Database Management Systems

Module 16: Relational Database Design/1

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

Database System Concepts, 6th Ed.

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

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

Module Objectives

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

Module Outline

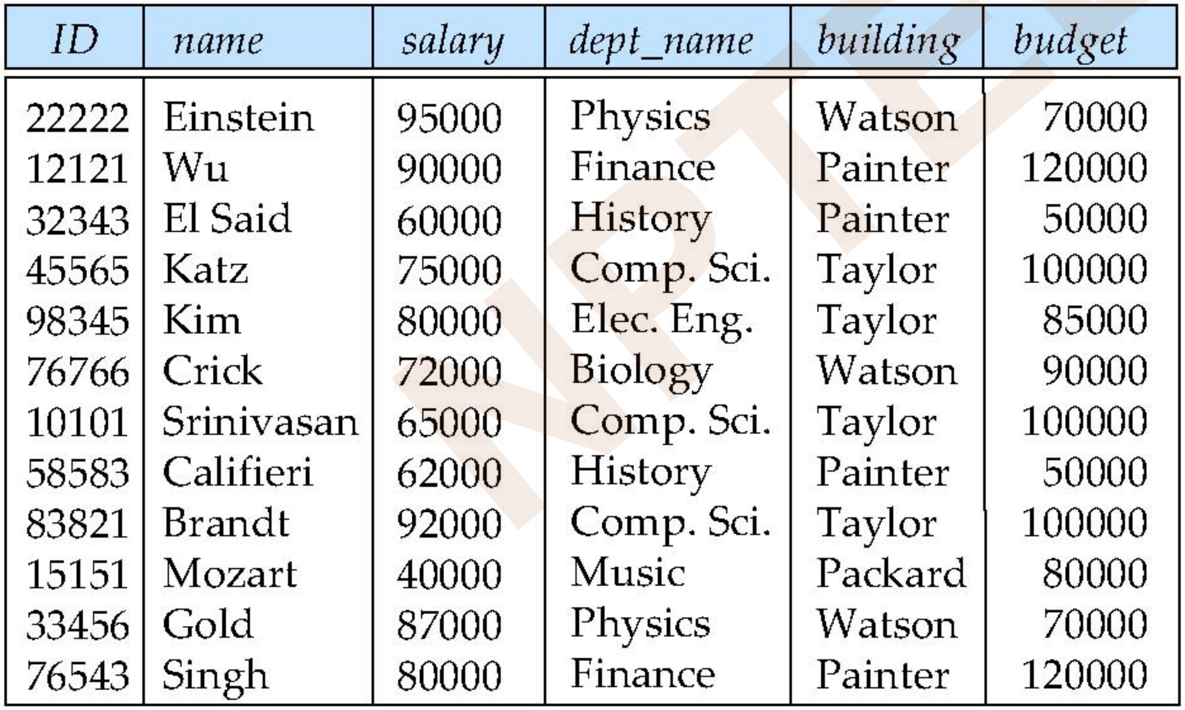
- Features of Good Relational Design
- Atomic Domains and First Normal Form
- Functional Dependencies

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

FEATURES OF GOOD RELATIONAL DESIGN

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

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

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 → *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

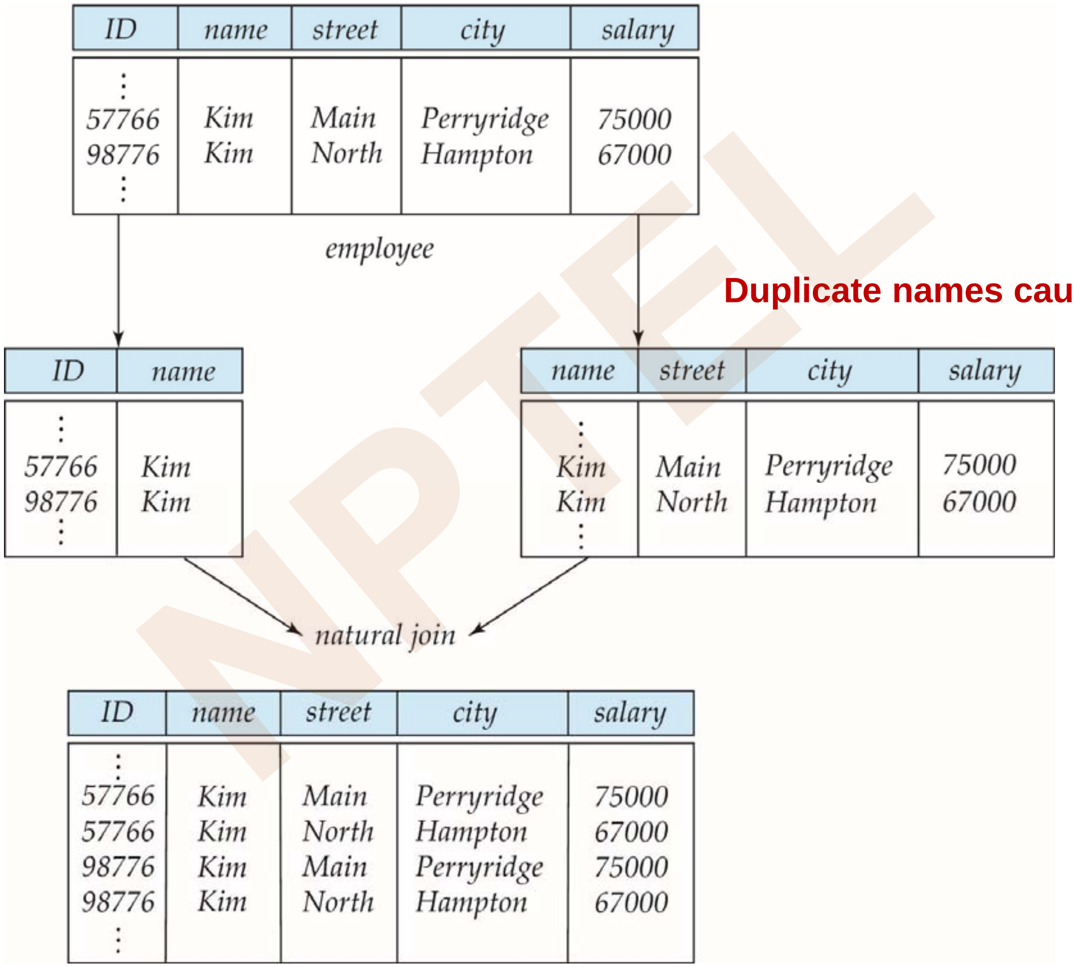
  employee1 (*ID*, *name*)

  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**.

PPD

A Lossy Decomposition



ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

Duplicate names cause the problem here.

Example of Lossless-Join Decomposition

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

  $R_1 = (A, B) \quad R_2 = (B, C)$

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

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

$\Pi_{\mathrm{A}} (\mathrm{r}) \bowtie \Pi_{\mathrm{B}} (\mathrm{r})$

A	B	C
α	1	A
β	2	B

PPD

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

ATOMIC DOMAINS AND FIRST NORMAL FORM

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

First Normal Form (1NF)

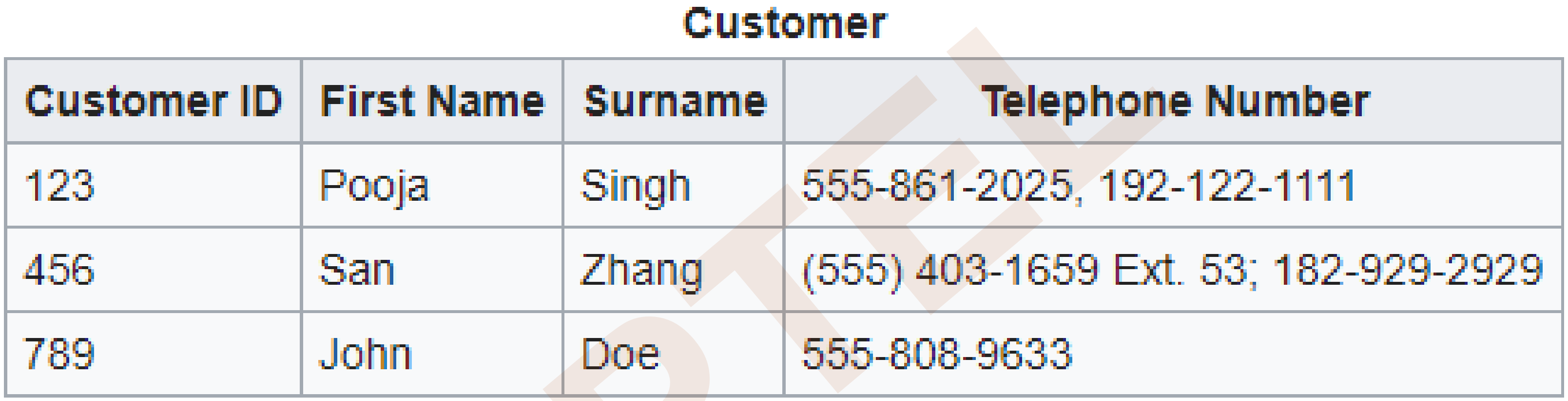
- Domain is **atomic** if its elements are considered to be indivisible units
  - Examples of non-atomic domains:
    - Set of names, composite attributes
    - Identification numbers like CS101 that can be broken up into parts
- A relational schema R is in **first normal form** if
  - the domains of 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

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

First Normal Form (Cont’d)

- The following is not in 1NF



Customer

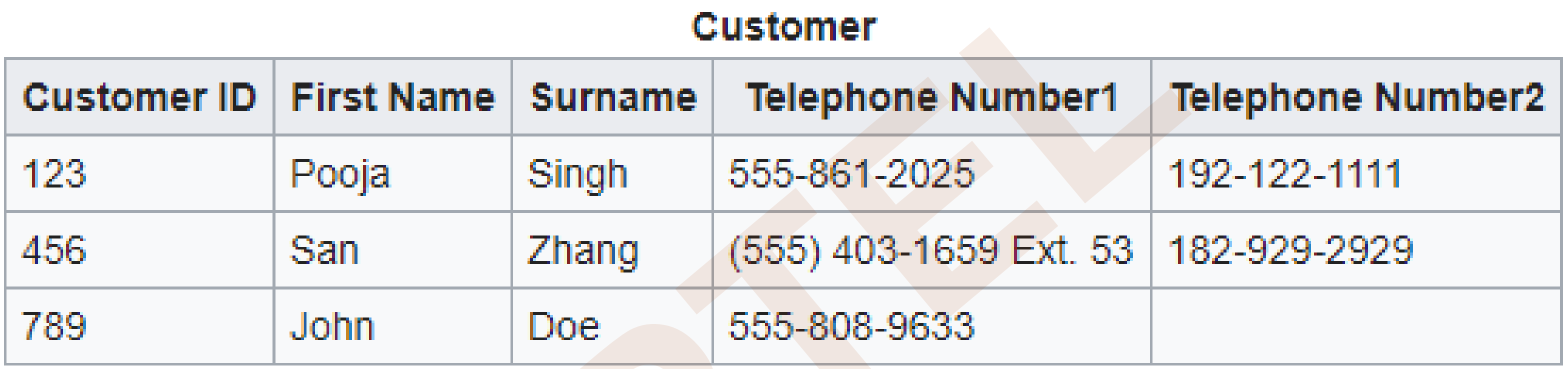
Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

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

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

First Normal Form (Cont'd)

- Consider:



Customer

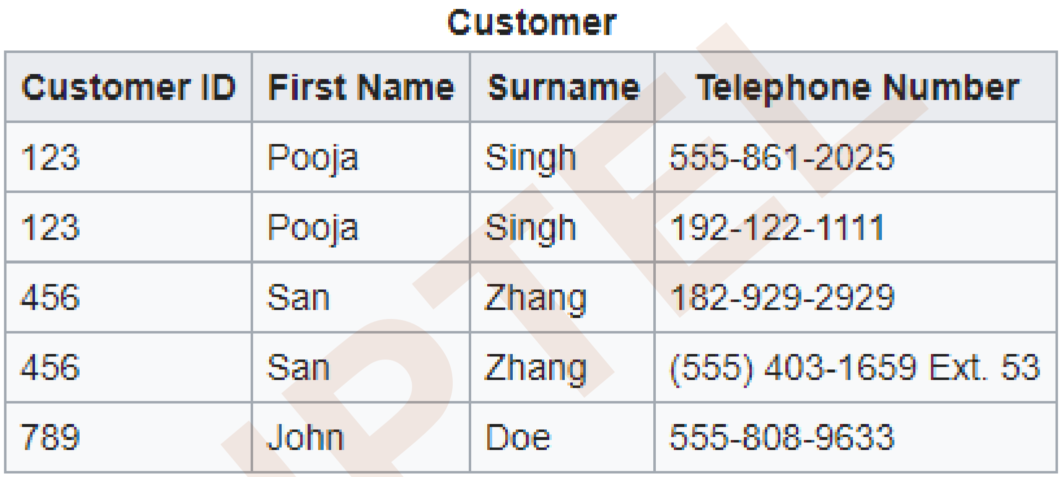
Customer ID	First Name	Surname	Telephone Number1	Telephone Number2
123	Pooja	Singh	555-861-2025	192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53	182-929-2929
789	John	Doe	555-808-9633	

- 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

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

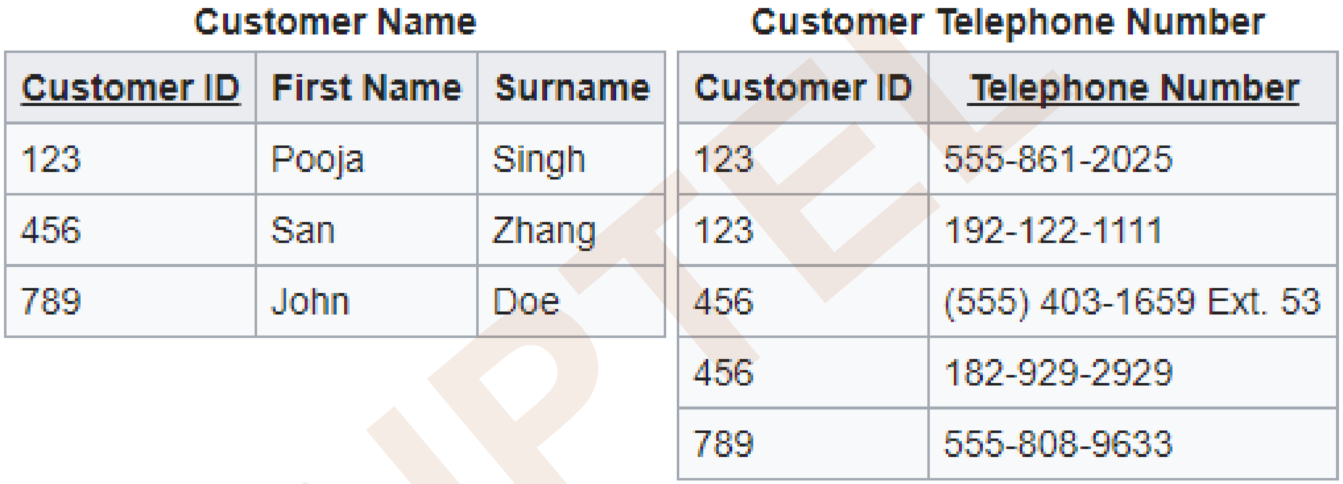
  - Duplicated information
  - ID is no more the key. Key is (ID, Telephone Number)

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

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

First Normal Form (Cont'd)

- Better to have 2 relations:



Customer Name

Customer ID	First Name	Surname
123	Pooja	Singh
456	San	Zhang
789	John	Doe

Customer Telephone Number

Customer ID	Telephone Number
123	555-861-2025
123	192-122-1111
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

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

FUNCTIONAL DEPENDENCIES

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

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 $\{R_1, R_2, ..., R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - functional dependencies
  - multivalued dependencies

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*

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* → *B* does **NOT** hold, but *B* → *A* does hold.

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:

  $$dept\_name \rightarrow salary$$

Use of Functional Dependencies

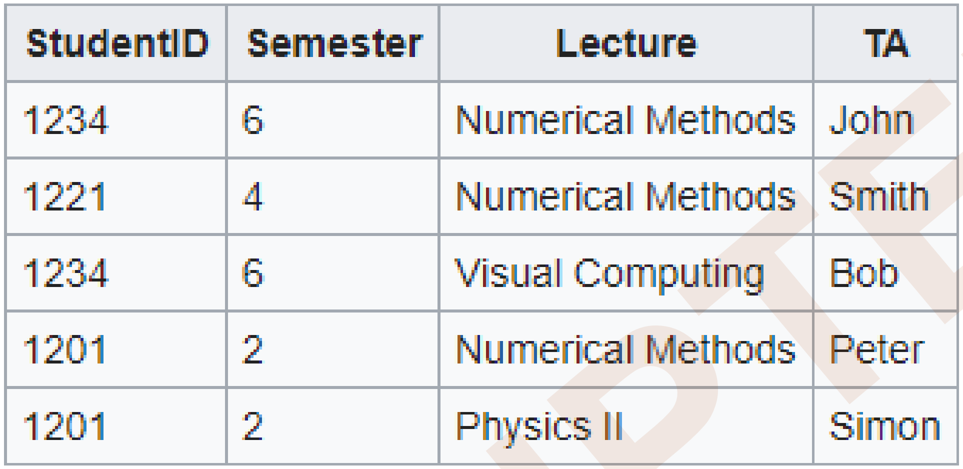
- We use functional dependencies to:
  - test relations to see if they are legal under a given set of functional dependencies.
    - If a relation *r* is legal under a set *F* of functional dependencies, we say that *r* **satisfies** *F*
  - specify constraints on the set of legal relations
    - We 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 $name \rightarrow ID$

Functional Dependencies (Cont.)

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

Functional Dependencies (Cont.)

- Functional dependencies are:

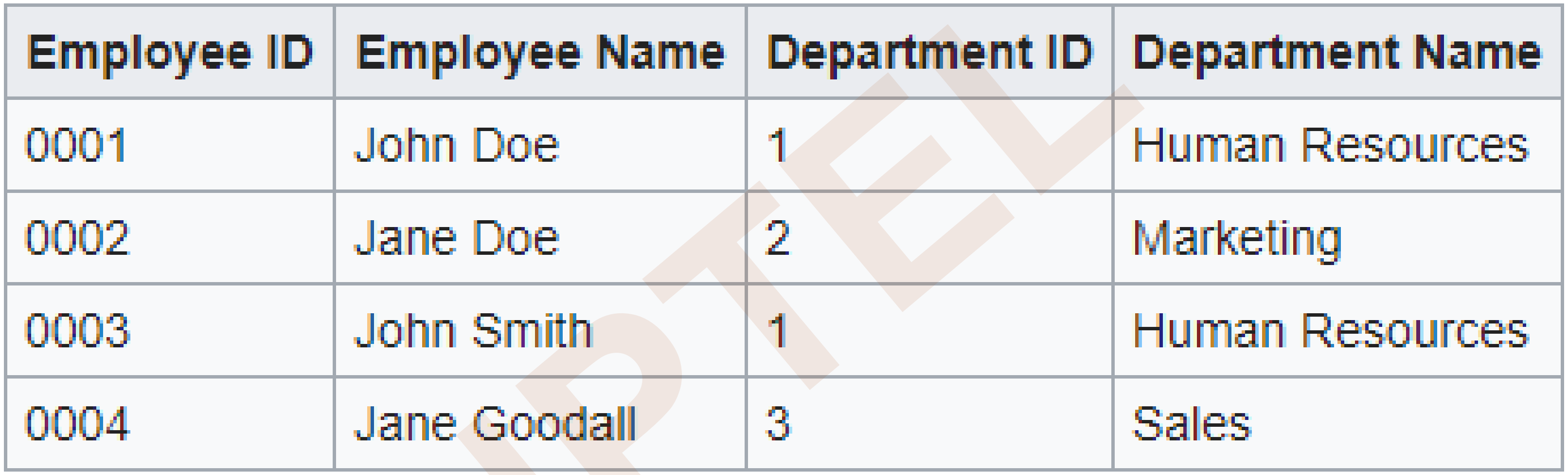


StudentID	Semester	Lecture	TA
1234	6	Numerical Methods	John
1221	4	Numerical Methods	Smith
1234	6	Visual Computing	Bob
1201	2	Numerical Methods	Peter
1201	2	Physics II	Simon

  - *StudentID* → *Semester*
  - *{StudentID, Lecture}* → *TA*
  - *{StudentID, Lecture}* → *{TA, Semester}*

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*

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
  - $F = \{A \rightarrow B, B \rightarrow C\}$
  - $F^+ = \{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

Module Summary

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

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".

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

Database Management Systems

Module 17: Relational Database Design/2

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
www.db-book.com

PPD

Module Recap

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

Module Objectives

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

Module Outline

- Decomposition Using Functional Dependencies
- Functional Dependency Theory

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

PPD

- **Decomposition Using Functional** Dependencies
- Functional Dependency Theory

DECOMPOSITION USING FUNCTIONAL DEPENDENCIES

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 (*<u>ID,</u> name, salary<u>, dept_name, </u>building, budget*)

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

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:

  - (α U β)
  - (*R* - (β - α))
- In our example,
  - α = *dept_name*
  - β = *building, budget*
  - *dept_name* → *building, budget*

  inst_dept is replaced by

  - (α U β) = (*dept_name, building, budget*)
    - *dept_name* → *building, budget*
  - (*R* - (β - α)) = (*ID, name, salary, dept_name*)
    - ID → *name*, *salary*, *dept_name*

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.*

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

Third Normal Form

- A relation schema *R* is in **third normal form (3NF)** if for all:

  $$\alpha \to \beta \text{ in } F^+$$

  at least one of the following holds:
  - $\alpha \to \beta$ 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)

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 $\{R_1, R_2, ..., R_n\}$ such that
  - each relation scheme is in good form
  - the decomposition is a lossless-join decomposition
  - Preferably, the decomposition should be dependency preserving

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

ID	*child_name*	*phone*
99999	David	512-555-1234
99999	David	512-555-4321
99999	William	512-555-1234
99999	Willian	512-555-4321

inst_info

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)

How good is BCNF? (Cont.)

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

inst_child

ID	*child_name*
99999	David
99999	David
99999	William
99999	Willian

inst_phone

ID	*phone*
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)

PPD

- Decomposition Using Functional Dependencies
- **Functional Dependency Theory**

FUNCTIONAL DEPENDENCY THEORY

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

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^+

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)

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 $CG \rightarrow$ CG*I*, and augmenting of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

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

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 $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds (**pseudotransitivity**)

  The above rules can be inferred from Armstrong's axioms

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 $\beta \subseteq result$ then  result := $result \cup \gamma$
            end
```

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

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 $\alpha^{+,}$ 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 $\gamma \rightarrow S$.

Module Summary

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

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".

Database Management Systems

Module 18: Relational Database Design/3

Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
www.db-book.com

Module Recap

- Decomposition Using Functional Dependencies
- Functional Dependency Theory

PPD

Module Objectives

- To Learn Algorithms for Properties of Functional Dependencies
- To Understand the Characterizations for Lossless Join Decomposition
- To Understand the Characterizations for Dependency Preservation

Module Outline

- Algorithms for Functional Dependencies
- Lossless Join Decomposition
- Dependency Preservation

- **Algorithms for Functional Dependencies**
- Lossless Join Decomposition
- Dependency Preservation

ALGORITHMS FOR FUNCTIONAL DEPENDENCIES

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

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 $(G)^+ \supseteq R$

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 $\alpha^{+,}$ 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 $\gamma \rightarrow S$.

Canonical Cover

- Sets of functional dependencies may have redundant dependencies that can be inferred from the others
  - For example: $A \rightarrow C$ is redundant in: $\{A \rightarrow B,\ \ B \rightarrow C, A \rightarrow C\}$
  - Parts of a functional dependency may be redundant
    - E.g.: on RHS: $\boldsymbol{\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow CD\}}$ can be simplified to
      $\boldsymbol{\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}}$
      - *In the forward: (1)* $A \rightarrow CD$ ➔ $A \rightarrow C$ *and* $A \rightarrow D$ *(2)* $A \rightarrow B,\ \ B \rightarrow C$ ➔ $A \rightarrow C$
      - *In the reverse: (1)* $A \rightarrow B,\ \ B \rightarrow C$ ➔ $A \rightarrow C$ *(2)* $A \rightarrow C, A \rightarrow D$ ➔ $A \rightarrow CD$
    - E.g.: on LHS: $\boldsymbol{\{A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D\}}$ can be simplified to
      $\boldsymbol{\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}}$
      - *In the forward: (1)* $A \rightarrow B,\ \ B \rightarrow C$ ➔ $A \rightarrow C$ ➔ $A \rightarrow AC$ *(2)* $A \rightarrow AC$, $AC \rightarrow D$ ➔ $A \rightarrow D$
      - *In the reverse:* $A \rightarrow D$ ➔ $AC \rightarrow D$
- Intuitively, a canonical cover of F is a “minimal” set of functional dependencies equivalent to F, having no redundant dependencies or redundant parts of dependencies

Canonical Cover: RHS

- $\{A \rightarrow B, \;\; B \rightarrow C, \;\; A \rightarrow CD\}$ ➔ $\{A \rightarrow B, \;\; B \rightarrow C, \;\; A \rightarrow D\}$
  - (1) $A \rightarrow CD$ ➔ $A \rightarrow C$ and $A \rightarrow D$ (2) $A \rightarrow B, \;\; B \rightarrow C$ ➔ $A \rightarrow C$
  - $A+ = ABCD$

- $\{A \rightarrow B, \;\; B \rightarrow C, \;\; A \rightarrow D\}$ ➔ $\{A \rightarrow B, \;\; B \rightarrow C, \;\; A \rightarrow CD\}$
  - $A \rightarrow B, \;\; B \rightarrow C$ ➔ $A \rightarrow C$
  - $A \rightarrow C, A \rightarrow D$ ➔ $A \rightarrow CD$
  - $A+ = ABCD$

Canonical Cover: LHS

- $\{A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D\}$ ➔ $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$
  - $A \rightarrow B,\ \ B \rightarrow C$ ➔ $A \rightarrow C$ ➔ $A \rightarrow AC$
  - $A \rightarrow AC\ ,\ AC \rightarrow D$ ➔ $A \rightarrow D$
  - $A+ = ABCD$
- $\{A \rightarrow B,\ \ B \rightarrow C,\ \ A \rightarrow D\}$ ➔ $\{A \rightarrow B,\ \ B \rightarrow C,\ \ AC \rightarrow D\}$
  - $A \rightarrow D$ ➔ $AC \rightarrow D$
  - $AC+ = ABCD$

Extraneous Attributes

- Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.
  - Attribute A is **extraneous** in α if $A \in \alpha$ and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
  - Attribute A is **extraneous** in β if $A \in \beta$ and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.
- *Note:* Implication in the opposite direction is trivial in each of the cases above, since a "stronger" functional dependency always implies a weaker one
- Example: Given $F = \{A \rightarrow C, AB \rightarrow C\}$
  - B is extraneous in $AB \rightarrow C$ because $\{A \rightarrow C, AB \rightarrow C\}$ logically implies $A \rightarrow C$ (I.e. the result of dropping B from $AB \rightarrow C$).
  - A+ = AC in $\{A \rightarrow C, AB \rightarrow C\}$
- Example: Given $F = \{A \rightarrow C, AB \rightarrow CD\}$
  - C is extraneous in $AB \rightarrow CD$ since $AB \rightarrow C$ can be inferred even after deleting C
  - *AB+ = ABCD in* $\{A \rightarrow C, AB \rightarrow D\}$

Testing if an Attribute is Extraneous

- Consider a set *F* of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in *F*.
- To test if attribute A $\in \alpha$ is extraneous in α
  1. Compute $(\{\alpha\} - A)^+$ using the dependencies in *F*
  2. Check that $(\{\alpha\} - A)^+$ contains β; if it does, *A* is extraneous in α
- To test if attribute *A* $\in \beta$ is extraneous in β
  1. Compute α^+ using only the dependencies in
     $F' = (F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$,
  2. Check that α^+ contains *A;* if it does*,* *A* is extraneous in β

Canonical Cover

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

- A **canonical cover** for *F* is a set of dependencies F_c such that
  - *F* logically implies all dependencies in $F_{c,}$ and
  - F_c logically implies all dependencies in *F,* and
  - No functional dependency in F_c contains an extraneous attribute, and
  - Each left side of functional dependency in F_c is unique
- To compute a canonical cover for *F*:

```
repeat
        Use the union rule to replace any dependencies in F
                $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$
        Find a functional dependency $\alpha \rightarrow \beta$ with an
                extraneous attribute either in $\alpha$ or in $\beta$
                /* Note: test for extraneous attributes done using $F_{c,}$ not F*/
        If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$
until F does not change
```

- Note: Union rule may become applicable after some extraneous attributes have been deleted, so it has to be re-applied

- ***Minimal Sets of Functional Dependencies***
- ***Irreducible Set of Functional Dependencies***

Computing a Canonical Cover

- $R = (A, B, C)$
  $F = \{A \rightarrow BC$
  $B \rightarrow C$
  $A \rightarrow B$
  $AB \rightarrow C\}$
- Combine $A \rightarrow BC$ and $A \rightarrow B$ into $A \rightarrow BC$
  - Set is now $\{A \rightarrow BC, B \rightarrow C, AB \rightarrow C\}$
- A is extraneous in $AB \rightarrow C$
  - Check if the result of deleting A from $AB \rightarrow C$ is implied by the other dependencies
    - Yes: in fact, $B \rightarrow C$ is already present!
  - Set is now $\{A \rightarrow BC, B \rightarrow C\}$
- C is extraneous in $A \rightarrow BC$
  - Check if $A \rightarrow C$ is logically implied by $A \rightarrow B$ and the other dependencies
    - Yes: using transitivity on $A \rightarrow B$ *and* $B \rightarrow$ C.
      - Can use attribute closure of A in more complex cases
- The canonical cover is: $A \rightarrow B$
  $B \rightarrow C$

Equivalence of Sets of Functional Dependencies

- Let F & G are two functional dependency sets.
  - These two sets F & G are equivalent if $F^+ = G^+$
  - Equivalence means that every functional dependency in F can be inferred from G, and every functional dependency in G an be inferred from F
- F and G are equal only if
  - F covers G: Means that all functional dependency of G are logically numbers of functional dependency set $F \Rightarrow F \supseteq G$.
  - G covers F: Means that all functional dependency of F are logically members of functional dependency set $G \Rightarrow G \supseteq F$

Condition	CASES			
F Covers G	True	True	False	False
G Covers F	True	False	True	False
Result	F=G	F⊃G	G⊃F	No Comparison

Practice Problems on Functional Dependencies

- **Find if a given functional dependency is implied from a set of Functional Dependencies:**
  1. For: A → BC, CD → E, E → C, D → AEH, ABH → BD, DH → BC
     a. Check: BCD → H
     b. Check: AED→C
  2. For: AB → CD, AF → D, DE → F, C → G, F → E, G → A
     a. Check: CF → DF
     b. Check: BG → E
     c. Check: AF → G
     d. Check: AB → EF
  3. For: A → BC, B → E, CD → EF
     a. Check: AD → F

Source: http://www.edugrabs.com/membership-test-for-functional-dependency/

Practice Problems on Functional Dependencies

- **Find Candidate Key using Functional Dependencies:**
  1. Relational Schema R(ABCDE). Functional dependencies: AB → C, DE → B, CD → E
  2. Relational Schema R(ABCDE). Functional dependencies: AB → C, C → D, B → EA

Source: http://www.edugrabs.com/how-to-find-candidate-key-using-functional-dependencies/

Practice Problems on Functional Dependencies

- **Find Super Key using Functional Dependencies:**
  1. Relational Schema R(ABCDE). Functional dependencies: AB → C, DE → B, CD → E
  2. Relational Schema R(ABCDE). Functional dependencies: AB → C, C → D, B → EA

Source: http://www.edugrabs.com/how-to-find-super-key-from-functional-dependencies/

Practice Problems on Functional Dependencies

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

- **Find Prime and Non Prime Attributes using Functional Dependencies:**
  1. R(ABCDEF) having FDs {AB→C, C→D, D→E, F→B, E→F}
  2. R(ABCDEF) having FDs {AB → C, C → DE, E → F, C → B}
  3. R(ABCDEFGHIJ) having FDs {AB → C, A → DE, B → F, F → GH, D → IJ}
  4. R(ABDLPT) having FDs {B → PT, A → D, T → L}
  5. R(ABCDEFGH) having FDs {E → G, AB → C, AC → B, AD → E,B → D, BC → A}
  6. R(ABCDE) having FDs {A → BC, CD → E, B → D, E → A}
  7. R(ABCDEH) having FDs {A → B, BC → D, E → C, D → A}

- **Prime Attributes –** Attribute set that belongs to any candidate key are called Prime Attributes
  - It is union of all the candidate key attribute: {CK1 ∪ CK2 ∪ CK3 ∪ ……}
  - If Prime attribute determined by other attribute set, then more than one candidate key is possible.
  - For example, If A is Candidate Key, and X→A, then, X is also Candidate Key .
- **Non Prime Attribute –** Attribute set does not belongs to any candidate key are called Non Prime Attributes

Source: http://www.edugrabs.com/prime-and-non-prime-attributes/

Practice Problems on Functional Dependencies

- **Check the Equivalence of a Pair of Sets of Functional Dependencies:**
  1. Consider the two sets F and G with their FDs as below :
     1. F : A → C, AC → D, E → AD, E → H
     2. G: A → CD, E → AH
  2. Consider the two sets P and Q with their FDs as below :
     1. P : A → B, AB → C, D → ACE
     2. Q : A → BC, D → AE

Source: http://www.edugrabs.com/equivalence-of-sets-of-functional-dependencies-example/

Practice Problems on Functional Dependencies

- **Find the Minimal Cover or Irreducible Sets or Canonical Cover of a Set of Functional Dependencies:**
  1. AB → CD, BC → D
  2. ABCD → E, E → D, AC → D, A → B

Source: http://www.edugrabs.com/questions-on-minimal-cover/

PPD

- Algorithms for Functional Dependencies
- **Lossless Join Decomposition**
- Dependency Preservation

LOSSLESS JOIN DECOMPOSITION

Lossless-join Decomposition

- For the case of $R = (R_1, R_2)$, we require that for all possible relations r on schema R

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- A decomposition of R into R_1 and R_2 is lossless join if at least one of the following dependencies is in F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$
- The above functional dependencies are a sufficient condition for lossless join decomposition; the dependencies are a necessary condition only if all constraints are functional dependencies

To Identify whether a decomposition is lossy or lossless, it must satisfy the following conditions :

- $R_1 \cup R_2 = R$
- $R_1 \cap R_2 \neq \Phi$ *and*
- $R_1 \cap R_2 \rightarrow R_1$ *or* $R_1 \cap R_2 \rightarrow R_2$

Example

- Consider **Supplier_Parts** schema: **Supplier_Parts(S#, Sname, City, P#, Qty)**
- Having dependencies: **S#** → **Sname**, **S#** → **City**, **(S#, P#)** → **Qty**
- Decompose as: **Supplier(S#, Sname, City, Qty)**: **Parts(P#, Qty)**
- Take Natural Join to reconstruct: **Supplier** ⋈ **Parts**

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

S#	Sname	City	Qty
3	Smith	London	20
5	Nick	NY	50
2	Steve	Boston	10
5	Nick	NY	40
5	Nick	NY	10

P#	Qty
301	20
500	50
20	10
400	40
301	10

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
5	**Nick**	**NY**	**20**	**10**
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10
2	**Steve**	**Boston**	**301**	**10**

- We get extra tuples! **Join is Lossy!**
- Common attribute **Qty** is not a superkey in **Supplier** or in **Parts**
- Does not preserve **(S#,P#)** → **Qty**

Source: http://www.edugrabs.com/lossy-join-decomposition/

Example

- Consider **Supplier_Parts** schema: **Supplier_Parts(S#, Sname, City, P#, Qty)**
- Having dependencies: **S#** → **Sname**, **S#** → **City**, **(S#, P#)** → **Qty**
- Decompose as: **Supplier(S#, Sname, City)**: **Parts(S#, P#, Qty)**
- Take Natural Join to reconstruct: **Supplier** ⋈ **Parts**

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

S#	Sname	City
3	Smith	London
5	Nick	NY
2	Steve	Boston
5	Nick	NY
5	Nick	NY

S#	P#	Qty
3	301	20
5	500	50
2	20	10
5	400	40
5	301	10

S#	Sname	City	P#	Qty
3	Smith	London	301	20
5	Nick	NY	500	50
2	Steve	Boston	20	10
5	Nick	NY	400	40
5	Nick	NY	301	10

- We get back the original relation. **Join is Lossless.**
- Common attribute **S#** is a superkey in **Supplier**
- Preserves all dependencies

Source: http://www.edugrabs.com/desirable-properties-of-decomposition/#lossless

Example

- $R = (A, B, C)$
  $F = \{A \rightarrow B, B \rightarrow C)$
  - Can be decomposed in two different ways
- $R_1 = (A, B), \quad R_2 = (B, C)$
  - Lossless-join decomposition:

    $$R_1 \cap R_2 = \{B\} \text{ and } B \rightarrow BC$$

  - Dependency preserving
- $R_1 = (A, B), \quad R_2 = (A, C)$
  - Lossless-join decomposition:

    $$R_1 \cap R_2 = \{A\} \text{ and } A \rightarrow AB$$

  - Not dependency preserving
    (cannot check $B \rightarrow C$ without computing $R_1 \bowtie R_2$)

Practice Problems on Lossless Join

- **Check if the decomposition of R into $\mathcal{D}$ is lossless:**
  1. R(ABC): F = {A → B, A → C}. $\mathcal{D} = R_1(AB), R_2(BC)$
  2. R(ABCDEF): F = {A → B, B → C, C → D, E → F}. $\mathcal{D} = R_1(AB), R_2(BCD), R_3(DEF)$
  3. R(ABCDEF): F = {A → B, C → DE, AC → F}. $\mathcal{D} = R_1(BE), R_2(ACDEF)$
  4. R(ABCDEG): F = {AB → C, AC → B, AD → E, B → D, BC → A, E → G}
     1. $\mathcal{D}1 = R_1(AB), R_2(BC), R_3(ABDE), R_4(EG)$
     2. $\mathcal{D}2 = R_1(ABC), R_2(ACDE), R_3(ADG)$
  5. R(ABCDEFGHIJ): F = {AB → C, B → F, D → IJ, A → DE, F → GH}
     1. $\mathcal{D}1 = R_1(ABC), R_2(ADE), R_3(BF), R_4(FGH), R_5(DIJ)$
     2. $\mathcal{D}2 = R_1(ABCDE), R_2(BFGH), R_3(DIJ)$
     3. $\mathcal{D}3 = R_1(ABCD), R_2(DE), R_3(BF), R_4(FGH), R_5(DIJ)$

Source: http://www.edugrabs.com/questions-on-lossless-join/

- Algorithms for Functional Dependencies
- Lossless Join Decomposition
- **Dependency Preservation**

DEPENDENCY PRESERVATION

Dependency Preservation

- Let F_i be the set of dependencies F^+ that include only attributes in R_i
  - A decomposition is **dependency preserving**, if

    $(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$
  - If it is not, then checking updates for violation of functional dependencies may require computing joins, which is expensive

Let R be the original relational schema having FD set F. Let R_1 and R_2 having FD set F_1 and F_2 respectively, are the decomposed sub-relations of R. The decomposition of R is said to be preserving if

- *$F_1 \cup F_2 \equiv F$ {Decomposition Preserving Dependency}*

- *If $F_1 \cup F_2 \subset F$ {Decomposition NOT Preserving Dependency} and*
- *$F_1 \cup F_2 \supset F$ {this is not possible}*

Testing for Dependency Preservation

- To check if a dependency $\alpha \rightarrow \beta$ is preserved in a decomposition of R into R_1, R_2, …, R_n we apply the following test (with attribute closure done with respect to F)

  -
    ```
    result = $\alpha$
    while (changes to result) do
      for each $R_i$ in the decomposition
            $t = (result \cap R_i)^+ \cap R_i$
            $result = result \cup t$
    ```
  - If *result* contains all attributes in β, then the functional dependency $\alpha \rightarrow \beta$ is preserved.
- We apply the test on all dependencies in F to check if a decomposition is dependency preserving
- This procedure takes polynomial time, instead of the exponential time required to compute F^+ and $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$

Example

- R(ABCDEF):
- F = {A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E}
- Ɗ = {ABCD, BF, DE}
- On projections:

ABCD (R1)	BF (R2)	DE (R3)
A → BCD BC → AD	B → F	D → E

- Need to check for: ~~A→BCD~~, **A→EF**, ~~BC→AD~~, **BC→E**, **BC→F**, ~~B→F~~, ~~D→E~~
- (BC)+/F1 = ABCD. (ABCD)+/F2 = ABCDF. **(ABCDF)+/F3 = ABCDEF**. Preserves **BC→E**, **BC→F**
- (A)+/F1 = ABCD. (ABCD)+/F2 = ABCDF. **(ABCDF)+/F3 = ABCDEF**. Preserves **A→EF**

Example

- R(ABCDEF): F = {A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E}. D = {ABCD, BF, DE}
- On projections:

ABCD (R1)	BF (R2)	DE (R3)
A → B, A → C, A → D, BC → A, BC → D	B → F	D → E

- Infer reverse FD's:
  - B+/F = BF: B → A cannot be inferred
  - C+/F = C: C → A cannot be inferred
  - D+/F = DE: D → A and D → BC cannot be inferred
  - A+/F = ABCDEF: A → BC can be inferred, but it is equal to A → B and A → C
  - F+/F = F: F → B cannot be inferred
  - E+/F = E: E → D cannot be inferred
- Need to check for: ~~A→BCD~~, **A→EF**, ~~BC→AD~~, **BC→E**, **BC→F**, ~~B→F~~, ~~D→E~~
  - **(BC)+/F = ABCDEF**. Preserves **BC→E**, **BC→F**
  - **(A)+/F = ABCDEF**. Preserves **A→EF**

Practice Problems on Dependency Preservation

- **Check whether the decomposition of R into Ɗ is preserving dependency:**
  1. R(ABCD): F = {A → B, B → C, C → D, D → A}. Ɗ = {AB, BC, CD}
  2. R(ABCDEF): F = {AB → CD, C → D, D → E, E → F}. Ɗ = {AB, CDE, EF}
  3. R(ABCDEG): F = {AB → C, AC → B, BC → A, AD → E, B → D, E → G}. Ɗ = {ABC, ACDE, ADG}
  4. R(ABCD): F = {A → B, B → C, C → D, D → B}. Ɗ = {AB, BC, BD}
  5. R(ABCDE): F = {A → BC, CD → E, B → D, E → A}. Ɗ = {ABCE, BD}

Source: http://www.edugrabs.com/question-on-dependency-preserving-decomposition/

Module Summary

- Studied Algorithms for Properties of Functional Dependencies
- Understood the Characterization for and Determination of Lossless Join
- Understood the Characterization for and Determination of Dependency Preservation

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

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".

Database Management Systems

Module 19: Relational Database Design/4

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
www.db-book.com

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

PPD

Module Recap

- Algorithms for Functional Dependencies
- Lossless Join Decomposition
- Dependency Preservation

Module Objectives

- To Understand the Normal Forms and their Importance in Relational Design
- To Learn the Decomposition Algorithm for a Relation to 3NF
- To Learn the Decomposition Algorithm for a Relation to BCNF

Module Outline

- Normal Forms
- Decomposition to 3NF
- Decomposition to BCNF

PPD

- **Normal Forms**
- Decomposition to 3NF
- Decomposition to BCNF

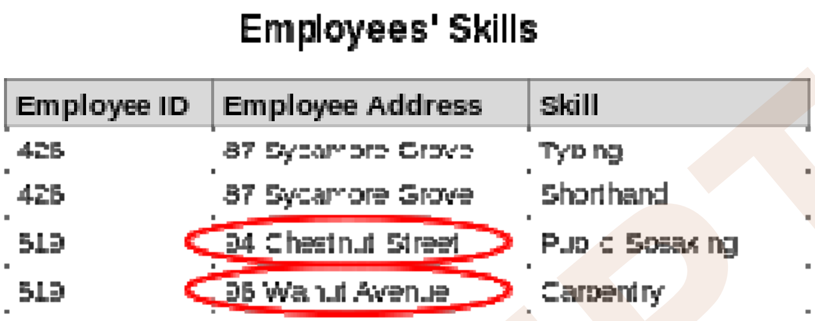
NORMAL FORMS

Normalization or Schema Refinement

- Normalization or Schema Refinement is a technique of organizing the data in the database
- A systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics
  - Insertion Anomaly
  - Update Anomaly
  - Deletion Anomaly
- Most common technique for the Schema Refinement is decomposition.
  - Goal of Normalization: Eliminate Redundancy
- Redundancy refers to repetition of same data or duplicate copies of same data stored in different locations
- Normalization is used for mainly two purpose:
  - Eliminating redundant (useless) data
  - Ensuring data dependencies make sense, that is, data is logically stored

Anomalies

1. **Update Anomaly**: Employee 519 is shown as having different addresses on different records



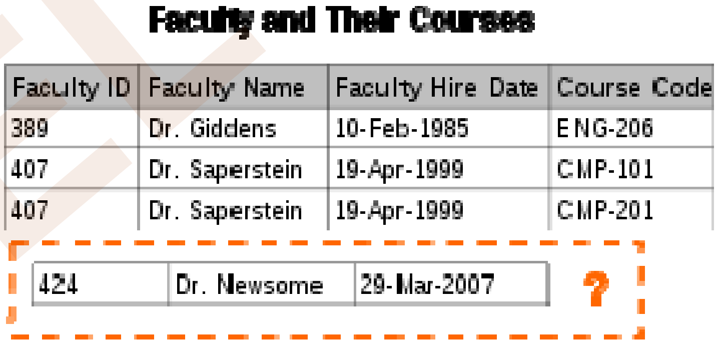
Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

Resolution: Decompose the Schema

1. *Update*: (ID, Address), (ID, Skill)
2. *Insert*: (ID, Name, Hire Date), (ID, Code)
3. *Delete*: (ID, Name, Hire Date), (ID, Code)

2. **Insertion Anomaly**: Until the new faculty member, Dr. Newsome, is assigned to teach at least one course, his details cannot be recorded

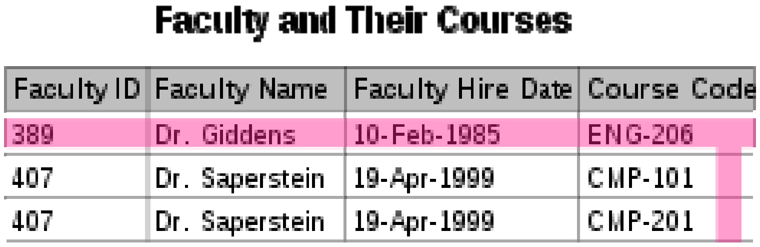


Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

424	Dr. Newsome	29-Mar-2007	?

3. **Deletion Anomaly**: All information about Dr. Giddens is lost if he temporarily ceases to be assigned to any courses.



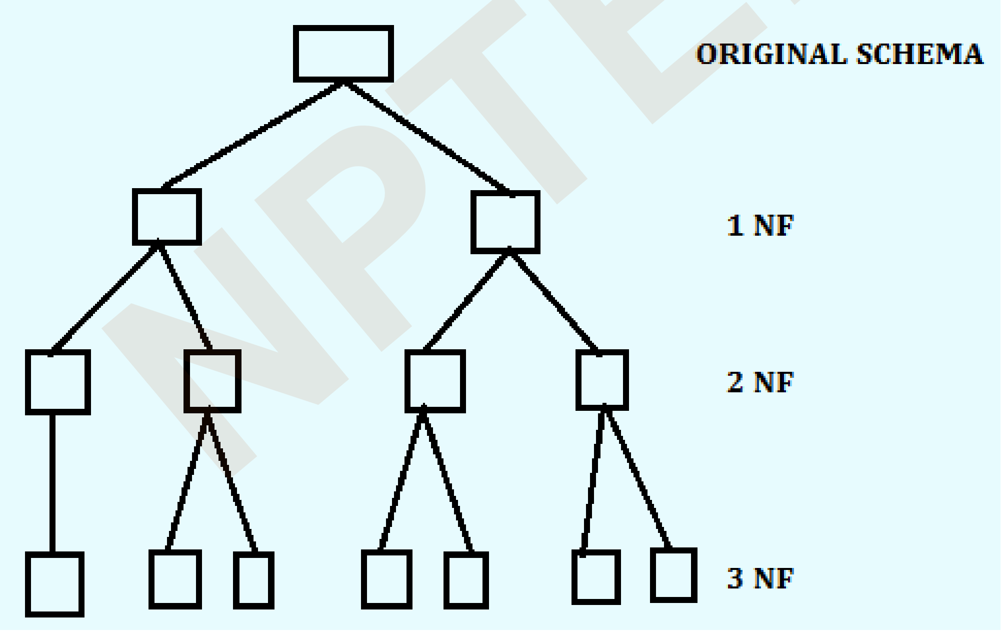
Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

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

Desirable Properties of Decomposition

- Lossless Join Decomposition Property
  - It should be possible to reconstruct the original table
- Dependency Preserving Property
  - No functional dependency (or other constraints should get violated)



Normalization and Normal Forms

- A normal form specifies a set of conditions that the relational schema must satisfy in terms of its constraints – they offer varied levels of guarantee for the design
- Normalization rules are divided into various normal forms. Most common normal forms are:
  - First Normal Form (1 NF)
  - Second Normal Form (2 NF)
  - Third Normal Form (3 NF)
- Informally, a relational database relation is often described as "normalized" if it meets third normal form. Most 3NF relations are free of insertion, update, and deletion anomalies

Normalization and Normal Forms

- Additional Normal Forms
  - Elementary Key Normal Form (EKNF)
  - **Boyce-codd Normal Form (BCNF)**
  - **Multivalued Dependencies And Fourth Normal Form (4 NF)**
  - Essential Tuple Normal Form (ETNF)
  - Join Dependencies And Fifth Normal Form (5 NF)
  - Sixth Normal Form (6NF)
  - Domain/Key Normal Form (DKNF)

First Normal Form (1 NF)

- A relation is in first Normal Form if and only if all underlying domains contain atomic values only
- In other words, a relation doesn't have multivalued attributes (MVA)
- Example:
  - ***STUDENT(Sid, Sname, Cname)***

Students		
SID	**Sname**	**Cname**
S1	A	C,C++
S2	B	C++, DB
S3	A	DB
SID : Primary Key		

MVA exists ➔ Not in 1NF

Students		
SID	**Sname**	**Cname**
S1	A	C
S1	A	C++
S2	B	C++
S2	B	DB
S3	A	DB
SID : Primary Key		

No MVA ➔ In 1NF

Source: http://www.edugrabs.com/normal-forms/#fnf

First Normal Form (1 NF): Possible Redundancy

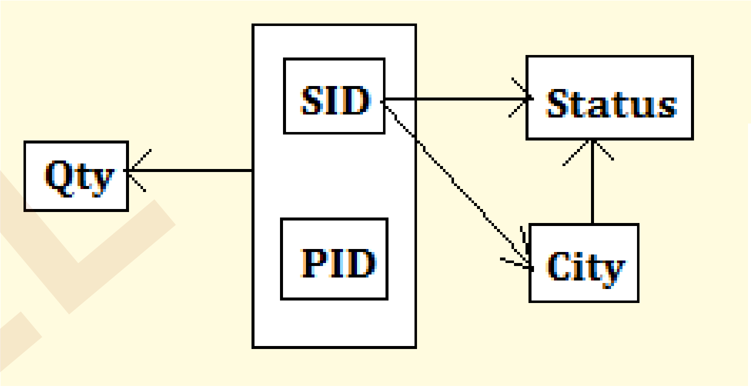
Source: http://www.edugrabs.com/normal-forms/#nf

- Example:
  - ***Supplier(SID, Status, City, PID, Qty)***

Supplier:

SID	Status	City	PID	Qty
S1	30	Delhi	P1	100
S1	30	Delhi	P2	125
S1	30	Delhi	P3	200
S1	30	Delhi	P4	130
S2	10	Karnal	P1	115
S2	10	Karnal	P2	250
S3	40	Rohtak	P1	245
S4	30	Delhi	P4	300
S4	30	Delhi	P5	315

Key : (SID, PID)



Drawbacks:

- **Deletion Anomaly –** If we delete the tuple <S3,40,Rohtak,P1,245>, then we loose the information about S3 that S3 lives in Rohtak.
- **Insertion Anomaly –** We cannot insert a Supplier S5 located in Karnal, until S5 supplies at least one part.
- **Updation Anomaly –** If Supplier S1 moves from Delhi to Kanpur, then it is difficult to update all the tuples containing (S1, Delhi) as SID and City respectively.

Normal Forms are the methods of reducing redundancy. However, Sometimes 1 NF increases redundancy. It does not make any efforts in order to decrease redundancy.

First Normal Form (1 NF): Possible Redundancy

- **When LHS is not a Superkey :**
  - Let X → Y is a non trivial FD over R with X is not a superkey of R, then redundancy exist between X and Y attribute set.
  - Hence in order to identify the redundancy, we need not to look at the actual data, it can be identified by given functional dependency.
  - Example : X →Y and X is not a Candidate Key
⇒ X can duplicate
⇒ corresponding Y value would duplicate also.

X	Y
1	3
1	3
2	3
2	3
4	6

- **When LHS is a Superkey :**
  - If X → Y is a non trivial FD over R with X is a superkey of R, then redundancy does not exist between X and Y attribute set.
  - Example : X →Y and X is a Candidate Key
⇒ X cannot duplicate
⇒ corresponding Y value may or may not duplicate.

X	Y
1	4
2	6
3	4

Source: http://www.edugrabs.com/normal-forms/#fnf

Second Normal Form (2 NF)

- Relation ***R*** is in Second Normal Form (2NF) only iff :
  - ***R*** should be in 1NF and
  - ***R*** should not contain any *Partial Dependency*

Partial Dependency:

Let ***R*** be a relational Schema and ***X,Y,A*** be the attribute sets over ***R*** where
X: Any Candidate Key, ***Y:*** Proper Subset of Candidate Key, and ***A:*** Non Key Attribute

If Y → A exists in R, then R is not in 2 NF.

(Y → A) is a Partial dependency only if

- ***Y:*** Proper subset of Candidate Key
- ***A:*** Non Prime Attribute

Source: http://www.edugrabs.com/2nf-second-normal-form/

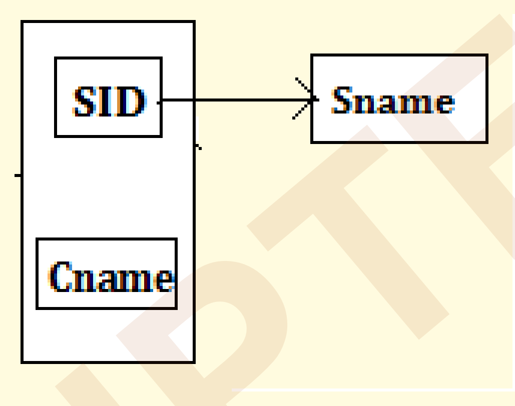
Second Normal Form (2 NF)

- Example:
  - ***STUDENT(Sid, Sname, Cname)*** (already in 1NF)

Students:

SID	Sname	Cname
S1	A	C
S1	A	C++
S2	B	C++
S2	B	DB
S3	A	DB

(SID, Cname): Primary Key



Functional Dependencies:
{SID,Cname} → Sname
SID → Sname

Partial Dependencies:
SID → Sname (as SID is a Proper Subset of Candidate Key {SID,Cname})

- **Redundancy?**
  - *Sname*
- **Anomaly?**
  - *Yes*

Post Normalization

R1:

SID	Sname
S1	A
S2	B
S3	A

{SID}: Primary Key

R2:

SID	Cname
S1	C
S1	C++
S2	C++
S2	DB
S3	DB

{SID,Cname}: Primary Key

The above two relations R1 and R2 are
1. Lossless Join
2. 2NF
3. Dependency Preserving

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

Source: http://www.edugrabs.com/2nf-second-normal-form/

Second Normal Form (2 NF): Possible Redundancy

- Example:
  - ***Supplier(SID, Status, City, PID, Qty)***

Supplier:

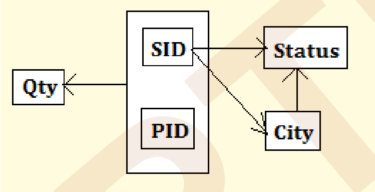
SID	Status	City	PID	Qty
S1	30	Delhi	P1	100
S1	30	Delhi	P2	125
S1	30	Delhi	P3	200
S1	30	Delhi	P4	130
S2	10	Karnal	P1	115
S2	10	Karnal	P2	250
S3	40	Rohtak	P1	245
S4	30	Delhi	P4	300
S4	30	Delhi	P5	315

Key : (SID, PID)

Partial Dependencies:

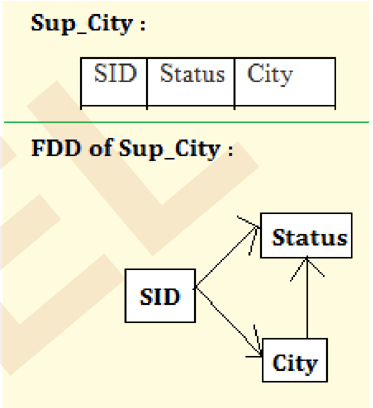
SID → Status

SID → City



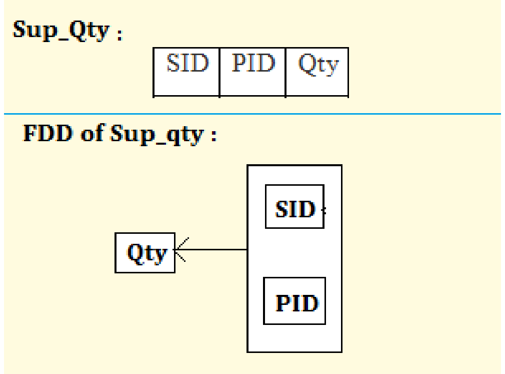
Source: http://www.edugrabs.com/2nf-second-normal-form/

Post Normalization


Sup_City :

SID	Status	City

FDD of Sup_City :


Sup_Qty :

SID	PID	Qty

FDD of Sup_qty :

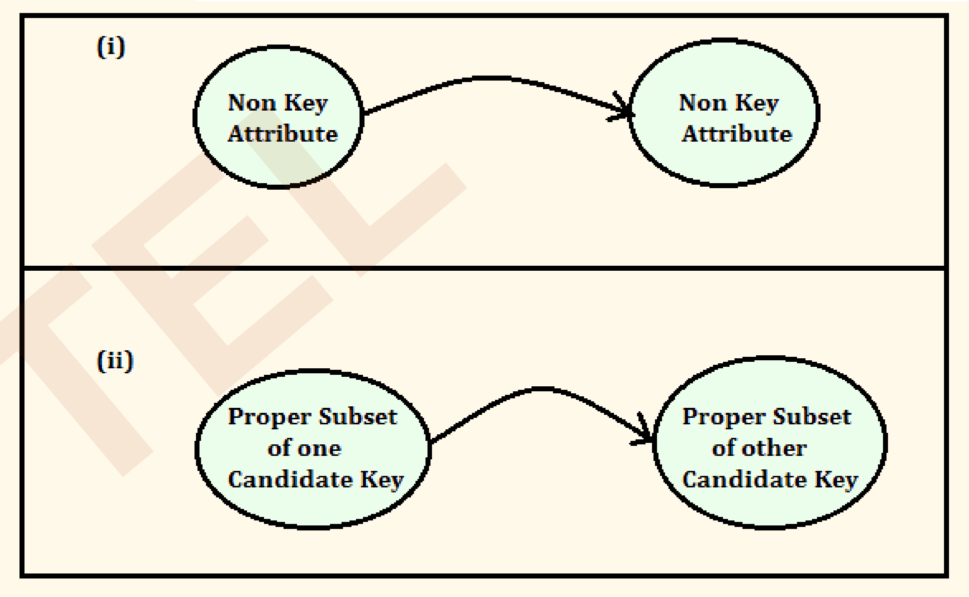
Drawbacks:

- **Deletion Anomaly –** If we delete a tuple in Sup_City, then we not only loose the information about a supplier, but also loose the status value of a particular city.
- **Insertion Anomaly –** We cannot insert a City and its status until a supplier supplies at least one part.
- **Updation Anomaly –** If the status value for a city is changed, then we will face the problem of searching every tuple for that city.

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

Second Normal Form (2 NF): Possible Redundancy

- In the **Sup_City** relation :
  - **City → Status**
    - Non Key Attribute → Non Key Attribute
- In the **STUDENT** relation:
  - **SID → Cname**
    - Proper Subset of 1 CK → Proper Subset of other CK


(i)
Non Key Attribute
Non Key Attribute
(ii)
Proper Subset of one Candidate Key
Proper Subset of other Candidate Key

Source: http://www.edugrabs.com/2nf-second-normal-form/

Third Normal Form (3 NF)

Let ***R*** be the relational schema.

- [E. F. Codd ,1971] ***R*** is in 3NF only if:
  - ***R*** should be in 2NF
  - ***R*** should not contain *transitive dependencies (OR,* Every non-prime attribute of R is non-transitively dependent on every key of R)
- [Carlo Zaniolo, 1982] Alternately, ***R*** is in 3NF iff for each of its functional dependencies $X \to A$, at least one of the following conditions holds:
  - *X* contains *A* (that is, *A* is a subset of *X*, meaning $X \to A$ is trivial functional dependency), or
  - *X* is a superkey, or
  - Every element of *A*-*X*, the set difference between A and X, is a *prime attribute* (i.e., each attribute in $A - X$ is contained in some candidate key)
- [Simple Statement} A relational schema **R** is in 3NF if for every FD $\mathbf{X} \to \mathbf{A}$ associated with R either
  - $\mathbf{A} \subseteq \mathbf{X}$ (i.e., the FD is trivial) or
  - **X** is a superkey of **R** or
  - **A** is part of some key (not just superkey!)

Source: http://www.edugrabs.com/3nf-third-normal-form/

Third Normal Form (3 NF)

- A **transitive dependency** is a functional dependency which holds by virtue of transitivity. A transitive dependency can occur only in a relation that has three or more attributes.
- Let A, B, and C designate three distinct attributes (or distinct collections of attributes) in the relation. Suppose all three of the following conditions hold:
  - A → B
  - It is not the case that B → A
  - B → C
- Then the functional dependency A → C (which follows from 1 and 3 by the axiom of transitivity) is a transitive dependency

Third Normal Form (3 NF)

- Example of **transitive dependency**
- The functional dependency {Book} → {Author Nationality} applies; that is, if we know the book, we know the author's nationality. Furthermore:
  - {Book} → {Author}
  - {Author} does not → {Book}
  - {Author} → {Author Nationality}
- Therefore {Book} → {Author Nationality} is a transitive dependency.
- Transitive dependency occurred because a non-key attribute (Author) was determining another non-key attribute (Author Nationality).

Book	Genre	Author	Author Nationality
Twenty Thousand Leagues Under the Sea	Science Fiction	Jules Verne	French
Journey to the Center of the Earth	Science Fiction	Jules Verne	French
Leaves of Grass	Poetry	Walt Whitman	American
Anna Karenina	Literary Fiction	Leo Tolstoy	Russian
A Confession	Religious Autobiography	Leo Tolstoy	Russian

Third Normal Form (3 NF)

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

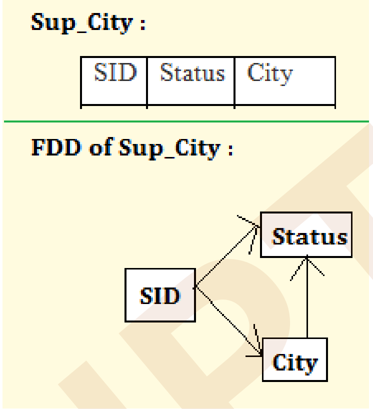
Source: http://www.edugrabs.com/2nf-second-normal-form/

- Example:
  - **Sup_City(SID, Status, City)** (already in 2NF)

Sup_City:

SID	Status	City
S1	30	Delhi
S2	10	Karnal
S3	40	Rohtak
S4	30	Delhi

SID: Primary Key



Functional Dependencies:
SID → Status, SID → City
City → Status

Transitive Dependency :
SID → Status {As SID → City and City → Status}

- **Redundancy?**
  - *Status*
- **Anomaly?**
  - *Yes*

Post Normalization

SC:

SID	City
S1	Delhi
S2	Karnal
S3	Rohtak
S4	Delhi

SID: Primary Key

CS:

City	Status
Delhi	30
Karnal	10
Rohtak	40

City: Primary Key

The above two relations SC and CS are

1. Lossless Join
2. 3NF
3. Dependency Preserving

Third Normal Form (3 NF)

- Example
  - Relation *dept_advisor*:
- ***dept_advisor* (*s_ID, i_ID, dept_name)***
- ***F* = {*s_ID, dept_name* → *i_ID, i_ID* → *dept_name*}**
- Two candidate keys: ***s_ID, dept_name***, and ***i_ID, s_ID***
- *R* is in 3NF
  - ***s_ID, dept_name* → *i_ID***
    - ***s_ID* , *dept_name*** is a superkey
  - ***i_ID* → *dept_name***
    - ***dept_name*** is contained in a candidate key

A relational schema **R** is in 3NF if for every FD **X** → **A** associated with R either

- **A** ⊆ **X** (i.e., the FD is trivial) or
- **X** is a superkey of **R** or
- **A** is part of some key (not just superkey!)

Redundancy in 3NF

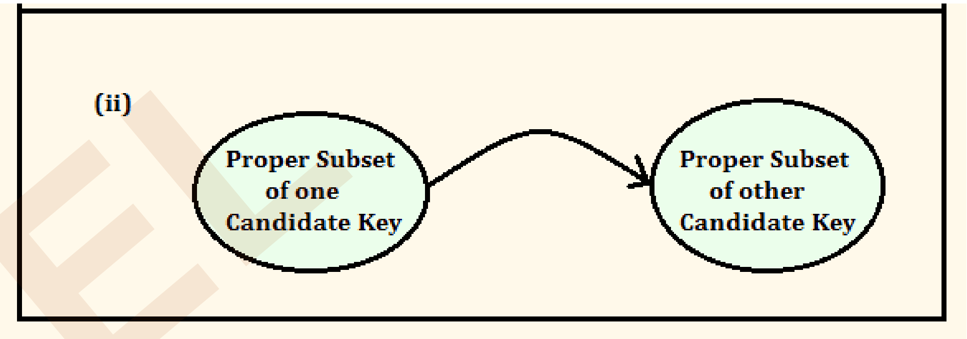
- **There is some redundancy in this schema**
- Example of problems due to redundancy in 3NF *(J: s_ID, L: i_ID, K: dept_name)*
  - $R = (J, L, K)$
    $F = \{JK \rightarrow L, L \rightarrow K\}$

J	L	K
j_1	l_1	k_1
j_2	l_1	k_1
j_3	l_1	k_1
null	l_2	k_2

- repetition of information (e.g., the relationship l_1, k_1)
  - *(i_ID, dept_name)*
- need to use null values (e.g., to represent the relationship l_2, k_2 where there is no corresponding value for J).
  - *(i_ID, dept_namel)* if there is no separate relation mapping instructors to departments

Third Normal Form (3 NF): Possible Redundancy

- A table is automatically in 3NF if one of the following hold :
  - (i) If relation consists of two attributes.
  - (ii) If 2NF table consists of only one non key attributes
- If X → A is a dependency, then the table is in the 3NF, if one of the following conditions exists:
  - If X is a superkey
  - If X is a part of superkey
- If X → A is a dependency, then the table is said to be NOT in 3NF if the following:
  - If X is a proper subset of some key (partial dependency)
  - If X is not a proper subset of key (non key)


(ii)
Proper Subset of one Candidate Key
Proper Subset of other Candidate Key

Source: http://www.edugrabs.com/2nf-second-normal-form/

- Normal Forms
- **Decomposition to 3NF**
- Decomposition to BCNF

DECOMPOSITION TO 3NF

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

Third Normal Form: Motivation

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

- There are some situations where
  - BCNF is not dependency preserving, and
  - Efficient checking for FD violation on updates is important
- Solution: define a weaker normal form, called Third Normal Form (3NF)
  - Allows some redundancy (with resultant problems; as seen above)
  - But functional dependencies can be checked on individual relations without computing a join
  - **There is always a lossless-join, dependency-preserving decomposition into 3NF**

Testing for 3NF

- Optimization: Need to check only FDs in *F*, need not check all FDs in F^+.
- Use attribute closure to check for each dependency $\alpha \rightarrow \beta$, if α is a superkey.
- If α is not a superkey, we have to verify if each attribute in β is contained in a candidate key of *R*
  - this test is rather more expensive, since it involve finding candidate keys
  - testing for 3NF has been shown to be NP-hard
  - Interestingly, decomposition into third normal form (described shortly) can be done in polynomial time

3NF Decomposition Algorithm

- Given: relation R, set F of functional dependencies
- Find: decomposition of R into a set of 3NF relation Ri
- Algorithm:
  1. Eliminate redundant FDs, resulting in a canonical cover Fc of F
  2. Create a relation Ri = XY for each FD X → Y in Fc
  3. If the key K of R does not occur in any relation Ri, create one more relation Ri=K

3NF Decomposition Algorithm (Formal)

```
Let $F_c$ be a canonical cover for $F$;
$i$ := 0;
for each  functional dependency $\alpha \rightarrow \beta$ in $F_c$ do
  if none of the schemas $R_j$, $1 \leq j \leq i$ contains  $\alpha\ \beta$
          then begin
                    $i := i + 1$;
                    $R_i := \alpha\ \beta$
              end
if none of the schemas $R_j$, $1 \leq j \leq i$ contains a candidate key for $R$
  then begin
              $i := i + 1$;
              $R_i$ := any candidate key for $R$;
          end
/* Optionally, remove redundant relations */

repeat
if any schema $R_j$ is contained in another schema $R_k$
        then /* delete $R_j$ */
          $R_j = R$;;
          $i$=$i$-1;
return ($R_1$, $R_2$, ..., $R_i$)
```

3NF Decomposition Algorithm

- Above algorithm ensures:
  - Each relation schema R_i is in 3NF
  - Decomposition is d
    - Dependency preserving and
    - Lossless-join

Example of 3NF Decomposition

- Relation schema:

  cust_banker_branch = (*customer_id, employee_id*, *branch_name, type*)

- The functional dependencies for this relation schema are:
  1. *customer_id, employee_id* → *branch_name, type*
  2. *employee_id* → *branch_name*
  3. *customer_id, branch_name* → *employee_id*

- We first compute a canonical cover
  - *branch_name* is extraneous in the r.h.s. of the 1st dependency
  - No other attribute is extraneous, so we get F_C =

    customer_id, employee_id → *type*
    employee_id → *branch_name*
    customer_id, branch_name → *employee_id*

Example of 3NF Decomposition

- The **for** loop generates following 3NF schema:

  (*customer_id, employee_id, type*)

  (*employee_id, branch_name*)

  (*customer_id, branch_name, employee_id)*

  - Observe that (*customer_id, employee_id, type*) contains a candidate key of the original schema, so no further relation schema needs be added

- At end of for loop, detect and delete schemas, such as (*employee_id, branch_name*), which are subsets of other schemas

  - result will not depend on the order in which FDs are considered

- The resultant simplified 3NF schema is:

  (*customer_id, employee_id, type*)

  (*customer_id, branch_name, employee_id)*

Practice Problem for 3NF Decomposition: 1

- R = ABCDEFGH
- FDs = {A→B, ABCD→E, EF→GH, ACDF→EG}

Solution is given in the next slide (hidden from presentation – check after you have solved)

Solution: Practice Problem for 3NF Decomposition: 1

- R = ABCDEFGH
- FD = F = {A→B, ABCD→E, EF→GH, ACDF→EG}
- Compute Canonical Cover (Fc)
  - Make RHS a single attribute: {A→B, ABCD→E, EF→G, EF→H, ACDF→E, ACDF→G}
  - Minimize LHS: ACD→E instead of ABCD→E
  - Eliminate redundant FDs
    - Can ACDF→G be removed?
    - Can ACDF→E be removed?
  - Fc = {A→B, ACD→E, EF→G, EF→H}
- Superkey = ACDF
- 3NF Decomposition = {AB, ACDE, EFG, EFH} U {ACDF}

This is a hidden slide giving solution. Check only after you have solved)

Practice Problem for 3NF Decomposition: 2

- R = CSJDPQV
- FDs = {C→CSJDPQV, SD→P, JP→C,J→S}

Solution is given in the next slide (hidden from presentation – check after you have solved)

Solution: Practice Problem for 3NF Decomposition: 2

- R = CSJDPQV
- FDs = F = $\{C \to CSJDPQV, SD \to P, JP \to C, J \to S\}$
- Compute Canonical Cover (Fc)
  - Minimal cover: $\{C \to J, C \to D, C \to Q, C \to V, JP \to C, J \to S, SD \to P\}$ (Combine LHS)
  - Fc = $\{C \to JDQV, JP \to C, J \to S, SD \to P\}$
- Superkey = C
- 3NF Decomposition = {CJDQV, JPC, JS, SDP}

This is a hidden slide giving solution. Check only after you have solved)

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

- Normal Forms
- Decomposition to 3NF
- **Decomposition to BCNF**

DECOMPOSITION TO BCNF

Testing for BCNF

- To check if a non-trivial dependency $\alpha \rightarrow \beta$ causes a violation of BCNF
  1. compute α^+ (the attribute closure of α), and
  2. verify that it includes all attributes of R, that is, it is a superkey of R.
- **Simplified test**: To check if a relation schema R is in BCNF, it suffices to check only the dependencies in the given set F for violation of BCNF, rather than checking all dependencies in F^+.
  - If none of the dependencies in F causes a violation of BCNF, then none of the dependencies in F^+ will cause a violation of BCNF either.
- However, **simplified test using only *F* is incorrect when testing a relation in a decomposition of R**
  - Consider $R = (A, B, C, D, E)$, with $F = \{ A \rightarrow B, BC \rightarrow D\}$
    - Decompose R into $R_1 = (A,B)$ and $R_2 = (A,C,D, E)$
    - Neither of the dependencies in F contain only attributes from (A,C,D,E) so we might be mislead into thinking R_2 satisfies BCNF.
    - In fact, dependency $AC \rightarrow D$ in F^+ shows R_2 is not in BCNF.

Testing Decomposition for BCNF

- To check if a relation R_i in a decomposition of R is in BCNF,
  - Either test R_i for BCNF with respect to the **restriction** of F to R_i (that is, all FDs in F^+ that contain only attributes from R_i)
  - or use the original set of dependencies F that hold on R, but with the following test:
    - for every set of attributes $\alpha \subseteq R_i$, check that α^+ (the attribute closure of α) either includes no attribute of $R_i - \alpha$, or includes all attributes of R_i.
    - If the condition is violated by some $\alpha \rightarrow \beta$ in F, the dependency
      $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$
      can be shown to hold on R_i, and R_i violates BCNF.
    - We use above dependency to decompose R_i

BCNF Decomposition Algorithm

1. For all dependencies $A \rightarrow B$ in F+, check if A is a superkey
   - By using attribute closure
2. If not, then
   - Choose a dependency in F+ that breaks the BCNF rules, say $A \rightarrow B$
   - Create $R1 = A\,B$
   - Create $R2 = A\,(R - (B - A))$
   - Note that: $R1 \cap R2 = A$ and $A \rightarrow AB$ (= R1), so this is lossless decomposition
3. Repeat for R1, and R2
   - By defining F1+ to be all dependencies in F that contain only attributes in R1
   - Similarly F2+

BCNF Decomposition Algorithm

```
$result$ := {$R$ };
$done$ := false;
compute $F^+$;
**while (not** $done$) **do**
    **if** (there is a schema $R_i$ in $result$ that is not in BCNF)
        **then begin**
                let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that
                    holds on $R_i$ such that $\alpha \rightarrow R_i$ is not in $F^+$,
                      and $\alpha \cap \beta = \emptyset$;
                  $result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta);$
                **end**
        **else** $done$ := **true;**
```

Note: each R_i is in BCNF, and decomposition is lossless-join.

Example of BCNF Decomposition

- $R = (A, B, C)$
  $F = \{A \rightarrow B$
  $\quad B \rightarrow C\}$
  Key = $\{A\}$
- R is not in BCNF ($B \rightarrow C$ but B is not superkey)
- Decomposition
  - $R_1 = (B, C)$
  - $R_2 = (A,B)$

Example of BCNF Decomposition

- *class* (*course_id*, *title*, *dept_name*, *credits*, *sec_id*, *semester*, *year*, *building*, *room_number*, *capacity*, *time_slot_id*)
- Functional dependencies:
  - *course_id*→ *title*, *dept_name*, *credits*
  - *building*, *room_number*→*capacity*
  - *course_id*, *sec_id*, *semester*, *year*→*building*, *room_number*, *time_slot_id*
- A candidate key {*course_id*, *sec_id*, *semester*, *year*}.
- BCNF Decomposition:
  - *course_id*→ *title*, *dept_name*, *credits* holds
    - but *course_id* is not a superkey.
  - We replace *class* by:
    - *course*(*course_id*, *title*, *dept_name*, *credits*)
    - *class-1* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number, capacity*, *time_slot_id*)

BCNF Decomposition (Cont.)

- *course* is in BCNF
  - How do we know this?
- *building*, *room_number*→*capacity* holds on *class-1*(*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number, capacity*, *time_slot_id*)
  - but {*building*, *room_number*} is not a superkey for *class-1*.
  - We replace *class-1* by:
    - *classroom* (*building*, *room_number*, *capacity*)
    - *section* (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*, *time_slot_id*)
- *classroom* and *section* are in BCNF.

BCNF and Dependency Preservation

- It is not always possible to get a BCNF decomposition that is dependency preserving
- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $L \rightarrow K\}$
  Two candidate keys = JK and JL
- R is not in BCNF
- Any decomposition of R will fail to preserve

$$JK \rightarrow L$$

This implies that testing for $JK \rightarrow L$ requires a join

Practice Problem for BCNF Decomposition

- R = ABCDE. F = $\{A \rightarrow B, BC \rightarrow D\}$
- R = ABCDE. F = $\{A \rightarrow B, BC \rightarrow D\}$
- R = ABCDEH. F = $\{A \rightarrow BC, E \rightarrow HA\}$
- R = CSJDPQV. F = $\{C \rightarrow CSJDPQV, SD \rightarrow P, JP \rightarrow C, J \rightarrow S\}$
- R = ABCD. F = $\{C \rightarrow D, C \rightarrow A, B \rightarrow C\}$

Comparison of BCNF and 3NF

- It is always possible to decompose a relation into a set of relations that are in 3NF such that:
  - the decomposition is lossless
  - the dependencies are preserved
- It is always possible to decompose a relation into a set of relations that are in BCNF such that:
  - the decomposition is lossless
  - it may not be possible to preserve dependencies.

S#	3NF	BCNF
1.	It concentrates on Primary Key	It concentrates on Candidate Key.
2.	Redundancy is high as compared to BCNF	0% redundancy
3.	It may preserve all the dependencies	It may not preserve the dependencies.
4.	A dependency X → Y is allowed in 3NF if X is a super key or Y is a part of some key.	A dependency X → Y is allowed if X is a super key

Module Summary

- Studied the Normal Forms and their Importance in Relational Design – how progressive increase of constraints can minimize redundancy in a schema
- Learnt how to decompose a schema into 3NF while preserving dependency and lossless join
- Learnt how to decompose a schema into BCNF with lossless join

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".

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

Database Management Systems

Module 20: Relational Database Design/5

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ernet.in

Srijoni Majumdar
Himadri B G S Bhuyan
Gurunath Reddy M

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
www.db-book.com

PPD

Module Recap

- Normal Forms
- Decomposition to 3NF
- Decomposition to BCNF

PPD

Module Objectives

- To understand multi-valued dependencies arising out of attributes that can have multiple values
- To define Fourth Normal Form and learn the decomposition algorithm to 4NF
- To summarize the database design process
- To explore the issues with temporal data

Module Outline

- Multivalued Dependencies
- Decomposition to 4NF
- Database-Design Process
- Modeling Temporal Data

- **Multivalued Dependencies**
- Decomposition to 4NF
- Database-Design Process
- Modeling Temporal Data

MULTIVALUED DEPENDENCY

Multivalued Dependency

- ***Persons(Man, Phones, Dog_Like)***

Person :			**Meaning of the tuples**
Man(M)	**Phones(P)**	**Dogs_Like(D)**	Man M have phones P, and likes the dogs D.
M1	P1/P2	D1/D2	M1 have phones P1 and P2, and likes the dogs D1 and D2.
M2	P3	D2	M2 have phones P3, and likes the dog D2.
Key : MPD			

There are no non trivial FDs because all attributes are combined forming Candidate Key i.e. MDP. In the above relation, two multivalued dependencies exists –

- **Man →→ Phones**
- **Man →→ Dogs_Like**

A man's phone are independent of the dogs they like. But after converting the above relation in Single Valued Attribute, each of a man's phones appears with each of the dogs they like in all combinations.

Post 1NF Normalization

Man(M)	Phones(P)	Dogs_Likes(D)
M1	P1	D1
M1	P2	D2
M2	P3	D2
M1	P1	D2
M1	P2	D1

Source: http://www.edugrabs.com/multivalued-dependency-mvd/

Multivalued Dependency

- If two or more independent relations are kept in a single relation, then Multivalued Dependency is possible. For example, Let there are two relations :
  - ***Student(SID, Sname)*** *where* ***(SID → Sname)***
  - ***Course(CID, Cname)*** *where* ***(CID → Cname)***
- There is no relation defined between Student and Course. If we kept them in a single relation named ***Student_Course***, then MVD will exists because of *m:n Cardinality*
- If two or more MVDs exist in a relation, then while converting into SVAs, MVD exists.

Student:

SID	Sname
S1	A
S2	B

Course:

CID	Cname
C1	C
C2	B

SID	Sname	CID	Cname
S1	A	C1	C
S1	A	C2	B
S2	B	C1	C
S2	B	C2	B

2 MVDs exist:

1. SID →→ CID
2. SID →→ Cname

Source: http://www.edugrabs.com/multivalued-dependency-mvd/

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

Multivalued Dependencies

- Suppose we record names of children, and phone numbers for instructors:
  - *inst_child*(*ID*, *child_name*)
  - *inst_phone*(*ID*, *phone_number*)
- If we were to combine these schemas to get
  - *inst_info*(*ID*, *child_name*, *phone_number*)
  - Example data:
    (99999, David, 512-555-1234)
    (99999, David, 512-555-4321)
    (99999, William, 512-555-1234)
    (99999, William, 512-555-4321)
- This relation is in BCNF
  - Why?

Multivalued Dependencies (MVDs)

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

Source: http://www.edugrabs.com/multivalued-dependency-mvd/

- Let *R* be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The **multivalued dependency**

$$\alpha \rightarrow\rightarrow \beta$$

holds on *R* if in any legal relation *r(R),* for all pairs for tuples t_1 and t_2 in *r* such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in *r* such that:

$$
\begin{aligned}
t_1[\alpha] = t_2[\alpha] &= t_3[\alpha] = t_4[\alpha] \\
t_3[\beta] &= t_1[\beta] \\
t_3[R - \beta] &= t_2[R - \beta] \\
t_4[\beta] &= t_2[\beta] \\
t_4[R - \beta] &= t_1[R - \beta]
\end{aligned}
$$

Example: A relation of university courses, the books recommended for the course, and the lecturers who will be teaching the course:

- **course →→ book**
- **course →→ lecturer**

Test: **course →→ book**

Course	Book	Lecturer	Tuples
AHA	Silberschatz	John D	t1
AHA	Nederpelt	William M	t2
AHA	Silberschatz	William M	t3
AHA	Nederpelt	John D	t4
AHA	Silberschatz	Christian G	
AHA	Nederpelt	Christian G	
OSO	Silberschatz	John D	
OSO	Silberschatz	William M	

→→

Example

- Let *R* be a relation schema with a set of attributes that are partitioned into 3 nonempty subsets.

  Y, Z, W

- We say that $Y \rightarrow\rightarrow Z$ (Y **multidetermines** Z) if and only if for all possible relations $r(R)$

  $< y_1, z_1, w_1 > \in r$ and $< y_1, z_2, w_2 > \in r$

  then

  $< y_1, z_1, w_2 > \in r$ and $< y_1, z_2, w_1 > \in r$

- Note that since the behavior of *Z* and *W* are identical it follows that

  $Y \rightarrow\rightarrow Z$ if $Y \rightarrow\rightarrow W$

Example (Cont.)

- In our example:

$$ID \rightarrow\rightarrow \textit{child\_name}$$
$$ID \rightarrow\rightarrow \textit{phone\_number}$$

- The above formal definition is supposed to formalize the notion that given a particular value of *Y* (*ID*) it has associated with it a set of values of *Z (child_name)* and a set of values of *W (phone_number)*, and these two sets are in some sense independent of each other.
- Note:
  - If $Y \rightarrow Z$ then $Y \rightarrow\rightarrow Z$
  - Indeed we have (in above notation) $Z_1 = Z_2$
    The claim follows.

Use of Multivalued Dependencies

- We use multivalued dependencies in two ways:
  1. To test relations to **determine** whether they are legal under a given set of functional and multivalued dependencies
  2. To specify **constraints** on the set of legal relations. We shall thus concern ourselves *only* with relations that satisfy a given set of functional and multivalued dependencies.
- If a relation *r* fails to satisfy a given multivalued dependency, we can construct a relations *r′* that does satisfy the multivalued dependency by adding tuples to *r.*

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

Theory of MVDs

	Name		Rule
C-	**Complementation**	:	If $X \rightarrow\rightarrow Y$, then $X \rightarrow\rightarrow \{R - (X \cup Y)\}$.
A-	**Augmentation**	:	If $X \rightarrow\rightarrow Y$ and $W \supseteq Z$, then $WX \rightarrow\rightarrow YZ$.
T-	**Transitivity**	:	If $X \rightarrow\rightarrow Y$ and $Y \rightarrow\rightarrow Z$, then $X \rightarrow\rightarrow (Z - Y)$.
	Replication	:	If $X \rightarrow Y$, then $X \rightarrow\rightarrow Y$ but the reverse is not true.
	Coalescence	:	If $X \rightarrow\rightarrow Y$ and there is a W such that $W \cap Y$ is empty, $W \rightarrow Z$, and $Y \supseteq Z$, then $X \rightarrow Z$.

- A MVD $\boldsymbol{X \rightarrow\rightarrow Y}$ in $\boldsymbol{R}$ is called a trivial MVD is
  - $\boldsymbol{Y}$ is a subset of $\boldsymbol{X}$ $\boldsymbol{(X \supseteq Y)}$ or
  - $\boldsymbol{X \cup Y = R}$. Otherwise, it is a non trivial MVD and we have to repeat values redundantly in the tuples.

Source: http://www.edugrabs.com/multivalued-dependency-mvd/

Theory of MVDs

- From the definition of multivalued dependency, we can derive the following rule:
  - If $\alpha \rightarrow \beta$, then $\alpha \rightarrow\rightarrow \beta$

  That is, every functional dependency is also a multivalued dependency
- The **closure** D$^+$ of *D* is the set of all functional and multivalued dependencies logically implied by *D*.
  - We can compute D$^+$ from *D*, using the formal definitions of functional dependencies and multivalued dependencies.
  - We can manage with such reasoning for very simple multivalued dependencies, which seem to be most common in practice
  - For complex dependencies, it is better to reason about sets of dependencies using a system of inference rules

$\rightarrow\rightarrow$

- Multivalued Dependencies
- **Decomposition to 4NF**
- Database-Design Process
- Modeling Temporal Data

DECOMPOSITION TO 4NF

Fourth Normal Form

- A relation schema *R* is in **4NF** with respect to a set *D* of functional and multivalued dependencies if for all multivalued dependencies in D^+ of the form $\alpha \rightarrow\rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following hold:
  - $\alpha \rightarrow\rightarrow \beta$ is trivial (i.e., $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$)
  - α is a superkey for schema *R*
- If a relation is in 4NF it is in BCNF

Restriction of Multivalued Dependencies

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

- The restriction of D to R_i is the set D_i consisting of
  - All functional dependencies in D^+ that include only attributes of R_i
  - All multivalued dependencies of the form

    $\alpha \rightarrow\rightarrow (\beta \cap R_i)$

    where $\alpha \subseteq R_i$ and $\alpha \rightarrow\rightarrow \beta$ is in D^+

4NF Decomposition Algorithm

1. For all dependencies A →→ B in D+, check if A is a superkey
   - By using attribute closure
2. If not, then
   - Choose a dependency in F+ that breaks the 4NF rules, say A →→ B
   - Create R1 = A B
   - Create R2 = A (R – (B – A))
   - Note that: R1 ∩ R2 = A and A →→ AB (= R1), so this is lossless decomposition
3. Repeat for R1, and R2
   - By defining D1+ to be all dependencies in F that contain only attributes in R1
   - Similarly D2+

4NF Decomposition Algorithm

```
result: = {R};
done := false;
compute D+;
Let D_i denote the restriction of D+ to R_i
while (not done)
   if (there is a schema R_i in result that is not in 4NF) then
      begin
         let $\alpha \rightarrow\rightarrow \beta$ be a nontrivial multivalued dependency that holds
           on $R_i$ such that $\alpha \rightarrow R_i$ is not in $D_i$, and $\alpha \cap \beta = \phi$;
         result :=  (result - $R_i$) $\cup$ ($R_i$ - $\beta$)  $\cup$ ($\alpha$, $\beta$);
      end
   else done:= true;
```

Note: each R_i is in 4NF, and decomposition is lossless-join

$\rightarrow\!\!\rightarrow$

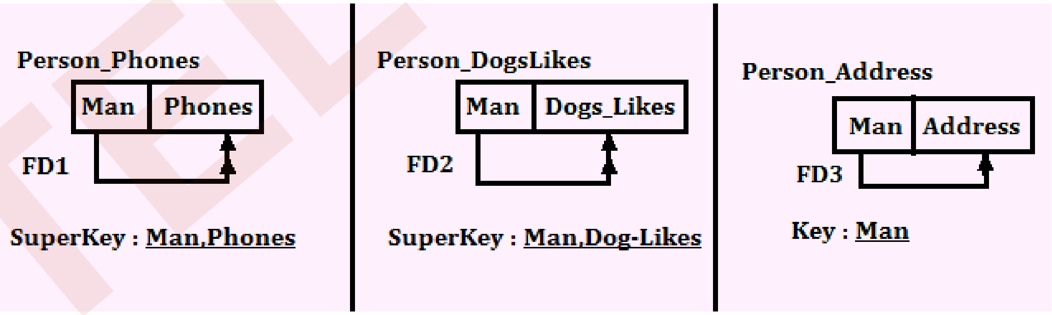
Example of 4NF Decomposition

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

Source: http://www.edugrabs.com/4nf-normal-form/

- Example:
  - ***Person_Modify(Man(M), Phones(P), Dog_Likes(D), Address(A))***
  - ***FDs:***
    - **FD1 : Man →→ Phones**
    - **FD2 : Man →→ Dogs_Like**
    - **FD3 : Man → Address**
  - Key = **MPD**
  - All dependencies violate 4NF

Post Normalization



Person_Phones

Man	Phones

FD1

SuperKey : Man,Phones

Person_DogsLikes

Man	Dogs_Likes

FD2

SuperKey : Man,Dog-Likes

Person_Address

Man	Address

FD3

Key : Man

Man(M)	Phones(P)	Dogs_Likes(D)	Address(A)
M1	P1	D1	49-ABC,Bhiwani(HR.)
M1	P2	D2	49-ABC,Bhiwani(HR.)
M2	P3	D2	36-XYZ,Rohtak(HR.)
M1	P1	D2	49-ABC,Bhiwani(HR.)
M1	P2	D1	49-ABC,Bhiwani(HR.)

In the above relations for both the MVD's – ***'X' is Man***, which is again not the super key, but as ***X ∪ Y = R*** i.e. (Man & Phones) together make the relation.
So, the above MVD's are trivial and in FD 3, Address is functionally dependent on Man, where **Man** is the key in **Person_Address**, hence all the three relations are in 4NF.

Example of 4NF Decomposition

- $R = (A, B, C, G, H, I)$

  $F = \{ A \rightarrow\rightarrow B$

  $B \rightarrow\rightarrow HI$

  $CG \rightarrow\rightarrow H \}$
- R is not in 4NF since $A \rightarrow\rightarrow B$ and A is not a superkey for R
- Decomposition

  a) $R_1 = (A, B)$ (R_1 is in 4NF)

  b) $R_2 = (A, C, G, H, I)$ (R_2 is not in 4NF, decompose into R_3 and R_4)

  c) $R_3 = (C, G, H)$ (R_3 is in 4NF)

  d) $R_4 = (A, C, G, I)$ (R_4 is not in 4NF, decompose into R_5 and R_6)

  - $A \rightarrow\rightarrow B$ and $B \rightarrow\rightarrow HI$ ➔ $A \rightarrow\rightarrow HI$, (MVD transitivity), and
  - and hence $A \rightarrow\rightarrow I$ *(MVD restriction to* R_4*)*

  e) $R_5 = (A, I)$ (R_5 is in 4NF)

  f) $R_6 = (A, C, G)$ (R_6 is in 4NF)

- Multivalued Dependencies
- Decomposition to 4NF
- **Database-Design Process**
- Modeling Temporal Data

DATABASE DESIGN PROCESS

Design Goals

- Goal for a relational database design is:
  - BCNF / 4NF
  - Lossless join
  - Dependency preservation
- If we cannot achieve this, we accept one of
  - Lack of dependency preservation
  - Redundancy due to use of 3NF
- Interestingly, SQL does not provide a direct way of specifying functional dependencies other than superkeys.

  Can specify FDs using assertions, but they are expensive to test, (and currently not supported by any of the widely used databases!)
- Even if we had a dependency preserving decomposition, using SQL we would not be able to efficiently test a functional dependency whose left hand side is not a key

Further Normal Forms

- Further NFs
  - Elementary Key Normal Form (EKNF)
  - Essential Tuple Normal Form (ETNF)
  - Join Dependencies And Fifth Normal Form (5 NF)
  - Sixth Normal Form (6NF)
  - Domain/Key Normal Form (DKNF)
- **Join dependencies** generalize multivalued dependencies
  - lead to **project-join normal form (PJNF)** (also called **fifth normal form**)
- A class of even more general constraints, leads to a normal form called **domain-key normal form**.
- Problem with these generalized constraints: are hard to reason with, and no set of sound and complete set of inference rules exists.
- Hence rarely used

Overall Database Design Process

- We have assumed schema *R* is given
  - *R* could have been generated when converting E-R diagram to a set of tables
  - *R* could have been a single relation containing *all* attributes that are of interest (called **universal relation**)
  - Normalization breaks *R* into smaller relations
  - *R* could have been the result of some ad hoc design of relations, which we then test/convert to normal form

ER Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization
- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - Example: an *employee* entity with attributes
    department_name and *building*,
    and a functional dependency
    department_name → *building*
  - Good design would have made department an entity
- Functional dependencies from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Denormalization for Performance

- May want to use non-normalized schema for performance
- For example, displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*
  - ***Course(course_id, title, …)***
  - ***Prerequisite(course_id, prereq)***
- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes: ***Course(course_id, title, prereq, …)***
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code
- Alternative 2: use a materialized view defined as
  Course $\bowtie$ *Prerequisite*
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Other Design Issues

- Some aspects of database design are not caught by normalization
- Examples of bad database design, to be avoided:

  Instead of *earnings* (*company_id, year, amount*), use

  - *earnings_2004, earnings_2005, earnings_2006*, etc., all on the schema (*company_id, earnings*).
    - Above are in BCNF, but make querying across years difficult and needs new table each year
  - *company_year* (*company_id, earnings_2004, earnings_2005, earnings_2006*)
    - Also in BCNF, but also makes querying across years difficult and requires new attribute each year.
    - Is an example of a **crosstab**, where values for one attribute become column names
    - Used in spreadsheets, and in data analysis tools

PPD

- Multivalued Dependencies
- Decomposition to 4NF
- Database-Design Process
- **Modeling Temporal Data**

MODELING TEMPORAL DATA

Modeling Temporal Data

- **Temporal data** have an association time interval during which the data are *valid.*
- A **snapshot** is the value of the data at a particular point in time
- Several proposals to extend ER model by adding valid time to
  - attributes, e.g., address of an instructor at different points in time
  - entities, e.g., time duration when a student entity exists
  - relationships, e.g., time during which an instructor was associated with a student as an advisor.

τ

- But no accepted standard
- Adding a temporal component results in functional dependencies like

  $ID \rightarrow street, city$

  not to hold, because the address varies over time
- A **temporal functional dependency** X → Y holds on schema *R* if the functional dependency X → Y holds on all snapshots for all legal instances r (*R*).

Modeling Temporal Data (Cont.)

- In practice, database designers may add start and end time attributes to relations
  - E.g., *course*(*course_id, course_title*) is replaced by

    course(*course_id, course_title, start, end*)

    - Constraint: no two tuples can have overlapping valid times
      - Hard to enforce efficiently
- Foreign key references may be to current version of data, or to data at a point in time
  - E.g., student transcript should refer to course information at the time the course was taken

Module Summary

- Understood multi-valued dependencies to handle attributes that can have multiple values
- Learnt Fourth Normal Form and decomposition to 4NF
- Discussed aspects of the database design process
- Studied the issues with temporal data

PPD

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, Instructor	ppd@cse.iitkgp.ernet.in	9830030880
Srijoni Majumdar, TA	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, TA	himadribhuyan@gmail.com	9438911655
Gurunath Reddy M	mgurunathreddy@gmail.com	9434137638

Slides used in this presentation are borrowed from http://db-book.com/ with kind permission of the authors.

Edited and new slides are marked with "PPD".