

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

Lecture - 16
Relational Database Design

Welcome to Module 16 of Database Management Systems till the last module which closed with the third week.

(Refer Slide Time: 00:31)

PPD

Week 03 Recap

- **Module 11: Advanced SQL**
 - Accessing SQL From a Programming Language
 - Functions and Procedural Constructs
 - Triggers
- **Module 12: Formal Relational Query Languages**
 - Relational Algebra
 - Tuple Relational Calculus (Overview only)
 - Domain Relational Calculus (Overview only)
 - Equivalence of Algebra and Calculus
- **Module 13: Entity-Relationship Model/1**
 - Design Process
 - E-R Model
- **Module 14: Entity-Relationship Model/2**
 - E-R Diagram
 - E-R Model to Relational Schema
- **Module 15: Entity-Relationship Model/3**
 - Extended E-R Features
 - Design Issues

SWAYAM: NPTEL-NOC MOOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-April, 2018

Database System Concepts - 6th Edition

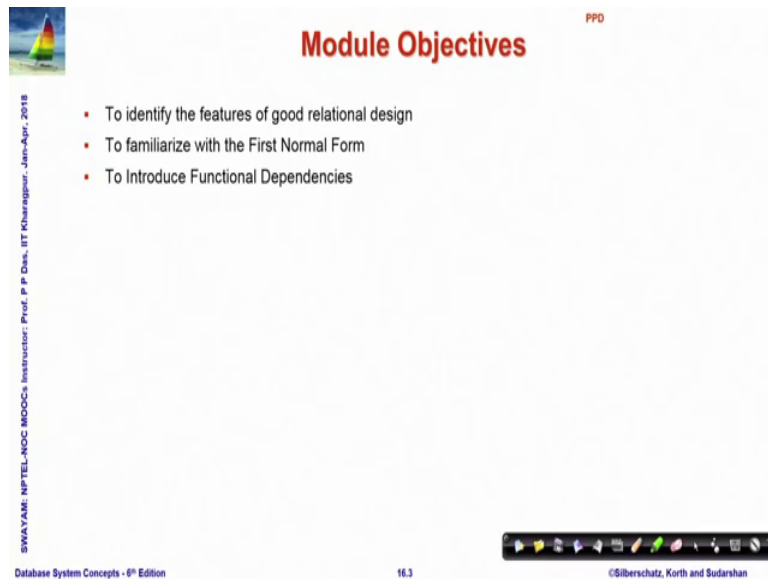
16.2

©Silberschatz, Korth and Sudarshan

Specifically in the third week, we talked about certain advanced features of SQL and the formal query language in terms of relational and algebra and calculi and then, we talked in a depth in terms of the entity relationship model, the first basic conceptual level representation of the real world that we can do in terms of designing a system.

Now, our next task would be to take it to more proper complete relational database design and this will have a lot of theory at different levels that we need to understand. We will slowly develop that and this discussion will span 5 modules that is we will take the whole week to complete.

(Refer Slide Time: 01:25)



PPD

Module Objectives

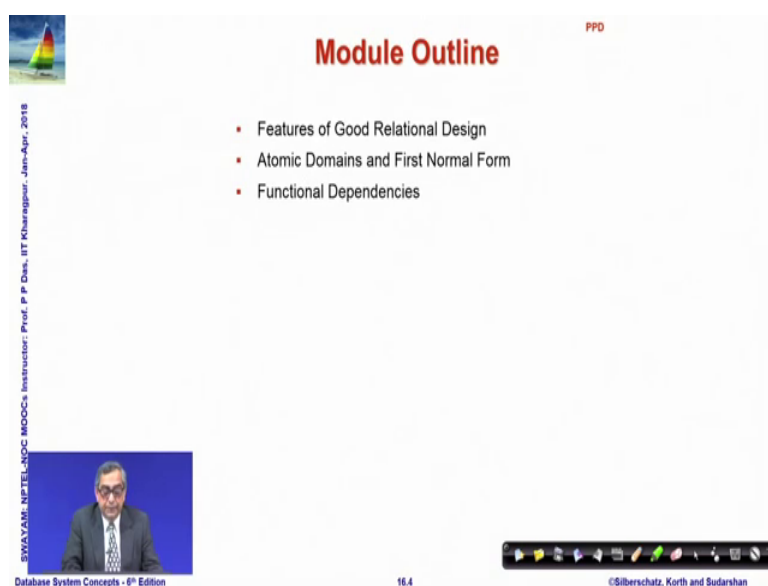
- To identify the features of good relational design
- To familiarize with the First Normal Form
- To Introduce Functional Dependencies

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

Database System Concepts - 6th Edition 16.3 ©Silberschatz, Korth and Sudarshan

So, the objective of the current module, the first of the Relational Design Module is to identify features of good relational design having done the year module. We have yeah we do the year model, we have the entity sets relationships, we convert them to schema. We have seen how to do that and immediately we have some design, but the question is, is it a good design. So, we will discuss about what are the features of a good design and then, we will introduce the formal definition of what is first normal form and we will introduce a very critical concept of relational database design, the functional dependencies.

(Refer Slide Time: 02:07)



PPD

Module Outline

- 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 16.4 ©Silberschatz, Korth and Sudarshan

These are the module outline for that.

(Refer Slide Time: 02:10)

PPD

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

FEATURES OF GOOD RELATIONAL DESIGN

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

Database System Concepts - 8th Edition 16.5 ©Silberschatz, Korth and Sudarshan

So, to start with the features of good relational design, let us take an example.

(Refer Slide Time: 02:13)

Combine Schemas?

- Suppose we combine *instructor* and *department* into *inst_dept*
- (No connection to relationship set *inst_dept*)

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Redundancy

Anomaly

update
insertion
deletion

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

Database System Concepts - 8th Edition 16.6 ©Silberschatz, Korth and Sudarshan

Suppose we have seen the instructor, relation instructor entity set as a relation. You have seen the department relation. Now, let us consider that if these two were not two separate relations, if they were all kept in a common relation that is all the attributes are kept in the common relation, so earlier if you recall that your instructor relation was this and your department relation was this much. So, if we keep everything together, of course we

are calling it inst dept, but please keep in mind this is not the same inst dept that we discussed in terms of the ER model. This is just putting these two together.

Now, the question is if you look into this data carefully, for example if you look into this particular row, if you look into this particular row and if you look into this particular row, these are rows of instructors who all belong to computer science. Now, earlier we were representing the information of instructor only in this part. So, we just knew that it is computer science and we represent the information of department in this part. So, given a department name say computer science, we knew, where is it located, the building and what budget it has. Now, when we are combined, we will see that naturally since computer science is located in the tailor building, we know that it has a budget of say 100,000. So, all of these records will have this information repeated.

So, this is not a very good situation. This is not a good situation because this kind of situation is typically in database is known as redundancy, that is you have the same data in multiple places. So, what is the consequence of redundancy? For example, there could be different kinds of anomaly when you have redundancy. What is an anomaly? An anomaly is the possibility of certain data getting inconsistent. For example, let us say computer science department moves from tailor building to painter building. Now, what will have to happen if it moves to painter building? Then, I will need to remove this, make it a painter, make this value painter. I have to also do this, make this painter. I have to also do this, make this painter. So, if I have a change, then I will have to make the change at multiple entries. Think about the earlier situation where I just had these three in my department relation, then naturally computer centered only one row and therefore, this change, this update could be done at only one place.

So, it is not only that if while doing this in case of this redundancy, I have to do this multiple times. It also has the difficulty that if I forget to update any one of them or more of them, then I have inconsistent data. Similarly, if I want to insert a new value, I will have to do that for all this redundant information. If I have to delete say for some reason let say the university decides to wind up the Physics department, then I have to delete all these rows which have physics as an entry and the consequence of that is the department is deleted, but as a consequence of that I will delete the whole row and therefore, I will not only remove the department, but I will also remove the corresponding instructor who was enrolled for that department.

So, this kind of redundancy can lead to different kinds of anomalies in a database design. On the other hand, if you look at, well why I am complicating the whole situation? We have already had a good design in terms of where these anomalies were, not their department, were separate instructor was separate. In that case, the situation is that to answer some of the queries, I may have to do a very expensive joint operation. For example, if I want to know if Einstein wants to know what is the budget of his department that cannot be found out from the earlier instructor database, instructor relation which had only these fields.

So, I have to pick up Einstein from here, do a join based on the department name, dept name with the department table department relation and then only, I will be able to find out that an Einstein belongs to Physics. Physics has a budget of 70000. So, Einstein's department has a budget 70000. So, there is a tradeoff between how much if data information you make redundant and lead to different anomalous situations or how much data you optimize in the representation, but get into the possible situation of having a higher cost in terms of answering your queries.

(Refer Slide Time: 07:49)

Combine Schemas?

- Suppose we combine *instructor* and *department* into *inst_dept*
 - (No connection to relationship set *inst_dept*)
- Result is possible repetition of information (*building* and *budget* against *dept_name*)

ID	name	salary	dept_name	building	budget
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

Database System Concepts - 6th Edition 16.6 ©Silberschatz, Korth and Sudarshan

So, this is one of the core design issues that we will start with. So, let us look into some more.

(Refer Slide Time: 07:54)



A Combined Schema Without Repetition

- Consider combining relations
 - $sec_class(sec_id, building, room_number)$ and
 - $section(course_id, sec_id, semester, year)$
 into one relation
 - $section(course_id, sec_id, semester, year, building, room_number)$
- No repetition in this case



Of these examples, let us say we look into another combined combination of schema. Suppose section is a relation which have the sections of a course which give the section id semester year and say section class is another relation which tell me for a section id, what is the building and room number where it is located. So, if we have this kind of relations combined into a common relation, then I have all of these coming from the section and this and these coming from the section class, but we can see that there is no repetition or redundant information in this case.

So, it is note that combining schemas is necessarily always bad in terms of repetition or in terms of redundancy. So, different situations will have to be assessed.

(Refer Slide Time: 08:57)



What About Smaller Schemas?

- Suppose we had started with $inst_dept$. How would we know to split up (**decompose**) it into $instructor$ and $department$?
- Write a rule "if there were a schema ($dept_name, building, budget$), then $dept_name$ would be a candidate key"
- Denote as a **functional dependency**:

$$dept_name \rightarrow building, budget$$
- In $inst_dept$, because $dept_name$ is not a candidate key, the building and budget of a department may have to be repeated.
 - This indicates the need to decompose $inst_dept$
- Not all decompositions are good. Suppose we decompose $employee(ID, name, street, city, salary)$ into

$$employee1 (ID, name)$$

$$employee2 (name, street, city, salary)$$
- The next slide shows how we lose information -- we cannot reconstruct the original $employee$ relation -- and so, this is a **lossy decomposition**.

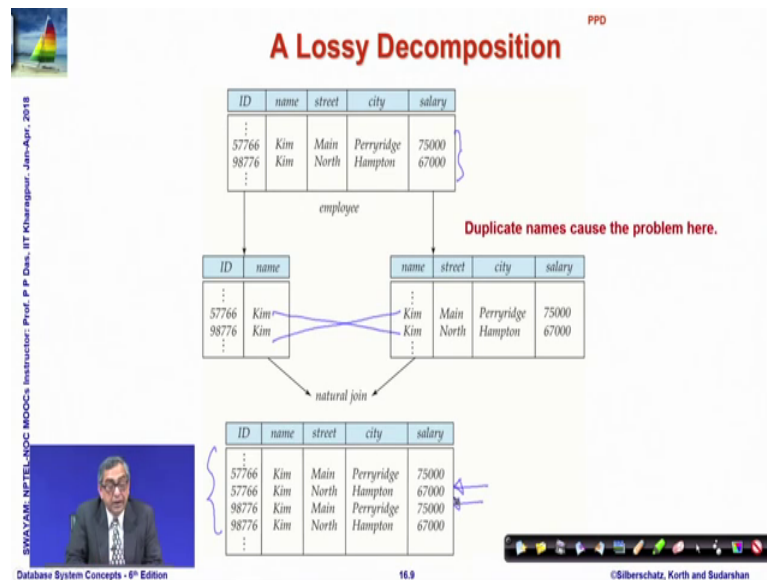
So, if we want to look at the other side that if we just as I said that if we make the schema smaller, so that we avoid redundancy and then, what we see that 12 from the combined inst dept relationship that we saw. So, let me just show you once more. So, this is if we look at the inst dept, then in this we can we know that from the earlier information about the department relationship that department name is a key, is a primary key of the relation which has department name, building and budget. What is the consequence of being a primary key? If it is a primary key, then no two records can match on the department name and be different in terms of the building and the budget.

If two records are there which have the same department name, they must be identical. So, they are distinguishable completely by that. So, let us see what is the consequence of this. So, we are saying that we write it as a rule that if there is a schema department, name, building, budget, then department name would be a candidate key and we write this observation that if two records match on the department name, they must match on the building and budget and very loosely, we will come to the formal definition. Very loosely we call this the functional dependency. We say that the building and budget is functionally dependent on the department name and that is a situation where we can split this inst dept and create a smaller relationship because department name is not a candidate key in the inst dept. It does not decide the records of inst dept uniquely.

So, since it does not, so when the values of this key, this attribute department name is duplicated or triplicated, the values of the building and budget are repeated and we have the redundancy. So, this is a situation, very common situation which is indicative of the fact that we need a decomposition into smaller , but at the same time we can also observe, I mean let us take a different example. If we are thinking that decomposition is the panacea of solving these kind of redundancy and related problems, then let us try to see a different relationship employee which has id, name, street, city, salary and we want to make it smaller and want to make two relations id and name and name, city street, salary.

So, if we do that, then how do we get the salary for a particular id? We will naturally have to join these two relations in terms of the common attribute name. We have seen that in the query and the question is when I do this joint, do I get back the original information or I lose some information.

(Refer Slide Time: 12:42)



Look at an example. So, here is an example of the combined instance and I have two different ids, but incidentally the names are same. The names of these two distinct employees are same. So, when I decompose, I get this relation which shows id and name. I get this relation which is against the name shows this, but when I try to join them by natural join, I not only get the combination of this with this which is what I need, but I also get this combination. So, if I say this is what I get as well in terms of natural join, this is what I get as well in terms of the natural join which are really not there in the original relation.

So, you can see that in the natural join, I get four records, I get four rows whereas, in the original one I had only two rows. So, I get some entries which are actually erroneous. These are not there in the database. So, this is when this happens. We say that we have loss of information and such joins are said to be lossy joins. So, when we decompose, we need to make sure that our joins are lossless in nature; otherwise that is not a good design.

(Refer Slide Time: 14:08)

Example of Lossless-Join Decomposition

- Lossless join decomposition
- Decomposition of $R = (A, B, C)$
 $R_1 = (A, B)$ $R_2 = (B, C)$

A	B	C
α	1	A
β	2	B

r

A	B
α	1
β	2

$\Pi_{A,B}(r)$

B	C
1	A
2	B

$\Pi_{B,C}(r)$

$\Pi_A(r) \bowtie \Pi_B(r)$

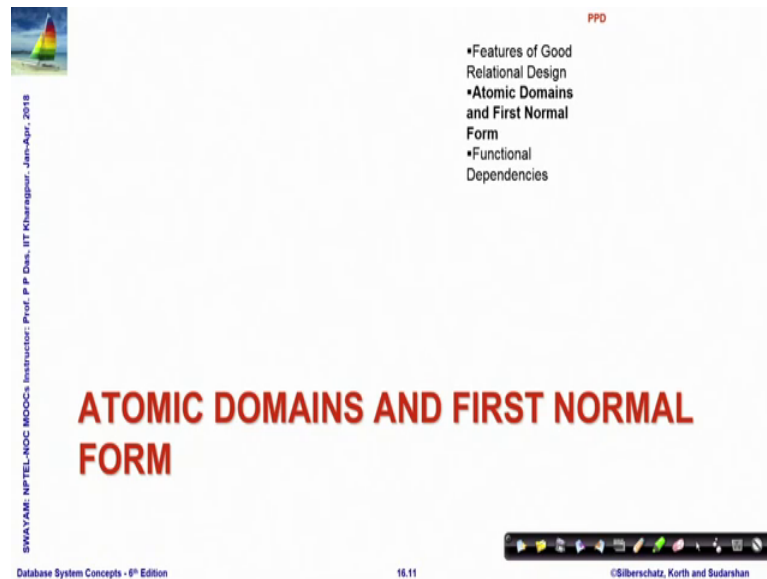
A	B	C
α	1	A
β	2	B

SWAMYAM, NIP TEL-ANOC MOOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018
 Database System Concepts - 6th Edition 16.10 ©Silberschatz, Korth and Sudarshan

So, you can see this is again a hypothetical example which shows three attributes in relation having three attributes. You have decomposed it into two relations having two attributes each and we have shown an instance and in this case, it shows that when I take the join, the original information I am sorry, wait.

When I take the join, the original information is completely retrieved. I get back the same table and when that happens, I say that the join is lossless. So, what we need to understand is on one side there is a need to decompose relations into smaller relations to reduce redundancy and while we do that, we will also have to keep this in mind that the smaller relations must be composable through certain natural join procedure to the original relation, and I must get back that original relation, otherwise I have a lossy join which is not acceptable. Also, the decomposition will have the costs of doing natural join every time I want to answer those queries.

(Refer Slide Time: 15:35)



PPD

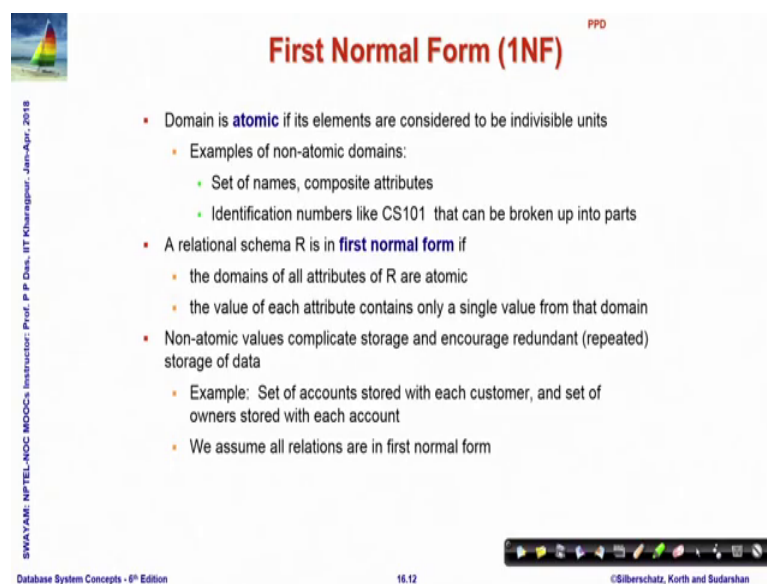
- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Functional Dependencies

ATOMIC DOMAINS AND FIRST NORMAL FORM

Database System Concepts - 9th Edition 16.11 ©Silberschatz, Korth and Sudarshan

The next that we look at is the way the relationships are categorized as first normal form.

(Refer Slide Time: 15:45)



PPD

First Normal Form (1NF)

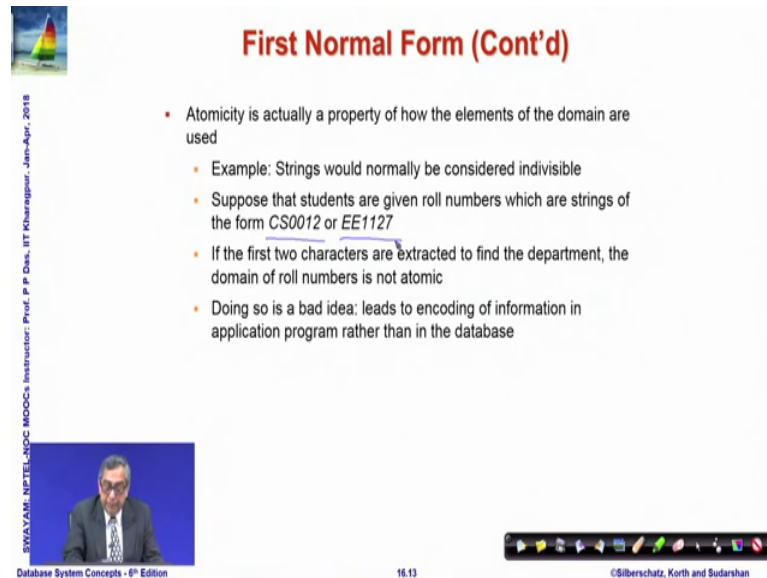
- Domain is **atomic** if its elements are considered to be indivisible units
 - Examples of non-atomic domains:
 - Set of names, composite attributes
 - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **first normal form** if
 - the domains of all attributes of R are atomic
 - the value of each attribute contains only a single value from that domain
- Non-atomic values complicate storage and encourage redundant (repeated) storage of data
 - Example: Set of accounts stored with each customer, and set of owners stored with each account
 - We assume all relations are in first normal form

Database System Concepts - 9th Edition 16.12 ©Silberschatz, Korth and Sudarshan

We consider that the domains of attributes are atomic if they are indivisible. So, anything that is a number string and so on is considered to be atomic and we say a relational schema is in its first normal form if the domains of all attributes are atomic and all attributes single value, there is no multi value attribute. If these conditions are satisfied, then we will say that every relate that relational schema is in its first normal form. So, we will slowly understand the purpose of defining such normal forms, but let us initially understand the definition. So, if we have attributes which are composite in nature,

naturally my relationship, my relational schema is not in first normal form if we have attributes which are multiple valued, it is not so.

(Refer Slide Time: 16:44)



First Normal Form (Cont'd)

- Atomicity is actually a property of how the elements of the domain are used
 - Example: Strings would normally be considered indivisible
 - Suppose that students are given roll numbers which are strings of the form CS0012 or EE1127
 - If the first two characters are extracted to find the department, the domain of roll numbers is not atomic
 - Doing so is a bad idea: leads to encoding of information in application program rather than in the database

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


Database System Concepts - 9th Edition

16.13

©Silberschatz, Korth and Sudarshan

So, if we say that we have possible values are like this, then if we just treat them as strings, then the corresponding relational schema is in first normal form, but if we say that from this string we can extract the first two characters which is CS which tells me what is a department. The next four characters gives me a number, the serial number of the particular student in the role. Then I am not actually using an atomic domain because my domain needs to be interpreted separately than just being a value. So, these are not parts of what can be a first normal form.

(Refer Slide Time: 17:28)



First Normal Form (Cont'd) PPD

- The following is not in 1NF

Customer

Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633


- A telephone number is composite
- Telephone number is multi-valued

Source: https://en.wikipedia.org/wiki/First_normal_form

Database System Concepts - 9th Edition 16.14 ©Silberschatz, Korth and Sudarshan

So, I have given some examples of what is not and what is first normal form. So, this is an example where at the telephone number field exists and there can be multiple telephone numbers. So, this is not in first normal form because the telephone number itself is composite because it has different components and also, you can have multiple telephone number. So, this relation is not in the first normal form.

(Refer Slide Time: 17:54)



First Normal Form (Cont'd) PPD

- Consider:

Customer

Customer ID	First Name	Surname	Telephone Number1	Telephone Number2
123	Pooja	Singh	555-861-2025	192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53	182-929-2929
789	John	Doe	555-808-9633	

- Is in 1NF if telephone number is not considered composite
- However, conceptually, we have two attributes for the same concept
 - Arbitrary and meaningless ordering of attributes
 - How to search telephone numbers
 - Why only two numbers?

Source: https://en.wikipedia.org/wiki/First_normal_form

Database System Concepts - 9th Edition 16.15 ©Silberschatz, Korth and Sudarshan

What you can do? You can separate out these phone numbers into two different attributes; Telephone number 1 and 2. Even then it is not exactly in first normal form because you do not know in which order they should be handled. If you have to search for a telephone number, then you will have to search multiple attributes which are

conceptually same and then, the question is why only two attributes. Cannot anybody have 3 phone numbers, 7 phone numbers and so on. So, this is really not a good option.

(Refer Slide Time: 18:26)

First Normal Form (Cont'd)

- Is the following in 1NF?

Customer			
Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025
123	Pooja	Singh	192-122-1111
456	San	Zhang	182-929-2929
456	San	Zhang	(555) 403-1659 Ext. 53
789	John	Doe	555-808-9633

- Duplicated information
- ID is no more the key. Key is (ID, Telephone Number)

Source: https://en.wikipedia.org/wiki/First_normal_form

Database System Concepts - 9th Edition 16.16 ©Silberschatz, Korth and Sudarshan

So, the other way could be that for every telephone number, you introduce a separate row. Once you do that you already know you have redundancy and you have possibilities of varied kinds of anomalies that could happen.

(Refer Slide Time: 18:40)

First Normal Form (Cont'd)

- Better to have 2 relations:

Customer Name			Customer Telephone Number	
Customer ID	First Name	Surname	Customer ID	Telephone Number
123	Pooja	Singh	123	555-861-2025
456	San	Zhang	123	192-122-1111
789	John	Doe	456	(555) 403-1659 Ext. 53
			456	182-929-2929
			789	555-808-9633

- One-to-Many relationship between parent and child relations
- Incidentally, satisfies 2NF and 3NF

Source: https://en.wikipedia.org/wiki/First_normal_form

Database System Concepts - 9th Edition 16.17 ©Silberschatz, Korth and Sudarshan

So, one way it could be achieved is we follow the principle that we had seen in ER modelling that this multivalued dependency can be represented in terms of a separate

relation where against the customer id we just keep the telephone number. So, we can keep multiple of them and we take that out from the customer name. So, one to many relationship between the parent and the child, between the customer name and telephone number, every customer may have more than one telephone number is possible and that makes it 2 NF relation, first normal form relation and we will later on see that it also is 2 NF and 3 NF, but that is a future story.

(Refer Slide Time: 19:29)

PPD

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

FUNCTIONAL DEPENDENCIES

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

Database System Concepts - 6th Edition

16.18

©Silberschatz, Korth and Sudarshan

Now, finally we come to the core of what the mathematical formulation which dictates much of the data base, relational database design is known as functional dependencies.

(Refer Slide Time: 19:47)



Goal — Devise a Theory for the Following

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form, decompose it into a set of relations $\{R_1, R_2, \dots, R_n\}$ such that
 - each relation is in good form ✓
 - the decomposition is a lossless-join decomposition


$$R_i = \text{set of attributes}$$
$$R = \bigcup R_i$$



I just talked about little bit of that while talking about department name building and budget. Now, to decide whether a particular relation is good or rather a particular relational scheme is good, we need to check against certain measures and if it is not good, we need to decompose it into a set of relations such that these conditions satisfy that every, each one of these, $R_1 R_2 R_n$. So, I mean if you have you know got rusted, then it is basically R_i is a set of attributes because it is a relational schema. A relational schema is a set of attributes.


So, naturally R will be the union of all of these, R_i the total set of attributes. So, instead of keeping all the information into one relation in one table, we are basically decomposing it into n different schemas.

(Refer Slide Time: 20:42)



Goal — Devise a Theory for the Following

- Decide whether a particular relation R is in "good" form.
- In the case that a relation R is not in "good" form, decompose it into a set of relations $\{R_1, R_2, \dots, R_n\}$ such that
 - each relation is in good form
 - the decomposition is a lossless-join decomposition
- Our theory is based on:
 - functional dependencies
 - multivalued dependencies




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

Database System Concepts - 9th Edition 16.19 ©Silberschatz, Korth and Sudarshan


So, what we need to guarantee is each one of these relation R_1, R_2, R_n is in good form. How do I get back the original relation? Original relation that was represented by all that attributes enough is to take a lossless join. This would take a join and that this decomposition must give me a lossless join. So, to ensure that; we make use of two key ideas more foundationally; functional dependencies and then, multivalued dependencies.

(Refer Slide Time: 21:20)



Functional Dependencies

- Constraints on the set of legal relations
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes
- A functional dependency is a generalization of the notion of a key



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

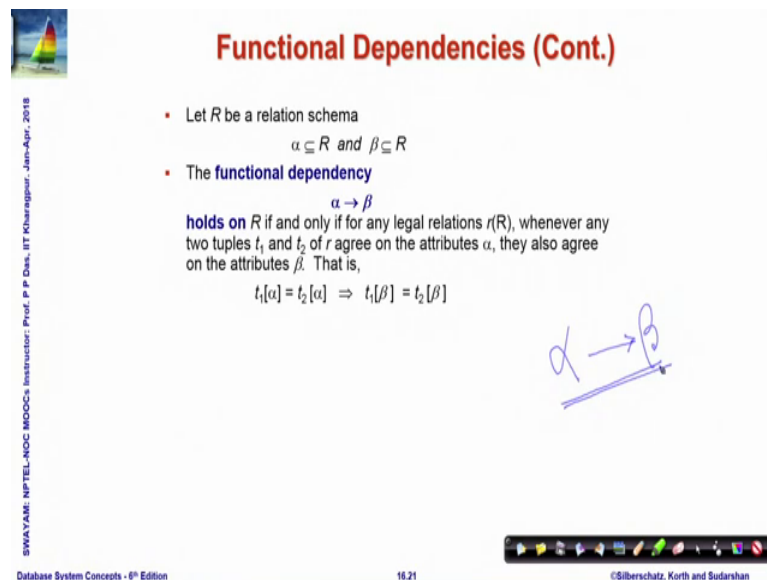
Database System Concepts - 9th Edition 16.20 ©Silberschatz, Korth and Sudarshan

A functional dependency is a constraint on the set of legal relation. So, mind you it is a constraint on the schema and once that constraint is defined, it must hold for all relations that the schema satisfied. So, here we need that the value of certain set of attributes uniquely determined the value of another set of attributes. So, I know the value of three

attributes, I should be able to say that the values of the other four attributes would be fixed. So, you have already seen this notion in terms of key or super key. You have seen that similar type of concept exists where we said a key is a set of attributes, so that if the values of two rows are identical over these set of attributes, then the two people, the two rows must be totally identical.

So, key is something which does a similar thing as a functional dependency, but is more specific. Functional dependencies are generalization.

(Refer Slide Time: 22:30)



Functional Dependencies (Cont.)

- Let R be a relation schema
 $\alpha \subseteq R$ and $\beta \subseteq R$
- The **functional dependency**
 $\alpha \rightarrow \beta$
holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α , they also agree on the attributes β . That is,
$$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

Handwritten diagram showing $\alpha \rightarrow \beta$ with an arrow pointing from α to β .

SWAYAM: NPTEL-NOC MBOCs Instructor: Prof. P P Das, IIT Khargpur, Jan-Apr, 2018
Database System Concepts - 6th Edition 16.21 ©Silberschatz, Korth and Sudarshan

So, let us formally define that let R be a relational schema which means that it is a set of attributes and let us say alpha and beta are two subsets of R , then we write this and note this notation. Alpha is a set of attributes; beta is another set of attributes. Both are subset of the same R and we say alpha functionally determines beta that is if I know the value of a tuple over the attributes of alpha, then the values of that tuple over the attributes of beta would be fixed or in other words, they say that if I have two tuples t_1 and t_2 and their values over the set of alpha attributes are same, then necessarily their values over the set of beta attributes must be same and mind you this is something which is a design constraint. It is not just an incidental property. It is not just the fact that a particular instance of a schema satisfies this, but when you say this is a functional dependency, we need all possible past, present and future instances of the schema must satisfy this.

(Refer Slide Time: 24:03)



Functional Dependencies (Cont.)

- Let R be a relation schema
 $\alpha \subseteq R$ and $\beta \subseteq R$
- The **functional dependency**
 $\alpha \rightarrow \beta$
holds on R if and only if for any legal relations $r(R)$, whenever any two tuples t_1 and t_2 of r agree on the attributes α , they also agree on the attributes β . That is,
 $t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$
- Example: Consider $r(A, B)$ with the following instance of r .

1	4
1	5
3	7

- On this instance, $A \rightarrow B$ does **NOT** hold, but $B \rightarrow A$ does hold.

So, consider this if you take a relation, a schema with an instance as given here between two attributes a and b, then we can say at least given this instance not we still do not know what happens in the whole schema for all instances, but on this instance we can say that functionally determines b does not hold because between the first and the second record, the value of a is same one, but the value of we are different 4 and 5, but we can certainly say that on this instance at least we functionally determines holds because whenever the value if we take any two tuples, their value over b does not at all match. If they does not match, then naturally there is no question of what happens to the value of the tuple over the set of attributes a. So, we will say that b functionally determining a holds in this instance.

(Refer Slide Time: 25:01)



Functional Dependencies (Cont.)

- K is a superkey for relation schema R if and only if $K \rightarrow R$
- K is a candidate key for R if and only if
 - $K \rightarrow R$, and
 - for no $\alpha \subset K$, $\alpha \rightarrow R$

Functional dependencies allow us to express constraints that cannot be expressed using superkeys. Consider the schema:

inst_dept (ID_name, salary, dept_name, building, budget).

We expect these functional dependencies to hold:

dept_name \rightarrow *building*

and *ID* \rightarrow *building*

but would not expect the following to hold:

dept_name \rightarrow *salary*

So, given this definition of functional dependency, now we can have a formal definition of what the super key is. Super key is naturally a subset of attributes which functionally determines the whole set and a candidate key is a super key which is minimal which means that k is a candidate key. If the two conditions have to satisfy this condition say there is a super key that it functionally determines all the attributes and the other condition says minimality that there is no subset α of k , such that α functionally determines r if there exists a subset α of k , the proper subset α of k . So, that α functionally determines r , then k would not be a candidate key. We will have to check for α . So, these two; what we had stated earlier in qualitative terms and now mathematically established. So, we can say that there are different functional dependencies. For example, in stepped combined relation if we look at, then we know that department name functionally determines building functionally.

So, these are functional dependencies that must hold, but certainly we would not expect department name to functionally determine salary. That would be too much, right. So, functional dependencies are facts about the real world that we try to understand from the real world and then, represent in terms of the functional dependency formulation in the database.

(Refer Slide Time: 26:41)



Use of Functional Dependencies

- We use functional dependencies to:
 - test relations to see if they are legal under a given set of functional dependencies.
 - If a relation r is legal under a set F of functional dependencies, we say that r **satisfies** F
 - specify constraints on the set of legal relations
 - We say that F **holds on** R if all legal relations on R satisfy the set of functional dependencies F
- Note: A specific instance of a relation schema may satisfy a functional dependency even if the functional dependency does not hold on all legal instances
 - For example, a specific instance of *instructor* may, by chance, satisfy $name \rightarrow ID$

So, we can use functional dependencies to test relations if they are valid under the set of functional dependencies. So, there could be multiple functional dependencies in the set and if a relation we are using small r here just to remind you that a relation means that a particular instance is legal under a set of functional dependencies. We will say that r satisfies that and if we have that it holds F will be satisfied by all possible instances of a relational schema capital R , then we say F holds on R .

So, a relation satisfies a functional set of functional dependencies and a relational schema for a relational schema, the functional depend set of functional dependencies holds on that schema which means that for all possible past, present and future instances relations, the relations will satisfy the functional dependencies. So, we have for example id . We know id functionally determines $name$ that if the id is distinct, then the $name$ has to be distinct, but we may find that instance where $name$ functionally determines id . So, we can say that $name$ functionally determines id is satisfied by a particular instance where it so happens that there is no two rows where the $name$ is identical, but we cannot, may not be able to infer that as this dependency holding on the relational scheme as a whole because tomorrow we can get another entry, so that two rows might match on the $name$, but could still be distinct entries not matching on id .

(Refer Slide Time: 28:46)



Functional Dependencies (Cont.)

- A functional dependency is **trivial** if it is satisfied by all instances of a relation
 - Example:
 - $ID, name \rightarrow ID$
 - $name \rightarrow name$
 - In general, $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$

So, that is how this will have to be looked at in specificity. We say that a functional dependency is trivial if the left hand side is a superset of the right hand side. So, if I have a bigger set of attributes on the left hand side id and name, then obviously id and name will functionally determine id, id and name will functionally determine name, name will functionally determine name. So, if you just think about because in a functional dependency the left hand side attributes that tuples have to match on the left hand side attribute and if they do, then they must match on the right hand side attribute. So, if the right hand side set of attributes is a subset of the left hand side, then obviously the functional dependency will be vacuously true and these are called trivial dependencies.

(Refer Slide Time: 29:33)



Functional Dependencies (Cont.)

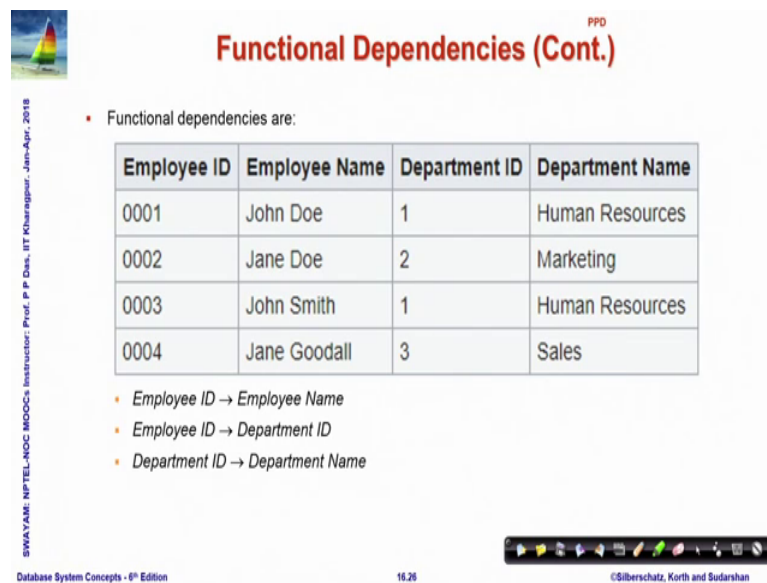
- Functional dependencies are:

StudentID	Semester	Lecture	TA
1234	6	Numerical Methods	John
1221	4	Numerical Methods	Smith
1234	6	Visual Computing	Bob
1201	2	Numerical Methods	Peter
1201	2	Physics II	Simon

- $StudentID \rightarrow Semester$
- $\{StudentID, Lecture\} \rightarrow TA$
- $\{StudentID, Lecture\} \rightarrow \{TA, Semester\}$

So, in the next couple of slides, I have shown few examples of functional dependencies of different tables. Here student id functionally determines semesters which mean that we are trying to model that a student cannot be at the same time in two semesters, then student id and lecture together functionally determines who is TA and so on and you can see for this particular relation student id and lecture, pair also happens to be the candidate key.

(Refer Slide Time: 30:05)



PPD

Functional Dependencies (Cont.)

- Functional dependencies are:

Employee ID	Employee Name	Department ID	Department Name
0001	John Doe	1	Human Resources
0002	Jane Doe	2	Marketing
0003	John Smith	1	Human Resources
0004	Jane Goodall	3	Sales

- Employee ID → Employee Name
- Employee ID → Department ID
- Department ID → Department Name

Database System Concepts - 6th Edition 16.26 ©Silberschatz, Korth and Sudarshan

These are another example. So, these are just go through them, try to convince yourself that these functional dependencies are very genuinely real world situations that can be modeled in this way.

(Refer Slide Time: 30:19)



Closure of a Set of Functional Dependencies

- Given a set F of functional dependencies, there are certain other functional dependencies that are logically implied by F
 - For example: 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^+
- F^+ is a superset of F
 - $F = \{A \rightarrow B, B \rightarrow C\}$
 - $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$



Given a set of functional dependencies, we can actually compute a closure. For example, if A functionally determines B and B functionally determines C , then we can infer that A functionally determines C because if two people match on A , they match on B . Now, if B functionally determines C also holds, then if they match on B , they match on C . So, if they match on A , then necessarily they may have to match on C . So, this is called the logical implication of a set of functional dependencies and we will see more of this later, but if we take all functional dependencies of a given set F , that are logically implied from this set F . We said that is a closure set and we represent that by F^+ .

So, F^+ necessarily is a superset of F . So, here in that above example, this is F and this is F^+ .

(Refer Slide Time: 31:00)



Module Summary

- Identified the features of good relational design
- Familiarized with the First Normal Form
- Introduced the notion of Functional Dependencies

So, we will continue more on the theory of functional dependencies, but let us conclude this module by summarizing that we have identified the features of good relational designs tradeoff between decomposition and lossless join properties that we need. We are familiarized with the first normal form and atomic domains and we have introduced the notion of functional dependencies on which we will build up more and try to get zero on very concrete strategies for good results.