity applies to two functional dependencies. So, for every pair of functional dependencies, we check whether they can be combined by transitivity. If they do, then the transitive closure of the transitive functional dependency that arise out of that is also added to F plus and mind you more and more functional dependencies you add, there are more and more opportunities to apply the Armstrongs Axiom rules and newer functional dependencies will continue to get added, but eventually you reach a point where F plus does not change any further and when that is achieved, we know that the functional, the closure of the functional dependencies have been obtained and that is our final set.

(Refer Slide Time: 21:32)

Closure of Functional Dependencies (Cont.)

- Additional rules:
 - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds (**union**)
 - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds (**decomposition**)
 - If $\alpha \rightarrow \beta$ holds and $\beta\gamma \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudotransitivity**)

The above rules can be inferred from Armstrong's axioms

Database System Concepts - 9th Edition 17:20 ©Silberschatz, Korth and Sudarshan

We can also observe that based on the rules of Armstrong, the Armstrongs Axioms we can also generate lot of derived rules. Some of those are shown here. For example, if A determines, if alpha determines beta holds and if alpha determines gamma, that also holds, then alpha determines beta and gamma together. This is called the union set. So, if there are two functional dependencies which are the same left hand side set of attribute,

then we can take the union of their right hand side attributes and that functional dependency will hold obviously, it is trivial to prove this.

If alpha determines beta gamma, then alpha determines beta holds and alpha determines gamma holds. This is called decomposition. So, kind of the other side of the union which also is trivial because alpha determines beta; Gamma says if two tuples match on alpha, they match on beta as well as gamma attributes. So, obviously you take the first part, you get alpha determines beta. You take the second part of the observation, you get alpha determines gamma. So, that is a composition rule. The third is interesting. It is called the pseudo transitivity which says that alpha determines beta if that holds and gamma beta determines delta if that holds, then alpha gamma will determine delta which is not difficult to get because if this holds, then I can augment gamma on both sides. I get beta gamma and then, I have given beta gamma determines delta. So, if I combine these two in terms of transitivity, I get alpha gamma determining delta.

So, this is called pseudo transitivity because here you are adding another attribute in the transitivity. So, often times it becomes easier to make use of these additional rules to quickly get to the closer set.

(Refer Slide Time: 23:45)

Closure of Attribute Sets

- Given a set of attributes α , define the **closure** of α under F (denoted by α^+) as the set of attributes that are functionally determined by α under F
- Algorithm to compute α^+ , the closure of α under F

```

result :=  $\alpha$ ; ✓
while (changes to result) do
  for each  $\beta \rightarrow \gamma$  in  $F$  do
    begin
      if  $\beta \subseteq \text{result}$  then result := result  $\cup$   $\gamma$ 
    end
  
```

The diagram illustrates the algorithm with handwritten annotations. It shows a set of attributes α and a set of functional dependencies F . The dependencies are $\alpha \rightarrow \beta$, $\beta \rightarrow \gamma$, and $\alpha \rightarrow \gamma$. The algorithm starts with $\text{result} = \alpha$. It then iterates through the dependencies. For $\alpha \rightarrow \beta$, since $\alpha \subseteq \text{result}$, β is added to the result. For $\beta \rightarrow \gamma$, since $\beta \subseteq \text{result}$, γ is added to the result. The final result is $\alpha \cup \beta \cup \gamma$.

©Silberschatz, Korth and Sudarshan

So, given a set of attributes we also compute the closure of a set of attributes. This is a second concept we have seen how to give the set of functional dependencies, how to compute the closure of the functional dependencies. Now, we are given a set of attributes

and we want to define the closure of this set of functional, this set of attributes under the set of functional dependencies and as the closure of functional dependencies F is denoted by F^+ , the closure of a set of attributes α under F is denoted by α^+ . So, this set of closure attributes of α is a set of attributes that are functionally determined by α under F . So, all set of attributes that are functionally determined by α under the set of functional dependencies is member of α^+ .

So, the following simple algorithm can compute the closure naturally. Initially let us say the result is the final closure set. So, initially we can say that result can be initialized with α because certainly the whole of α would necessarily belong to α^+ by the reflexivity condition, then for each functional dependency $\beta \rightarrow \gamma$, we check if β is a subset of the result. If β is a subset of the current set of attributes that form result which mean that α functionally determines β , it will have to because result is the set of all attributes that α functionally determines. So, if β is a subset of the result, then necessarily α functionally determines β is a consequence of this and we know that this is their β functionally determines γ combined by transitivity.

So, I know α functionally determines γ . If function α functionally determines γ , then it must get into the result and this is exactly what the statement is saying that take result and add α , add γ , the set of attributes γ to the result. How long should you do that? Naturally you will do that as long as over a full iteration of functional dependencies in F , if there is no change to the result, then you know that all future iterations will have no change. So, you reach a fixed point and you declare that the closure of the set of attributes have been obtained.

(Refer Slide Time: 26:44)

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- $(AG)^+$
 1. result = AG
 2. result = ABCG ($A \rightarrow C$ and $A \rightarrow B$)
 3. result = ABCGH ($CG \rightarrow H$ and $CG \subseteq AGBC$)
 4. result = ABCGHI ($CG \rightarrow I$ and $CG \subseteq AGBCH$)

AG → ABCGHI

SWAYAM - NPTEL_NOC MOOCs Instructor: Prof. P P Das, IIT Khargpur, Jan-Apr, 2018

Database System Concepts - 8th Edition

17.22

©Silberschatz, Korth and Sudarshan

Now, this closure information is very interesting and we just show an example here based on the same set of attributes and same set of functional dependencies.

So, we are trying to find the closure of the set of attributes AG. So, AG plus initially it will be AG. Now, since A functionally determines C, so given that I can say that C will get included in this set in the same iteration. If I look at A functionally determines B, so B will get included in this set. So, after this first iterative loop I will have the result as ABCG. If ABCG is there and I am looking at the next iteration, then CG functionally determines H. So, H comes into the set because CG is a subset of that I comes into the set because CG functional determines I and at this point, it eventually ends in this case. In this particular example, you can see that all attributes have got included. So, you can see that it immediately gives you another information as a byproduct of the closure that closure of AG is all attributes which mean that AG is a key. It has to be a key because AG functionally determines all attributes now.

So, what is the meaning of AG plus being this? So, if the meaning of this is AG functionally determines the set of attribute ABCGHI, right, so we will see that this closure set has a lot of valuable information in this.

(Refer Slide Time: 28:31)

Example of Attribute Set Closure

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- $(AG)^*$
 1. $result = AG$
 2. $result = ABCG$ ($A \rightarrow C$ and $A \rightarrow B$)
 3. $result = ABCGH$ ($CG \rightarrow H$ and $CG \subseteq AGBC$)
 4. $result = ABCGHI$ ($CG \rightarrow I$ and $CG \subseteq AGBCH$)
- Is AG a candidate key?
 1. Is AG a super key?
 1. Does $AG \rightarrow R$? $==$ Is $(AG)^* \supseteq R$
 2. Is any subset of AG a superkey?
 1. Does $A \rightarrow R$? $==$ Is $(A)^* \supseteq R$
 2. Does $G \rightarrow R$? $==$ Is $(G)^* \supseteq R$

Database System Concepts - 8th Edition 17.22 ©Silberschatz, Korth and Sudarshan

So, we can say that AG is a candidate key and because of this, we can also check whether AG is a super key or not. All that we need to do is drop some member from AG, we drop G and check whether a functional determines R which means we check whether A plus is equal to R or not. We check we drop a from a g and check whether G functionally determines R which means G plus has to be equal to R and by that we can easily determine whether the set of attributes is a key or not.

(Refer Slide Time: 29:14)

Uses of Attribute Closure

There are several uses of the attribute closure algorithm:

- Testing for superkey:
 - To test if α is a superkey, we compute α^* and check if α^* contains all attributes of R .
- Testing functional dependencies
 - To check if a functional dependency $\alpha \rightarrow \beta$ holds (or, in other words, is in F^+), just check if $\beta \subseteq \alpha^*$.
 - That is, we compute α^* by using attribute closure, and then check if it contains β .
 - Is a simple and cheap test, and very useful
- Computing closure of F
 - For each $\gamma \subseteq R$, we find the closure γ^* , and for each $S \subseteq \gamma^*$, we output a functional dependency $\gamma \rightarrow S$.

Database System Concepts - 8th Edition 17.23 ©Silberschatz, Korth and Sudarshan

So, there are several ways. The attribute closure can be used as we have just seen. It helps you determine whether something is a super key. We can check for testing functional dependencies because if we have to check whether a functional dependency α determines β hold, all that we will have to do is to compute the closure of the set of attributes α that is α^+ and check whether β is a subset of that. If it is, then certainly holds. If it is not, then it does not hold. So, it is simple and useful test that can be made use of.

So, it can also be used in computing the closure of F that for example, for every subset γ of R , if we find γ^+ that is a closure of the set of attributes of γ and then, for each subset of γ^+ , we know that there is a functional dependency γ determining S which is just is the same statement being made in you know or in different forms and the closure of attributes is a very nice concept which help you play around in this multiple ways and we will see subsequently many of the algorithms for normalization.

(Refer Slide Time: 30:35)

Module Summary

- Discussed issues in 'good' design in the context of functional dependencies
- Introduced the theory of functional dependencies

SWAYAM: NPTEL-NOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 17.24 ©Silberschatz, Korth and Sudarshan

How they make effective use of this closure set, notion of both closure of functional dependencies and in very practical implementation algorithms, the closure of attributes. So, to summarize this module, we have discussed issues for, issues in the good design in the context of functional dependencies and in the process, we have also extended the

theory of functional dependencies and we will continue it this in the next module to get more insight into the algorithms that actually work with the functional dependencies.